# An Animated Virtual Drummer

*Martijn Kragtwijk, Anton Nijholt, Job Zwiers*

Department of Computer Science
University of Twente
PO Box 217, 7500 AE Enschede, the Netherlands
Phone: 00-31-53-4893686
Fax: 00-31-53-4893503
email: {kragtwij,anijholt,zwiers}@cs.utwente.nl

## ABSTRACT

We describe a system for the automatic generation of a 3D animation of a drummer playing along with a given piece of music. The input, consisting of a sound wave, is analysed to determine which drums are struck at what moments. The Standard MIDI File format is used to store the recognised notes. From this higher-level description of the music, the animation is generated. The system is implemented in Java and uses the Java3D API for visualisation.

## 1. INTRODUCTION

In this paper we describe preliminary results of our research on virtual musicians. The objective of this project is to generate animated virtual musicians, that play along with a given piece of music. The input of this system consists of a sound wave, originating from e.g. a CD or a real-time recording.

There are many possible uses for an application like this, ranging from the automatic generation of music videos to interactive music performance systems where musicians play together in a virtual environment. In the last case, the real musicians could be located on different sites, and their virtual counterparts could be viewed in a virtual theatre by a worldwide audience. Additionally, our department is currently working on instructional agents that can teach music, for which the work we describe in this paper will be a good foundation.

For our first virtual musicians application, we have restricted ourselves to an animated drummer. However, the system is flexible enough to allow an easy extension to other instruments.

As figure 1 shows, the total task can be separated into two independent subtasks:

- An analysis of the sound signal and transcription of the percussion part. The system has to determine which drums are hit, at what moments in time. Concentrating on percussion sounds has certain advantages and disadvantages; this is further discussed in section 2.
- The creation of the the movements of a 3D avatar playing on a drum kit. A more detailed explanation on this part is given in section 3.

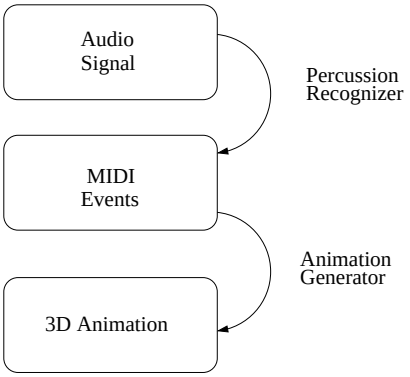


Figure 1: An overview of the system

## 2. THE PERCUSSION RECOGNISER

This part of the system is responsible for the translation from a 'low level' description of the music (the sound wave) to a abstract, 'high level' description of all percussion sounds that are present in the signal. These recognised notes are stored as MIDI events.

Many attempts in the field of musical instrument recognition concentrate on pitched sounds [2]. As explained in [4], this is a rather different task than recognising percussive sounds, which have a sharp attack, short duration, and no clearly defined pitch. As shown in [4], indi-

vidual, monophonic samples of drums and cymbals can be classified very well. In this approach, a few frames of the spectrum, measured from the onset of the sounds, were matched against a database of spectral templates.

In our highly polyphonic, 'real-life' situation, however, the input signal may contain many percussive sounds played simultaneously, and non-percussive instruments (such as guitar and vocals) may be mixed through the signal as well. Therefore, special techniques are needed to separate the percussive sounds from the other sounds. Other researchers have already tried to solve the same problem [6, 7]: Sillanpää et. al. subtract harmonic components from the input signal to filter out non-percussive sounds. Furthermore, they stress the importance of top-down processing: using temporal predictions to recognise soft sounds that are partially masked by louder sounds [7]. Puckette's Pure Data program has an object called *bonk* that uses the difference between subsequent short-time spectra, to determine whether a new 'attack' has occurred.

We are still developing this part of the system, therefore we cannot yet present a final solution of this problem. We plan to solve the problem of polyphony by adding examples that consist of multiple sounds played together to the collection of spectral templates. For example, a bass drum, snare drum and hihat played together. For an off-line situation, where the complete input signal is already known, we plan to apply clustering methods on all fragments of the signal that contain a strong attack. This is based on our hypothesis that specific drum sounds will sound very similar throughout a piece of music. This is especially plausible for commercial recordings, and/or in the case that the music contains sampled drum sounds.

## 3. THE ANIMATION GENERATOR

In our current off-line implementation, the piece of music to be played is completely known in advance as a list of MIDI events. Therefore, the entire animation can be computed before it is started. In a real-time situation, where the system has to respond to incoming MIDI events, this would not possible. In that case, a short animation should be constructed and started immediately for each note that occurs in the input.

A great advantage of pre-calculating the entire animation is that the transitions between two strokes will be much smoother: for each note we already know which drum will be struck next, and the arm can already start moving towards that drum.

Figure 2: A screenshot of our example configuration

Figure 2 shows a screenshot of our example configuration, where we use a humanoid avatar derived from an H-Anim compliant model by C. Babski [1]. All arms and legs of this avatar have seven degrees of freedom, the drum stick is attached to the hand object.

### 3.1. Polyphony Issues

Monophonic instruments (such as the trumpet or the flute) are relatively easy to animate, because each possible sound corresponds to exactly one 'pose' of all fingers, valves, etcetera, and only one pose can be active at each moment in time. Highly polyphonic instruments (such as the piano) are much more difficult, because there are many different ways ('fingerings') to play the same piece of music, and a search method is needed to find a good solution [3]. The drum kit could be viewed in between these two extreme examples: up to four sounds can be started simultaneously.

### 3.2. Drum Kit Model

The 47 percussive sounds that are defined in the General MIDI standard are mapped onto a set of 'drum events' that is specified in the Drum Kit Model, along with a set of parameters that contain the properties of the drum kit:

- For each event type, a preferred hand.
- For each pair of drum event types, a preferred distribution of the hands for these two events.

- For each drum event type, an elasticity value $el_eventType$ that determines how far a drum stick bounces back to its initial position after contact.

### 3.3. MIDI parsing

First, the list of MIDI events is transformed into a list of animation events according to the drum event types defined in the drum kit model (see 3.2). Besides belonging to a drum event type, each animation event has an associated velocity $vel_{event}$. Secondly, the list of animation events is parsed to remove double events and distribute the events over the different animated objects.

Objects in the scene respond to a subset of drum event types, e.g.:

- Right leg: only the 'BASS' event
- Left leg: 'HIHAT OPEN','HIHAT PEDAL' and 'HIHAT CLOSED'
- Both arms support all events except 'BASS' and 'HIHAT PEDAL'.

Drum events that can be played by both hands need to be distributed between left and right in a natural looking way:

1. First of all, no more than 2 of these events are allowed to have the same time stamp.
2. For two events with the same time stamp, the hand assignment is completely determined by the pair-wise preference as defined in section 3.2. The more time there is between two subsequent events, the more this constraint is releaved and a hand-preference per drum event is taken into account.
3. Finally, the hands can be alternated for fast rolls.

### 3.4. Key frame creation

A graphical user-interface is provided to create 'poses'. For each limb, two poses should be specified for each drum object that it supports: the 'DOWN' pose (the exact situation on contact) and the 'UP' pose (the situation just before and just after the hitting moment).

For each animated object, its abstract time line parsed in the correct temporal order. For each animation event, a 'stroke' is added to the animation time line. A 'stroke' consists of three key frames: $[TR_{before}, TR_{contact}, TR_{after}]$, where $TR_{contact}$ has the same time stamp as the animation event. These key frames are calculated by interpolating between the key frames 'UP' and 'DOWN' in the following way (see also figure 3):

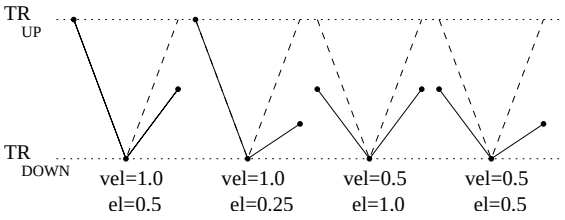


Figure 3: The effect of different values for *vel* and *el*.

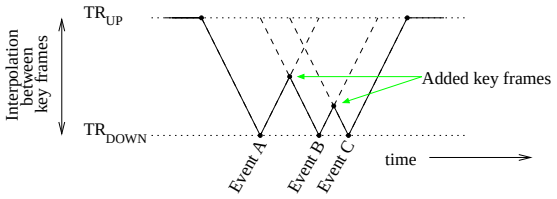


Figure 4: When events overlap, new key frames are created

$$TR_{before} = TR_{DOWN} + vel_{event} \times diff$$
$$TR_{contact} = TR_{DOWN}$$
$$TR_{after} = TR_{DOWN} + vel_{event} \times el_{eventType} \times diff$$
$$diff = TR_{UP} - TR_{DOWN}$$

If two strokes overlap, $TR_{after}$ of the first stroke and $TR_{before}$ of the second are replaced by a new key frame, as can be seen in figure 4.

After the basic key frames are set, the motion is fine-tuned by applying a different interpolation script between certain key frame types ('before'/'contact'/'after'). These scripts can also be different for each joints.

The example scripts shown in figure 5 create very convincing results, because the hand moves slightly 'behind' the lower arm; this results in a whip-like movement. These interpolation scripts are derived by observing the motion of a human drummer.

We have chosen for a GUI-based pose editor and script-based key frame interpolation. This proves to be a very flexible solution, since there are only a small number of poses, and they have to be set only once for a new drum kit configuration. The system could be extended with motion capturing, dynamics and inverse kinematics to create even more realistic behaviour [8], but at the cost of losing simplicity and flexibility. The interpolation scripts create natural motion, while the hand assignment algorithm ensures the arms will not cross. Motion capture would require the setup of the virtual drum kit

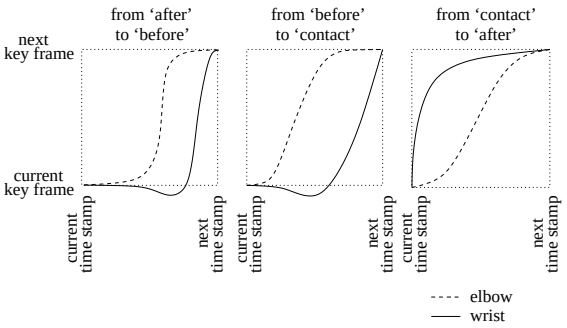


Figure 5: Example interpolation scripts for the elbow and the wrist joints

to exactly match the setup of the 'real' kit, so changes cannot easily be made. Additionally, our pose editor is used for other compound objects besides the avatar, e.g. the different parts of the bass pedal, the cymbal stands and the hihat.

### 3.5. Implementation Notes

The Java3D API is used for the implementation, because it is platform-independent and supports a wide range of geometry file formats. Moreover, the our virtual theatre [5] is currently being ported from VRML to Java3D.

The SMF format (Standard MIDI File) is used as intermediate file format between the percussion recogniser and the animation generator. A great advantage of using the SMF is, that it allows us to use MIDI files (which are widely available on the WWW) to test the animation generator independent from the percussion recognizer.

For the synchronisation of the animation and the sound, a seperate thread is used, which looks up the current audio position and adjusts the start time of the animation accordingly.

## 4. CONCLUSION AND FUTURE WORK

We have presented a system that generates a 3D animated virtual musician based on a music signal. The music signal is converted to MIDI, which proves to be a very convenient format to store abstract descriptions of animations.

Currently we have implemented a virtual drummer, but the our framework is general enough to control other types of animation through MIDI. Setting the poses of the drummer manually is a flexible alternative to inverse kinematics, and the stroke structure results in quite convincing movements.

The transformation from sound waves to MIDI events is far from trivial, and still requires a lot of work.

## 5. REFERENCES

[1] Babski, C. and D. Thalmann, "Define Virtual Humans On The Web," *Software Focus*, Wiley, Vol. 1, No1, pp.6-14

[2] Cemgil,Ali Taylan and Fikret Gurgen, "Classification of Musical Instrument Sounds Using Neural Networks," Department of Computer Engineering, Bogazii University, Istanbul Turkey, 1997.

[3] Kim, J., "Computer Animation of Pianist's Hand," *Eurographics '99*, Short Papers and Demos, Milan, 1999, 117-120.

[4] Kragtwijk, M., "Recognition of Percussive Sounds Using Evolving Fuzzy Neural Networks," Report of a practical assignment, University of Otago, New Zealand, July 2000.

[5] Nijholt, A. and J. Hulstijn, "Multimodal Interactions with Agents in Virtual Worlds," Chapter 8 in *Future Directions for Intelligent Systems and Information Science*. N. Kasabov (ed.), Physica-Verlag: Studies in Fuzziness and Soft Computing, 2000, 148-173.

[6] Puckette, M., "Pure Data: Recent Progress," *Proceedings of the Third Intercollege Computer Music Festival*, Tokyo, pp.1-4, 1997.

[7] Sillanpää, J. et. al., "Recognition of Acoustic Noise Mixtures By Combined Bottom-Up and Top-Down Processing," *Proceedings of the European Signal Processing Conference EUSIPCO*, 2000.

[8] Zordan, Victor B., J.K. Hodgins, "Tracking and Modifying Upper-body Human Motion Data with Dynamic Simulation", *Computer Animation and Simulation '99*, Springer-Verlag Wien, 1999.