

# Music-Driven Character Animation

DANIELLE SAUER and YEE-HONG YANG

University of Alberta

---

Music-driven character animation extracts musical features from a song and uses them to create an animation. This article presents a system that builds a new animation directly from musical attributes, rather than simply synchronizing it to the music like similar systems. Using a simple script that identifies the movements involved in the performance and their timing, the user can easily control the animation of characters. Another unique feature of the system is its ability to incorporate multiple characters into the same animation, both with synchronized and unsynchronized movements. A system that integrates Celtic dance movements is developed in this article. An evaluation of the results shows that the majority of animations are found to be appealing to viewers and that altering the music can change the attractiveness of the final result.

Categories and Subject Descriptors: J.5 [Arts and Humanities]—*Performing arts*; I.3.4 [Computer Graphics]: Graphics Utilities—*Software support*; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—*Animation*

General Terms: Design, Experimentation, Human Factors

Additional Key Words and Phrases: Character animation, motion synthesis, music analysis, primitive movements

## ACM Reference Format:

Sauer, D. and Yang, Y.-H. 2009. Music-driven character animation. *ACM Trans. Multimedia Comput. Commun. Appl.* 5, 4, Article 27 (October 2009), 16 pages. DOI = 10.1145/1596990.1596991 <http://doi.acm.org/10.1145/1596990.1596991>

---

## 1. INTRODUCTION

Animations, whether they are in movies, television, or video games, always capture the viewer's interest more if they are accompanied by music. Music has the capability of setting the mood for a scene and can alter the viewer's perception of what they are seeing. The ability to tie in the correct type of music with an animation is a difficult and time-consuming process. Not only is choosing the proper type of music important, but proper synchronization of music with the events in an animation is essential when attempting to secure the attention of a viewer. An interesting animation brings with it a "wow" factor, enticing the viewer to watch and appreciate the work. This can be achieved through a good combination of interesting movements and relevant music. This article proposes a method that attains this combination by using musical attributes such as the beat and dynamics to build an animation that fits user specifications and is tailored to the music. The user is able to choose any type of music they desire and create an animation that is not only automatically synchronized to the music, but also projects key elements of the music's intent as well. Our system provides a user-friendly method for

---

This research was supported by NSERC and AutoDesk.

Authors' address: Department of Computing Science, The University of Alberta, Edmonton AB Canada T6G 2E8; email: yang@cs.ualberta.ca.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or [permissions@acm.org](mailto:permissions@acm.org).

© 2009 ACM 1551-6857/2009/10-ART27 \$10.00 DOI 10.1145/1596990.1596991 <http://doi.acm.org/10.1145/1596990.1596991>

ACM Transactions on Multimedia Computing, Communications and Applications, Vol. 5, No. 4, Article 27, Publication date: October 2009.

creating a high-quality character animation where the user chooses predesigned movements to build a motion sequence. Through the use of a script file, the user can choose the order of specific movements and build a dance routine for a character, or set of characters, of their choosing. Our technique concentrates on using simple movements to create complex motion while providing maximum user control.

## 2. PREVIOUS WORK

Direct synchronization of an already existing animation with a piece of music is the technique most similar to our music-driven character animation method. The purpose of most synchronization methods is to take an already existing animation and synchronize it so that movement changes line up with beats in a given piece of music. Movement transition graphs are a popular technique for achieving this. Approaches proposed by Alankus et al. [2005] and Kim et al. [2003] use transition graphs to synthesize new motion sequences from motion capture data. The graph is used to choose which movements best fit with the beats of the music, as well as create a movement ordering where transitions between motions occur smoothly. Shiratori et al. [2004, 2006] also separate the original motion capture data into smaller sections for synthesizing new motion. In their method, rhythmic similarity between music and movement segments is used to synchronize the animation with the song. Some synchronization techniques use motion curves rather than motion data. Cardle et al. [2002] implemented a system that performs motion editing directly onto the keyframed motion signal. For example, motion warping can be mapped to the musical beat by adding a point to the displacement map for each beat, resulting in a jump in the signal at each displacement point.

It is our belief, along with others [Fod et al. 2002; Woch and Plamondon 2004], that complex motion can be simplified into a combination of basic movements called *primitives*. Dancing is a real-world example that supports this theory. Long dance sequences can be split into routines that consist of separate dance moves. The individual dance moves are the primitives that are combined together to create dance routines and performances. Fod et al. [2002] also implemented an algorithm for automatically detecting and segmenting primitives from movement data. Another method exists, in which the characterization of primitive movements is built from the kinematic theory and its deltalognormal model,  $\Delta\Delta$  [Woch and Plamondon 2004]. These techniques support the construction of complex motion from primitive movements as performed by our system.

MIDI files are the most popular input format for songs because it is less difficult to extract musical attributes from the data [Cardle et al. 2002; Taylor et al. 2005]. These files are not easily accessible and imply musical knowledge by the user, so we choose to use .wav files and use signal processing techniques to extract the tempo, beat onsets, and dynamics from the music datum. Beats can be considered regular pulsations of music and determine the tempo of the music, which is the overall pace of a composition of music, for example, fast, slow. Dynamics represent the variation of loud and soft levels in a song and establish the emotion of the music. Methods for performing beat detection vary but the use of tempo tracking is a popular technique. Both Scheirer [1998] and Dixon [2003] use tempo detection to reinforce their beat detection algorithms. Scheirer uses filterbanks and comb filters to track tempo changes and beat positions, while Dixon uses multiple agents and clustering. These techniques led to the development of our beat detection algorithm. We combine Tzanetaki et al.'s [2001] tempo detection algorithm with Goto's beat detection algorithm [Goto 2001; Goto and Muraoka 1994, 1999] to create a technique that uses the tempo to help determine the beat onsets. We develop our own algorithm for dynamics extraction using the power spectrum.

The contributions of this article are as follows:

—We develop a system that builds a new animation directly from musical attributes, rather than synchronizing an already existing animation to music.

- We present a signal processing-based onset detection algorithm that is based on the modification and adoption of two existing methods, as well as a dynamics extraction algorithm.
- We present a simple method using script file, which allows for the animation of several characters and the ability to specify and build different movement routines for each character.

### 3. PROPOSED METHOD

Our character animation system is made up of two principal components. The first is music analysis, where the musical attributes used by the system are extracted from the input music file. All music analysis work is performed in Matlab 7.1. The second component is character animation in the form of Celtic dance moves. This portion of the system controls the motion of the character in the scene, including the timing and expression of the movements. The animation system is built as a C++ plug-in for Autodesk's Maya, where Maya's interface is used to create the character, background, and lighting, and the plug-in is called to perform the movement. The result of these two components is a unique animation developed from movements chosen by the user and timed according to the music.

#### 3.1 Music Analysis

Music analysis involves analyzing an input signal and extracting specific musical features. The features extracted by this system include the tempo, beat positions, and dynamics. These attributes are the most recognizable aspects of a piece of music, especially to the untrained ear. Analysis on the song is performed by combining two different algorithms: Tzanetakis et al.'s [2001] tempo detection method and Goto's onset tracking method [Goto 2001; Goto and Muraoka 1994, 1999]. Tzanetakis' method was faithfully followed in the implementation, but several changes have been made to Goto's method in order to make it work better for our purposes. We chose to use these two algorithms because they had minimal complexity and met our system requirements, but it is important to note that other algorithms can also be used in their place.

**3.1.1 Beat Detection.** The process behind beat detection is analyzing a musical signal and finding the positions of all the beats. Onset detection is performed and then repetition is used to determine which onsets are the beats. Goto's original onset algorithm uses drum patterns and chord change information to make the system more robust, but we have not included these features in our implementation. Instead, we rely on Tzanetakis' tempo algorithm to give us more accuracy in determining a beat. Our system does not require the precision that Goto's algorithm strives for so we fashioned a simpler version of his system that runs in close to real time and does not require additional musical knowledge. The same values of parameters as those used in Goto's paper are used here as well. Our changes to his algorithm are discussed in the following. It is important to note that other onset detection algorithms can be used in place of this method. Examples of some algorithms can be found in Bello et al.'s [2005] review of onset detection in musical signals.

Goto's algorithm uses a power spectrum to extract the onsets from a signal with a sampling rate of 44.1 KHz. He uses the term *frame-time* to describe the unit of time used in his system, where one frame-time is equivalent to 11.61 ms or 44,100 samples. He divides the onsets into seven frequency bands for further analysis. This can result in a large number of possible onsets. To narrow down the range of possible onset positions, a threshold is used to remove onsets with the smallest amplitudes. This is based on the assumption that beat sounds are fairly high in amplitude compared to other musical features. The threshold is computed by multiplying the maximum value of each frequency band with a percentage value. The percentage value usually ranges from 80–90% of the signal's amplitude, meaning the onset components that fall within the highest 80–90% of the signal's amplitude are retained and the rest are discarded. It is important to note that the amplitude of the beat is dependent on the amplitude

Table I.

The results from performing beat detection with the new beat detection algorithm on ten synthetic signals with a tempo of 153.3682. Each signal was created with a different random seed and eight noise levels were used, ranging from 1/100 to 1/2 of the beep's amplitude

Signal #	Number of Beats Detected (/58) for Each Noise Level							
Noise/Signal	$1/100$	$1/50$	$1/25$	$1/16$	$1/8$	$1/6^1$	$1/4^2$	$1/2^3$
SNR (dB)	40	34	28	24	18	16	12	6
1	58	58	58	58	58	58	58	27
2	58	58	58	58	58	58	58	22
3	58	58	58	58	58	58	58	18
4	58	58	58	58	58	58	58	22
5	58	58	58	58	58	58	58	17
6	58	58	58	58	58	58	57	11
7	58	58	58	58	58	58	58	19
8	58	58	58	58	58	58	58	17
9	58	58	58	58	58	58	58	19
10	58	58	58	58	58	58	58	16
<b>Avg # Beats</b>	58	58	58	58	58	58	57.9	18.8
	<b>58</b>	<b>58</b>	<b>58</b>	<b>58</b>	<b>58</b>	<b>58</b>	<b>58</b>	<b>19</b>

of the signal. If the dynamics of the signal at a point in time are soft, then the amplitude of the beat will be low to match this, as will the amplitudes of the other musical features. This detail is the reasoning behind the choice of the percentage value. If the percentage value does not cover the softer ranges of music then the beats are not detected in those time intervals. The percentage value that works best for the threshold changes from song to song and is manually set based on experimentation.

The estimated onset times are then compared across frequency bands and only the positions where an onset has been detected in two or more bands are stored. The tempo detected by Tzanetakis' algorithm is then used to calculate an interbeat-interval (IBI). An IBI is the distance between two beats and can be approximated from the tempo. A direct relationship occurs between the speed of the song and the distance between beats and this relationship is used to compute the IBI directly from the tempo. The first onset is stored as the first true beat of the signal and used as a comparison point for the next onset in the list. The distance between this first actual beat and the next onset is calculated. If the distance is greater than an error threshold subtracted from the IBI, where the error threshold is five frame-times, then it is stored as the next actual beat in the signal. This distance threshold check ensures that the final beat positions are not too close together, as can be the case when the algorithm detects weak beats. The error threshold was chosen based on several tests and was determined to be a value that consistently worked well for any tempo. Weak beats are the beat sounds that occur between the actual beats of a song. They are generally found at twice the tempo rate and half the distance between two actual beats and can be mistaken by beat detection algorithms as real beats. Tracking of these beats is avoided by using the IBI to ensure only beat positions that occur around or further than the known interval are chosen. This procedure is followed for all the onsets in the list and the end result is a vector of actual beats for the entire song.

**3.1.2 Testing and Results.** Testing of the beat detection algorithm has to be performed manually in order to assure accuracy. Both visual data and audio data are used to compare the generated results with the true beat positions in the musical signals. Visual data is used for the synthesized signals where the beat positions are discernable. Table I displays the results for testing the beat detection algorithm on the ten synthetic signals with increasing amounts of noise. The algorithm uses a threshold value of

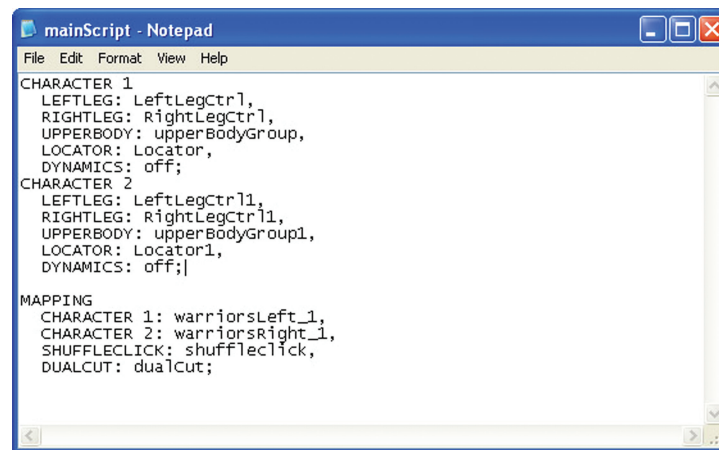
90% to obtain the majority of the results seen in Table I. The superscripts in the noise level row (the second row) denote that different threshold values were used to obtain these results than the threshold used in the first six columns. The threshold used by the algorithm in column 7 <sup>(1)</sup> is 72% while the threshold used in column 8 <sup>(2)</sup> is 65% and the threshold for column 9 <sup>(3)</sup> is 25%. This indicates that our algorithm is quite robust because its threshold value can be altered to reflect the signal. The results are consistently very good until the last noise level is reached, at which they drop off considerably.

Testing of each song or signal is performed by the system writing a synthetic signal at each position in the music where the beat is detected and then a user with extensive musical experience listening to the altered file to determine if the synthetic signals are placed in the correct positions. For each signal, the threshold value is initially set to 90%, which sometimes results in a few missing beats. It is adjusted manually in order to detect all the beats, but the manual adjustment is generally small. The threshold values for the songs used in Section 4.1 range from 80% to 90%, with four of the songs using a threshold of 90%. Although the threshold adjustment process would be easier for a musically trained ear, depending on the music and how distinguishable its beats are, most untrained users would be able to do a fairly good job in a short amount of time.

**3.1.3 Dynamics Extraction.** Dynamics consist of the louds and softs of the music, including transitions between the two that are also known as crescendos (soft to loud) and decrescendos (loud to soft). The dynamics levels are extracted because they are useful in creating corresponding movements. The purpose of this extraction algorithm is to detect the 50 positions in the music where the dynamic level is highest and 50 positions where the dynamic level is lowest. The fixed number of 100 total dynamics positions was chosen because it was felt that 50 of each was enough to get a good representation of the dynamics scheme of the piece. The algorithm will not retrieve as much dynamics information from longer pieces as it will from shorter songs, but it will still provide a good summary of the overall dynamics outline since the highest and lowest points of a song are chosen. These positions represent the loud and soft dynamics respectively. Dynamics are extracted by using a moving window with a size of 44,100 samples to compute the power spectrum of a music signal with a sampling rate of 44.1 kHz. The same analysis window is used for dynamics extraction as was used in the beat detection algorithm detailed in Section 3.1.1. The power spectrum is performed on the information in each window. The inverse FFT is performed on the outcome. Since the signal is symmetric, the second half of the signal is removed and the algorithm proceeds to calculate the absolute values for the signal's first half. The regional maxima are detected from the remaining values by using Matlab's *imregionalmax* method, and the highest peak and the lowest peak are added to a list before the window is moved. The highest and lowest peaks are chosen by taking the peaks in the window with the largest and smallest magnitude values. This technique is performed for each window until the entire signal has been analyzed, with the resulting list being comprised of the highest and lowest values from each window. Finally, the algorithm determines the 50 highest and lowest values in the temporary list and stores them as the dynamic positions. The system detects 50 of the highest and 50 of the lowest values because we believe that 100 dynamic positions are enough to build a complete representation of the dynamic structure of the song. Crescendos and decrescendos can also be represented by the dynamic positions. A transition over time from a high dynamic value to a low dynamic value signifies a decrescendo while a transition from a low dynamic value to a high dynamic value signifies a crescendo.

## 3.2 Celtic Animation System

This animation system integrates a unique music-driven approach to character animation. It generates an animation that looks like Celtic dancing, but is a unique variation of existing performances. Celtic dancing was chosen because it is an interesting and exciting dance where the movements are performed



```

mainScript - Notepad
File Edit Format View Help
CHARACTER 1
LEFTLEG: LeftLegCtrl,
RIGHTLEG: RightLegCtrl,
UPPERBODY: upperBodyGroup,
LOCATOR: Locator,
DYNAMICS: off;
CHARACTER 2
LEFTLEG: LeftLegCtrl1,
RIGHTLEG: RightLegCtrl1,
UPPERBODY: upperBodyGroup1,
LOCATOR: Locator1,
DYNAMICS: off;|
MAPPING
CHARACTER 1: warriorsLeft_1,
CHARACTER 2: warriorsRight_1,
SHUFFLECLICK: shuffleclick,
DUALCUT: dualcut;

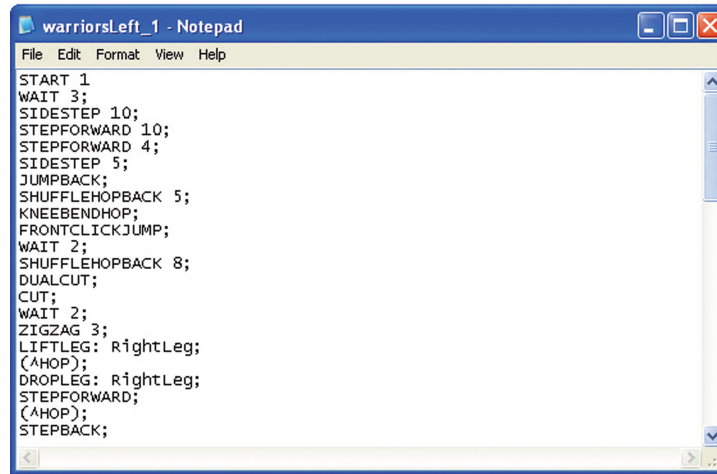
```

Fig. 1. An example of a master script file using two characters that are each mapped to different secondary script files.

almost entirely by the legs. Using only three major body parts (two legs and the torso) simplifies the system and allows us to concentrate on the main movements. The system is provided with knowledge of Celtic dancing, including several preprogrammed primitive movements and routines.

Producing high-quality character animation has proven to be difficult for inexperienced users. Animation systems such as Autodesk's Maya are intimidating for a new user because of the enormous number of features they provide. Setting up and animating a character is extremely time consuming and it generally takes practice and experience for a user to satisfactorily manipulate a human body. This system provides a user-friendly method for creating a high-quality character animation where the user chooses prebuilt movements to build a motion sequence. These movements are programmed into the system by an animator using C++. Through the use of a script file, the user can choose the order of specific movements and build a dance routine for a character, or set of characters, of their choosing. This ensures that they do not have to struggle with positioning character joints in order to achieve a specific motion. The system also gives the user the chance to experiment with different types of characters by supporting interchangeable characters. The user can change the appearance of the characters in the animation and easily use different characters in the same motion sequence. Maximum user control is provided by this system without relying on the user for the key components of the animation.

**3.2.1 Script Files.** Script files are utilized to give the user control over what occurs in the animation. They are simple text files that list Celtic primitives and routines that the user wants performed in the resulting animation. The system reads the script file using a specially designed parser and records each movement in the system as it is read in the script file. The script file is an easy and user-friendly method of allowing the user to create their own animation through a combination of built-in primitives, built-in routines, and user-designed routines. The script file is also designed to allow for multiple characters in a scene. There is no limit to the number of characters that can be specified by the user. The system is designed so that multiple characters can use the same script file to perform the same sequence of movements or they can use different script files to perform different animation sequences. There are two script files that are used to define the animation. The first script file is the master script (see Figure 1) and it defines the characters and which secondary script file each one uses. The secondary script file (see Figure 2) is used to define the animation by listing the movements in the order they should be performed.



```

warriorsLeft_1 - Notepad
File Edit Format View Help
START 1
WAIT 3;
SIDESTEP 10;
STEPFORWARD 10;
STEPFORWARD 4;
SIDESTEP 5;
JUMPBACK;
SHUFFLEHOPBACK 5;
KNEEBENDHOP;
FRONTCLICKJUMP;
WAIT 2;
SHUFFLEHOPBACK 8;
DUALCUT;
CUT;
WAIT 2;
ZIGZAG 3;
LIFTLEG: RightLeg;
(AHOP);
DROPLEG: RightLeg;
STEPFORWARD;
(AHOP);
STEPBACK;

```

Fig. 2. An example of a secondary script file in which the user designs their motion sequence.

The master script file provides the system with the information it needs to create the animation. It is used to link the system to the correct body parts on the character in Maya. The user relates the body parts on a character directly to the body parts needed by the system to perform the dance. This script file also performs mappings between each defined character and a secondary script file. The secondary script file that a character is mapped to will be the dance that the character performs in the animation.

The secondary script file provides a blueprint of how the animation will look. This script file defines the dance by using primitive movements, built-in routines and user-designed routines. The user can construct their motion sequence simply by listing the movements they want included in their final animation. Each primitive movement and routine has a corresponding name that needs to be specified in order to execute the motion. The user must stick to these naming conventions when creating the secondary script file or the correct movement will not be called.

Two or more physical files are used to create the animation definition because they allow for better organization than a single file, they permit easier integration of multiple characters into an animation, and they render it easier for the user to make changes. If users were constrained to a single physical file, all the movements for each character in the animation would have to be added to that file. Given an animation with several characters, the file would not be easily readable by the user, nor would it be simple to determine where changes need to be made. By having a master script file that organizes the animation at a high level and several secondary script files that organize it at a low level, the ease of use of the system increases.

In some cases it may be necessary to start the motion sequence at a certain video frame or divide the sequence into large intervals of time. The system is implemented so that the user can choose a start time for each segment using the keyword *START*. When a user adds this keyword into the secondary script file, the system will perform the first movement at the video frame number specified directly after *START*.

A movement or routine can be performed several times in a row by specifying the name of the movement and then the number of times it should be performed directly after it in the script line. There is no limit to the number of times a movement can be looped through. An example of this can be found in the second line of Figure 2. Along with animation timing, the system provides the ability to

control the timing of individual movements. The user can influence the timing of the movements through the application of brackets and rests. Brackets are used to indicate that more than one movement is performed at the same time. In many of the built-in routines, several movements are performed at the same time to create a realistic motion. The user can copy this by putting brackets around the movements occurring in the same time interval.

Rests are used to stagger the starting and stopping times for movements, as well as decrease their duration. The concept is adopted from music, where rests denote breaks between musical tones. The rest is specified in the system by the “`” character. Each rest is worth 1/8 of a beat, which means that the length of a rest will change from one piece of music to the next. The faster the song is, the shorter a rest will be. Rests can be placed before or after a movement name and one movement can use several rests. If the rest is placed before a movement, the movement will wait a fraction of a beat before beginning and its duration will decrease by the same amount. If the rest is placed behind a movement, it will end a fraction of a beat earlier.

*3.2.2 Mappings.* The main purpose of our animation system is to use music as the prime vehicle to drive an animation. Musical attributes such as the beat are mapped to Celtic movements and used to alter the motion based on the music. This system does not simply synchronize an already existing animation with a piece of music, but it actually builds the animation according to details extracted from the input song. Unlike synchronization methods, the movements in our system change along with the music. We create a final animation that is tailored to fit the music chosen by the user while providing an interesting and entertaining sequence of motion.

The timing of the movements is based almost entirely on the tempo of the music, where the faster the song, the faster the movements are performed. The position of the beats are inputted into the animation component by the music analysis component and used to determine the length of each movement’s time interval. Celtic dancing is a fairly high-speed dance where several movements occur in the space of one beat. In this system two primitive movements are performed for each beat. This rule applies to routines with multiple primitives as well. Several routines use three or four primitive movements and result in taking 1.5 or 2 beats to finish. Rather than performing each routine in a single beat, we choose to map two primitives to one beat because it provides smoother motion and better transitions between primitives.

Our system uses two units of time to coordinate the animation: video frames and beats. Beats are taken directly from the music and are made up of several video frames in sequential order. The START keyword is the only portion of the mapping process that uses the video frame unit. All movements and routines use the beat time units. Movements are triggered at the onset and are shaped across the beat by sine and cosine curves until the video frame preceding the next onset in the music is reached. The number of video frames within a beat depends on the speed of the music.

The dynamics extracted from the music can also be used to affect the movements used to dance to a particular song. In real life, small and timid motions are not used on a song that is loud, and large extreme motions are not used on a song that is consistently soft. This system is designed to take this into account by creating a dynamics range that is used to alter the movement of certain motions so that they correspond better to the music. The dynamics levels in the system range from 1 to 5, where 1 denotes soft dynamics and 5 denotes loud ones. There exist several primitive movements where the dynamics level affects the distance moved by a body part or the height of a jump or kick. The higher the dynamic level, the higher the height or the longer the distance will be. As the dynamics in the song change, the dynamics level corresponding to the current frame will change as well. If a movement is currently in progress, the system will not change the dynamics level. This is to prevent jerky motions during



the course of a movement. A change in dynamics level will only be incorporated at the beginning of a primitive movement. Incorporating dynamics levels into the performance helps to make the resulting movement more tailored to the input music.

**3.2.3 Constraints.** Foot position is an extremely important aspect of Celtic dance. It can help to determine the next movement in a motion sequence or the direction the character moves in around the stage. In many cases, the front foot is used as the starting foot for a routine or movement. This is the main reason that the system keeps track of which foot is in front and which is behind at each frame. We incorporate this Celtic knowledge into the system through the implementation of constraints. These constraints are used in some primitive movements and all built-in routines. Their purpose is to ensure that a primitive or routine is being performed by the correct body part according to the rules of Celtic dance. For example, in a built-in routine the constraints ensure that the primitives are performed by the correct body parts and in the correct order according to the Celtic routine it simulates.

An example of constraints used in a primitive movement can be found in the StepForward motion. This movement switches the front foot with the back foot by taking a step forward. Essentially, the back foot moves forward until it is positioned in front of the opposite foot, similar to a walking motion. A constraint is used in this movement to make sure that only the back foot performs it. The body part specified by the user to perform this movement is compared with the system's knowledge of which body part is currently the back foot, and the movement is either allowed or disallowed based on this information. It is important to note that Celtic movements are dependent on the horizontal positioning of the feet (front or back) rather than the vertical positioning (left or right). A body constraint is specified by the animator identifying in the movement itself which body part must be used for the motion.

More complicated constraints are used in the built-in routines. Body constraints are used to ensure that the correct foot is performing the correct movement. These constraints are especially dependent on the system's knowledge of which foot is in front when the front foot and back foot are constantly switching. Movement order constraints are used, along with Celtic knowledge, to make sure that the primitive movements involved in the routine are combined in the correct order. They are formally specified by the animator, who calls the primitive movements in the desired order and specifies both the movement start time and duration. The start time and duration are necessary because movements can overlap in a built-in routine and the animator can specify the timing of these movements. An example of this is the FrontClickJump routine. This routine performs a scissor-kick jump. The process starts with the front leg being lifted into the air. It is then lowered at the same time the back leg is lifted into the air, so the character hops in place. The back leg is finally lowered down to the ground to complete the motion sequence. Both body and movement order constraints are imperative for making this routine look as close to the real Celtic sequence as possible.

Constraints incorporate system knowledge of the positions of the character's feet with Celtic knowledge of how movements and routines should be performed. The use of constraints in a movement or routine is decided entirely by the animator and cannot be altered by the user. Constraints are used to enforce the integrity of Celtic dance and make it easier for the user to put together realistic motion.

**3.2.4 Routines.** The dance routines implemented in this system are more complex dance steps than those provided by the primitive movements. In many cases, Celtic dance has a dance step that consists of several primitive motions, but it is referred to by the name of the dance step rather than the primitives individually. Combining several primitive movements allows for these complex routines to be created and used by the system. The user can use these routines by specifying them by name. The routine will automatically call the appropriate primitive movements to create the movement. The system handles

two different types of routines. The first is the built-in routine, as programmed by the animator, and the second is a user-designed routine.

The built-in routine is implemented directly into the system by the animator. It makes use of several primitive movements and controls their timing to create an actual Celtic dance step. The purpose of a routine is to make the animating process easier for the user. Rather than having the user continuously specify small primitive movements in the same order, they can call a routine that does the same thing. Routines save the user time and frustration because the animator has already worked out the timing of the primitive movements so that the routine is correct. This makes it easier for the user to create an entire Celtic dance based on known Celtic movements. These routines are similar to how a person would learn to do Celtic dance and are taken directly from Dunne [1996]. The built-in routines are as follows:

—ClickZigZag	—Shufflehopback
—Cut	—SideStep
—CutBack	—SlidingStep
—FrontClickJump	—Turn
—JumpBack	—ZigZag
—KneeBendHop	

In some cases the user may want to use routines that are not implemented in the Celtic system. The system allows for user-designed routines in which the user can define their own routines through text files. The user can create their own dance moves by specifying primitives or built-in routines and their order. There is no maximum length limit for a routine, so the user is free to use as many primitives as necessary. The user-designed routine makes it easier to create an animation sequence because the user can define routines with combinations of movements that are continuously used in the animation. For example, if the user finds that they are constantly using three primitives in the same order in several places in their animation, they can put them into a routine. Rather than specifying the three primitives each time they want that specific combination, they can specify their specially designed routine instead. The system will retrieve the routine as input and perform the primitives found in that routine. Once the routine files are designed they can be reused in any animation and changed easily by the user.

The main difference between built-in routines and user-defined routines is the incorporation of Celtic knowledge. Built-in routines are made of primitives that are always put together in Celtic dance to create a specific dance step. The animator, according to the instruction video [Dunne 1996], has worked out the timing and order of the primitives with respect to each other. These routines are commonly used in Celtic dance and can provide a good starting point for a user with no Celtic experience. User-defined routines provide a way for the user to create their own dance steps that can be reused and altered. When a user finds that a certain combination of primitives and built-in routines work well together, they can put them into a built-in routine and call the routine within their animations. User-defined routines are an excellent experimentation tool for experienced users who want to create and save their own combinations of dance steps for later use.

*3.2.5 Primitive Movements.* It is our belief that small primitive movements can be combined to create more interesting and complex motions. One of the main purposes of the Celtic system is to demonstrate that any type of primitive movement can be combined with other primitives to create an interesting sequence of motions. The primitive movements implemented in this system were determined by studying videos of Celtic dancing and establishing the simple movements that make up the larger routines [Dunne 1996]. A total of twenty-four primitive movements have been implemented into the system. They can be used in different combinations to create routines. The primitive movements are listed as follows:



Fig. 3. Sequential images displaying the different positions involved in the “FrontClickJump” Celtic routine.

—ClickHeelsIn	—LongStep
—ClickHeelsOut	—ShortHop
—Cross	—SlideBehindStep
—CutBend	—Stamp
—DropLegBehind	—StepForward
—DropLeg	—StampDown
—HeelsUp	—StepBack
—HeelsDown	—SwingHeelsIn
—Hop	—SwingHeelsOut
—HopForward	—TapOut
—KneeBend	—TapBack
—LiftLeg	—Wait

#### 4. RESULTS AND EVALUATION

The main purpose of this animation system is to create a unique animation with the structure of a Celtic dance, but which is tailored to suit the chosen music. The resulting animation needs to be interesting, exciting, and expressive of the corresponding music. Our results show that primitive movements can be grouped in different combinations to create appealing motion. Figure 3 displays the “FrontClickJump” routine, which is built from three primitive movements: “LiftLeg,” “DropLeg,” and “Hop.” The use of these three primitives on different body parts and in a certain order creates one of the most interesting Celtic routines.

Some animations were created where multiple characters are involved in the dance performance. The Celtic system supports two types of multiple character movement: synchronized and unsynchronized.

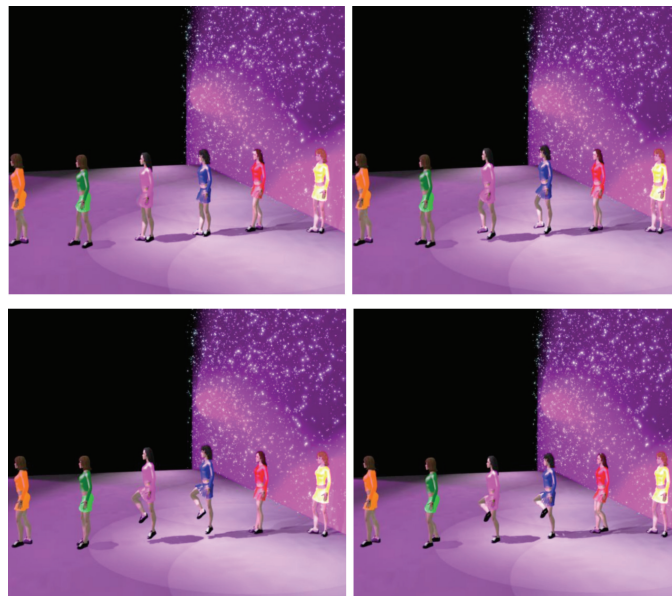


Fig. 4. Results from six characters performing unsynchronized movement. The characters are split into three groups of two, with each group performing a routine different from the other groups.

Synchronized movement involves all the characters performing the same movement at the same time. Figure 4 demonstrates unsynchronized movements of six dancers. The first and sixth characters are performing a “Sidestep” movement in all the images, while the second and fifth characters are performing a “Shufflehopback” routine. The third and fourth characters are performing a “Cut” motion. Each group of characters is performing at the same time as the other groups but their movements are not the same, resulting in an unsynchronized performance. These results demonstrate how different characters can possess different personalities and yet still fit into the overall presentation.

#### 4.1 Evaluation

The evaluation of a piece of music or a dance performance is generally subjective and extremely dependent on the preferences of the listener or viewer. This makes it exceptionally difficult to quantitatively determine if an animation is good or not. A qualitative evaluation was designed to assess the success of the Celtic system. There are two objectives in performing this evaluation. The first is to determine if the approach taken by the Celtic system is successful in creating appealing animations. The second is to establish if changing the music can also create appealing animations.

The evaluation involves three groups of six subjects per group. Each group represents a different user background. The first group incorporates subjects with dancing experience, including three subjects with extensive experience in Celtic dance. These subjects apply their knowledge of movement to determine if an animation is good or not. The second group includes subjects with computer programming experience and no knowledge of Celtic dance. This group of subjects has a technical background and will view the animations less artistically than the previous group. They will be able to focus on how well the parts fit together rather than concentrating on how accurate the movements are. The third group consists of subjects with neither Celtic dancing nor programming experience. These subjects can view the animations without any previous prejudices or expectations and are representative of an inexperienced user who may find the system useful. The knowledge of the subjects in each group

Table II.  
Overall results of the evaluation, taking into account the responses of all 18 people involved in the assessment of the animations

Animation	Number of “yes” responses	Number of “no” Responses	Percentage of People Who Liked the Animation	Tempo of Song (bpm)
BrownEyedGirl	12	6	67%	76
Eminem	9	9	50%	171
FieryNights	17	1	94%	110
Finale	13	5	72%	113
GetItStarted	14	4	78%	105
Nutcracker	7	11	39%	67
Warriors	16	2	89%	138
WideOpenSpaces	14	4	78%	90

Table III.  
Results of the evaluation split up by group into evaluators with dancing experience, evaluators with computer programming experience, and evaluators with experience in neither

Animation	Dancing Experience		Computer Programming Experience		Neither	
	Yes	No	Yes	No	Yes	No
BrownEyedGirl	5	1	4	2	3	3
Eminem	4	2	1	5	4	2
FieryNights	5	1	6	0	6	0
Finale	4	2	4	2	5	1
GetItStarted	4	2	6	0	4	2
Nutcracker	3	3	1	5	3	3
Warriors	6	0	4	2	6	0
WideOpenSpaces	4	2	6	0	4	2

does not cross over into other groups. For example, none of the dancers have experience in computer programming.

The evaluation involves eight animation videos with a single dancer in each. One of our objectives is to determine how different music affects the end result, so a different piece of music is used for each video, but the animation remains the same from video to video. The music types used include celtic, hip-hop, rap, rock, country, and classical. The tempos range from 67 bpm to 171 bpm. Each evaluator was given the same set of instructions for performing the evaluation. The evaluator is asked to specify for each animation whether or not they liked the animation. The answer choices are a simple “yes” or “no.” They are then asked to state reasons for their answer. The reasons can give us a good idea of how a subject’s background affects their opinion. The evaluation document requests that the subject form an opinion based solely on the merits of a single animation, without comparison to other animations. The evaluation concentrates on determining how successful our approach is by observing how the changing system parameters affect the subject’s opinion. Tables II and III display the results of the evaluation according to the responses of all 18 participants.

The two animations with the highest number of ‘yes’ answers are both animations using Celtic music. As noted in Table II, the FieryNights animation was found appealing by 94% of the evaluators, while the Warriors animation was appreciated by 89% of the evaluators. It is interesting to note that the animations with the highest tempo (Eminem at 171 bpm) and the lowest tempo (Nutcracker at 67 bpm) are the animations found the least appealing by the majority of evaluators. The Eminem animation was only enjoyed by 50% of the evaluators, while the Nutcracker animation was liked by only 39%.

These songs, however, also belong to musical types that do not typically suit dancing. Both rap and classical are difficult styles for an average person to dance to, so it makes sense that most people would feel that the dancing does not suit the music. The majority of respondents enjoy the remaining four animations, all of which correspond to music types that are traditionally easy to dance to. `GetItStarted` and `WideOpenSpaces` were appealing to 78% of evaluators, 72% of participants enjoyed the `Finale` animation, while `BrownEyedGirl` was appreciated by 65% of those involved.

The results from Table III have been divided based on their respective evaluator groupings. Several animations exist where all members of a group have found the result appealing. Participants with previous dancing experience enjoy `Warriors` best, with `FieryNights` and `BrownEyedGirl` tied for second. Those with computer programming experience enjoy `FieryNights`, `GetItStarted`, and `WideOpenSpaces` the most of all the animations. Evaluators with no experience like `FieryNights` and `Warriors` the best, with `Finale` a close second. It is interesting to note that the animations liked best by the programming group all fall within the tempo range of 90-110 bpm. The participants with no experience overwhelmingly enjoy the animations with Celtic style music the most. The group of dancers also seems to enjoy the animations with Celtic style, as two of the top three animations were paired with Celtic music.

## 5. CONCLUSIONS AND FUTURE WORK

This article presents a new music-driven character animation system that supports data-driven mappings of musical features to movements. The system helps users of all experience levels to produce appealing animations based on input music of any type and primitive dance moves and routines. One of the major contributions of this work to the area of character animation is its ability to build a motion sequence directly from extracted musical features. Unlike synchronization-based methods that simply alter an existing animation's timing in accordance to the musical beat, this system creates movements based on the musical beats and dynamics. The movements are automatically changed to reflect the mood and timing of the music, a feature that is not possible in systems similar to ours.

Another feature that is not supported in other systems is the ability to control multiple characters with different personalities in an animation. The user can build and easily integrate a troupe of dancers into the system (see Figure 5). The dancers are not limited to performing the same movements, as the Celtic system is set up so that each character can use its own script file. Synchronization between characters is encouraged, but individuality makes the animation less mundane. It is possible, in principle, for other systems to support multiple characters of different personalities by synchronizing them individually, but this does not allow choreographers to easily design dance moves that flow naturally between the characters. Individual synchronization does not support an animation where the dancers are coordinated because it will not allow for complex routines or dance patterns between several characters. A successful dance is made of several dancers performing a routine designed specifically for a group, not several dancers each performing an individual routine with no consideration for the others, or how their routine fits into the dance as a whole.

Our system is designed to be flexible for both the user and the animator. The system is set up to support extra primitive movements, as well as more dance types than just Celtic. The addition of other types of movements will encourage experimentation between dance structures, allowing a choreographer to easily mix moves from across different dance categories. Flexibility for the user is provided through both the script file and the musical input. Any type of music with noticeable beats can be used by the system to generate a specifically tailored animation that expresses the music. The script file gives the user a high level of control over the final animation and results that reflect their style and preference.

Future work is planned for both the musical analysis section and the animation component. The occasional inaccuracy of the beat detection algorithm needs to be addressed, as well as its need for manual tweaking. The tempo detection algorithm currently only distinguishes a single tempo measure



Fig. 5. The system is easily able to accommodate multiple characters in the same scene, as demonstrated in this picture. Sixteen girls are utilized in this particular performance.

for the entire piece, which limits our beat detection to songs without changes in tempo. Future work on this algorithm will include the ability to handle varying tempos. We also plan to design an automatic algorithm that is more accurate, as well as extract more musical features, such as the note pitch and melody, from the input signal.

We also intend to add more primitives and routines into the system in order to more faithfully represent Celtic dance. The system does not need to be limited to Celtic motion, however. Different types of dances can be added to future versions in order to increase the scope of the system and encourage experimentation between styles. Ballroom dances such as the waltz or culture-based dances such as the Spanish flamenco are among the possible dance types that could be incorporated into the Celtic system.

Last, the ability to randomly generate sections of a dance, or even an entire dance, automatically is a concept that should be included in the Celtic system. This function can be used to demonstrate the system to new users or fill in movements when a user has run out of ideas. It would increase the flexibility of the system and provide extra help for users with little experience or only a short amount of time.

#### ACKNOWLEDGMENTS

The authors would like to thank detailed comments provided by the reviewers to improve the quality of this article.

#### REFERENCES

- ALANKUS, G., BAYAZIT, A. A., AND BAYAZIT, B. 2005. Automated motion synthesis for dancing characters. *Comput. Animation Virtual Worlds* 16, 3-4, 259–271.
- BELLO, J. P., DAUDET, L., ABDALLAH, S., DUXBURY, C., DAVIES, M., AND SANDLER, M.B. 2005. A Tutorial on onset detection in music signals. *IEEE Trans. Speech Audio Proc.* 13, 5, 1035–1047.

- CARDLE, M., BARTHE, L., BROOKS, S., AND ROBINSON, P. 2002. Music-driven motion editing: local motion transformation guided by music analysis. In *Proceedings of Eurographics*. 38–44.
- DIXON, S. 2003. On the analysis of musical expression in audio signals. In *Proceedings of the Conference on Storage and Retrieval for Media Databases*. vol. 5021. 122–132.
- DUNNE, C. 1996. *Celtic Feet*. Acorn Media.
- FOD, A., MATARIC, M. J., AND JENKINS, O. 2002. Automated derivation of primitives for movement classification. *Autonomous Robots* 12, 1, 39–54.
- GOTO, M. 2001. An audio-based real-time beat tracking system for music with or without drum-sounds. *J. New Music Res.* 30, 2, 159–171.
- GOTO, M. AND MURAOKA, Y. 1994. A beat tracking system for acoustic signals of music. In *Proceedings of ACM Multimedia*. 365–372.
- GOTO, M. AND MURAOKA, Y. 1999. Real-time beat tracking for drumless audio signals: chord change detection for musical decisions. *Speech Comm.* 27, 3-4, 311–335.
- KIM, T., IL PARK, S., AND SHIN, S. Y. 2003. Rhythmic-motion synthesis based on motion-beat analysis. *ACM Trans. Graph.* 22, 3, 392–401.
- SCHWEIRER, E. 1998. Tempo and beat analysis of acoustic musical signals. *J. Acoustical Soc. Amer.* 103:1, 588–601.
- SHIRATORI, T., NAKAZAWA, A., AND IKEUCHI, K. 2004. Detecting dance motion structure through music analysis. In *Proceedings of the International Conference on Face and Gesture Recognition*. 857–862.
- SHIRATORI, T., NAKAZAWA, A., AND IKEUCHI, K. 2006. Dancing-to-music character animation. *Eurographics* 25, 3.
- TAYLOR, R., TORRES, D., AND BOULANGER, P. 2005. Using music to interact with a virtual character. In *Proceedings of New Interfaces for Musical Expressions*. 220–223.
- TZANETAKIS, G., ESSL, G., AND COOK, P. 2001. Audio analysis using the discrete wavelet transform. In *Proceedings of the WSES International Conference on Acoustics and Music: Theory and Applications*.
- WOCH, A. AND PLAMONDON, R. 2004. Using the framework of the kinematic theory for definition of a movement primitive. *Motor Contr.* 18, 547–557.

Received January 2007; revised September 2007; accepted October 2007