# BEAT-SYNC-MASH-CODER: A WEB APPLICATION FOR REAL-TIME CREATION OF BEAT-SYNCHRONOUS MUSIC MASHUPS

*Garth Griffin, Youngmoo E. Kim*

Drexel University
Music & Entertainment Technology Laboratory
Philadelphia, PA

*Douglas Turnbull*

Swarthmore College
Computer Science Department
Swarthmore, PA

## ABSTRACT

We present the Beat-Sync-Mash-Coder,[1] a new tool for semi-automated real-time creation of beat-synchronous music mashups. We combine phase vocoder and beat tracker technology to automate the task of synchronizing clips. Freeing the user from this task allows us to replace the traditional audio editing paradigm of the Digital Audio Workstation with an intuitive clip selection interface. The application is completely web-based and operates in the ubiquitous cross-platform Flash framework. The efficiency of our implementation is reflected in performance tests, which demonstrate that the system can sustain real-time phase vocoding of 5-9 simultaneous audio signals on consumer-level hardware. This allows the user to easily create dynamic, intricate and musically coherent acoustic soundscapes. Based on an initial user study with 24 high school students, we also find that the Beat-Sync-Mash-Coder is engaging and can get students excited about music and technology.

***Index Terms—*** music, interactive systems, phase vocoder, beat tracker, mashup

## 1. INTRODUCTION

The practice of combining many snippets of video or audio into a new piece of digital art is a phenomenon that has recently been gaining interest. For example, Israeli musician and producer Ophir Kutiel (stage name 'Kutiman') received international media attention [1] for his ingenious music videos comprised entirely of fragments from YouTube clips.[2] Projects like Kutiman's have been dubbed *mashups*, a term we will adopt throughout this paper.

Creating a mashup is laborious and time-consuming. Source material must be located, cut up, and spliced together into a new work of art. For the result to be musically cohesive, every shred must be meticulously synchronized. We focus exclusively on audio mashups and look to the domain of Digital Signal Processing (DSP) for ways to help facilitate the creative process.

[1] http://music.ece.drexel.edu/bsmc

[2] http:// www.thru-you.com

In general, mashups are created in one of two ways. The first is by combining very small units of sound (100-200ms) into an *audio collage* according to some criteria. The user specifies the desired audio output (e.g. recreating Led Zeppelin's "Stairway to Heaven" from chime and bell sounds), and the computer chooses samples and constructs the waveform. This affords limited control and restricts creativity. The second is to have the user manually splice and layer the raw audio using a Digital Audio Workstation (DAW). A DAW is a digital version of the analog mixing board used in recording studios, and can be found in such products as GarageBand or ProTools. While the DAW is a powerful tool for manipulating an individual track, combining multiple tracks is a laborious process of innumerable miniscule adjustments. Our system uses DSP to automatically combine audio clips in a musically meaningful way. This provides the best of both worlds: the user chooses what clips are played when, thereby retaining expressive potential, but is freed from the burden of manually aligning and layering the clips.

The algorithms we need are quite computationally demanding. Most web applications requiring intensive DSP handle the computation on a powerful server using a fast native language like C. This centralized model fails to harness clients' increasingly powerful hardware, and lacks the scalability of performing the DSP on the client. However, platform-independent frameworks suitable for web development (such as Flash) typically add substantial overhead, which drastically restricts how much DSP can be done by the client. We circumvent this difficulty by using a new framework from Adobe called *Alchemy*, which allows C code to be compiled to optimized bytecode that can be executed in the cross-platform Flash environment. This provides the power of C without loss of platform independence, enabling our system to perform all the DSP client-side. This represents a new model for web-based DSP that improves scalability, lowers cost, and expands access [2].

## 2. RELATED WORK

There are a variety of audio collage applications. Zils and Pachet's *Musical Mosaicing* [3] treats sample selection as a

constraint problem, where the constraints could be a target song or a measure of concatenation quality for possible sample pairings. Lazier and Cook's *MoSievius* [4] selects clips with a technique called sound sieving and uses overlap/add to make an evolving audio mosaic with real-time control.

There are a number of websites that cater to DAW users, such as Jamglue[3], Kompoz[4], ACIDplanet[5], and ccMixter[6]. Typically, a website such as these bills itself as a community of users who share their music specifically so that other people can make remixes with it. They offer standard web community features such as message boards and searching, often with file upload/dowload capability, and a few even include an online DAW.

## 3. SYSTEM DESIGN

The system relies on two DSP modules, a phase vocoder and a beat tracker. The core functionality of the system is the automatic, synchronized playback of multiple clips using the phase vocoder (Fig. 1C, Fig. 2). In order to perform this operation, the system must have the beat locations for each playing clip. We obtain this information with the beat tracker whenever a user adds a new clip (Fig. 1A). The server (Fig. 1B) stores both the audio and beat data for every clip, but does not perform any DSP. We use the de facto audio standard of 44100 samples per second.

### 3.1. Phase Vocoder

Phase vocoding enables pitch-invariant time-scaling. This allows the system to change the tempo of each clip to match the user-specified target during playback, while minimally changing the user's perception of the pitch and timbre.

Our implementation is similar to [5], and consists of altering the phases of specific ranges in the frequency domain such that the perceived pitch of the signal is changed by a target ratio without changing the length of the audio. This time-invariant pitch-shifting can be combined with resampling for pitch-invariant time-scaling. For example, to scale a clip from 102 bpm to 110 bpm, we would need a pitch-invariant time-scaling of $\frac{110}{102} = 107.8\%$. This can be achieved by first resampling at a factor of 107.8% to reach the desired speed, and then pitch-shifting at a factor of $\frac{102}{110} = 92.7\%$ to correct the change in pitch. The order of the two operations is irrelevant.

Our system allows real-time control by operating incrementally. For a given call to the phase vocoder module we only compute enough output to fill the output buffer. As is standard practice, we overlap and add every frame with its predecessors. This necessitates dedicated input and output buffers for every active clip, allowing interruption and resumption without loss of data. We use a window of length 1024 samples (23 ms) and overlap 75%. These parameters were determined experimentally to maximize the number of simultaneous audio tracks without degrading the acoustic quality of the output.

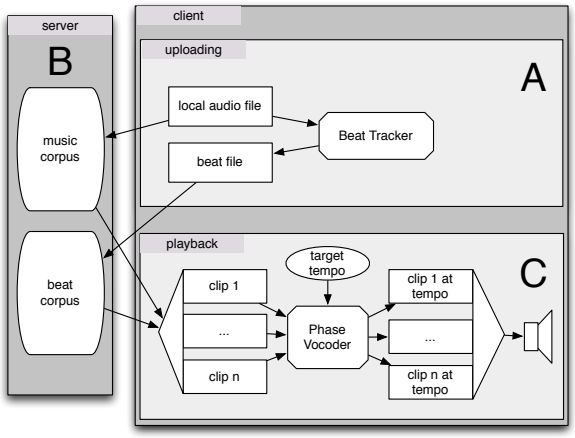


**Fig. 1**. Overview of system architecture: (A) A typical entry point is the user uploading a clip they wish to use in a mashup. The client-side beat tracker module analyzes the clip to find the beat locations. (B) Both the audio data and the beat data are uploaded to the server. The user may select several clips from the server to be mashed together. (C) Given a tempo input from the user, the system uses the beat information to determine the timescale ratio and start time for each clip, and the client-side phase vocoder module is used to play each clip at the target tempo, synchronized with the other clips.

### 3.2. Beat Tracker

Beat tracking enables the system to detect the locations of beats in an audio clip. The system must synchronize the beats of every clip during playback to ensure that the overall output is perceived as musically coherent. When a user uploads a clip, it is first analyzed by a beat tracker. The beat tracker is a modular component, and so the system can be configured to work with any beat tracking algorithm.

The first approach we tried was an implementation by Ellis [6] that uses dynamic programming. We found that it required ad-hoc post-processing tailored to our dataset. The second implementation was based on a technique developed by Scheirer [7], which uses a large bank of comb filters acting as resonators. This approach had better accuracy on our dataset, and is currently implemented in the system. However, our system can easily be configured to work with any number of beat-tracking approaches. As proof of concept for the modularity of the system, we also tested a third algorithm, developed by Jehan [8], which uses autocorrelation on a timbre-based auditory spectrogram. This algorithm is made publicly available as an API through EchoNest[7], and our system can use this API as well.

[3] http://www.jamglue.com
[4] http://www.kompoz.com
[5] http://www.acidplanet.com
[6] http://www.ccmixter.org
[7] http://echonest.com

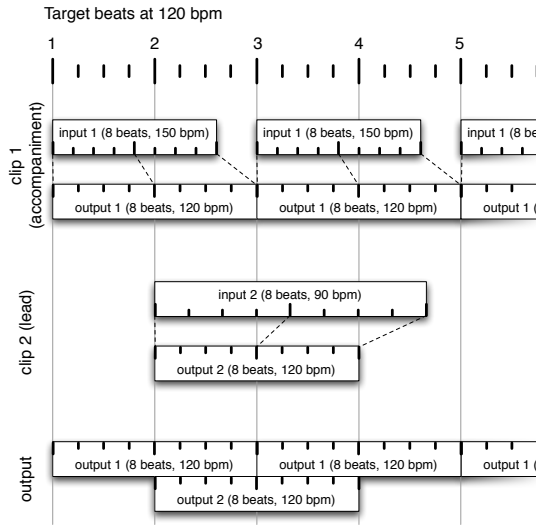


**Fig. 2**. Playback scenario for two clips: The first is an eight-beat accompaniment clip, which is looping. The second is an eight-beat lead clip, which is played once starting in the second measure. Both clips are timescaled from their original tempi to match the target of 120. Though the target tempo in this example is shown as constant, the user could change the tempo parameter at any time during playback and the system would adjust without needing to recompute any audio data.

### 3.3. Music and Beat Corpora

Clips are the auditory building blocks of our system. Currently, they are divided into two conceptual groups, designated "lead parts" and "accompaniment parts". Examples of lead parts are solo vocal tracks or solo melodic instruments. Examples of accompaniment parts are drum and bass tracks or chordal instrument parts. The system is designed such that additional categories could easily be implemented for increased specificity.

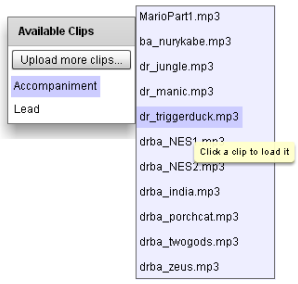


**Fig. 3**. Users select clips from the server-side repository and upload clips from their local machines.

Whenever a user wishes to add a clip to the system, the clip must be uploaded to the server. The system first analyzes the new track with a beat-tracker, and then uploads both the audio and the beat information. The user is also required to indicate the category of the clip (e.g. lead or accompaniment). Once a clip is uploaded to the server, it becomes part of the music corpus that is accessible to every user.

Each clip in the music corpus must have a corresponding entry in the beat corpus, which stores the location of each beat in the clip. In order to minimize server load, the beat-tracking computation is done on the client side before the audio is uploaded. Once computed, the beats are stored on the server so that the information is available to anyone wishing to use the clip. The beat locations are quantized by sample number.

### 3.4. Playback

Playback begins with clip selection. The user can manually choose each clip, instruct the system to randomly choose a number of each type of clip (e.g. two accompaniment and three leads), or have a mix of manually and automatically selected clips. Clips can be toggled between modes while playing. Automatically chosen clips are periodically swapped out according to the user's parameters. For example, A DJ might have the system automatically play a randomly chosen accompaniment that changes every 16 measures and manually add lead parts.

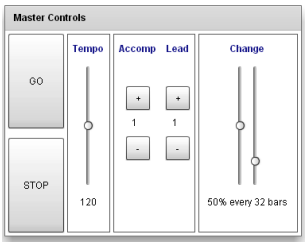


**Fig. 4**. A master control panel handles automatic addition and removal of clips, as well as the global tempo parameter.

Once a clip has been selected, its audio and beat data are read from the server into local memory. Up-front loading eases server demand and allows faster access to the audio data during playback. Flash does not have built-in support for multithreading, so when necessary we handle concurrent task execution by incrementally operating on each task once per render frame.

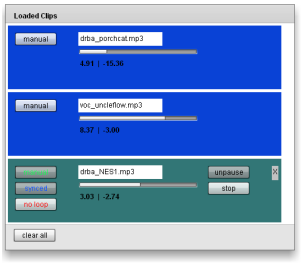


**Fig. 5**. Loaded clips are displayed with status information and buttons to control playback.

In order to adapt quickly to changes in parameters, the system computes output audio incrementally with a 4096-sample (92 ms) buffer. Increasing the length of the buffer would result in more efficient computation but also increase the latency associated with changing parameters. When nearly all the data in the buffer has been played, the system computes enough output to refill it.

The system keeps track of how long it has been since the last beat occurred, $L$. The tempo value specified by the user allows us to calculate the desired beat length, $B$. How much of the current beat has already been played is then $A = \frac{L}{B}$. For each clip $i$, we keep track of which beat was last played, and so can access its location as $beats_i[j]$. The length of the current beat in the original audio is then $b_i = beats_i[j+1] - beats_i[j]$. Thus, the ratio $r_i = \frac{B}{b_i}$ represents the timescaling ratio for clip $i$ to be at the desired tempo. We calculate the

**Table 1**. Performance on consumer-level hardware

| Processor | | Memory | | |
|---|---|---|---|---|
| Speed (gHz) | Type | Size (gb) | Speed (mHz) | Maximum simultaneous |
| 2.16 | Intel core duo | 2 | 667 | 5 |
| 2.0 | Intel core 2 duo | 1 | 667 | 6 |
| 2.4 | Intel core 2 quad | 8 | 800 | 7 |
| 2.66 | Intel core 2 duo | 4 | 800 | 9 |

location of playback within clip $i$ as $s_i = r_i A + beats_i[j]$. We now instruct the phase vocoder module to compute frames clip $i$, timescaled at ratio $r_i$, starting from $s_i$, until the output is the length of the buffer. The phase vocoder module updates the position in the source audio as it operates, ensuring that $j$ is updated as necessary. When this process has been done for every clip, all the outputs are summed and normalized, and the result is sent to the output buffer.

If the user moves the tempo slider, only $B$ changes in the above calculations. Adding or removing a clip is simply a matter of updating which clips are in the above loop. As soon as the buffer is refilled, the user will hear the change. We need not discard any of the audio that has already been computed, and the latency is at most 92 ms, which is barely perceptible to the human ear.

When a clip is added it must be aligned with the clips that are already playing. Currently, we assume that all clips are in 4/4 time. We set the playback position this clip such that its first beat will occur at the next perceived downbeat, with the effect of either truncating some samples or padding with silence. We can achieve a similar alignment when starting in the middle of the track by matching the next beat of the clip that is an even multiple of four with the location of the next perceived downbeat. Figure 2 shows the alignment operation of the system over a period of several measures.

## 4. PERFORMANCE EVALUATION

### 4.1. Benchmarks

We tested the performance of our system on various processors by adding clips to the playback stream at 120 bpm until the system could no longer play the audio (Table 1). These results were consistent across the major browsers and operating systems. They show that top of the line hardware is not necessary for this system to handle several simultaneous real-time phase vocoder operations.

### 4.2. User Study

In addition to the laboratory tests for speed, we also performed a user study. The subjects were high school students participating in a free one-week educational program.[8] Applicants were selected based on a demonstrated interest in music and technology, but not necessarily on the basis of skill in those areas. The program was divided into sessions, one of which was dedicated to the Beat-Sync-Mash-Coder. The students were given an hour to experiment with minimal assistance and supervision. Overall, students seemed to enjoy the experience, and understood the functionality provided by the system. In their feedback, some students offered general praise, writing "the whole concept is pretty cool" and "it was easy to use." Others indicated that they especially enjoyed the "tempo adjust" or "timestretching capability". One student said they liked that "you could choose which songs to load and pick sections that you wanted". Several ranked the Beat-Sync-Mash-Coder session as their favorite part of the five-day program.

[8] Summer Music Technology. Drexel University, Philadelphia PA. http://www.drexel.edu/smt

## 5. CONCLUSION

We present this system as a new interface for music mashup creation, breaking the paradigm of the traditional DAW through novel use of two DSP technologies. Delivering beat-tracking and real-time phase vocoding over the web brings cutting-edge technology to the fingertips of those who might not otherwise benefit from these technologies. We believe this system, and others like it, could serve as an inspiration to students at all educational levels with an interest in music and technology, enriching their lives and motivating discovery in the field of signal processing as a whole.

## 6. REFERENCES


[1] V. Heffernan, "World music," *New York Times*, 29 Apr 2009, 10 Sept 2009 `http://www.nytimes.com/2009/05/03/magazine/03wwln-medium-t.html`.

[2] T. Doll, R. Migneco, J. Scott, and Y. Kim, "An audio dsp toolkit for rapid application development in flash," in *International Workshop on Multimedia Signal Processing*, Brazil, 2009.

[3] A. Zils and F. Pachet, "Musical mosaicing," in *Conference on Digital Audio Effects*, Paris, 2001.

[4] A. Lazier and P. Cook, "Mosievius: Feature driven interactive audio mosaicing," in *Conference on Digital Audio Effects*, London, 2003.

[5] M. Dolson, "The phase vocoder: A tutorial," *Computer Music Journal*, vol. 10, no. 4, pp. 14–27, 1986.

[6] D. Ellis, "Beat tracking by dynamic programming," *New Music Research*, vol. 36, no. 1, pp. 51–60, 2007.

[7] E. Scheirer, "Tempo and beat analysis of acoustic musical signals," *Journal of the Acoustical Society of America*, vol. 103, no. 1, pp. 588–601, Jan 1998.

[8] T. Jehan, "Event-synchronous music analysis/synthesis," M.S. thesis, MIT, Massachusetts, USA, Sept 2001.