# A Malleable Drum

Christoph von Tycowicz[*] Jörn Loviscach[†]
Hochschule Bremen (University of Applied Sciences)

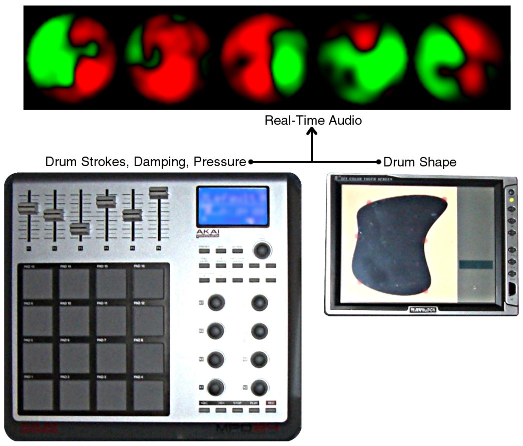

**Figure 1:** *The user can hit or press the drum at 16 locations and at the same time control the drum's shape on the touch panel.*

## 1 Introduction

We present a virtual drum with a large range of interaction modes: The user can change the shape of the drum while playing it through a controller that offers hitting the virtual drum skin at 16 different points as well as simultaneously damping it and pushing it in. The real-time, low-latency simulation leverages the computing power of a GPU, which allows the resolution of the mesh to be increased greatly. This and the interactive control of shape and local tension are differences to the work of Jones and Schloss [2007].

The synthesis of percussive sounds can be accomplished very efficiently through modal synthesis, where a set of eigenfrequencies is computed in a preprocessing step, see for instance Raghuvanshi and Lin [2006]. Modal synthesis, however, cannot easily handle changes in a drum's physical properties. More versatile options are finite-element methods and waveguide meshes [Murphy et al. 2007]. Aiming at a minimum complexity for the GPU-based implementation, we use a finite-difference method, as known from the simulation of shallow water waves.

## 2 Method

To input drum strokes and to control the local tension and damping by pressure, the MIDI controller Akai MPD24 is used. It can transmit the pressure independently for each of its 16 pads. The drum shape is edited on a touch panel display: The user can edit the position of anchor points, which define the rim's shape through a cardinal spline with adjustable tension.

The drum skin is modeled by a rectangular mesh of $64 \times 64$ point masses, each of which is connected to its four next neighbors with springs and dampers. We assume that each spring has a force constant $k$ and has been stretched from a rest length $l_0$ to a length $l_1$ when tuning the drum. Only the vertical elevation $h_i$ of each mesh element $i$ of the skin is taken into account, because it is more prominent around the rest state and it is more effective for sound radiation. Hence, the spring force onto the $i$th mesh element is $k\sum_{j\in N_i}\left(1-l_0^2/\sqrt{l_1^2+(h_j-h_i)^2}\right)(h_j-h_i)$, where $N_i$ is the 4-neighborhood of $i$. This is nonlinear in $h_i$; a hard drum stroke or the application of pressure to the skin increase the pitch.

Placing a finger on the drum causes a friction force of a constant times $-\dot{h}_i$. The skin's intrinsic damping force, which also helps to stabilize the integration, is modeled by a constant times $\sum_{j\in N_i}(\dot{h}_j-\dot{h}_i)$. We use a leapfrog integration, which is less precise in the short term than other methods, but does not lose energy in the long term, so that sounds that take several seconds to decay can be modeled reliably.

Each simulation step is accomplished through one render call. For sound output, the value of the central $\dot{h}_i$ is used. To optimize the communication with the CPU, 256 render calls are issued before the data is read back from the GPU. One audio sample results from each of these render calls; each sample is written to a different texel of an off-screen texture. We create two textures, each with four channels of 32-bit floating point numbers. One texture is used for reading, the other for writing; their roles are exchanged after each simulation step. The texture channels store the heights, the velocities, the audio samples computed so far, and—as sign and magnitude within one channel—the applied pressure and the binary mask of the rim's shape. Each time the data is read from the GPU, the height field is also displayed as a red/green pattern. If the user hits a drum pad, an appropriately placed soft disk is rendered into the height channel of the texture. This render command is issued immediately after the render command for the current audio sample; the system does not wait for the current chunk of 256 samples to finish.

The touch panel to control the rim's shape is connected as a second display. The audio output is realized through an low-latency ASIO audio interface as used for professional music making. The prototype software has been built in C# and HLSL.

## 3 Results and Outlook

On a PC equipped with a 2.4 GHz dual-core processor (AMD Athlon 64 X2 4600+) and an Nvidia GeForce 8800 GTS, the prototype software can sustain a sampling rate of 22,050 Hz with a block size of 12 ms. The sound can be varied among kettledrum, metal box and unheard-of percussive noises. Future work may look into proprietary GPGPU support such as AMD CTM and Nvidia CUDA. To increase the accuracy, one may use a triangular mesh and model the resonance body of a closed drum.

## References

JONES, R., AND SCHLOSS, A. 2007. Controlling a physical model with a 2D force matrix. In *Proc. of NIME '07*, 27–30.

MURPHY, D., KELLONIEMI, A., MULLEN, J., AND SHELLEY, S. 2007. Acoustic modeling using the digital waveguide mesh. *IEEE Signal Processing Magazine 24*, 2, 55–66.

RAGHUVANSHI, N., AND LIN, M. C. 2006. Interactive sound synthesis for large scale environments. In *Proc. of I3D '06*, 101–108.

[*]e-mail: izual@fbe.hs-bremen.de
[†]e-mail:joern.loviscach@hs-bremen.de