Audio Hockey Table

Chet Gnegy

November 18, 2013

1 Introduction

The purpose of this project is to create a real-time effects processor with a graphical interface that integrates the physics of real objects as well as the actions of the user into the parameter space of the signal chain. The basic design is as follows: There are a bunch of different modules that can be placed in a 2D plane, each of these modules has physical properties assigned to it and can impact the sound of some input source. The modules are represented by short cylinders, called Discs. The Discs are each associated with some unit generator, whether it be an input source or an audio effect. The proximity of the Discs to one another will determine the signal path as well as the mix level for the effects. The user, who can interact with the environment through clicking, can drag the Discs around the world, changing the parameter space. The Discs are bound to the laws of physics, however, and will bounce off of one another as well as with the walls of the environment. The physics engine will also compute friction and angular velocities, the latter of which could also be mapped to a parameter of the unit generators.

2 Audio Framework

This software utilizes the RtAudio[1] engine, which conveniently allows the system to communicate with the sound card by periodically filling buffers with audio data. To keep the system modular, the graphical display and the audio components are completely independent of each other and interact only through the parameter space of the Disc objects. The RtAudio callbacks are handled through by UGenChain class. UGenChain contains some data structure SPAGHETTI CAT!!of UnitGenerators that pass the audio signal from the input source to the next element in the chain and finally to the array designated as the output buffer.

2.1 Signal Path

The class UGenChain, is responsible for three main points: handling the audio stream, providing the next sample of audio data on request, and making sure that Midi information is delegated to the signal flow graph.
The first of these responsibilities involves only initializing the RtAudio engine and opening the audio stream. The provision of the next sample is fairly straightforward given that the chain of unit generators. We simply "tick", or compute one sample, of audio from each unit generator in order and pass the result from the next. We of course start with an input source, who simply returns a sample of the audio input. Midi sources return a single frame of audio data based on the Midi information provided and the type of waveform. The effects units are similar but involve some calculation, often state-based depending on its parameters, and also on the samples that have been processed by it at previous times.
Finally, the signal chain is determined. This is done by the UGenGraphBuilder class. The UGenGraphBuilder considers the locations of the Discs in the world and contracts a Minimal Spanning Tree, where the Discs are treated as Nodes, and the connections between them as Edges.

2.2 Unit Generators

Several unit generators have been defined for this software. Many of them are loosely based around well known algorithms. The unit generators feature two main methods, "tick" and "set_params". As previously mentioned, "tick" processes a single sample of audio data. The method "set_params" allows for the parameters of the unit generator to be changed. This does not include the parameters that are determined by the position of the Discs, but some internal parameters to the unit generator. The "Input" unit generator is trivial and simply returns the current sample of audio from the input buffer.

Midi Unit Generators

Several Midi Unit Generators are provided, all using classic waveform generators to provide audio data to the other modules. The Midi data is passed from a callback in the UGenChain to the UGenGraphBuilder and then to the individual Midi modules. Each midi module has an attack and sustain parameter allowing some configuration of the sound.

Bit Crusher

The bit crusher effect is very handy for reproducing vintage low-fi audio effects. The audio signal, typical 16 bits in resolution, is quantized down to a specified level on the range of 1 to 16, 16 being the unquantized signal. By using a value of 8 bits, we can achieve "chip music" sounds that resemble those of older game systems. The bit crusher also features a downsampling parameter which effectively reduces the sampling rate of the signal. We can downsample by an integer number on the range 1 to 16, where we quickly experience the effects of aliasing as this parameter increases.

Chorus

The chorusing effect used is modeled as described in Jon Dattorro's paper on delay-line modulation[2]. The effect is achieved by summing the dry signal, $x(t)$, with a delayed copy of itself, $x(t-\tau(t, f_c, d_c))$, where $\tau(t, f_c, d_c) = d_c \sin(f_c t)$. The modulation rate, f_c is bounded on the range 0.02 - 10.0 Hz, and the depth of the effect,d is bounded by 0.0125. There is also a negative feedback path with a delay of $\tau(t, f_c, 0)$, corresponding to the average delay length of the feedforward path. The weighting of each of these paths is given by Dattorro's "white chorus".

Delay

This is a simple delay line. The time in seconds can be changed as well as the amount of feedback.

Distortion

SPAGHETTI CAT!!

Filters

There is a choice of low pass, high pass, and band pass filters, all selectable using the Filter and Bandpass unit generators. The Filter ugen can be toggled between high and low pass. For all filters, the cutoff frequency and Q can be changed.

Looper

The looper waits for the user to trigger a start event and begins to count down from 4. It then records its input for a given number of beats at a specified tempo. Immediately after completing the recording, it begins to replay the buffer on repeat. Effects can be applied to the looper as if it is an input.

Ring Modulator

The ring modulator multiplies the input by a simple sinusoid. The frequency of which is chosen by the user.

Reverb

This is an implementation of Freeverb[3] using feedback comb filters and all pass filters. The size of the simulated room and the damping can be altered in real time.

Tremolo

The tremolo module provides simple low frequency amplitude modulation with a sinusoidal carrier wave. The modulation rate, f_t is bounded on the range 0.02 - 10.0 Hz.

3 Graphics

The OpenGL library is used to provide a graphical interface for the project. The two main classes of displayable objects are the "World" and the "Disc". Each instance of these classes implements an interface Drawable, which allows for clean integration with the OpenGL module. OpenGL need only hold on to a list of Drawable objects, and the individual classes are responsible for their own rendering instructions. The drawable objects have an associated priority. High priority objects sit at the beginning of the list and are drawn first. The menu and the world have the highest priority, followed by the discs and the particles that float around them, called orbs.

3.1 The Drawable Interface

Drawable is an abstract class containing only pure virtual methods. These methods are:

- draw() - draws the object, with its center around the point (0,0,0) and with no rotation
- get_origin(&x, &y, &z) - stores the origin of the object in the passed coordinates. This allows the object to be shifted to where it thinks it should be externally to its draw method.
- get_rotation(&x, &y, &z) - rotates the object to the coordinates specified by its internal parameters
- set_attributes() - must be called before drawing, sets up OpenGL settings and pushes them to the attributes stack.
- remove_attributes() - must be called after drawing, pops the attributes stack
- prepare_graphics() - called as soon as item is added to the list of drawables

3.2 World

The world consists of a square shaped grid with wireframe boxes as walls. There is a glowing grid along the bottom of the world with orbs that randomly traverse from one side to the other. Both the grid and the orbs are implemented as textures. The grid and the orbs brighten and fade in sync with each other. The borders glow out of phase with the center of the grid to give the appearance that energy is being transferred from one to the other.

3.3 Discs

The discs are simple cylinders with orbs floating around them. The discs feature an image that details what the disc's purpose is. For example, the audio input disc has an audio waveform on it, and the delay disc has an image of a stop watch. The input discs, as well as the looper disc have orbs that float around them. When sounds are played, the orbs are passed along the signal chain. The discs move with physics, they bump off of each other and also with the walls of the world. The mouse can be used them to drag the discs around the world. The discs are pulled with a damped spring force. The damping makes them easier to control because it prevents them from oscillating.

3.4 Orbs

The orbs are made from textures that use additive blending. They do not interact with the world, other than when they are passed from disc to disc. The orbs are given an anchor point, usually the center of a disc and they are pulled around as the disc moves. The orbs have a repulsive force, an inverse square law, that keeps them from getting to close to the center of the disc as well as an attractive force that keeps them from straying too far. They maintain an average distance of twice the radius of the disc from the disc's center. They also wander around randomly within this region to make it more visually appealing. We use a damping force to reduce oscillations just as we did for the discs. The discs can be assigned to any disc, or none at all. If they are unassigned, they fly upwards towards the viewer. Once they are out of the field of view, they delete themselves and all references to them and also free the associated memory.

4 User Interaction

The graphics module also contains a list of items that implement an interface, Moveable. As the name suggests, these are items that can be moved by the user. When the user clicks on the screen, a ray is casted into the screen from the camera through the point where the user has clicked. The coordinate at which it intersects the plane containing the top face of the Discs is returned. This is done using OpenGL's unproject functionality. When a disc is clicked, an offset from the center is stored so that the object moves around the clicked point rather than the center of mass. There is also a menu on the left half of the screen that allows the user to create new discs. By clicking on one of the buttons we can create a disc and drag it onto the world. Once it is dropped into the world, it begins to interact with other modules both as a physical entity and as a audio unit generator. Discs dropped onto other discs or not within the bounds of the world are discarded.

4.1 Menu

The menu provides the user with the ability to create, modify, and destroy unit generators. The interface was designed in Photoshop and the coordinates of the users click are mapped to the pixels on the image. When a disc is selected on the menu, it appears underneath the cursor with transparency. Until it is placed on a clear place on the map, it interacts with the cursor only and will not collide with other objects.

4.2 Parameter Modification

If the user right clicks on a disc, the menu will display its parameters in the lower control menu. Here the user can drag sliders to change the parameters of the unit generators. To prevent the constant reallocation of buffers, the parameters do not smoothly drag, but only change once the user has removed the click.

5 Physics

The translational motion of the discs is implemented using a simple rectangular integration. This helps to achieve increased accuracy for a only slightly increased amount of computation as compared to the rectangular method integration. There is also support for collision detection for simple round objects.

5.1 Translational motion

SPAGHETTI CAT!!At every time step, the motion of the objects is updated. We first update the position of the object according to the formula $x(t + \Delta t) = x(t) + v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2$. We then ask each object in turn for the force that is currently begin applied to it, this force, divided my the mass of the object is equal to its acceleration, $a(t + \Delta t)$. The velocity is then updated, $v(t + \Delta t) = v(t) + \frac{1}{2}(a(t) + a(t + \Delta t))\Delta t$. This is enough for smooth translational motion provided our time steps are chosen correctly.

5.2 Collision Detection

Whenever the discs come close to the wall, or close to another disc, we must prevent them by intersecting by reversing their directions along some axis. Because they are discs, we know that the point of collision will always be at a point along a line drawn between the objects' centers. We project the velocity vectors of each disc on to that axis and multiply that component by -1. To ensure that linear momentum is conserved, we take the momentum of each disc into account and scale these components based the initial velocities and the masses of the discs. To prevent objects from getting stuck inside of each other, we only allow movement along the component through the centers that increases the distance between the centers. This of course, is only applicable when the discs have intersected.
SPAGHETTI CAT!!fix the "quotes" with bold or something to signify that it is code SPAGHETTI CAT!!
[1] http://www.music.mcgill.ca/ gary/rtaudio/ [2] Jon Dattorro - Part 2: Delay-Line Modulation and Chorus https://ccrma.stanford.edu/ da
torro/EffectDesignPart2.pdf [3]https://ccrma.stanford.edu/ jos/pasp/Freeverb.html