Spring 2017

KodySynth

Ryan Kody
rsk35@zips.uakron.edu

Please take a moment to share how this work helps you through this survey. Your feedback will be important as we plan further development of our repository. Follow this and additional works at: http://ideaexchange.uakron.edu/honors_research_projects

Part of the Other Computer Sciences Commons

Recommended Citation

This Honors Research Project is brought to you for free and open access by The Dr. Gary B. and Pamela S. Williams Honors College at IdeaExchange@UAkron, the institutional repository of The University of Akron in Akron, Ohio, USA. It has been accepted for inclusion in Honors Research Projects by an authorized administrator of IdeaExchange@UAkron. For more information, please contact mjon@uakron.edu, uapress@uakron.edu.
KodySynth

Ryan Kody

4/27/2018
Abstract:

KodySynth is a VST digital synthesizer that can be loaded into any digital audio workstation such as FL Studio, Ableton, or Garageband. The program was made using the JUCE library which helped with streamlining the process and making it easier to create clean and intuitive looking graphics. The project also needs the Maximilian Library of sounds to create the sound waves that listeners will hear when playing the synthesizer. Users will also be able to plug in a MIDI keyboard to play KodySynth. The functionalities include a choice of 5 oscillator wave types, as well as an attack, decay, sustain, and release envelope. A filter is also a section that includes a choice of two types as well as cutoff and resonance knobs.

Introduction:

When first given the task that is my Honors project, I had no idea what I wanted to do. There are so many possibilities that I could choose based on the knowledge that the classes at the University of Akron have taught me. The realm of Computer Science is a broad topic and I had to choose from it. I decided to make sure that what I did was fun and entertaining to me and to whoever looked at it. This led me to consider my interest in music and through that my interest in electronic music. I knew that there were programmers that wrote what I use to make music in my own time, and I wanted to see how to start working on synths and other digital audio software.

A program that is used to make music on the computer is called FL Studio. This program is one of the many programs that musicians use to make music and they are called Digital Audio Workstations (DAW). In the program it is possible to load different digital synthesizers and
effects that can be programed to play notes based on data called Musical Instrument Digital Interface, which is more commonly known as MIDI. Some of the digital synthesizers that are used to play MIDI are called VSTs (Virtual Studio Technology) which is actually the form of program they are. There are other forms of these synths that are called AAX and others but most of them are in the form of a VST. Because I wanted my project to be entertaining and interesting to me I decided to make my own VST synthesizer.

Utilities:

**JUCE:**

For this project, 3 tools/resources were used to complete it. One of them was a library called the JUCE library. From Wikipedia:

“JUCE is a partially open-source cross-platform C++ application framework, used for the development of desktop and mobile applications. JUCE is used in particular for its GUI and plug-ins libraries. The general goal of JUCE is to allow software to be written such that the same source code will compile and run identically on Windows, Mac OS X and Linux platforms. It supports various development environments and compilers.”

JUCE was a major part in the completion of this project. A couple of great features it has is the fact that one can take a JUCE project and load it into any computer that has JUCE on it and compile it using their IDE or work environment. It works on Mac, Windows using XCode and Visual Studio. This is a great feature because if there was a group working on the same project, they could use their own computers seamlessly
with no issues with respect to the operating system. Going along with JUCE, there is a resource that uses the JUCE library to run the code. In other words, JUCE is just the library, while this program, the Projucer, is what is able to run cross-platform code from JUCE. From Wikipedia:

“The "Projucer" is an IDE tool for creating and managing JUCE projects. When the files and settings for a JUCE project have been specified, the Projucer automatically generates a collection of 3rd-party project files to allow the project to be compiled natively on each target platform. It can currently generate Xcode projects, Visual Studio projects, Linux Makefiles, Android Ant builds and CodeBlocks projects. As well as providing a way to manage a project's files and settings, it also has a code editor, an integrated GUI editor, wizards for creating new projects and files, and a live coding engine useful for user interface design.”
The Projucer is what this honors project was made in. Visual Studio 2017 was always opened when compiling and building was needed. Using the dropdown at the top of the Projucer as you can see in Figure 1, it will let you choose what to open the project in, whether that be Xcode, Visual Studio, etc. This was a great tool to use and it made it really easy to make sure that what was made would work inside a DAW. The Projucer automatically makes the files that are necessary for a VST file. There is a framework that is included to create the .dll extension to open up in a DAW.

Maximilian:

I used an audio library called Maximilian to create this project as well. From their Website they say:

“Maximilian is an open source, MIT licensed C++ audio synthesis library. It’s designed to be cross platform and simple to use. The syntax and program structure are based on the popular ‘Processing’ environment. Maximilian provides standard waveforms, envelopes, sample playback, resonant filters, and delay lines. In addition, equal power stereo, quadraphonic and 8-channel ambisonic support is included. There’s also Granular synthesisers with Timestretching, FFTs and some Music Information Retrieval stuff.”

I used this library to generate the waveforms that my synth actually played. For example, instead my plug-in creating a sine wave from a formula, it uses the formula that Maximilian already has and plays that. This made sure I did not have to use math to generate the wave to play.
The sine wave function from Maximilian can be seen in figure 2. This function takes in a frequency and uses the sin function to generate the phase at a particular point in time and uses the frequency to refresh at a certain speed to make the wave a particular note. I used this function and many other functions to generate what KodySynth actually plays when it is given MIDI data.

Project:

When starting out with JUCE, there are many options to choose from when creating a project. It is important to note that JUCE is a platform that generates the actual VST program that can be loaded into a DAW, usually in the form of a .dll file. No code was written to generate these files, just the code that used JUCE and Maximilian to generate audio from MIDI Data. The first thing that I had to work on was a program that actually read in MIDI data. This was the first step so that I could not only load it is a DAW, but also plug in a MIDI Keyboard into my computer to play on that as well.
I did this with the startNote and stopNote functions. The function was already written and needs to take in 4 parameters. For the project I only used 2 of them because they are not required as they are only there to add more functionality. For example the last parameter “currentPitchWheelPosition” takes the position of hardware pitch wheel on a keyboard. This will change the frequency of the note. It is nice to have the option for it right in the function that is used. At first I only had the line where frequency gets the midi note. The functions used in that line of code (line 88 in figure 3) were used to translate a midi note number from the hardware keyboard, translate that to hertz, and send that value to frequency. To test that it was actually reading data in the first place I made it print out the midi number each time I pressed a key. The stopNote function was just to make sure that the data would stop being sent from the keyboard once the key was lifted. The function will stop playing a note unless the envelope has a tail off. This tail off comes from the release value from the envelope which will be discussed later.
After the initial set up I also created an envelope class. This was to further add to the functionality of what the audio would sound like in the final form. The envelope edited the attack, decay, sustain, and release of the note generated. This envelope was strictly over the amplitude of the wave generated. Attack is the amount of time that it takes the soundwave to get up to the amplitude indicated. For example, if I pressed a keyboard key with a velocity of 1 and the attack was .5, it would take .5 seconds for the amplitude of the wave to get to 1. That is just an example though because in actual coding the numbers are a bit different. Attack makes the synth sound like a pad, which are very widely used in all kinds of music. Sustain and decay work hand in hand with each other. Sustain is the amplitude at which one wants the sound to be at when it is held for an extended period of time. Sustain is how long it takes for the amplitude to reach the sustain value once a note is being held. Release then, is the amount of time it takes for the amplitude to reach 0 once the key is released. So after all is said and done, there is a multitude of things that can be done with these parameters to make the sound unique and recognizable.
**Figure 5: Visual Representation of ADSR from** [https://blog.landr.com/adsr-envelopes-infographic/](https://blog.landr.com/adsr-envelopes-infographic/)
Figure 6: Code for the Envelope

As you can see in figure 6, a lot of the code is strictly for the graphics or the range of the sliders. I learned all about this from the JUCE documentation online. I did a lot of reading to figure out what functions were already written to do what I needed. The function to translate midi to a hertz is one of those functions that really helped and improved my code which was talked about earlier. In summary, figure 6 shows the creation of the ADSR sliders, and then lines 48-51 link the value from the slider to the ADSR values that are sent around throughout the code.
The envelope has to have a sound wave to change the amplitude of. This comes from the Oscillator section in KodySynth. I gave the user an option of 5 different wavetypes to choose from. They are chosen from the drop down seen in figure 7. The choices are sine, triangle, square, saw, and noise. These are just different types of waves that have a different sound to them. These sound waves came from the Maximilian library. The Maximilian library has many more options for sound waves but I decided to just use some of the most common wave forms. In my code I made a maxiOsc object which I called osc1. This object was given a different waveform based on what the user selects in the oscillator menu.

```c
double setOscType()
{
    if (wave == 0)
        return osc1.sinewave(frequency);
    if (wave == 1)
        return osc1.triangle(frequency);
    if (wave == 2)
        return osc1.square(frequency);
    if (wave == 3)
        return osc1.saw(frequency);
    if (wave == 4)
        return osc1.noise();
    else
        return osc1.sinewave(frequency);
}
```

After setting the waveform to the oscillator object, the function in figure 8 returns the wave form at a specific frequency. This is done so that I can call the function and return the exact formula the wave form actually is. The final section of KodySynth is the filter section.
A filter boosts or limits certain frequencies of a sound. KodySynth has 2 types of filters that a user can choose from; they are a high pass and a low pass. A low pass cuts off high frequencies starting at the cutoff point, and a high pass filter cuts off low frequencies starting at the cutoff point. In the low pass filter, that higher the cutoff is the more frequencies you hear. For example if the cutoff was at 500 hertz, the listener would hear the frequencies between 0 and 500. If the filter was a high pass it would be the other end of the frequency spectrum, where the listener would hear 500 to whatever the highest frequency in the sound was. This functionality is also paired with a resonance knob. In figure 9, there are 2 knobs underneath the filter selection. The left is cutoff and the right is resonance. Resonance boosts the amplitude of the frequencies that the cutoff is currently at. The amount of boost is determined by how much of the resonance knob is turned. This creates a strange noise in some sounds. It will also make the sound generated more unique.
Finally, this is where all the magic happens in KodySynth. The processBlock function is the one that makes the sound that the user hears. Everything is done with a variable named myVoice that is an object of the synthesizer class in the JUCE library. This object gets the parameters from the envelope, than get the type of wave from the getOscType function. After that the sound gets filtered by the filter parameters. The final result is a sound that comes from all of these parameters combined. This is called over and over again by the sample rate which is determined in the DAW itself. I have not found any issues with changing the sample rate in FL Studio.

Future Work:

I would like to do so much more with this plug-in, and I think that I have the ability to do it. I want to add more oscillators and envelopes, but I also want to add more effects. These things should also come with a new UI to make it look intuitive as well as sleek. The oscillators
should not be limited to a list of waveforms to choose from. There should be a UI of the waveform where a user could load in a wave or draw their own. This would make it so that the user could have an infinite number of possibilities of wave type. I would also like to add in a functionality where a user could type a math equation and the subsequent wave would be generated on screen. This would be a really interesting way a user could make music. I would be extremely curious how someone would be creative and make a song with a math theme. It may not sound very good to the ear but I am sure people could appreciate it for what it is.

![Figure 11: Screenshot of Xfer's Serum filter section](image)

In addition, the effects section should be wildly expanded. All that KodySynth has at this time is a filter section with two options of filters, a low pass and a high pass filter and a cutoff with resonance knob. Similarly to the oscillator section, the filter section should have a better UI and be able to draw in filters based on a picture. The filter should have an infinite number of possibilities to be creative with. There is a whole range of harmonics that a filter could modulate on any given sound and being able to have complete control of them makes for some very unique sounds. As you can see in figure 11, the wave you seen is actually a filter shape which can be modulation from the knobs below. KodySynth should have something similar if not even more. There are many different types of filters that a user should be able to choose
from, as well as sliders and knobs for changing that filter. This should be used as a supplement to the fact that users should be able to draw their own filters with the same sliders and knobs for modulation.

KodySynth should have many more options for effects. There are so many different kinds of effects that one could load into a DAW and use to effect the soundwaves that a plugin generates, but I want my plugin to have an effect section already in the plugin itself. This can include reverb, delay, phasors, flanger, chorus, and compression. This would make my plugin very unique and hopefully set it apart from others enough that it could be used professionally.

The last thing that I should work on is the removal of the Maximilian Library. There is no need to be limited to just what the library has when KodySynth would have infinite possibilities in the oscillator section. A user will be using a visual wave instead of a math generated wave. This is why the Maximilian library will be obsolete.

Conclusion/Takeaways:

In conclusion this project was about learning to create a VST plugin through the JUCE library and also using a third party sound library through Maximilian. The goal was met in the fact that I did everything that I set out to do. There is still so much more that I could do with this type of project, and I plan on continuing to work on it in my spare time. Someday I might even make something that is very worthwhile to sell or give to music makers. I got a lot of enjoyment out of making KodySynth. I am looking forward to what it could be in the future.
Works Cited:

*JUCE*, juce.com/learn/documentation.


Maximilian, maximilian.strangeloop.co.uk/.

Picture: [https://blog.landr.com/adsr-envelopes-infographic/](https://blog.landr.com/adsr-envelopes-infographic/)