PS4 Synthesizing a Plucked String Sound

The Karplus-Strong algorithm was used to simulate a plucked guitar string sound in this project. PS4a and PS4b were the two portions of the assignment. In PS4a, we were supposed to create the CircularBuffer class, which would be utilized to keep a ring buffer feedback mechanism running. PS4b would then implement the StringSound class, which would create a CircularBuffer with N samples depending on a 44,100Hz sample rate and a set frequency. The primary function's key pushes would be paired with these frequencies, which would be broadcast via SFML audio.

Key concepts

Due to the fact that StringSound uses CircularBuffer as a dynamic array, each class has to deal with dynamic memory allocation and copy/move/destructor operations. Exceptions were utilized for various functions in both classes. The StringSound(frequency) constructor, for example, includes one exception that precludes it from being used with a frequency of 0. When dynamically allocating memory for the buffer during construction and utilizing the CircularBuffer routines in its own functions, StringSound handles errors given by CircularBuffer.

What I Accomplished

PS4a: CircularBuffer simulates a ring buffer feedback mechanism by storing N vibration samples in an array of type int16_t. To operate with the Karplus-Strong algorithm, the class implements the array as a queue, containing operations like enqueue and dequeue. Exception handling is included in the CircularBuffer constructor and queue routines, which I tested using Boost libraries in test.cpp.

PS4b: StringSound builds a CircularBuffer of length equal to sample rate / frequency from a frequency. To mimic noise, the function pluck() fills the buffer of a string sound with random values in the Int16 range. The sound's buffer is advanced to the next stage by the function tic(), and the sample() function returns the first sample from the buffer. The main method first constructs a vector of Int16 vectors containing the samples of each string sound, which is then loaded into a vector of sf::SoundBuffers, which is then loaded into a vector of sf::Sounds. A sound will be played inside the SFML display loop when the relevant key is hit.

What I Learned

I gained a lot of knowledge about the Karplus-Strong algorithm for streaming digital music and how to interact with audio in SFML. Speaking of which, SFML's Keyboard library is quite useful for handling a piano, guitar, or other instrument – I was thinking of how it might be used in basic 2D games, which, when paired with the PS2's planet things, could produce a very good space invaders-style game.