Ps4: Karplus-Strong String Simulation

Assignment Description:

For this assignment we were tasked to simulate the plucking of a guitar string using the Karplus-Strong algorithm. This algorithm combined with a queue called a Ringbuffer creates the sound of a string reverberating. We were tasked to take the keyboard input and produce a different note of a guitar being played. However I was unable to make the program produce different notes.

Key Concepts and Algorithms:

The key algorithm for this assignment was the Karplus-Strong Algorithm which calculates the energy decay from the string being plucked. The algorithm is:

0.996 * 1/2(a + b) where a and b are the next 2 frequencies in the Ring Buffer.

The program uses a class called StringSound that contains a shared pointer to a Ring Buffer. My understanding of shared pointers from the previous assignment made this assignment easier to accomplish. The class is inherited from sf::Sound this allows for it to be played with .play().

What I learned in this assignment:

While making this assignment I learned how c++ modules and sf::Sound works based on reading the Documentation for the class. This further understanding on how to use different libraries in c++ is something I didn't truly understand until this assignment. The ability to create music notes from a keyboard input could become useful in future projects within c++ and I look forward to creating them in the future.

Ps4: Screen shot

Ps4a Source Code: Makefile

```
CFLAGS = -Wall -Werror -std=c++11 -pedantic
DEPS = -lboost_unit_test_framework


ps4a: test.o RingBuffer.o
	g++ test.cpp RingBuffer.cpp headers/RingBuffer.h -o ps4a $(CFLAGS) $(DEPS)

test.o:
	g++ test.cpp RingBuffer.cpp headers/RingBuffer.h -o test.o $(CFLAGS) $(DEPS)

RingBuffer.o:
	g++ -c RingBuffer.cpp headers/RingBuffer.h $(CFLAGS)

clean:
	rm ps4a
	rm *.o
	rm headers/*.gch
	rm *.out
```

Ps4b Source Code: Makefile

```
CC= g++
CFLAGS= -Wall -Werror -std=c++11
DEPS= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio

KSGuitarSim:	KSGuitarSim.o StringSound.o RingBuffer.o
	$(CC) KSGuitarSim.o StringSound.o RingBuffer.o -o KSGuitarSim $(DEPS)

main.o: main.cpp StringSound.h
	$(CC) -c main.cpp StringSound.h $(CFLAGS) $(DEPS)

StringSound.o: StringSound.cpp StringSound.h
	$(CC) -c StringSound.cpp StringSound.h $(CFLAGS) $(DEPS)

RingBuffer.o: RingBuffer.cpp RingBuffer.h
	$(CC) -c RingBuffer.cpp RingBuffer.h $(CFLAGS) $(DEPS)

clean:
	rm *.o
	rm *.gch
	rm KSGuitarSim
```

Ps4a Source Code: test.cpp

```cpp
// Copyright 2020 John Simonson

#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/unit_test.hpp>
#include"headers/RingBuffer.h"


// construtor tests
BOOST_AUTO_TEST_CASE(constructor) {
BOOST_REQUIRE_THROW(RingBuffer(0), std::invalid_argument);
BOOST_REQUIRE_THROW(RingBuffer(-1), std::invalid_argument);
BOOST_REQUIRE_NO_THROW(RingBuffer(21));
}

// size test
BOOST_AUTO_TEST_CASE(size) {
RingBuffer temp(21);
BOOST_REQUIRE(temp.size() == 0);
}

// isEmpty test
BOOST_AUTO_TEST_CASE(isEmpty) {
RingBuffer temp(21);
BOOST_REQUIRE(temp.isEmpty() == true);
}

// isFull test
BOOST_AUTO_TEST_CASE(isFull) {
RingBuffer temp(21);
BOOST_REQUIRE(temp.isFull() == false);
}

// enqueue test
BOOST_AUTO_TEST_CASE(enqueue) {
RingBuffer temp(1);
BOOST_REQUIRE(temp.size() == 0);
temp.enqueue(50);
BOOST_REQUIRE(temp.size() == 1);
BOOST_REQUIRE_THROW(temp.enqueue(21), std::runtime_error);
}

// dequeue test
```

```
BOOST_AUTO_TEST_CASE(dequeue) {
RingBuffer temp(21);
BOOST_REQUIRE_THROW(temp.dequeue(), std::runtime_error);
}


// peek test
BOOST_AUTO_TEST_CASE(peek) {
RingBuffer temp(21);
BOOST_REQUIRE_THROW(temp.peek(), std::runtime_error);
}
```

Ps4b Source Code: KSGuitarSim.cpp

```
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Audio.hpp>
#include <SFML/Window.hpp>

#include <math.h>
#include <limits.h>

#include <iostream>
#include <string>
#include <exception>
#include <stdexcept>
#include <vector>

#include "RingBuffer.h"
#include "StringSound.h"

#define CONCERT_A 440.0
#define SAMPLES_PER_SEC 44100
const int num_of_keys = 37;
std::string keyboard = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' ";
std::vector<sf::Int16> makeSamples(StringSound gs) {
std::vector<sf::Int16> samples;

  gs.pluck();
  int duration = 8;  // seconds
  int i;
  for (i= 0; i < SAMPLES_PER_SEC * duration; i++) {
    gs.tic();
    samples.push_back(gs.sample());
  }

  return samples;
}

int main() {
  sf::RenderWindow window(sf::VideoMode(300, 200), "ps4b");
  sf::Event event;
  double freq;
  std:: vector<sf::Int16> samples;
  std::vector<std::vector<sf::Int16>> vector_of_samples(num_of_keys);
  std::vector<sf::SoundBuffer> sound_buffer(num_of_keys);
  std::vector<sf::Sound> sounds(num_of_keys);
  for(int i = 0; i < num_of_keys; i++){
  freq = CONCERT_A * pow(2, 3.0/12.0);
  StringSound gs2 = StringSound(freq);
  samples = makeSamples(gs2);
  vector_of_samples[i] = samples;

 sound_buffer[i].loadFromSamples(&vector_of_samples[i][0], vector_of_samples[i].

 sounds[i].setBuffer(sound_buffer[i]);
}

  while (window.isOpen()) {
    while (window.pollEvent(event)) {
    if (event.type == sf::Event::Closed)
                window.close();
    if(event.type == sf::Event::TextEntered){
    char ascii = (char)(event.text.unicode);
    for(int i = 0; i < num_of_keys; i++){
    if(keyboard[i] == ascii)
        sounds[i].play();
}
      window.clear();
      window.display();
    }
  }
}
  return 0;
}
```

Ps4a Source Code: RingBuffer.hpp

```cpp
// Copyright 2020 John Simonson
#ifndef RINGBUFFER_H_
#define RINGBUFFER_H_
#include<stdint.h>
#include<iostream>
#include<queue>

class RingBuffer{
 public:
RingBuffer(int capacity);
// create an empty ring buffer, with given max capacity
const double ENERGY_DECAY_FACTOR = 0.996;
int     size();  // return number of items currently in the buffer
bool    isEmpty();  // is the buffer empty (size equals zero)?
bool    isFull();  // is the buffer full  (size equals capacity)?
void    enqueue(int16_t x);  // add item x to the end
int16_t dequeue();  // delete and return item from the front
int16_t peek();  // return (but do not delete) item from the front
 private:
int front;
int back;
int Capacity;
int Size;
std::queue<int16_t> ringBuffer;
};
#endif
```

Ps4b Source Code: StringSound.hpp

```cpp
#ifndef STRINGSOUND_H
#define STRINGSOUND_H
#include <SFML/Audio.hpp>
#include <string>
#include "RingBuffer.h"
#include <memory>
#include <stdint.h>
#include<vector>

class StringSound {
 public:
    StringSound(double frequency);      // create a guitar string sound of the
                        // given frequency using a sampling rate
                        // of 44,100
    StringSound(std::vector<sf::Int16> init);       // create a guitar string wi
                        // size and initial values are given by
                        // the vector
    StringSound(RingBuffer init);
    void pluck();                   // pluck the guitar string by replacing
                        // the buffer with random values,
                        // representing white noise
    void tic();                     // advance the simulation one time step
    sf::Int16 sample();                 // return the current sample
    int time();                     // return number of times tic was called
                        // so far
    void generate();
 private:
    std::shared_ptr<RingBuffer> ring_buffer;
    int tics;
};

#endif
```

Ps4a Source Code: RingBuffer.cpp

```
// Copyright 2020 John Simonson
#include "RingBuffer.h"
using std::queue;
using std::runtime_error;
using std::invalid_argument;


RingBuffer::RingBuffer(int capacity) {
if (capacity < 1) {
     throw
     invalid_argument("Capacity must be 1 or greater");
}
     front = 0;
     back = 0;
     Capacity = capacity;
     Size = 0;
}

int RingBuffer::size() {
     return ringBuffer.size();
}

bool RingBuffer::isEmpty() {
     return ringBuffer.empty();
}

bool RingBuffer::isFull() {
     if (Size == Capacity) {
     return true;
}
     return false;
}

void RingBuffer::enqueue(int16_t x) {
     if (Size == Capacity) {
     throw
     runtime_error("cannot push onto full buffer");
}
     ringBuffer.push(x);
     Size++;
     return;
}

int16_t RingBuffer::dequeue() {
      if (ringBuffer.empty()) {
      throw
      runtime_error("cannot dequeue an empty buffer");
}
      int16_t temp = ringBuffer.front();
      ringBuffer.pop();
      Size--;
      return temp;
}

int16_t RingBuffer::peek() {
      if (ringBuffer.empty()) {
      throw
      runtime_error("cannot peek at an empty buffer");
}
      return (ringBuffer.front());
}
```

Ps4b Source Code: StringSound.cpp

```
#include"StringSound.h"
#include<random>
#define Rand std::uniform_int_distribution;
```

```
#define Gen std::default_random_engine;

const int16_t MIN = 1 << 15;
const int16_t MAX = ~MIN;

StringSound::StringSound(double frequency){
    if(frequency <= 0){
        throw
        std::invalid_argument("frequency must be greater than zero");
}
    ring_buffer = std::shared_ptr<RingBuffer>(new RingBuffer(frequency));
    tics = 0;
}

StringSound::StringSound(RingBuffer rb){
    ring_buffer = std::shared_ptr<RingBuffer>(new RingBuffer(rb));
    tics = 0;
}

StringSound::StringSound(std::vector<sf::Int16> init ){
    ring_buffer = std::shared_ptr<RingBuffer>(new RingBuffer(init.size()));
    tics = 0;
}

void StringSound::pluck(){
    std::default_random_engine rand_gen;
    std::uniform_int_distribution<int16_t> gen = std::uniform_int_distribution<i
    while(!ring_buffer->isFull())
        ring_buffer->enqueue(gen(rand_gen));
}

sf::Int16 StringSound::sample(){
    return ring_buffer->peek();
}

void StringSound::tic(){
    tics++;
    int16_t temp1 = ring_buffer->dequeue();
    int16_t temp2 = ring_buffer->peek();
    temp1 = temp1 + temp2;
    temp1 = temp1 / 2;
    temp1 = temp1 * 0.996;
    ring_buffer->enqueue(temp1);
 }

int StringSound::time(){
 return tics;

}
```