# 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.

```
    1: CC= g++
    2: CFLAGS= -g -Wall -Werror -std=c++0x -pedantic
    3: SFLAGS= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
    4: BOOST= -lboost_unit_test_framework
    5:
    6:
    7: all: KSGuitarSim SStest SSLite
    8:
    9: KSGuitarSim: KSGuitarSim.o StringSound.o CircularBuffer.o
   10:         $(CC) KSGuitarSim.o StringSound.o CircularBuffer.o -o KSGuitarSim
 $(SFLAGS)
   11:
   12: SSLite: SSLite.o StringSound.o CircularBuffer.o
   13:         $(CC) SSLite.o StringSound.o CircularBuffer.o -o SSLite $(SFLAGS)
   14:
   15: SStest: SStest.o StringSound.o CircularBuffer.o
   16:         $(CC) SStest.o StringSound.o CircularBuffer.o -o SStest $(BOOST)
   17:
   18: KSGuitarSim.o: KSGuitarSim.cpp StringSound.hpp
   19:         $(CC) -c KSGuitarSim.cpp StringSound.hpp $(CFLAGS)
   20:
   21: SSLite.o: SSLite.cpp StringSound.hpp
   22:         $(CC) -c SSLite.cpp StringSound.hpp $(CFLAGS)
   23:
   24: StringSound.o: StringSound.cpp StringSound.hpp
   25:         $(CC) -c StringSound.cpp StringSound.hpp $(CFLAGS)
   26:
   27: CircularBuffer.o: CircularBuffer.cpp CircularBuffer.hpp
   28:         $(CC) -c CircularBuffer.cpp CircularBuffer.hpp $(CFLAGS)
   29:
   30: SStest.o: SStest.cpp
   31:         $(CC) -c SStest.cpp $(Boost)
   32:
   33: clean:
   34:         rm *.o
   35:         rm *.gch
   36:         rm KSGuitarSim
   37:         rm SStest
   38:         rm SSLite
   39:
```

```
    1: #include <SFML/Graphics.hpp>
    2: #include <SFML/System.hpp>
    3: #include <SFML/Audio.hpp>
    4: #include <SFML/Window.hpp>
    5:
    6: #include <math.h>
    7: #include <limits.h>
    8:
    9: #include <iostream>
   10: #include <string>
   11: #include <exception>
   12: #include <stdexcept>
   13: #include <vector>
   14:
   15: #include "CircularBuffer.hpp"
   16: #include "StringSound.hpp"
   17:
   18: #define CONCERT_A 440.0
   19: #define SAMPLES_PER_SEC 44100
   20: const int keyboard_size = 37;
   21:
   22: std::vector<sf::Int16> makeSamples(StringSound gs)
   23: {
   24:     std::vector<sf::Int16> samples;
   25:
   26:     gs.pluck();
   27:     int duration = 8; // seconds
   28:     int i;
   29:     for (i = 0; i < SAMPLES_PER_SEC * duration; i++) {
   30:         gs.tic();
   31:         samples.push_back(gs.sample());
   32:     }
   33:
   34:     return samples;
   35: }
   36:
   37: int main()
   38: {
   39:     sf::RenderWindow window(sf::VideoMode(800, 800), "SFML KSGuitarSim");
   40:     sf::Event event;
   41:
   42:     double frequency;
   43:     std::vector<sf::Int16> sample;
   44:
   45:     std::vector<std::vector<sf::Int16>> samples(keyboard_size);
   46:     std::vector<sf::SoundBuffer> buffers(keyboard_size);
   47:     std::vector<sf::Sound> sounds(keyboard_size);
   48:
   49:     std::string keyboard = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' ";
   50:
   51:     for (int i = 0; i < (signed)keyboard.size(); i++) {
   52:         frequency = CONCERT_A * pow(2, ((i - 24) / 12.0));
   53:         StringSound tmp = StringSound(frequency);
   54:
   55:         sample = makeSamples(tmp);
   56:         samples[i] = sample;
   57:
   58:         if (!buffers[i].loadFromSamples(&samples[i][0],
   59:                                         samples[i].size(), 2, SAMPLES_PER
_SEC)) {
   60:             throw std::runtime_error("sf::SoundBuffer: failed to load fro
m samples.");
   61:         }
   62:
   63:         sounds[i].setBuffer(buffers[i]);
```

```
    64:     }
    65:
    66:     while (window.isOpen()) {
    67:         while (window.pollEvent(event)) {
    68:             if (event.type == sf::Event::TextEntered) {
    69:                 if (event.text.unicode < 128) {
    70:                     char key = static_cast<char>(event.text.unicode);
    71:
    72:                     for (int i = 0; i < (signed)keyboard.size(); i++) {
    73:                         if (keyboard[i] == key) {
    74:                             std::cout << "Keyboard key is: " << keyboard[
i] << "\n";
    75:                             std::cout << "Attempting to play sound...\n";
    76:                             sounds[i].play();
    77:                             break;
    78:                         }
    79:                     }
    80:                 }
    81:             }
    82:
    83:             if (event.type == sf::Event::Closed) {
    84:                 window.close();
    85:             }
    86:         }
    87:
    88:         window.clear();
    89:         window.display();
    90:     }
    91:     return 0;
    92: }
```

```
    1: // Copyright 2022 Matthew Lorette Anaya, matthew_loretteanaya@student.uml
.edu
    2:
    3: #ifndef _USERS_MATTHEWLORETTEANAYA_DOCUMENTS_UML_COMP_IV_PS4B_CIRCULARBUF
FER_HPP_
    4: #define _USERS_MATTHEWLORETTEANAYA_DOCUMENTS_UML_COMP_IV_PS4B_CIRCULARBUF
FER_HPP_
    5:
    6: #include <stdint.h>
    7: #include <iostream>
    8: #include <string>
    9: #include <sstream>
   10: #include <exception>
   11: #include <stdexcept>
   12: #include <vector>
   13:
   14: class CircularBuffer {
   15:  public:
   16: // create an empty circular buffer, with given max capacity
   17:   explicit CircularBuffer(int capacity);
   18:   int size();
   19:   bool isEmpty();
   20:   bool isFull();
   21:   void enqueue(int16_t x);
   22:   int16_t dequeue();
   23:   int16_t peek();
   24:   void output();
   25:
   26:  private:
   27:   std::vector<int16_t> buff;
   28:   int first;
   29:   int last;
   30:   int cap;
   31:   int s;
   32: };
   33:
   34: #endif  // _USERS_MATTHEWLORETTEANAYA_DOCUMENTS_UML_COMP_IV_PS4B_CIRCULAR
BUFFER_HPP_
```

```
    1: /*
    2:   Copyright 2022 Matthew Lorette Anaya, matthew_loretteanaya@student.uml.
edu
    3: */
    4:
    5: #include "CircularBuffer.hpp"
    6:
    7: CircularBuffer::CircularBuffer(int capacity) {
    8:     if (capacity < 1) {
    9:         throw std::invalid_argument
   10:          ("Circular Buffer constructor: capacity must be greater than zer
o");
   11:     }
   12:
   13:     last = 0;
   14:     first = 0;
   15:     s = 0;
   16:     cap = capacity;
   17:     buff.resize(capacity);
   18:
   19:     return;
   20: }
   21:
   22: int CircularBuffer::size() {
   23:     return s;
   24: }
   25:
   26: bool CircularBuffer::isEmpty() {
   27:     if (s != 0) {
   28:         return false;
   29:     } else {
   30:       return true;
   31:     }
   32: }
   33:
   34: bool CircularBuffer::isFull() {
   35:     if (s == cap) {
   36:         return true;
   37:     } else {
   38:       return false;
   39:     }
   40: }
   41: void CircularBuffer::enqueue(int16_t x) {
   42:     if (isFull()) {
   43:         throw std::runtime_error("enqueue: can't enqueue to a full buffer
");
   44:     }
   45:     if (last >= cap) {
   46:         last = 0;
   47:     }
   48:     // Continue
   49:     buff.at(last) = x;
   50:     last++;
   51:     s++;
   52: }
   53:
   54: int16_t CircularBuffer::dequeue() {
   55:     if (isEmpty()) {
   56:         throw std::runtime_error("dequeue: can't dequeue an empty buffer"
);
   57:     }
   58:     int16_t retFirst = buff.at(first);
   59:     buff.at(first) = 0;
   60:     first++;
   61:     s--;
```

```
   62:
   63:     if (first >= cap) {
   64:         first = 0;
   65:     }
   66:
   67:     return retFirst;
   68: }
   69:
   70: int16_t CircularBuffer::peek() {
   71:     if (isEmpty()) {
   72:         throw std::runtime_error("peek: can't peek an empty buffer");
   73:     }
   74:     return buff.at(first);
   75: }
   76:
   77: void CircularBuffer::output() {
   78:     std::cout << "   First: " << first << "\n";
   79:     std::cout << "    Last: " << last << "\n";
   80:     std::cout << "Capacity: " << cap << "\n";
   81:     std::cout << "    Size: " << s << "\n";
   82:     std::cout << "Vector size: " << buff.size() << "\n";
   83:     std::cout << "Vector capacity: " << buff.capacity() << "\n";
   84:     std::cout << "Buffer (no blanks): \n";
   85:
   86:     int x = 0;
   87:     int y = first;
   88:
   89:     while (x < s) {
   90:         // Make the loop go back to 0 to continue printing.
   91:         if (y >= cap) {
   92:             y = 0;
   93:         }
   94:
   95:         std::cout << buff[y] << " ";
   96:         y++;
   97:         x++;
   98:     }
   99:
  100:     std::cout << "\nDump the entire buffer (including blanks): \n";
  101:
  102:     for (int x = 0; x < cap; x++) {
  103:         std::cout << buff[x] << " ";
  104:     }
  105:
  106:     std::cout << "\n\n";
  107: }
```

```
 1: #ifndef STRINGSOUND_HPP
 2: #define STRINGSOUND_HPP
 3:
 4: #include <SFML/Audio.hpp>
 5: #include <SFML/Graphics.hpp>
 6: #include <SFML/System.hpp>
 7: #include <SFML/Window.hpp>
 8: #include <cmath>
 9: #include <iostream>
10: #include <string>
11: #include <vector>
12: #include "CircularBuffer.hpp"
13:
14: const int SAMPLING_RATE = 44100;
15: const double ENERGY_DECAY_FACTOR = 0.996;
16:
17: class StringSound {
18:  public:
19:   explicit StringSound(double frequency);
20:
21:   explicit StringSound(std::vector<sf::Int16> init);
22:
23:   void pluck();
24:
25:   // advance the simulation one time step
26:   void tic();
27:
28:   // return the current sample
29:   sf::Int16 sample();
30:
31:   // return number of times tic was called
32:   int time();
33:
34:  private:
35:   CircularBuffer buff;
36:   int num;
37:   int tictic;
38: };
39: #endif
```

```
 1: #include "StringSound.hpp"
 2: #include <vector>
 3:
 4:
 5: StringSound:: StringSound(double frequency):
 6:   buff(ceil(SAMPLING_RATE / frequency)) {
 7:   num = ceil(SAMPLING_RATE / frequency);
 8:
 9:   for (int i = 0; i < num; i++) {
10:     buff.enqueue((int16_t)0);
11:   }
12:   tictic = 0;
13: }
14:
15:
16: StringSound:: StringSound(std::vector<sf::Int16> init):
17:   buff(init.size()) {
18:   num = init.size();
19:
20:   std::vector<sf::Int16>::iterator it;
21:
22:   for (it = init.begin(); it < init.end(); it++) {
23:     buff.enqueue((int16_t)*it);
24:   }
25:   tictic = 0;
26: }
27:
28: void StringSound::pluck() {
29:   for (int i = 0; i < num; i++) {
30:     buff.dequeue();
31:   }
32:
33:   for (int i = 0; i < num; i++) {
34:     buff.enqueue((sf::Int16)(rand() & 0xffff));
35:   }
36:
37:   return;
38: }
39:
40:
41: void StringSound::tic() {
42:   int16_t first = buff.dequeue();
43:   int16_t second = buff.peek();
44:
45:   int16_t avg = (first + second) / 2;
46:   int16_t karplus = avg * ENERGY_DECAY_FACTOR;
47:
48:   buff.enqueue((sf::Int16)karplus);
49:
50:   tictic++;
51:
52:   return;
53: }
54:
55:
56: // return  current sample
57: sf::Int16 StringSound::sample() {
58:
59:   sf::Int16 sample = (sf::Int16)buff.peek();
60:
61:   return sample;
62: }
63:
64:
65: // number of tics called
```

```
66: int StringSound::time() {
67:   return tictic;
68: }
```

```
    1: /*
    2:   Copyright 2015 Fred Martin,
    3:   Y. Rykalova, 2020
    4: */
    5:
    6: #include <SFML/Graphics.hpp>
    7: #include <SFML/System.hpp>
    8: #include <SFML/Audio.hpp>
    9: #include <SFML/Window.hpp>
   10:
   11: #include <math.h>
   12: #include <limits.h>
   13:
   14: #include <iostream>
   15: #include <string>
   16: #include <exception>
   17: #include <stdexcept>
   18: #include <vector>
   19:
   20: #include "CircularBuffer.hpp"
   21: #include "StringSound.hpp"
   22:
   23: using namespace std;
   24:
   25: #define CONCERT_A 220.0
   26: #define SAMPLES_PER_SEC 44100
   27:
   28: vector<sf::Int16> makeSamples(StringSound gs) {
   29:   std::vector<sf::Int16> samples;
   30:
   31:   gs.pluck();
   32:   int duration = 8;  // seconds
   33:   int i;
   34:   for (i= 0; i < SAMPLES_PER_SEC * duration; i++) {
   35:     gs.tic();
   36:     samples.push_back(gs.sample());
   37:   }
   38:
   39:   return samples;
   40: }
   41:
   42: int main() {
   43:   sf::RenderWindow window(sf::VideoMode(300, 200), "SFML Plucked String S
ound Lite");
   44:   sf::Event event;
   45:   double freq;
   46:   vector<sf::Int16> samples;
   47:   freq = CONCERT_A;
   48:   StringSound gs1 = StringSound(freq);
   49:   sf::Sound sound1;
   50:   sf::SoundBuffer buf1;
   51:   samples = makeSamples(gs1);
   52:   if (!buf1.loadFromSamples(&samples[0], samples.size(), 2, SAMPLES_PER_S
EC))
   53:       throw std::runtime_error("sf::SoundBuffer: failed to load from samp
les.");
   54:   sound1.setBuffer(buf1);
   55:
   56:   freq = CONCERT_A * pow(2, 3.0/12.0);
   57:   StringSound gs2 = StringSound(freq);
   58:   sf::Sound sound2;
   59:   sf::SoundBuffer buf2;
   60:   samples = makeSamples(gs2);
   61:   if (!buf2.loadFromSamples(&samples[0], samples.size(), 2, SAMPLES_PER_S
EC))
```

```
   62:       throw std::runtime_error("sf::SoundBuffer: failed to load from samp
les.");
   63:   sound2.setBuffer(buf2);
   64:
   65:   while (window.isOpen()) {
   66:     while (window.pollEvent(event)) {
   67:       switch (event.type) {
   68:       case sf::Event::Closed:
   69:         window.close();
   70:         break;
   71:
   72:       case sf::Event::KeyPressed:
   73:         switch (event.key.code) {
   74:         case sf::Keyboard::A:
   75:           sound1.play();
   76:           break;
   77:         case sf::Keyboard::C:
   78:           sound2.play();
   79:           break;
   80:         default:
   81:           break;
   82:         }
   83:
   84:       default:
   85:         break;
   86:       }
   87:
   88:       window.clear();
   89:       window.display();
   90:     }
   91:   }
   92:   return 0;
   93: }
   94:
```

```
 1: #define BOOST_TEST_DYN_LINK
 2: #define BOOST_TEST_MODULE Main
 3: #include <boost/test/unit_test.hpp>
 4:
 5: #include <vector>
 6: #include <exception>
 7: #include <stdexcept>
 8:
 9: #include "StringSound.hpp"
10:
11: BOOST_AUTO_TEST_CASE(GS)
12: {
13:   std::vector<sf::Int16> v;
14:   v.push_back(0);
15:   v.push_back(2000);
16:   v.push_back(4000);
17:   v.push_back(-10000);
18:
19:   BOOST_REQUIRE_NO_THROW(StringSound ss = StringSound(v));
20:
21:   StringSound ss = StringSound(v);
22:
23:   // StringSound = 0 2000 4000 -10000
24:   BOOST_REQUIRE(ss.sample() == 0);
25:
26:   // StringSound = 2000 4000 -10000 996
27:   ss.tic();
28:   BOOST_REQUIRE(ss.sample() == 2000);
29:
30:   // StringSound = 4000 -10000 996 2988
31:   ss.tic();
32:   BOOST_REQUIRE(ss.sample() == 4000);
33:
34:   // StringSound = -10000 996 2988 -2988
35:   ss.tic();
36:   BOOST_REQUIRE(ss.sample() == -10000);
37:
38:   // StringSound = 996 2988 -2988 -4483
39:   ss.tic();
40:   BOOST_REQUIRE(ss.sample() == 996);
41:
42:   // StringSound = 2988 -2988 -4483 1984
43:   ss.tic();
44:   BOOST_REQUIRE(ss.sample() == 2988);
45:
46:   // StringSound = -2988 -4483 1984 0
47:   ss.tic();
48:   BOOST_REQUIRE(ss.sample() == -2988);
49:
50:   //more
51:   ss.tic();
52:   BOOST_REQUIRE(ss.sample() == -4483);
53:   ss.tic();
54:   BOOST_REQUIRE(ss.sample() == 1984);
55:   ss.tic();
56:   BOOST_REQUIRE(ss.sample() == 0);
57: }
```

```
 1: #include "CircularBuffer.hpp"
 2:
 3: int main(){
 4:     std::cout << "Test main.\n";
 5:
 6:     CircularBuffer test(100);
 7:     test.enqueue(1);
 8:     test.enqueue(2);
 9:     test.enqueue(3);
10:     std::cout << "Peek: " << test.peek() << "\n";
11:
12:     std::cout << "Deq 1: " << test.dequeue() << "\n";
13:     std::cout << "Deq 2: " << test.dequeue() << "\n";
14:
15:     test.output();
16:
17:     // Test looping back around
18:     CircularBuffer test2(3);
19:
20:     test2.enqueue(1);
21:     test2.enqueue(2);
22:     test2.enqueue(3);
23:
24:     test2.dequeue();
25:     test2.dequeue();
26:     test2.dequeue();
27:
28:     test2.enqueue(1);
29:     test2.enqueue(2);
30:     test2.enqueue(3);
31:     test2.dequeue();
32:     test2.enqueue(4);
33:
34:     test2.output();
35:
36:     return 0;
37: }
```

```
 1: #define BOOST_TEST_DYN_LINK
 2: #define BOOST_TEST_MODULE Main
 3: #include <boost/test/unit_test.hpp>
 4:
 5: #include "CircularBuffer.hpp"
 6:
 7: // Tests various aspects of the contructor.
 8: BOOST_AUTO_TEST_CASE(Constructor)
 9: {
10:         // Shouldn't fail.
11:         BOOST_REQUIRE_NO_THROW(CircularBuffer(100));
12:
13:         // Should fail.
14:         BOOST_REQUIRE_THROW(CircularBuffer(0), std::exception);
15:         BOOST_REQUIRE_THROW(CircularBuffer(0), std::invalid_argument);
16:         BOOST_REQUIRE_THROW(CircularBuffer(-1), std::invalid_argument);
17: }
18:
19: // Checks the size() method
20: BOOST_AUTO_TEST_CASE(Size)
21: {
22:         CircularBuffer test(1);
23:
24:
25:         BOOST_REQUIRE(test.size() == 0);
26:
27:         test.enqueue(5);
28:
29:
30:         BOOST_REQUIRE(test.size() == 1);
31:
32:         test.dequeue();
33:         BOOST_REQUIRE(test.size() == 0);
34: }
35:
36: // Checks the isEmpty() method
37: BOOST_AUTO_TEST_CASE(isEmpty)
38: {
39:         // True
40:         CircularBuffer test(5);
41:         BOOST_REQUIRE(test.isEmpty() == true);
42:
43:         // False
44:         CircularBuffer test2(5);
45:         test2.enqueue(5);
46:         BOOST_REQUIRE(test2.isEmpty() == false);
47: }
48:
49: // Checks the isFull() method
50: BOOST_AUTO_TEST_CASE(isFull)
51: {
52:         CircularBuffer test(5);
53:         BOOST_REQUIRE(test.isFull() == false);
54:
55:         CircularBuffer test2(1);
56:         test2.enqueue(5);
57:         BOOST_REQUIRE(test2.isFull() == true);
58: }
59:
60: // Test enqueue
61: BOOST_AUTO_TEST_CASE(Enqueue)
62: {
63:         // These test basic enqueuing
64:         CircularBuffer test(5);
65:
```

```
66:         BOOST_REQUIRE_NO_THROW(test.enqueue(1));
67:         BOOST_REQUIRE_NO_THROW(test.enqueue(2));
68:         BOOST_REQUIRE_NO_THROW(test.enqueue(3));
69:         BOOST_REQUIRE_NO_THROW(test.enqueue(4));
70:         BOOST_REQUIRE_NO_THROW(test.enqueue(5));
71:         BOOST_REQUIRE_THROW(test.enqueue(6), std::runtime_error);
72: }
73:
74: // Test dequeue
75: BOOST_AUTO_TEST_CASE(Dequeue)
76: {
77:         CircularBuffer test(5);
78:
79:         test.enqueue(0);
80:         test.enqueue(1);
81:         test.enqueue(2);
82:
83:         BOOST_REQUIRE(test.dequeue() == 0);
84:         BOOST_REQUIRE(test.dequeue() == 1);
85:         BOOST_REQUIRE(test.dequeue() == 2);
86:         BOOST_REQUIRE_THROW(test.dequeue(), std::runtime_error);
87: }
```