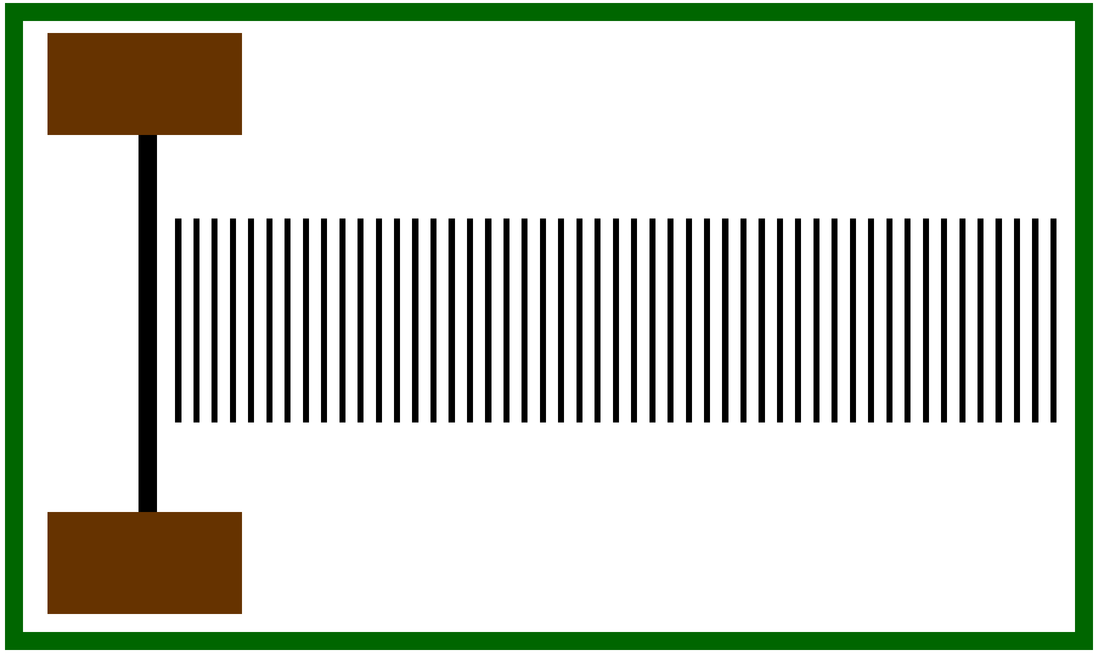
Guitar Hero



Guitar Strings

- A sound wave is produced by a vibrating object
- As a guitar string vibrates, it sets surrounding air molecules into vibrational motion
- The frequency at which these air molecules vibrate is equal to the frequency of vibration of the guitar string

Guitar Strings

- The back and forth vibrations of the surrounding air molecules creates a pressure wave which travels outward from its source.
- This pressure wave consists of compressions and rarefactions
- This alternating pattern of compressions and rarefactions is known as a sound wave

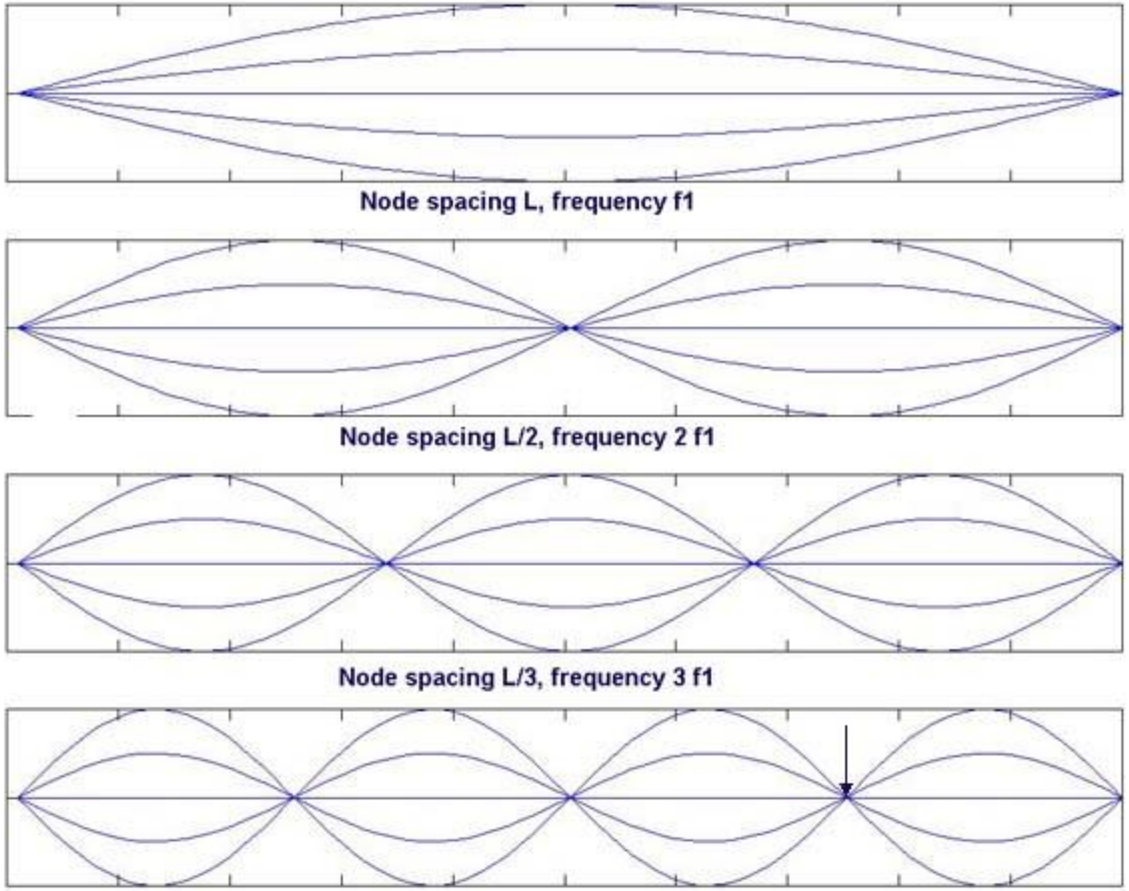
Guitar Strings

- A string fixed at both ends, as on a guitar, can vibrate in a "*standing wave*" mode at several different frequencies
- The lowest frequency, the fundamental, is such that the length of the string L matches half of a wavelength λ

$$L = \lambda/2L = \lambda/2$$

- The middle of the string has the maximum displacement from rest and the two ends don't move.
- In this case, the mid-point of the string is called an antinode and the ends are (always) nodes

Guitar Strings

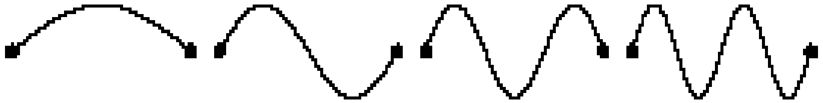


Node spacing L, frequency f1

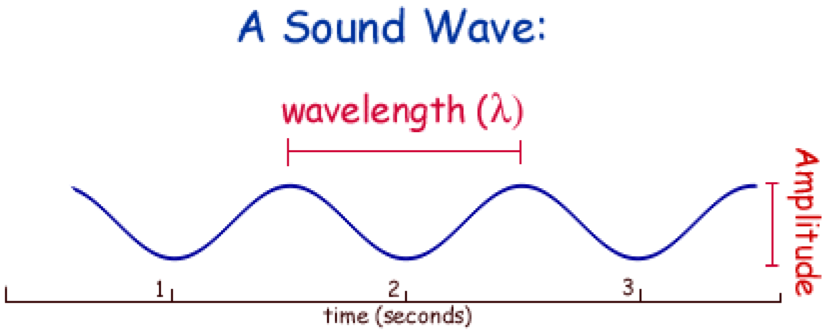
Node spacing L/2, frequency 2 f1

Node spacing L/3, frequency 3 f1

Node spacing L/4, frequency 4 f1

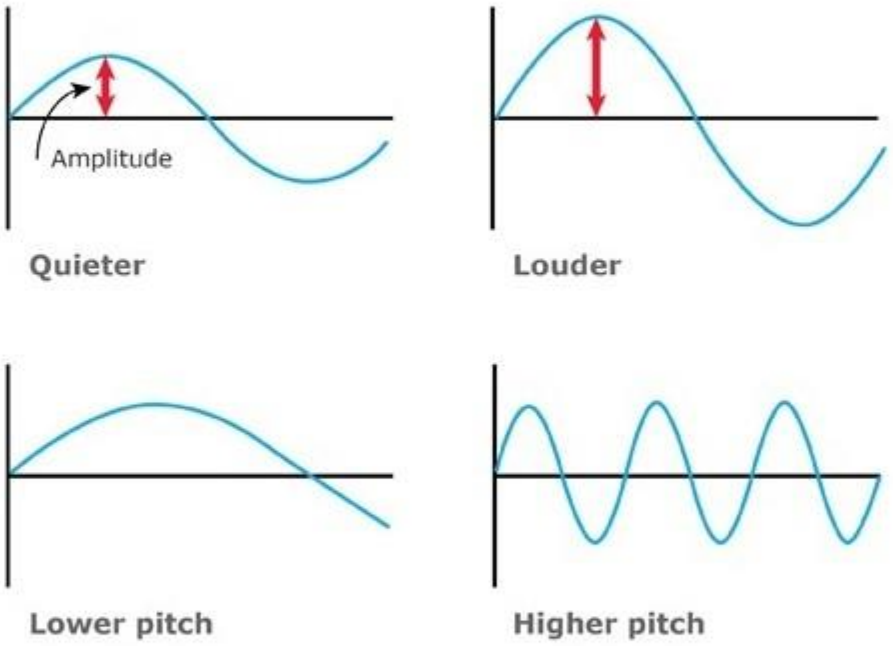


Guitar Strings



period = time of one cycle (wavelength)
above: one second from crest to crest

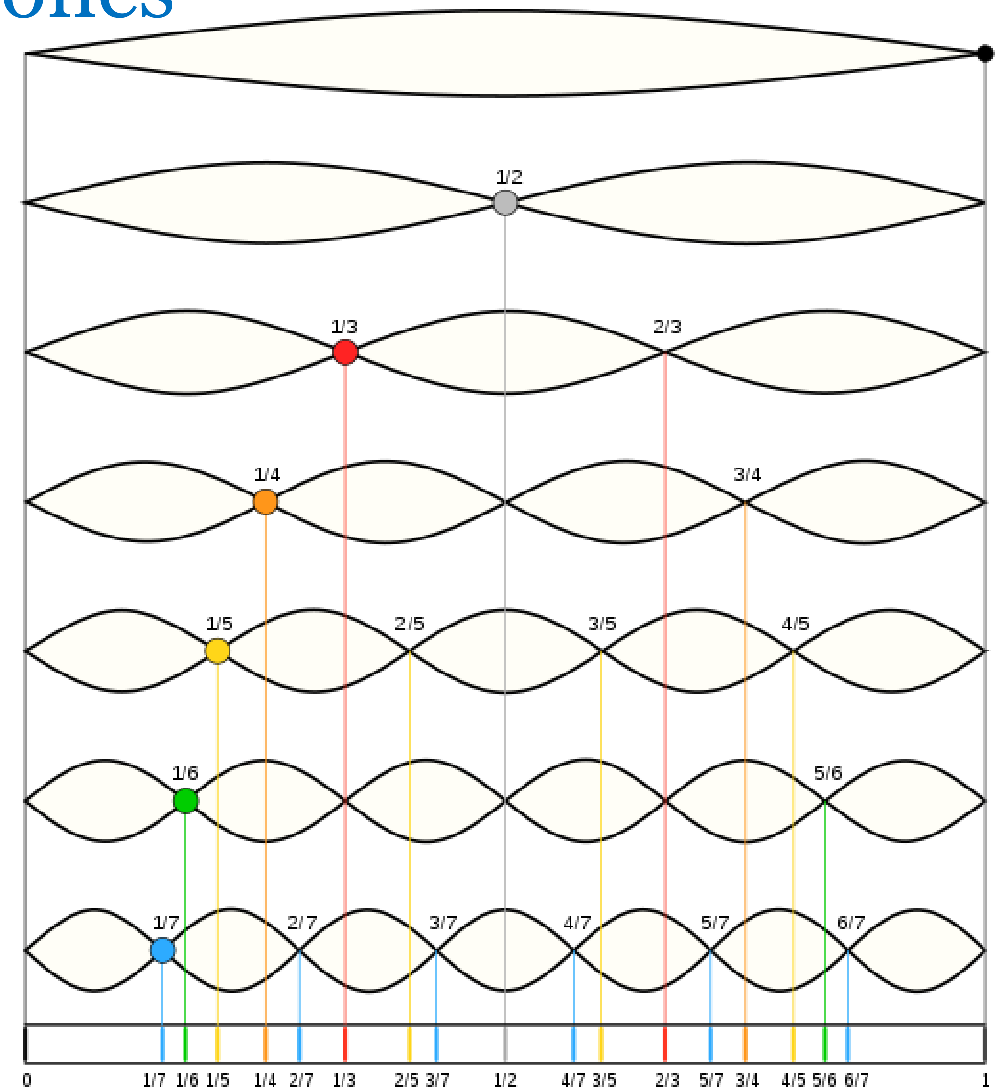
frequency = number of cycles/second
above: 1 cycle/sec = 1 Hertz (Hz)



Range of audible frequencies as 20 to 20,000 cycles per second (cps, also known as hertz, abbreviated Hz = 1 vibration/second)

Guitar Strings

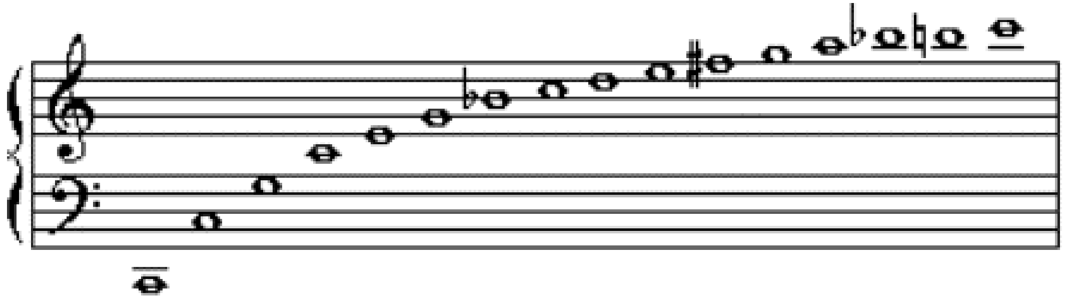
Harmonic tones



Guitar Strings

Harmonic tones

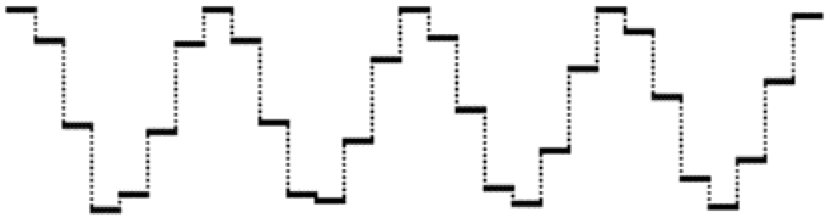
- *Spectrum* - the combination of frequencies and their amplitudes that are present in a sound
- Jean-Baptiste Joseph Fourier: any periodic wave, no matter how complex, is the sum of different harmonically related frequencies
- When the fundamental frequency is multiplied by a power of 2 -- 2, 4, 8, 16, etc. -- the perceived musical pitch increases by one octave



Harmonic partials of a fundamental frequency *f*, where *f* = 65.4 Hz = the pitch low C

Guitar Strings

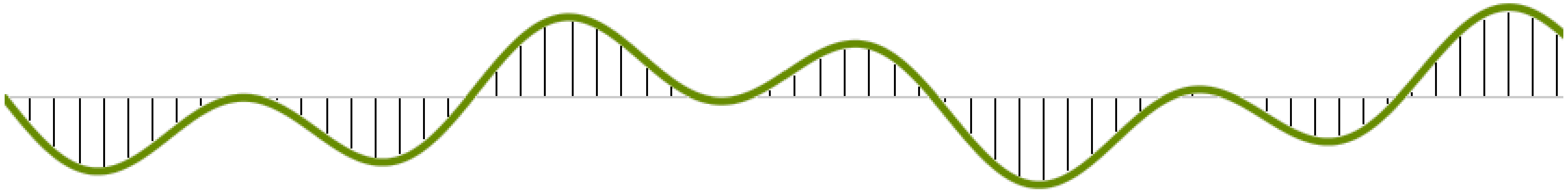
- A sound wave is an analog signal, which means that over time it is composed of infinitely many values.
- Computers have limited space, and therefore cannot store an infinite number of values in memory
- *Analog-to-Digital Converter* (ADC) transduce (convert one form of energy into another) the change in air pressure into an analogous change in electrical voltage



Time-varying voltage sampled periodically

Guitar Strings

- In order to represent a sound wave using a computer, we sample it at specific intervals.
  - wave is a continuous function f(x)
  - sampling simply means choosing certain x-values at which to compute the function, and storing the resulting y-values in a list



Guitar Strings

- We've noted that it's necessary to take at least twice as many samples as the highest frequency we wish to record
- This was proven by Harold Nyquist, and is known as the Nyquist-Shannon theorem
- Stated another way, the computer can only accurately represent frequencies up to half the sampling rate
- One half the sampling rate is often referred to as the Nyquist frequency or the Nyquist rate

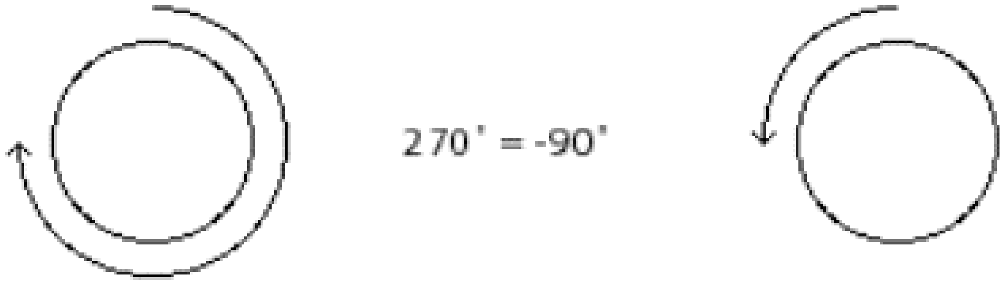
https://docs.cycling74.com/max5/tutorials/msp-tut/mspdigitalaudio.html

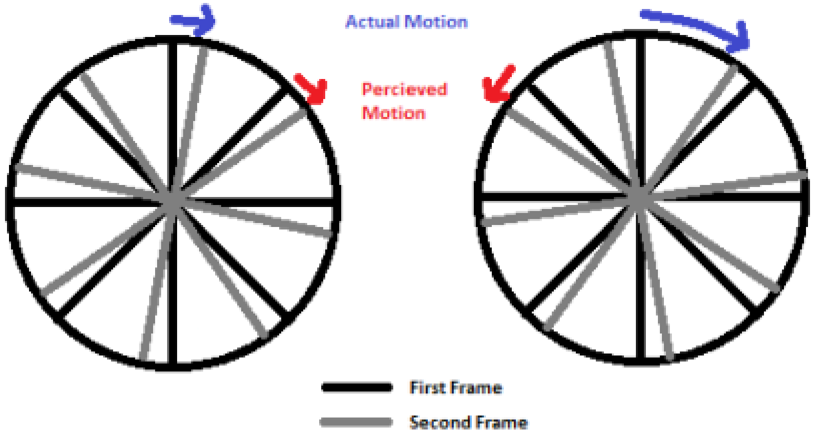
Guitar Strings

- For example, 16,000 samples of an audio signal per second, we can only capture frequencies up to 8,000 Hz
- Any frequencies higher than the Nyquist rate are perceptually '*folded*' back down into the range below the Nyquist frequency
- If the sound we were trying to sample contained energy at 9,000 Hz, the sampling process would misrepresent that frequency as 7,000 Hz -- a frequency that might not have been present at all in the original sound
- This effect is known as *foldover* or *aliasing*

Guitar Strings

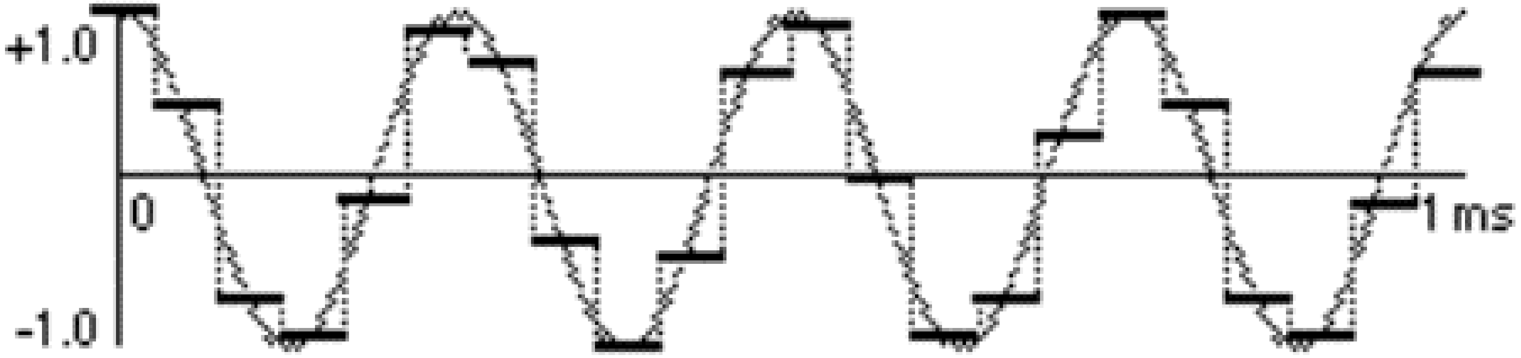
- A film camera shoots 24 frames per second
- If the car wheel spins at a rate greater than 12 revolutions per second, it's exceeding half the ‘sampling rate’ of the camera.
- The wheel completes more than 1/2 revolution per frame
- If, for example it actually completes 18/24 of a revolution per frame, it will appear to be going backward at a rate of 6 revolutions per second





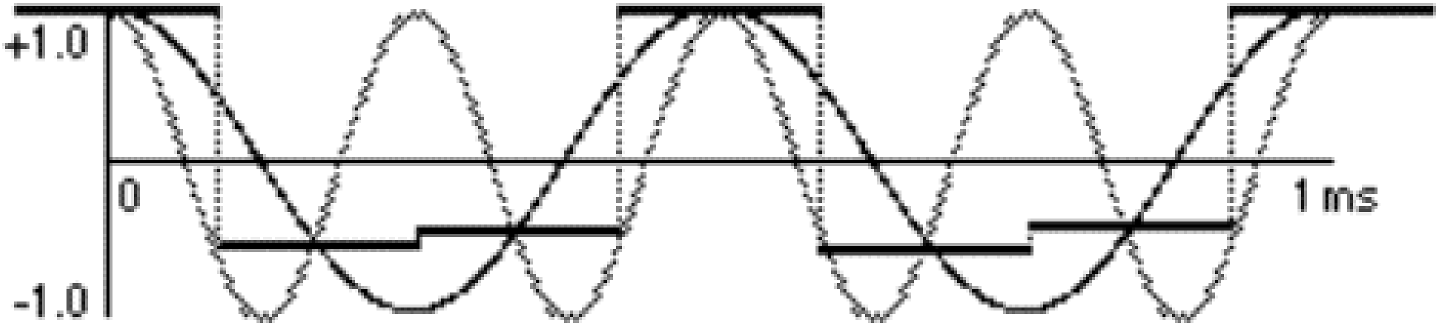
Guitar Strings

- For audio sampling, the phenomenon is practically identical
- Example
  - a 4,000 Hz cosine wave (energy only at 4,000 Hz) being sampled at a rate of 22,050 Hz
  - 22,050 Hz is half the CD sampling rate, and is an acceptable sampling rate for sounds that do not have much energy in the top octave of our hearing range

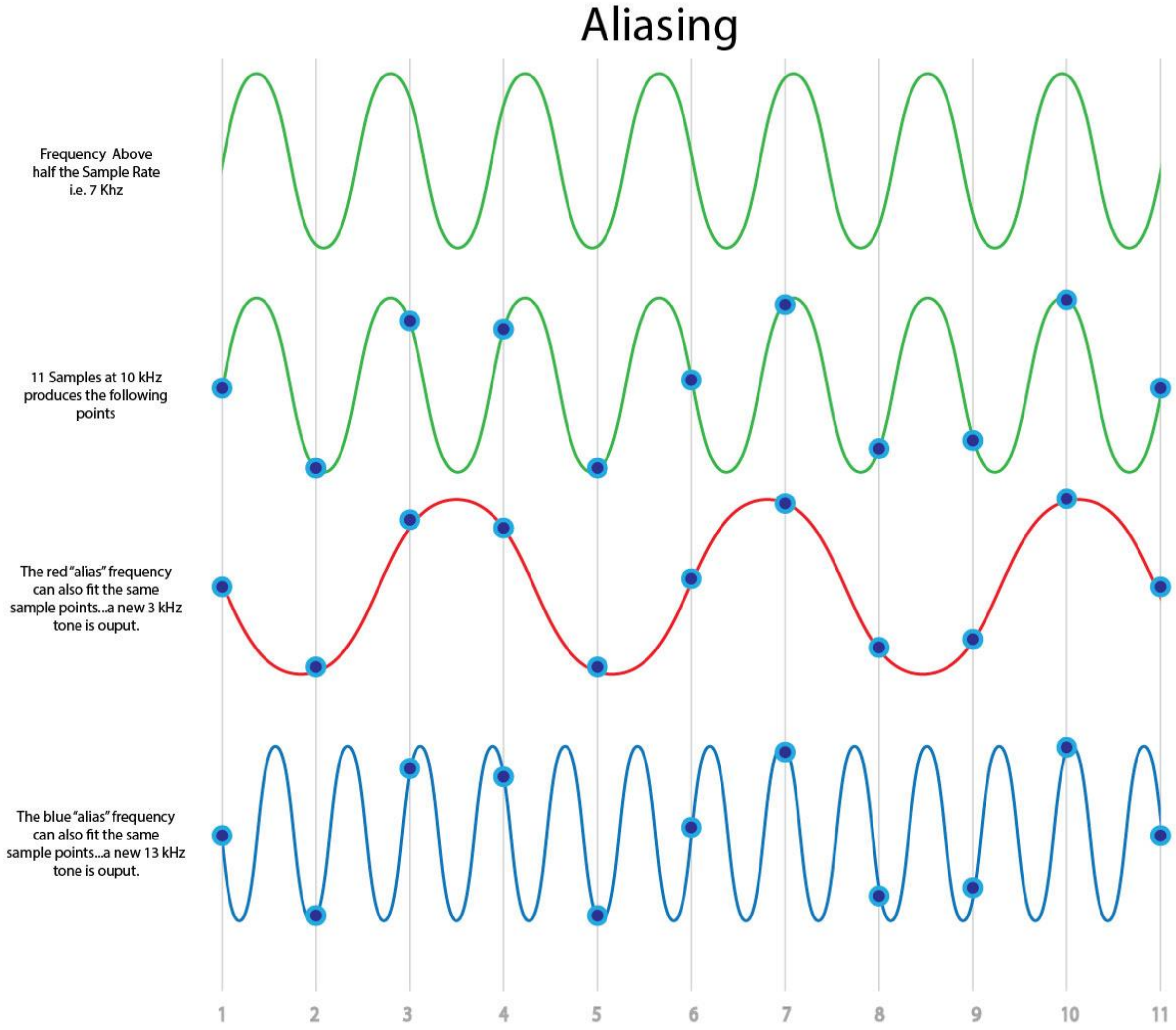


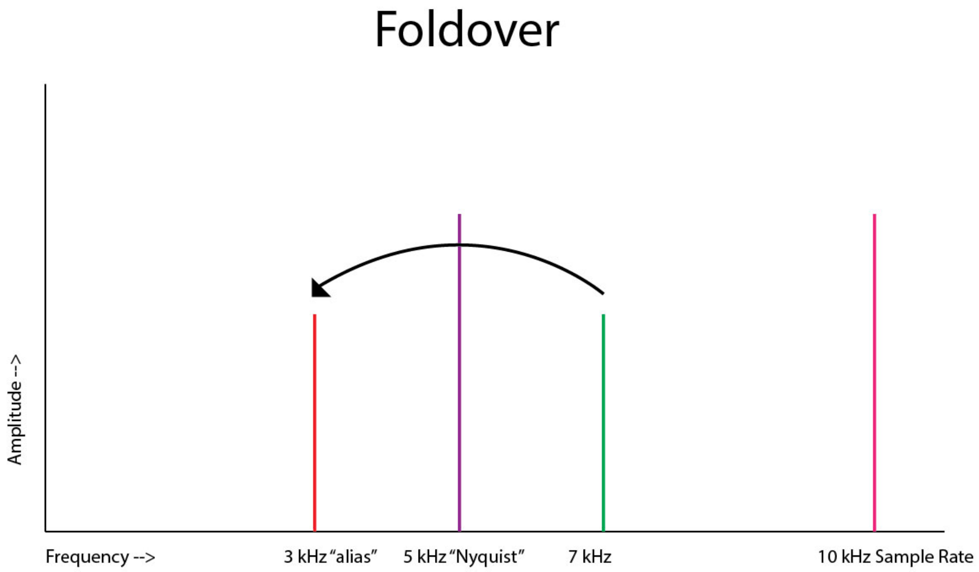
Guitar Strings

- Example
  - a 4,000 Hz cosine wave (energy only at 4,000 Hz at an inadequate rate, such as 6,000 Hz
  - The wave completes more than 1/2 cycle per sample, and the resulting samples are indistinguishable from those that would be obtained from a 2,000 Hz cosine wave.



Guitar Strings


Aliasing
Frequency Above half the Sample Rate i.e. 7 Khz
11 Samples at 10 kHz produces the following points
The red "alias" frequency can also fit the same sample points...a new 3 kHz tone is ouput.
The blue "alias" frequency can also fit the same sample points...a new 13 kHz tone is ouput.
1
2
3
4
5
6
7
8
9
10
11

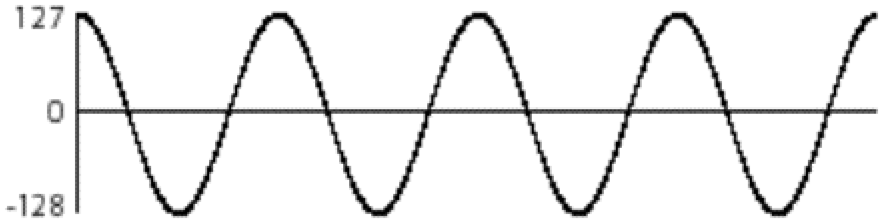


http://www.realhd-audio.com/?p=2983

Guitar Strings

Precision of quantization

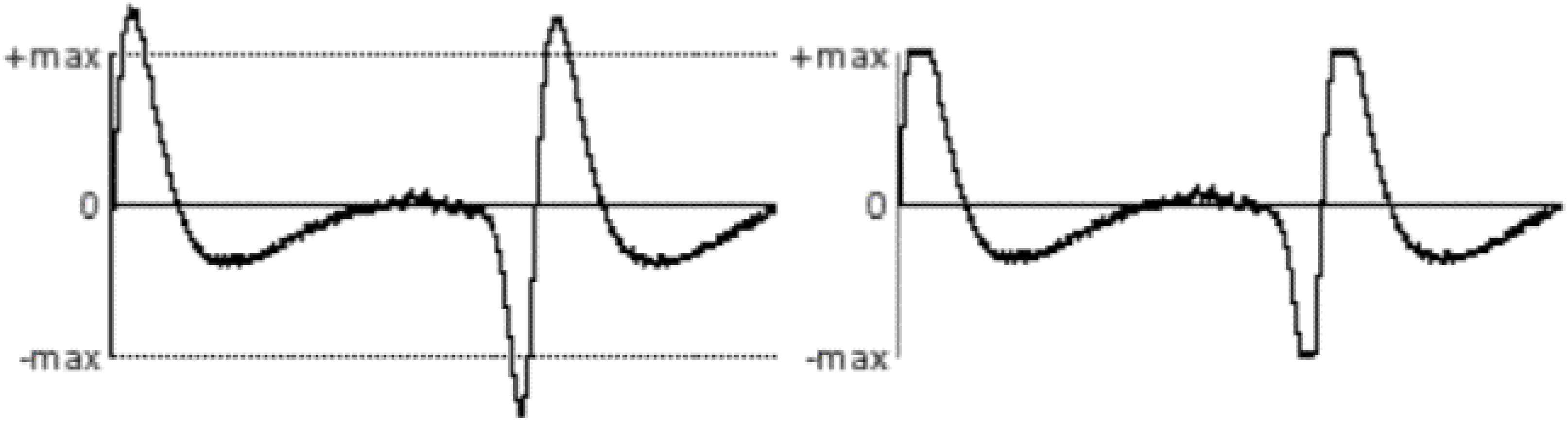
- Each sample of an audio signal must be ascribed a numerical value to be stored in the computer
- A single byte (8 bits) of computer data can express one of $2^8 = 256$ possible numbers
- It determines the resolution with which we can measure the amplitude of the signal
- If we use only one byte to represent each sample, then we must divide the entire range of possible amplitudes of the signal into 256 parts since we have only 256 ways of describing the amplitude.



Guitar Strings

Clipping

- If the amplitude of the incoming electrical signal exceeds the maximum amplitude that can be expressed numerically, the digital signal will be a clipped-off version of the actual sound



Guitar Strings

- Standard CD sound is 44.1KHz, or 44,100 samples per second, which means the highest note it can record is about 20kHz
- Most professional audio programs are capable of recording at 48kHz, and 96kHz isn't unheard of
- Analog : although the human ear cannot hear tones above about 20KHz (or 22KHz, or even 48KHz), those tones nonetheless affect the character of the overall signal at any given moment. Therefore, eliminating those tones makes the tone "inaccurate."
- Digital : there are far more opportunities for distortion in analog forms of recording and playback

Guitar Strings

- Some filmmakers are now using infrasound to induce fear in audiences
- These extreme bass waves or vibrations have a frequency below the range of the human ear
- While we may not be able to hear infrasound, it has been demonstrated to induce anxiety, extreme sorrow, heart palpitations and shivering
- Producers of the 2002 French psychological thriller *Irreversible* admitted to using this technique
- On 31 May 2003 a group of UK researchers held a mass experiment, where they exposed some 700 people to music laced with soft 17 Hz sine waves

The Karplus-Strong Algorithm

- Now that we have a physical idea of what's happening in a plucked string, how can we model it with a computer?
- The *Karplus-Strong* algorithm
  - Karplus, Kevin, and Alex Strong. "*Digital Synthesis of Plucked-String and Drum Timbres*." Computer Music Journal 7, no. 2 (1983): 43–55.

The Karplus-Strong Algorithm

- Wavetable synthesis was invented in the 1970's

http://flothesof.github.io/Karplus-Strong-algorithm-Python.html

The Karplus-Strong Algorithm

Plucking of a guitar string

- The *length* of the string determines its *fundamental frequency* of vibration.
- Model a guitar string:
  - sampling its displacement (a real number between –1/2 and +1/2) at **N** equally spaced points in time
- The integer **N** equals the sampling rate (44,100 Hz) divided by the desired fundamental frequency, rounded up to the next integer

The Karplus-Strong Algorithm

How it works:

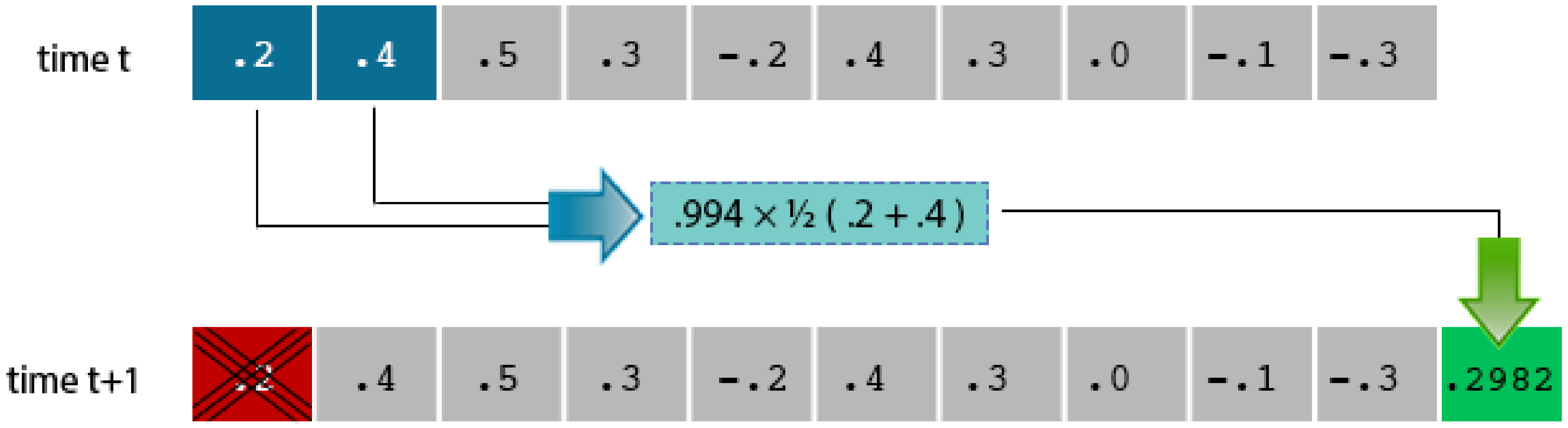
- Start with a sequence full of random values: noise.
- Generate new values by replicating previous ones at a particular frequency p
- The Karplus-Strong update is defined in terms of previous samples:

Y is a sequence of samples, and p is the frequency of the note we want to simulate (e.g., 440 for an A)

$$Y_t = 0.996 * \frac{Y_{t-p} + Y_{t-p+1}}{2}$$

The Karplus-Strong Algorithm

- The Karplus-Strong algorithm simulates vibration by maintaining a ring buffer of the **N** samples *scaled* by an energy decay factor of 0.994



time t	.2	.4	.5	.3	-.2	.4	.3	.0	-.1	-.3	
time t+1	~~.2~~	.4	.5	.3	-.2	.4	.3	.0	-.1	-.3	.2982

.994 × ½ (.2 + .4)

the Karplus-Strong update

Guitar Strings

Why does this work?

- If you had a perfect pluck at the perfect location with the perfect transfer of energy, you’d get a note that was “perfect”
- It wouldn’t be a pure sine wave since strings have harmonics (integer multiple frequencies) beyond their basic tuning, but it would be a pure note

http://amid.fish/karplus-strong

Guitar Strings

- In reality, immediately after plucking the string, there are a lot of vibrations in there that “don’t belong” due to the imperfect pluck
- The string is tuned
  - it wants to be vibrating a specific way
  - over time the vibrations evolve from the imperfect pluck vibrations to the tuning of the guitar string
- Averaging the samples together removes the higher frequency noise/imperfections
  - Averaging is a crude low pass filter
  - This makes it converge to the right frequencies over time

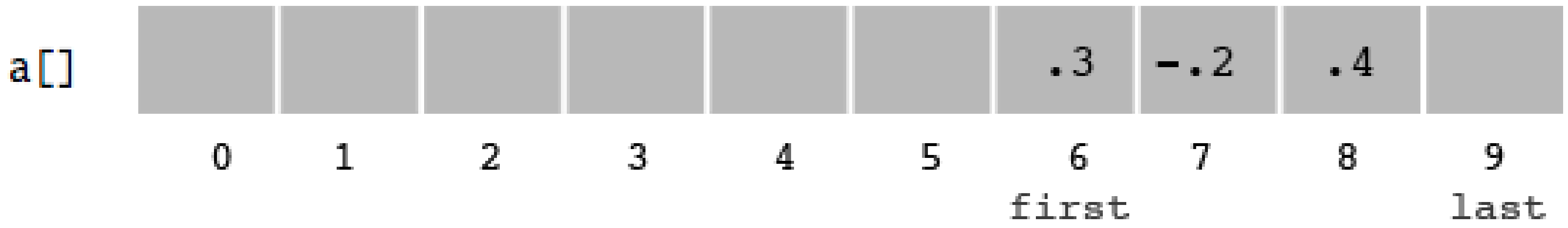
Guitar Strings

- With a real stringed instrument, when you play a note, the high frequencies disappear before the low frequencies
- The *averaging / low pass filter* makes that happen and helps it sound so realistic
- The energy in the string is being diminished as it becomes heat and sound waves and such, so the noise gets quieter over time
- You are simulating this loss when you multiply the values by a feedback value which is less than 1

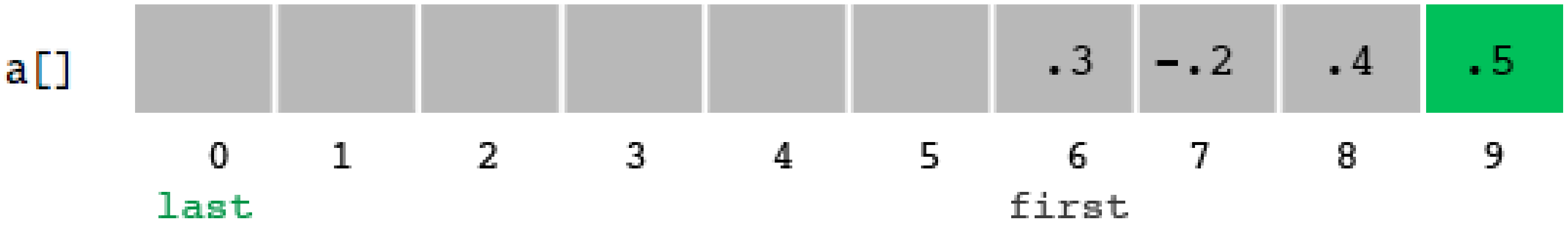
RingBuffer

- For efficiency, your RingBuffer must wrap around in the array
  - variable **`first`** - stores the index of the least recently inserted item
  - variable **`last`** - stores the index one beyond the most recently inserted item
- *Insert* : put it at index **`last`** and increment **`last`**
- *Remove* : take it from index **`first`** and increment **`first`**
- When either index equals capacity, make it *wrap-around*
- Keep in mind, the size of the *RingBuffer* is not the same as the size of the array

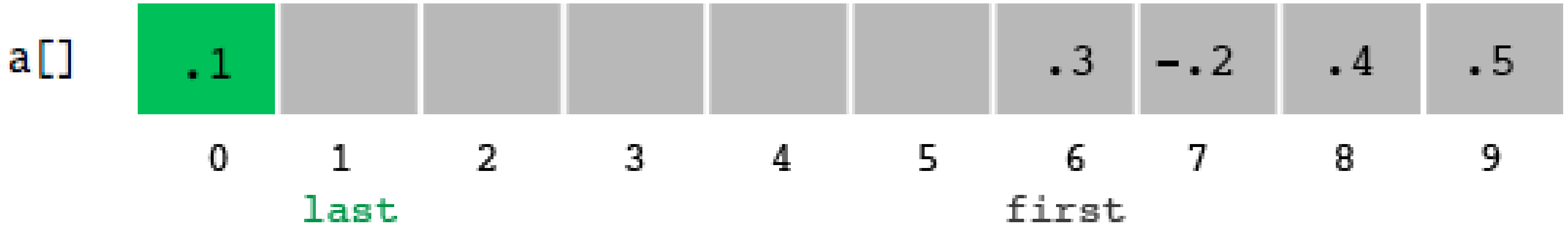
- Initial State:



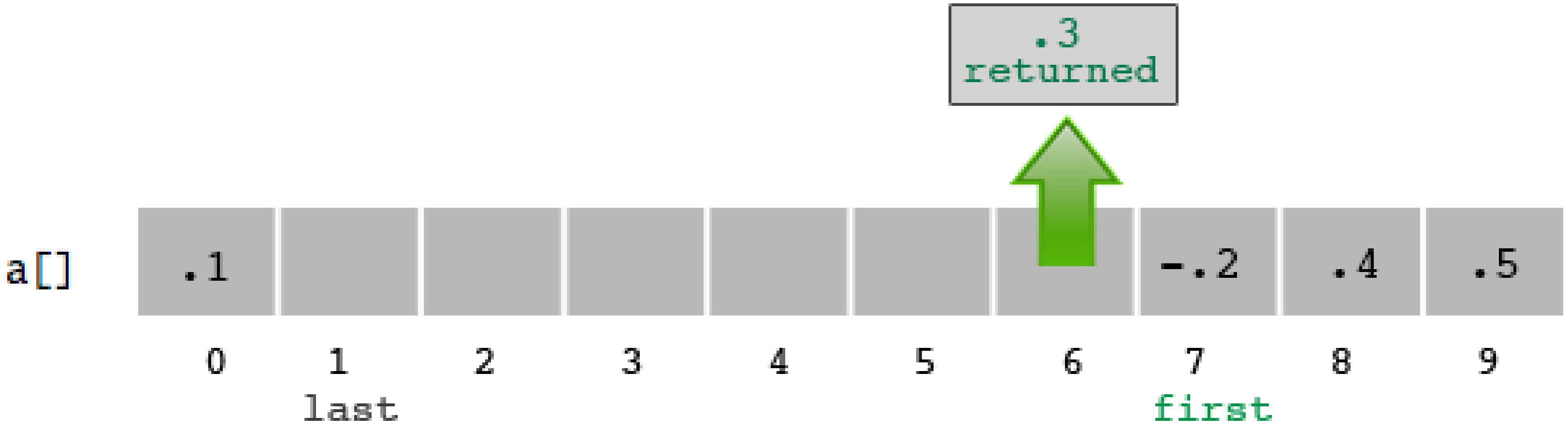
- enqueue(0.5)



- enqueue(0.1)



- dequeue()



RingBuffer

- Keep in mind, the size of the *RingBuffer* is not the same as the size of the array
- Initial State:

a[]							.3	-.2	.4	
	0	1	2	3	4	5	6	7	8	9
							first			last

- enqueue(0.5)
- enqueue(0.1)
- dequeue()

RingBuffer

- To get an accurate count of the number of elements in your `RingBuffer`, increment the instance variable `size` each time you call `enqueue()`, and decrement it each time you call `dequeue()`

GuitarString: Constructors

```
class GuitarString {
 public:
  explicit GuitarString(double frequency);
  explicit GuitarString(vector<sf::Int16> init);
  GuitarString (const GuitarString &obj) {};
      // no copy constructor

  ~GuitarString();

  void pluck();
  void tic();
  sf::Int16 sample();
  int time();

 private:
  RingBuffer* _rb;
  int _time;
};
```

GuitarString

Implicit constructor is a term commonly used to talk about two different concepts in the language

- ***implicitly declared constructor***
  - default or copy constructor
- ***implicitly defined constructor*** is a implicitly declared constructor that is used in the language and for which the compiler will provide a definition

https://codeyarns.com/2010/12/17/explicit-constructor-in-c/

GuitarString: Constructors

- A constructor with a single non-default parameter (until C++11) that is declared without the function specifier **`explicit`** is called a ***converting*** constructor
- C++11: this restriction was lifted, and constructors taking multiple parameters can now be converting constructors
- Both constructors (other than copy/move) and user-defined conversion functions may be function templates; the meaning of **`explicit`** doesn't change

http://www.dreamincode.net/forums/topic/119550-the-explicit-keyword/

GuitarString: Constructors

C++03:

- The compiler provides a default constructor, a copy constructor, a copy assignment operator (operator=), and a destructor
    - The programmer can override these defaults
    - C++ also defines several global operators (such as operator new) -- can override
    - Little control over creating these defaults
    - In the case of the default constructor, the compiler will not generate a default constructor if a class is defined with any constructors

https://en.wikipedia.org/wiki/C%2B%2B11#Explicitly_defaulted_and_deleted_special_member_functions

GuitarString

- The **explicit** keyword is used to prevent any **implicit** conversions from happening
- Example of a **const** GuitarString constructor that can use implicit conversion

```
class foo
{
public:
    foo(int x): m_data(x) {}
    int Get() { return m_data; }
private:
    m_data;
};
```

```
void Bar(foo object)
{
    int i = object.Get();
}
int main()
{
    Bar(100);
    return 0;
}
```

http://www.dreamincode.net/forums/topic/119550-the-explicit-keyword/

GuitarString: Constructors

C++11:

- allows the explicit defaulting and deleting of these special member functions

```
struct SomeType {
    SomeType() = default; // The default
    //constructor is explicitly stated
    SomeType(OtherType value);
};
```

- The `= delete` specifier can be used to prohibit calling any function

GuitarString: Constructors

C++11:

- certain features can be explicitly disabled

```
struct NonCopyable {
    NonCopyable() = default;
    NonCopyable(const NonCopyable&) = delete;
    NonCopyable& operator=(const NonCopyable&) = delete;
};
```

GuitarString: Constructors

C++11:

- When a function has been deleted, any use of that function is considered a compile error
- Note: the copy constructor and overloaded operators may also be deleted in order to prevent those functions from being used

http://www.learncpp.com/cpp-tutorial/9-13-converting-constructors-explicit-and-delete/

Implicit

Pros

- Can make a type more usable and expressive by eliminating the need to explicitly name a type when it's obvious
- Implicit conversions can be a simpler alternative to overloading.
- List initialization syntax is a concise and expressive way of initializing objects

http://google.github.io/styleguide/cppguide.html

Implicit

Cons

- Hide type-mismatch bugs
- Make code harder to read (especially with overloading)
- Constructors that take a single argument may *accidentally* be usable as implicit type conversions
- It's not always clear which type should provide the conversion
- Problem with List initialization if the destination type is implicit, particularly if the list has only a single element

Implicit

- Use the **`explicit`** keyword for *conversion* operators and *single-argument constructors*
- COPY and MOVE constructors should *not* be explicit
- Constructors that cannot be called with a single argument should usually omit explicit.
- Constructors that take a *single* **`std::initializer_list`** *parameter* should also omit explicit, in order to support copy-initialization
  - **`MyType m = {1, 2};`**

http://google.github.io/styleguide/cppguide.html

Frequencies of music notes

- In the table of frequencies, you'll find A = 440 Hz, and then

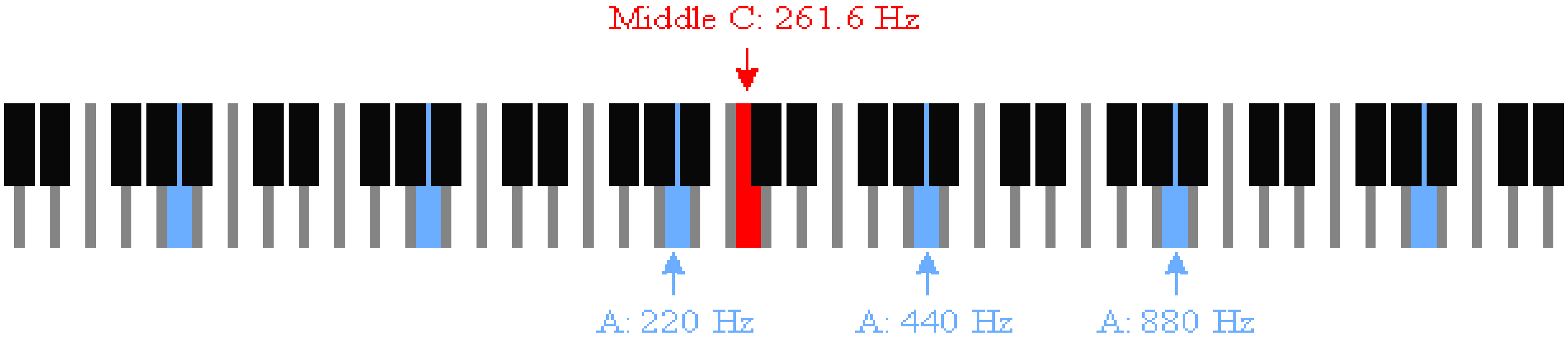
  A# = 466.16 Hz

  B = 493.88 Hz

  C = 523.25 Hz

  ...

- **Also, you can find Middle C: 261.63 Hz**

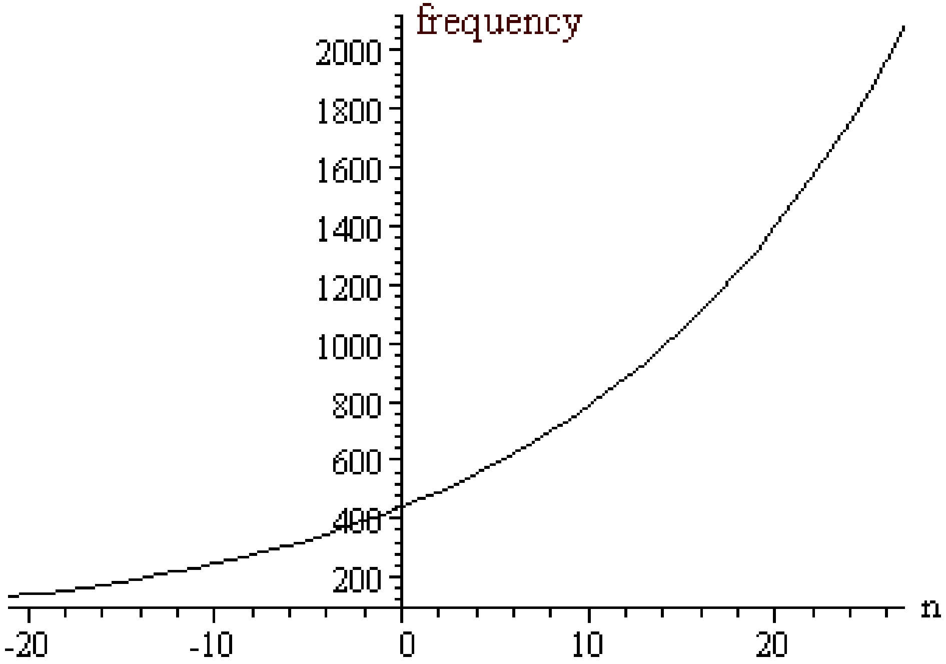


http://www.intmath.com/trigonometric-graphs/music.php

frequencies of music notes

Note	Frequency	
A	220	440
A#	233.08	466.16
B	246.94	493.88
C	261.63	523.25
C#	277.18	554.37
D	293.66	587.33
D#	311.13	622.25
E	329.63	659.26
F	349.23	698.46
F#	369.99	739.99
G	392	783.99
G#	415.3	830.61
A	440	880

frequencies of music notes



$\mathtt{frequency} = 440\times2^{n/12}$

GuitarString: "perfect 5th"

- To get a "perfect 5th" we need to play a note which has 1.5 times the frequency of A
- On a violin (or viola or any fretless stringed instrument) this is possible: perfect E at 440×1.5=660 Hz
- But a piano playing the same note will play E =659.26 Hz
    - just a little flat!

http://www.intmath.com/trigonometric-graphs/music.php

GuitarString

- **400 years ago**: keyboards (harpsichords and organs) were tuned for a particular group of keys
- Several different tuning systems in use during Bach's time
  - meantone
  - Werckmeister's 1691 tuning which allowed composers to create music in any key

GuitarString

In the early **20th century**

- Tune keyboards so that the notes were evenly spaced
- This is called equal tempered tuning
- Equal tempered tuning means all stringed instruments have to allow for the slight differences in tunings between instruments when keyboards are also involved
- Strings are usually happiest when playing with other strings only

Keys SFML

https://www.sfml-dev.org/documentation/2.5.1/classsf_1_1Keyboard.php

https://www.sfml-dev.org/tutorials/2.5/window-events.php

http://en.sfml-dev.org/forums/index.php?topic=327.0

Shallow vs Deep Copies

A *shallow copy* of an object copies all of the member field values

- This works well if the fields are values, but may not be what you want for fields that point to dynamically allocated memory
- The pointer will be copied. but the memory it points to will not be copied -- the field in both the original object and the copy will then point to the same dynamically allocated memory, which is not usually what you want
- The default copy constructor and assignment operator make shallow copies

http://www.fredosaurus.com/notes-cpp/oop-condestructors/shallowdeepcopy.html

Shallow vs Deep Copies

A *deep copy* copies all fields, and makes copies of dynamically allocated memory pointed to by the fields

- To make a deep copy, you must write a copy constructor and overload the assignment operator, otherwise the copy will point to the original, with disastrous consequences

http://www.learncpp.com/cpp-tutorial/915-shallow-vs-deep-copying/

Shallow vs Deep Copies

Deep copies need

- If an object has pointers to dynamically allocated memory, and the dynamically allocated memory needs to be copied when the original object is copied, then a deep copy is required
- A class that requires deep copies generally needs:
  - A constructor to either make an initial allocation or set the pointer to NULL
  - A destructor to delete the dynamically allocated memory
  - A copy constructor to make a copy of the dynamically allocated memory
  - An overloaded assignment operator to make a copy of the dynamically allocated memory

http://www.fredosaurus.com/notes-cpp/oop-condestructors/shallowdeepcopy.html

Shallow vs Deep Copies

http://www.learncpp.com/cpp-tutorial/915-shallow-vs-deep-copying/

rand()

https://en.cppreference.com/w/cpp/numeric/random
https://channel9.msdn.com/Events/GoingNative/2013/rand-Considered-Harmful