

# ChuckK

*Introducing on-the-fly Audio Programming  
Language*

*AJAY KAPUR*



# Outline

- Objected Oriented Musical Programming
- Introducing Chuck
- Concepts: types, arrays, control structures, classes, objects
- Synthesis, Analysis, Composition
- Real-time MIDI Devices
- Final Project:
  - Live Performance
  - Short Composition

# What is Chuck?

- Audio Programming Language
- Author: Ge Wang (and others)
- MIDI, OSC, HID Device, Multi-Channel Audio
- MacOS X, Windows, Linux
- Real-Time
  - Synthesis
  - Composition
  - Performance
  - Analysis
- FREE!!!!

# Chuck Key Words

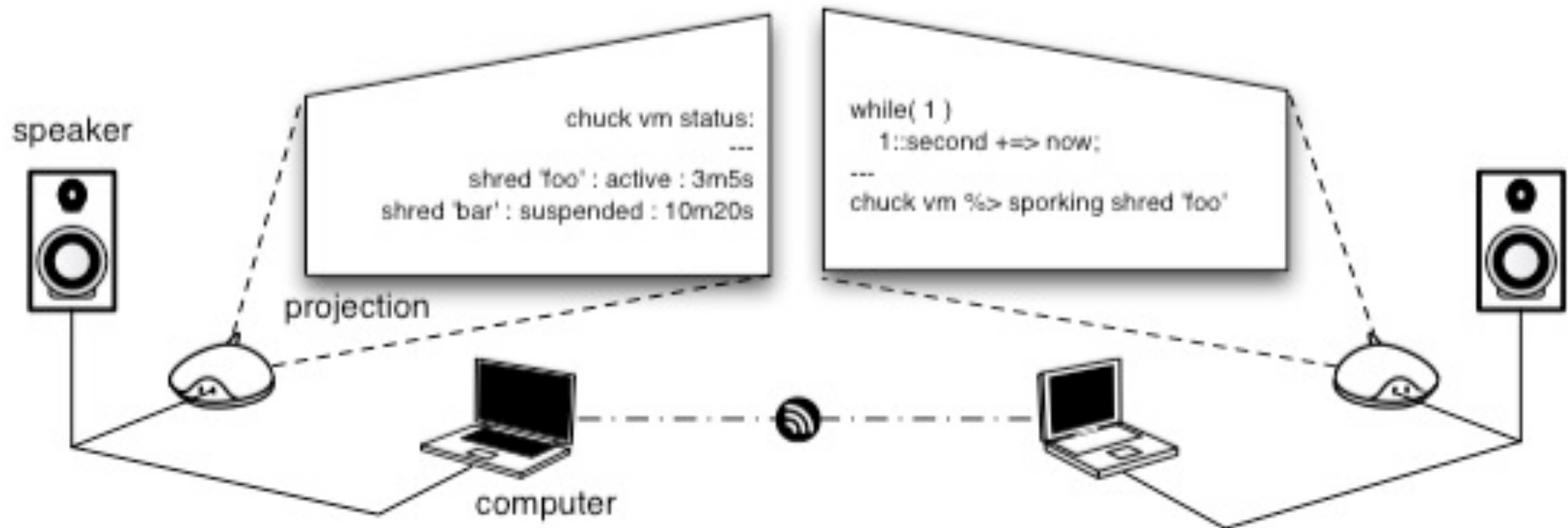
- What makes Chuck Different?
  - “strongly-timed”
  - “Concurrent programming model”
  - Modify code “on-the-fly”

# Chuck on Stage

Examples of work  
From Ge Wang, Perry Cook and Ajay Kapur

# Live Coding

- “On-the-fly Counterpoint”
- Ge Wang and Perry Cook
- 2003.11.14



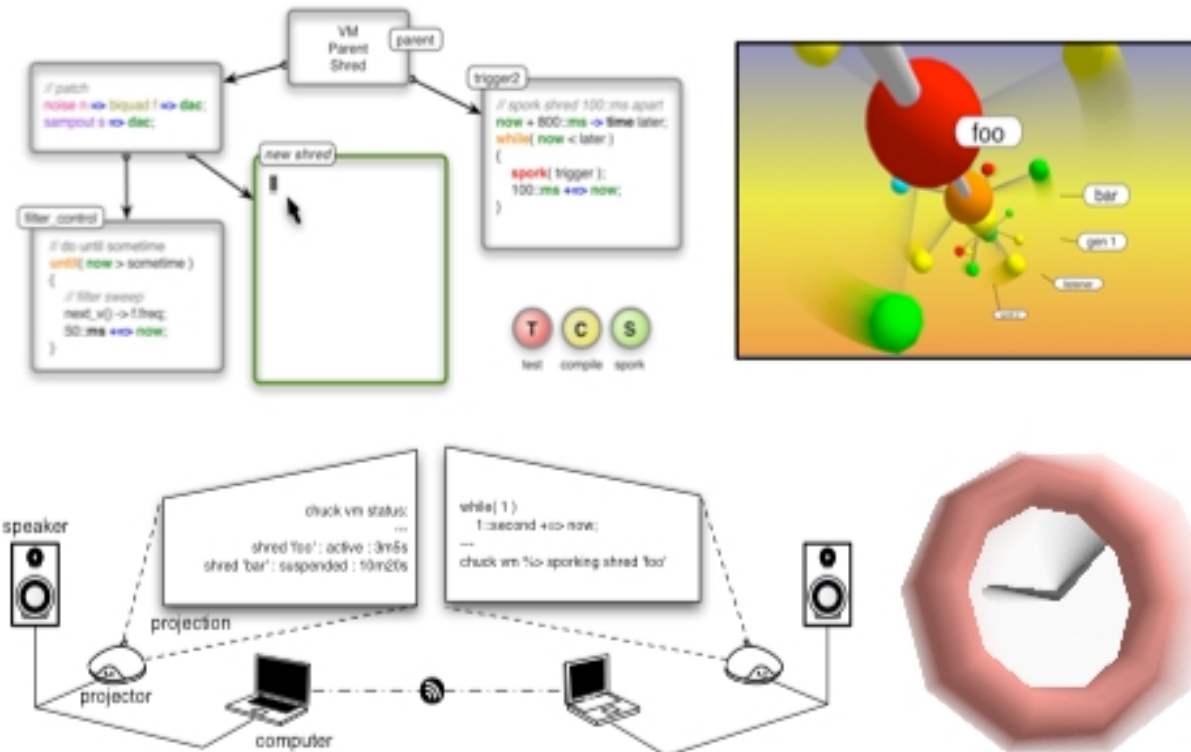
# Evolution with Controllers

- “On-the-Fly Counterpoint”
- Ge Wang and Perry Cook
- SIGGRAPH 2006



# The Audicle

- Graphical Programming Environment using Chuck
- Demo [video](#)



# Plork

- Princeton Laptop Orchestra
- ["More Specific"](#)
- Gamelan Djembe Fusion



# The KiOm

- MIDI Accelerometer Controllers
- Demo [video](#)



# Robotics

- Interfacing the ESitar with the MahaDeviBot
- [Singapore \(2007\) Video](#)
- Jazz Fest (2007) Video



# ChuckK Basics

Installation

“Hello World”

SinOsc

Types and Variables

# Installation

- <http://chuck.cs.princeton.edu>
- Follow Links for [Downloading ChuckK](#)
  - Executable (don't need source, yet :)
    - For MAC: Need to use Terminal (Applications/Utilities)
    - For Windows: Need to install Cygwin and DirectX (LINUX kernel)
- miniAudicle [Mac Windows](#)

# Hello World

- Open MidiAudicle
- Start Virtual Machine
- Type code into box
  - <<<"Hello World">>>;
- Add Shred

# Sine Waves

```
SinOsc s => dac;  
.2 => s.gain;  
220 => s.freq;  
1::second => now;
```

# Durations

- “Now” variable to control Time!
- Reserved Words:
  - samp
  - ms
  - second
  - minute
  - hour
  - day
  - week
- Example:
  - Set Duration to “day”
  - Change s.freq
  - Replace Shreds
  - Remove Shred

# Comments

- Notes to yourself and other programmers

```
// this is a comment  
int foo; // another comment
```

```
/*  
  this is a block comment  
  still going...  
*/
```

# Types

- Integer
  - int
  - ex. 42
- Floating Point
  - float
  - ex. 3.2424
- Chuckian time
  - time
- Chuckian Duration
  - dur
  - 5::second
- Complex Numbers
  - complex
- Polar Coordinates
  - polar
- No type
  - void

# Variables (int)

- `int foo;`
  - `2 => foo;`
  - `<<<foo>>>;`
- `2 => int foo;`
  - `<<<foo>>>;`
- `10 * foo => foo;`
- `10 *=> foo;`

# Variable (float)

- 5.5835 => float f;  
    <<<f>>>;

# Make your own Durations

- `1::second => dur note;`
- `4::note => dur measure;`
- `SinOsc s => dac;`
  - `.2 => s.gain;`
  - `220 => s.freq;`
  - `1::measure => now;`

# Time

- `now => time t;`  
`<<<t>>>;`
- `10::second => now;`
  - `<<<now>>>;`

# Control Structures

If, else  
for  
While  
until  
Continue/break

# if

- The if statement executes a block if the condition is evaluated as non-zero

```
if ( condition )  
{  
    // insert code here  
}
```

# If - example

```
Sin0sc s => dac;  
220 => s.freq;  
.2 => s.gain;  
1 => int k;  
  
1::second => now;  
  
if (k == 1)  
{  
    440 => s.freq;  
    .2 => s.gain;  
    1::second => now;  
}
```

# else

- The else statement can be put after the if block to handle the case where the condition evaluates to 0

```
if (condition)
{
    // your code here
}
else
{
    // your code here
}
```

# nested if

```
if (condition)
{
    // code here
    if (condition)
    {
        // something
    }
}
```

# for loop

- A loop that iterates a given number of times.
- Temporary variable is declared that keeps track of the current index and is evaluated and incremented at each iteration.

```
// for loop
for (o => int foo; foo < 4; foo++ )
{
    // debug print value 'foo'
    <<< foo >>>;
}
```

# for (example)

```
SinOsc s => dac;  
.2 => s.gain;
```

```
for (1 => int foo; foo < 8; foo++)  
{  
    // debug print value of 'foo' <<<foo>>>;  
    50*foo => s.freq;  
    1::second => now;  
}
```

# while

- While statement is a loop that repeatedly executes the body as long as the condition evaluates to non-zero

```
//infinite loop
```

```
while (true)
```

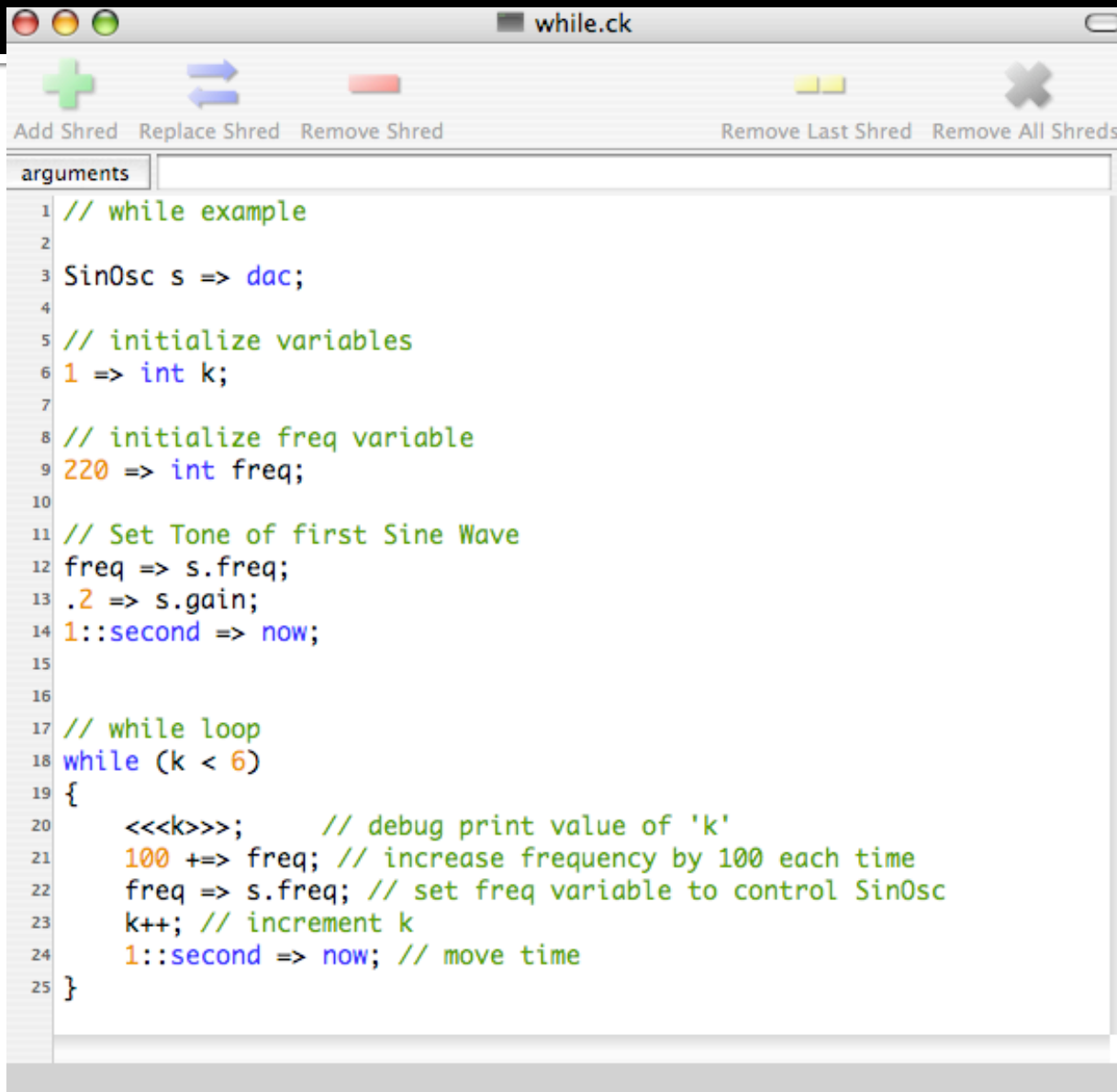
```
{
```

```
    // loop forever
```

```
    1::second => now;
```

```
}
```

# While example



The image shows a code editor window titled "while.ck". The window has a toolbar with icons for "Add Shred" (a green plus), "Replace Shred" (a blue double arrow), "Remove Shred" (a red minus), "Remove Last Shred" (a yellow double arrow), and "Remove All Shreds" (a grey X). Below the toolbar is a tab labeled "arguments" and a text area containing the following code:

```
1 // while example
2
3 SinOsc s => dac;
4
5 // initialize variables
6 1 => int k;
7
8 // initialize freq variable
9 220 => int freq;
10
11 // Set Tone of first Sine Wave
12 freq => s.freq;
13 .2 => s.gain;
14 1::second => now;
15
16
17 // while loop
18 while (k < 6)
19 {
20     <<<k>>>; // debug print value of 'k'
21     100 +=> freq; // increase frequency by 100 each time
22     freq => s.freq; // set freq variable to control SinOsc
23     k++; // increment k
24     1::second => now; // move time
25 }
```