“ChucK: A Strongly-Timed Music Programming Language”

excerpt (pp. 170-175) from *Artful Design*,
Chapter 4 "Programability and Sound Design"

https://artful.design/
Chuck
A STRONGLY-TIMED MUSIC PROGRAMMING LANGUAGE!

Chuck is a programming language for sound generation and music creation. It was designed as a tool for researchers, composers, and sonic tinkerers to program musical sounds by working directly with a notion of time itself. It is open-source and freely available. (And, as I like to say, it crashes equally well on all commodity operating systems!) It has a personality, and is pretty easy to learn.

I started designing Chuck back in 2002 (when I was in grad school). Since that time, Chuck has been used to craft instruments for laptop orchestras and is the audio engine in OCARINA, running inside millions of phones...

2002, in the bowels of the computer science department at Princeton...

Chuck represents an extreme expression of imperative programming, asking the programmer to explicitly specify even the passage of time to control audio synthesis. What possessed you to design it like that?

Chuck's design choices present a different way of thinking. A different aesthetic of programming sound. I wanted to create a tool that could specify precisely how and when things happen. The way Chuck handles time and parallelism is designed as a way to think about music itself...

2002, in the bowels of the computer science department at Princeton...

Georg Essl
Fellow computer music researcher, Big Brother

Perry R. Cook
Advisor, Life Mentor, Chuck co-conspirator, Zen Master

We can dissect our Bleep/Bloop code example, written in Chuck, and point out some of its functionalities...

This symbol, the Chuck operator, connects sound-processing modules like a digital audio patch cable!

This generates a sine wave!

SinOsc ada => dac;

while (true)
{
    Math.random2f(30, 10000) => ada.freq;
    100:ms => now;
}

This is a loop. The code within the { } is run repeatedly!

This represents the audio output (e.g., loudspeakers, headphones).

Now is Chuck's notion of "the current time". We tell Chuck what time "now" should be (e.g., "let it be 100 milliseconds in the future"), and Chuck makes it so by waiting the specified duration.

Most computer languages have ways to deal with time (e.g., a "wait" directive), but these approaches are often coarse and unpredictable. In Chuck, time is ultra-precise because it is inferred from the digital audio stream itself. Sound in Chuck is both the output and the means by which Chuck keeps track of time.

The functionalities of a programming language determine what you can do with it. The ways in which a language presents its functionalities to you constitute its aesthetics -- they shape how you think about what you want to do.
THE SAME GENERAL FUNCTIONALITY MAY BE AVAILABLE IN DIFFERENT PROGRAMMING LANGUAGES, BUT THE SPECIFIC WAY A PARTICULAR LANGUAGE EXPRESSES THAT FUNCTIONALITY HAS TO DO WITH THE AESTHETICS OF THAT LANGUAGE -- AND CHANGES HOW YOU THINK ABOUT THE TASK AT HAND!

I THINK THIS MAY BE WHY WE HAVE SO MANY PROGRAMMING LANGUAGES! IT’S NOT SO MUCH THAT THEY ALL DO DIFFERENT THINGS, BUT THAT EACH ONE MAKES YOU THINK DIFFERENTLY...

FOR EXAMPLE, THE WAY YOU WORK WITH TIME IN CHUCK LEADS ITSELF TO REASONING CLEARLY ABOUT WHEN THINGS HAPPEN -- IT’S STRAIGHTFORWARD TO BUILD A MENTAL FLOW CHART OF WHAT HAPPENS AND WHEN...

EXECUTE LOOP BODY?

YES

GENERATE A RANDOM NUMBER -- AND SET IT AS ADA’S FREQUENCY

WAIT 100 MILLISECONDS, MOVING CHUCK’S INTERNAL CLOCK FORWARD AND SYNTHESIZING AUDIO

NO

END HERE*

*BY THE WAY, THIS PARTICULAR CODE IS PROGRAMMED TO NEVER REACH THIS STATE. IT ALWAYS TAKES THE YES PATH.

FOR EXAMPLE, HERE IS A DRUM MACHINE, WITH FOUR INDEPENDENT PROGRAMS OF SOUND-GENERATING CODE, EACH “DOING ITS OWN THING” IN TIME.

THIS IS A FORM OF TIME-BASED CONCURRENT PROGRAMMING, WHERE SEVERAL PROGRAMS CAN EXECUTE SIMULTANEOUSLY (INSTEAD OF ONE AFTER ANOTHER), SYNCHRONIZED BY TIME.

IN ESSENCE, TIME AND CONCURRENCY IN CHUCK ARE TWO SEPARATE DIMENSIONS THAT CAN WORK TOGETHER -- AS A GENERALIZED WAY TO MODEL SOUND BOTH AS A PHENOMENON OVER TIME AND AS A MIXTURE OF MANY ELEMENTS HAPPENING AT THE SAME TIME.
CHUCK CODE, IN DESIGN, IS A COMPLETE SPECIFICATION OF NOT ONLY WHAT, BUT ALSO WHEN THINGS HAPPEN. IT COMPULSES THE PROGRAMMER TO BE EVER AWARE OF TIME WHEN WRITING CODE, AND TO BE ABLE TO REASON ABOUT TIME PRECISELY WHEN READING IT. THIS WAY OF WORKING AIDS TO FOCUS THE PROGRAMMER’S MENTAL EFFORTS ON SOUND AND HOW IT CHANGES OVER TIME, LEAVING THE LOW-LEVEL DETAILS FOR CHUCK TO HANDLE.

WHEN YOU RUN THE CODE, CHUCK GOES TO WORK! HERE’S A VISUALIZATION OF HOW OUR TIME-BASED CODE BECOMES SOUND...

OUR CHUCK CODE RUNS AT SPECIFIC POINTS IN TIME (E.G., EVERY 100MS), CONTROLLING SOUND PARAMETERS (E.G., SINE WAVE FREQUENCY), CHUCK INTERNALLY KEEPS ITS OWN REPRESENTATION OF TIME (KNOWN TO THE PROGRAMMER AS "NOW") AND AUTOMATICALLY USES THIS INFORMATION TO DETERMINE WHEN THINGS HAPPEN AS SPECIFIED BY THE CODE.

TIME IN CHUCK ILLUSTRATES HOW DESIGN IS CRUCIAL WHEN MAKING A NEW TOOL. TOOLS DO MORE THAN SERVE A PURPOSE — THEY SHAPE OUR THINKING. A USEFUL TOOL SUGGESTS PARTICULAR WAYS OF WORKING (E.G., "IF YOU HAVE A HAMMER, THEN EVERYTHING LOOKS LIKE A NAIL").

THAT’S WHY DESIGN IS RELEVANT IN CREATING ANY TOOL. THE CHOICES THAT GO INTO ITS DESIGN IMPACT HOW USERS THINK ABOUT WHAT THEY DO WITH THE TOOL AND HOW THEY FEEL WHEN THEY ARE USING IT, ULTIMATELY SHAPING THE KINDS OF THINGS THEY CREATE WITH THE TOOL.

MY INTENTION IN DESIGNING CHUCK WAS TO DEVELOP A SIMPLE YET FLEXIBLE WAY TO PROGRAM TIME AS A MUSICAL CONSTRUCT. IT WAS AN EFFORT TO HIDE THE LOW-LEVEL COMPLEXITIES OF DIGITAL AUDIO SYNTHESIS WHILE EXPOSING AN ULTRA-PRECISE HIGH-LEVEL WAY TO CONTROL IT. THE RESULT IS BOTH A TOOL AND A WAY OF THINKING TO WORK WITH MUSICAL SOUND, CENTERED AROUND THE FLOW OF TIME.

...I CALL IT STRONGLY-TIMED PROGRAMMING!

WITH THIS IN MIND, LET’S EXAMINE A MORE COMPLEX CASE STUDY OF CONTROLLING SOUND OVER TIME...