FINAL TRANSCRIPT

Tubes

by
Sarah Thompson

Narration dialogue edits by
Chuck Fry

SGT Inc., NASA Ames Research Center,
MS269-3, Moffett Field, CA 95014
Email: sarah.thompson@nasa.gov

TITLE SEQUENCE

Scene opens with a star field.

SUPER: NASA, ARC and SGT logos, then the caption, “RSE Tech Area, Code TI”

Camera POV descends through clouds, ending within the PROGRAMMER’s cube.

CUT TO:

INT. CUBE FARM - DAY

A PROGRAMMER is at work. In TIMELAPSE, we see the programmer become increasingly frustrated.

NARRATOR (V.O.)
Programming is hard. Programmers are the heros of our time, toiling long hours at the code face to construct the products we all rely upon in our daily lives. The job of a programmer is to write code. Code is expressed in a programming language, commonly C, C++ or Java. When the problem in hand fits the capabilities of the language, this is often straightforward. But, when the problem doesn’t fit the language, things get less pleasant. Such code tends to be much more long-winded, difficult to understand and often buggy.

INT. MACHINE SHOP - DAY

We see a CNC MILLING MACHINE perform a repetitive task.

NARRATOR (V.O.)
Nearly all commonly used languages are procedural. This makes them really good at expressing processes where everything happens in a very specific, well understood order.
(MORE)

(CONTINUED)

CONTINUED:

NARRATOR (V.O.) (CONT'D)
Unfortunately, however, this paradigm tends to break down when code has to deal with the need to do more than one thing at a time.

CUT TO:

INT. MOORE’S LAW SEQUENCE

Multiple copies of the programmer move across the screen increasingly quickly.

SFX: Clock ticks faster and faster

NARRATOR (V.O.)
Moore’s law has given us faster and faster processors capable of running procedural code ever quicker with each new generation.

CUT TO:

EXT. DESERT ROAD - DAY

The Programmer is apparently driving a car, wearing a helmet as if in a race.

Montage: Speed limit sign, 65 MPH. No aircraft, no space rockets signs accompany it.

NARRATOR (V.O.)
Unfortunately, however, physics is getting in the way of making individual processor cores faster.

CUT TO road sign, “50 MPH, Warning! Speed limit enforced by laws of physics, SLOW”

BACK TO the programmer driving the car. The CAMERA pulls back and up, revealing them to really be driving 25 cars bolted together in a 5 by 5 grid.

NARRATOR (V.O.)
Consequently, the only practical way to move forward is to run multiple processor cores in parallel.

(CONTINUED)

CONTINUED:

NARRATOR (V.O.)
We now live in a world where even our cellphones have multiple processor cores.

CUT to helicopter flyby of cars driving down the desert road.

NARRATOR (V.O.)
Today's most commonly used programming languages simply weren't designed to deal with such parallelism. All too often, programmers must manage today's highly concurrent workloads with languages designed for serial applications.

CUT TO:

INT. CUBE FARM - DAY

The programmer sits at a table with a pile of plumbing parts. They are trying to build something, but none of the parts fit properly -- they throw the rejects over their shoulder without looking, but they faithfully hit the waste bin every time.

NARRATOR (V.O.)
Programmers face a second difficulty. It has long been a purported advantage of modern programming languages that code reuse is possible -- that is, it should be possible to create software components that can be slotted together later to make new applications without having to rewrite everything from scratch. Unfortunately, however, in practice this rarely works well, with a lot of messy glue code being necessary to join everything together.

The programmer grabs a lump of clay and slaps it on top of some parts, then sticks some more tubes into it.

SCREEN SHOT: EMAIL CLIENT

We see a photo as an inclusion in an email to their boss, with a subject line of, “There, I fixed it!”

In stop motion, we see the clay grow menacing tentacles.

NARRATOR (V.O.)
Glue code tends to be difficult and often boring to write. Consequentially, it can be quite bug prone.

SCREEN SHOT: NOKIA/TROLL TECH WEB SITE FOR QT

We zoom into a code example showing signals & slots.

NARRATOR (V.O.)
In the 1990s, a small Norwegian company, Troll Tech, invented an extension to the C++ language that introduced the concept of signals and slots.

CUT TO Qt demo application with an LED numeric readout, a large DIAL and a RESET BUTTON. The number changes as the dial is rotated with the mouse. Clicking the reset button snaps the dial to 0 (hard left/anticlockwise). The code fragments responsible for routing the relevant signals appear as the controls are used.

NARRATOR (V.O.)
Their product, Qt, was a graphical user interface library, now owned by Nokia, that built on their signals and slots technology to greatly simplify the difficulty of implementing user interfaces in C++.

CUT TO:

INT. CUBE FARM - DAY

We see the programmer attempting the same problem that they attempted with the clay. This time, however, they have some tubes that they are using to connect the badly fitting parts. As the tubes click into place, they GLOW in different primary colours, underscored by SFX.

(CONTINUED)

CONTINUED:

NARRATOR (V.O.)
Signals and slots made it possible to connect objects together that weren’t designed to work together. Because a signal/slot connection only needs to match types on the connection itself, not on the whole object, very little glue code is needed. Unfortunately for our programmer, though signals and slots make huge inroads to the problem of code reuse, they don’t help with concurrency.

MONTAGE: CIRCUIT BOARDS

We see a series of macro shots of circuit boards. (Images courtesy cgtextures.com)

NARRATOR (V.O.)
There is already a well-known paradigm that works well for concurrency. Electronic circuits are by their nature concurrent -- all their parts typically are all working at once, not one at a time. The digital electronics paradigm is centred on data flow, rather than control flow. In practice, doing control flow with digital electronics tends to be difficult, often requiring tricky work-arounds.

TITLE CARD: TUBES

We see a whiteboard stop motion animation that serves as the title card for Tubes. One of the frames of this sequence is shown on the whiteboard behind the Programmer on the wall of their cube, and was also used as the basis of the CNC program being executed by the milling machine in an earlier scene.

NARRATOR (V.O.)
Signals and slots make it much easier to express data flow in C++, but Qt concentrated mainly on single-threaded code.
(MORE)

(CONTINUED)

CONTINUED:

NARRATOR (V.O.) (CONT'D)
Tubes are a generalization of signals and slots that extends the concept to encompass concurrency.

CUT TO:

MOTION GRAPHICS SEQUENCE

We see the outline of the squarer class.

NARRATOR (V.O.)
Tubes extend the C++ concept of classes to include inputs and outputs, analogous to the pins on a chip. Adding inputs is easy -- declare the class as an endpoint...

Overlay includes ‘: public async_endpoint’

NARRATOR (V.O.)
Inputs are just ordinary member functions.

INPUT arrow appears to the left of the class, pointing at the ‘void input(int x)’ line.

NARRATOR (V.O.)
You can have as many of them as you like. If you name one ‘input’...

‘input’ HIGHLIGHTS.

NARRATOR (V.O.)
... it becomes the default input for the class. Outputs are declared by adding tubes.

Overlay includes ‘tube1<int> output;’. OUTPUT arrow appears on the right hand side of the class.

NARRATOR (V.O.)
As with inputs, you can have as many outputs as you like, but the default output is always called output.

‘output’ HIGHLIGHTS.

(CONTINUED)

Code zooms into the distance. A main class moves in on the left: a tube is declared as ‘tube1<int> numbers’, and a loop sends a series of numbers to it.

NARRATOR (V.O.)
So how do we use our squarer class? It’s actually pretty simple. First we’re going to want to see what’s coming out of it:

A class appears that implements printing moves in from the right.

NARRATOR (V.O.)
All we need to do now is wire everything up.

The single line of code ‘numbers >> squarer >> printer;’ appears. The classes snap together, outputs connected to inputs. We see the program’s output scroll by as an OVERLAY.

FADE TO STL based code example, stripping the loop that generates numbers out and replacing it with an STL collection. Relevant parts of the code HIGHLIGHT as necessary.

NARRATOR (V.O.)
Tubes are compatible with the Standard Template Library -- they are function objects that work with many of the STL algorithms. Here, we’ve replaced the loop with some code that puts some numbers into an STL set. Sending those numbers down a tube just takes one line of code.

HIGHLIGHT the line ‘for_each(tosend.start(), tosend.end(), numbers);’

NARRATOR (V.O.)
We’re able to use the existing STL for_each function. And that’s it, it’s really that simple! It is fair to mention that some of what we’ve seen so far could have been done with signals and slots, though our syntax may arguably be a bit nicer.

PAN TO squarer class.

(CONTINUED)

NARRATOR (V.O.)
Signals and slots are actually a special case with respect to Tubes. Tubes handle the single-threaded case just fine...

Class rotates in 3D, then is duplicated several times in parallel.

NARRATOR (V.O.)
... but extend this to also support the most common kinds of concurrency. Let’s say our processing task is a bit tougher than squaring a number. Something like calculating a Mandelbrot set.

We now see the Mandelbrot generator class from the tubes example code.

NARRATOR (V.O.)
Generating images of the Mandelbrot set is computationally expensive, and also tricky to tune for performance well because execution time varies widely and unpredictably.

Show a Mandelbrot set being calculated (and displayed) in 4 threads, without special care being taken to load balance the threads. A large countdown appears in the upper right making the number of running threads clear -- it becomes obvious that the approach is not efficiently using all processor cores.

NARRATOR (V.O.)
A common approach is to split the image up and allocate each part to separate threads. When execution time is unpredictable, it can be difficult to load balance multiple threads of execution, which makes it harder to exploit all available processor power.

FADE TO code fragment looping across 2048 scan lines sending them in sequence down a tube (declared ‘tube1<work> output’).

NARRATOR (V.O.)
Normally, writing the code for this kind of thing can be pretty tricky.
(MORE)

(CONTINUED)

CONTINUED: (3)

NARRATOR (V.O.) (CONT'D)
Tubes has some tools that make this much easier. We’ve already seen tubes used for single-threaded code.

HIGHLIGHT the tube declaration and the ‘output(todo);’ lines.

NARRATOR (V.O.)
This will work fine...

Show a Mandelbrot being calculated in linear order left to right -- it runs visibly slowly.

NARRATOR (V.O.)
...but it will only use one processor core. If we change one line of code...

The ‘tubes1<int>’ becomes ‘spawn1<int>’ and HIGHLIGHTS briefly.

NARRATOR (V.O.)
The spawn tube, rather than sending messages in the current thread, spawns a new thread for each message. The destination then receives those messages in parallel.

Show the Mandelbrot calculating in parallel.

NARRATOR (V.O.)
So with a single-line change, we have gone from a single-threaded implementation to a multi-threaded alternative. This works well, but most operating systems don’t respond well to having code spawn thousands of concurrent threads. Tubes has a second option that covers this case.

We see the ‘spawn1<int>’ change to ‘queue1<int>’, which HIGHLIGHTS.

NARRATOR (V.O.)
A queue is so-called because it behaves like a queue -- messages can be sent to it immediately, without blocking. They are queued, and a pool of threads send them on in the background.
(MORE)

(CONTINUED)

CONTINUED: (4)

NARRATOR (V.O.) (CONT'D)
By default, a queue tube creates a thread pool with the same number of threads as the machine has cores.

Show the Mandelbrot set being evaluated rapidly left to right.

NARRATOR (V.O.)
Thread pools typically provide the best performance for most parallelizable problems because they avoid swamping the operating system with too many threads, but still allow all cores to be fully utilized.

CUT TO:

WHITEBOARD STOP MOTION ANIMATION

Show a hand-drawn on whiteboard performance graph that compares spawned threads with the thread pool approach.

NARRATOR (V.O.)
Traditionally, implementing thread pools has been very tricky to get right, but as this example shows, Tubes makes this near trivial.

A whiteboard eraser wipes the screen clear.

BACK TO:

MOTION GRAPHICS SEQUENCE

Show the Mandelbrot generator class

NARRATOR (V.O.)
Another common issue in concurrent programs is synchronizing access to resources that are being used concurrently by many threads at once. Tubes makes this very easy. Let’s say that we wanted our Mandelbrot generator class to log progress information as it proceeds.

HIGHLIGHT the ‘tube1<string> logmessage;’ and ‘logmessage(”Starting processing”);’ lines.

(CONTINUED)

PAN TO logger class.

NARRATOR (V.O.)
Let’s implement a class that logs output to the console, whilst enforcing strict synchronization. That is, only one log message may be sent at a time, even though many threads may attempt to do so at once. By specifying that the class is a synchronous endpoint...

HIGHLIGHT ‘: public sync_endpoint’

NARRATOR (V.O.)
...Tubes automatically synchronizes all incoming messages with a mutex. So we just wire it up...

SHOW OVERLAY ‘mandelbrot mb; logger lgr; mb >> lgr;’.

Mandelbrot class MOVES IN and CONNECTS to the logger class.

NARRATOR (V.O.)
...and we’re done.

OVERLAY of program running showing all the start/end messages flying past.

CUT TO:

INT. CUBE FARM - DAY

Programmer looks satisfied.

OVERLAY: Bullet points appear over the image. The programmer looks up as they appear.

NARRATOR (V.O.)
So, to sum up, Tubes builds on the code reuse advantages of signals and slots, whilst also providing very powerful concurrency constructs that are easy to use and fully Standard Template Library compliant. The prototype implementation supports all major operating systems.

(CONTINUED)

CONTINUED:

The programmer reaches over and grabs one of the bullet points, looks at it briefly then tosses it over their shoulder into the trash can, which causes the CUBE WALL to CRASH to the ground, revealing an outdoor horizon.

FADE OUT.