1 Live weaving

Live coding is the exploratory practice of changing code while it runs, using a programming language as a live interface to a running process, and therefore its inputs and outputs. The prototypical application of live coding in the performing arts, such as music, video or choreography. Live coding may be used to refer to any use of code in any kind of live situation, but in the present paper we focus on its classic use in live improvisation, where code is written without a fixed aim, often from a notional 'blank slate'.

Weaving is a textile craft, where parallel threads known as the *warp* are held under tension, allowing a second set of threads (known as the *weft*) to passed over and under them to create a fabric.

2 Setting aside the Jacquard machine

The Jacquard machine is a well known device for intricate control of loom warp lifts, classically using punch cards. Across both computer science and popular culture, the Jacquard machine is often invoked as part of the origin story of computation, particularly as Babbage mentions it as an influence. However the Jacquard does not *do* computation, it is merely an input mechanism. The Jacquard machine therefore brings a fundamental misunderstanding to the topic of weaving and computation which is very difficult to work around.

Yes, weaving is computational, and yes, the Jacquard machine allowed abstract data to be fed into that computation. But the same computational nature is present in all weaving, including traditions of hand weaving developed over millennia. The computation was already there before Jacquard, and the his device only takes humans further away from that computation. So while Jacquard's machine is often given as an example of the beginning of the relationship between weaving and computing the opposite is true - it was an end.

So lets try to wipe the Jacquard machine from our minds in the following discussion. Once we do that, we are able to see that the such machinery did not introduce any computation to weaving - the computation was there already. As will become evident in this paper, the computation isn't in the machine, but in the weave.

3 Introducing the live loom

Having set one mechanism aside, I introduce another. The Live Loom is a warp-weighted loom, with solenoids attached so that warp threads may be individually picked from software. First I explain the technology of the warp-weighted loom, and later explain the electro-mechanical attachments.

We can say that primary purpose of any loom is to hold a group of parallel threads,the *warp*, in parallel and under tension, allowing *weft* threads to be woven up and down between the weft threads. The warp-weighted loom is an ancient technology, where tension comes from the effects of gravity, in particular by hanging weights at the bottom of warp threads. This is counter to what is seen on modern looms, where warp threads is generally horizontal, and held in tension through mechanical means. The core components of a loom are therefore very simple, consisting of a frame holding two horizontal bars in place, one to hang the warp from, another further down to help keep the threads in order, and potentially create a 'natural' default shed for the weft to pass through.

The weaving process involves a weft thread going over and under the warp threads, following one of a large possibilities of patterns, creating for example tabby, twill or satin structures. In practice selective warp threads are pulled forward, creating a 'shed' - a gap between the pulled and non-pulled warp threads - through which the weft travels in a straight line. When the warp threads are returned, the weft is trapped inside, and the next shed is prepared.

4 Weaving technology

Although you could say that the live loom has a contemporary 'maker' aesthetic, due to its lasercut plywood construction, at its core, it is a handloom following ancient warp-weighted loom design. The additional electro-mechanical

parts do not replace the core function of the warp-weighted loom, but rather augment it in order to allow threads to be selected using a computer language as well as directly by hand.

The warp-weighted loom has a simple structure of a frame with hanging threads, but it is important to bear in mind that to some extent, the simpler the loom, the more possibilities there are to weave on it. The computational complexity of weaving comes not from the loom, but from the processes of weaving on it.

4.1 Solenoid actuators

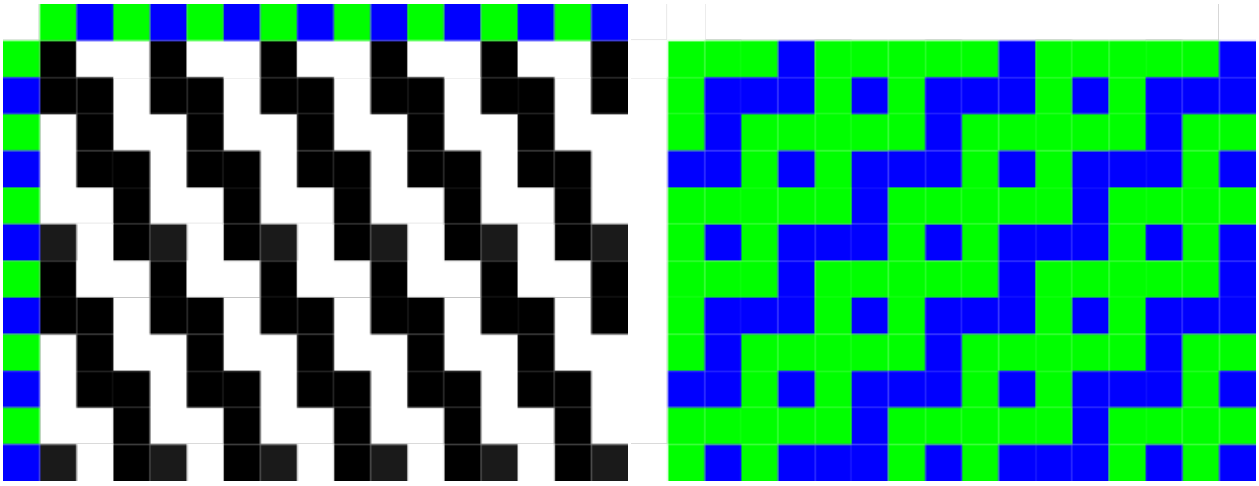
The live loom is fitted with a number of solenoids (currently sixteen), mounted on two axes to double the number that could otherwise fit on the loom. The solenoids are controlled by an arduino microcontroller, via a bank of relays. When activated, each relay will push against a stick, which pulls its corresponding warp thread forward via a string heddle. In this paper we refer to the wooden stick and string heddle collectively as the *heddle*.

Crucially, the solenoid movement is *not* designed to create the shed (gap between two layers of pulled and not-pulled warp threads) through which the weft thread passes. Instead this movement is only to 'offer up' warp threads to the human weaver-coder, who then pulls the threads further with their hands. This seems like a deficiency of power and leverage, but is not; this 'offering up' means the weaver can choose whether or not to pull each thread. This is particularly useful at the edges of the weave, where adjustments are often required to produce a good fabric. This reminds of the live coding choreographic work of Sicchio (xxx); in a similar way, we are not directly live coding a textile weave, but movements of the body in producing that weave. It is humane to respect the ability of people to exercise creativity and agency in the way they interpret instructions given to them.

5 Computing a weave

Before introducing a language for live coding the loom, lets first look closer at the computational nature of weaving itself, focussing on colour and weave effects. Such effects bring together different dimensions or systems. Firstly, the structure of the weave - the arrangement of ups and downs in the grid created by the meeting points of warp and weft. Secondly, the colour patternings of warp and weft threads. The visible colour at a particular point of the weave then depends on two things - whether the warp or weft thread is visible (i.e. whether the weft is under or over the warp) and what colour that thread is. The result is an interference pattern between these two systems, creating a deterministic, logical outcome that is nonetheless very difficult for the layperson to predict.

As a simple example of this, consider the weave structure shown in Fig. 1(a), known as a draft pattern. The central black-and-white grid shows the pattern of weft ups and downs represented as white and black squares respectively. For example, the first row shows that a weft thread going under one warp, over two warps, and repeat. The second row shows a weft thread going under two warps, over one warp, and then repeating.



(a) Draft pattern (b) Result

Figure 1: A draft pattern and the visual result of virtually weaving it.

There are also coloured squares at the top and bottom, showing the pattern of warp and weft colours respectively, in this case both alternating between green and blue. In order to find what colour will be visible, we look at the weave structure. For black squares, we know the warp colour is shown, so we follow the column up to find its colour, otherwise we follow the row the left. [1]

If we plot out the result of this interference between thread colour and weave structure, we arrive at the image shown in Fig. 1(b). This result will be surprising to a layperson, not only is the vertical and horizontal stripe of warp and weft not visible, but the diagonal runs in a completely different direction to the underlying weave structure. This experience will be familiar to those who have explored algorithmic interference patterns in livecoding software such as TidalCycles or Hydra, simple inputs often create unexpected, more unexpected results.

Finally, Fig. 2 shows a fabric woven using this structure and alternating white and blue threads, created on the live loom.

The same features hold in the weave itself, but not too well defined, due to interaction between the threads, and variation in density. The left and right edges are a mess, because in practice such a structure simply cannot be woven at the edges. Weft threads generally travel from left to right for one row, and from right to left on the next. Therefore, if a weft ends a row over a warp, and begins the next row also over a warp, then it will not be woven at that point. An experienced weaver will make consistent changes at the selvage, such as adding some rows of plain weave, to ensure a coherent result.

6 Origins of computation

The above description of colour and weave effects should give us pause for thought. Weaving predates computer programming and indeed discrete mathematics in general, but nonetheless is a computational system, of which any handloom affords exploration. When considering the computational nature of weaving then, we must be careful not to be dazzled by the machinery or electronics of industrial and contemporary weaving technology, when it is the ancient technology of the threads themselves that provides the environment for computation.

7 Coding the draft

We have already seen that weaving drafts are a form of code, which can compute unexpected results when interpreted. Such weaving drafts are themselves binary, digital images, developed well before electronic digital computers. It is therefore possible to add a further level of abstraction by using a programming language to create a draft. The purpose of doing so is to create patterns from patterns, making a rich space to explore weaving and gain tacit knowledge both about how it works and its relation to computation as it is more conventionally understood in the context of programming languages.

Each layer of abstraction takes us further away from the material, but just as live coding of music brings together the experience of coding and listening, the live loom brings together coding with seeing and touching.

Fig xxx. shows the current version of the live loom coding interface. The code is shown on the left, using the visual live coding Texture (xxx), originally designed as an exploratory interface for TidalCycles but here repurposed for a simpler system designed for discrete, binary draft patterns. The set of available symbols and keywords on the top right are, which may be dragged into the code using a mouse. On the bottom right a window into the draft pattern is shown, with the row most recently being sent to the live loom marked with blue squares on either side. Finally

[1] In practice, there are other variables which change which colour thread is visible, for example if weft threads are tightly packed, warp threads are hidden completely.

the row number is written below (which in this case is higher than the number of rows shown, as the previous rows have scrolled off the top). The weave-coder can manipulate the code with a mouse, while using arrow keys on a keyboard to step forwards (or backwards) through the draft, sending each warp lift to the loom to be actuated by the solenoids and woven by the weaver.

Perhaps most notable about this software is what is *not* shown in the interface. In particular, the simulation shown earlier could easily be included, but is not, indeed thread colour is not dealt with at all in the software, only on the loom. Keeping colour on the loom takes focus away from any simulation on screen and onto the 'ground truth' of the material. After all, colour is only one quality of thread, alongside thickness, material, ply, tightness and direction of twist, tension and density of warp and weft, and so on. Trying to simulate all of these continuous variables on-screen is hardly possible, while focussing the software on the singular task of planning the discrete structure of ups and downs works very well.

7.1 Live loom language

The language currently used by the Live loom is the pure functional programming language Haskell, based around its list datatype. Standard Haskell lists are lazily evaluated, which means that infinitely long lists can be represented, by being calculated on demand. A weave structure is simply represented by a one dimensional list of Boolean values, where *true* and *false* stands for *up* and *down*.

The range of functions for composing draft patterns on the live loom is small but already provides very rich space of possibility. The weave-coder begins with a list of ups and downs, then applies functions to transform that list and/or combine it with other lists. The following table describes the functions and operators available in the live loom code interface.

name	description
cycle	Repeats a list forever (standard Haskell function)
backforth	Reverses every other row
offset n	Offsets each row from the last, by the given number of threads
shift	Shifts each row by one thread
rev	Reverses each rows
every n f	Selectively applies function f to every nth row
invert	Turns all ups to downs, and vice versa
zipAnd a b	Combines two lists, resulting in up when both lists have an up
zipOr a b	Combines two lists, resulting in up when one or both lists has an up
zipXOr a b	Combines two lists, resulting in up when only one lists have an up

The result is a language interface that produces surprisingly complex results from simple elements.

8 Working at the Live Loom

The above figure shows the live loom software interface next to the woven outcome. This starkly shows the perceptual gap between code, draft and weave, with little visual correspondance despite the structures of the weave and draft being constructed by their leftmost neighbour. The code is represented as a branching tree (due to the visual interface directly showing the branching normally represented by parenthesis). This creates a pattern which perhaps has the appearance of vines growing up a wall. When this structure interferes with the alternating colours of warp and weft, the final result appears as (to my eyes) legs leaping into the air.

It is humbling that the logical leap from draft to weave is ancient knowledge, demonstrating mathematical logic while predating our conventional view of mathematics.

8.1 Music of the loom

The solenoids are not triggered at once but in sequence, to even out the use of electrical power, as less ampage is required to hold a solenoid than to move it (xxx is this true in practice?). The most time-efficient way to do this would be to trigger the 'up' threads, pulling the warps one after the other, evenly spaced in time. However I have found it much more useful to include 'down' threads in the timing, so each row takes the same amount of time to actuate, no matter how many warps are being pulled forward. This gives a clear rhythm to each row, where the 'clunk' of a solenoid are heard for ups, and a 'rest' or gap are heard for downs. This rhythm breathes life into the weaving process, making it easier to orient myself in the pattern and spot errors, as I compare the rhythm I hear with the threads that I see. It also brings rhythmic enjoyment to repetetive nature of weaving.

The solenoids have a particular 'duty cycle', meaning that it is best not to keep them activated for too long, otherwise they may overheat. Once all the solenoids are activated, the microcontroller holds them in place for two seconds, plenty of time for the weaver to place a hand on the heddles and pull the selected warp threads forward. Although born from a technical need, these two seconds add an additional sense of regulated timing to the process of weaving. However if the heddles are not caught in time, the weave-coder can repeat the lift with a quick press of the up arrow.

9 Live Coding

So far we have discussed action, but not live reaction. We have looked at coding the loom with a draft, and coding the coding of the loom by introducing a language for composing the draft, but we haven't discussed live coding - the changing of code in response. Let's do that now.

9.1 Changing patterns

Live coding of music is often characterised by slow, continuous changes. Changes are heard immediately, but the complexity of music grows with the code. The experience of the live loom is rather different, where a small change tends to have a large, global effect, but each change takes time to become apparent, rows are only produced at a rate of a few per minute, and it might take two or three repeats of a pattern before its nature can really be perceived. These big differences from small edits are due to the multiple levels of interference, between code, draft and weave.

A change from one pattern to the next also presents a problem of transition, where one pattern does not sit well with the next, potentially creating a physically uneven structure, with undesireable floats (see below). It can take a disturbed row or two before the weave settles into the next structure. There are certainly parallels here with live coding music and indeed music in general, where a sudden change can be jarring, without a careful transition. Managing this transition is probably best done at the loom, adjusting the lift at the heddles by hand.

At this slow pace of change though, we are in the domain analogous to slow coding rather than frenzy of an algorave, each decision can be considered carefully. Furthermore in weaving we work with physical thread, rather than with the metaphorical thread of time as with live coding of music. This means that we are able to undo a weave in a way that we cannot undo music, and change our minds. By unweaving, the weaver, like the mythological figure of Penelope, resists external forces.

9.2 Embracing error

Live coders are known for embracing error, and so it is fortunate that it is so easy to produce a draft which is unweavable. For example, there is the problem of 'floats', lengths of unwoven fibre created wherever there is a contiguous series of either ups or downs in the warp or weft direction. If there are only either ups or downs in any row or column, that thread will not be woven into the fabric. In response to a problematic draft the weaver can do one of three things - change the code to look for a more weavable draft, ignore activated heddles or pull additional ones to change the weave directly, or just attempt to weave the pattern anyway.

In the above example, the draft looked unweavable to my naive eyes, due to the pairs of identical rows within it. Where this happens, essentially two wefts are passed through the same shed. I thought this would result in a mess, but out of curiosity went ahead anyway, and found that with care the wefts would still run parallel and stay in order, maintaining the 'correct' pattern. Furthermore, because the repeat in the draft consists of an odd number of rows, and I was weaving with two different wefts, the wefts would alternate from one repeat to the next travelling from one side to the other together, and travelling in opposite directions. A surprising, pleasing, and subtle result.

10 Conclusion

This paper has explored how the principles of live coding may apply to the warp-weighted loom. However, in connecting a live coding pattern language to the practice of weaving, we find that weaving is already abundant with computational patterns, and in particular that historical weaving draft techniques already demonstrate a similar computational abstraction from the resulting woven textile as code does from media in the live coded performing arts. Nonetheless by adding another layer of abstraction to that which has been present in weaving since ancient times, and using solenoids in communicating movement from the code to the weaver, the live loom allows creative exploration of woven patterns in a way that is sympathetic to the repetitive, yet cognitive nature of handweaving.