Algorithmic Patterns on the Live Loom

Alex McLean

Algorithmic pattern is an expansive practice running across the arts and crafts, that is ancient, yet core to the advance of contemporary creative technology[1]. In this chapter I introduce the Live Loom as a case study in algorithmic pattern, bringing together the ancient weaving technology of warp-weighted looms with the contemporary technology of computer programming languages. The Live Loom is (perhaps literally) a tangle of old and new, and by attempting to untangle its threads and wires from around its physical wooden frame and metaphysical syntax trees, I explore algorithmic pattern as an ancient, technological and developing craft tradition.

I introduce the Live Loom in terms of its many layers of physical qualities starting with the threads themselves, their warping, the crossings of warp and weft threads, then progressing into metaphysical representations of drawdowns, syntax trees and procedures. In so doing, I explain how interactive notations reveal what is normally hidden by notation - a lively world of threads that compels us to reconsider our relationship with technology as craft.

[1] Mclean, ‘Algorithmic Pattern’.

Figure 1: The Live Loom, a hybrid, open hardware, warp-weighted hand loom, with 16 computer-controllable warp threads. Photo credit: James Hendy

Computation in Craft

Ada Lovelace, known as the first computer programmer for her work with Charles Babbage, is often cited for recognising common patterns across weaving and computing. However her foresight carries a misapprehension that has since been carried through the history of computer science. The motivation for creating the Live Loom is to try to expose this misapprehension, by connecting computer programming directly with weaving.

Lovelace shares her thoughts on weaving and computing in her extensive notes as translator of a piece written about Babbage's Analytical Engine: "By the introduction of the system of *backing* into the Jacquard-loom itself, patterns which should possess symmetry, and follow regular laws of any extent, might be woven by means of comparatively few cards."[2] This suggestion is on one hand visionary; following the technology transfer of the Jacquard punch card reader from weaving to computing, Lovelace sees how the computational procedure of looping a subset of cards could be imported back from the Analytical Engine to the Jacquard

[2] Menabrea, *Sketch of the Analytical Engine Invented by Charles Babbage, with Notes by the Translator Ada Lovelace.*

device. However as Ellen Harlizius-Klück has previously argued[3], weaving has *always* been computational, and in not recognising this, Lovelace seems to have mistaken the opportunity to *reintroduce* computation to weaving for the opportunity to introduce it for the first time. This would not only be a misinterpretation of history, but would miss the opportunity to incorporate advanced understanding embedded in traditional weaving practice in the development of new computational machines. Today, we still follow Lovelace's thinking in assuming that the youngest technological practice is the most advanced one, which does not stand to reason, and leads us to continually re-invent a poorer wheel.

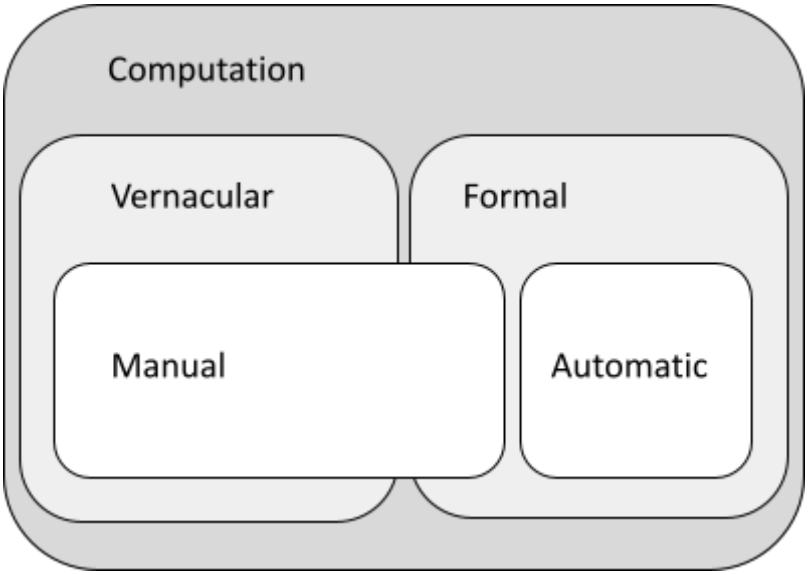


Figure 2: Euler diagram showing the logical relationship between vernacular, formal, manual and automatic forms of computation.

To differentiate the Live Loom from the Jacquard mechanism, the Euler diagram in Fig. 2 shows the logical relationships between different kinds of computational technology. One dichotomy drawn in the diagram is between vernacular and formal computation. Vernacular computation includes the kind often examined in the field of ethnomathematics; ways of working that are passed down by word of mouth. Formal computation is a process which is unambiguously notated in such forms as a computer programmer's code listing, a mathematician's lambda calculus, or a weaver's draft. The other dichotomy shown is between manual and automatic computation. For computation to be automated, it must be formalised, so that it may be followed by an electronic or (in Lovelace's time) a mechanical computer. However, where a computation is manually worked through by a human, this could be done either on a vernacular or formal basis. A human is perfectly capable of following formal instructions (such as a weaver's treadling pattern, a knitting pattern, or recipe) accurately, as

[3] Harlizius-Klück, Ellen. 'Weaving as Binary Art and the Algebra of Patterns'.

well as applying embodied knowledge, accrued through the vernacular - informal instruction or personal exploration. However, since the industrial revolution, formal computation has become closely associated with automation, and so craft practices based on manual exploration of computational, algorithmic patterns have been sidelined. What I try to show in the following, by introducing and using the Live Loom, is the generative, creative possibilities of manual, yet formal systems of computational craft.

Threads

In the present day, we relate the zeroes and ones of binary (base two) counting with modern computing, but we know from the field of ethnomathematics that humans have always engaged with binaries.[4] This is clear even when looking at a single thread, which has two possible spin directions - S (holding a thread vertically, we see the spin travel diagonally top left to bottom right) or Z (top right to bottom left). If you tighten the spin of a thread and hold its two ends together, it will self-ply - creating a shorter 'doubled' yarn with the opposite spin direction - a logical negation.

A thread is typically composed of smaller strands, which are ultimately composed of fibres, the thread itself already being the result of a long process of production. Threads therefore exhibit fractal self-similarity (fibres within strands within yarns, etc) all the way to microscopic properties which still affect the visual and physical qualities of the resulting weave. Nonetheless, as a useful simplification for present purposes, in the following we treat the thread as the atomic fundamental unit of a weave. While threads are marked by discrete[5] binaries such as spin direction, they also flow continuously. Threads are therefore both discrete and continuous, or in other words both digital and analog. In weaving, a loom is a device for focussing on binary interactions, at discrete crossing points, and therefore it is these binary and discrete qualities that we focus on here. Regardless of the presence of electricity or electronics, any loom is fundamentally a device for working with discrete crossing points and weaving is therefore fundamentally a digital craft.

For a detailed examination of the binaries of the threads themselves, we could look to Andean khipu makers, for example the historical practice of record keeping within the former Inka empire, and the earlier, more improvisatory use by the Wari. As far as we understand, khipus integrate spin direction, knot direction and several other binary properties with a decimal

[4] Babbitt, Lyles, and Eglash, 'From Ethnomathematics to Ethnocomputing'.
[5] Something is discrete when it is countable in clear units.

system of knot-tying, in order to record discrete numerical information.[6] Most textile artists deal with threads, but in many cases that is where shared understanding ends. For example knitting, braiding, and weaving are three fundamentally different families of structures, and so knitters, braiders, weavers and indeed khipu makers do not necessarily have much to say to each other. For the present chapter, we focus on weaving structures.

Threads are highly susceptible to disorder - drop a few threads on the floor and they will seem to tangle almost autonomously. The role of a weaver is to impose order on threads, structuring them in order to create structures which hold together as a fabric, starting with the warp, which we examine now.

Warping

A core prerequisite for weaving is to order parallel threads and hold them in place under tension. These are called the *warp* threads, and in weaving diagrams they are generally conceived as running vertically from the top to the bottom of the page, with perpendicular weft threads introduced later, from side to side. These weft threads are interlaced with the warp according to a particular pattern of movement. Setting up a loom with an arrangement of differently coloured warp threads introduces rich creative constraints on what visual patterns can later be woven on that loom, and any decision about how to do so is not taken lightly; the warping process might take weeks, depending on the thread count and complexity of the loom.[7] The Live Loom is designed for experiment and learning, and has a very low thread count of only 16 warps, but still the process of warping the loom takes over an hour.

Warp weights

One of the primary functions of a loom is to hold the warp under tension, using technologies that have developed over millennia. Modern looms tend to create tension between two drums, as the warp is rolled off one drum, and on to another as woven fabric. Careful warping techniques ensure that all the threads are held at the same tension. A much older (indeed ancient) approach is to add *weights* to the warp. This is how tension is created on the Live Loom, by suspending warp threads from the top of a frame, with each thread kept under tension by a weighted bobbin, dangling at the bottom. Each warp is wound around the

[6] There are also ‘narrative quipus’ with a different structure and are understood even less than the numerical ones.

[7] Here we assume the warp threads are visible in the resulting textile, this is not the case in what are called *weft-faced* weaves.

bobbin, so that the length of the resulting fabric is not limited by the size of the loom, but by the length of thread that is wound around each bobbin. As the weaving progresses, it is wound onto the top bar, and more warp is released from the bobbins, so that they are near, but not touching the surface on which the loom is placed.

Warp-weighting is anachronistic technology, but nonetheless has a number of advantages over modern looms. Tension comes only from gravity, which is of course a constant, resulting in an even tension from equal weights. Therefore warp-weighting creates more reliably uniform tension than the modern drums mentioned above. Furthermore, the threads are only attached to the loom at one end, meaning that the warps may easily be rolled up for transport and storage.

The weighted bobbins used on the live loom are technologies borrowed from another ancient textile art - Japanese kumihimo braiding. I was introduced to kumihimo through tuition from braiding expert Makiko Tada, using traditional *Tama* bobbins. In kumihimo, each bobbin must hold itself in place at the end of its thread, while also allowing the braider to quickly and easily release more thread. Traditionally this is done with a slipping hitch, where the thread twists back on itself on the bobbin in such a way that when the braider pulls the thread in the right direction, more thread is released. I use hobbyist plastic bobbins (with metal weights incorporated), but using the same slipping hitch I learned from Tada-Sensei.

Passing the weft

With the warp held under tension from top to bottom, the weft can be introduced, from one side to the other, and back. To create a weave, the weft travels over and under (or in front of/behind) the warp. However, generally it is the *warp* threads which move to create the structure - for example by alternate warp threads being pulled forward. This creates a gap between the warps which have been pulled forward, and those which have been left behind, called the *shed*. The weft is then passed through this shed. On the Live Loom, warp threads can be selected and pulled forward using wooden sticks as levers, each one attached to a single warp thread (using a *string heddle*).

Having passed the weft from left to right, the weaver then pulls forward a new selection of warp threads, creating a new shed. They then pass the weft back through, this time from right

to left.[8] Creating the new shed also has the function of trapping the previous weft in the weave, and physically pulling the shed open will pack that weft into place. The weft can be packed in further using a flat stick known as a *sword*. Each weft is then co-dependent on the weft before and after, so that the overall structure holds together as a textile weave.

Weave

The selection of warp threads results in a particular weave structure, with profound impact on both the behaviour and appearance of the resulting textile. The translation from binary ups and downs, into physical cloth with real-world properties, is fascinating.

Weaving structure can be conceptualised and notated as a binary grid, but this is an abstraction - *weaves are three dimensional*. The depth of this statement can be hard to grasp, partly because we are so used to the image of Jacquard's two dimensional, binary punched cards, as well as the older two-dimensional, binary form seen in weaver's grid-based notations, known as lift plans, drafts and drawdowns. But these two dimensional grids only notate sheds (and e.g. on shaft looms, tie-ups and treadling), and not the real-world complexity of the woven outcome.

One place that the three-dimensionality of weaving becomes clear is at either edge of the fabric, known as the *selvage*. For handweavers, adjustments to the structure must be made here for the weave to hold together consistently. This requires thought and agility that is hardly possible at modern machine looms, which instead simply cut off the edges as they are woven. By contrast handweavers put thought into creating edges which hold together for consistent results, perhaps even integrating a weave with a very different structure such as tablet weaving around its border.[9] Adjustments might also be needed elsewhere, to make sure every thread is properly integrated into the weave, applying methods to avoid unwanted loose threads, known as *floats*.

The difference between notation and weave is particularly stark in doubleweave. When some structures are woven, they result in two (or more!) fabrics, one lying on top of the other. This happens when there are 'binding points' with some previous wefts but not others, causing

[8] Alternatively, they might select and use a different weft, for example one of a different colour.
[9] Harlizius-Klück, 'Weaving as Binary Art and the Algebra of Patterns'.

distinct layers to form, an outcome which can confuse and surprise beginner weavers such as myself. Indeed a well-cited paper on the geometry of weaves misunderstands doubleweave, describing such structures as 'falling apart';[10] but when you actually weave these structures, they do result in well structured textiles, just more than one of them. This demonstrates the danger of trying to understand weaving 'on paper' rather than at the loom. In practice, when a weaver changes from a doubleweave structure to another one, these different layers are able to reintegrate into a single fabric, the doubleweave section creating a pocket.

There is confusion then between the structure that we visualise, and the resulting textile we see and touch. Apart from special cases such as doubleweave, there *is* a direct correspondence between the grid structure on the page, and the up-down structure of the threads in the textile. However, although this structure is present it may be obscured, especially when we vary warp and weft colours. This introduces interferences, where the colour pattern in the warp and weft threads interfere with the pattern of the weave structure imposed on those threads. The result of this interference is a *colour-and-weave effect*, emerging from the different patterns of thread colour and weave structure, as Ellen Harlizius-Klück explains in the first chapter of this collection, and I demonstrate later in the present chapter.

The woven structure not only influences the visual appearance, but also the behaviour of the textile. For example the woven textile's strength, how it drapes, how layered the textile is, and how it reflects light. We know denim for its strength and durability, and satin for its shiny front, dull back and draping characteristics - which are much more properties of the denim and satin weave structures rather than the material that is used. As discussed, weaving structure is generally represented using binary grids, but the Live Loom adds an additional layer in order to generate such grids - a pattern language, which I introduce next.

Pattern language

We have already considered weaving drafts above, as a form of notation commonly seen in weaving crafts and industry. This notation is used while pre-planning a weave that is later put into production, and may also be created when analysing an existing weave, 'reverse engineering' its structure. Modern industrial weaving is a resource- and time-intensive process, that needs careful planning and testing to arrive at an intended product. However, the

[10] Grünbaum and Shephard, 'Isonemal Fabrics'.

Live Loom is not designed to be productive, but to instead support creative exploration and learning at the loom itself, perhaps reflecting the improvisational approaches to developing new fabrics seen in traditional weaving. The Live Loom is too small, with too low a thread-count to produce cloth in useful dimensions, but that is not its aim. It is instead intended as a platform for *live coding* weaves, where decisions are made and remade during the weaving process itself.

Live coding is a movement co-founded by the present author,[11] emerging from the digital arts and particularly the algorithmic music communities around the turn of the millennium. It began as a way of making improvised music, where a practitioner performs by creating and manipulating code, while that code generates sound, during a performance. A live coder's screen is generally projected into the performance space so that the audience can see the code being written while it generates the music that they hear. As the community has developed, other performing arts have taken on and developed live coding techniques, and live coded video and choreographic performances are now common.

As a live programmable loom, the Live Loom resonates with a particular part of live coding culture, namely *slow coding.*[12] Slow coding eschews high-pressured 'algorave' performances where live coders work furiously on their code to keep their audience dancing, instead each edit is considered carefully over extended periods (in sympathy with the 'slow food' movement). Whereas a live coding musician might make an edit every few seconds, a weaver on the Live Loom might change their code once every ten minutes or more.

The Live Loom language is a library of combinators for the pure functional programming language *Haskell*. In general terms this means that it consists of a collection of words and symbols, each of which stands for a way to construct or transform binary up/down patterns, as shown in Table 1. This is a small and straightforward library, building on standard functionality of the Haskell language.

up / down	Keywords representing a warp either being up (pulled forward) or down (left behind) when creating a shed.
[ ]	An empty list.
:	Used to add an `up` or `down` to a list (starting with the empty list

[11] Blackwell et al., *Live Coding*.
[12] Hall, 'Towards a Slow Code Manifesto'.

	above).
cycle	Repeats a list of `up`/`down`s indefinitely.
offset n	Successively ‘shifts’ each row by the given number ‘n’ of warp threads.
shift	Shifts a row by one (e.g. takes the first `up`/`down` in the row and moves it to the end of the row)
every n f	Selectively applies the transformation ‘f’ every ‘n’ rows
backfort h	Reverses every other row. Shorthand for ‘every 2 reverse’.
invert	Turns all the ‘up’s to ‘down’s and vice-versa
zipAnd	Combines two weaves - where a warp remains ‘up’ only where warps on both weaves are ‘up’.
zipOr	Combines two weaves - where a warp remains ‘up’ only where warps on either or both weaves are ‘up’.
zipXOr	Combines two weaves - where a warp remains ‘up’ only where one (but not both) warp are ‘up’.

Table 1. A list of the main functions available in the Live Loom language.

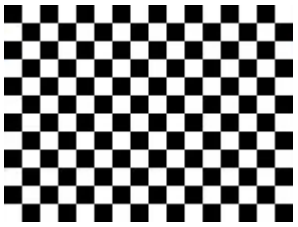
These twelve keywords, along with the natural numbers, are enough to produce an astonishingly wide range of weaving patterns with a very small amount of code. Each one stands either for a unit, or a simple operation on those units - the explosion of possibilities comes from the very large number of ways in which they can be combined into compound operations.

At the Live Loom

Both computing and textile industries have been overridden by automation over the past decades and centuries, respectively. Indeed there would seem to be motivation to develop the Live Loom so that more of it is automated - the shed could be fully opened by the mechanism without hand control, or the weft could be passed automatically, using one of the machine loom techniques developed in the textile industry. However, each step taken towards full automation separates the live coder further from their weave, until they are working only with two dimensional grids rather than three dimensional material. By keeping hands both on

coding and weaving in a physical feedback loop, the weaver-coder has the maximum opportunity to learn. In the following session I step through a weaving session on the Live Loom, recounting the problems met and decisions taken.

I began with a few rows of the simplest possible weave, known as the *plain* weave or *tabby*, to make sure the threads were in good order and the solenoids were working. A tabby weave follows the following lift plan:



Black stands for warp up (and therefore weft down), and white for warp down (and therefore weft up). If we were weaving with paper strips, with black warp and white weft strips, then the woven result would look the same as the above image. One way to express such a tabby weave in the Live Loom language is as follows:

```
backforth (cycle [up, down])
```

This repeats the plain weave's up and down structure, with the *backforth* function added in order to reverse every other row. Without applying this function the textile would not hold together; every weft would follow the same structure, creating no 'binding points' (where weft interlaces with warp and vice-versa) from one weft to the next.

The Live Loom language has a largely mouse-operated programming interface, which is shown on the left hand side of Figure 3. In this interface words are dragged from a palette on the right, and arranged into a program on the left. These words are automatically connected together by the software (based on type compatibility and proximity) into a diagram representing the syntax tree of the program. This allows functions to be moved around the code playfully, to look for interesting interactions with other functions. The code generates the binary grid of a weaving lift plan, which is displayed in the interface, below the word palette. The grey squares on either side of the lift plan highlight the row which was most recently woven.

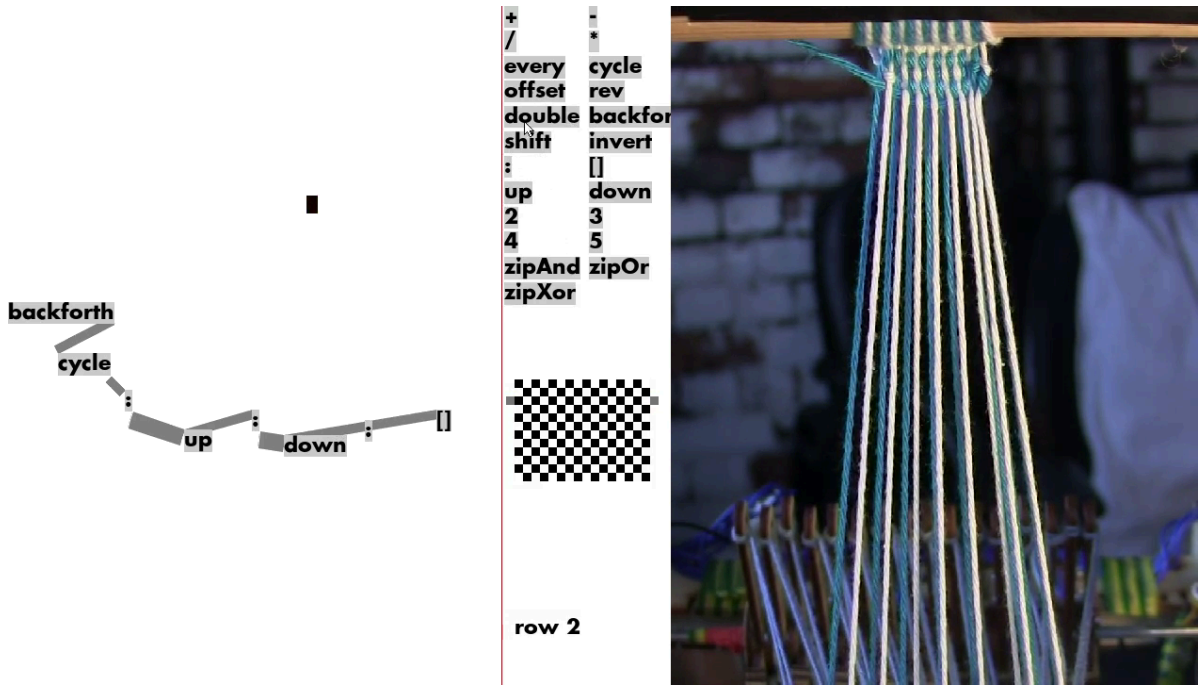


Figure 3: The Live Loom interface, showing code on the left, a menu of available words (functions and values) to its upper right, and the resulting weaving lift plan to its lower right.

A camera feed from the weave itself is shown to the right of the interface. Note that the camera feed shows the back face of the weave, which can look very different from the front. When the weaver-coder is happy with their lift plan, it is time to send it to the loom. The lift plan is sent one shed at a time, by pressing the right arrow key on the computer keyboard. The same shed can be re-sent with the up arrow, and the weaver can send the previous one with the left arrow. In this way, the weaver uses the solenoid-driven heddles on the Live Loom to enact successive rows of the lift plan. This lift plan therefore represents both the binary up/down movements of the solenoids, and the structure of the woven textile that results from these movements. In some cases the structure of the weave is readily visible in the end result. In particular, if the warps are of one colour, and the weft threads of another,

then the front face of the resulting weave will often closely resemble the lift plan pattern, repeated across the fabric. However, as mentioned earlier, if different threads within the warp and/or weft have different colours, then the thread colours interfere with the structure, creating colour-and-weave effects that are surprising to the lay weaver. In the following I have warped the Live Loom with alternating white and blue threads, and also alternate between white and blue wefts. This simple set-up already provides rich ground for exploring colour-and-weave interactions.

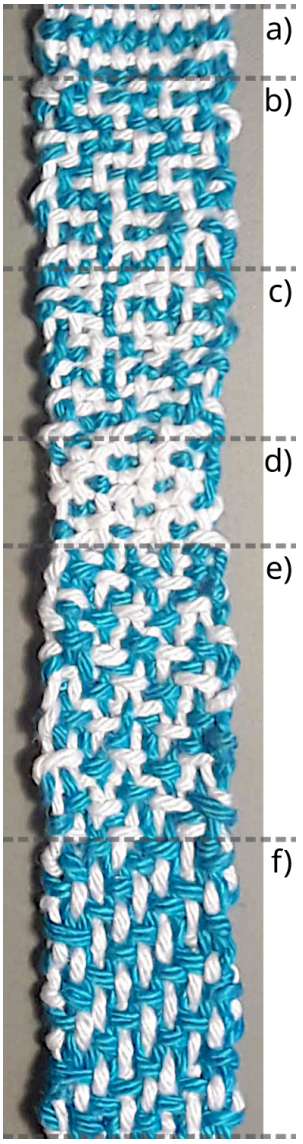


Figure 4. Front face of weave.

In the case of the plain weave shown in Figure 3, we can see that the checkerboard pattern of the lift plan has resulted in vertical stripes in the weave shown on the right. This is because alternating blue/white pattern of the warp and weft threads mean that the white weft is always under the blue warp, and the blue weft is always under the white warp, so the colours of warp and weft always match in the warp direction. However for reasons of practicality the camera here points to the *back* of the weave. If we compare the front and the back side in figures 4 and 5 respectively, we see this section a) appears as horizontal stripes on the front, and vertical stripes on the back. Such effects make the difference clear between a lift plan and a textile. It is often said that "the map is not the territory", and in this case we can view the lift plan as a *map* for the textile as a *territory*. While in this case the map is used to create the territory, we can only really read and fully understand the map once we know the territory. This may seem like a paradox, but a lift plan has at least two purposes, one being as a plan for weaving, and the other as an analytical map for understanding what has then been woven. The textile itself has many properties which are not present in the abstract notation of the lift plan, and which only emerge in its making.

After four rows of plain weave, I change the lift plan by adding a single 'down' instruction to the code, which now reads "backforth (cycle [up, down, down])" (for brevity, I don't include the visual representation of this code here). This produces the following lift plan.

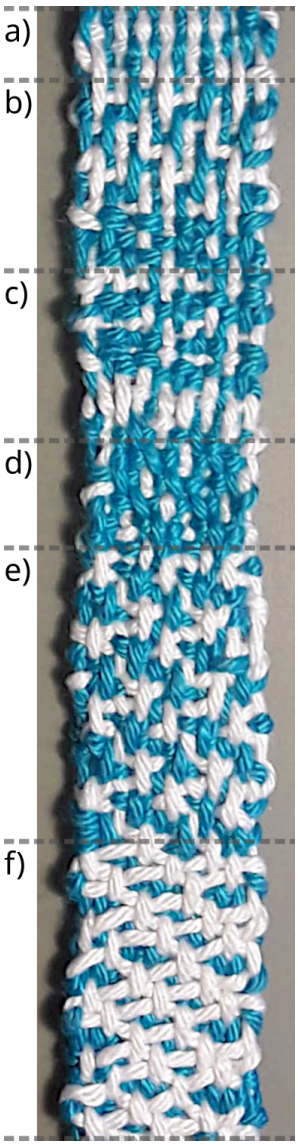
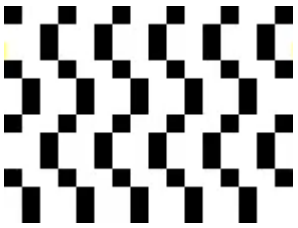


Figure 5. Back face of weave.



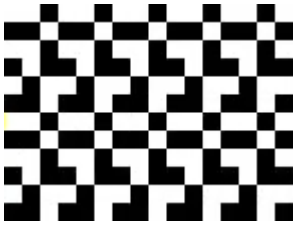
With the structure following a back-and-forth path across the weave, the result is a wave that repeats every three warps and six wefts. Within this repeat, there are consecutive identical sheds, which in practice means that two wefts are passed through the same shed. However I had woven this particular pattern before, and so knew this would not be a problem; these pairs of wefts are held in order by the warps, and the textile holds together well.[13] Looking at the lift plan, we can see unbroken lines formed by four white warps. This creates clear 'floating' warp threads on the back of the weave shown in Fig. 5b, each of which travels over four wefts (if they were shown in black on the lift plan, these floats would have instead been visible on the front face). These floats occur on both odd and even warps, and because I alternate thread colours, therefore both blue and white floats are created. Using these floats as markers in comparing the lift plan with the woven result, we can see how the structure is present in the weave. However squinting our eyes, visually the angular repeating pattern in the weave looks very different from the wave running down the lift plan.

I should point out that while here I use the 'backforth' transformation to construct the lift plan, this does not match with the reality of how I weave it. We could say that 'backforth' simulates the path of a single weft, from left to right, and back again from right to left. But when it comes to weaving, I am using two wefts - I first pass the white and then the blue thread from left to right through the first two sheds, and then back from right to left for the following two sheds. This seems a small technicality but demonstrates that abstractions are at play - the language works with a model of the weaving process as a notional machine. The

[13] Note that if I had been weaving with a single weft, I would have had problems. A single weft would not be fixed at the edges of the fabric between two identical sheds; the first pass of the weft would be undone by the second one.

term *notional machine* is borrowed from computer programming education research[14], and is the idea that when writing software we do not address the computer we are using, but an idealised version of the computer. The properties of this notional machine are implied by the constructs of the programming language and not the hardware. When coding the lift plan I work within such abstractions, but when it comes to weaving the resulting patterns I am free to work beyond the constraints set out in the code. In this case, I introduce additional constraints in the form of colour patterning not represented in the code at all.

For my next edit to the pattern, I introduce the transformation `every 3 shift` to the existing code, so that every third row is offset by one thread in the warp direction[15]. This transformation resulted in a lift plan which I decided would not be interesting or even possible to weave - the floats in the warp direction were too long. Rather than alter this transformation I add an additional transformation on top of it - `every 2 invert`, which creates an interesting-looking lift plan of tessellated, rotated 'L's:

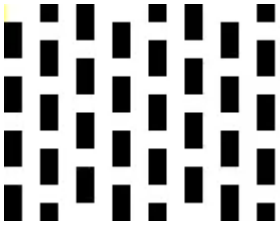


However when it came to following this lift plan, I had unexpected problems. The wefts would not behave in a uniform way and it took some time to begin to understand why. The results shown in Figures 4c and 5c felt like a mess, without clear patterning as seen in the above lift plan. The source of my confoundment was in the pairs of sheds forming each row of tessellated 'L's. Where one row adds 'up' warps, but does *not* add any 'down' warps (or vice-versa), the previous weft is lifted up with (or left behind by) the warps. Indeed, this is how doubleweave is created, where wefts are lifted or left behind in order to weave on separate layers. If I returned to this pattern to weave it again, I would likely be able to produce a more consistent result, having realised that I need to pack one weft behind the other, rather than trying to work them into a clear sequence. Watching the video recording of

[14] https://computinged.wordpress.com/2012/05/24/defining-what-does-it-mean-to-understand-computing/
[15] Looking back at this session, I realise that there was a bug in the 'shift' operation, where the final warp on the right is not shifted. This causes some corruption in the pattern at the selvedge on the right hand side, although this does not impact the rest of the weave.

me working, you can see where I try to pull the shed apart in order to pack the wefts in, but this does not work for doubleweave - they need to be 'beaten in' with a weaver's 'sword'.[16] I continued the fabric the following day, and with a new piece of code, this time combining two patterns into one. In particular combining the repeating cycle 'down, up, up' with the repeating cycle 'up, down', using a logical 'and' operation. The code and the resulting lift plan are shown below.

```
zipAnd (cycle [up, down]) (cycle [down, up, up])
```



Because these repeating sequences have differing lengths of two and three steps, the resulting repeat is the common multiple of six steps. Taking 'up' for true and 'down' for false, we then combine the sequences 'false, true, true, false, true, true' with 'true, false, true, false, true, false'. The logical 'and' returns true only for those steps which are true in both sequences, giving 'false, false, true, false, true, false'. This might seem an arcane way to produce such a short sequence of six binary values, but its usefulness is in its generative nature. Once the code is written, we can quickly modify, add and remove elements to quickly explore a very wide range of possibilities.

Unfortunately, the lift plan resulting from this code is hardly weavable - every other warp is not integrated into the weave, or in other words those warps float completely under the weave. So, I added an `every 2` to the logical 'and' operation `zipAnd`. This means that the 'down up up' sequence is only combined with the 'up down' sequence every other shed, and is otherwise left as a simple three-step sequence:

```
every 2 (zipAnd (cycle [up, down])) (cycle [down, up, up])
```

[16] At the scale of the Live Loom, I use a 'lolly pop' stick for such a sword.

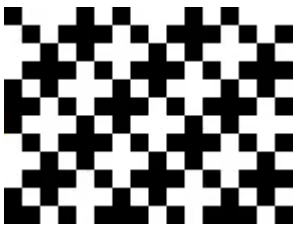
This lift plan has an interesting diagonal twill-like structure, but after weaving 10 wefts with my usual alternating colours I did not feel inspired by the results (see Fig. 4d/5d), so I looked for a more interesting pattern. I settled on adding an additional `rev` instruction, so that the `every 2` now reverses every other row, rather than operating on the logical `zipAnd` operation, which there now applies to *all* the wefts.

```
every 2 rev (zipAnd (cycle [up, down]) (cycle [down, up, up]))
```

This new `every 2 rev` code actually does exactly the same transformation as `backforth`, just expressed in a different way, with different opportunities for being changed. The resulting lift plan resembles a diagonal brickwork-like structure, and although I persist with it for 24 wefts this time, looking for an interesting repeat to emerge, I was still left feeling that the lift plan looked more interesting than the woven results (see Fig. 4e/5e).

Finally, I made one more edit in place of the `every 2 rev`, trying a few different functions and numbers before settling on `every 3 inv`, i.e. swapping the 'ups' and 'downs' for every third weft. The result was a lift plan pattern of tessellated '+' figures.

```
every 2 rev (zipAnd (cycle [up, down]) (cycle [down, up, up]))
```



As with my earlier experience with the unanticipated doubleweave effect, the wefts were again unruly, but this time in a more interesting way. This weave has floats running in both the warp and weft direction. These interact on the back face of the weave seen in Fig. 5f,

resulting in wefts which seem to run diagonally across the textile, defying the rules of the weave (where wefts can only run perpendicular to the warp). This is partly due to the warp occluding the weft, with the visual result that the paths of different wefts are joined through Gestalt perception, creating the illusion of a continuous diagonal path. It was fascinating to see this interaction emerge over 26 wefts. Still, I felt the need to finish this unruly section with four tidier rows of plain weave, which I picked by hand, bypassing the code and solenoids.

Through the process of weaving the above, I move between code, lift plan and weave, building a mental model of the working threads, including doubleweave. From this I am able to adjust my way of working to match the mental model I build, in order to eventually produce better results. This is the process of building tacit knowledge, which (aside from the process of writing this chapter) is not written down, despite growing through interaction with live code.

Digital blindspot

Colour and weave effects are an example of a computational procedure in weaving that arises from the combination of discrete, patterned elements into a more complex whole. Once we recognise this computational nature of weaving, we must also recognise that traditional weaving is a digital artform. This is however at odds with the usual narrative around digital arts. Normally, digital art is described as a recent development, which for example Christiane Paul introduces[17] as a culture of practice that grew in the 1990s, noting that the first digital computer (the ENIAC) was constructed in 1946, with theoretical foundations laid earlier. But this short-term view of digital art breaks down on close examination - digital technology is any which deals with discrete, countable elements. There are many examples of such technologies, from the abacus to the loom, which predate the ENIAC by thousands of years.[18] It seems then that the automation of computation in the 20th century through mechanical and electronic means has produced a deeply problematic blindspot. This blindspot confuses our understanding of digital art and algorithmic patterns in general. Thanks to industrial automation, we now think of computers as being separate from ourselves, whereas beforehand computing was something that humans did; indeed a computer was a job title,

[17] Paul, *Digital Art*.
[18] McLean, Harlizius-Klück, and Griffiths, ‘Digital Art’.

often filled by women.[19] The same blindspot is seen in weaving; much is made of Babbage and Lovelace's references to weaving in the design of their mechanical computer via the industrial Jacquard device, but long before the Jacquard device was attached to looms handweaving was already an computational art, just one performed by humans not machines. Indeed, we could only automate weaving by simplifying it, in the process severing the historical connection between humans and algorithms. This severing is the cause of the blindspot; where historical context is lost, digital art seems like it is new, but is impoverished without its original grounding in craft.

The live loom aims to confront this blind spot by eschewing mechanical automation in order to expose the hands-on creativity of both programming and weaving, and bring them together in a single system for exploring algorithmic patterns. It includes aspects which we more conventionally think of in terms of computing, a programming language with a software user interface. But truly, the handweaving loom interface and process is just as computational as the software interface and process.

Encoding doubleweave

So far I have explored colour-and-weave and doubleweave structures through naive improvisation, which seem magical when first encountered. Before concluding, I would like to have a closer look in order to try to understand what is really going on. Let's start with the following chart, to think about the possibilities offered by alternating colours of weft and warp threads:

	x	**o**	**x**	**o**	**x**	**o**	**x**	**o**
x	X	?	X	?	X	?	X	?
o	?	O	?	O	?	O	?	O
x	X	?	X	?	X	?	X	?
o	?	O	?	O	?	O	?	O
x	X	?	X	?	X	?	X	?
o	?	O	?	O	?	O	?	O
x	X	?	X	?	X	?	X	?
o	?	O	?	O	?	O	?	O

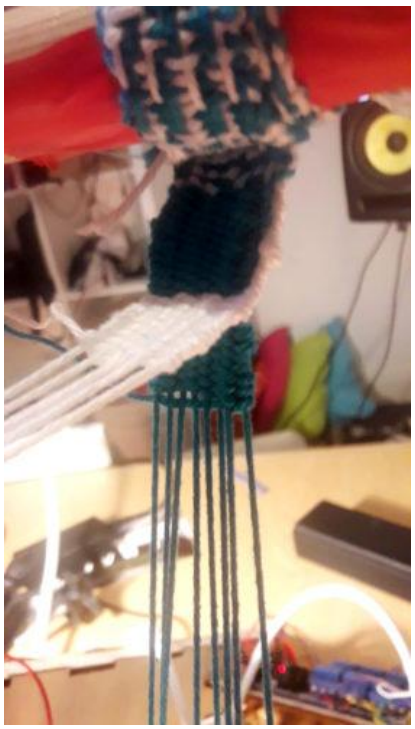
[19] Hicks, Aspray, and Misa, *Programmed Inequality*.

The lowercase `x` and `o` represent the two colours alternating for the warp and weft, and the central grid shows `X` for crossing points which always have colour `x` on top, `O` where colour `o` is always on top, and `?` where it depends whether the weft is over or under. This reveals a clear constraint to the motifs that may be woven using alternating warp and weft colours - revealing a grid of possibly connected points. Looking at the structure, we see a grid of `X`s, diagonally offset from a grid of `O`s. This reminds me also of double-weave, where two layers of plainweave may be produced from a simple two dimensional pattern, one diagonally offset from the other. From this, I realised that a doubleweave structure with alternating thread colours, should result in differently coloured layers.

To test my naive thought, I did some weaving on the Live Loom, using the following code:

```
every 2 invert (offset 1 (intersperse down (cycle [up, down])))
```

That worked pretty well! After weaving that for a while, I wanted to try swapping the two layers. After a bit more thought, I tried simply swapping all the ‘up’s with ‘down’s, by adding an `invert` instruction to the code:



```
invert $ every 2 invert $ offset 1 $ intersperse down $ cycle [up, down]
```

I had some problems with my fabric unweaving, but with some adjustments, this worked very well, with the bottom layer passing perfectly through the other to become the top layer. There are no knots involved here, this is one fabric passing through the other.

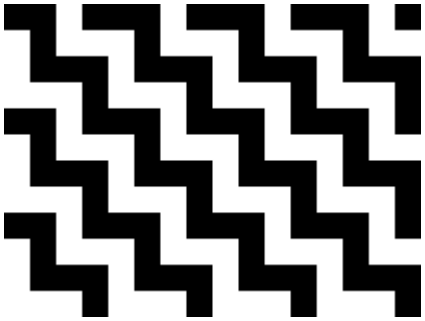
Turning to the literature to try to understand this better, I came across the following from Bauhaus artist Anni Albers:

"Double weaves have a special nimbus about them for reasons not clear to me. They are thought to be intricate, hard to grasp, open only to advanced students. To my mind they are simple to understand and can be handled by anyone with just common sense — which, I admit, sometimes seems rare." – Anni Albers

As someone struggling to comprehend doubleweave, I was a little taken aback by this thought, that a structure that seems somewhat mystical for me, could be considered simple. Perhaps this is the problem with coming at weaving from the naive perspective of computer science. When looking at a binary grid it is too easy to think about weaving in two dimensional terms, and it is only at the loom that the three dimensional interactions of

weaving become simple, through embodied understanding. When dealing with such ancient technologies, much humility is required on the part of computer scientists.

Still, there is something about the code I wrote to make the double weave. Here is the lift plan created by the code:



As a naive weaver, the above structure does not, to my naive eyes, say anything about how weaving it produces two separate layers of fabric. But then I realised that *the code does!* Reading it backwards:

```
every 2 invert (offset 1 (intersperse down (cycle [up, down])))
```

1. `cycle [up, down]` – this is the structure that each layer ends up with – the simple repeated (cycled) up-and-down (also known as over-and-under) steps of the plain weave
2. `intersperse down` – this puts an additional ‘down’ between each step, ‘making room’ for the extra layer
3. `offset 1` – this progressively offsets each row by one step, which provides the diagonal movement in the structure. This does two things – it alternates between the upper and lower layer on the warps, and also creates the single warp offset of the plain weave
4. `every 2 invert` – every other row, this swaps the ups with downs, in effect alternating between the upper and lower layer on the wefts.

I find this really interesting. I wrote this line of code with an entirely practical task in mind – to produce the above binary grid of the weave structure in the clearest way I could think of. In so doing, I've ended up with an abstract, linguistic description of the woven structure, which actually opens a window for understanding the three-dimensional, woven results.

What's also interesting is that now that I have better understood doubleweave, it allows me a new way of *perceiving* colour-and-weave effects. I can now imagine two layers of textile, which interweave to produce the effect. This is not of course one true way of seeing these effects, rather one of many. But still, gaining this embodied, conceptual understanding through weaving, has altered the way I perceive the world, a little bit.

Conclusion

In conclusion, we look back to history. What is the historical precedent of bringing together live coding in the exploration of woven algorithmic patterns? The notion of algorithmic pattern, as a symbolic transformation carried out during the process of making, seems to be firmly rooted in textiles. Indeed knitting patterns routinely contain computational operations such as logical branching and looping, with weaving drawdowns offering alternative approaches to computation based on matrix multiplication and thread-based binary logic. However the extent to which weaving drawdowns and knitting patterns are used as tools of thought is unclear. The more visible purpose of these notations is mass communication (commercial printing and distribution of knitting patterns) or relatedly, mass production (control of industrial looms).

The problem is, that you do not need a notation in order to knit or to weave; experienced craftspeople work through memory, and by 'feel' or tacit knowledge. Indeed ethnomathematics began as a field exploring mathematics in craft cultures that do not have a system of writing. Even if many experienced contemporary knitters and weavers make notes while designing something new, these ad-hoc notations are not for sharing but for thinking with, and so will rarely be preserved alongside the final, singular textile piece. A further reason for not preserving a notation is that in a sense, a textile is its own notation. The structure of some textiles is much more difficult to observe than others, but where it is visible, a craftsperson can follow the threads, and in the context of their craft knowledge,

‘reverse engineer’ the pattern used to make them. So like the live code of the Live Loom, these pattern notations are for thinking and generally not for recording.

So while it is important to recognise that algorithmic pattern builds on ancient technology, we should also recognise what contemporary computers bring to it. By thinking with notation while it is being automatically interpreted by a computer, we are able to work with it as a live meta-material, even while physical material is being produced by that live interpretation. The design of computer programming languages for creative use is about making notations that are formal enough to be interpreted by a computer, yet flexible enough to support human expression. The software industry has been led by the strong commercial motivations of effortless mass production offered by digital media, but what we are concerned with here is not reproducibility, but the less well appreciated affordance of code as a medium for thinking through craft.

Ursula Franklin[20] takes a historical view of technology, drawing a clear distinction between holistic technologies in craft, and prescriptive technologies in large-scale production. A handweaver is able to make decisions as they weave, in response to what they have already woven. On a production line (and Franklin gives historical examples such as ancient vase making, as well as in modern day industry), a culture of compliance is required, as individual contributions must be carefully prescribed in order to fit together in the end. This is the difference between ‘growth’ as a commercial imperative (potentially leading to unregulated ‘overgrowth’ and ultimately environmental destruction), and ‘growth’ as a means to follow an idea to a unique outcome. Algorithms then have a very different part to play in holistic rather than prescriptive technologies. In prescriptive technologies, the breaking down of making processes into formalised tasks is necessary for automation and mass production of quality-assured identical artefacts. In holistic technologies, algorithms instead allow us to respond to artefacts as they emerge, by working with the structures of making.

References

Babbitt, Bill, Dan Lyles, and Ron Eglash. ‘From Ethnomathematics to Ethnocomputing: Indigenous Algorithms in Traditional Context & Contemporary Simulation’. In *Alternative Forms of Knowing (in) Mathematics*, edited by Swapna Mukhopadhyay and Wolff-Michael Roth, 205–19. Rotterdam: SensePublishers, 2012.

[20] Franklin, *The Real World of Technology*.

https://doi.org/10.1007/978-94-6091-921-3_10.
Blackwell, Alan, Emma Cocker, Geoff Cox, Alex McLean, and Thor Magnusson. *Live Coding: A User's Manual*. MIT Press, 2022. https://doi.org/10.5281/zenodo.7383848.
Franklin, Ursula. *The Real World of Technology*. 2nd edition. House of Anansi Press, 1999.
Grünbaum, Branko, and G. C. Shephard. 'Isonemal Fabrics'. *The American Mathematical Monthly* 95, no. 1 (1 January 1988): 5–30. https://doi.org/10.1080/00029890.1988.11971960.
Hall, Tom. 'Towards a Slow Code Manifesto'. Blog. ludions, 1 April 2007. http://www.ludions.com/texts/2007a/.
Harlizius-Klück, Ellen. 'Weaving as Binary Art and the Algebra of Patterns'. *TEXTILE* 15, no. 2 (3 April 2017): 176–97. https://doi.org/10.1080/14759756.2017.1298239.
Hicks, Marie, William Aspray, and Thomas J. Misa. *Programmed Inequality: How Britain Discarded Women Technologists and Lost Its Edge in Computing*. 1st edition. Cambridge, MA: MIT Press, 2017.
Mclean, Alex. 'Algorithmic Pattern'. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, 265--270. Birmingham, UK, 2020. https://zenodo.org/record/4813352.
McLean, Alex, Ellen Harlizius-Klück, and David Griffiths. 'Digital Art: A Long History'. Porto, Portugal, 2018. https://doi.org/10.5281/zenodo.2556604.
Menabrea, Luigi Federico. *Sketch of the Analytical Engine Invented by Charles Babbage, with Notes by the Translator Ada Lovelace.* R. & J. E. Taylor, 1843.
Paul, Christiane. *Digital Art*. 1st edition. London ; New York, N.Y: Thames & Hudson Ltd, 2003.