score #1 sketches

archeology ::: geology
sound imprint ::: artifact
pattern language

figure 1 : weaving

figure 2 : weaving

figure 3 : textile drawing

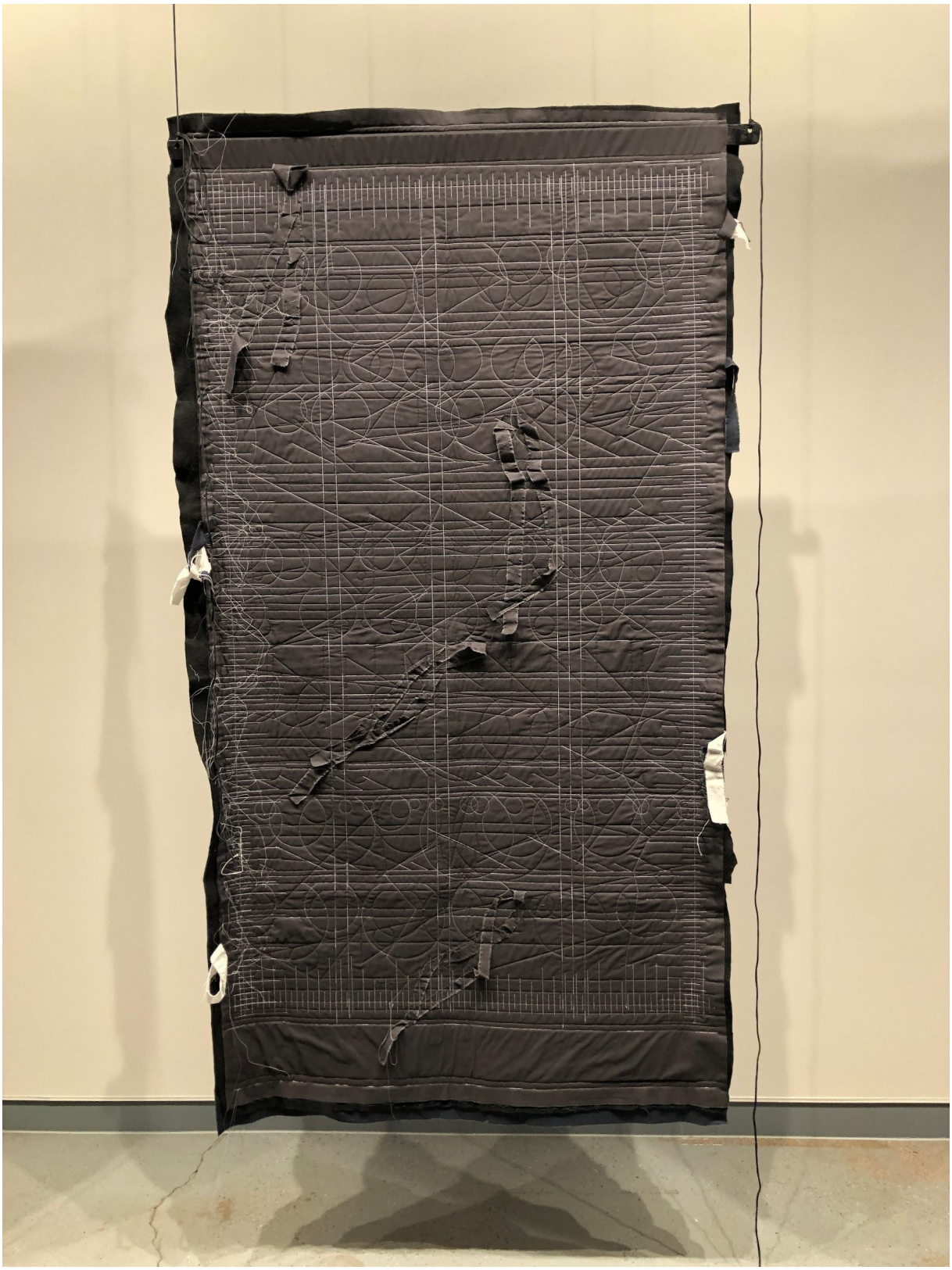


figure 4 : textile drawing

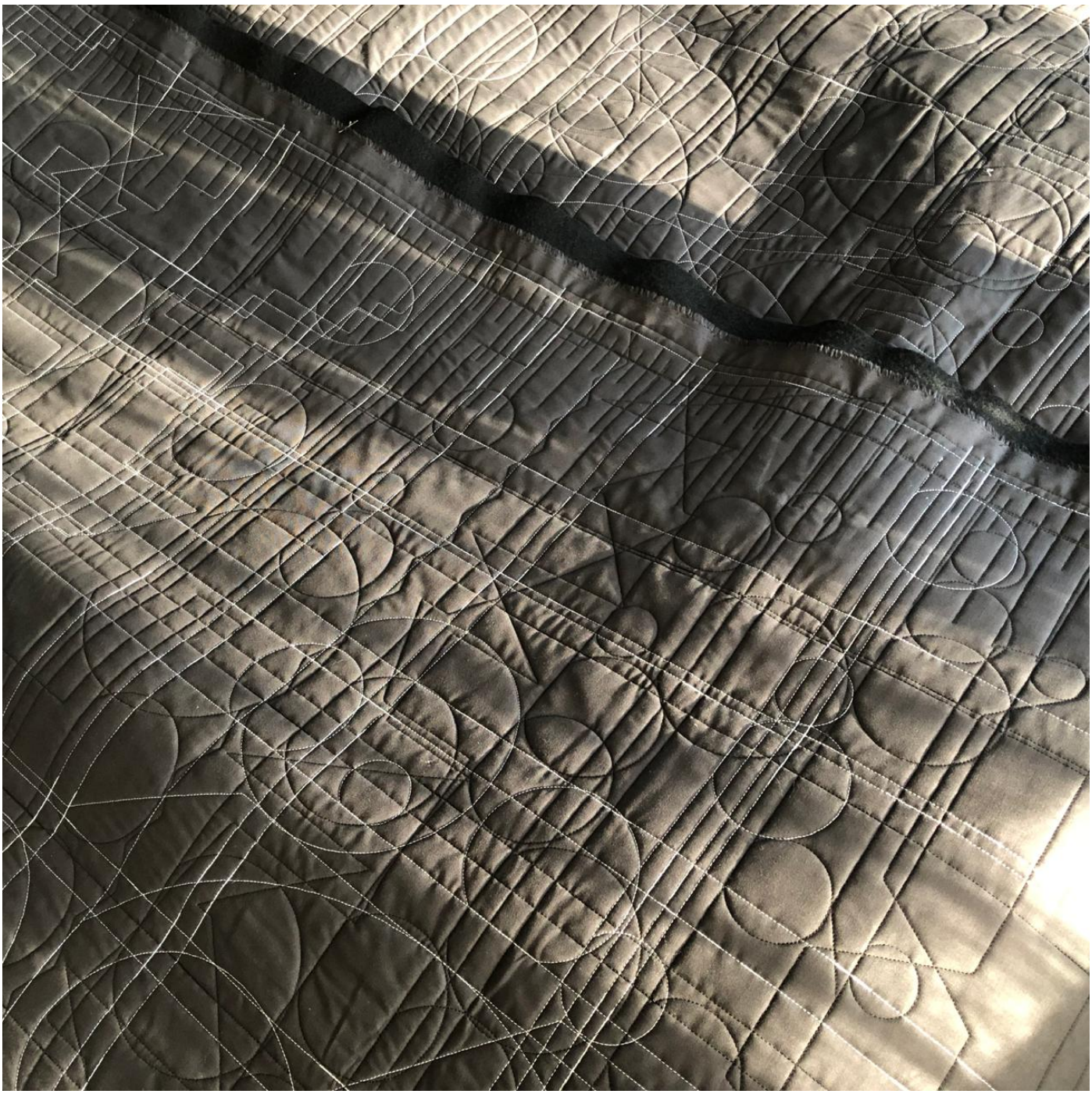


figure 5 : textile drawing

figure 6 : textile drawing

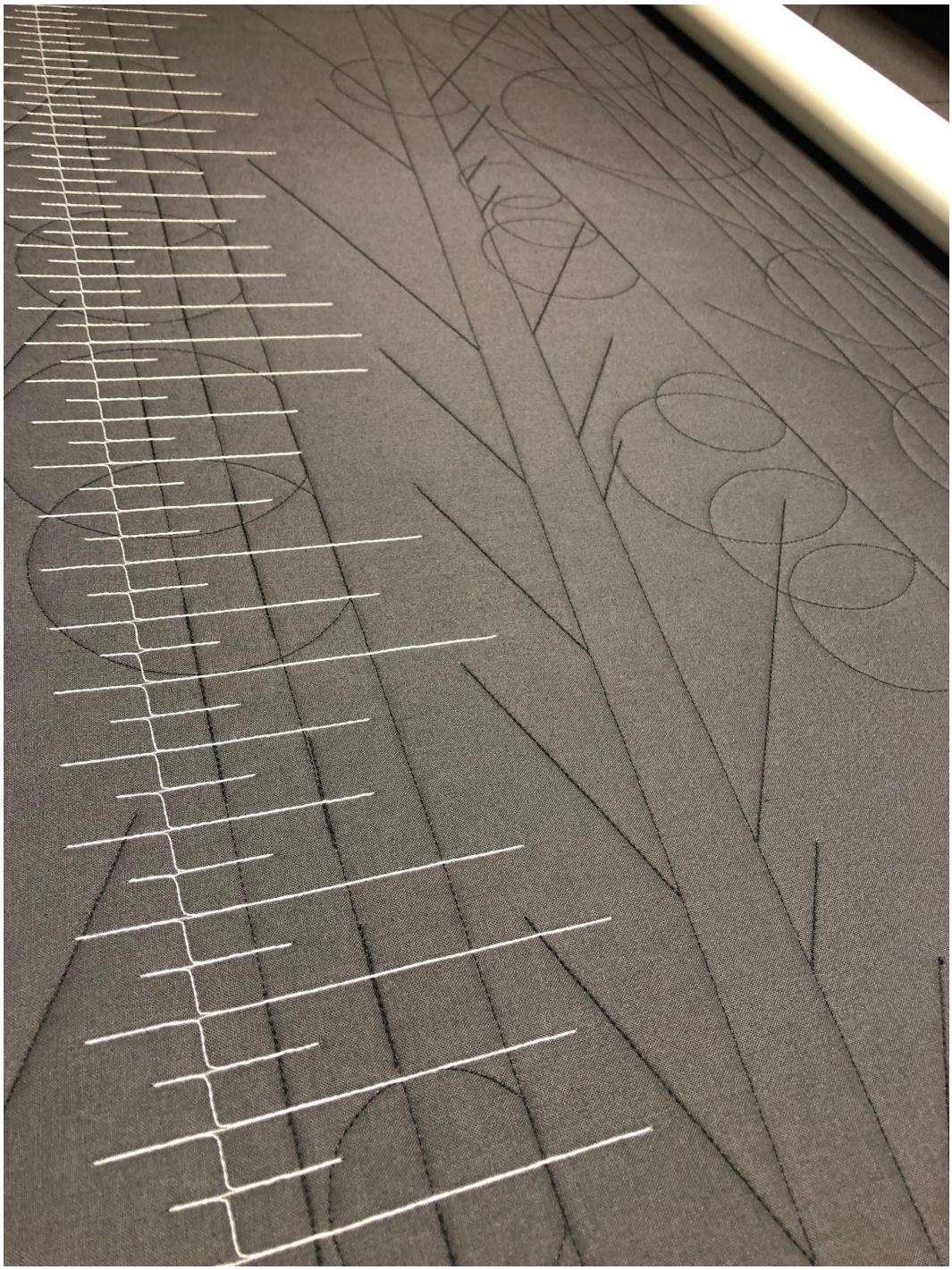


figure 7 : textile drawing

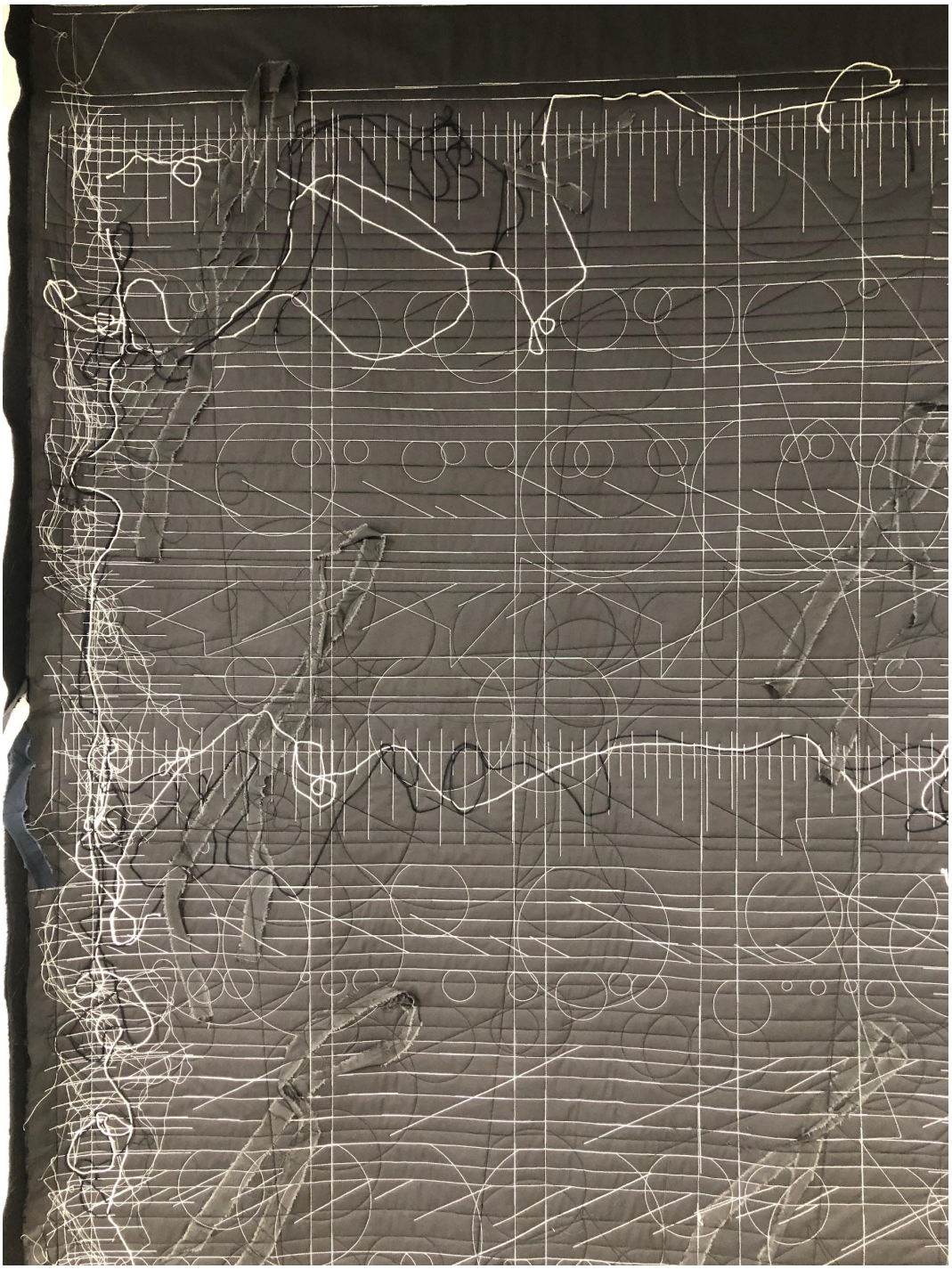


figure 8 : textile drawing

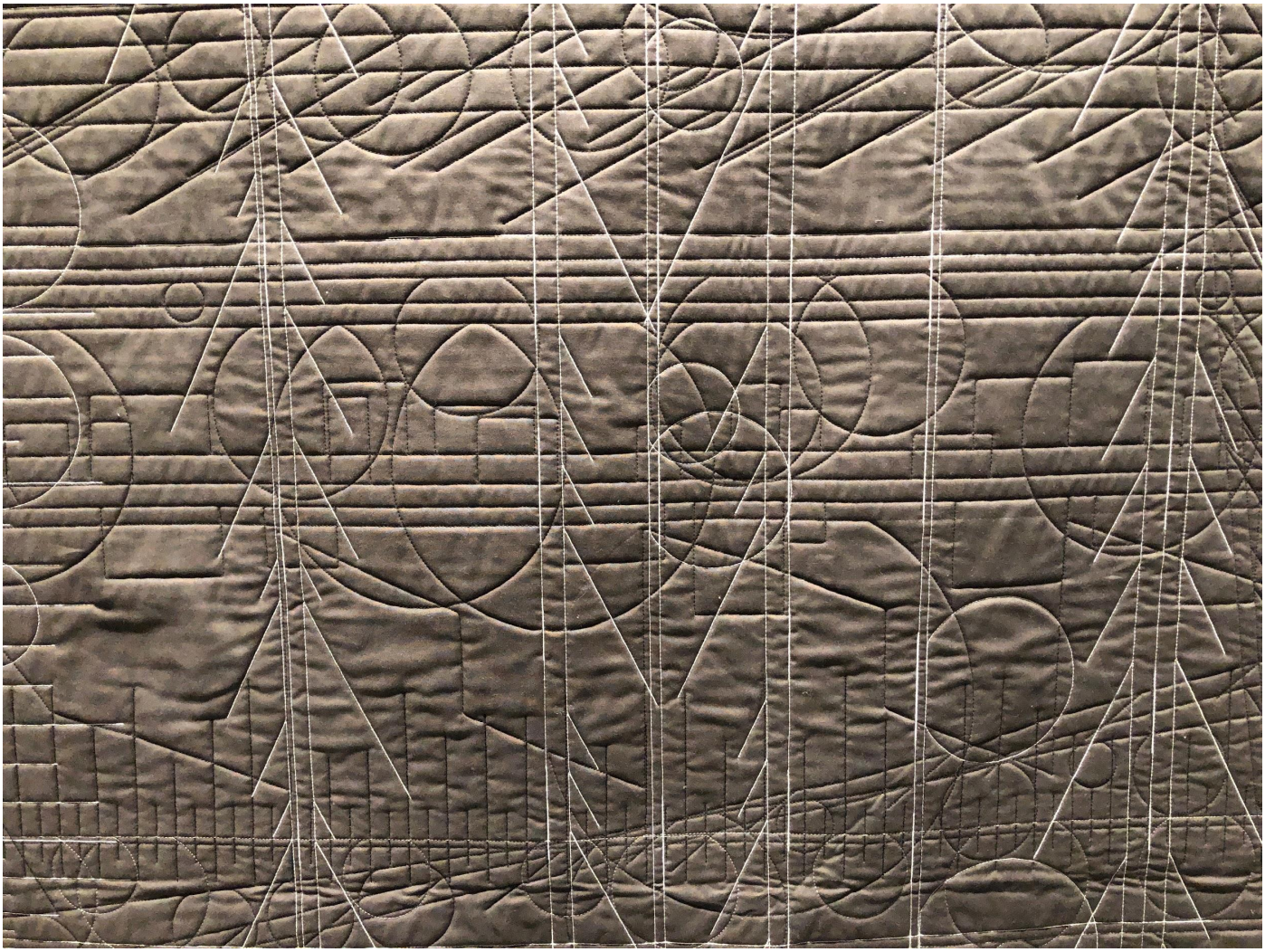


figure 9 : textile drawing

figure 10 : textile drawing

figure 11 : textile drawing

figure 12 : generative books ::: series

figure 13 : computer punchcards

figure 14 : jacquard loom punchcards

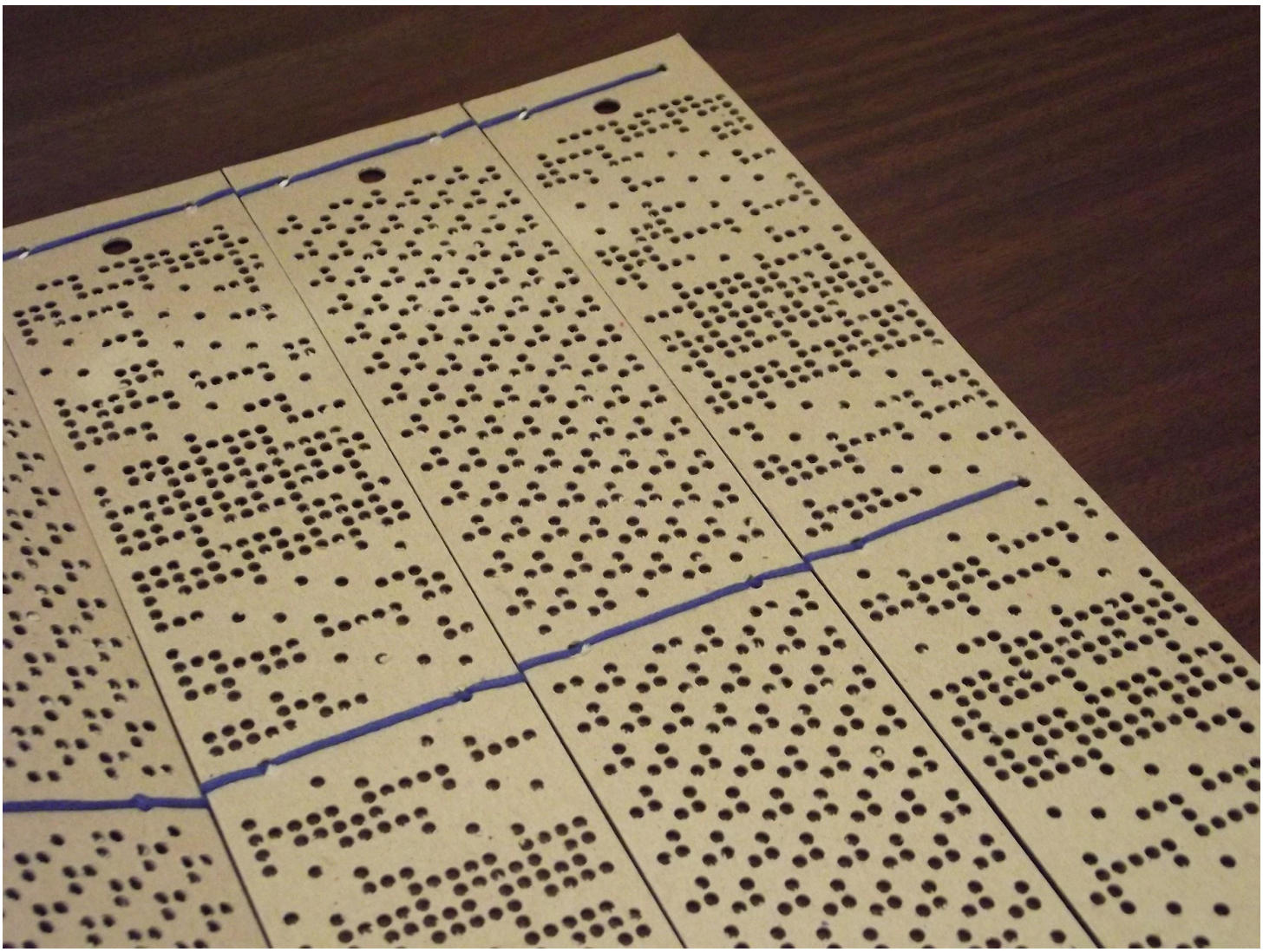
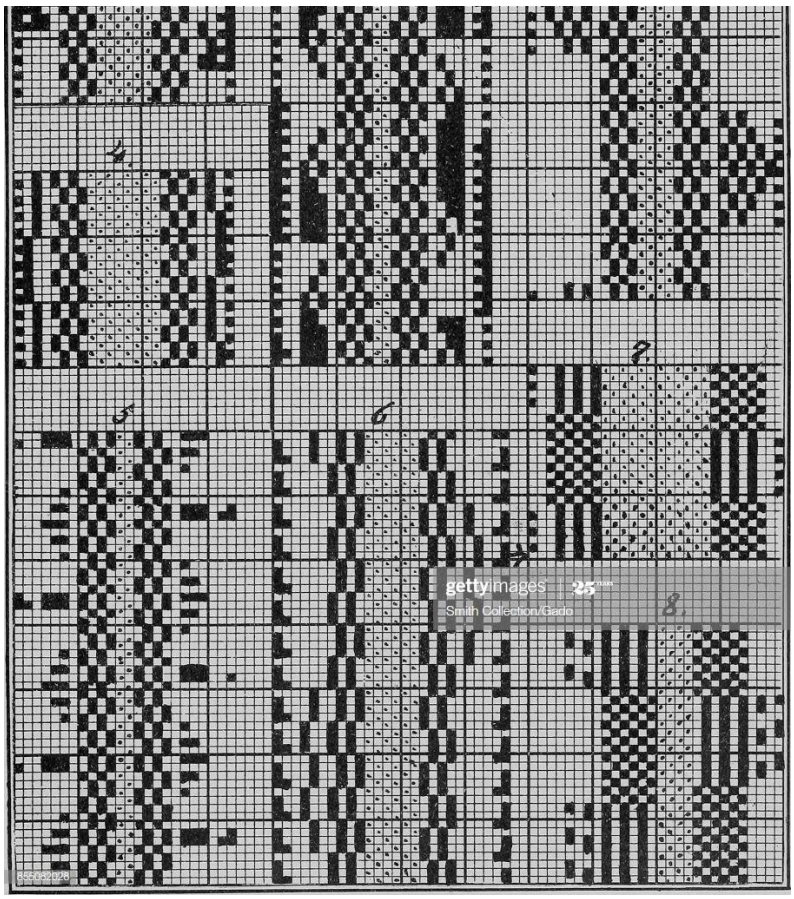


figure 15 : jacquard loom punchcards



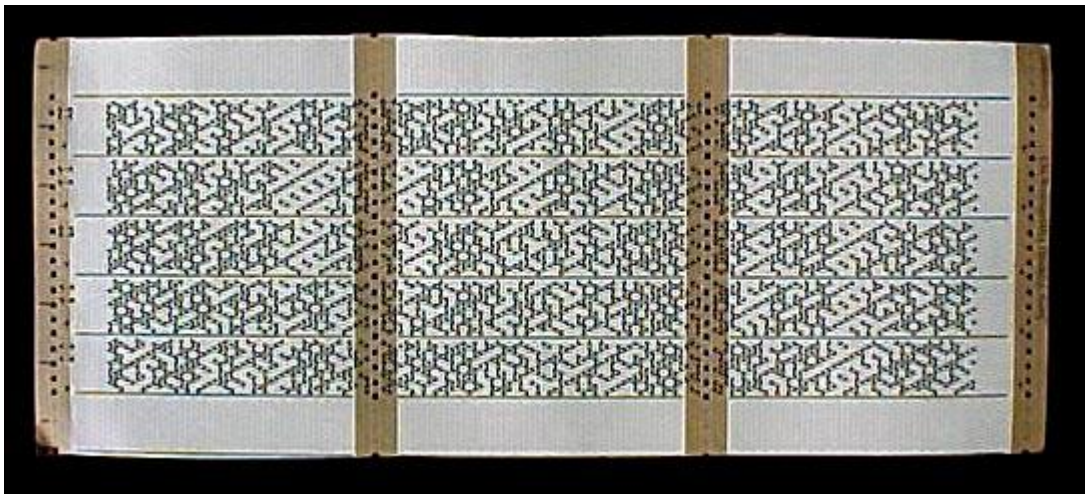


figure 16 : weaving draft

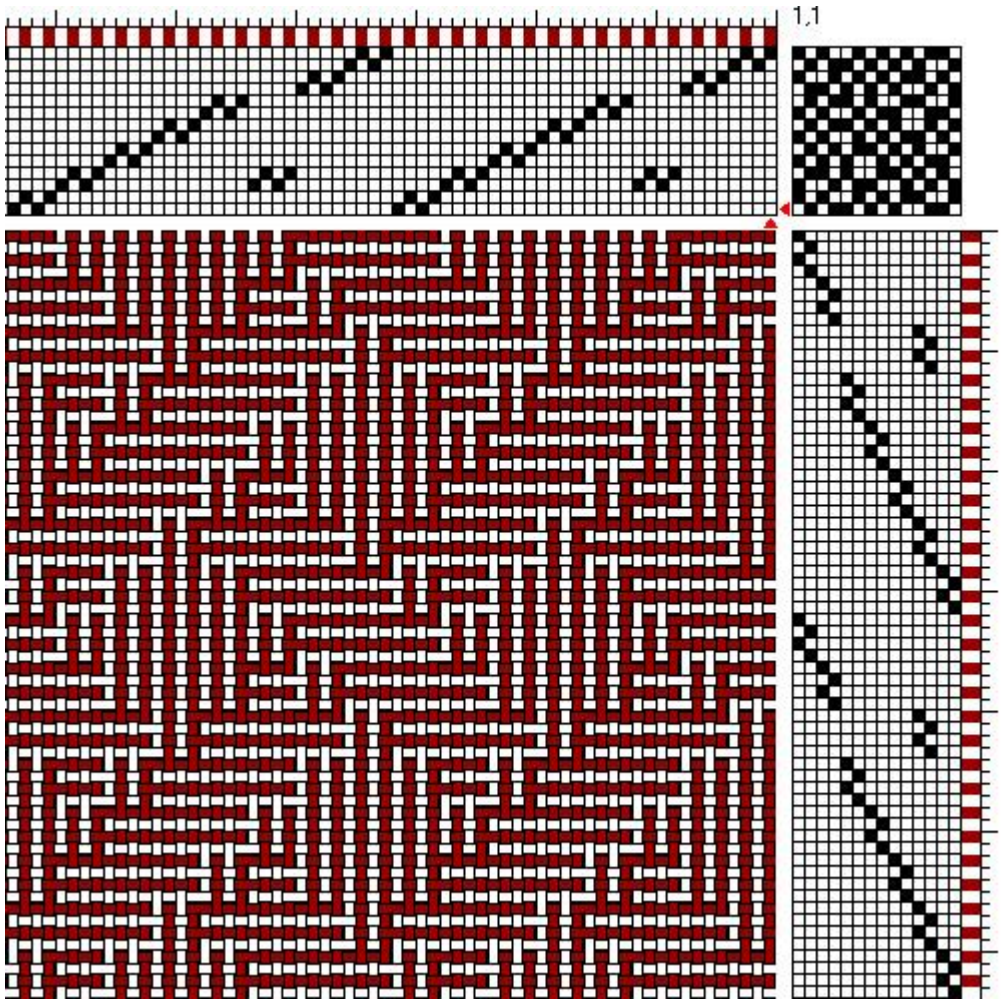


figure 17 : piano player roll

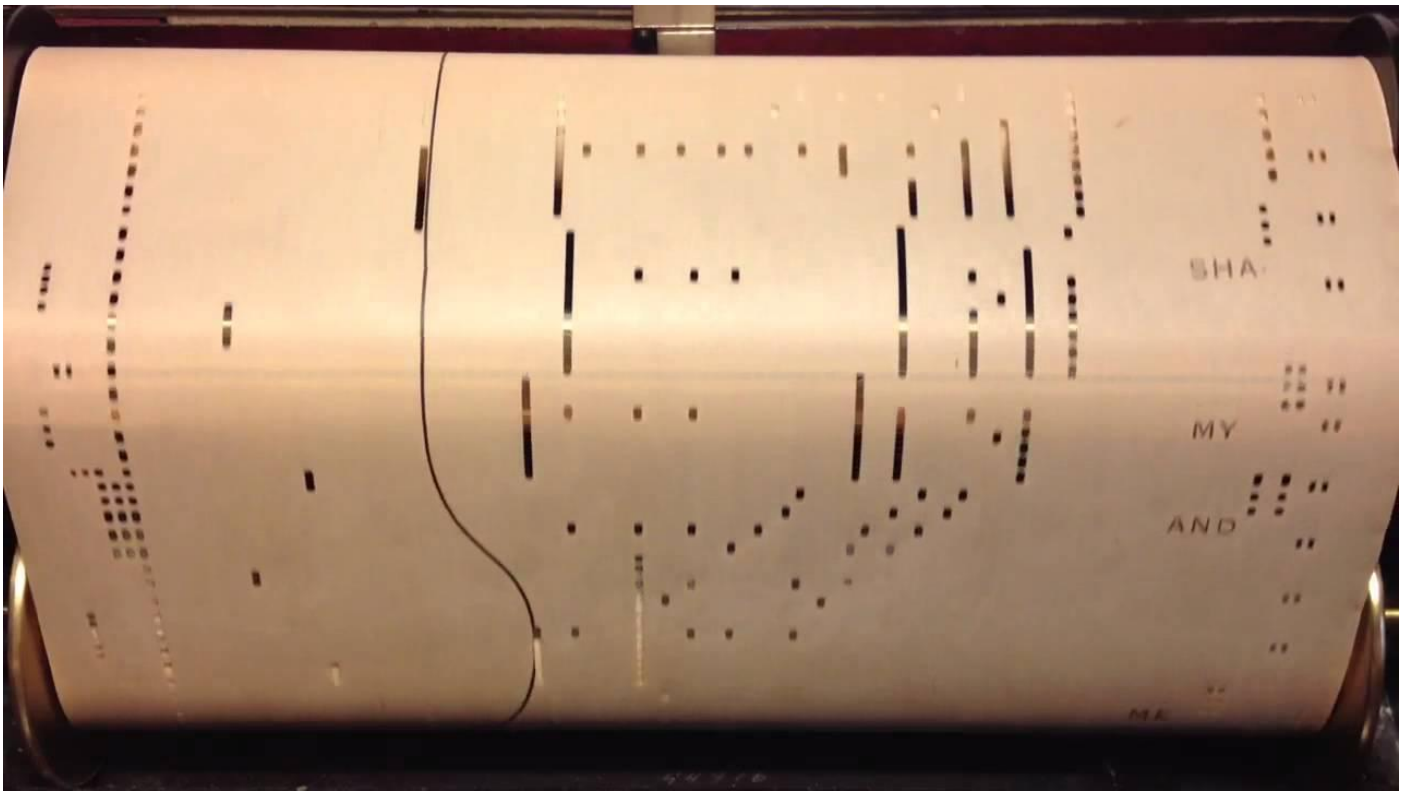


figure 18 : quipu

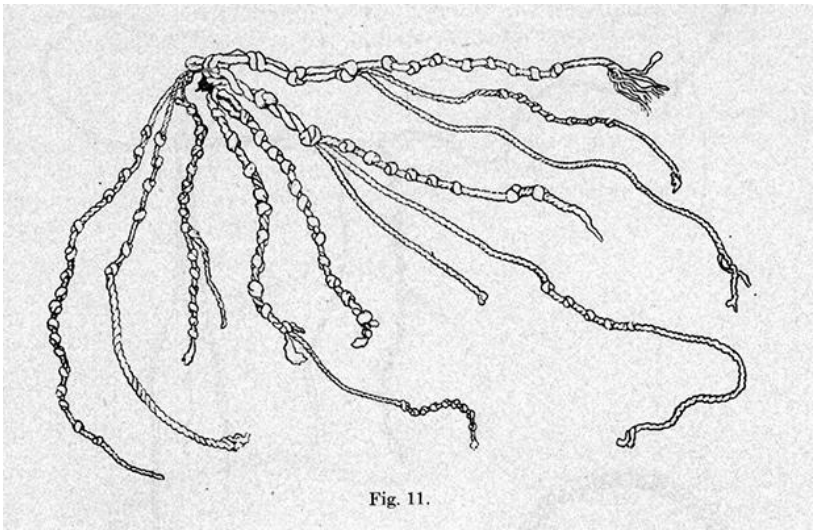


Fig. 11.

figure 19 : score

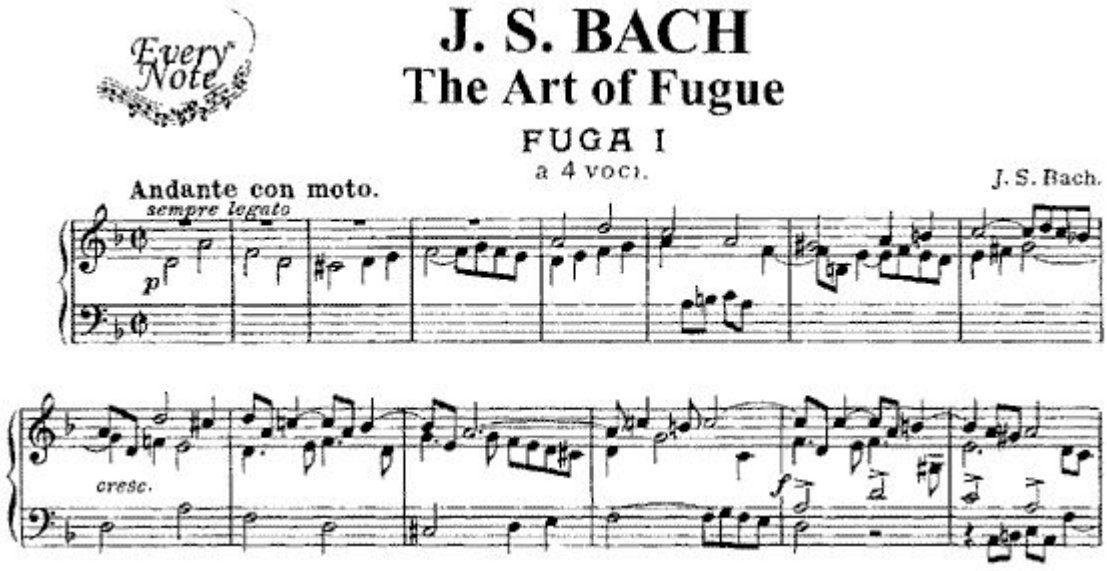


figure 20 : code

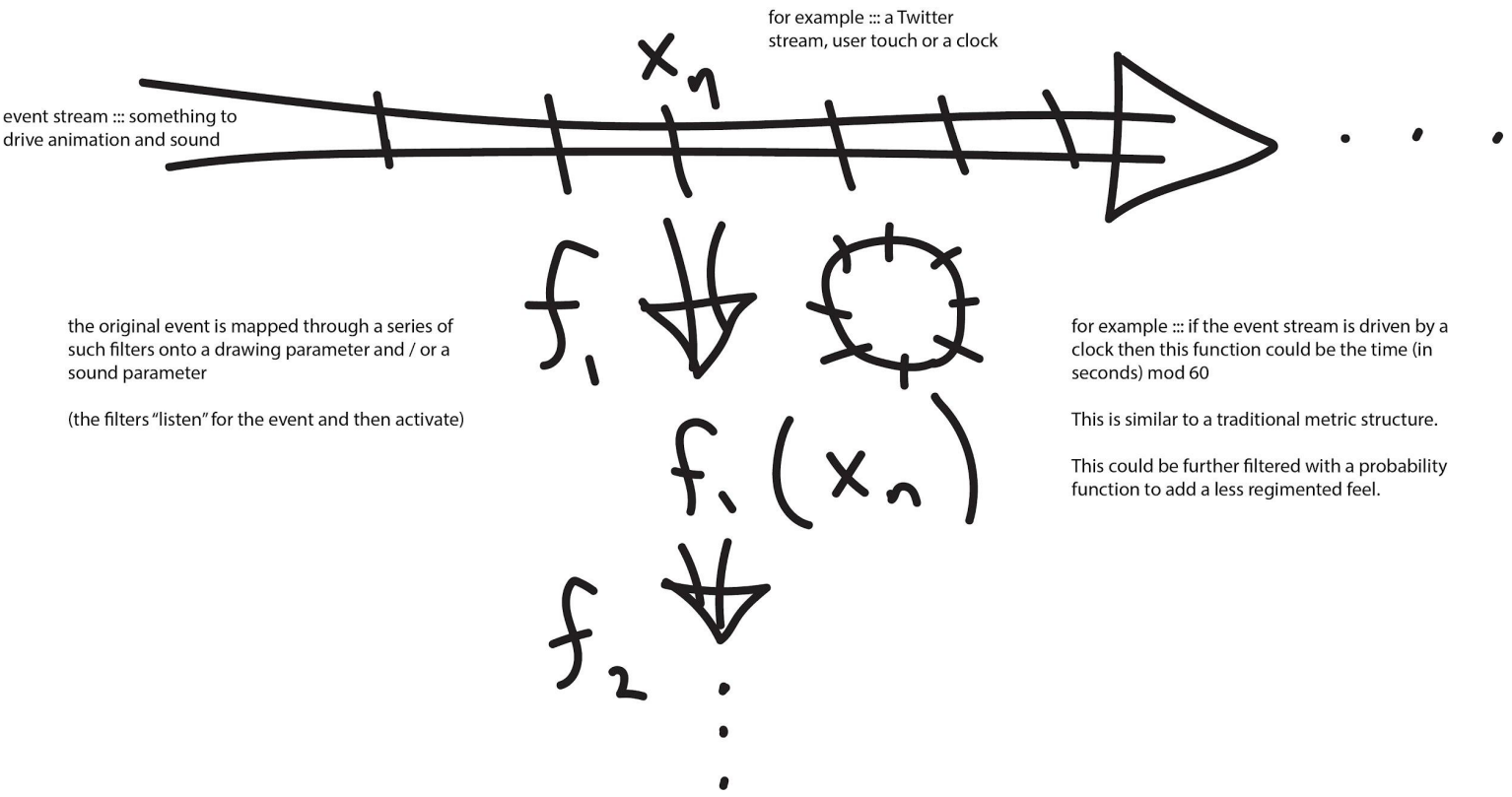
```
let createdrawp = z => {

	// ***** clock stream ---------
	(function() {
		let name = "tick";
		let dt = 1; //in seconds
		let date0 = new Date();
		let t0 = Math.floor(date0.getTime()/1000);
		let tostring = function(e) {return "clock"};
		let clock0 = {
			date: date0,
			t: t0, count: 0,
			changed: false,
			count: 0,
			past: Math.floor(t0 / 1000),
			dt:dt, t0:t0, tostring: tostring, name:name
		};
		z.streams[name] = Kefir.withInterval( 1000, emitter => { emitter.emit( { date: new Date() } ) })
			.scan( (state, e) => {
				state.date = e.date;
				state.past = state.t;
				state.t = Math.floor(e.date.getTime()/1000);
				state.changed = state.t !== state.past ? true : false;
				state.count = state.count + 1;
				return state;
			}, clock0  )
		z.streams[name].onValue( e => {
			// z.tools.logmsg(JSON.stringify(e));
			z.elements["clock"].el.innerHTML = z.tools.datestr(new Date(e.t*1000));
		});
	})();

	// ***** sound set stream ---------
	(function() {
		let name = "sounds";
		let dt = 38; //in seconds
		let date0 = new Date();
		let t0 = Math.floor(date0.getTime()/1000);
		let tostring = function(e) {return "sounds"};
		let sounds0 = {
			sounds: z.score.orchestration[ Math.floor(t0/dt)% z.score.orchestration.length ],
			count: 0,
			past: ["piano1"],
			dt:dt, tostring: tostring, name:name
		};
		z.streams[name] = z.streams["tick"].filter( e => e.t%dt===0 )
			.scan( (state, e) => {
				state.past = state.sounds;
				state.sounds = z.score.orchestration[ Math.floor(e.t/dt)% z.score.orchestration.length ],
				state.count = state.count + 1;
				return state;
			}, sounds0  )
		z.streams[name].onValue( e => {
			// z.tools.logmsg(JSON.stringify(e));
		});
	})();

	// ***** sound stream ---------
	(function() {
		let name = "sound";
		let dt = 1; //in seconds
		let tostring = function(e) {return "sound"};
		let sound0 = {
			count: 0,
			dt:dt, tostring: tostring, name:name
		};
		z.streams[name] = Kefir.combine([z.streams["tick"].filter( e => e.t%dt===0 && z.score.soundplaying && z.tools.ra
			.scan( (state, e) => {
				state.tick = e.tick;
```

figure 21 : sound maps : functional programming (streams ::: filter / map / fold)



the original event is mapped through a series of such filters onto a drawing parameter and / or a sound parameter

(the filters "listen" for the event and then activate)

for example ::: if the event stream is driven by a clock then this function could be the time (in seconds) mod 60

This is similar to a traditional metric structure.

This could be further filtered with a probability function to add a less regimented feel.

figure 22 : sound maps : vibrating strings / pitch transformations