MATERIAL PROGRAMMING

In the near future every other smart material will have computational power embedded in the form of graphene transistors or nanotubes [cf. 5]. These will be the ultimate *computational composites*: materials that hold classic material qualities, such as structural durability, flexibility, texture, weight, and color, but additionally being capable of sensing, actuating, and computing [6]. Indeed, computers will not be things in and by themselves, but embedded into the materials that make up our surroundings. This also means that the way we interact with computers and the way we program them, will change. Consequently we ask what the practice of programming and giving form to such materials would be like? How would we be able to familiarize ourselves with the dynamics of these materials and their different combinations of cause and effect? Which tools would we need and what would they look like? Will we program these computational composites through external computers and then transfer the code them, or will the programming happen closer to the materials? In this feature we outline a new research program that floats between imagined futures and the development of a material programming practice [5].

ENVISIONING A MATERIAL PROGRAMMING PRACTICE

Central to the practice of interaction design is crafting the couplings and relations between user actions and artifact functions. To design interactive artifacts therefore requires an understanding of the potential dynamics between sensory and actuating mechanisms in the materials we design with. It is a matter of "getting a feel" for the potential compositions of cause and effect. Gaining such embodied understanding, however, is only really possible through explorations with the materials we are to design with. With the rise in variety and complexity of computational composites over the coming decade, it becomes pertinent to develop a design practice that enables the designer to maintain this level of explorations. We envision material programming as becoming such a practice [5].

Supporting Kinesthetic Creative Practice

Material programming would complement traditional crafting of physical form with the crafting of temporal form and together they would make up the future practice of interaction design [6]. Indeed, material programming would be a programming practice that enables the designer to stay in the material realm. The designer would program directly on the material and thus have first hand access to explore and experience the outcome of different interactive compositions. It would minimize the distance between programming and execution, and provide the designer (programmer) access to real-time situated experiences of causes and effects. This immediacy would bridge the intellectual and physical gap we know from other detached programming practices and provide an opportunity for kinesthetic thinking [4].

Tools for Material Programming

A material programming practice would be a programming practice using physical tools. With tools in hand working directly with the material, the designer would be able to achieve an embodied sense of its interactive and expressive properties. Such tools would each have a specific function designed from the designer's point of view – rather than a programming logics point of view. By also limiting the scope for each tool's action space, it will be possible to create rather sophisticated tools in terms of what the designer can do with them. Such tools might require some learning and expertise, but we assume that professional interaction designers are willing to invest the time and effort needed. Yet, they would not demand highly technical skills from the designer, only interaction design skills. Essentially, we imagine a future design practice, where we use traditional material tools and machines to develop the physical form of the designs and the material programming tools to develop the temporal form of the interactive artifacts.

Situated and Real-time

Material programming would happen on-site, instead of through a detached desktop, with physical tools working directly on the materials. This would lower the threshold for the designer to truly explore the potential of a new material in context and thus give the designer a better sense of the design space. Such expanded support of kinesthetic creative practice and bespoke designs would likely result in more sophisticated expressions, fitted to their context of use. Material programming would, however, be limiting if it were to the only means of programming interactive artifacts. Therefore, we envision integration with more complex back-end algorithmic programming and access to databases when needed. In that sense, material programming can be seen as a sort of interface programming. In some cases, however, there may not be any back-end at all and the interactive artifacts can be designed from working with these computational composites alone.

SKETCHING A MATERIAL PROGRAMMING PRACTICE

To give a better sense of what we mean by tools for material programming we here present some sketches of physical tools for programming computational composites. In order to convey the functionality of these tool sketches, we assume the existence of a very particular computational composite: a shape-changing material that can respond to airflow. We imagine this computational composite to be used in interior design and architecture, for instance in interactive facades or in furniture (See Figure 1). Through speculating and enacting the practice of shaping the behavior of this composite, we discuss qualities of what material programming for interaction designers could look like.

For instance, we suggest that the tools we need for programming shape-changes would be a Select tool and a Force tool (See Figure 2). The Select tool is used to indicate which area of the material is activated for programming by brushing over it. The Select tool can also be used for copying and pasting a programmed area to other areas. The Force tool is used to program the shape-changing behavior with respect to when, where, and how force should be applied in the material by simulating a pulling motion. Both tools are inspired by known techniques for manipulating materials, such as brushing (selecting), and pulling (moving). To minimize the distance between programming and execution, the tools work on the material by wirelessly connecting to the embedded computational power in the material. The material is activated when the tools are brought in close proximity to them. This allows for exchanging information from the tool to the material and *vice versa*. Similarly, moving the tool away from the material will 'disconnect' the embedded computers and the tools.

In this particular material we are interested in programming the relation between the airflow and a resulting shape change of the material. The first step is therefore to select the area that should change its shape by brushing over it with the Select tool (See Figure 3). By adjusting the distance measure on the tool, larger areas of the material than one's arms range can easily be selected. This would, for instance, be needed if the design demands a large material surface instead of a small one exemplified in this feature. Using the Select tool allows the designer to program different behaviors into different parts of the material. Next step is to connect the Force tool to the material. Independently of input, the designer can start out exploring the expression space of the shape changes and become familiar with the expressive properties of the particular computational composite (See Figure 4). By pulling or pushing the sliders in the Force tool the material responds with protrusions in the corresponding direction. The sliders are operated directly with the hands and the tool is responsive to the pressure applied (pace of the fingers), which is then translated into the strength of the force in the material (pace of shape-change). Playing around with different forces applied to the selected area, the designer is able to get a feel for the shape changing qualities of the material, and the relation between the actions with the Force tool and the material reaction. Since one (continuous) swipe on the Force tool only results in one (continuous) movement in the material, the tool also allows for layers of forces, making it possible to compose more intricate forms of shape-change.

Afterwards the designer (or designers) can use the same tool concurrently with increasing the airflow (input) at the desired areas of the material (See Figure 1). Again, the designer can play around with different reaction patterns – whether it should be a simple action-reaction or if an increase in airflow should result in more elaborate shape-changing patterns. Finally, when the designer has found a desired relation between expression and the airflow, the Select tool can be used to copy and paste this to other parts of the material – or another piece of the material if needed.

Programming materials can thus be akin to enacting a composed dance or gradually shaping forms in clay. Depending on the designer's experience it can be a craft-like explorative practice or a meticulously composed design practice. The more experience, the more intricate expressions the designer will be able to compose. The key to this is the open-endedness of the tools and the real-time reaction to input.

BUILDING ON RELATED PROGRAMMING PRACTICES

In most cases the default mode of programming computers is textual. There are, however, alternatives to textual programming languages that are relevant to discuss in relation to material programming. Visual programming, tangible programming, and programming-by-example, for instance, all support explorative design practices by minimizing the distance (mentally as well as physically) between programming and execution. We will here discuss the relation between the qualities of these programming practices and material programming.

Visual Programming

Visual programming works by replacing textual code with visual notations (i.e. 2D representations) and tools as means to construct software (See Figure 5) [cf. 2]. Thus, visual programming utilizes people's ability to easily recognize and work with visual patterns, and thereby minimizes the need for learning. Visual programming is good in aiding rapid development, in particular in the early stages of design. This is partly due to the low threshold of changing the logical structure of a program, which makes it easy to experience multiple design alternatives in an explorative manner.

Material programming would also utilize this latter quality since programming and execution happens in the same material realm. A textual or graphical overview of the data structure and algorithms would, however, not be an integrated part of the practice although it could be made available elsewhere.

Tangible Programming

Tangible programming environments use physical objects to represent various programming elements, commands, and flow control structures (See Figure 6) [cf. 1]. Here, the manipulation and arrangement in space of these objects are used to construct an algorithm. Similar to visual programming, tangible programming enables a visible and tangible organization of a program which eliminates levels of abstraction. Yet, by relying on physical manipulation, tangible programming is even less abstract than visual programming, which means it is even less capable of supporting development of complex algorithms. The important advantage is, however, that it references some of our experiences in the physical world.

Both tangible and material programming thus operate in the physical world. However, while the tangibility in tangible programming typically remains a rather cognitive activity removing the designer from the material at hand, material programming would be an embodied activity tightly coupled to the material's expressive potential.

Programming-by-Example

Programming-by-example is a programming practice where the programmer demonstrates an algorithm to a system by recording a set of actions through an artifact/interface, which can then be played back in that artifact/interface (See Figure 7)[cf. 3]. Programming by example is typically applied in situations where the artifact is a one-off, accessible, and tangible, such as in the design of shape-changing interfaces and robots. Like visual and tangible programming, programming-by-example has a low threshold for beginners and non-technical disciplines. Further, its complete lack of abstractions makes composing the behavior immediate. The allure of programming-by-example in a design context is how it allows designers to use their tacit or bodily knowledge in a manner similar to how non-computational products are designed, however constrained by what the materials, actuators, and sensors in the artifact allow.

In programming-by-example we recognize material programming's quality of working almost directly with the design material to be programmed. However, programming-by-example often results in a very limited and artifact specific design space. Instead, the tools used in material programming allow the designer at least one layer of abstraction, enabling a larger action space and thus potentially more sophisticated designs. Also, the envisioned tools would allow a wider array of applications exclusively utilizing a computational composite's properties, due to their specific connections to the materials.

WHY A MATERIAL PROGRAMMING PRACTICE?

In this feature we presented the notion of material programming as a future practice for designing computational composites [5]. Such practice would be a way for designers to explore and experience the dynamics of the computational materials they are working with. This will in turn support the designers' kinesthetic creative practice and we believe they will become capable of

composing more sophisticated and complex temporal forms in their designs. We propose this practice knowing that the current technology and materials are not entirely ready to support it, but we are convinced that they could be in a not too distant future. The future material programming practice will not look like the one proposed above, so the contribution here is therefore in arguing for the qualities such practice would embody which is fourfold:
First, a material programming practice would not rely on any direct representation of the programming actions performed on the material, beyond the material itself. A material programming practice thus unites the 'programming' and 'running' modes, while avoiding unnecessary abstractions that could move the attention away from the material. The argument being that the better the interaction designer knows the material at hand, the more sophisticated and finished designs we can expect. Instead of shaping these temporal dimensions through detached means (e.g. by writing code on a detached computer only), the actual interactive behavior of the material is explored and programmed on the material, in real time and in-situ.
Second, we see how the tools bring us closer to an actual form-giving practice in interaction design. Through this practice the unique interactive and physical properties of the particular materials easily play a key role in both concept development and actual creation. In a way, material programming could be more in line with traditional crafting practices, where several dedicated tools are used for crafting a material, and can be mastered through practice and skill gained over time. This is not unlike a silversmith's practice. The Force tool, for example, provides the possibility to explore different rhythms and directions of movement in a shape-changing computational composite, supporting an understanding of the properties of the computational composite at hand.

Third, since the physical interaction with the material is central to this programming practice, the designer can slowly develop tacit bodily skills and knowledge of how to use the expressive properties of both tools and materials. The tools allow the interaction designer to use her body in ways similar to that of crafting non-computational materials, enabling and utilizing the designer's expressive potential. This could, for example, be reflected in the smooth and refined actions of sliding the thumbs on the Force tool's sliding areas to explore the speed and acceleration of a material's shape-change.

Fourth, the tools allow for at least one level of abstraction, which enables the designer to utilize the ability of programmed cause-and-effect in the computational composites. In other words, the input/output relation does not have to be one-to-one but can assume other temporal forms and dependabilities in between. Further, we also see a good possibility for these materials to be coupled with more advanced computational power, once embedded in designs. Thus, we imagine the programming of the material coupled with more advanced algorithms and databases in a back-end design, which would probably rely on traditional textual programming. In that sense, computational composites and material programming can be seen as the front-end of cloud-based Internet-of-Things from a programming perspective.
Finally, an important bonus is that a material programming practice would likely appeal to a wider array of design and craft practitioners. As such, the design of our future artifacts and environments would not only rely on designers brought up in technological educations and practices. We envision that this wider array of participants could probably lead to a more varied range of material expressions.

As proposals of new ideas and research programs go, a full realization of a material programming practice would not happen tomorrow. Working towards it would require collaborations from material science, computer science, interaction- and industrial design. And in the end, it will look quite different than the sketched tools proposed here. With this work however, we intend to start giving form to the new possibilities we have before us.

1. Felix Hu, Ariel Zekelman, Michael Horn, & Frances Judd. 2015. Strawbies: explorations in tangible programming. *In the Proceedings of the 4th International Conference on Interaction Design and Children (IDC '15)*, pp. 410-413.
2. Brad A. Myers. 1986. Visual Programming, Programming by Example, and Program Visualization: A Taxonomy. *In the Proceedings of the Conference on Human Factors in Computing Systems (CHI '86)*, pp. 59-66. 10.1145/22627.22349
3. Hayes Solos Raffle, Amanda J. Parkes, & Hiroshi Ishii. 2004. Topobo: A Constructive Assembly System with Kinetic Memory. *In the Proceedings of the Conference on Human Factors in Computing Systems (CHI'04)*, pp.
4. Dag Svanæs. 1997. Kinaesthetic Thinking: The Tacit Dimension of Interaction Design. *Computers in Human Behavior*. 13, 4. 443-463.
5. Anna Vallgårda, Laurens Boer, Vasiliki Tsaknaki, & Dag Svanaes. 2016. Material Programming: a Design Practice for Computational Composites. *In the Proceedings of the Nordic Conference on Human-Computer Interaction (NordiCHI'16)*, p. 46.
6. Anna Vallgårda and Johan Redström. 2007. Computational Composites. *In the Proceedings of the Conference on Human Factors in Computing Systems (CHI'07)*, pp. 513-522.