Supporting Expressive Procedural Art Creation through Direct Manipulation

Jennifer Jacobs[1,2], **Sumit Gogia**[1], **Radomír Měch**[2], **Joel Brandt**[2]
[1] MIT Media Lab [2] Adobe Research
jacobsj@media.mit.edu, summit@mit.edu {rmech, joel.brandt}@adobe.com

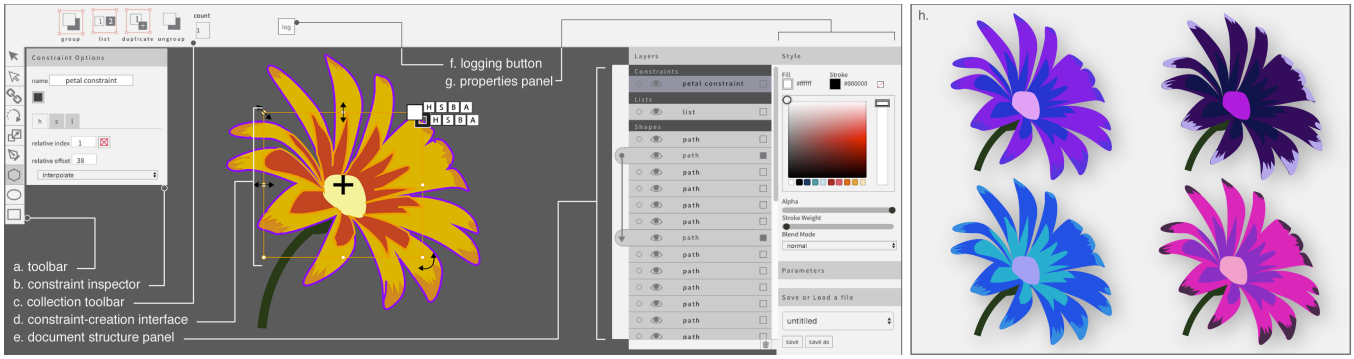

Figure 1. Left: Para's interface. Right: Para enables artists to quickly set up constraints using direct manipulation interactions with their art. Here, an artist has set up a series of color constraints that makes exploring color variations highly efficient.

ABSTRACT

Computation is a powerful artistic medium. Artists with experience in programming have demonstrated the unique creative opportunities of using code to make art. Currently, manual artists interested in using procedural techniques must undergo the difficult process of learning to program, and must adopt tools and practices far removed from those to which they are accustomed. We hypothesize that, through the right *direct manipulation interface*, we can enable *accessible and expressive* procedural art creation. To explore this, we developed Para, a digital illustration tool that supports the creation of declarative constraints in vector artwork. Para's constraints enable procedural relationships while facilitating live manual control and non-linear editing. Constraints can be combined with duplication behaviors and ordered collections of artwork to produce complex, dynamic compositions. We use the results of two open-ended studies with professional artists and designers to provide guidelines for accessible tools that integrate manual and procedural expression.

Author Keywords

Procedural Art; Generative Art; Programming

ACM Classification Keywords

H.5.2. Information Interfaces and Presentation: User Interfaces — Graphical user interfaces.

CHI 2017, May 06 - 11, 2017, Denver, CO, USA

ACM 978-1-4503-4655-9/17/05 · $15.00
DOI : http : //dx.doi.org/10.1145/3025453.3025927

INTRODUCTION

New tools and technologies have emerged that support the production of procedural, parametric, and generative forms of art. While differing in their applications, these art forms share the common property of being defined by a representational system of rules, relationships, and behaviors, which makes them flexible, adaptable, and capable of systematic revision. Currently, most professional procedural art is produced through computer programming. Describing art with code allows artists to manage complex structures, automate processes, and generalize and reuse operations [38, 27]. More broadly, procedural tools support exploration, experimentation, and play because their processes can be revised and iterated upon without loss of quality or affordance [26].

There is increasing interest in the artistic applications of programming, as indicated by the growth of coding platforms for art [37, 3]. Yet programming still presents many challenges for artists. In general, programming is challenging to learn. Programming languages present barriers to incremental learning by requiring people to learn many concepts before accomplishing even simple tasks [20]. Furthermore, while manual artists and craftspeople learn mainly through observation and action of tacit manual skills [26], programming education often emphasizes a structured top-down approach of learning formal principles for application to concrete, predefined tasks [10]. Victor states that learning programming may always be challenging because learning abstraction is hard [53], but he argues that *tools* can be designed to make that challenge more tractable [51].

In addition to the challenges of learning to program, there are significant interaction differences between programming tools and software for manual art. Graphic illustration soft-

ware is concrete: it approximates physical art media through direct manipulation interactions. Concrete tools often have a gradual learning curve [46], and can play an important role in engendering creative diversity [41] and supporting the conception of new ideas [45]. Programming tools, by comparison, are typically representational [50], where users edit a *description* of the work rather than the work itself[1]. Representational tools can present an additional challenge for artists accustomed to working with and thinking in terms of concrete mediums.

Contrasting the creative opportunities of procedural art with the challenges of programming raises our research question: *Can we support accessible and expressive procedural art by enabling people to describe procedural relationships through direct manipulation?* Prior work in developing accessible procedural tools for art has focused on creating simplified textual programming languages and environments [34, 35, 14], augmenting graphical user interface (GUI)-based tools with visual programming languages [7], or linked editing between textual programming and direct manipulation [15, 2, 6, 11]. Our approach is to augment the conventions of graphic art software to support the creation and modification of procedural relationships exclusively through manipulation of concrete artwork. In doing so, we build on prior work using direct manipulation to create dynamic relationships in data visualization [53], interactivity [17], and architecture [24], but with the new objective of supporting and extending the practices of manual fine artists. This goal is challenging because it requires developing a procedural representation that supports expressive creation, yet is compatible with direct-manipulation conventions.

This paper makes the following contributions: First, we introduce Para, a new direct-manipulation tool that supports procedural art through live, non-linear, and continuous manipulation. Second, we demonstrate core declarative procedural operations that can be expressed through direct manipulation, are easy to use, and enable diverse creative outcomes. These operations include declarative geometric and stylistic constraints, visually represented ordered lists that can be used to specify how constraints map to collections of objects, and declarative duplication to enable dynamic copying. Constraints, lists, and duplication can be combined in different ways to produce different outcomes. Third, we provide evidence for how the direct manipulation of procedural relationships can extend manual practice through two open-ended studies. These studies evaluate the accessibility and expressiveness of our system in the hands of professional artists by examining how they work and the artifacts they produce. To our knowledge, these are the first open-ended studies in the domain of direct manipulation procedural art. Fourth, we provide recommendations for building expressive direct manipulation procedural tools, as gleaned from evaluation.

BACKGROUND AND RELATED WORK

To inform the development of Para, we reviewed literature that discussed support for learning and expressiveness through programming. We also examined research about the practices of fine and commercial artists and visual designers. While artists and designers differ in their goals, they frequently rely on the same tools; for example, Photoshop and Processing are both used for art and design. Questions about how tools support different creative motivations are outside the scope of this paper. Rather, our research focused on how different tools support or hinder creative practice. Therefore, we examine creative tool use in commercial and fine-art. In the comparison of procedural and manual tools for art and design, we focused on two primary tensions: ease of use versus expressiveness, and concrete versus representational tools.

Ease of use versus expressiveness

Programming languages are challenging for end users because they must learn a great deal in order to produce simple results [28]. This tension is reflected in procedural tools for art and design. Creative coding applications like openFrameworks [23] and Processing [35] are extremely expressive; however, these tools require people to learn numerous textual programming conventions and lengthy APIs before producing basic artwork. These severe learning thresholds are common in end-user programming and can present insurmountable barriers to novices [20]. Visual programming languages [7, 1, 39, 8] can provide a more accessible entry point by providing an impression of directly constructing a program, rather than abstractly designing it [29], and represent an important initiative to support artists, designers, and other end-users in programming. We describe interaction differences between visual languages and our system in the following subsection.

A similar conflict between ease of use and expressiveness also exists in computer-aided design (CAD). Entry-level parametric tools often enable *customization* of existing designs, however the ability to *compose* new designs requires learning significantly more advanced tools and workflows. Many GUI-based CAD tools, including Rhino and Inventor, support procedural and parametric functionality by enabling designers to author scripts and programs that act on objects created in the GUI. These features offer new opportunities but are difficult to use and learn in comparison with the GUI [24]. When adapted for people without formal knowledge in math, geometry, and logic, CAD tools often take the form of customizable parametric models [13, 42]. Although these tools can increase participation, studies indicate that they do not produce a diversity of creative outcomes, nor scaffold novice use of advanced, open-ended CAD tools [30].

We seek to create accessible procedural tools that retain forms of expressiveness relevant to a specific domain. Research that demonstrates this approach in other domains includes interaction design via parallel source editing and tunable control interfaces [9], user-interface (UI) programming through combination of declarative constraints, state machines, and a live, visual notation [32], and the specification of UI behaviors through direct manipulation of their ordering [25].

In developing procedural tools for artists, much can be borrowed from research in supporting novice programmers in other domains. In supporting young people and novice CAD

[1]Our use of the term *representational* is from computer science. It is distinct from its meaning in art to describe realistic imagery.

users, it is important to balance accessibility with expressiveness. Often this can be achieved by presenting a small number of programming concepts in a relevant, creative context. Resnick and Silverman asserted that for young people, *a little bit of programming goes a long way*, and focused on developing tools with a minimal number of carefully selected programming concepts. This enables children to explore a variety of creative outcomes, while minimizing learning thresholds [40]. The Logo [34] and Scratch [39] programming languages are two examples of this approach, and apply carefully designed simple languages with computational drawing and interactive storytelling, respectively. Similarly, in the CAD domain, MetaMorphe presents a digital fabrication design framework in a format similar to web scripting, making parametric design accessible and expressive for people with familiarity in web programming [48], and Jacobs and Buechley assisted novices in expressive forms of computational design by linking a constrained programming environment with physical crafting [14]. We designed Para around a minimal set of programming concepts in order to make it accessible to novice coders. We structured these concepts within the software so that they could be used to produce diverse outcomes.

Concrete interaction versus symbolic representation

In fine art, concrete tools and media are common. These range from the physical (brushes, carving tools, hands), to digital (using direct-manipulation software to manually adjust bitmaps or vectors). There is evidence that concrete engagement aides creative decision making; designers often conceive ideas through reflection-in-action and through direct engagement with physical media [45]. Many digital interaction designers find it difficult to conceive of novel concepts while working in the immaterial domain of software [33]. Concrete engagement also can play a role in learning. Klemmer describes how concrete interactions can support learning of new concepts in ways that reading and listening do not [19]. In contrast to concrete media, programming is abstract and often requires working through a textual representation. Visual programming languages can reduce some of the challenges of textual languages; however, they still require the artist to work through an abstract representation of the artwork rather than on an artifact itself [50]. Our approach is different from both textual and visual programing languages because it uses direct manipulation of the artwork to define and manipulate procedural relationships.

While it is difficult to describe certain computational concepts in concrete form, direct manipulation can be used to describe dynamic relationships in ways that are comparable to programming languages. The earliest example is SketchPad, which demonstrated that parametric relationships could be described by selecting and manipulating geometric shapes [47]. More recently, Kitty and Skuid aimed to reduce difficulty in animations and interactive infographics by creating dynamic behaviors through direct manipulation of a relational graph [17, 18] and Recursive Drawing enabled nonlinear editing of self-similar designs through the nesting of drawing canvases [43]. Hoarau and Conversy demonstrated graphic design tools that enable users to indicate dependencies between the properties of objects [12], Xia et al. applied object-oriented principles to reduce reliance on WIMP UIs for touch and tablet interfaces [54], and Blackwell demonstrated a direct-manipulation system for procedural editing of bitmaps [5]. In data visualization, Drawing Dynamic Data Visualizations [53] and Apparatus [44] support the creation of interactive visualizations through direct manipulation.

Common	Rare
Complex forms and patterns through procedural duplication and object-oriented programming.	Close integration of procedural and manual illustration.
Iterative variation across scale, form and color.	Procedural transformation of hand-drawn artwork.
Regular geometric/ symmetrical distribution of form.	
Generative compositions through random distributions (normal, uniform, etc.)	.

Table 1. Techniques identified in professional procedural artwork.

Our work is differentiated from prior work in two ways: the domain in which we work, and our approach to evaluation. Whereas prior tools support data visualization, interactivity, and animation, or emphasize new interaction modalities, our objective is to develop *accessible* procedural tools that extend the practices of *fine artists*. This distinction is emphasized by Para's support of *iterative variation and dynamic duplication*, which is absent from prior tools [12, 54, 17, 18]. We are inspired by prior work that suggests the creative potential of dynamic direct manipulation, but recognize the need to evaluate this approach in actual art practice. Much of the prior research lacks evaluation; Hoarau and Conversy are unsure if their interface is understandable due to lack of user evaluation, and Victor recognizes his tools are untested stating that they are "not necessarily the correct way of doing things" but rather a starting point for exploring new interfaces [53]. Other tools are evaluated, but focus on tool accessibility and efficiency through predefined tasks [54, 17, 18]. Conversely, we seek to understand the *expressive* potential of our system in professional art practice; therefore, we evaluate it through extended open-ended studies and present polished, original artwork in our results.

PARA SOFTWARE DESIGN AND IMPLEMENTATION

Para is a software tool aimed at accessible yet expressive integration of procedural techniques in visual art through a direct-manipulation interface.[2] Before building Para, we developed design guidelines by compiling procedural artworks by professional artists including Joshua Davis, Casey Reas, Marius Watz, Emiliy Gobielle, Theo Watson, Christopher Coleman, and Erik Naztke. Analysis of this artwork revealed procedural aesthetics and methods consistent across the artworks (table 1, common). We also highlighted qualities found in only a few artworks that blended manual and procedural creation (table 1, rare). Following this initial analysis, we prototyped interactions that supported similar approaches and aesthetics but were compatible with direct manipulation. The primary conventions we sought to preserve were image-based communication of structural and organizational relationships, support for live and non-linear edits, removal of textual commands and expressions, and integration of interface components from direct manipulation art tools. We iteratively re-

[2]Para is available for use at http://paradrawing.com/ and the source is at http://github.com/mitmedialab/para.

Iterative Behavior	Description
Interpolation	Evenly distributed values sampled from a Lagrange polynomial derived from the reference.
Random	A uniform distribution of random values between the min and max values of the reference. Random seed is re-calculated when min or max is manually changed.
Gaussian	A normal distribution of values with the mean derived from the first reference object and a standard deviation derived from the distance between the first and last object.
Radial	Polar values based on a diameter determined by the min and max values of the reference. list.
Alternate	A series which cycles through the values of the objects in the reference (e.g for reference values 360,45,250 the result will be 360,45,250,360,45…)

Table 2. Iterative constraint behavior options.

vised the prototype interactions into a unified system that could produce sample artwork. We informed this revision through regular meetings with one artist (Natzke), and interviews with two others (Gobielle and Coleman). In the remainder of this section, we detail the interface and feature set of Para in relation to these guidelines, and illustrate how these features enable different forms of expression.

Interface

We developed Para as vector drawing software because it enabled us to structure the interface and interactions around software used by artists. The interface features analogs of elements in digital illustration software, including tools for manual shape creation, selection, and transformation (fig 1 a); a panel adjusting stylistic and geometric properties of vector artwork (fig 1 g); and a document structure panel that displays the current components in the drawing (fig 1 e). Para is implemented in JavaScript using Backbone.js [4], and the Paper.js vector graphics scripting framework [21]. Although Paper.js contains methods for managing hierarchies of vector graphics, we did not use this functionality; because procedural nature of Para required us to develop our own representation for document structure, which we detail below.

Constraints

In textual procedural tools, complex relationships often are comprised of simpler, low-level operations and relationships. Para builds on this approach through object-to-object constraints, which serve as the fundamental building block for advanced procedural functionality. Constraints allow artists to create relationships between geometric or stylistic properties of a minimum of two vector objects: a *reference* (the object whose properties are referenced in the constraint) and a *relative* (the object being constrained). Constraints satisfy three important conventions of direct manipulation. First, they support non-linear edits without creating errors or inconsistencies. Second continuous direct-manipulation edits to a constraint reference results in live updates to corresponding relatives. Third, constraints provide a visual means of describing mathematical relationships between graphical forms. When created, constraints preserve the current state of the drawing. In their simplest form, one-to-one constraints are represented as expressions of the form $f(x) = x + o$, with x the value of the reference property and o the existing offset between the reference property and relative property prior to constraint. This format preserves any difference in values between the reference and relative properties when the constraint is created, enabling artists to describe constraint

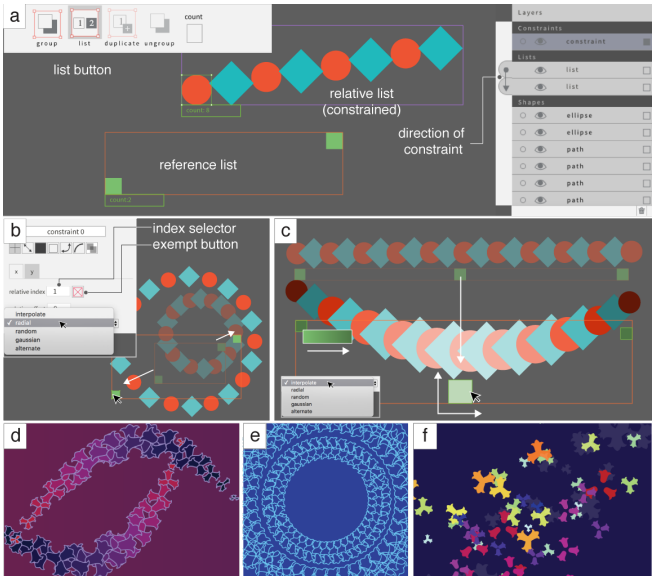

Figure 2. Variations on many-to-many constraints. (a) Basic list constraint between 8 relatives and 2 references. (b) Radial pattern produced through radial distribution with two reference shapes. (c) Arc patterns and color gradient produced with polynomial interpolation. (d-f) Variations of polynomial, radial, and random distributions.

relationships by drawing them rather than by writing them as a mathematical expression. The constraint tool ( ) enables the creation of *one-to-one constraints* by selecting the relative object and reference in succession. When the reference is selected, a set of icons appears over the reference object (fig 1 d), which enables the artist to specify the property to constrain: position, scale, rotation, fill and stroke color, stroke weight, or a sub-property of these. We extended Constraint.js [31] to handle constraint propagation and added functionality to detect and prevent constraint cycles.

Figure 1 demonstrates one application of constraints to produce variations in the colors of a flower illustration. Here, the fill color of the inner red portion of the flower is constrained to the fill color of the yellow center and the outer orange portion is constrained to the red portion. As the hue, lightness, or saturation of the center is changed, the colors of the outer petals shift while maintaining their original offset. This enables global relative color changes across a drawing from a single edit (fig 1 h). This also demonstrates how edits propagate across chained constraints to secondary relatives. By varying which sub-properties are constrained, artists can fine-tune how different parts of the drawing respond to an edit.

Lists

Our discussions with procedural artists highlighted the importance of supporting efficient manipulation of large numbers of visual elements. Textual tools enable management of multiple objects through abstract datatypes to store collections of data and control structures like loops to efficiently generate iterative variations across collections. In Para, multiple objects are managed through lists: visually represented, ordered collections of artwork that can be procedurally manipulated. Lists are higher-level structures and can be used to map transformations to multiple objects. Lists are created from selected objects using the list button and appear in the

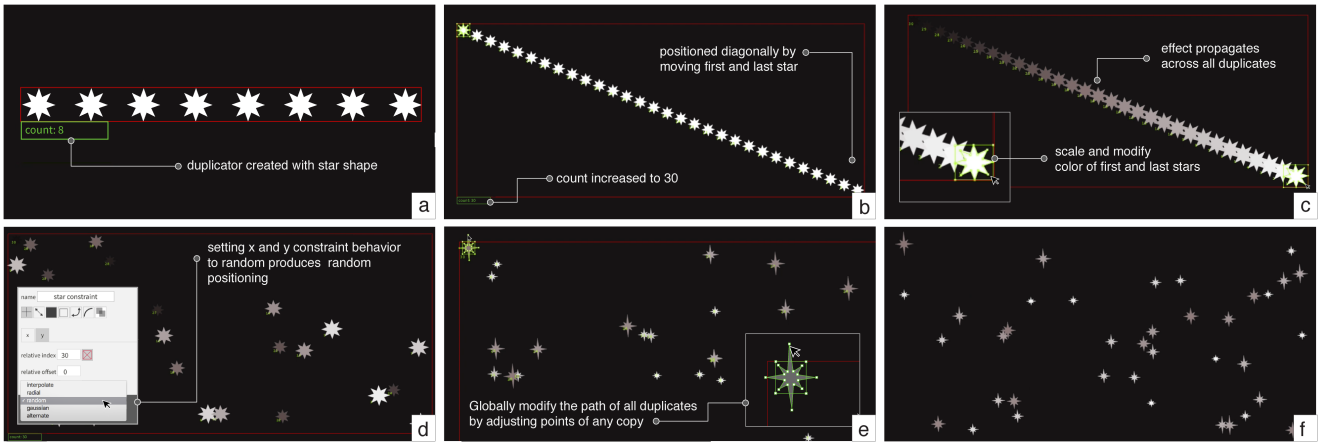

Figure 3. Random starfield creation: (a) duplicating a star, (b) creating a row of stars, (c) creating variation in scale and color, (d) producing a random distribution, (e) manually manipulating points of star, and (f) finished starfield.

list section of the document structure panel (fig 2 a). Manual transformations on a list are mapped to each member. For example, rotating a list will rotate each member around its own center, rather than the centroid as happens in geometric groups. This behavior allows *one-to-many constraints* where multiple objects are subject to one constraint. In the flower example, we can constrain the 14 small orange outer portions of each petal (fig 1) using a list and a single constraint.

Para's lists also facilitate iterative variation similar to that found in textually generated procedural art. This is achieved through *many-to-many constraints*: constraints in which both the reference and relative are lists. Many-to-many constraints create iterative mappings where each object in the relative list is constrained differently based on its index. This behavior is possible because unlike other tools [12, 17], Para's lists are ordered. The values to constrain the relative are calculated by interpolating across the values of the objects in the reference list. Figure 2 demonstrates how a many-to-many constraint can be used to dynamically modify the distribution of a pattern. In this example, the shapes of a repeating ellipse and diamond pattern are constrained on the x and y axes by a reference list containing two green rectangles (fig 2 a). The artist can update the distribution of the diamond ellipse pattern by modifying the position of the rectangles. In the process, the geometric offsets are maintained between each member of the reference and relative.

Different types of distributions can be created by altering the content of the reference list. If the reference list contains more than two members, a non-linear reference value set is produced (fig 2 c). These values are sampled from a Lagrange polynomial that is produced from the values of the objects of the reference list. Moving the second rectangle in the reference list up or down changes the polynomial and results in a curved distribution of the relative pattern. This interpolation technique can produce many different types of non-linear distributions across any property, including parabolas, waves, and ellipses, by adding more shapes to the reference list. Because interpolation does not support all distributions, Para allows artists to select different mapping behaviors for constraints from a drop-down menu in the constraint inspector (fig 2 c). Table 2 lists the current options and their calculation mechanisms and figure 2, b, e, and f demonstrate distributions derived from radial and random mappings. The constraint inspector also enables artists to modify the individual offsets for each member of the reference list using the relative index selector, or exclude a relative member from being affected using the exception button (fig 2 b).

Duplicators

Our analysis of procedural art demonstrated the importance of automatically creating new forms for generative compositions. Constraints and lists facilitate procedural relationships, but cannot generate new objects by themselves. To address this, we added duplicators to Para. Duplicators are specialized lists with the ability to declaratively control the number of objects they contain. Duplicators are initialized on a single object. Doing so generates or deletes new shapes when the count is increased or decreased. When first created, duplicators contain self-referencing constraints on all geometric and stylistic properties. Practically, this results in iterative variations across all members of a duplicator on any property by directly manipulating either the first or last objects. Updates to the number of copies of a duplicator will preserve these variations. Like other constraints, duplicator constraints can be modified using the constraint interface to change offsets, specify exceptions, and change constraint behavior for different properties and sub-properties. Duplicators also can act as references or relatives in manually created constraints. To preserve correspondence across the geometry of their members, duplicators enact a constraint-like behavior on the paths of all of the duplicates wherein changes to the points of any individual shape are propagated across the other copies. Duplicators build on the low-level functionality of lists and constraints to enable creation and management of distributions with many shapes, for example, the creation of a random star field in Figure 3. A single five-point star is drawn and duplicated (fig 3 a). The start and end copies are manually altered by position, scale, and fill color to produce a diagonal line of stars growing in size and brightness (fig 3 b). The x-position constraint mapping is changed to random, producing a vertical gradient from light to dark (fig 3 c). Changing the mapping of the y-position constraint to random results in

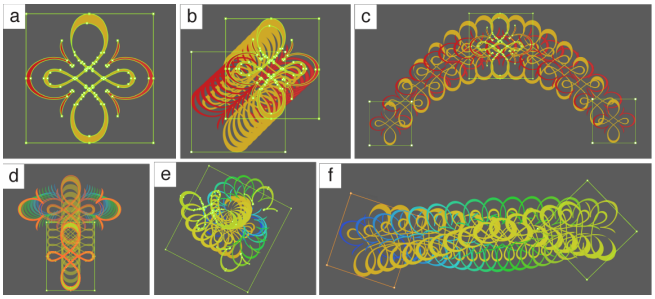

Figure 4. Group duplicators. (a) Group with three members. (b) Duplicator initialized on group. (c) Arc pattern produced with three references (selected). (d) modifying position/color of individual group members (orange). (e-f) variations from modifying group members.

a field of stars with a random distribution of brightness and scale (fig 3 d). The random distribution can be recalculated by manually dragging the first or last star, and the number of stars can be increased by modifying the count. The shape of each star can be altered by modifying the points of any individual star (fig 3 f). Geometric groups also can be converted into duplicators, thereby enabling more complex distributions (fig 4). Groups are a structural way to make compound objects and do not serve a procedural purpose on their own. For group duplicators, constraints are created between each respective member of a group across each copy to ensure that correspondence between each copy is maintained if the structure of a single group is altered (fig 4 d-f). In Para's internal representation, duplicator constraints are identical to many-to-many constraints. Duplicator constraints are between a reference list consisting of the first and last member of the duplicator, and a relative list consisting of the duplicator's children. Self-referencing is possible because objects in Para can be in multiple lists simultaneously.

EVALUATIONS

The evaluation of Para had two objectives: to evaluate the accessibility and expressiveness of Para in an open-ended setting and to understand how procedural tools could support manual artists. We conducted two studies targeting both objectives. The first evaluation was a breadth-based workshop with 11 artists, aimed at revealing the creative trade-offs between direct and textual tools. The second evaluation was an in-depth study with a single artist, aimed at evaluating Para's performance in realistic creative practice. Building from the the opportunities and challenges highlighted in our background research and our analysis of professional procedural artwork, we developed the following evaluation criteria:

Ease of use: Is it easy for novice programmers to use the tool? Can people understand the tool's artistic applications?

Creative outcomes: Does the tool support the creation of different procedural aesthetics? Does the tool enable creating variations of a piece?

Process: How does working with the tool compare to traditional art tools? Can artists integrate their prior expertise? Can people move between manual and procedural creation?

Reflection: Does the tool encourage thinking about procedural relationships? Does the tool affect how people think about making art?

Id	Background	Manual Art	Digital Art	Coding
p1	graphic design	4	5	1
p2	graphic design	4	4	1
p3	design	3	3	2
p4	printmaking	5	5	1
p5	art and CS	3	3	3
p6	illustration	4	4	1
p7	installation design	5	5	5
p8	architecture	5	5	1
p9	illustration	4	4	1
p10	art and CS education	4	4	5
p11	Product Design	4	5	2
Smith	painting graphic design	5	5	2

Table 3. Participant backgrounds and experience in manual art, digital art and coding (1 being no experience and 5 being expert.)

In each study we collected data through recorded discussions, surveys, and participant artwork. Surveys contained attitudinal questions relating to our evaluation criteria, using 5-point Likert scales, with 5 as the optimal response. Survey results, discussion transcripts, and artwork were analyzed with respect to our evaluation criteria. The majority of the results are qualitative, triangulated from open-ended survey responses and group or individual discussions. When scale data are used, we present the median and standard deviation.

Workshop study methodology

Our workshop was conducted with 11 participants from a range of art and design backgrounds. The majority were inexperienced programmers (table 3). The workshop lasted 14 hours over two days and covered Para and the textual procedural art tool Processing. We compared Para to Processing because Processing shares many of the same accessibility objectives as Para. Our goal was to compare Para's accessibility and expressiveness to textual procedural tools, therefore we did not compare Para to non-procedural design tools like Adobe Illustrator. Illustrator emphasizes manual drawing tools, which are not the focus of Para's innovation. Participants were first introduced to Para and given incremental demonstrations in the use of duplicators and groups, followed by a period for free experimentation. Participants then were introduced to Processing and provided instruction in syntax, drawing and transforming shapes, loops, mathematical expressions and conditionals and then given opportunities to freely experiment. Our instruction in Processing was based on approaches for beginners from the Processing Handbook [36], and demonstrated methods for producing designs that were similar to those possible using Para. Following the instruction period, participants had 3 hours to work on a piece of their choice with either or both tools. We also allowed participants to use other tools in conjunction with Processing and Para (e.g. Photoshop and Illustrator) for bitmap manipulation and detailed manual drawing, tasks not intended to be supported by Para or Processing.

Limitations

Our study required professionals to rely on prototype software with less sophisticated drawing tools than commercial software, which people found limiting. This is common in systems-building research. Although Processing was the most relevant textual procedural art tool to compare with Para, it also includes features for animation and interactivity. These features do not exist in Para and are, therefore,

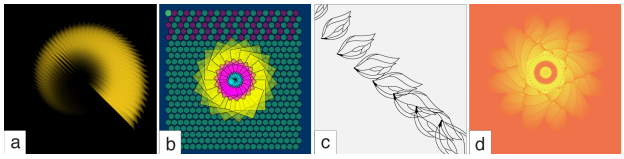

Figure 5. Artwork from workshop. Left: conditional-design from Processing. Right: Designs generated from multiple hexagonal duplicators from Para combined with a center motif from Processing.

not directly comparable. We compensated in our study by focusing on Processing's capabilities for static procedural art. Para's cycle-prevention feature was not implemented prior to our workshops; however, we notified participants of the issue, and with few exceptions, people did not create cycles. Our in-depth study surveys the experiences of one person; however, evaluations with additional artists from different backgrounds would likely reveal additional insights. We believe the variety of aesthetics and styles produced by one person suggests the potential of our approach to support a variety of artists. Finally, it is challenging to measure expressiveness because different artists are expressive in different ways. We therefore present the artwork created with our system in order to provide concrete examples of its expressiveness.

Workshop results

Overall results indicate that participants found the direct-manipulation interface of Para more intuitive to use than textual programming in Processing. People who specialized in graphic design and illustration prior to the workshop felt that Para fit well with their current practices and they were interested in using it in the future. People with backgrounds outside of graphic art and illustration were less interested. Participants felt that textual code could provide greater control for experienced programmers. In the following section, we detail these results in the context of our evaluation criteria.

Ease of use: All participants were able to follow our instructions for using Para but struggled to understand our instructions for Processing. These observations were substantiated by survey responses: seven participants stated that they felt Para was easy to learn, and three felt Processing was easy to learn. Many participants said the difficulty of Processing increased their desire to practice it when they had the benefit of instructor support. Conversely, participants said Para was familiar and felt confident they could learn it independently.

Process: All participants experimented with both Processing and Para throughout the workshop. Participants exhibited a mix of responses with regard to how Processing fit with their prior art practice (mean:3.6, std:0.92); however, nearly all participants expressed interest in using the tool in the future (mean: 4.3, std:0.47). In survey and verbal responses, participants indicated they struggled with understanding the logic of Processing programs. Several people described their process as one of trial and error. In addition, people were frustrated by not being able to adjust artwork manually. Despite the challenges in learning Processing, several people talked about how they could exercise greater control and perform tasks not possible in Para. Four participants stated that Para fit well with their existing practice, and six (including the first four) stated that they could see themselves using Para in future work. People who had expertise in manual art and minimal programming experience were the most interested in using Para in the future. For people with interest in using procedural tools for 3D design and interactivity, Para was less relevant by design. Primary frustrations with Para centered on reduced sophistication of the direct-manipulation tools in comparison with commercial equivalents.

Creative outcomes: Because of time constraints, participants were not able to create polished artwork with either Para or Processing, although participants made a series of sketches that suggested different directions for more sophisticated work (fig 5). Participants who relied exclusively on Processing primarily created variations of a radial distribution example we presented in the instruction, incorporating randomness on opacity, scale, or rotation of shapes (fig 5 a,b). Participants who used Para exhibited a range of approaches: one person used portions of an illustration created prior to the workshop in a duplicated pattern (fig 5 c), while another experimented with the duplication functionality to simulate 3D rotated columns. Another participant generated a series of incrementally rotated geometric shapes. Using Illustrator, he combined these forms with a radial pattern from Processing (fig 5 d). Eight people stated that Processing enabled them to create things they would have difficulty creating otherwise (mean:4, std:1). Ten people stated that Para enabled them to create things they would have difficulty making otherwise (mean:4.1, std:0.83).

Reflection: In discussions and surveys, most participants felt that textual programming could be a powerful tool for art; however, they had a mix of reactions to using Processing. For several participants, using Processing underscored their prior associations of textual programming as inaccessible. The majority of participants felt that they would have to practice extensively with Processing to create sophisticated projects. In discussion and open-ended responses, participants expressed greater levels of confidence using Para because it had features similar to digital illustration programs. Others described Para's interface as more intuitive than Processing. Several people wrote that they found the live feedback in Para helpful; however, these responses were solely from people with prior experience with textual programming tools. None of the people new to programming commented on the live aspects, suggesting that people who only use direct manipulation may take liveness as a given in digital tools. Several participants talked about using Para and Processing for different purposes. One participant said Para was ideal for creating complex static illustrations, but Processing offered the opportunity for interactive pieces. Another participant said that Para and Processing were appropriate for different points in her artistic process, and said she would use Processing if she had a clear goal in mind.

In-depth study methodology

The workshop underscored the learnability of our system but did not demonstrate how Para performs in real art practice. We performed a second study to understand what kinds of artwork a person could create with Para through extended use. We commissioned the professional artist Kim Smith to use

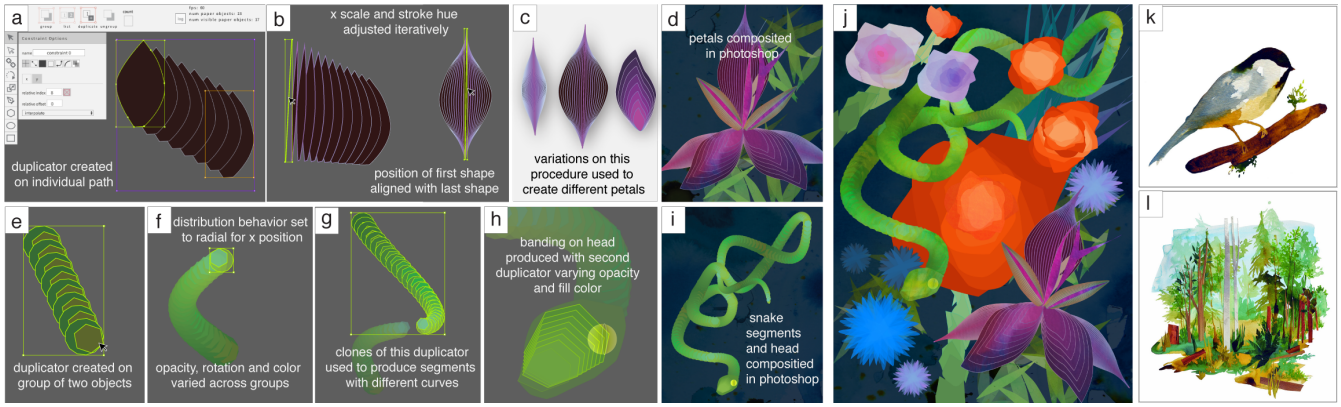

Figure 6. a-j Illustrative work with para: (a-d) process for creating orchid, (e-i) process for creating snake, (j) finished piece, composited in Photoshop and textured using image overlay. k-l examples of Smith's illustrations prior to study. ©Kim Smith.

Para for two weeks to create several pieces of art. Smith has extensive experience in abstract painting (fig 7 a), realistic illustration (fig 6 k,l), and graphic design. She is an expert with both manual and digital tools (table 3). She had an interest in applying procedural techniques in her work but was reluctant to invest the time needed to learn programming.

We met with Smith in person six times over the course of the study, every 2-3 days, for 1-2 hours apiece. First, we introduced her to Para in a 1-hour training session. In later meetings we reviewed the artwork she produced and discussed her experience. We gave her the choice of what to create during the study, but suggested experimenting with abstract and illustrated styles. Similar to the workshop, we allowed her to incorporate other digital tools into her work with Para. In the results, we distinguish between portions of artwork produced with Para and those produced by other means. In addition to written surveys and interviews, Smith wrote short reflections after every working session. We used automated surveys within Para to assess her experience while using the system [40]. Automated dialogs appeared every 20 minutes asking questions about her current task, goals and difficulties, and a snapshot of her work was saved. Throughout the study, we used Smith's feedback to iterate on Para's functionality.

In-depth study results

Smith found Para to be intuitive and felt she could use it more effectively and with greater confidence than textual programming and parametric CAD tools. She produced seven finished pieces and applied Para to abstract and illustrative styles. In her artwork, she used duplicators and iteratively constrained lists, but did not find one-to-one constraints as useful. Smith felt Para was compatible with her painting practice and also offered new opportunities for creative exploration. Using Para altered the way she thought about her artistic process, and *she requested to continue using the software afterwards.*

Ease of use: Smith's feedback from the experience sampling indicated that she struggled initially with Para's vector selection and editing tools, which were less refined than those in commercial software. Otherwise, she described the Para interface as accessible and enjoyable. Specifically, she said the emphasis on visual manipulation was "welcoming and friendly," and contrasted this with using complex CAD software and textual programming tools. She also stated that Para made her "comfortable starting with a blank canvas" because the visual interface gave her confidence that she could produce something, and develop a process. When asked about the minimal procedural feature set, she said,"I think there's a lot of options in a very small parameter set."

Smith contrasted her use of Para with her use of textual programming tools: *It's like this big wall that is slowing me down. I would imagine if I was good with Processing there would be a lot more I could do with it... But the entry point for me is high. For a long time it's going to be in my way.*

Smith found some procedural aspects of Para to be difficult. Initially she struggled with understanding the order in which shapes should be selected for one-to-one constraints. We modified the constraint tool to automatically pull up the property selection interface for selected objects when the tool was enabled. Smith said these revisions made the tool more intuitive. Although she experimented with one-to-one constraints and felt they might be useful for other artists, Smith never found a way to use them in her work. Smith also encountered organizational challenges when creating extremely complex work. At times it was difficult for her to identify and select specific procedural relationships. We partially addressed this by adding the ability to label constraints, lists, and geometry; however, organization remained an issue. In discussion, Smith pointed out how organization was also a challenge in non-procedural digital art tools, but felt that organizational mechanisms were particularly important for tools like Para because they made it easier to quickly build complexity.

Creative Outcomes: In week 1, Smith created four abstract pieces using Para (see examples in (fig 7 b,d). She produced them by drawing vector shapes in Para, and used duplicators to create multiple copies with iterative changes in color, scale, and position. In week 2, Smith made two more abstract pieces. These incorporated procedural variation of opacity and blend modes, and constrained lists fig 7 c). She also made an illustration (fig 6 j), that applied duplicators, lists, and iterative constraints to the creation of flowers (fig 6 a-d) and a snake (fig 6 e-i). In abstract and illustrative pieces, Smith used Photoshop to add additional color and texture by using painted imagery as a transparent overlay and background.

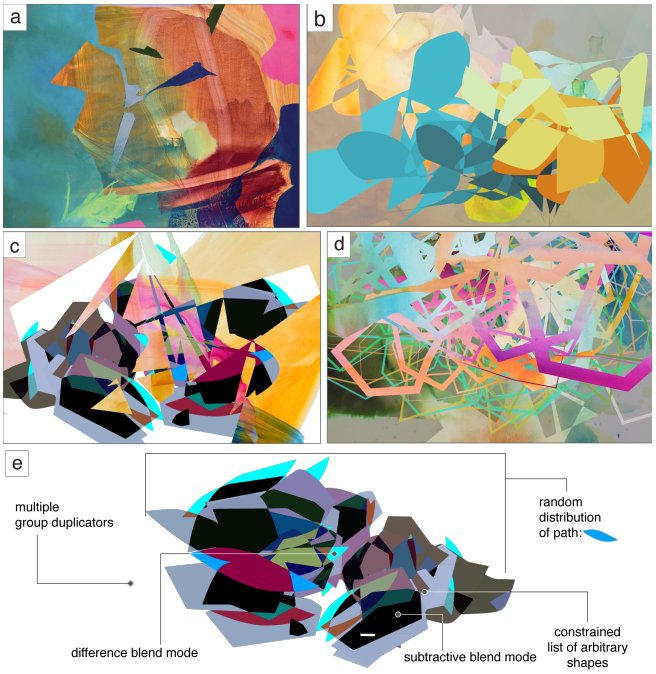

Figure 7. Comparison between Smith's prior art and abstract work created with Para. (a) sample of prior art, (b-d) finished pieces containing procedurally created forms overlaid with hand-painted imagery in Photoshop, (e) composition created in Para to produce c. ©Kim Smith.

Process: For her abstract work, Smith described her use of Para as "exploratory" and "experimental." She felt the unexpected effects and forms possible through Para's generative features (random and Gausssian mappings) were an advantage because they generated starting points for new work: *There is an element of applying a procedure and not knowing exactly what will happen . . . and this is a really great quality about Para. This software allows for an experimental process . . . In some ways, it mirrors the experience of painting for me, in that frequently I am open to chance and random occurrences within my materials and techniques.*

Smith associated the exploration possible with Para with traditional art media, stating: *In a lot of ways [Para] feels similar to traditional tools. The freedom to implement quickly, understanding what's happening, but not needing to have the foresight ahead of time [allows for] a more sculptural way of creating.*

Because she was interested in variations in color and layering, Smith requested support for opacity and blend modes[3], which were features she relied on in Photoshop. We added UI components for opacity and blend mode that matched those in Photoshop and made it possible to iteratively constrain these properties. Smith incorporated these features into her abstract work (fig 7 c,e) and described how procedural control of opacity and blend enabled her to use subtractive masks and filters in a dynamic fashion. The ability to quickly and effectively incorporate features from Photoshop into the procedural structure of Para demonstrated how our system could extend existing digital art tools.

[3]Blend mode determines how an object is composited with those beneath it.

Smith initially focused on producing abstract work because this was representative of her fine-art practice. She felt Para was useful for illustrative work, but felt the goal-oriented nature of illustration required using the system with greater control. The lack of refined drawing tools made the illustrative work laborious at times. While Smith found constraints and duplicators useful in both abstract and illustrative work, she used other procedural aspects less. Lists were confusing at first, and it was labor-intensive to manually constrain them. We simplified lists to function similar to duplicators with automatically created self-referencing constraints. Smith said this modification corresponded with her objectives, and applied the new lists in her illustrations (fig 6 j) and abstract work (fig 7 c). Producing numerous variations with Para led Smith to identify the need for versioning functionality.

Reflection: Smith reflected on how Para was uniquely suited for combining abstract and illustrative forms: *I think there's a space between abstraction and representation*[4] *. . . this ambiguous in-between space where Para could be very useful.*

She also described how the process behind a work of art was as important as the work itself: *I want to create things that surpass the tool and it's not so obvious that some other process did all the interesting work.*

As a result, she felt more successful when using Para in a way that obscured the use of obvious procedural aesthetics. In the closing interview, Smith described how working with Para had changed the way she thought about her own art process: *The inclusion of Para changes not just the visual output. It changes the process, and that's a lot of what I think about — how the tool changes the way you make things.*

DISCUSSION

The components of Para provide one entry point into procedural art. Here we discuss the creative opportunities that resulted from the design of those components through three principle questions. Was Para compatible with the skills of manual artists? Did Para offer meaningful creative opportunities? What are the limitations of direct manipulation? In addressing these questions, we provide the following recommendations for building expressive direct manipulation procedural tools: 1) Extending manual drawing tools with declarative relationships allows experienced artists to leverage existing skills. 2) It is possible to preserve expressiveness in direct-manipulation procedural tools by designing the tool around a small number of procedural concepts that can be combined in different ways to produce different outcomes. 3) Procedural tools with concretely represented relationships can enable productive exploration of form, color, and composition for professional artists.

Was Para compatible with the skills of manual artists?

The design of Para was partially based on the theory that a direct-manipulation procedural tool would be compatible with manual practice and enable manual artists to leverage existing skills. Two points of evidence support this theory. First, there are clear stylistic similarities between Smith's

[4]Referring to abstract vs realistic (illustrative) imagery.

prior work and the pieces she created with Para, despite being produced with different tools. The forms and compositions in her abstract paintings mirror her abstract work in Para, and her use of color and blending in her watercolor illustrations corresponds with her Para illustrations. These commonalities suggest that in addition to making use of its procedural affordances, Smith was able to transfer her prior skills and practices to Para. Second, both Smith and participants in the workshop described Para as "intuitive", "enjoyable" and "familiar," in contrast to their experience with textual tools.

The intuitive qualities of Para are important because, aside from supporting accessibility, intuitive tools support specific creative practices. People who think through movement, intuition, and visual impression require computational tools that validate intuitive and relational mindsets in order to be personally expressive [49]. In line with this, Smith and many of our workshop participants stated that intuition played an important role in their art. This suggests that direct-manipulation procedural tools offer two ways to extend manual practices: by enabling intuitive manipulation of procedural relationships and by providing procedural techniques that are aligned with both the affordances and interaction paradigms of conventional graphic art tools.

Did Para offer meaningful creative opportunities?

A common trade-off in tool design is that the more accessible tools are, the less expressive they become. However, the results of our studies demonstrate that Para's accessibility *did not* prohibit expressive creation by professionals. Although our system contained a minimal number of procedural concepts, the artwork people created contained procedural forms and patterns comparable to those created with textual tools. The sketches produced in the workshop (fig 5 d) demonstrated radial distributions comparable in complexity and appearance to those created in Processing (fig 5 a-b). Smith's illustration demonstrates parametric modulation in color, scale, and rotation across repeated forms, and her illustrative and abstract work contains generative forms and patterns. Parametric variation, generative form, and complexity via repetition are considered to be primary affordances of applying programming to art and design [38].

Para also offered new creative opportunities by facilitating new processes for making art. Smith described how the ability to rapidly produce and manipulate complex compositions facilitated exploration and iteration. She also felt that the generativity in Para led to new ways of thinking about the creative process, describing it as "a way of expanding one's creative mind". Finally, Smith identified unique creative opportunities in combining procedural exploration with manual control, which she noted were unique to Para. Overall study results and participant artwork demonstrate how a small number of reconfigurable procedural concepts can support new processes and aesthetics through procedural exploration of form, color, and composition, as well as support new ways of thinking about the creative process itself.

What are the limitations of direct manipulation?

Despite the intuitive qualities of direct-manipulation tools, they are limited in some ways compared to textual tools. In Para, people sometimes struggled with interpreting and controlling complex relationships. Symbolic textual expressions can concisely and unambiguously represent complex relationships in ways that images cannot [26]. People also became frustrated attempting to make precise patterns through direct selection, or in managing the organization of particularly complex compositions. In comparison to Para, and direct-manipulation software in general, textual programming tools can better support numerical evaluation and accuracy. Furthermore, computational abstraction as expressed through textual tools can greatly aid in complex organizational tasks.

While the representational nature of programming can create a barrier for manual creation, it also offers opportunities for reflection. Code can simultaneously serve as a functional object and as a record of ideas and process [22]; thus, writing code can enable active reflection on the relationships that define one's work. Understanding abstraction is difficult [52], and our studies demonstrate that on its own, direct manipulation does not lead to the understanding of representational concepts. However, given the confidence people expressed when using Para, direct manipulation of procedural relationships may provide a way to make the challenge of learning abstract representational tools more approachable. Alan Kay theorized that direct manipulation might aid in the learning process of representational systems by helping people to develop mental maps of abstract concepts [16]. With further development, tools like Para could assist manual artists in the process of understanding computational abstraction and benefiting from its application to their art. Overall we believe there is potential expand the expressiveness of systems like Para through close integration of direct manipulation and textual programming paradigms. Systems that enable artists to express relationships textually and then experiment with them through direct manipulation may scaffold learning and support new forms of creative expression.

CONCLUSION AND FUTURE WORK

Through development and evaluation of a direct manipulation procedural art tool, we demonstrate that it is possible to support concrete and manual creation while simultaneously enabling expressive forms of procedural design. Going forward, we see additional opportunities to evaluate procedural direct manipulation within the context of self efficacy and our system's correspondence with the mental models of visual artists. We are also excited about the creative opportunities of procedural direct manipulation, and the potential for future systems that integrate this approach with other forms of programming. Overall, we see Para as a platform for developing new technologies that further support the combination of manual and procedural creation.

ACKNOWLEDGMENTS

We thank all the artists that participated in our workshop. Special thanks to Kim Smith, and also to Michael Craig for contributing to development. We would also like to thank N. Gillian, R. Jacobs, D. Mellis, A. Zoran, E. Natzke, D. Ito, the Dynamic Medium Group, and M. Resnick and the Lifelong Kindergarten Group.

REFERENCES

1. Cycling '74. 2016. Max. http://cycling74.com/products/max
2. Robert Aish. 2012. DesignScript: origins, explanation, illustration. In *Computational Design Modelling*. Springer, 1–8.
3. Arduino. 2017. Arduino Introduction. http://www.arduino.cc/en/Guide/Introduction
4. J. Ashkenas and DocumentCloud. 2010. Backbone.js. http://backbonejs.org
5. Alan F. Blackwell. 2014. Palimpsest: A layered language for exploratory image processing. *Journal of Visual Languages Computing* 25, 5 (2014), 545 – 571. DOI: http://dx.doi.org/10.1016/j.jvlc.2014.07.001
6. Ravi Chugh, Jacob Albers, and Mitchell Spradlin. 2015. Program Synthesis for Direct Manipulation Interfaces. *CoRR* abs/1507.02988 (2015). http://arxiv.org/abs/1507.02988
7. Scott Davidson. 2007. Grasshopper. http://www.grasshopper3d.com.
8. Experimental Media Research Group. 2004. NodeBox. http://www.nodebox.net
9. Björn Hartmann, Loren Yu, Abel Allison, Yeonsoo Yang, and Scott R. Klemmer. 2008. Design As Exploration: Creating Interface Alternatives Through Parallel Authoring and Runtime Tuning. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology (UIST '08)*. ACM, New York, NY, USA, 91–100. http://doi.acm.org/10.1145/1449715.1449732
10. B. Harvey. 1991. Symbolic Programming vs. the A.P. Curriculum. *The Computing Teacher* 56 (February 1991), 27–29.
11. Brian Hempel and Ravi Chugh. 2016. Semi-Automated SVG Programming via Direct Manipulation. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST '16)*. ACM, New York, NY, USA, 379–390. DOI: http://dx.doi.org/10.1145/2984511.2984575
12. Raphaël Hoarau and Stéphane Conversy. 2012. Augmenting the Scope of Interactions with Implicit and Explicit Graphical Structures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 1937–1946. DOI: http://dx.doi.org/10.1145/2207676.2208337
13. MakerBot Industries. 2015. Thingiverse Customizer. http://www.thingiverse.com/apps/customizer
14. Jennifer Jacobs and Leah Buechley. 2013. Codeable Objects: Computational Design and Digital Fabrication for Novice Programmers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 1589–1598.
15. Jennifer Jacobs, Mitchel Resnick, and Leah Buechley. 2014. Dresscode: supporting youth in computational design and making. In *Constructionism*. Vienna, Austria.
16. Alan C. Kay. 1990. *User Interface: A Personal View*. Addison-Wesley. 191–207 pages.
17. Rubaiat Habib Kazi, Fanny Chevalier, Tovi Grossman, and George Fitzmaurice. 2014. Kitty: Sketching Dynamic and Interactive Illustrations. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology (UIST '14)*. ACM, New York, NY, USA, 11.
18. Rubaiat Habib Kazi, Tovi Grossman, Nobuyuki Umetani, and George Fitzmaurice. 2016. SKUID: Sketching Dynamic Drawings Using the Principles of 2D Animation. In *ACM SIGGRAPH 2016 Talks (SIGGRAPH '16)*. ACM, New York, NY, USA, Article 84, 1 pages. DOI: http://dx.doi.org/10.1145/2897839.2927410
19. Scott R. Klemmer, Björn Hartmann, and Leila Takayama. 2006. How Bodies Matter: Five Themes for Interaction Design. In *Proceedings of the 6th Conference on Designing Interactive Systems (DIS '06)*. ACM, New York, NY, USA.
20. Andrew J. Ko, Brad A. Myers, and Htet Htet Aung. 2004. Six Learning Barriers in End-User Programming Systems. In *Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing (VLHCC '04)*. IEEE Computer Society, Washington, DC, USA, 199–206. http://dx.doi.org/10.1109/VLHCC.2004.47
21. J. Lehni and J. Puckey. 2011. Paper.js. http://paperjs.org/
22. Golan Levin. 2003. Essay for Creative Code. http://www.flong.com/texts/essays/essay_creative_code
23. Z. Lieberman, T. Watson, and A. Castro. 2015. openFrameworks. http://openframeworks.cc/about
24. Maryam M. Maleki, Robert F. Woodbury, and Carman Neustaedter. 2014. Liveness, Localization and Lookahead: Interaction Elements for Parametric Design. In *Proceedings of the 2014 Conference on Designing Interactive Systems (DIS '14)*. ACM, New York, NY, USA.
25. John H. Maloney and Randall B. Smith. 1995. Directness and liveness in the morphic user interface construction environment. In *Proceedings of the 8th annual ACM symposium on User interface and software technology*. ACM.
26. M. McCullough. 1996. *Abstracting Craft: The Practiced Digital Hand*. The MIT Press, Cambridge, Massachusetts.
27. W.J. Mitchell. 1990. *The Logic of Architecture: Design, Computation, and Cognition*. MIT Press, Cambridge, MA, USA.

28. Brad Myers, Scott E Hudson, and Randy Pausch. 2000. Past, present, and future of user interface software tools. *ACM Transactions on Computer-Human Interaction (TOCHI)* 7, 1 (2000), 3–28.
29. Brad A Myers. 1990. Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing* 1, 1 (1990), 97–123.
30. Lora Oehlberg, Wesley Willett, and Wendy E. Mackay. 2015. Patterns of Physical Design Remixing in Online Maker Communities. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. ACM, New York, NY, USA.
31. Stephen Oney, Brad Myers, and Joel Brandt. 2012. ConstraintJS: Programming Interactive Behaviors for the Web by Integrating Constraints and States. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology (UIST '12)*. ACM, New York, NY, USA.
32. Stephen Oney, Brad Myers, and Joel Brandt. 2014. InterState: A Language and Environment for Expressing Interface Behavior. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology (UIST '14)*. ACM, New York, NY, USA.
33. Fatih Kursat Ozenc, Miso Kim, John Zimmerman, Stephen Oney, and Brad Myers. 2010. How to Support Designers in Getting Hold of the Immaterial Material of Software. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. ACM, New York, NY, USA.
34. S. Papert. 1980. *Mindstorms: children, computers, and powerful ideas*. Basic Books.
35. C. Reas and B. Fry. 2004. Processing. http://processing.org
36. C. Reas and B. Fry. 2007. *The Processing Handbook*. MIT Press, Cambridge, Massachusetts, USA.
37. C. Reas and B. Fry. 2016. Processing Overview. http://processing.org/overview
38. C. Reas, C. McWilliams, and LUST. 2010. *Form and Code*. Princeton Architectural Press, New York, NY, USA.
39. Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. 2009. Scratch: Programming for All. *Commun. ACM* 52, 11 (Nov. 2009).
40. Mitchel Resnick and Brian Silverman. 2005. Some Reflections on Designing Construction Kits for Kids. In *Proceedings of the 2005 Conference on Interaction Design and Children (IDC '05)*. ACM, New York, NY, USA.
41. David Roedl, Shaowen Bardzell, and Jeffrey Bardzell. 2015. Sustainable Making? Balancing Optimism and Criticism in HCI Discourse. *ACM Trans. Comput.-Hum. Interact.* 22, 3 (June 2015).
42. J. Rosenkrantz and J. Louis-Rosenberg. 2015. Nervous System Tools. http://n-e-r-v-o-u-s.com/tools
43. Toby Schachman. 2012. Alternative Programming Interfaces for Alternative Programmers. In *Proceedings of the ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! 2012)*. ACM, New York, NY, USA.
44. Toby Schachman. 2015. Apparatus: a hybrid graphics editor / programming environment for creating interactive diagrams. In *Strange Loop*.
45. Donald Schön and John Bennett. 1996. Bringing Design to Software. ACM, New York, NY, USA, Chapter Reflective Conversation with Materials.
46. Ben Shneiderman. 1984. The Future of Interactive Systems and the Emergence of Direct Manipulation. In *Proc. Of the NYU Symposium on User Interfaces on Human Factors and Interactive Computer Systems*. Ablex Publishing Corp., Norwood, NJ, USA, 1–28. http://dl.acm.org/citation.cfm?id=2092.2093
47. Ivan E. Sutherland. 1998. Seminal Graphics. ACM, New York, NY, USA, Chapter Sketchpad: a Man-machine Graphical Communication System, 391–408. http://doi.acm.org/10.1145/280811.281031
48. Cesar Torres and Eric Paulos. 2015. MetaMorphe: Designing Expressive 3D Models for Digital Fabrication. In *Proceedings of the 2015 ACM SIGCHI Conference on Creativity and Cognition (C&C '15)*. ACM, New York, NY, USA.
49. S. Turkle and S. Papert. 1992. Epistemological Pluralism and the Revaluation of the Concrete. *Journal of Mathematical Behavior* 11, 1 (March 1992).
50. B. Victor. 2011. Dynamic Pictures. http://worrydream.com/DynamicPicturesMotivation.
51. B. Victor. 2012. Learnable Programming: Designing a programming system for understanding programs. http://worrydream.com/LearnableProgramming.
52. B. Victor. 2013a. Drawing Dynamic Data Visualizations Addendum. http://worrydream.com/DrawingDynamicVisualizationsTalkAddendum.
53. B. Victor. 2013b. Drawing Dynamic Data Visualizations (Talk). http://vimeo.com/66085662.
54. Haijun Xia, Bruno Araujo, Tovi Grossman, and Daniel Wigdor. 2016. Object-Oriented Drawing. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 4610–4621. DOI: http://dx.doi.org/10.1145/2858036.2858075