Variational Pictures

Martin Erwig and Karl Smeltzer(✉)

Oregon State University, Corvallis, USA
{erwig,smeltzek}@oregonstate.edu

Abstract. Diagrams and pictures are a powerful medium to communicate ideas, designs, and art. However, authors of pictures are forced to use rudimentary and ad hoc techniques in managing multiple variants of their creations, such as copying and renaming files or abusing layers in an advanced graphical editing tool. We propose a model of variational pictures as a basis for the design of editors and other tools for managing variation in pictures. This model enjoys a number of theoretical properties that support exploratory graphical design and can help systematize picture creators' workflows.

1 Introduction

Visual media such as diagrams and pictures are ubiquitous in the modern world. These take many forms and are employed by a variety of professionals, ranging from architects and industrial designers using CAD tools, graphic designers and photographers using photo editing tools, to scientists and business owners creating charts and technical diagrams to analyze and share data.

While the software tools for these applications are quite sophisticated, they offer little or no support for managing variation in the produced artifacts, forcing users to employ rudimentary techniques to manage multiple versions of their work. For example, a graphic designer who might want to showcase changes to a logo design might be forced to overuse the layer system in their editing tools or simply create multiple copies of the picture file. A data scientist generating a series of similar or related charts and tables might have to manually copy files or images and rename them meaningfully to be able to view and compare them. Architects and engineers frequently make revisions to their drawings and designs and their tools offer minimal, if any support, forcing them to adopt ad hoc approaches.

The need for variation comes in two different forms. First, one might want to create several concurrent variants of an artifact. Second, one might need some kind of version control for pictures. We propose *variational pictures* as an underlying model to support both forms of picture variation. A variational picture is simply a picture (here a grid of pixel values) that offers an explicit way of representing and selecting different variant pictures. For example, a picture

This work is partially supported by the National Science Foundation under the grants IIS-1314384 and CCF-1717300.

P. Chapman et al. (Eds.): Diagrams 2018, LNAI 10871, pp. 55–70, 2018.
https://doi.org/10.1007/978-3-319-91376-6_9

along with its history (or undo) states can be viewed as a variational picture. Similarly, a collection of related but different designs is also a variational picture. It is even possible to view an animation as a variational picture in which the variants are the frames with a temporal ordering.

Consider a landscape architect working on the plan for a new city park, the features of which are not fully decided yet. For example, an additional wooded area may be cleared to make more space for the park if the budget is determined to allow for it. In another part of the park, a pavilion may be added to provide covered seating. Finally, on the condition that the trees are cleared, a small fountain may be installed where they were. Even with such a small number of undetermined features, this already means there are six possible park layouts. It is easy to imagine this growing out of control rather quickly with more options. By using a model of variational pictures, the architect could produce a single plan with the areas of variability clearly marked that also allows toggling between the options to show them to the final decision makers. We show a sequence of such variational pictures in Fig. 1.

The example does not only show that variational pictures are useful, it also illustrates that managing variability is a nontrivial matter that requires a number of operations for creating, eliminating, and adapting variability. We will return to this example later.

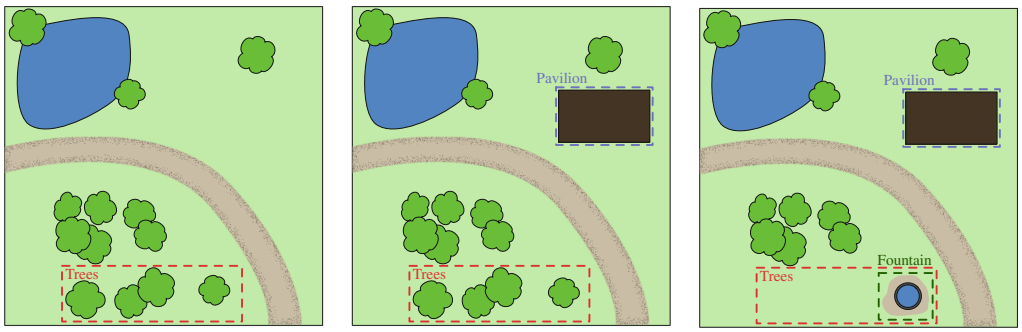


Fig. 1. A sequence of variational pictures showing the design of a park. In (a) we have one area of variability for the potential removal of trees, in (b) we have an independent dimension for a pavilion area, and in (c) we have a nested dimension for a small fountain that can only exist if the trees are removed.

Since variational pictures have a significantly more complicated structure than plain pictures, their precise meaning should be captured in a formal model that can serve as a specification for guiding the implementation of editors and other tools. In particular, a variational picture model should explain the exact behavior of operations for creating, modifying, and navigating variational pictures, as well as for the splitting and merging of variational pictures.

In the following we will present a model for variational pictures and show how it supports editors and other tools for creating and managing variational pictures. Specifically, this work makes the following contributions:

1. A *model of variational pictures* based on the choice calculus [1]. We will present the required background of the choice calculus and how it is used to define a model of variational pictures in Sect. 2.
2. A set of *properties* that describe the generality and usefulness of the presented variational picture model. We will present these properties in Sect. 3.
3. *Requirements* for maintaining variational pictures and a demonstration how the presented model satisfies these requirements. We will discuss the requirements and how they are supported by our variational picture model in Sect. 4.
4. *Variational area trees* to explore and navigate variational pictures diagrammatically, which are presented in Sect. 5.

In Sect. 6 we discuss related work, and we present conclusions and directions for future work in Sect. 7.

2 Representing Variational Pictures

The definition of variational pictures is based on a generic model of variational values that is discussed in Sect. 2.1, which we then apply to a model of plain pictures described in Sect. 2.2. The resulting model of variational pictures is then presented in Sect. 2.3. The different degrees of variability in a variational picture give rise to a notion of variation type, introduced in Sect. 2.4, and a corresponding notion of region, which is discussed in Sect. 2.5. Finally, in Sect. 2.6 we introduce an operation that can create variational pictures automatically from a sequence of plain pictures.

2.1 A Formal Model of Variation

In order to model and provide structure to variational values we make use of the choice calculus [1], a formal system for managing variation based on the core notion of a choice.

Choices represent sets of *alternatives* associated with names called *dimensions*. For example, we can define a variational integer $A\langle 1, 2\rangle$ as a choice between the two values 1 and 2. In this paper we consider only binary choices, that is, choices of two alternatives. This is not an essential restriction, since choices can be nested to represent a larger number of alternatives. For example, the variational integer $vx = A\langle B\langle 1, 2\rangle, 3\rangle$ contains three total alternatives and has an outer choice in dimension A and a nested choice in dimension B. Choice expressions such as this form binary tree structures where n dimension names in the internal nodes lead to $n + 1$ plain values at the leaves.

Each dimension D in a choice expression gives rise to two selectors $D.l$ and $D.r$. Selectors can be used to extract alternatives from choices via a *selection*

operation defined as follows (we use x to range over plain, non-variational values, vl and vr to range over (potentially) variational values, and s to range over selectors):

$$\lfloor D\langle vl, vr\rangle \rfloor_s = \begin{cases} \lfloor vl \rfloor_s & \text{if } s = D.l \\ \lfloor vr \rfloor_s & \text{if } s = D.r \\ D\langle \lfloor vl \rfloor_s, \lfloor vr \rfloor_s \rangle & \text{otherwise} \end{cases}$$
$$\lfloor x \rfloor_s = x$$

For example, $\lfloor A\langle 1, 2\rangle \rfloor_{A.l} = 1$, $\lfloor vx \rfloor_{A.l} = B\langle 1, 2\rangle$ and $\lfloor vx \rfloor_{B.r} = A\langle 2, 3\rangle$. Dimensions synchronize choices, which means selecting an alternative in dimension D selects that same alternative in all occurrences of D. For example, we have $\lfloor B\langle A\langle 1, 2\rangle, A\langle 3, 4\rangle\rangle \rfloor_{A.l} = B\langle 1, 3\rangle$.

Since one selector can eliminate only choices in one dimension, we generally need a set of selectors, called a *decision*, to extract a plain value from a variational one. This is done by repeated selection with the selectors from the decisions. With $\delta = \{s_1, s_2, \ldots, s_n\}$, we have:

$$\lfloor vx \rfloor_\delta = \lfloor \lfloor \ldots \lfloor vx \rfloor_{s_1} \ldots \rfloor_{s_{n-1}} \rfloor_{s_n}$$

A decision that eliminates all choices from a variational value is said to be *complete*. The order of selection is irrelevant. That is, for any variational value vx and pair of selectors $D.d$ and $D'.d'$ (with $d, d' \in \{l, r\}$):

$$D \neq D' \implies \lfloor \lfloor vx \rfloor_{D'.d'} \rfloor_{D.d} = \lfloor \lfloor vx \rfloor_{D.d} \rfloor_{D'.d'}$$

This will be an important property for variational pictures. As areas of variability will be represented by choices, we can make decisions about optional features in pictures in any order. In the example from the Introduction we could decide, for example, about the fountain before or after we decide about removing the trees.

However, the nesting of the choices *does* matter because it defines dependencies among the decisions to be made. In that same example, having the fountain area nested inside the tree area means that it is possible the outer one will make the inner one irrelevant. This has important consequences for the design of a variational picture editor.

The semantics of a choice calculus expression is given as a mapping from decisions to plain expressions. (This will be made precise in Sect. 2.3.) In general, one variational value can be represented by different choice calculus expressions, which gives rise to a notion of equivalence for expressions that denote the same variational values. For example, choice synchronization is the reason for the choice domination laws that allow the elimination of nested choices in the same dimension:

$$D\langle D\langle vx, vy\rangle, vz\rangle \equiv D\langle vx, vz\rangle \qquad D\langle vx, D\langle vy, vz\rangle\rangle \equiv D\langle vx, vz\rangle$$

Moreover, idempotent choices have no effect on variability:

$$D\langle vx, vx\rangle = vx$$

We also have choice commutation rules, that is, for $D \neq D'$:

$$D'\langle D\langle x, y\rangle, z\rangle \equiv D\langle D'\langle x, z\rangle, D'\langle y, z\rangle\rangle$$
$$D'\langle x, D\langle y, z\rangle\rangle \equiv D\langle D'\langle x, y\rangle, D'\langle x, z\rangle\rangle$$

Although choice commutation preserves the choice calculus semantics, it is still an important consideration for variational pictures, since the nesting of choices relates directly to their structure.

2.2 Plain Pictures

We base our model of variational pictures on a model of plain pictures that is basically defined as a set of pixels.

Specifically, given a finite domain of locations *Loc*, a *picture* is a finite mapping from *Loc* to some type T. Here T typically is a set of colors, but it can be any type that has an equality predicate defined on it. Moreover, in most cases pictures are given by fixed, rectangular grid of pixels, which means that the type of locations is of the form $Loc^{n,m} = \{1, \ldots, n\} \times \{1, \ldots, m\}$. However, the definitions that follow do not depend on this particular structure, so that we can simply assume a finite set *Loc* of elements on which equality is defined.[1]

Thus the type of T pictures over the domain *Loc* is defined as $Pic_T = Loc \to T$. A picture is an element of that type, and a *pixel* of a picture $p \in Pic_T$ is given by a pair $(l, x) \in p$ with $l \in Loc$ and $x \in T$.

Here is a small example of a 2×2 picture over a type of symbols $S = \{\circ, \bullet, \star\}$: $p = \begin{smallmatrix}\circ\bullet\\ \bullet\circ\end{smallmatrix}$. Since the structure of T doesn't really matter, we will mostly omit it in the following and consider this parameter implicitly fixed, that is, we simply use the type *Pic*.

2.3 Adding Choices to Pictures

A *variational picture* of type T is a mapping from locations to variational T values, that is, $VPic_T = Loc \to V(T)$. One could consider a more general definition that also allows variability in the location domain. However, such a type would complicate the following definitions considerably without gaining much. In fact, if the type T contains some unit or null value, one can simulate differently sized pictures using such a designated value. On the other hand, the chosen definition still facilitates the local application of variability, which is an important feature of our model to be discussed later.

Corresponding to plain pictures, a *variational pixel* is an element of a variational picture $vp \in VPic_T$ where $vp = (l, vx)$ with $l \in Loc$ and $vx \in V(T)$. Again, we omit the T parameter from the type in the following.

Here is a variational 2×2 picture over type S that varies the pixels in vp's second column in dimension A: $vp_A = \begin{smallmatrix}\circ & A\langle\bullet,\circ\rangle\\ \bullet & A\langle\circ,\circ\rangle\end{smallmatrix}$. Since vp_A contains only choices

[1] This generality follows from the fact that our picture model doesn't require a notion of connectedness.

in one dimension, it encodes two plain variant pictures that can be extracted using selection, that is, $\lfloor vp_A \rfloor_{A.l} = p = \begin{smallmatrix}\circ\bullet\\ \bullet\circ\end{smallmatrix}$ and $\lfloor vp_A \rfloor_{A.r} = \begin{smallmatrix}\circ\circ\\ \bullet\circ\end{smallmatrix}$.

In general, a variational picture may contain multiple choices in different dimensions. Here is a picture that varies the upper right pixel in vp_A again using a nested B choice in A's left alternative: $vp_{AB} = \begin{smallmatrix}\circ & A\langle B\langle\bullet,\star\rangle,\circ\rangle\\ \bullet & A\langle\circ,\circ\rangle\end{smallmatrix}$. Selecting the left alternative of A in vp_{AB} now does not produce a plain picture, since the B choice has not been eliminated: $\lfloor vp_{AB} \rfloor_{A.l} = \begin{smallmatrix}\circ & B\langle\bullet,\star\rangle\\ \bullet & \circ\end{smallmatrix}$. This means that we need to also perform a selection for B. In the example we get $\lfloor vp_{AB} \rfloor_{\{A.l,B.l\}} = p$ and $\lfloor vp_{AB} \rfloor_{\{A.l,B.r\}} = \begin{smallmatrix}\circ & \star\\ \bullet & \circ\end{smallmatrix}$. On the other hand, selecting only the right alternative still produces a plain picture $\lfloor vp_{AB} \rfloor_{A.r} = \begin{smallmatrix}\circ\circ\\ \bullet\circ\end{smallmatrix}$.

The semantics of a variational picture is a mapping from decisions to plain pictures. The semantics definition iterates over all variational pixels and extracts and lifts the decisions to the level of pictures, effectively commuting decisions and locations.

$$\begin{aligned}[\![\cdot]\!] &: VPic \to V(Pic)\\ [\![vp]\!] &= \{(\delta,(l,x)) \mid (l,vx) \in vp, (\delta,x) \in vx\}\end{aligned}$$

The type of the semantic function helps explain its definition: since $V(X) = Dec \to X$, $Pic = Loc \to T$, and $VPic = Loc \to V(T)$, the type of the semantic function reads in expanded form as $(Loc \to (Dec \to T)) \to (Dec \to (Loc \to T))$.

As the examples vp_A and vp_{AB} illustrate, the ability to apply choices to individual pixels, makes variability a localized feature. This is an important property, since it allows only those parts to be varied that need it and keeps non-variable picture parts constant across different variant pictures, which supports editing by avoiding update anomalies [2]. For example, if we change the upper left pixel in vp_{AB} from $\circ$ to $\bullet$, this change has to be done only once and will still correctly affect all variants of the picture.

In order to support precise operations to create and modify the variability in a variational picture, we employ the notion of a *view decision*, which is simply a choice calculus decision that specifies the plain picture variant that is currently visible in the editor. View decisions are always complete, that is, they contain exactly one selector for each unique dimension contained anywhere in the variational picture.

2.4 Variability Types

The inclusion of choices in pictures suggests a classification of pixels according to their variability and the grouping of pixels with the same variability into regions. To formalize this idea we first define the notion of a *variability type*. The type of a plain, non-variational value is $\diamond$ (called *unit*), and the type of a variational value is given by its choice structure, which is obtained by replacing all plain values in it by $\diamond$. The variability type of a value can be determined by the function φ, which is defined as follows.

$$\begin{aligned}\varphi : V(T) &\to V(\{\diamond\}) \\ \varphi(D\langle vx, vy\rangle) &= D\langle \varphi(vx), \varphi(vy)\rangle \\ \varphi(x) &= \diamond\end{aligned}$$

The type of a pixel is given by the type of its value, that is, $\varphi(l, vx) = (l, \varphi(vx))$.

With this definition, all pixels in $\begin{smallmatrix}\circ\bullet\\ \bullet\circ\end{smallmatrix}$ have type $\diamond$. In $vp_{AB} = \begin{smallmatrix}\circ A\langle B\langle\bullet,\star\rangle,\circ\rangle\\ \bullet A\langle\circ,\circ\rangle\end{smallmatrix}$, the pixels in the left column have type $\diamond$, the lower right pixel has type $A\langle\diamond,\diamond\rangle$, and the upper right pixel has type $A\langle B\langle\diamond,\diamond\rangle,\diamond\rangle$. For notational convenience we also write more succinctly A for a type $A\langle\diamond,\diamond\rangle$. With this abbreviation we can say that the lower right pixel has type A and the upper right pixel has type $A\langle B,\diamond\rangle$.

A variability type tells us exactly what decisions are needed to extract all plain values from a variational value. For example, the set of plain values contained in vp_{AB} is given by $\{\lfloor vp_{AB}\rfloor_{\{A.l,B.l\}}, \lfloor vp_{AB}\rfloor_{\{A.l,B.r\}}, \lfloor vp_{AB}\rfloor_{A.r}\}$. We can compute the set of decisions required for extracting all plain values from a variational value a particular type with the following function *decs*.

$$\begin{aligned}decs : V(\{\diamond\}) &\to 2^{Dec} \\ decs(D\langle vx, vy\rangle) &= \{\delta \cup \{D.l\} \mid \delta \in decs(vx)\} \cup \{\delta \cup \{D.r\} \mid \delta \in decs(vy)\} \\ decs(x) &= \varnothing\end{aligned}$$

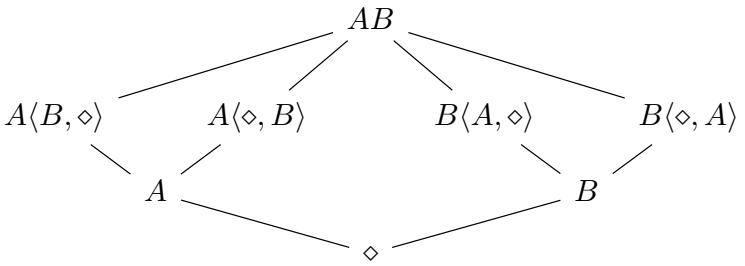
Based on the function *decs* we can define a *variability type equivalence* that holds for types that describe the same variability.

$$\phi \sim \phi' :\Longleftrightarrow decs(\phi) = decs(\phi')$$

Note that $\sim$ is not a simply derived from $\equiv$. For example, whereas $D\langle x, x\rangle \equiv x$ (due to idempotency), $\varphi(D\langle x, x\rangle) = D \not\sim \diamond = \varphi(x)$. We can generalize the notion to type equivalence naturally to a *refinement ordering* on variability types, again based on the decisions that are represented by the variability types.

$$\phi \gtrsim \phi' :\Longleftrightarrow \forall \delta' \in decs(\phi') : \exists \delta \in decs(\phi) : \delta \supseteq \delta'$$

We have, for example, $A\langle B, B\rangle \gtrsim A\langle B,\diamond\rangle$ and $A\langle\diamond, B\rangle \gtrsim A$. The variability refinement is a partial order that give rise to a lattice structure. Here is a small excerpt of this lattice involving the two dimensions A and B. Note the two types $A\langle B, B\rangle$ and $B\langle A, A\rangle$ are equivalent. This means that the nesting is not relevant, and we can write the type more accurately as AB to indicate that neither A nor B is in any way privileged over the another.



This diagram indicates that the nesting of dimensions does not matter in fully variationalized values, since the types are equivalent. This is an important property we will come back to in Sect. 3.

We write more shortly $D \in \delta$ whenever $D.l \in \delta$ or $D.r \in \delta$, and say that a dimension D *depends* on dimension D' in a type ϕ, written as $D' \leftarrow_\phi D$, if $D \in \mathit{decs}(\phi) \implies D' \in \mathit{decs}(\phi)$, that is, in order to make a selection in D one also has to make a selection in D'. For example, B depends on A in $A\langle B, \diamond\rangle$. We notice that in the type AB (which is equal to $A\langle B, B\rangle$ and $B\langle A, A\rangle$), B depends on A and A depends on B. In such situations, when two dimensions D and D' depend on one another in a type ϕ, we say that D and D' are *co-dependent* in ϕ, written as $D' \leftrightarrow_\phi D$.

2.5 Variability Regions

Based on variability types we can define a notion of regions that have the same variability. All pixels with the same type have the same variability structure, which means that they are mapped to plain variants by the same set of decisions. This property partitions the set of all pixels (or more precisely, their locations) into a set of disjoint regions. Specifically, for each variation type $\phi \in V(\{\diamond\})$ we define the *region of ϕ-variability* (or *ϕ-region* for short) as follows.

$$R_\phi(vp) = \{(l, vx) \in vp \mid \varphi(vx) \sim \phi\}$$

In the park example from the Introduction, the part of the image affiliated with the fountain is given by the region $R_{\mathit{Trees}\langle\diamond,\mathit{Fountain}\rangle}$.

The following lemma is a direct consequence of the definition of ϕ-regions.

Lemma 1. *For every variational picture vp, the set of (non-empty) regions $R_\phi(vp)$ forms a partition of vp.*

Since regions are identified by the common types of their pixels/locations, we can derive a refinement relation for regions based on the type refinement defined earlier. Specifically, $R_{\phi'} \gtrsim R_\phi$ if and only if $\phi' \gtrsim \phi$.

As indicated by the example scenario in the Introduction, (variational) pictures are typically the result of a sequence of operations performed in an editor. In particular, variability is introduced into a picture by marking an area and assigning a dimension to it. After that the resulting two alternatives can be edited in different ways and will generally contain different content.

In many cases, however, not every pixel in the marked area will be different in both alternatives. Consider again the picture $vp_A = \begin{smallmatrix}\circ & A\langle\bullet,\circ\rangle \\ \bullet & A\langle\circ,\circ\rangle\end{smallmatrix}$. Both pixels in the right column are variational: they are both of type A and thus belong to the same region. However, only the upper pixel differs in its alternatives. Since both alternatives of the lower pixel are equal, we could apply the idempotency law of the choice calculus and replace $A\langle\circ, \circ\rangle$ by $\circ$ without changing the semantics of the variational picture. Such a change would, however, change the region partition for vp_A. By systematically applying transformations for eliminating idempotent choices ($D\langle x, x\rangle \mapsto x$) as well as dominated choices ($D\langle D\langle x, y\rangle, z\rangle \mapsto D\langle x, z\rangle$

and $D\langle x, D\langle y, z\rangle\rangle \mapsto D\langle x, z\rangle$) to all pixels in a variational picture, we can shrink regions and thus increase the sharing in the picture. We can define a corresponding region shrinking operation as follows. First, we define the operations on variational values. Note that pattern matching on dominated choices is insufficient, since they can occur at arbitrary depths, so we use selection to avoid them instead.

$$\begin{aligned}\dot{\rho}(D\langle vx, vy\rangle) &= \begin{cases}\dot{\rho}(vx) & \text{if } \dot{\rho}(vx) = \dot{\rho}(vy)\\ D\langle \dot{\rho}(\lfloor vx \rfloor_{D.l}), \dot{\rho}(\lfloor vy \rfloor_{D.r})\rangle & \text{otherwise}\end{cases}\\ \dot{\rho}(x) &= x\end{aligned}$$

Region shrinking of variational pictures then simply works by applying the operation $\dot{\rho}$ to all values in all pixels.

$$\begin{aligned}\rho &: VPic \to VPic\\ \rho(vp) &= \{(l, \dot{\rho}(vx)) \mid (l, vx) \in vp\}\end{aligned}$$

Note that a variational picture obtained by ρ is *not* guaranteed to have maximized the sharing of values and thus is not minimal. Consider, for example, the variational value $vx = A\langle B\langle x, z\rangle, B\langle y, z\rangle\rangle$. The definition of $\dot{\rho}$ cannot extract and share the value z, since $\dot{\rho}(vx) = vx$. However, the variational value $vy = B\langle A\langle x, y\rangle, z\rangle$, which is equivalent to vx, is smaller than vx and does share z.

Note that the redundant value could occur deeply nested in the two alternatives, which means that a simple one-level dimension rotation is, in general, insufficient to expose redundant values. Nevertheless, redundant choices in idempotent or dominated choices will be eliminated by the region shrinking operation. It checks explicitly for identical alternatives and, at every step, uses selection on both branches to ensure no nested choices in matching dimensions.

2.6 Distilling Variational Pictures

Given two plain pictures p and p', we can automatically merge them and produce a variational picture that captures the differences between p and p' in choices of some dimension D and keeps all common parts as plain values. First, we define an operation $\dot{\Delta}$ for comparing individual pixel values. If we only needed $\dot{\Delta}$ for generating one variational picture from two plain pictures, its type could be $T \times T \to V(T)$, but since we actually want to apply the merge operation repeatedly to a number of pictures, its type should be $T \times V(T) \to V(T)$, which allows a plain picture to be merged with a variational picture. The operation can, of course, still be used with two plain pictures since a plain picture is just a special case of a variational one.

$$\begin{aligned}\dot{\Delta} &: T \times V(T) \to V(T)\\ \dot{\Delta}(x, vx) &= \begin{cases}x & \text{if } x = vx\\ D\langle x, vx\rangle & \text{otherwise}\end{cases}\end{aligned}$$

Note that the equality between x and vx can only hold when vx is a plain value. Note also that we have not specified which dimension D is to be used in the definition, since the concrete name does not really matter. We have to postulate however, that D has not been used in any of the vx values that $\dot{\Delta}$ is applied to. Effectively, we want to use a new dimension for every new picture that is merged into a variational picture.

Using $\dot{\Delta}$, the operation Δ for merging a plain picture into a variational one and capturing their differences in choices can be defined as follows. We assume that both pictures are defined over the same domain of locations. Note that we use the same l in both qualifiers of the set comprehension to express a parallel iteration over all pixels in both pictures.

$$\Delta : Pic \times VPic \to VPic$$
$$\Delta(p, vp) = \{(l, \dot{\Delta}(x, vx)) \mid (l, x) \in p, (l, vx) \in vp\}$$

We can generalize the definition of Δ to work on not just two but a whole set of pictures in a straightforward way. The only side condition, which is not formalized here, is that the a fresh dimension is used in each new call to $\dot{\Delta}$. This could be formalized by threading a set of dimension names through the successive applications of Δ and $\dot{\Delta}$, but this doesn't contribute much to the understanding of the operations, and we therefore omit it here for brevity.

$$\Delta^* : 2^{Pic} \to VPic$$
$$\Delta^*(\{p\}) = p$$
$$\Delta^*(\{p\} \cup P) = \Delta(p, \Delta^*(P))$$

We can see this in action using the small plain pictures $p_1 = \begin{smallmatrix}\circ\bullet\\ \bullet\circ\end{smallmatrix}$, $p_2 = \begin{smallmatrix}\circ\circ\\ \bullet\bullet\end{smallmatrix}$, and $p_3 = \begin{smallmatrix}\circ\star\\ \bullet\bullet\end{smallmatrix}$. Now we need to evaluate $\Delta^*(\{p_1, p_2, p_3\})$, which we can expand to $\Delta(p_3, \Delta(p_1, p_2))$. Evaluating $\Delta(p_1, p_2)$ produces $p_{12} = \begin{smallmatrix}\circ A\langle\bullet,\circ\rangle\\ \bullet A\langle\circ,\bullet\rangle\end{smallmatrix}$. Finally, we can evaluate $\Delta(p_3, p_{12})$ and get the variational picture $\begin{smallmatrix}\circ B\langle\star, A\langle\bullet,\circ\rangle\rangle\\ \bullet B\langle\bullet, A\langle\circ,\bullet\rangle\rangle\end{smallmatrix}$.

3 Properties of Variational Pictures

In this section we collect a number of general properties of variational pictures that serve as additional characterizations of the concept and also justify the chosen design.

The first observation is that variational picture distillation and semantics are in some sense inverse operations of each other. Distillation is, in fact, what users do when they have a set of pictures in mind that they want to represent in a variational one. Of course, users typically won't encode the differences as efficiently as the operation Δ^*, which will often result in a variational picture with maximal sharing. Now we can show that the semantics of a variational picture that is generated by Δ^* from a set of plain pictures produces exactly the original plain pictures.

Theorem 1. $\forall P \in 2^{Pic} : range(\llbracket \Delta^*(P) \rrbracket) = P.$

This theorem states that the semantics of variational pictures are correct; it says that distilling a variational picture from a set of plain pictures does not lose any information, because the original set of pictures can be extracted by the semantics.

Since a choice tree with n leaves contains $n-1$ dimensions (as mentioned in Sect. 2.1), that also means that n pictures are distilled into a variational picture with $n-1$ dimensions.

A closely related result is that from the plain pictures encoded in a variational picture we can recover an equivalent variational picture using Δ^*. Note that the reconstructed variational picture may not be identical to the original one in terms of how the variation is represented, that is, in general we have $\Delta^*(\mathit{range}([\![vp]\!])) \neq vp$; all we can guarantee is that the reconstructed picture has the same semantics.

Theorem 2. $\forall vp \in VPic : [\![\Delta^*(\mathit{range}([\![vp]\!]))]\!] = [\![vp]\!]$.

As the example scenario in the Introduction has shown, areas of variability are often nested, which leads to correspondingly nested choices in the pixel values. This is the case, for example, for the optional fountain. The fountain itself is an area of variability, but it is also nested inside one alternative of another area in which a wooded area is cleared. If we call these dimensions *Trees* and *Fountain*, we would expect to see many pixel values with the type $\mathit{Trees}\langle\diamond, \mathit{Fountain}\rangle$.

One concern is that the initially chosen area and thus choice nesting commits a user to a particular nesting that cannot be changed at a later stage of editing. The next theorem shows that this is actually *not* the case and that variation creation, in particular, the nesting of choices, does not lead to a premature commitment to a particular variation structure.

Consider the situation that an area for choice B has been created inside an area for choice A, and let's assume that the B choice lives inside A's left alternative (just as in the example vp_{AB} from Sect. 2.3). The type of the pixels inside the B area is $A\langle B, \diamond\rangle$, and the type of the pixels outside of B but inside of A is simply A. We call a pair of regions such as R_A and $R_{A\langle B,\diamond\rangle}$ a *region refinement pair*, since $A\langle B, \diamond\rangle \gtrsim A$ (and thus $R_{A\langle B,\diamond\rangle} \gtrsim R_A$), and write such a pair as $R_A[R_{A\langle B,\diamond\rangle}]$ to indicate that it was probably the result of creating a B choice in an A area (or creating an A choice around a B area).[2]

We can observe that for the pixels in $R_{A\langle B,\diamond\rangle}$, B depends on A. If we consider the dual case of a region refinement pair $R_B[R_{B\langle A,\diamond\rangle}]$, we can see that the dependency is the other way around, since for the pixels in $R_{B\langle A,\diamond\rangle}$, A depends on B. So it seems that the chosen nesting for the areas and dimensions determines two incompatible dependencies among the dimensions. However, there is a straightforward way to reconcile these different refinement pairs by refining the region $R_{A\langle B,\diamond\rangle}$ to $R_{A\langle B,B\rangle} = R_{AB}$ (or dually refining $R_{B\langle A,\diamond\rangle}$ to $R_{B\langle A,A\rangle} = R_{AB}$).

[2] Since regions are derived from the pixel types, such region pairs do not necessarily occur in any particular geometric relationship. However, such region pairs typically result from editor actions that create one choice area within another.

A region refinement $R_{A\langle B,\diamond\rangle}$ to R_{AB} can be achieved by simply expanding the right alternative of all A choices in the region's pixels into idempotent B choices, that is, by replacing $A\langle B\langle x, y\rangle, z\rangle$ with $A\langle B\langle x, y\rangle, B\langle z, z\rangle\rangle$.[3]

We can then apply a similar region refinement to R_A, turning it as well into a region of type AB. This automatically merges the region refinement pair $R_A[R_{A\langle B,\diamond\rangle}]$ into one region R_{AB}, and if we assume that the region is not nested within another area, we can consider it together with the surrounding non-variational pixels of type $\diamond$ as a new, bigger region refinement pair $R_\diamond[R_{AB}]$. Next we can refine (part of) the region $R_\diamond$ to R_B, leading to $R_B[R_{AB}]$. We can summarize the sequence of transformations as follows.

$$R_A[R_{A\langle B,\diamond\rangle}] \rightsquigarrow R_A[R_{AB}] \rightsquigarrow R_{AB}[R_{AB}] \rightsquigarrow R_{AB} \rightsquigarrow R_\diamond[R_{AB}] \rightsquigarrow R_B[R_{AB}]$$

We can perform a similar transformation for the region pair $R_B[R_{A\langle B,\diamond\rangle}]$:

$$R_B[R_{A\langle B,\diamond\rangle}] \rightsquigarrow R_B[R_{AB}] \rightsquigarrow R_{AB}[R_{AB}] \rightsquigarrow R_{AB} \rightsquigarrow R_\diamond[R_{AB}] \rightsquigarrow R_A[R_{AB}]$$

This shows that while we can't turn $R_A[R_{A\langle B,\diamond\rangle}]$ into $R_B[R_{A\langle B,\diamond\rangle}]$ (or vice versa) without removing information, we can invert the nesting of choices if we refine the innermost region sufficiently.

Theorem 3. $R_A[R_{A\langle B,\diamond\rangle}] \rightsquigarrow^* R_B[R_{AB}]$ *and* $R_B[R_{A\langle B,\diamond\rangle}] \rightsquigarrow^* R_A[R_{AB}]$

We consider an application of this theorem in the next section.

4 Maintenance of Variational Pictures

It is unrealistic to expect variational picture authors to know the precise and final locations of variability they will need from the outset. This means that our model of pictures should support changing areas of variability that have already been defined.

Consider the park example from the Introduction. Suppose that, after creating the design shown, the architect is told that the *Fountain* area needs to include pipes connecting from the water main. This means that the associated area needs to not only grow, but grow such that the nesting order with the *Trees* area is reversed. The final result can be seen in Fig. 2.

Fortunately, we have already seen in Sect. 3 that region refinement allows us to change the nesting order without issue. We just need to transform the appropriate pixels by expanding them all with an outer *Fountain* choice. There are three transformations to make, namely for those pixels originally non-variational which are now contained in *Fountain*, those which were originally in *Trees* but outside of *Fountain*, and those inside of *Fountain*. They are transformed as follows.

[3] There are situations in which other transformations, such as $A\langle B\langle x, y\rangle, B\langle z, y\rangle\rangle$ may be more appropriate, but the point is that a region refinement is easy to achieve.

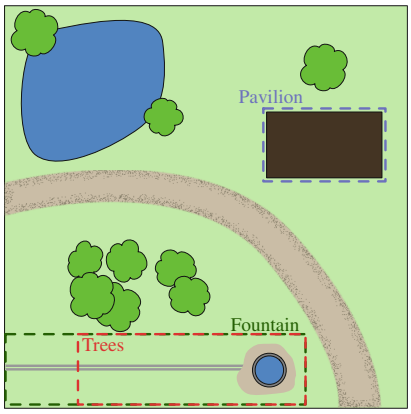


Fig. 2. The variational picture after resizing the previously nested *Fountain* area to contain the *Trees* area, in order to depict the connecting water pipes.

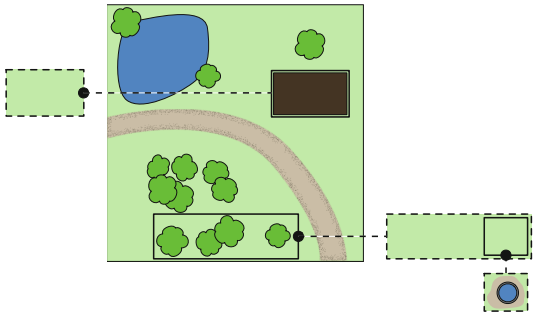


Fig. 3. A Variational Area Tree (VAT) that showing an entire variational picture at once. Here the VAT for the view decision $\{Trees.l, Fountain.l, Pavilion.r\}$ is shown.

$$
\begin{aligned}
x &\mapsto Fountain\langle x, x\rangle \\
Trees\langle x, y\rangle &\mapsto Fountain\langle Trees\langle x, y\rangle, Trees\langle x, y\rangle\rangle \\
Trees\langle x, Fountain\langle y, z\rangle\rangle &\mapsto Fountain\langle Trees\langle x, y\rangle, Trees\langle x, z\rangle\rangle
\end{aligned}
$$

Although we do not depict it here, we could similarly envision scenarios where we want to shrink regions. The swapping works in exactly the opposite way to expanding. The only difference is we must remove part of the variability by performing a selection of the dimension that we are shrinking. Since we need to choose one alternative or the other to select, we defer to the value of the view decision. This gives the user control over the different possibilities.

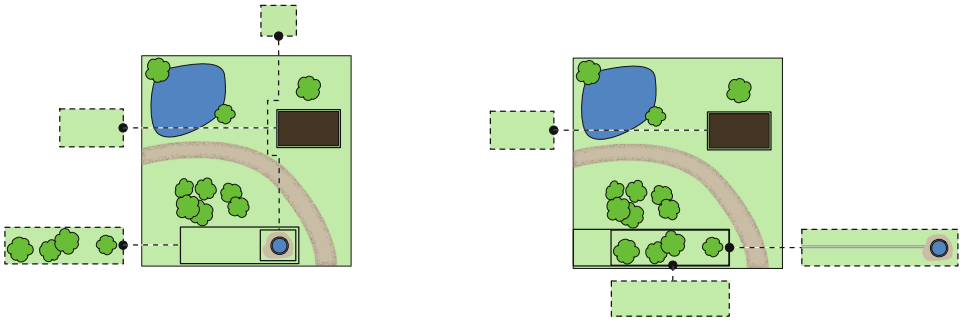
5 Variational Area Trees

Although the described model of variational pictures is sufficiently flexible to build variational pictures, so far we have limited ourselves to viewing a single variant at a time. While this simplifies editing operations, there is still a need to produce overviews to better understand and navigate the pictures, which calls for a visual language with a *graphical* domain [3]. To this end, we present *variational area trees* (VATs), a diagrammatic approach to viewing an entire variational pictures simultaneously.

VATs show the currently selected variant in its entirety as a root node, and then also all of the other possible selections as branches. Left and right alternatives are always connected via a dotted line and shown either to the left and right of one another or above and below, in order to use space more efficiently (we assume a reasonable layout algorithm). Dotted outlines show unselected

variants and solid outlines indicate selected ones. A variational area tree for the view decision $\{Trees.l, Fountain.l, Pavilion.r\}$ the park example are shown in Fig. 3.

VATs have a number of use cases. Obviously, they provide a concise overview of an entire variational pictures including all of its variants. In addition, the design of VATs has a number of useful properties.



(a) (b)

Fig. 4. Additional configurations of the park picture VAT. In (a) we see the view decision $\{Trees.r, Fountain.r, Pavilion.r\}$, and (b) shows the case after the commuting of the *Trees* and *Fountain* regions for the view decision $\{Trees.l, Fountain.l, Pavilion.r\}$.

First, the total number of regions shown gives the total number of different drawing areas and provides a clue to the variability of the picture. Second, since unselected areas are never nested and thus all unselected areas are always placed on the top level, counting all top-level unselected areas provides a concrete measure of what is hidden in the current view. Third, the number of boundaries that are crossed by a (horizontal or vertical) connector line indicates the nesting level of that choice/variational area, and the types of the crossed boundaries (unselected vs. selected) tell immediately what decisions have to be flipped (at minimum) to make the area visible. Fourth, VATs illustrate nicely where in the choice tree the current variant is located. For example, the VAT in Fig. 4(a) is located rightmost/bottommost, which means that the right alternative must be selected in all dimensions.

VATs can also serve as quick navigation tools. It is easy to conceive of an interface in which a user can zoom out to a VAT and then change the selection of arbitrary dimensions quickly. Finally, VATs could also support compound operations or queries. For example, a picture creator may want to view the parts of the picture that are not variational or all the areas affected by a particular dimension. These kinds of operations could be supported by a simple filtering operation on VATs. Figure 4 demonstrates some additional examples for different selections and after expanding the *Fountain* region as done in Sect. 4.

6 Related Work

Although there are a large number of potential applications for variational pictures, the research in this area is limited in scope. There are many tools design to offer digital image version control and asset management, although most are proprietary and do not describe their specific model. Examples of these include Adobe Drive and AutoDesk Vault. Some more general version control tools offer support for image diff operations including Perforce and Git via services such as Github.

This topic also emerges in the field of information visualization, Heer et al. [4] performed a large survey of graphical history tools from the context of information visualization and exploratory analysis which covers this topic well beyond the scope of our work. Chen et al. [5] proposed a graph-based revision control system for images. Being based on graphs, it focuses on paths of editing operations rather than pixels or image objects, and challenges such as diff and merge are solved with graph operations. They also offer support for selective undo and "nonlinear exploration", in which the user can adjust parameters to operations that have already been performed. Gleicher et al. [6] demonstrated comparative visualization as a fundamental idea, which can be viewed as an application of variability to pictures within the scope of information visualization. To our knowledge, none of this work describes a general variability-aware model either.

Another related body of work is on model difference techniques. Kolovos et al. [7] includes an overview of the topic. Specific examples include Ohst et al. [8] who described an approach to model difference in UML and Cicchetti et al. [9] who showed a technique for representing differences between models independent of the metamodel. As models are generally expressed using graphs, none of this work describes a pixel-based approach.

Terry and Mynatt proposed Side Views [10] for previewing commands such as image rotation and coordinate transformations, which makes use of variational pictures but does not model them explicitly. Terry et al. [11] expanded on this by managing variations more explicitly. Their tool, Parallel Paths, allows operations to be applied to individual picture variants or groups of them and also tracks history to navigate throughout their notion of a variational picture. That work is primarily focused on a specific user interface, however, and not on a formal model of variational pictures. Hartmann et al. [12] described an approach to creating variational interactions and user interfaces based on editing linked source code alternatives and parallel execution. Finally, Foo et al. [13] summarize existing work and propose new ideas in the challenge of identifying similar (in the sense of different resolutions, compression techniques, fragments, etc.) images. This work focuses on identifying similar images rather than explicitly managing the variability.

7 Conclusions and Future Work

In this work we have introduced the notion of a variational pictures and introduced a formal model that captures their functionality and behavior based on

the choice calculus. We have established several basic properties of the model that support tools for editing and transforming variational pictures. The generality of the presented model (as reflected, for example, by the general types *Loc* and *T*) allows more specific models of variational pictures to be targeted at specific application domains, which provides a rich area for future work.

References

1. Erwig, M., Walkingshaw, E.: The choice calculus: a representation for software variation. ACM Trans. Soft. Eng. Methodol. **21**(1), 6:1–6:27 (2011)
2. Date, C.J.: Database in Depth: Relational Theory for Practitioners. O'Reilly Media Inc., Sebastopol (2005)
3. Erwig, M., Smeltzer, K., Wang, X.: What is a visual language? J. Vis. Lang. Comput. **38**(C), 9–17 (2017)
4. Heer, J., Mackinlay, J., Stole, C., Agrawala, M.: Graphical histories for visualization: supporting analysis, communication, and evaluation. IEEE Trans. Vis. Comput. Graph. **14**, 1189–1196 (2008)
5. Chen, H.T., Wei, L.Y., Chang, C.F.: Nonlinear revision control for images. In: ACM SIGGRAPH, pp. 105:1–105:10 (2011)
6. Gleicher, M., Albers, D., Walker, R., Ilir, J., Hansen, C.D., Robers, J.C.: Visual comparison for information visualization. Inf. Vis. **10**, 289–309 (2011)
7. Kolovos, D.S., Ruscio, D.D., Pierantonio, A., Paige, R.F.: Different models for model matching: an analysis of approaches to support model differencing. In: ICSE Workshop on Comparison and Versioning of Software Models, pp. 1–6 (2009)
8. Ohst, D., Welle, M., Kelter, U.: Differences between versions of UML diagrams. In: European Software Engineering Conference Held Jointly with ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 227–236 (2003)
9. Cicchetti, A., Di Ruscio, D., Pierantonio, A.: A metamodel independent approach to difference representation. J. Object Technol. **6**(9), 165–185 (2007)
10. Terry, M., Mynatt, E.D.: Side views: persistent, on-demand previews for open-ended tasks. In: ACM Symposium on User Interface Software and Technology, pp. 71–80 (2002)
11. Terry, M., Mynatt, E.D., Nakakoji, K., Yamamoto, Y.: Variation in element and action: supporting simultaneous development of alternative solutions. In: SIGCHI Conference on Human Factors in Computing Systems, pp. 711–718 (2004)
12. Hartmann, B., Yu, L., Allison, A., Yang, Y., Klemmer, S.R.: Design as exploration: creating interface alternatives through parallel authoring and runtime tuning. In: ACM Symposium on User Interface Software and Technology, pp. 91–100 (2008)
13. Foo, J.J., Sinha, R., Zobel, J.: Discovery of image versions in large collections. In: Cham, T.-J., Cai, J., Dorai, C., Rajan, D., Chua, T.-S., Chia, L.-T. (eds.) MMM 2007. LNCS, vol. 4352, pp. 433–442. Springer, Heidelberg (2006). https://doi.org/10.1007/978-3-540-69429-8_44