# Codifying Visual Representations

Wode Ni[1(✉)], Sam Estep[1], Hwei-Shin Harriman[1], Jiří Minarčík[2], and Joshua Sunshine[1]

[1] Carnegie Mellon University, Pittsburgh, PA, USA
nimo@cmu.edu
[2] Pittsburgh, USA

**Abstract.** Making visually appealing and meaningful diagrams involves craftsmanship in designing the visual representation, drawing shapes, and laying them out. Can the effort spent on diagrams by an expert be reused by others, especially those without the expertise in design and drawing? In this paper, we outline our prior work on PENROSE, a diagramming tool with first-class support for reusing visual representations. The nature of our approach to reusability necessitates a domain-agnostic method to automatically lay out a diagram. We highlight our existing approach for general diagram layout and styling, and propose a new composable approach for codifying visual representations to reuse expertise that cuts across domains.

**Keywords:** Diagram Authoring Tools · Automatic Diagram Layout · Natural Diagramming Interface

## 1 Defining Visual Representations

Authoring good diagrams requires both knowledge of the domain being illustrated and graphic design sense. An effective diagram author develops *visual representations* by illustrating domain concepts with appropriate visual elements and spatial layout. In prior work, we interviewed diagram authors and found that defining visual representations is an important part of their diagramming process and authors cared about reusing these representations to improve productivity [2]. However, existing tools limited reuse to either duplicating prior diagrams and manually tweaking them. Authors found these methods to be tedious and time-consuming.

This result suggests that diagramming tools should be *representationally salient*: tools should allow authors to define visual representations for domain-specific concepts in a manageable, scalable, and composable way. With representation salience as an explicit goal, we build an open-source tool called PENROSE [4]. Authors make diagrams in PENROSE using two languages: SUBSTANCE describes the conceptual content of a diagram while STYLE describes the visual

W. Ni, S. Estep and H.-S. Harriman—Authors contributed equally.
J. Minarčík—Independent Researcher.

J. Lemanski et al. (Eds.): Diagrams 2024, LNAI 14981, pp. 454–457, 2024.
https://doi.org/10.1007/978-3-031-71291-3_37

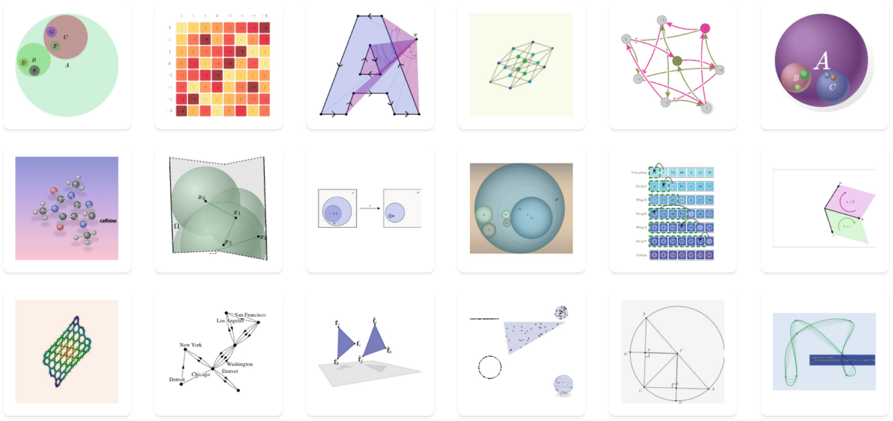
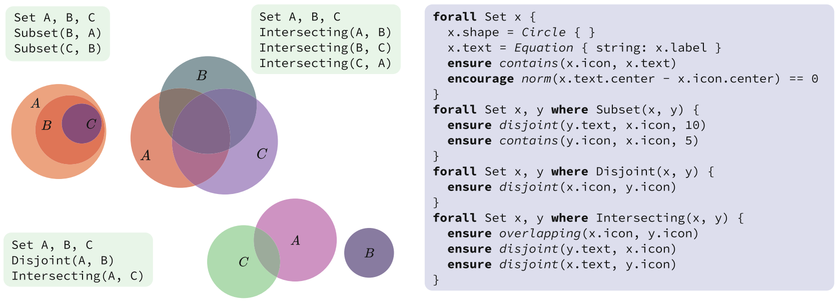


**Fig. 1.** Selected PENROSE-authored diagram examples. Each diagram is generated from a different visual representation, codified as its own STYLE program.

representation. To date, PENROSE has been used to define visual representations across a wide range of domains,[1] including diagrams shown in Fig. 1, and PENROSE generates diagrams for external tools [1,3].

## 2 Reusing Visual Representations

In PENROSE, the STYLE language encodes visual representations by mapping conceptual objects from SUBSTANCE to primitive visual shapes and descriptions of the visual layout. For instance, in Fig. 2, the STYLE program describes how to map sets and relationships in the domain to visual shapes (e.g., `Circle`) and constraints, and this STYLE is reused to generate multiple Euler diagrams.



```
Set A, B, C
Subset(B, A)
Subset(C, B)
```

```
Set A, B, C
Intersecting(A, B)
Intersecting(B, C)
Intersecting(C, A)
```

```
Set A, B, C
Disjoint(A, B)
Intersecting(A, C)
```

```
forall Set x {
  x.shape = Circle { }
  x.text = Equation { string: x.label }
  ensure contains(x.icon, x.text)
  encourage norm(x.text.center - x.icon.center) == 0
}
forall Set x, y where Subset(x, y) {
  ensure disjoint(y.text, x.icon, 10)
  ensure contains(y.icon, x.icon, 5)
}
forall Set x, y where Disjoint(x, y) {
  ensure disjoint(x.icon, y.icon)
}
forall Set x, y where Intersecting(x, y) {
  ensure overlapping(x.icon, y.icon)
  ensure disjoint(y.text, x.icon)
  ensure disjoint(x.text, y.icon)
}
```

**Fig. 2.** Three Euler diagrams generated from different SUBSTANCE programs (left) using the same visual representation from a single STYLE program (right).

[1] https://penrose.cs.cmu.edu/examples.

We say that Style achieves representation salience because it encodes the visual representation of Euler diagrams for all possible Substance programs. Importantly, representation salience *necessitates* automatic layout. Alternatives could include specifying layout in the Substance program or requiring manual tweaks, both of which imply that important aspects of the visual representation were not sufficiently encoded in Style. That is, reusability is a necessary consequence of representation salience.

While Style serves to abstract visual layout from the perspective of the Substance writer, it is itself built on top of a set of Penrose primitives for describing a visual representation as a numerical optimization problem. The `ensure` keyword denotes a hard constraint which must be satisfied, while `encourage` denotes a soft objective which the system should strive to achieve.

The `disjoint`, `contains`, and `overlapping` constraints used in Fig. 2 are examples of visual layout primitives provided by Penrose. Many different Style programs need to talk about shapes that must not overlap or must be nested, for instance, so we've developed a mathematical framework to describe these for arbitrary shapes. Using the signed distance function

$$\phi_A(x) = \begin{cases} -d(x, \partial A) & x \in A, \\ d(x, \partial A) & x \notin A. \end{cases} \qquad \text{where} \qquad d(x, \partial A) = \min_{y \in \partial A} |x - y|$$

and the Minkowski difference $A - B = \{a - b \colon a \in A, b \in B\}$, we can perform layout by composing together these two operations in various ways:

$$\begin{aligned} \texttt{disjoint(A, B)} \quad &\iff \quad \text{minimize } \max(0, -\phi_{A-B}(0)) \\ \texttt{contains(A, B)} \quad &\iff \quad \text{minimize } \max(0, -\phi_{A^{\complement}-B}(0)) \\ \texttt{overlapping(A, B)} \quad &\iff \quad \text{minimize } \max(0, \phi_{A-B}(0)) \end{aligned}$$

We find these Minkowski penalties particularly useful for label placement, which is often a very tedious subtask of diagramming. To support Style construction, Penrose provides a library of over 200 built-in functions and over 50 pre-defined layout constraints and objectives. These functions and primitives are useful across many domains, and are thus reused in many Style files.

## 3 Composing Visual Representations

We have observed that visual representations in different domains often share common visual components and layout patterns. For instance, multiple examples in Fig. 1 include circles with nearby text labels. Further, common visual techniques are widely used in diagramming to convey domain-independent concepts, such as using varying opacity or line weight to highlight parts of a diagram, maintaining layout consistency across multiple diagrams to form a visual narrative, or using sliders or other widgets to drive real-time physical simulations in interactive webpages. It seems natural to separate out these common patterns into their own components, suggesting that Penrose's existing reusability of visual representations in Style does not provide sufficient flexibility for the needs of digital diagrammers.

In the current version of Penrose, authors can reuse geometric and layout primitives (Sect. 2) to create new Style programs, and users consume these programs by writing different Substance programs with them. Each Style program is standalone and self-contained, meaning that everything from the styling of points to the color palettes must be defined within that program. In practice, this means that common visual design patterns are copied and pasted between Style files. Additionally, it is common for individual diagrams within a domain to have customized visual elements to draw focus or illustrate a concept. Currently, the only way to override the domain-wide visual style in Penrose is by using workarounds that involve more copying/pasting code in Style. These two limitations result in repetitive and lengthy programs that require high effort to edit and maintain, even for expert Penrose users.

While code duplication and multiple versions of Style may be manageable on a small scale (e.g., Fig. 1), we plan to build a broader ecosystem of diagrams and this requires more flexible reuse mechanisms. We propose **composability** as the main design goal for improving Penrose. The existing layout primitives are an example of composability: authors can reuse and *combine* multiple primitives to form new layout problems. Looking forward, we plan to allow diagrammers to create *modules* of visual components and layout patterns. Through this mechanism, an author can draw together multiple different modules they need for their own diagram. And these modules can themselves be composed from other modules: for instance, a module for visualizing complex analysis might make use of lower-level modules for visualizing a coordinate plane and plotting curves, but build on top of that with domain-specific visuals for singularities in holomorphic functions. In addition to user-defined modules, there are also opportunities to build domain-independent visual techniques, such as individual object-level highlighting or annotations, into our languages or as standard library modules.

We believe this composable approach will open up new possibilities for diagrammers to collaborate and create more flexible, reusable, and expressive visual representations. Going forward, we plan to leverage research on common building blocks of and layout patterns in specific domains of diagramming, to construct a substrate for composable visual representations.

## References

1. Clark, C., Bohrer, R.: Homotopy type theory for sewn quilts. In: Proceedings of the 11th ACM SIGPLAN International Workshop on Functional Art, Music, Modelling, and Design, FARM 2023 (2023)
2. Ma'ayan, D., Ni, W., Ye, K., Kulkarni, C., Sunshine, J.: How domain experts create conceptual diagrams and implications for tool design. In: Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, CHI 2020 (2020)
3. Nawrocki, W., Ayers, E.W., Ebner, G.: An extensible user interface for lean 4. In: Proceedings of the 14th International Conference on Interactive Theorem Proving, ITP 2023 (2023)
4. Ye, K., et al.: Penrose: from mathematical notation to beautiful diagrams. ACM Trans. Graph (2020)