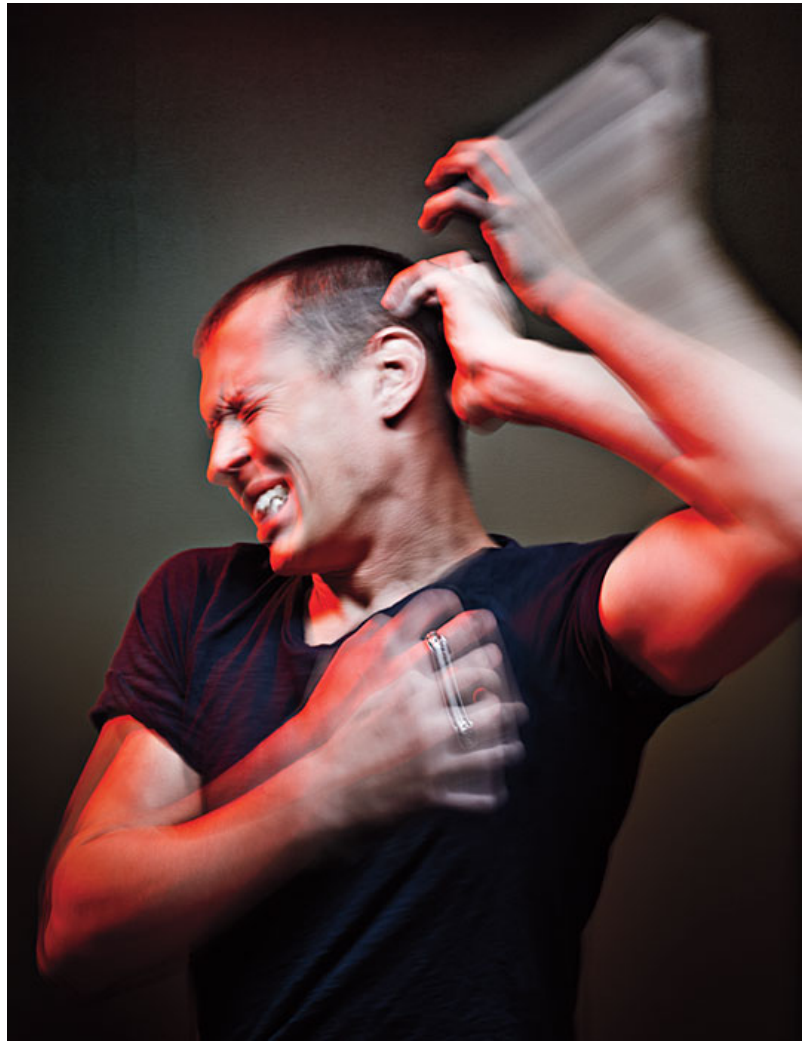
Declarative, embedded drawing with `diagrams`: past and future

Brent Yorgey
University of Pennsylvania

Hac φ
May 21, 2010



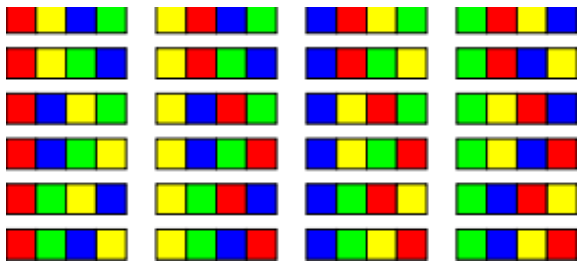
The diagrams library started to scratch a personal itch.

```
dia :: Diagram
dia = arrange $ map colorsToStrip (permutations [red, yellow, blue, green'])
  where green' = rgb 0 1 0

arrange :: [Diagram] -> Diagram
arrange = hsep 10 . map (vsep 5) . groups
  where groups = takeWhile (not . null) . map (take 6) . iterate (drop 6)

colorsToStrip :: Color c => [c] -> Diagram
colorsToStrip = hcat . map (\c -> fc c $ rect 10 10)

main = renderAs PNG "permutations.png" (Width 300) dia
```

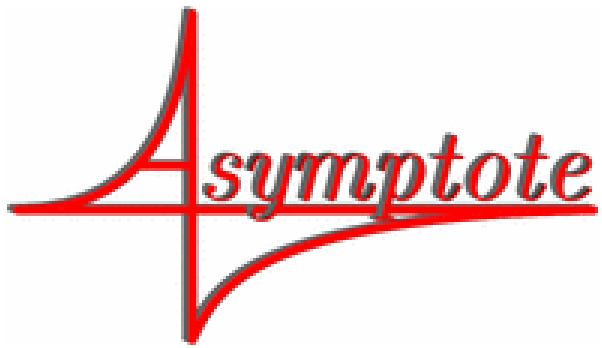


The goal: programmatically generate drawings and diagrams in a way that is declarative, powerful, and semantically elegant.

METAPOST

I first looked into existing solutions. The most obvious candidate is MetaPost.

But it's not sufficiently declarative, and uses a weird ad-hoc language which isn't general-purpose.



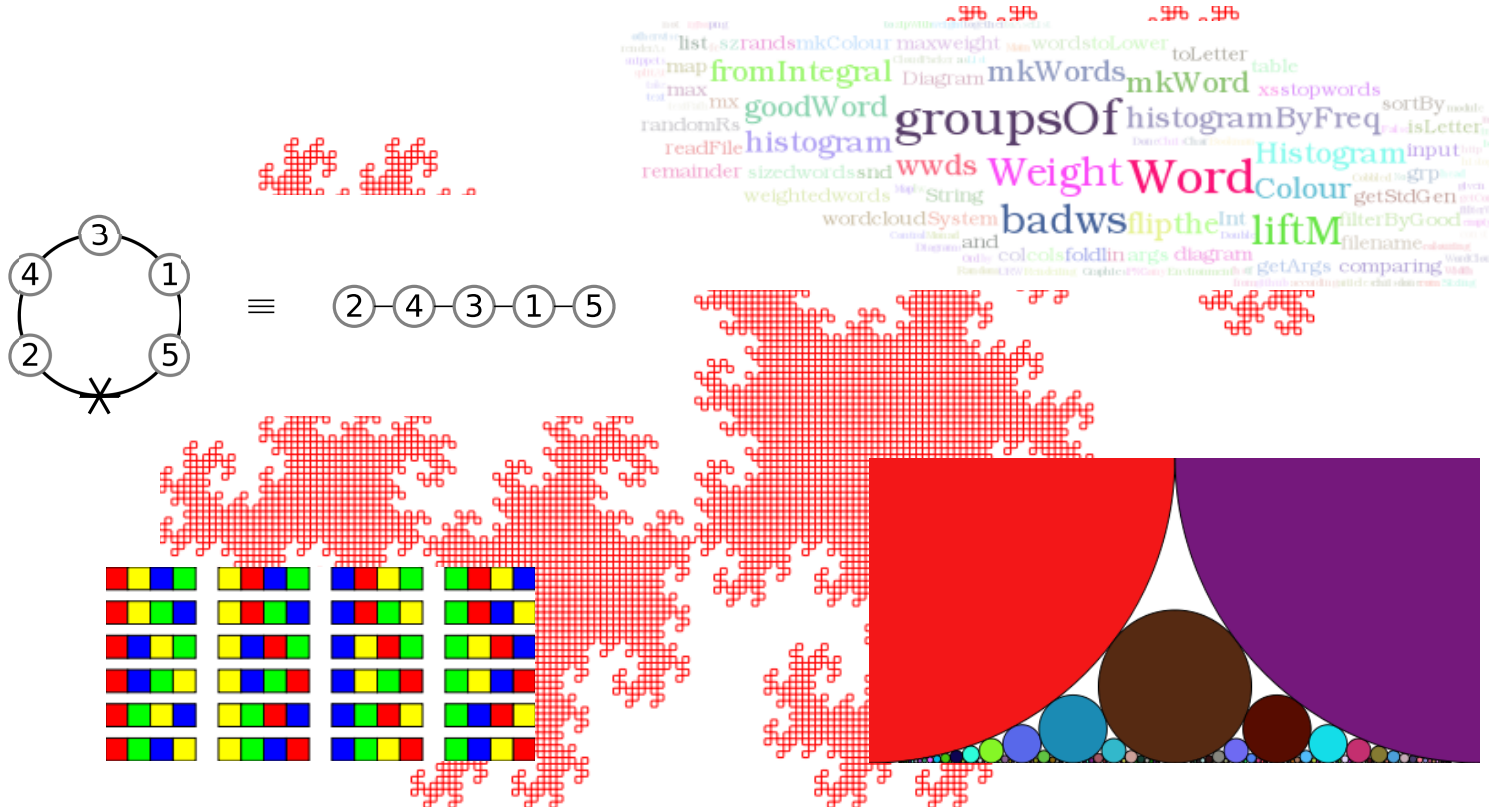
What about Asymptote, which is supposed to be a modern replacement for MetaPost?

No thanks: at least it uses a general-purpose language (so we can compute the diagrams we want to describe), but it's a TERRIBLE language combining the worst features of C++ and Java.

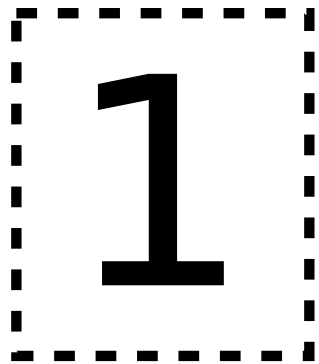
PGF/TikZ

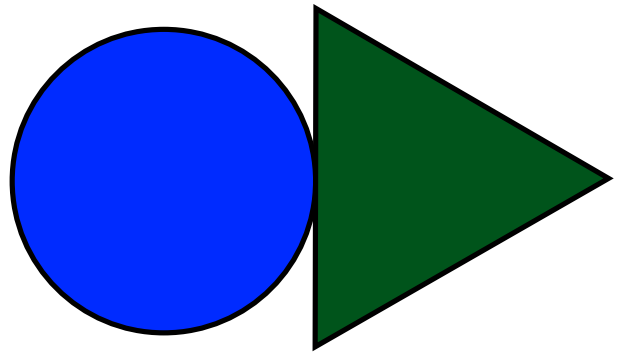
What about PGF/TikZ?

That's right, it uses an ad-hoc, non-general purpose language and. . . sigh.

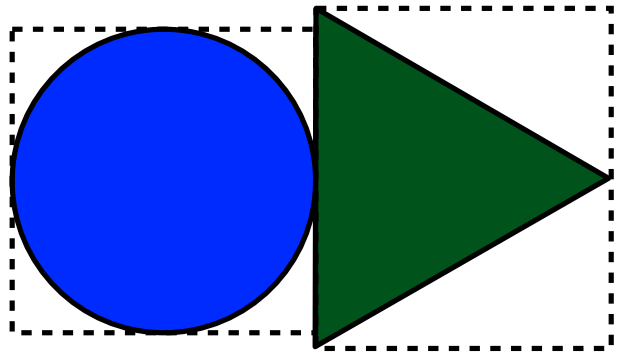


Thus, the `diagrams` library was born! It's gotten a bit of use, people seem to like it. It got quite a few things right, but let's look at some things it got wrong.

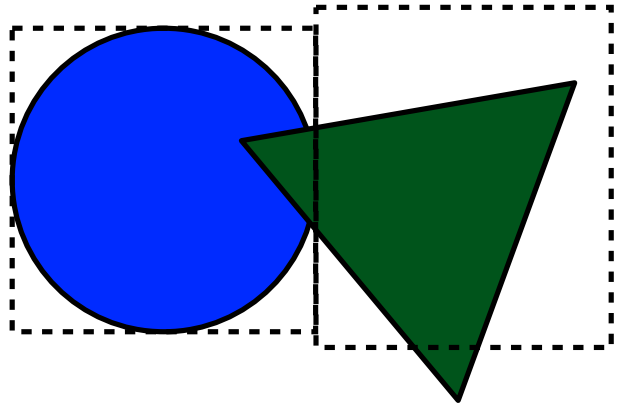

1



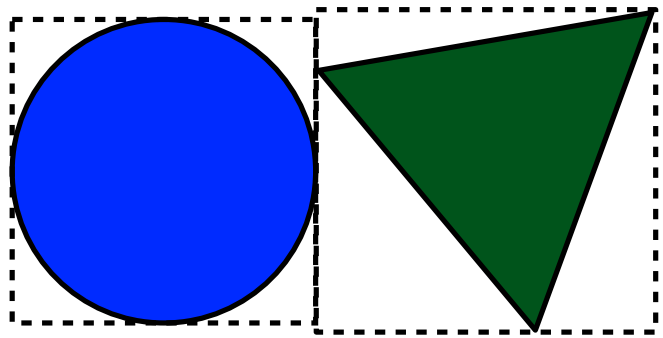
A fundamental ability of the library is to put two diagrams next to each other to create a larger diagram.



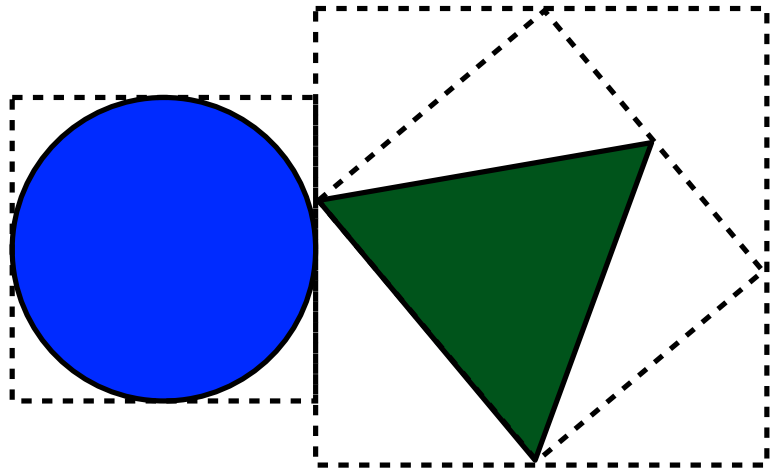
An obvious way to accomplish this is with bounding boxes.



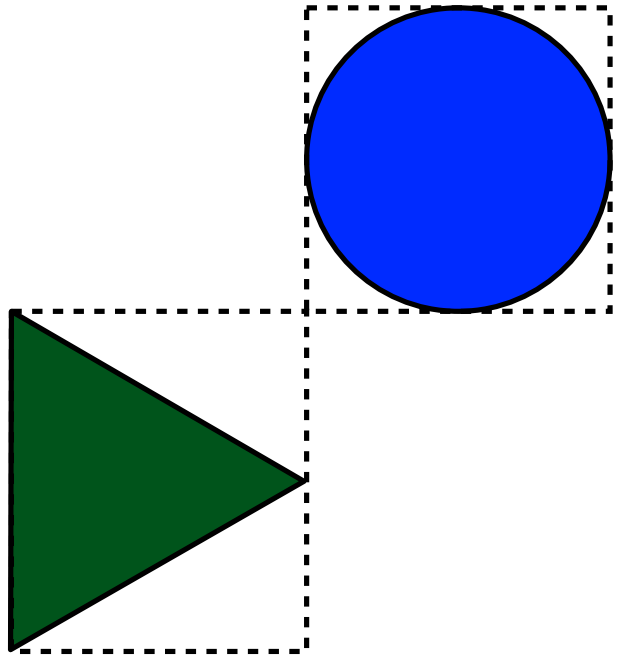
What happens when we want to rotate the triangle? The above is what the currently released version of `diagrams` does — I've gotten bug reports about it, and I agree it's a bug. But it's not clear what the real solution is.



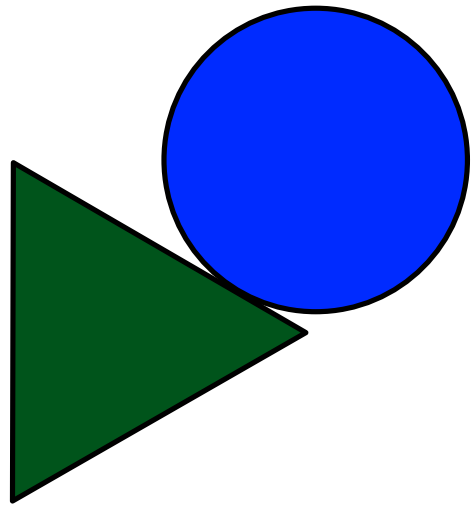
Why not just adjust the box? Not so fast—this requires knowing more about the diagram than just its bounding box in the first place!



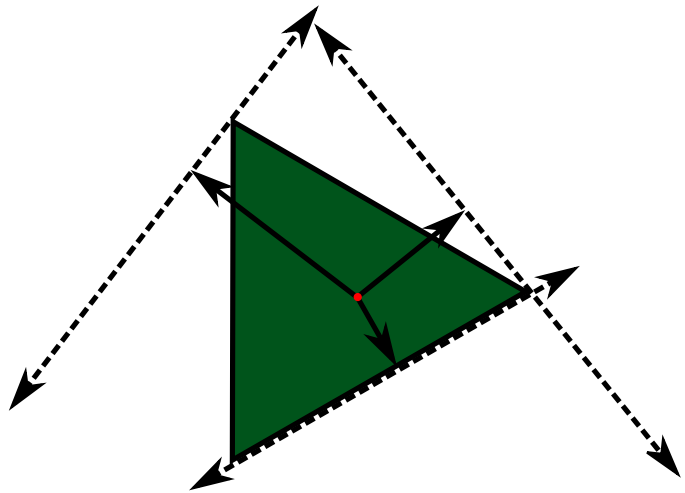
We could do this, but now transformations don't compose—rotating by A then by B gives a different bounding box than just rotating by A+B.



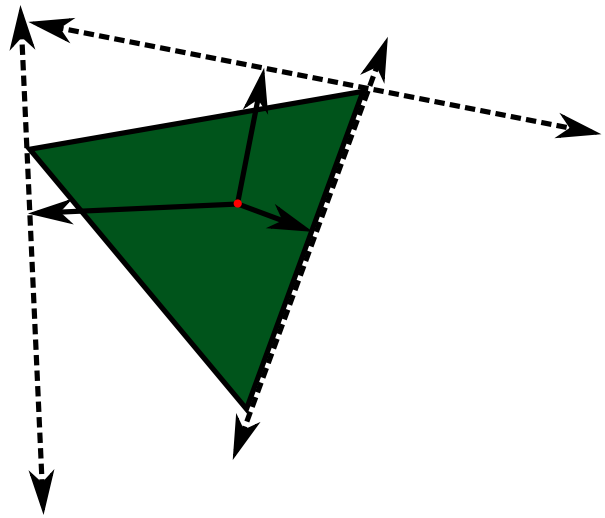
And what about putting things next to each other along a line that isn't vertical or horizontal? We get this...



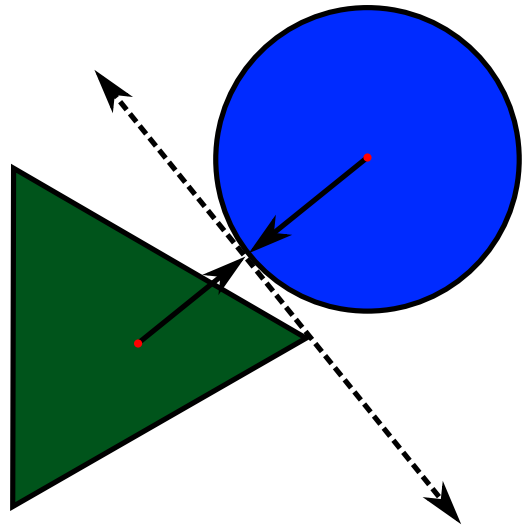
. . . instead of this.



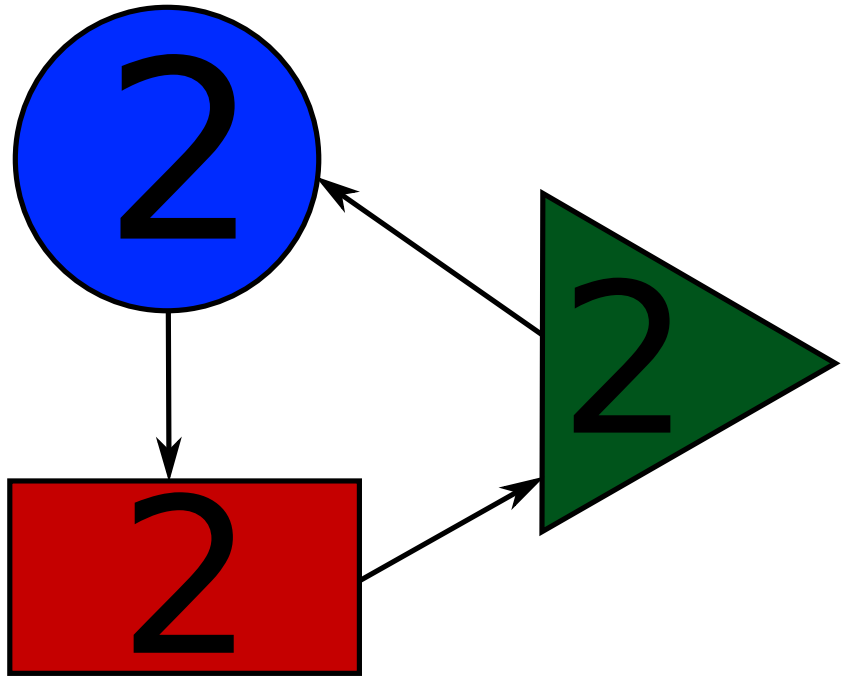
An elegant solution, suggested by Sebastian Setzer, is to have a function giving the distance to the nearest enclosing (hyper)plane in a given direction (relative to some distinguished base point). In some sense this gives us a functional representation of a convex bounding region.



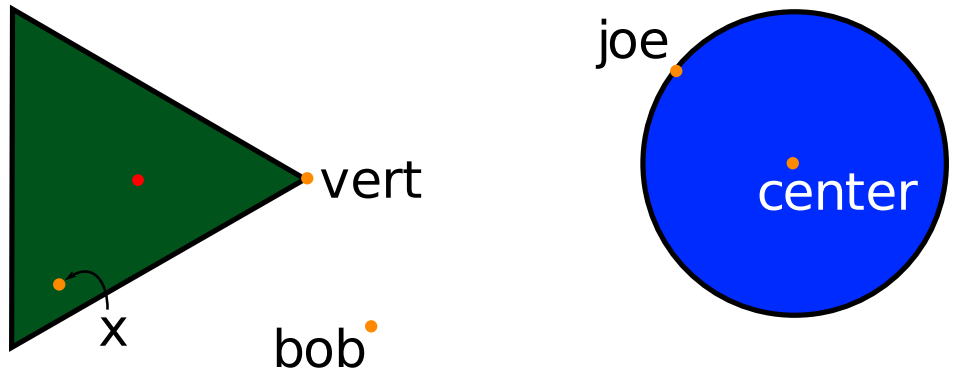
This obviously works beautifully with rotation! In fact, it works with any affine transformation.



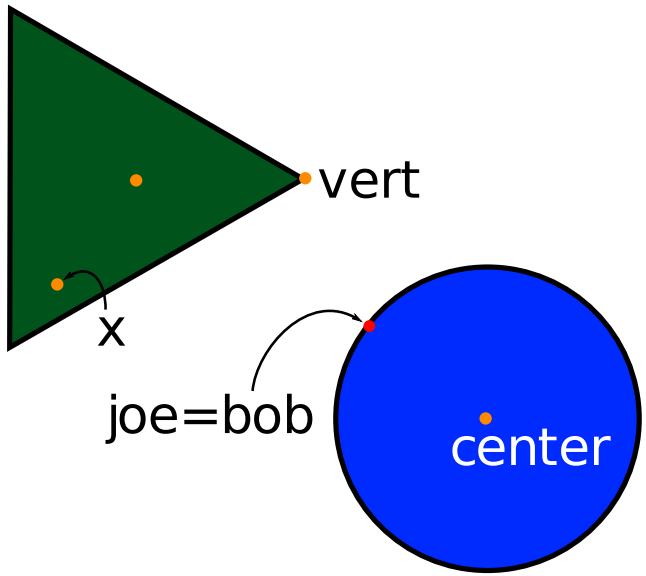
We can put two diagrams next to each other along any vector by putting them alongside a separating plane. It's not "perfect" — notice the small gap in this case — but it's pretty good, and simple/consistent; it's easy to predict what will happen.



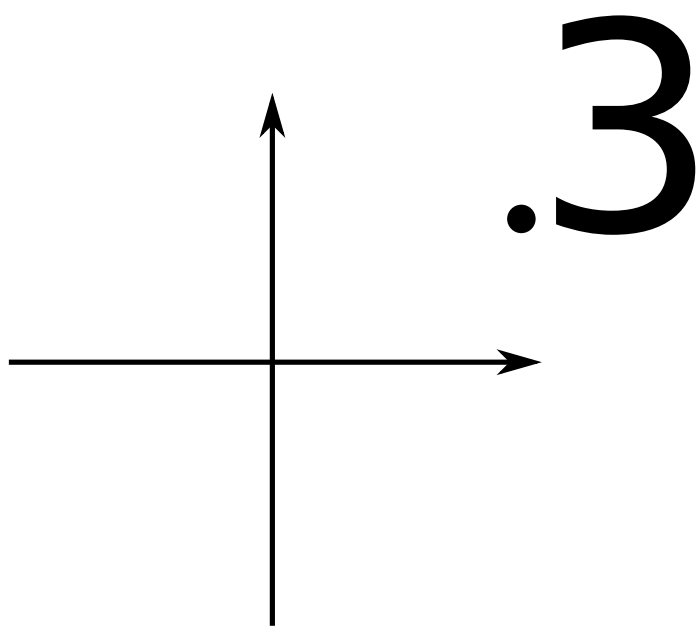
Here's something else that the current version gets wrong—there's no way to refer to previously laid out diagrams, so making a diagram like this one is very tedious; there's no good way to do it.



In the new version, every diagram will have an implicit distinguished "control point" thought of as the origin of a local coordinate system. Other points can be defined and named relative to the origin and each other, using a simple language for linear expressions.

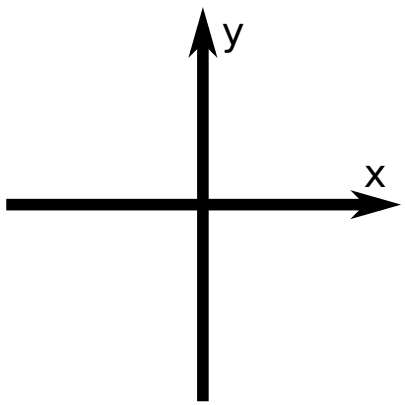


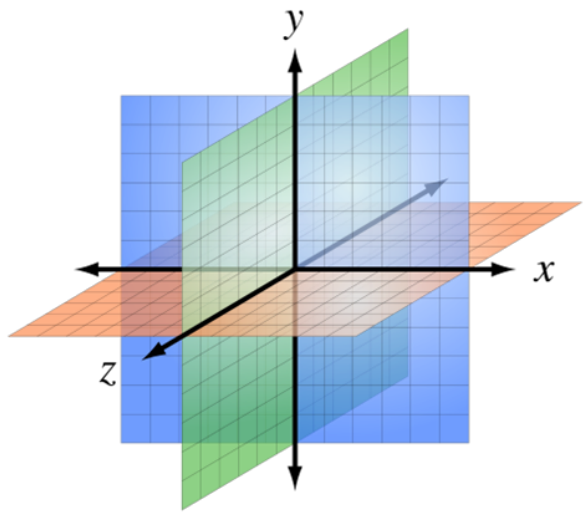
We can compose two diagrams by identifying a point from each, with the identified point becoming the new origin. All other diagram combinators can be implemented in terms of this operation.



A third limitation of the first version is that it only works for two-dimensional diagrams!

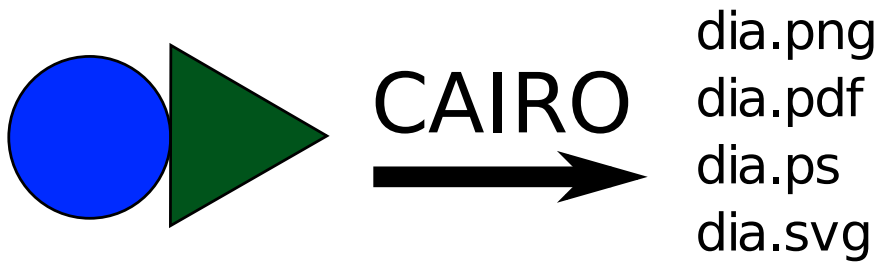
$\mathbb{R}^2$ $\mathbb{R}^3$ $\mathbb{R} \to V \ldots$



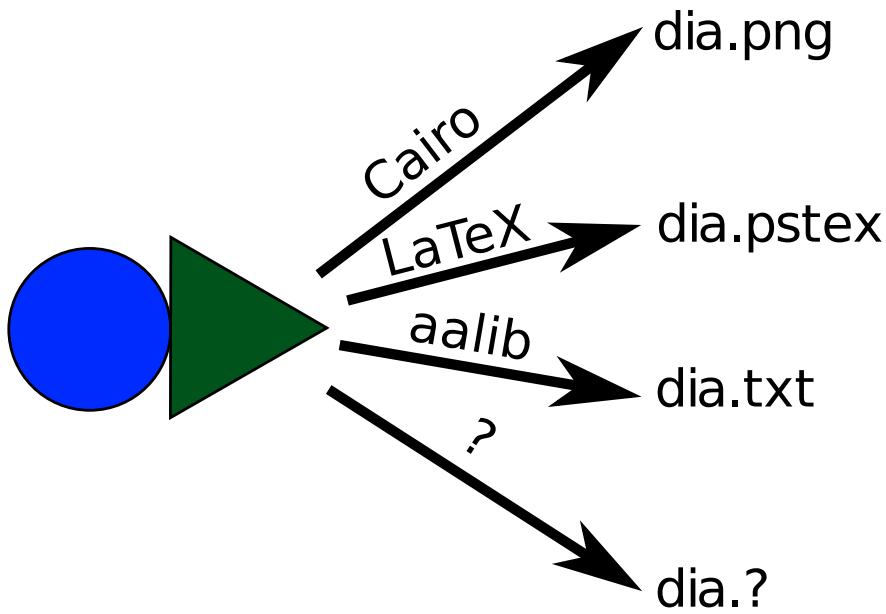


The new version is polymorphic over the vector space used, so we can have 2D diagrams, 3D diagrams, animations. . .


4

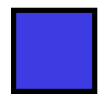


Something else the original version got wrong was to require Cairo as a rendering backend.

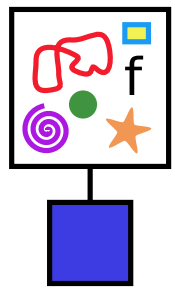


The new version will be more modular, allowing anyone to easily create a new rendering backend.

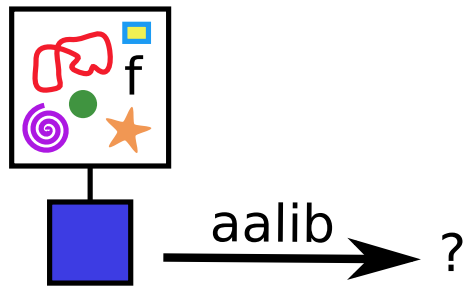
How you can help:



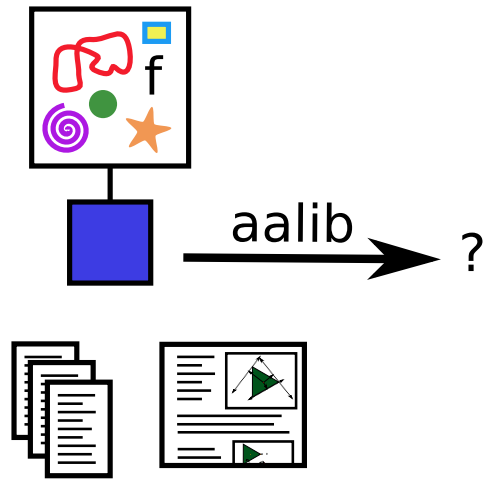
Help work on the core library.



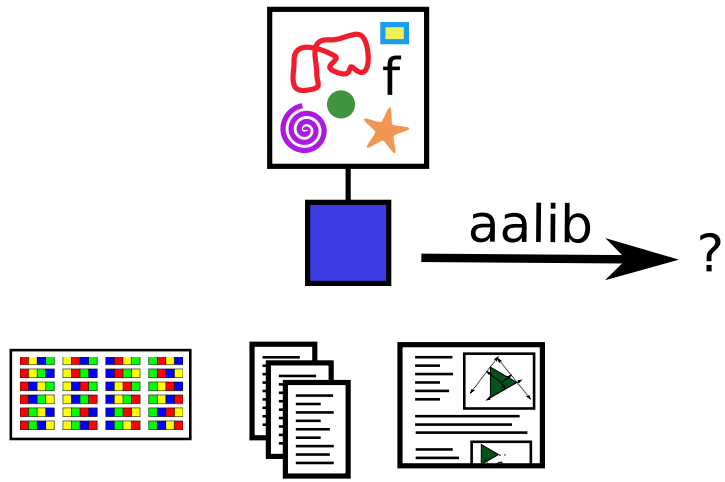
Help write a blessed standard library of convenient/common things implemented in terms of core primitives.



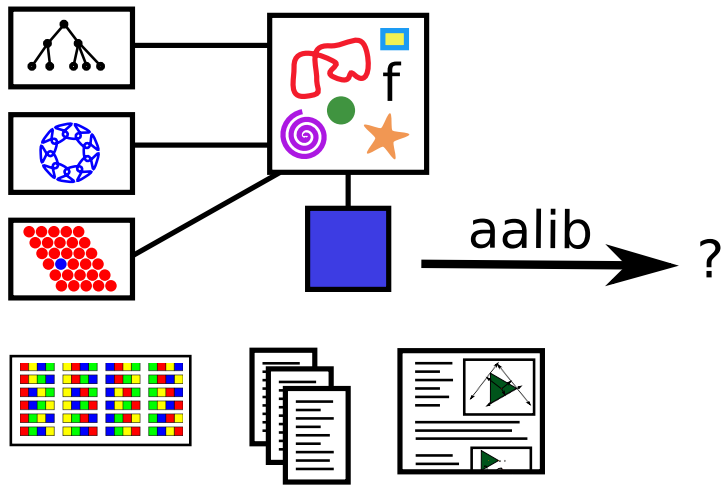
Write a backend.



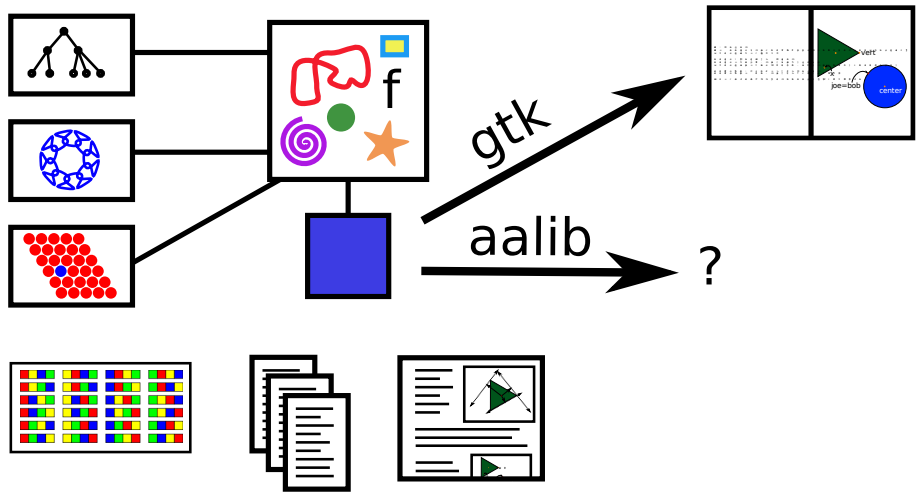
Write documentation or a tutorial.



Make some examples for fun and to help drive development.



Write a higher-level extension library.



Write an application for real-time visualization/editing, or a gitit plugin, or...

. . . design a better logo.