Project 1 : Shape Server

Pierre Sibut-Bourde

Trinity College Dublin

November 26, 2021

Contents

1 Assignment

In this project you will create a drawing eDSL (you can take your design from the Shape language we used in the lectures, or from the Pan language, or both). You will combine it with the two web eDSL languages we saw (Scotty and Blaze) to produce a web application capable of delivering images. The web app does not need to be interactive (that is, the images can be hard-coded in the app), the point is to create a DSL that could be used to specify images. You can use the JuicyPixels library to create the images (you will receive sample code for this. The library can serve as an almost drop-in replacement for the Ascii rendering layer in the example code, though there are also more efficient ways to make use of it). This project is overall worth 25% (the second project will be worth 35%). The project is due in at midnight on Friday November 12th (that's 5 weeks from the day of release). Any extensions must be agreed in advance, in writing. The project tasks, in detail, are:

- Design a suitable drawing eDSL (either by extending the the Shape, from scratch).
  - Provide at least the following basic shapes: Circle, Rectangle, Ellipse, Polygon (this last should be a closed convex polygon defined by a series of points).
  - Provide the following set of basic transformations: Scale, Rotate, Translate, and functions to combine them with shapes to produce drawings (note: you don't have to preserve the design of the original Shape language where these are maintained as a simple list of tuples).
  - Provide a way to specify colour for each shape. This can be either a simple case with a single colour, or a function which can provide gradients.
  - Finally, provide a way to mask images so that when one is overlaid with another the user can specify which one is seen. This can either be a simple boolean mask that allows only one image to show through, or a more sophisticated blending function that specifies how much each image contributes to each pixel.
- As a UI provide a Scotty application which can render some (hard-coded) sample images that demonstrate the result. The images should be returned as PNG graphics rendered using JuicyPixels. You should include the text of the DSL program that produced the image in the web page, so that the user of the web app can see how the image was produced (the idea is that a future improvement could be to allow the user to edit this text and re-render it.)
- Finally, implement at least one optimisation to the DSL program that is run before it is rendered. This could be ensuring that a shape that cannot be seen (because it is behind another shape that masks it off, for example) is removed from the drawing prior to rendering, or it could be that transformations are optimised (for example, multiple translations could be merged), or something else.

2 Shortlist of project requirements

- Shapes :
  During the project, I failed to produce a function to properly render a polygon, but every other shape can be displayed with a colour and with a hierarchy that indicates which shape is on top or on the bottom. I did not find a way to compute gradient colour for every shape, even though I tried at one point, before moving on simpler functions so that I can focus on more important requirements in the project (see below on DSL design).

- Web :
  On the server aspect of the project, I displayed HTML with CSS properly, and images were displayed after being generated from a set of examples hard-coded in the main function.

- Optimization :
  Working on optimization, I tried to concatenate transformations, and I only did it when the same transformation was applied (the real optimization would have been to work with a list of transformation and sort it when the two operation are commutative, for example similarities). I did not changed the way transformations were handled, however, but that could be a future challenge.

3 eDSL design

This section will present how I worked on the eDSL design.

- Firstly, I took the Shapes data type given in the weekly sample solution and I tried to add a rectangle, an ellipse and a polygon. For the first two objects, the idea was to think of them as scaled square and circle, respectively. We could define them either as a `Transform -> Shape -> Shape`, or as a `Double -> Double -> Shape`. However, I wanted to keep transform and shape modules clearly separated and I chose the second solution : the use of scaling was used in the `inside` check, as we divided coordinates of the point by the parameters of the shape, so that we would come back to simpler shapes (ellipse to circle with $\|.\|_2$, rectangle to square with $\|.\|_\infty$). For the polygon, it was not so easy to do an inside check, and I tried to work with inside angles comparing it to outside angles, yet I failed to produce a correct image in the end.

- Secondly, I needed to add colours to the shape, using an `PixelRBG8` object : one main problem about it was to actually get the colour of the shape: with no hierarchy at this point, I chose the first one of a list of pixels, yet I needed to find a way to handle black pixels : I used the `Maybe` monad in order to get a `Just` value when the point was inside a shape, and a `Nothing` value if not. At this point, the pixel color was the first one in the newly-created `(Shape, PixelRGB8)` data type. I needed to filter a list so that we can actually get a real pixel and not the `Maybe` pixel.

- Adding the hierarchy was actually to add an integer to the previous data type (that became `(Shape, PixelRGB8, Int)`) and to work with it. One main aspect of this change was to handle the hierarchy integer through the workflow, that actually changed to carry that important information from start to list filtering on values. For this part, we needed to sort a list of RGB Pixels associated with hierarchy, which was not so easy to do.

Fortunately enough, this eDSL could be handled by the rendering library, as long as we got an actual pixel from our processing, and not any other type of information. To do so, however, the types involved in the middle-end functions were clearly custom, and not really simple. I am not sure this workflow could be used if we were to compute the rendering with SVG on Blaze.

Final remark : I tried to comment the code as much as possible in order to explain the workflow. Please see `Shapes.hs` and `Main.hs` in the repository (also available on GitHub, see link on home page of the server). This report is also included directly on the website !