Functional Programming - Assignment 1 - Shape Server - Nov 27 2020
Ernests Kuznecovs - Student no. 17332791

Design Choices

The Shape language used the ShapeExample2 code from the exercises as a base. Not much was known about what the DSL was supposed to be, so I chose to gain foresight of drawing DSL by first expanding the current base and seeing what things could be improved.

I also chose the ShapeExample2 as it looked like a nice way to reason about creating algorithms for drawing shapes i.e for each point, ask whether it's inside the shape or not.

Throughout working on this assignment I kept wondering who the end user of this application would be. Whichever end user might be consuming these rendered pictures, I assumed that the speed at which the pictures rendered would allow the highest amount of flexibility for use cases where it can be applied.

I noticed that the more shapes that were being specified, the longer it would take to render by a significant amount. So I thought a redesign of the existing rendering pattern would be necessary, but in order to gain more insight about other aspects, and also to get the assignment done in time, I decided to keep going with the way it was.

There were two levels of shape language interfaces that we're being considered.

- The consumer of the Haskell shape library.
- The user of the shape server website.

It would be ideal that the consumer of the Haskell library could have maximum rendering performance while being presented with a simple and intuitive interface, with the possibility to build on top of the Shape library further and host it on the Shape Server for users on the website to use.

The level of interface the website would provide would depend on what most of the users would come and use it for.

Depending on how the website is interfaced, an interpretation system of the inputs from the website would need to be developed. If the input is almost like the Haskell shape language code, then it seems to me that a whole lexer and parser would need to be developed. On the other hand, if the interface is constrained to simple operations, possibly making it more accessible and simpler to use, then only a few different inputs would need to be accounted for in the interpreter, this might be simpler to build for a simple system, but the complexity might grow the more features added, while the lexer and parser would be able to maintain itself.

A more specific design question was: What arguments should a shape definition take, and what should it's constraints be?

This question came up as I was figuring out the convex polygon shape. The code requires for a valid convex set of points to be declared in clockwise/anti-clockwise order. I can imagine this would be a pain for a user if they can't find any documentation.

The shape language could provide its own error messages stating the points provided don't form a convex set.

The arguments to the rectangle and ellipse are also quite ambiguous which would cause confusion for the user.

It seems like calculating SVGs instead of rendering png bytestrings would remove a lot of the work needed on performance of calculating the shapes, and save the amount of bandwidth needed to send over the network, which would also improve the speed at which the shapes on the webpage can be updated.

Project Deliverables

The basic shapes: Circle, Rectangle, Ellipse, and Polygon are delivered.
Although the Polygon is lacking error checking and requires a specific order of point specification.

Transformations: All original transformations front the base Shape language work with the new shapes.

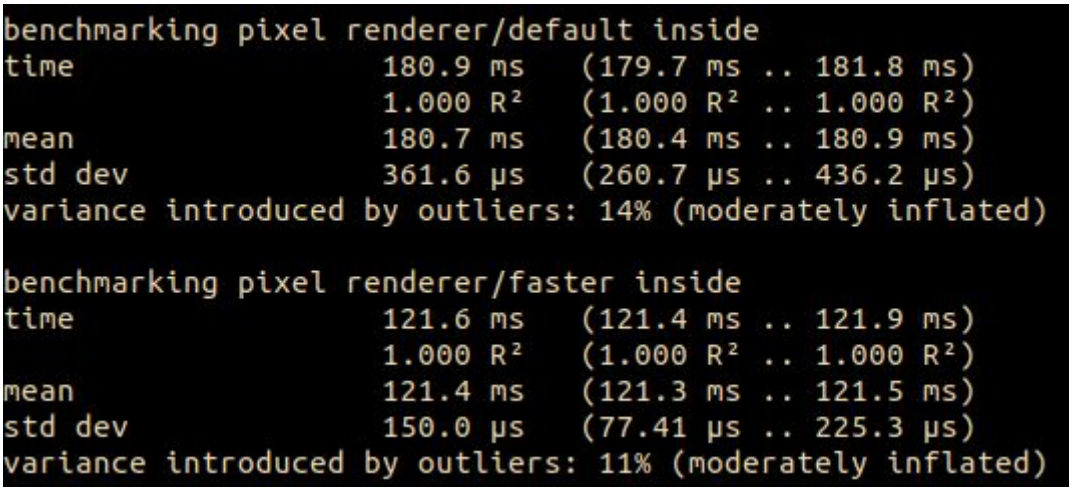
Specify colour of each shape: Each shape to be drawn takes an RGB value as an extra argument which specifies its colour.

Image mask: Provide the z-index when defining shapes, the shape with the highest z-index will be on top.

Scotty UI: The server serves a page with an img tag that as the src attribute, has the png bytestring encoded in it, removing the need for a second request to the server for a file.
The bytestring is generated by a predefined drawing that needs to be rendered, and a render is triggered upon a request of the route.

Optimization: The optimization consisted of not rendering the shapes with a lower z value than shapes with a higher one in the same position. This was done by sorting the list of shapes by their z value and when a point is being rendered, the algorithm will start from the highest z value and will move on to the next pixel as soon as it finds a shape it belongs to.

Doing a stack bench command:


benchmarking pixel renderer/default inside
time 180.9 ms (179.7 ms .. 181.8 ms)
1.000 R² (1.000 R² .. 1.000 R²)
mean 180.7 ms (180.4 ms .. 180.9 ms)
std dev 361.6 µs (260.7 µs .. 436.2 µs)
variance introduced by outliers: 14% (moderately inflated)
benchmarking pixel renderer/faster inside
time 121.6 ms (121.4 ms .. 121.9 ms)
1.000 R² (1.000 R² .. 1.000 R²)
mean 121.4 ms (121.3 ms .. 121.5 ms)
std dev 150.0 µs (77.41 µs .. 225.3 µs)
variance introduced by outliers: 11% (moderately inflated)