Image eDSL Server Documentation

20313326 Minjuan Luo

1. Introduction

This report outlines the design and implementation of the Haskell-based image rendering engine, which facilitates the generation of PNG graphics via an embedded domain-specific language (eDSL) for drawing. This project is composed of three main modules: Render.hs for rendering logic,
Shapes.hs for defining geometric shapes and affine transformations,
Main.hs which serves as the entry point, leveraging Scotty and Blaze to present a web application interface.

2. Design Choices

(1) Render.hs employs JuicyPixels for image creation and converts shapes into pixel data. The Window type defines the rendering viewport while sampling functions will calculate the precise pixel representation of shapes.

(2) Shapes.hs provides the eDSL for shape definition. It offers constructors for basic shapes and complex polygons along with transformation functions such as translation and rotation. The module also defines a color data type with predefined colors and a function for color blending.

Shape and Affine Transform Definition
Basic Shapes: Circle, Rectangle, Square, Ellipse, and Polygon.
Affine Transformations: Translate, Scale, Rotate, ShearX, ShearY, and ShearXY.

Color Definition
The eDSL incorporates an extensive palette of colors defined with RGBA values, such as red, green, blue, pink, aquamarine, salmon, gold, khaki, turquoise, and powderblue.

(3) Main.hs facilitates the rendering process for various shapes, applies color, and layers them using predefined transformations. It uses Scotty to map HTTP requests in order to correspond image responses, and then render the user interface through Blaze-generated HTML.

UI Logic:
Buttons are created for each shape, triggering the display of the corresponding image when clicked.
Descriptive text outlines the transformations and colors applied to each shape.

3. Optimization: switchingLayer

The switchingLayer function represents an optimization in the rendering process, managing the visibility of layered shapes. It adjusts the alpha value of shapes based on their overlap, optimizing the final image by preventing the rendering of fully obscured elements. This approach reduces computational overhead and improves performance, especially in complex drawings with multiple overlapping shapes.

4. Reflection

This project demonstrates the use of Haskell in creating an image rendering engine. However, challenges were encountered in optimizing performance, particularly in managing the overlapping of shapes with alpha blending. Additionally, the current implementation of the switchingLayer function assumes certain overlaps. Hence the program still has the potential need for a more dynamic overlap detection mechanism in future work.

As SVG is a vector-based format, it will inherently manage overlaps differently than raster images. Transitioning to SVG layering even CSS for managing visibility and transparency could lead to a more declarative style of defining shape visibility within the SVG DOM, potentially simplifying the rendering logic.