`wumpus-core` Guide

Stephen Tetley

September 18, 2010

1 About `wumpus-core`

`wumpus-core` is a Haskell library for generating 2D vector pictures. It was written with portability as a priority, so it has no dependencies on foreign C libraries. Output to PostScript and SVG (Scalable Vector Graphics) is supported.

`wumpus-core` is rather primitive, the basic drawing objects are paths and text labels. A second library `wumpus-basic` contains code for higher level drawing but it is still missing main functionalities e.g. connectors, arrowheads.

Although `wumpus-core` is heavily inspired by PostScript it avoids PostScript's notion of an (implicit) current point and the movements `lineto`, `moveto` etc., instead `wumpus-core` aims for a more *coordinate free* style.

2 Exposed modules

`wumpus-core` exports the following modules:

`Wumpus.Core.` Top-level *shim* module to import all the exposed modules. Some internal data types are also exported as opaque - the implementation is hidden, but the type name is exposed so it can be used in the type signatures of *userland* functions. Typically, where these data types need to be *instantiated*, smart constructors are provided.

`Wumpus.Core.AffineTrans.` The standard affine transformations (scaling, rotation, translation) implemented as type classes, with a of derived operations - reflections about the X or Y axes, rotations through common angles.

`Wumpus.Core.BoundingBox.` Data type representing bounding boxes and operations on them. Bounding boxes are important for Pictures and they support the definition of *Picture composition operators*.

`Wumpus.Core.Colour.` A single colour type `RGBi` is supported. This type defines colour as a triple of integers (Word8) - black is 0, 0, 0; white is 255, 255, 255. Some named colours are defined, although they are hidden by the top level shim module to avoid name clashes. `Wumpus.Core.Colour` can be imported directly if the named colours are required.

Wumpus.Core.FontSize. Various calculations for font size metrics. Generally not useful to a user but exposed so that variations of the standard Label type are possible.

Wumpus.Core.Geometry. Usual types an operations from affine geometry - points, vectors and frames. The `Pointwise` type class which is essential for defining transformable drawable types.

Wumpus.Core.GraphicsState. Data types modelling the attributes of PostScript's graphics state (stroke style, dash pattern, etc.). Note `wumpus-core` annotates primitives - paths, text labels - with their rendering style. PostScript has a mutable graphics state, changing via inheritance how the curent object is drawn.

Wumpus.Core.OutputPostScript. Functions to write PostScript or encapsulated PostScript files.

Wumpus.Core.OutputSVG. Functions to write SVG files.

Wumpus.Core.Picture. Operations to build *pictures* - paths and labels within an affine frame. Generally the functions here are convenience constructors for types from the hidden module `Wumpus.Core.PictureInternal`. The types from `PictureInternal` are exported as opaque signatures by `Wumpus.Core.WumpusTypes`.

Wumpus.Core.PtSize. Text size calculations in `Core.FontSize` use points (i.e. 1/72 of an inch). The `PtSize` module is a numeric type to represent them.

Wumpus.Core.TextEncoder. Types for handling non-ASCII character codes. This module is perhaps under-cooked although it appears adequate for Latin-1.

Wumpus.Core.TextLatin1. A instance of the TextEncoder type for mapping Latin 1 characters to the PostScript and SVG escape characters. Typically this encoder is associated with the fonts - Helvetica, Courier and Times-Roman.

Wumpus.Core.TextSymbol. A instance of the TextEncoder type for the Symbol font.

Wumpus.Core.VersionNumber. Current version number of `wumpus-core` .

Wumpus.Core.WumpusTypes. This module collects internal types for Pictures, Paths etc. and presents them as opaque types - i.e. their constructors are hidden.

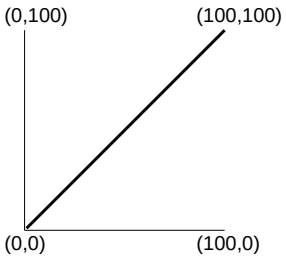

Figure 1: The world frame, with origin at the bottom left.

3 Drawing model

`wumpus-core` has two main drawable primitives *paths* and text *labels*, ellipses are also a primitive although this is a concession to efficiency when drawing dots (which would otherwise require 4 to 8 Bezier arcs to describe). Paths are made from straight sections or Bezier curves, they can be open and *stroked* to produce a line; or closed and *stroked*, *filled* or *clipped*. Labels represent a single horizontal line of text - multiple lines must be composed from multiple labels.

Primitives are attributed with drawing styles - font name and size for labels; line width, colour, etc. for paths. Drawing Primitives is unfortunately complicated due to the need to support hyperlinks in SVG output. Primitives have to be lifted to a `PrimElement` before they can be placed within a `Picture` - using the shorthand constructors in `Wumpus.Core.Picture` does this lifting automatically. The function `frame` assembles a list of primitives into a `Picture` with a standard affine frame where the origin is at (0,0) and the X and Y axes have the unit bases.

`wumpus-core` uses the same picture frame as PostScript where the origin at the bottom left, see Figure 1. This contrasts to SVG where the origin at the top-left. When `wumpus-core` generates SVG, the whole picture is produced within a matrix transformation [1.0, 0.0, 0.0, -1.0, 0.0, 0.0] that changes the picture to use PostScript coordinates. This has the side-effect that text is otherwise drawn upside down, so `wumpus-core` adds a rectifying transform to each text element.

Once labels and paths are assembled as a *Picture* they are transformable with the usual affine transformations (scaling, rotation, translation).

Once assembled into pictures graphics properties (e.g. colour) are opaque - it is not possible to write a transformation function that turns a picture blue. In some ways this is a limitation - for instance, the `Diagrams` library appears to support some notion of attribute overriding; however it does keep `wumpus-core` conceptually simple. If one wanted to draw blue or red arrows with `wumpus-core` , one would make drawing colour a parameter of the arrow creation function.

4 Affine transformations

For affine transformations Wumpus uses the `Matrix3'3` data type to represent 3x3 matrices in row-major form. The constructor `(M3'3 a b c d e f g h i)` builds this matrix:

$$\begin{matrix} a & b & c \\ d & e & f \\ g & h & i \end{matrix}$$

Note, in practice the elements g and h are superflous. They are included in the data type to make it match the typical representation from geometry texts. Also, typically matrices will implicitly created with functions from the `Core.Geometry` and `Core.AffineTrans` modules.

For example a translation matrix moving 10 units in the X-axis and 20 in the Y-axis will be encoded as `(M3'3 1.0 0.0 10.0 0.0 1.0 20.0 0.0 0.0 1.0)`

$$\begin{matrix} 1.0 & 0.0 & 10.0 \\ 0.0 & 1.0 & 20.0 \\ 0.0 & 0.0 & 1.0 \end{matrix}$$

Affine transformations are communicated to PostScript as `concat` commands. Effectively `wumpus-core` performs no transformations itself, delegating all the work to PostScript or SVG. This means transformations can generally be located in the output if a picture needs to be debugged, though as this might not be very helpful in practice. Internally `wumpus-core` only performs the transformation on the pictures bounding box - it needs to do this so transformed pictures can still be composed with the `picBeside` combinator.

PostScript uses column-major form and uses a six element matrix rather than a nine element one. The translation matrix above would produce this concat command:

```
[1.0 0.0 0.0 1.0 10.0 20.0] concat
```

Similarly, it would be communicated to SVG via a `group` element:

```
<g transform="matrix(1.0, 0.0, 0.0, 1.0, 10.0, 20.0)"> ... </g>
```

For efficiency reasons `wumpus-core` supports some transformations on Primitives. These are not affine transformations as Primitives are not in an affine frame until they are lifted to Pictures (Primitives have no notion of origin). For Paths, all the transformations are precomputed before the output is generated. Unfortunately scaling and rotation cannot be precomputed for labels and ellipses, so matrix operations are generated in the PostScript and SVG output.

5 Font handling

Font handling is quite primitive in `wumpus-core` . The bounding box of text label is only estimated - based on the length of the label's string rather than the metrics of the individual letters encoded in the font. Accessing the glyph metrics in a font would require a font loader to read TrueType font files. This would be a significant development effort, probably larger than the effort put into `wumpus-core` itself; for `wumpus-core` 's intended use - producing diagrams and pictures rather than high quality text - its primitive font handling is not such a draw back.

In both PostScript and SVG mis-named fonts can cause somewhat inscrutable printing anomalies - usually falling back to a default font but not always. At worst, PostScript may do no subsequent drawing after a font load error. `wumpus-core` uses `scalefont` in the generated PostScript, this semingly works for any integer size and not just the regular font sizes (10, 12, 18, 24, 36). Older versions of `wumpus-core` mention that using non-standard sizes may cause font loading problems, however this does not appear to be the case.

The following table lists PostScript fonts and their SVG equivalents, the package `wumpus-basic` includes a module `Wumpus.Basic.SafeFonts` encoding the fonts in this list to avoid typographical slips.

PostScript name	SVG name
Times-Roman	Times New Roman
Times-Italic	Times New Roman - style="italic"
Times-Bold	Times New Roman - font-weight="bold"
Times-BoldItalic	Times New Roman - style="italic", font-weight="bold"
Helvetica	Helvetica
Helvetica-Oblique	Helvetica - style="italic"
Helvetica-Bold	Helvetica - font-weight="bold"
Helvetica-Bold-Oblique	Helvetica - style="italic", font-weight="bold"
Courier	Courier New
Courier-Oblique	Courier New - style="italic"
Courier-Bold	Courier New - font-weight="bold"
Courier-Bold-Oblique	Courier New - style="italic", font-weight="bold"
Symbol	Symbol

6 Acknowledgments

PostScript is a registered trademark of Adobe Systems Inc.