Graphics.Gloss

Safe Haskell	None
Language	Haskell98

Gloss hides the pain of drawing simple vector graphics behind a nice data type and a few display functions.

Getting something on the screen is as easy as:

```
import Graphics.Gloss
main = display (InWindow "Nice Window" (200, 200) (10, 10)) white (Circle 80)
```

Once the window is open you can use the following:

- Quit - esc-key.
- Move Viewport - left-click drag, arrow keys.
- Rotate Viewport - right-click drag, control-left-click drag, or home/end-keys.
- Zoom Viewport - mouse wheel, or page up/down-keys.

Animations can be constructed similarly using the `animate`.

If you want to run a simulation based around finite time steps then try `simulate`.

If you want to manage your own key/mouse events then use `play`.

Gloss uses OpenGL under the hood, but you don't have to worry about any of that.

Gloss programs should be compiled with `-threaded`, otherwise the GHC runtime will limit the frame-rate to around 20Hz.

To build gloss using the GLFW window manager instead of GLUT use `cabal install gloss --flags="GLFW -GLUT"`

```
Release Notes:

For 1.9:
 Thanks to Elise Huard
  * Split rendering code into gloss-rendering package.

For 1.8
 Thanks to Francesco Mazzoli
  * Factored out ViewPort and ViewState handling into user visible modules.

For 1.7:
  * Tweaked circle level-of-detail reduction code.
  * Increased frame rate cap to 100hz.
  Thanks to Doug Burke
  * Primitives for drawing arcs and sectors.
  Thanks to Thomas DuBuisson
  * IO versions of animate, simulate and play.

For 1.6:
  Thanks to Anthony Cowley
  * Full screen display mode.

For 1.5:
  * O(1) Conversion of ForeignPtrs to bitmaps.
  * An extra flag on the Bitmap constructor allows bitmaps to be cached
    in texture memory between frames.
```

For more information, check out http://gloss.ouroborus.net.

Documentation

module Graphics.Gloss.Data.Picture

module Graphics.Gloss.Data.Color

module Graphics.Gloss.Data.Bitmap

data **Display** Source

Describes how Gloss should display its output.

Constructors

InWindow String (Int, Int) (Int, Int)	Display in a window with the given name, size and position.
FullScreen (Int, Int)	Display full screen with a drawing area of the given size.

Instances

Eq Display Source

Read Display Source

Show Display Source

display Source

:: Display	Display mode.
-> Color	Background color.
-> Picture	The picture to draw.
-> IO ()	

Open a new window and display the given picture.

Use the following commands once the window is open:

- Quit - esc-key.
- Move Viewport - left-click drag, arrow keys.
- Rotate Viewport - right-click drag, control-left-click drag, or home/end-keys.
- Zoom Viewport - mouse wheel, or page up/down-keys.

animate Source

:: Display	Display mode.
-> Color	Background color.
-> (Float -> Picture)	Function to produce the next frame of animation. It is passed the time in seconds since the program started.
-> IO ()	

Open a new window and display the given animation.

Once the window is open you can use the same commands as with `display`.

simulate Source

:: Display	Display mode.
-> Color	Background color.
-> Int	Number of simulation steps to take for each second of real time.
-> model	The initial model.

`-> (model -> Picture)`	A function to convert the model to a picture.
`-> (ViewPort -> Float -> model -> model)`	A function to step the model one iteration. It is passed the current viewport and the amount of time for this simulation step (in seconds).
`-> IO ()`	

Run a finite-time-step simulation in a window. You decide how the model is represented, how to convert the model to a picture, and how to advance the model for each unit of time. This function does the rest.

Once the window is open you can use the same commands as with `display`.

---

play Source

`:: Display`	Display mode.
`-> Color`	Background color.
`-> Int`	Number of simulation steps to take for each second of real time.
`-> world`	The initial world.
`-> (world -> Picture)`	A function to convert the world a picture.
`-> (Event -> world -> world)`	A function to handle input events.
`-> (Float -> world -> world)`	A function to step the world one iteration. It is passed the period of time (in seconds) needing to be advanced.
`-> IO ()`	

Play a game in a window. Like `simulate`, but you manage your own input events.
