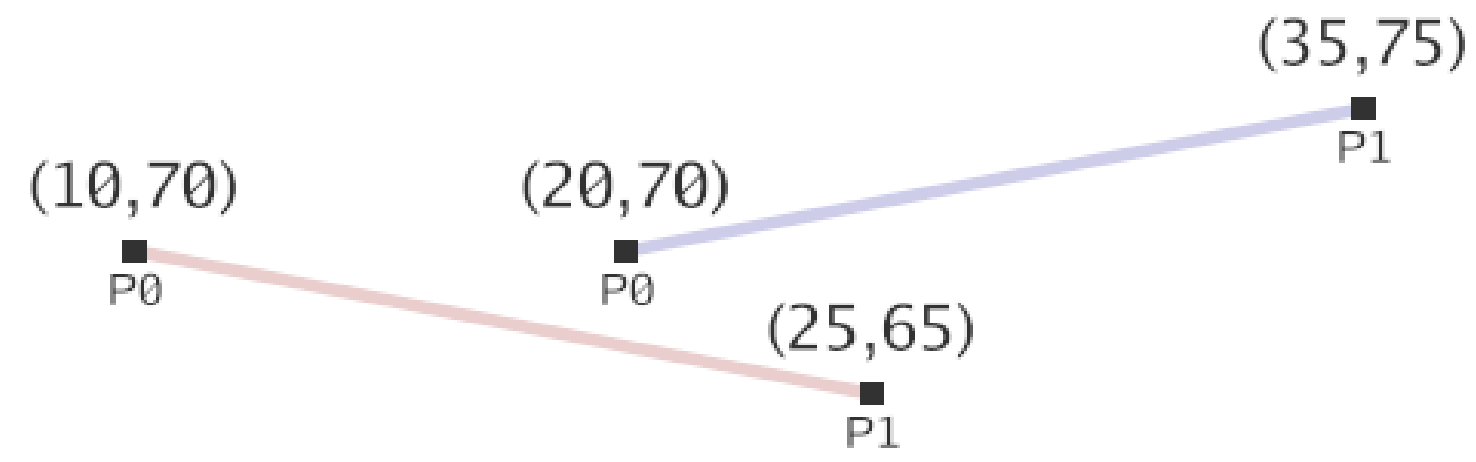


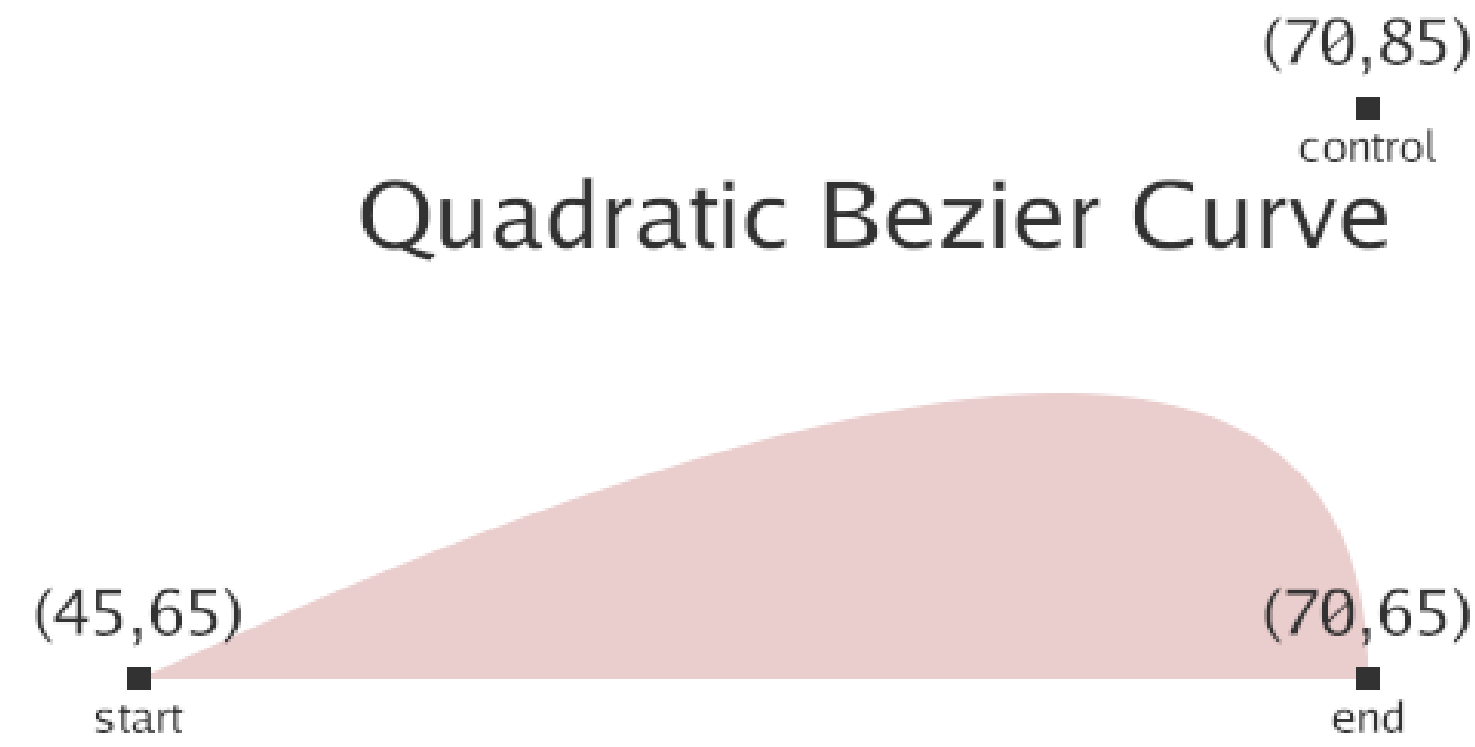
Canvas API

A canvas API for Gio applications using high-level objects and a percentage-based coordinate system (<https://github.com/ajstarks/giocanvas>)

Line



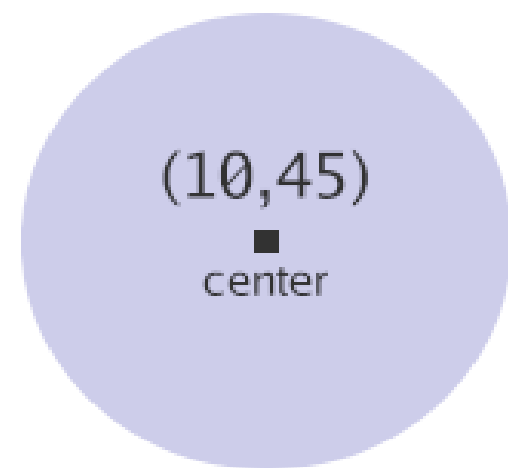
Quadratic Bezier Curve



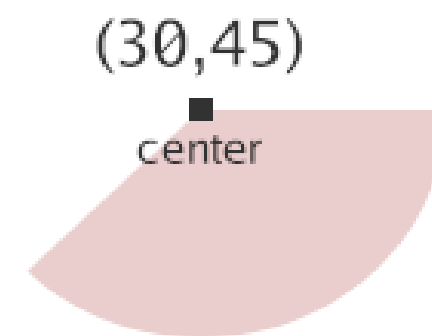
Rectangle



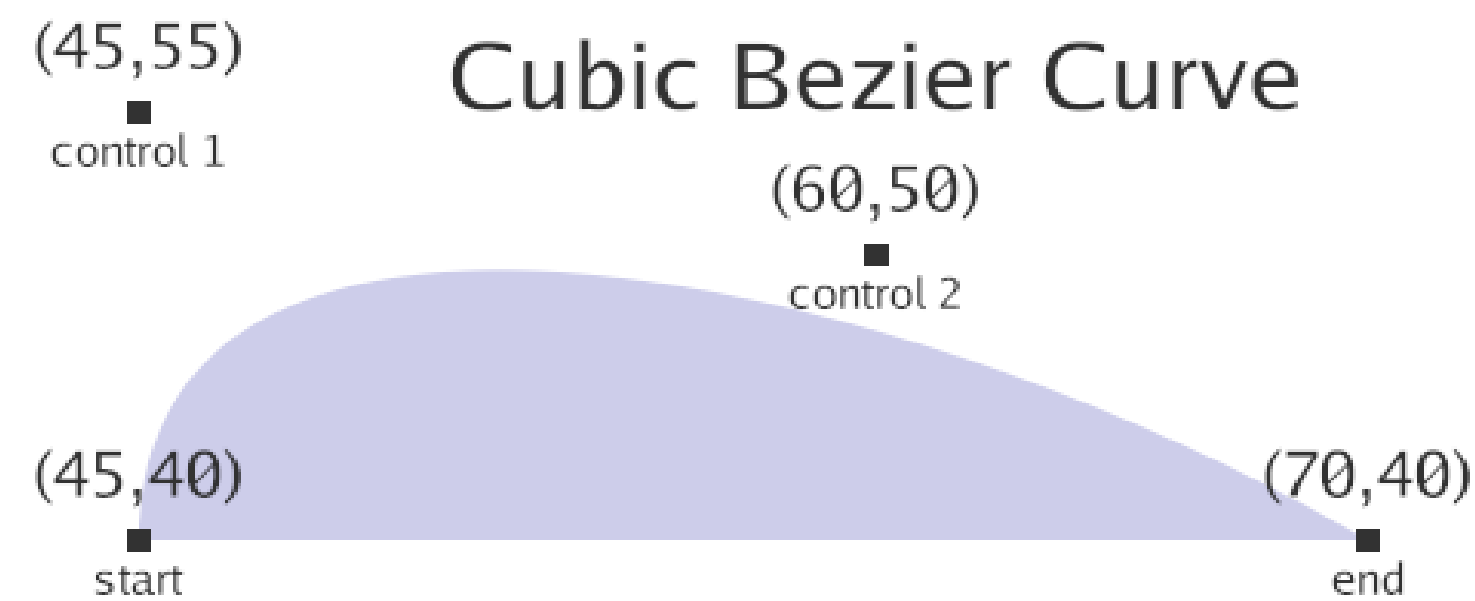
Circle



Arc



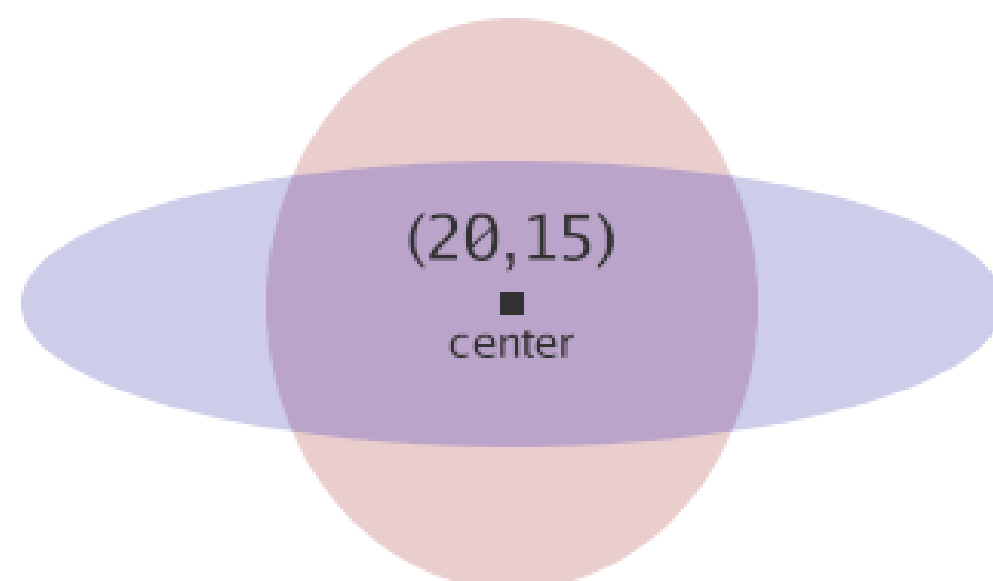
Cubic Bezier Curve



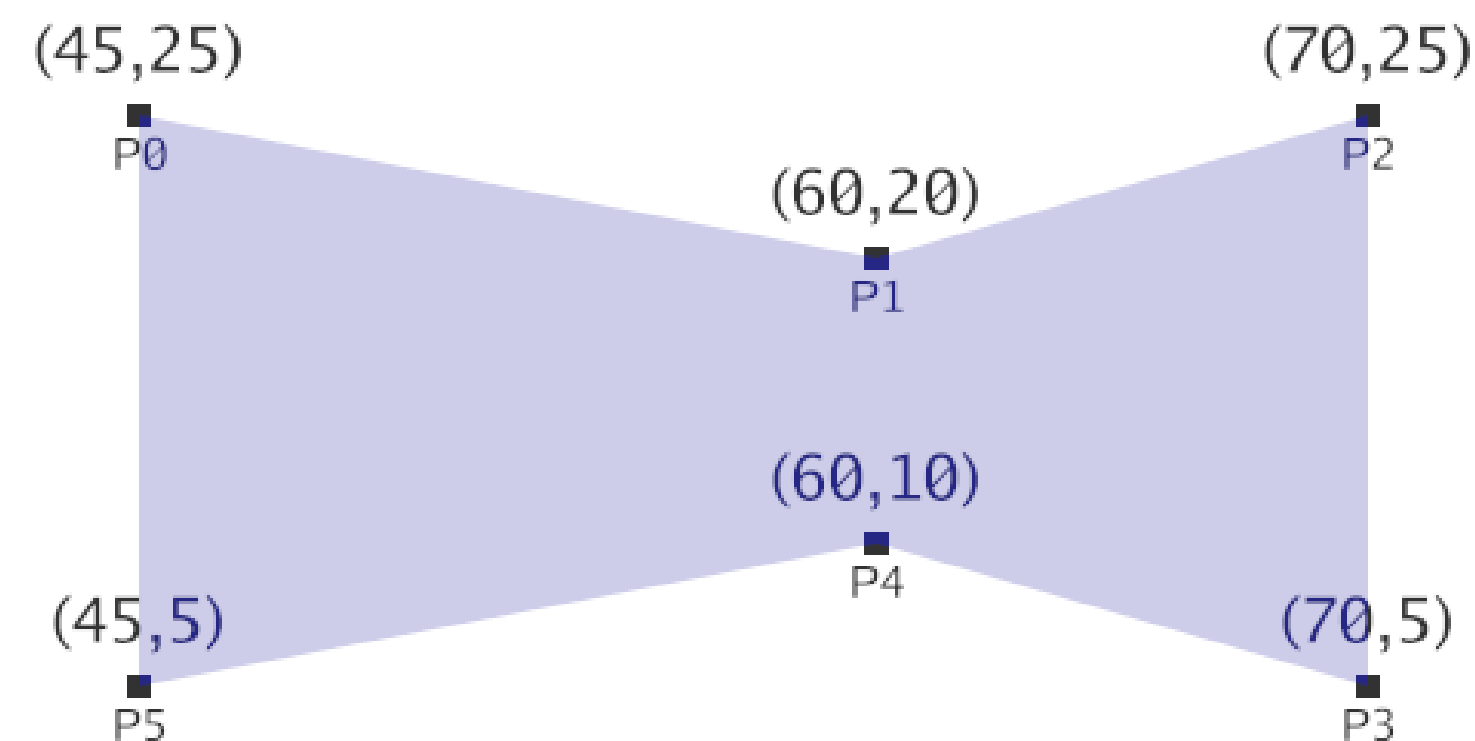
Square



Ellipse



Polygon



Image



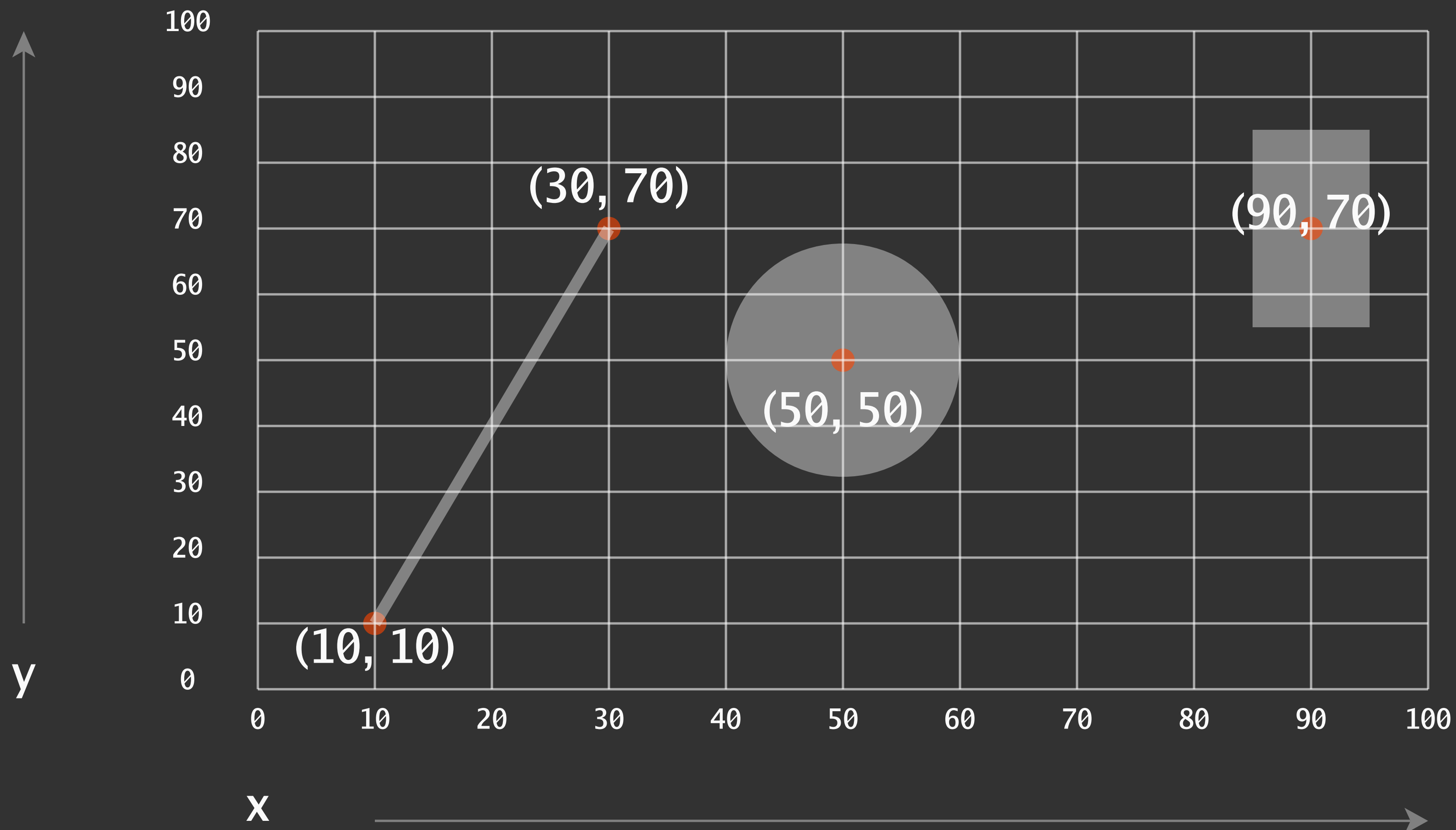
Motivation

The desire for a high-level Go API for developers and designers to think in terms of high level objects that make up a visual display. The objects will be familiar to anyone using a modern illustration program (text, images, lines, arcs, circles, curves, etc). The API should facilitate the artful arrangement of these elements on a scalable 2D canvas.

Use Cases: Information Displays, Data Visualization, Creative Coding, Presentations

The Percent Grid

Using the Percent Grid



Rect(90, 70, ...)

Circle(50, 50, ...)

Line(10, 10, 30, 70, ...)

Methods on *Canvas

Make a new canvas

`NewCanvas (width, height float32, e system.FrameEvent) *Canvas`

Place an image from file

`Image (name string, x, y float32, w, h int, scale float32)`

Place an image from image.Image

`Img (im image.Image, x, y float32, w, h int, scale float32)`

Line from (x0,y0) to (x1,y1)

`Line (x0, y0, x1, y1, size float32, stroke color.RGBA)`

Circle centered at (x,y), radius r

`Circle (x, y, r float32, fill color.RGBA)`

Ellipse centered at (x,y), radii (w,h)

`Ellipse (x, y, w, h float32, fill color.RGBA)`

Square centered at (x,y)

`Square (x, y, w float32, fill color.RGBA)`

Rectangle centered at (x,y)

`Rect (x, y, w, h float32, fill color.RGBA)`

Arc at (x,y), radius r, angle a1–a2

`Arc (x, y, r float32, a1, a2 float64, fill color.RGBA)`

Cubic Bezier Curve

`CubeCurve (x, y, cx1, cy1, cx2, cy2, ex, ey float32, fill color.RGBA)`

Quadratic Bezier Curve

`Curve (x, y, cx, cy, ex, ey float32, fill color.RGBA)`

Filled Polygon

`Polygon (x, y []float32, fill color.RGBA)`

Left–Aligned Text

`Text (x, y, size float32, s string, fill color.RGBA)`

Centered Text

`CText (x, y, size float32, s string, fill color.RGBA)`

End–Aligned Text

`EText (x, y, size float32, s string, fill color.RGBA)`

Transformations and Convenience Functions

Rotate at (x,y) around angle

`Rotate (x, y, angle float32) op.StackOp`

Scale at (x,y) by factor

`Scale (x, y, factor float32) op.StackOp`

Shear at (x,y) by angle1, angle2

`Shear (x, y, ax, ay float32) op.StackOp`

Translate by (x,y)

`Translate (x, y float32) op.StackOp`

End Transformation

`EndTransform(stack op.StackOp)`

Map one range to another

`MapRange (value, low1, high1, low2, high2 float64) float64`

Show annotated coordinates

`Coord (x, y, size float32, s string, fill color.RGBA)`

Set the background color

`Background (fill color.RGBA)`

Show a grid

`Grid (x, y, w, h, size, interval float32, linecolor color.RGBA)`

Polar to Cartesian (radians)

`Polar (cx, cy, r, theta float32) (float32, float32)`

Polar to Cartesian (degrees)

`PolarDegrees(cx, cy, r, theta float32) (float32, float32)`

giocanvas hello, world

```
package main

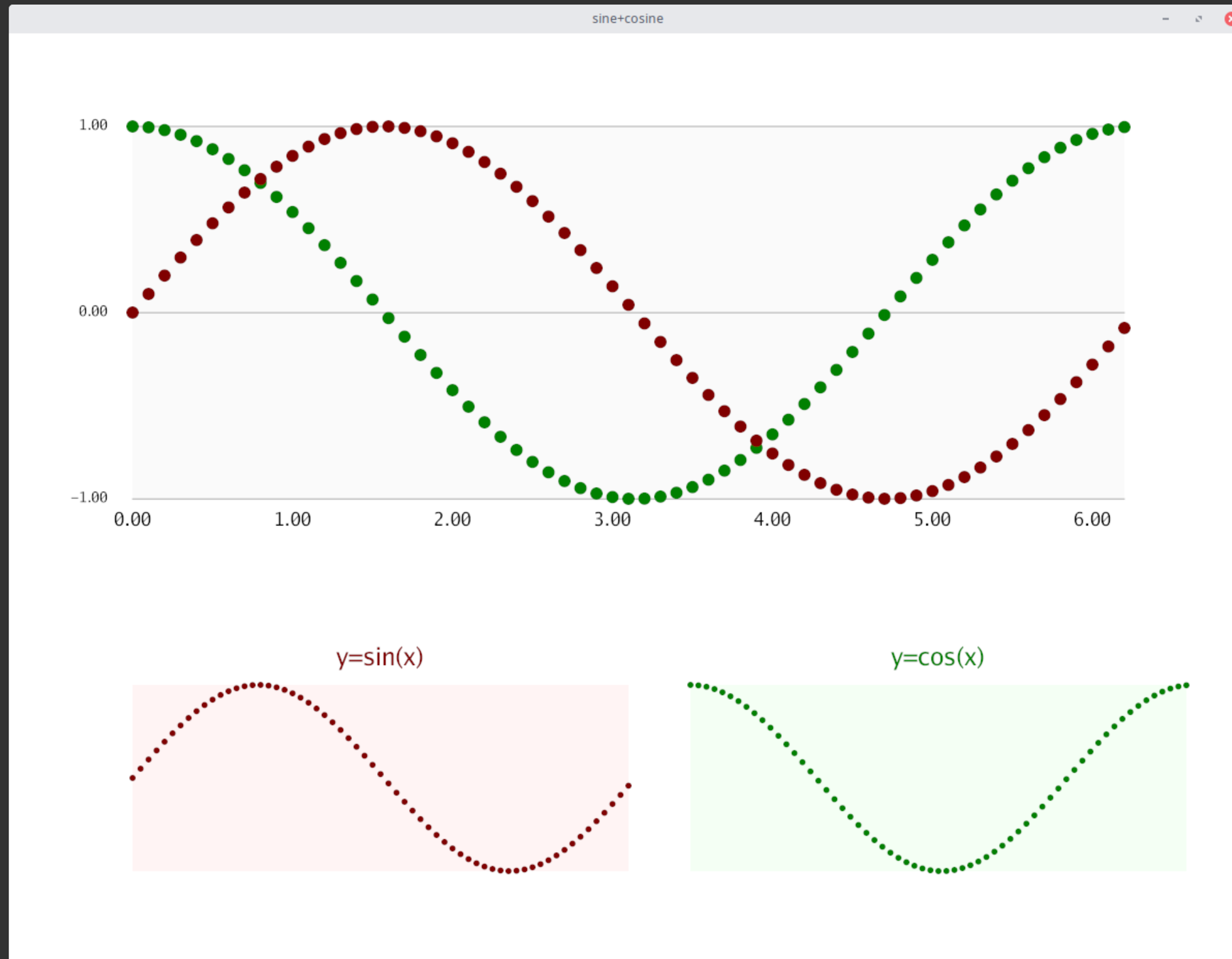
import (
    "gioui.org/app"
    "gioui.org/io/system"
    "gioui.org/unit"
    gc "github.com/ajstarks/giocanvas"
)

func hello(title string, width, height float32) {
    win := app.NewWindow(app.Title(title), app.Size(unit.Px(width), unit.Px(height)))
    for e := range win.Events() {
        switch e := e.(type) {
        case system.FrameEvent:
            canvas := gc.NewCanvas(width, height, e)
            canvas.Background(gc.ColorLookup("black"))
            canvas.Circle(50, 0, 50, gc.ColorLookup("blue"))
            canvas.TextMid(50, 20, 10, "hello, world", gc.ColorLookup("white"))
            canvas.CenterImage("earth.jpg", 50, 70, 1000, 1000, 30)
            e.Frame(canvas.Context.Ops)
        }
    }
}

func main() {
    go hello("hello", 1000, 1000)
    app.Main()
}
```



giocanvas/chart



giocanvas/chart data structures

```
// NameValue is a name,value pair
type NameValue struct {
    label string
    note  string
    value float64
}

// ChartBox holds the essential data for making a chart
type ChartBox struct {
    Title          string
    Data           []NameValue
    Color          color.RGBA
    Top, Bottom, Left, Right float64
    Minvalue, Maxvalue    float64
    Zerobased         bool
}
```

methods on *ChartBox

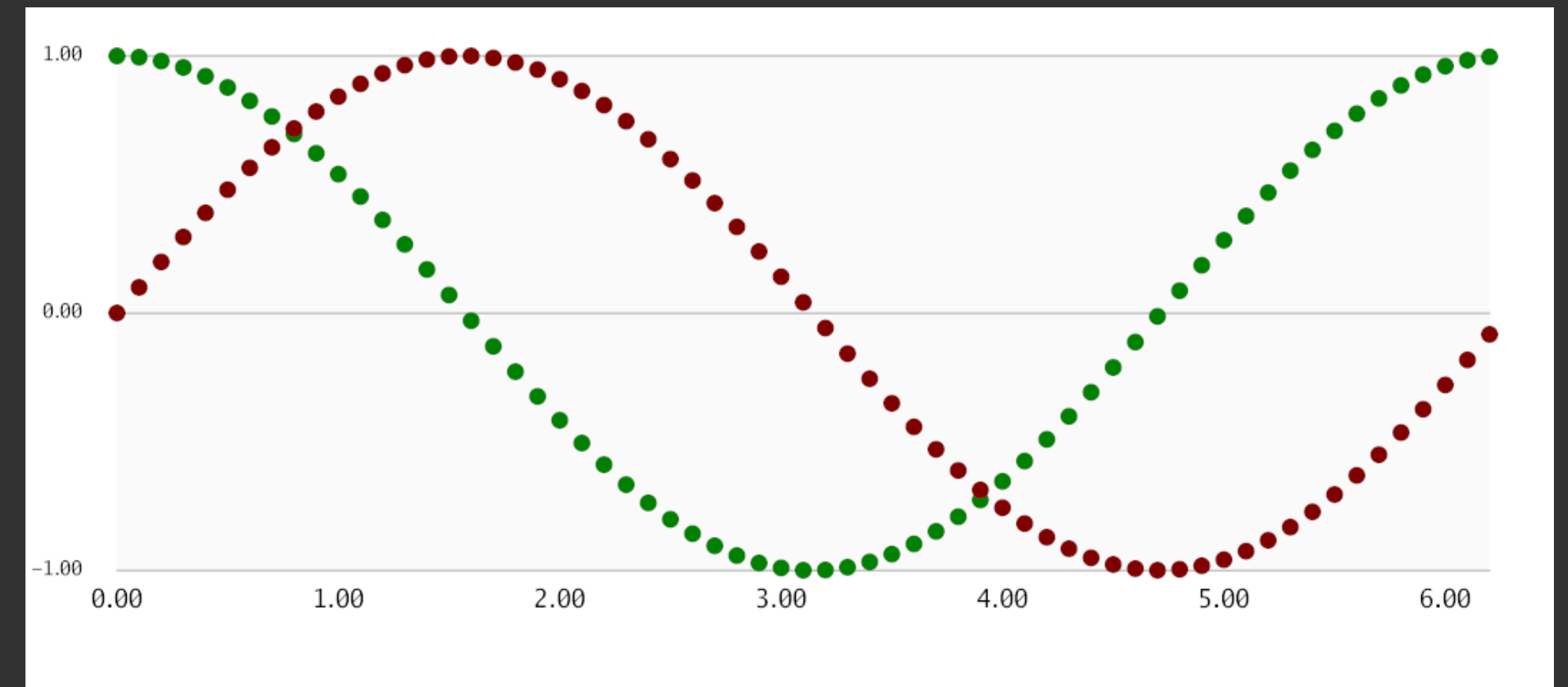
Read data int ChartBox	<code>func DataRead(r io.Reader) (ChartBox, error)</code>
Bar Chart	<code>Bar (canvas *gc.Canvas, size float64)</code>
Horizontal Bar Chart	<code>HBar (canvas *gc.Canvas, size, linespacing, textsize float64)</code>
Line Chart	<code>Line (canvas *gc.Canvas, size float64)</code>
Area Chart	<code>Area (canvas *gc.Canvas, opacity float64)</code>
Scatter Chart	<code>Scatter (canvas *gc.Canvas, size float64)</code>
Pie Chart	<code>Pie (canvas *gc.Canvas, size float64)</code>
Centered Title	<code>CTitle (canvas *gc.Canvas, size, offset float64)</code>
Chart Frame	<code>Frame (canvas *gc.Canvas, op float64)</code>
X Axis Label	<code>Label (canvas *gc.Canvas, size float64, n int)</code>
Y axis	<code>YAxis (canvas *gc.Canvas, size, min, max, step float64, fmt string, grid bool)</code>

giocanvas/chart: read data

sr, err := os.Open("sin.d")	# y=sin(x)	# y=cos(x)
if err != nil {	0.00 0.0000	0.00 1.0000
return err	0.10 0.0998	0.10 0.9950
}	0.20 0.1987	0.20 0.9801
cr, err := os.Open("cos.d")	0.30 0.2955	0.30 0.9553
if err != nil {	0.40 0.3894	0.40 0.9211
return err	0.50 0.4794	0.50 0.8776
}	0.60 0.5646	0.60 0.8253
sine, err := chart.DataRead(sr)	0.70 0.6442	0.70 0.7648
if err != nil {	0.80 0.7174	0.80 0.6967
return err	0.90 0.7833	0.90 0.6216
}	1.00 0.8415	1.00 0.5403
cosine, err := chart.DataRead(cr)
if err != nil {	6.00 -0.2794	6.00 0.9602
return err	6.10 -0.1822	6.10 0.9833
}	6.20 -0.0831	6.20 0.9965

giocanvas/chart: composite charts

```
cosine.Zerobased = false  
sine.Zerobased = false  
cosine.Frame(canvas, 5)  
sine.Label(canvas, 1.5, 10)  
cosine.YAxis(canvas, 1.2, -1.0, 1.0, 1.0, "%0.2f", true)  
cosine.Color = color.RGBA{0, 128, 0, 255}  
sine.Color = color.RGBA{128, 0, 0, 255}  
cosine.Scatter(canvas, 0.5)  
sine.Scatter(canvas, 0.5)
```



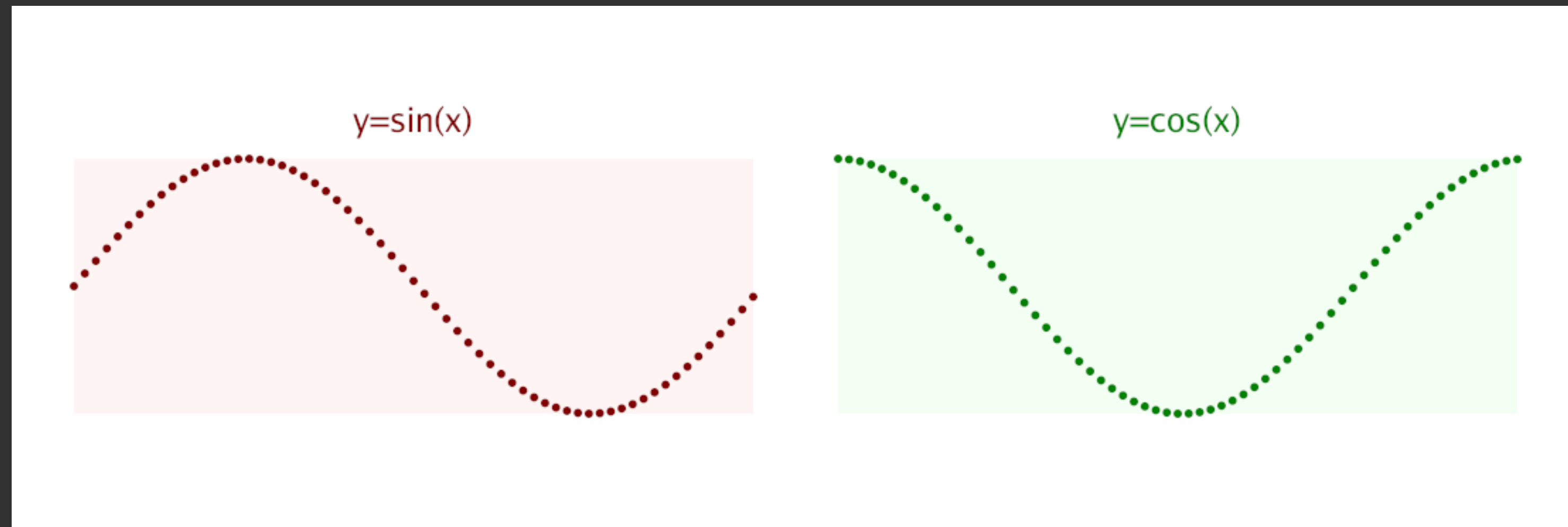
giocanvas/chart: side by side

```
sine.Left = 10  
sine.Right = sine.Left + 40  
sine.Top, cosine.Top = 30, 30  
sine.Bottom, cosine.Bottom = 10, 10
```

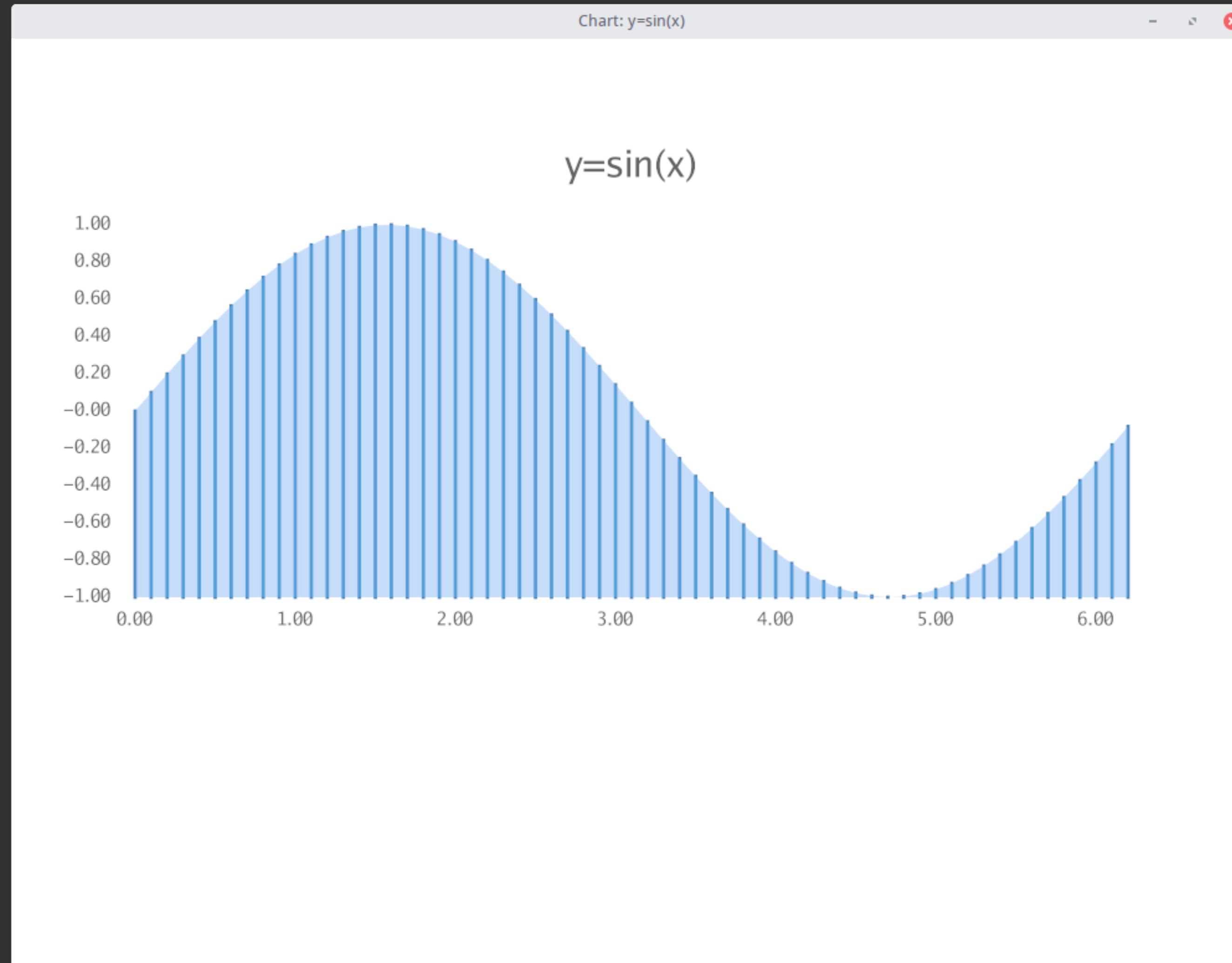
```
sine.CTitle(canvas, 2, 2)  
sine.Frame(canvas, 10)  
sine.Scatter(canvas, 0.25)
```

```
offset := 45.0  
cosine.Left = sine.Left + offset  
cosine.Right = sine.Right + offset
```

```
cosine.CTitle(canvas, 2, 2)  
cosine.Frame(canvas, 10)  
cosine.Scatter(canvas, 0.25)
```

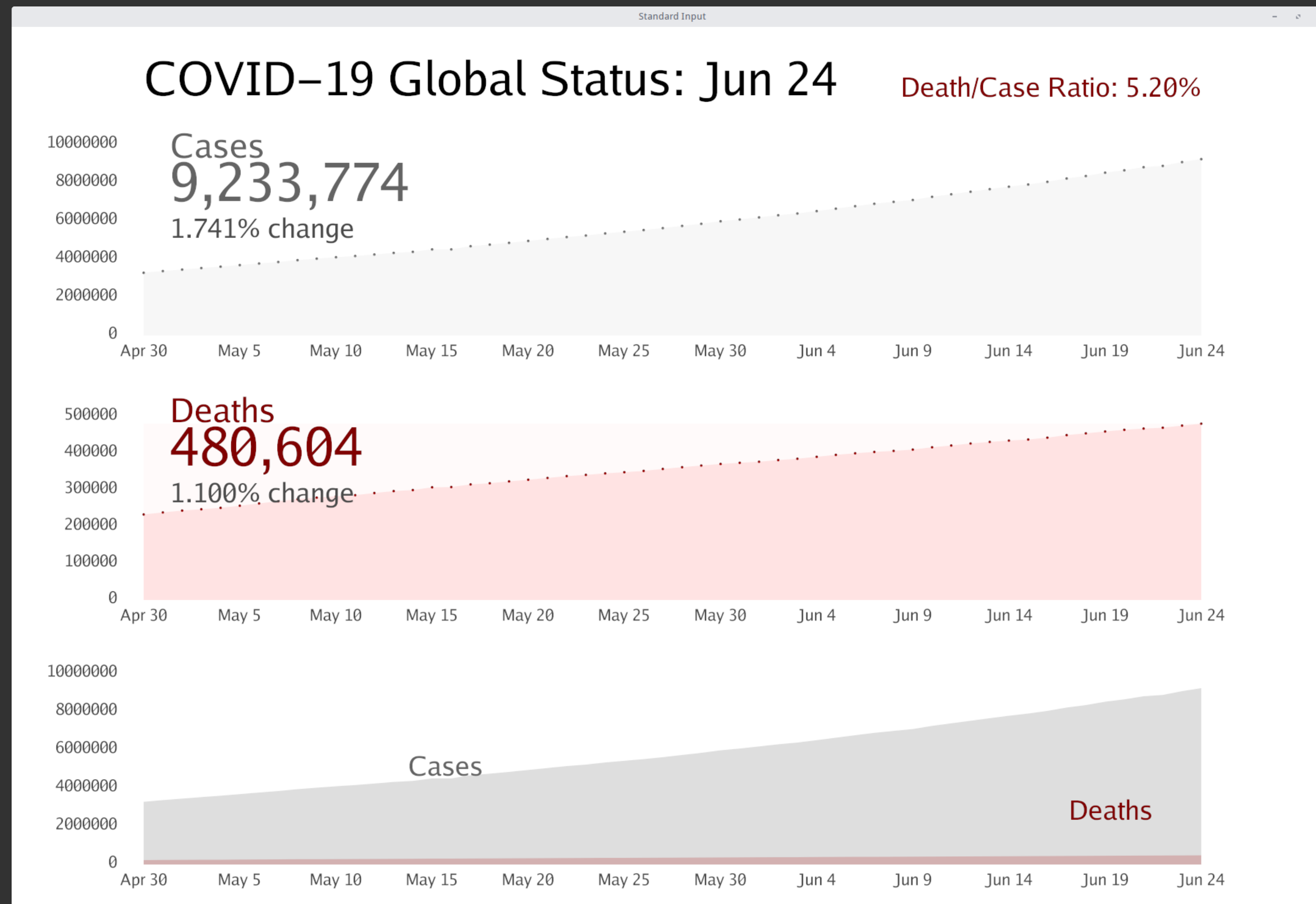


gchart



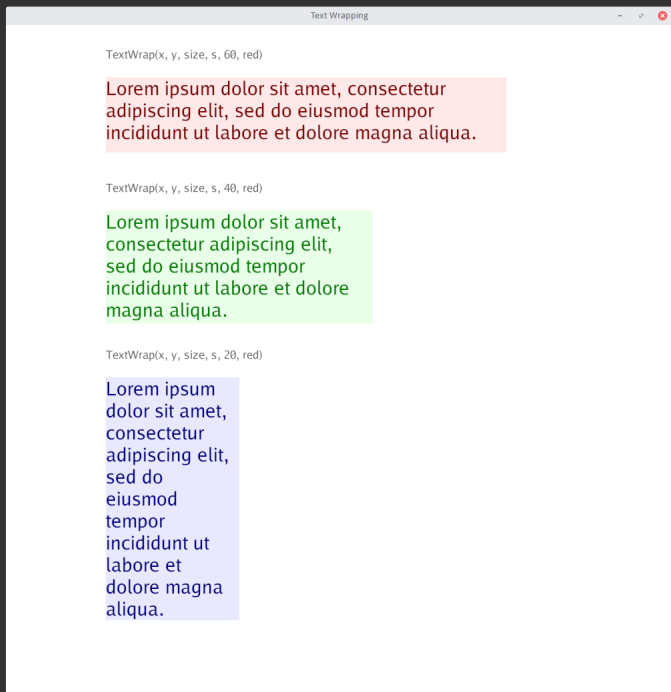
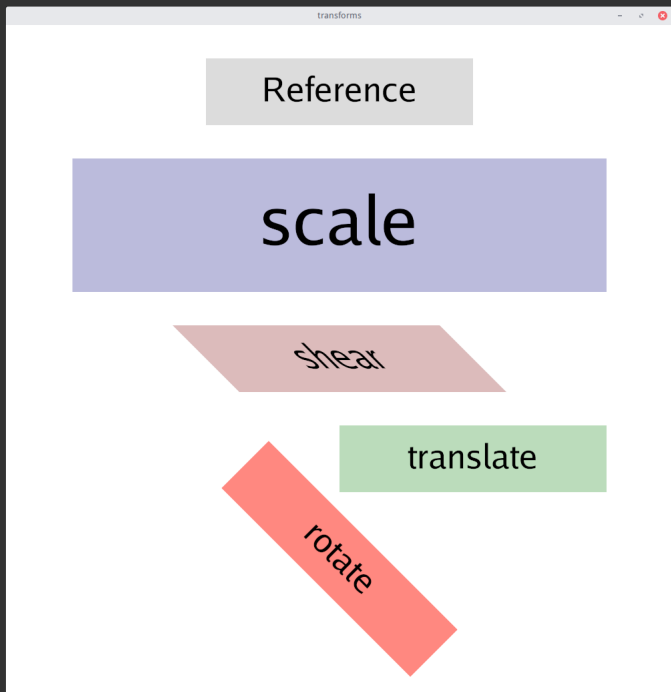
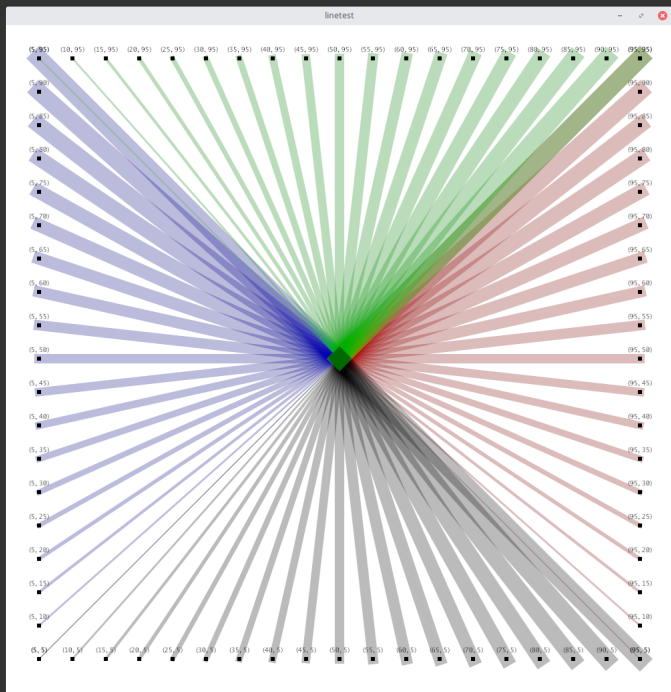
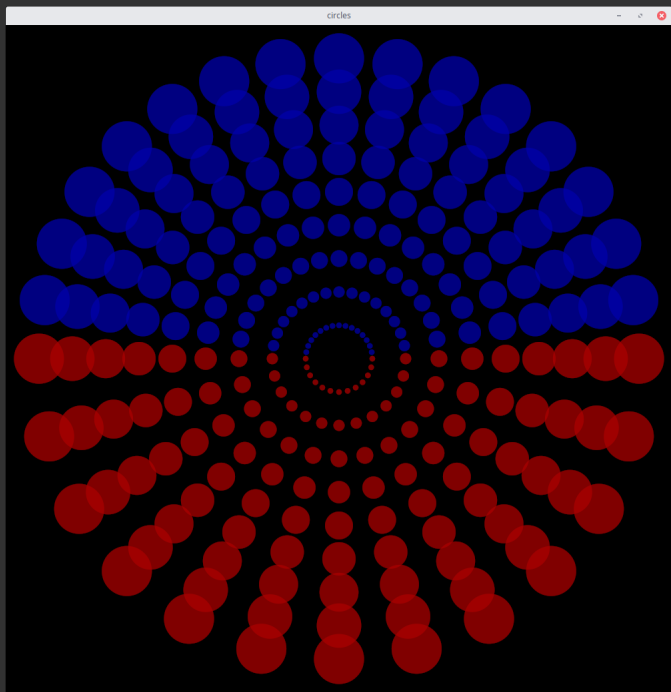
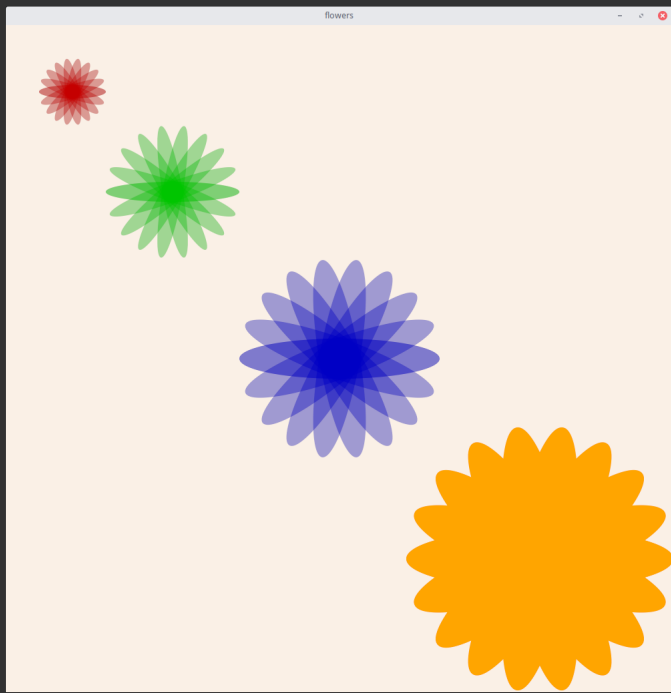
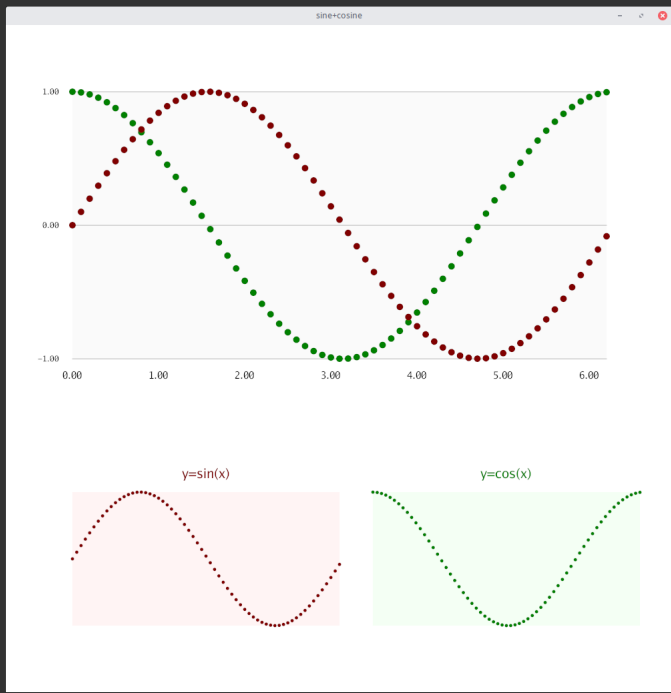
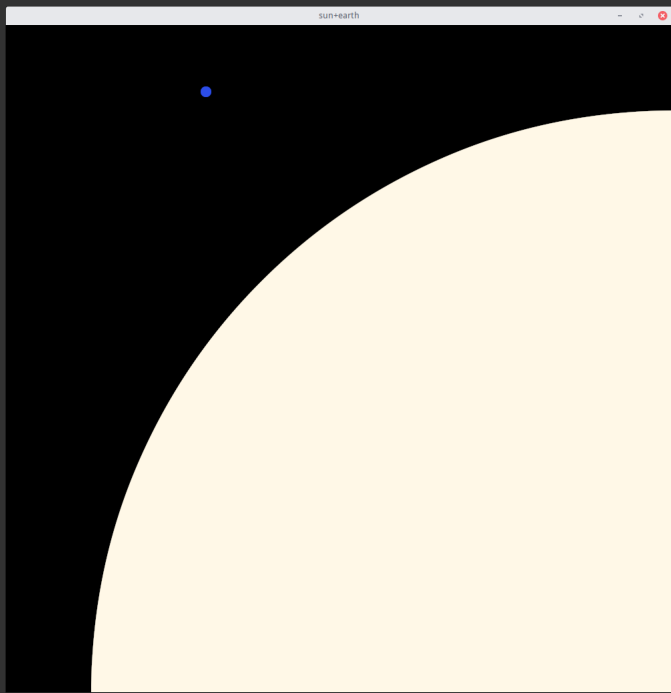
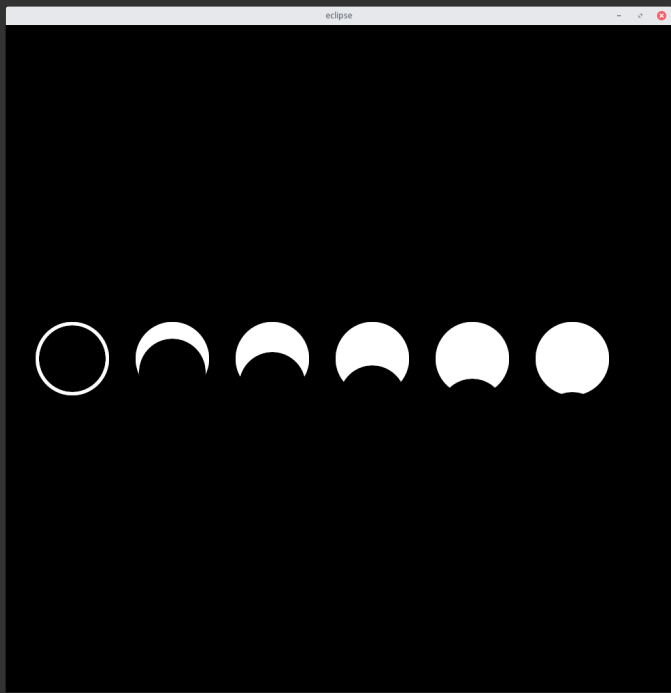
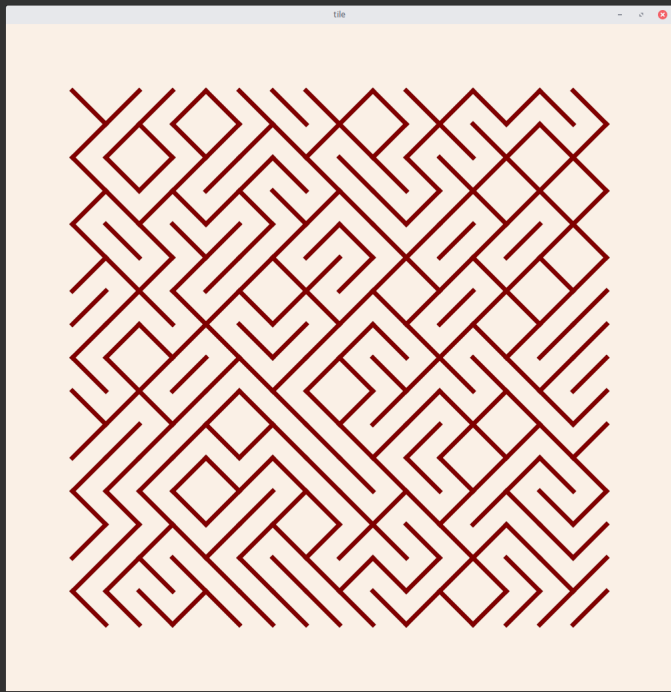
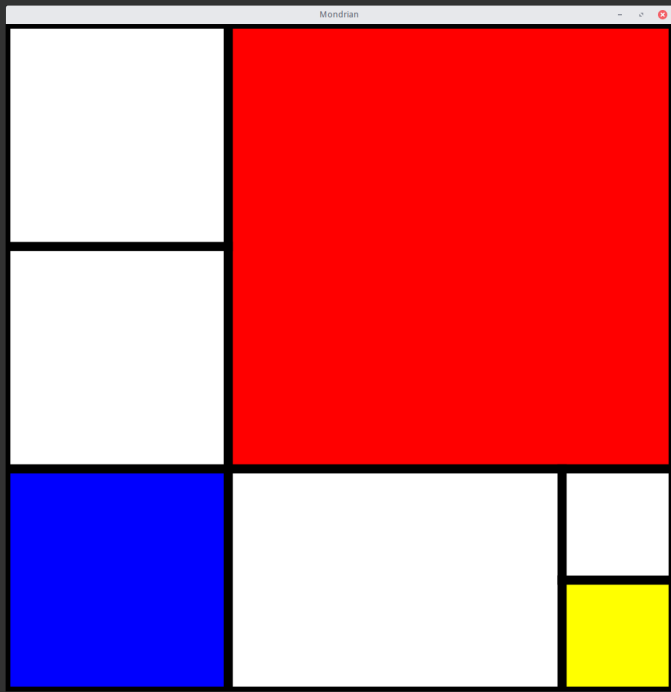
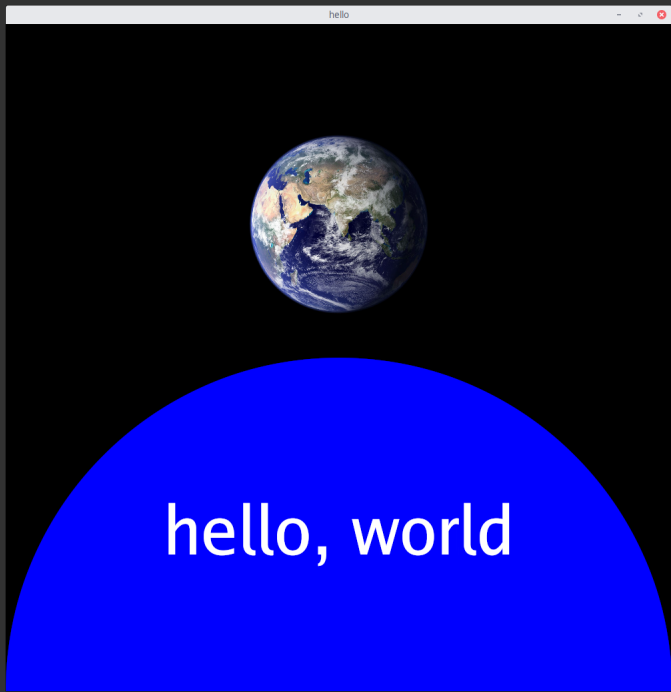
```
gchart -area -bar -barwidth=0.2 -zero=f -xlabel=10 -yrange=-1,1,0.2 -yfmt=%.2f -h 750 sin.d
```

gcdeck: deck viewer



```
c19chart -cyr=0,1e7,2e6 -dyr=0,5e5,1e5 | gcdeck -pagesize 1800,1200
```

Clients



go get it

github.com/ajstarks/giocanvas