Go Graphics

`gg` is a library for rendering 2D graphics in pure Go.

Stars

Installation

```
go get -u github.com/fogleman/gg
```

Alternatively, you may use gopkg.in to grab a specific major-version:

```
go get -u gopkg.in/fogleman/gg.v1
```

Documentation

- godoc: **https://godoc.org/github.com/fogleman/gg**
- pkg.go.dev: **https://pkg.go.dev/github.com/fogleman/gg?tab=doc**

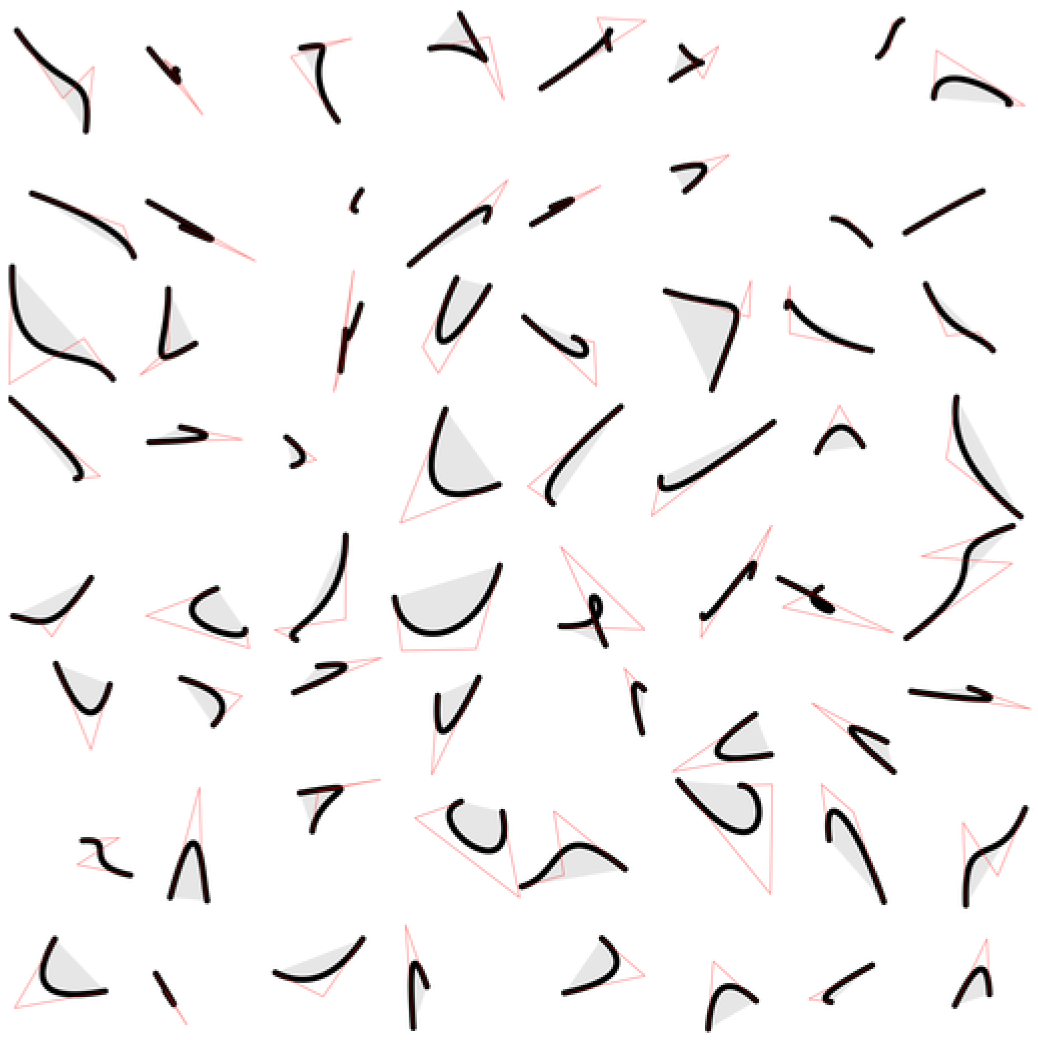
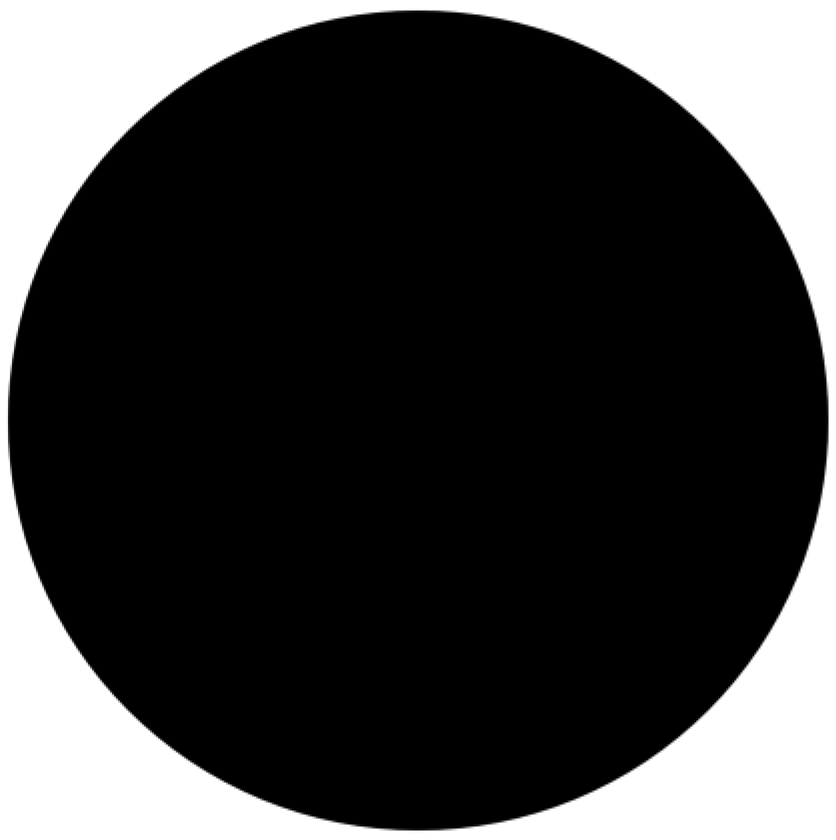
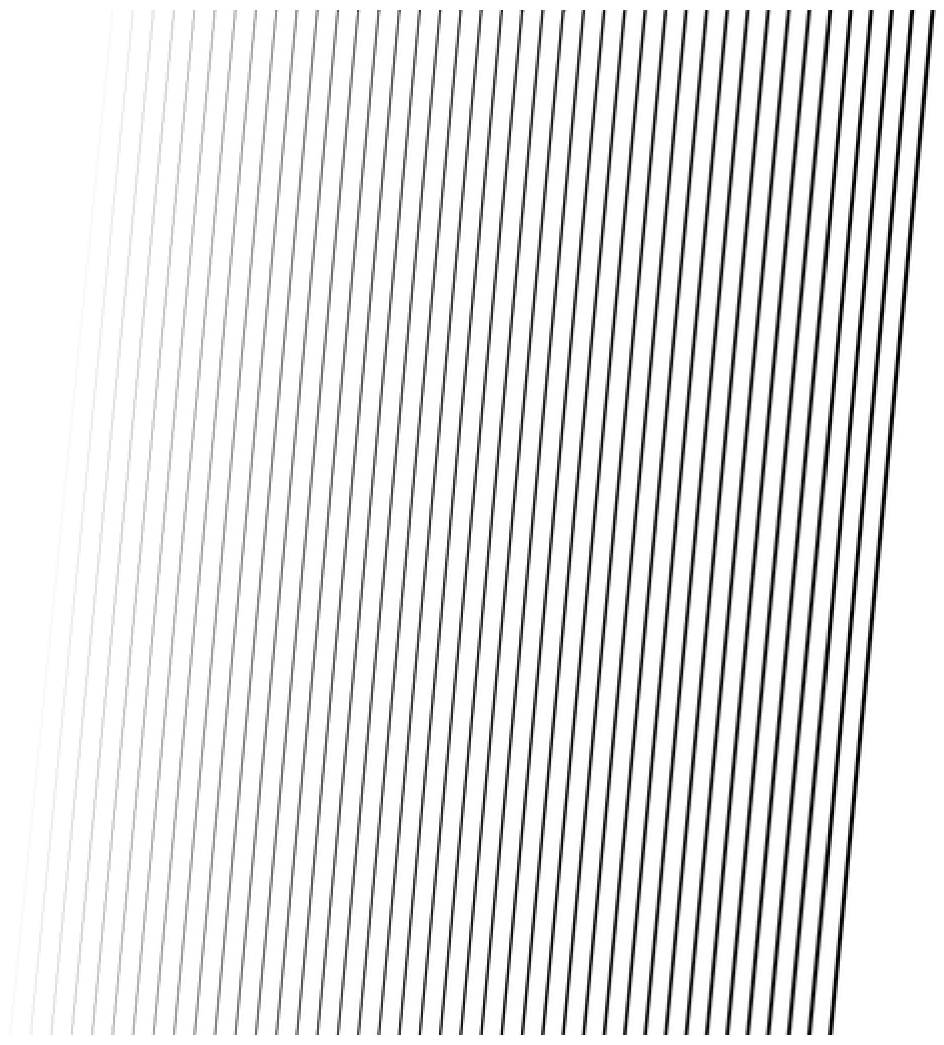
Hello, Circle!

Look how easy!

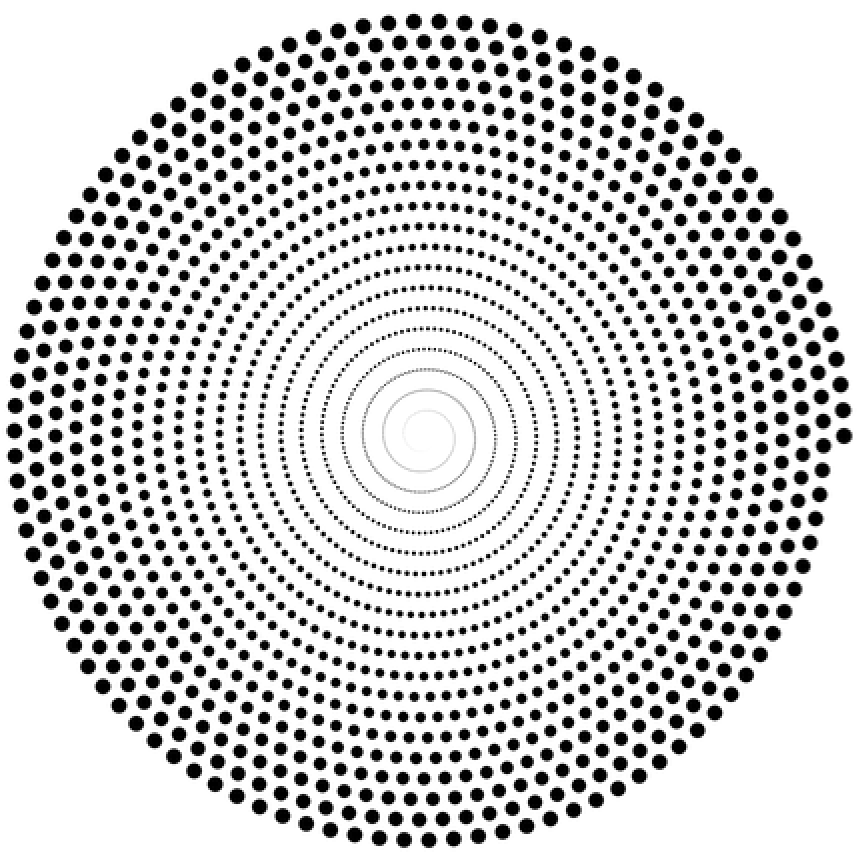
```go
package main

import "github.com/fogleman/gg"

func main() {
    dc := gg.NewContext(1000, 1000)
    dc.DrawCircle(500, 500, 400)
    dc.SetRGB(0, 0, 0)
    dc.Fill()
    dc.SavePNG("out.png")
}
```

Examples

There are **lots of examples** included. They're mostly for testing the code, but they're good for learning, too.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat
non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.



Creating Contexts

There are a few ways of creating a context.

```
NewContext(width, height int) *Context
NewContextForImage(im image.Image) *Context
NewContextForRGBA(im *image.RGBA) *Context
```

Drawing Functions

Ever used a graphics library that didn't have functions for drawing rectangles or circles? What a pain!

```
DrawPoint(x, y, r float64)
DrawLine(x1, y1, x2, y2 float64)
DrawRectangle(x, y, w, h float64)
DrawRoundedRectangle(x, y, w, h, r float64)
DrawCircle(x, y, r float64)
DrawArc(x, y, r, angle1, angle2 float64)
DrawEllipse(x, y, rx, ry float64)
DrawEllipticalArc(x, y, rx, ry, angle1, angle2 float64)
DrawRegularPolygon(n int, x, y, r, rotation float64)
DrawImage(im image.Image, x, y int)
DrawImageAnchored(im image.Image, x, y int, ax, ay float64)
SetPixel(x, y int)

MoveTo(x, y float64)
LineTo(x, y float64)
QuadraticTo(x1, y1, x2, y2 float64)
CubicTo(x1, y1, x2, y2, x3, y3 float64)
ClosePath()
ClearPath()
NewSubPath()

Clear()
Stroke()
Fill()
StrokePreserve()
FillPreserve()
```

It is often desired to center an image at a point. Use `DrawImageAnchored` with `ax` and `ay` set to 0.5 to do this. Use 0 to left or top align. Use 1 to right or bottom align. `DrawStringAnchored` does the same for text, so you don't need to call `MeasureString` yourself.

Text Functions

It will even do word wrap for you!

```
DrawString(s string, x, y float64)
DrawStringAnchored(s string, x, y, ax, ay float64)
DrawStringWrapped(s string, x, y, ax, ay, width, lineSpacing float64, align Align)
MeasureString(s string) (w, h float64)
MeasureMultilineString(s string, lineSpacing float64) (w, h float64)
WordWrap(s string, w float64) []string
SetFontFace(fontFace font.Face)
LoadFontFace(path string, points float64) error
```

Color Functions

Colors can be set in several different ways for your convenience.

```
SetRGB(r, g, b float64)
SetRGBA(r, g, b, a float64)
SetRGB255(r, g, b int)
SetRGBA255(r, g, b, a int)
SetColor(c color.Color)
SetHexColor(x string)
```

Stroke & Fill Options

SetLineWidth(lineWidth float64)
SetLineCap(lineCap LineCap)
SetLineJoin(lineJoin LineJoin)
SetDash(dashes ...float64)
SetDashOffset(offset float64)
SetFillRule(fillRule FillRule)

Gradients & Patterns

`gg` supports linear, radial and conic gradients and surface patterns. You can also implement your own patterns.

SetFillStyle(pattern Pattern)
SetStrokeStyle(pattern Pattern)
NewSolidPattern(color color.Color)
NewLinearGradient(x0, y0, x1, y1 float64)
NewRadialGradient(x0, y0, r0, x1, y1, r1 float64)
NewConicGradient(cx, cy, deg float64)
NewSurfacePattern(im image.Image, op RepeatOp)

Transformation Functions

Identity()
Translate(x, y float64)
Scale(x, y float64)
Rotate(angle float64)
Shear(x, y float64)
ScaleAbout(sx, sy, x, y float64)
RotateAbout(angle, x, y float64)
ShearAbout(sx, sy, x, y float64)
TransformPoint(x, y float64) (tx, ty float64)
InvertY()

It is often desired to rotate or scale about a point that is not the origin. The functions `RotateAbout`, `ScaleAbout`, `ShearAbout` are provided as a convenience.

`InvertY` is provided in case Y should increase from bottom to top vs. the default top to bottom.

Stack Functions

Save and restore the state of the context. These can be nested.

Push()
Pop()

Clipping Functions

Use clipping regions to restrict drawing operations to an area that you defined using paths.

Clip()
ClipPreserve()
ResetClip()
AsMask() *image.Alpha
SetMask(mask *image.Alpha)
InvertMask()

Helper Functions

Sometimes you just don't want to write these yourself.

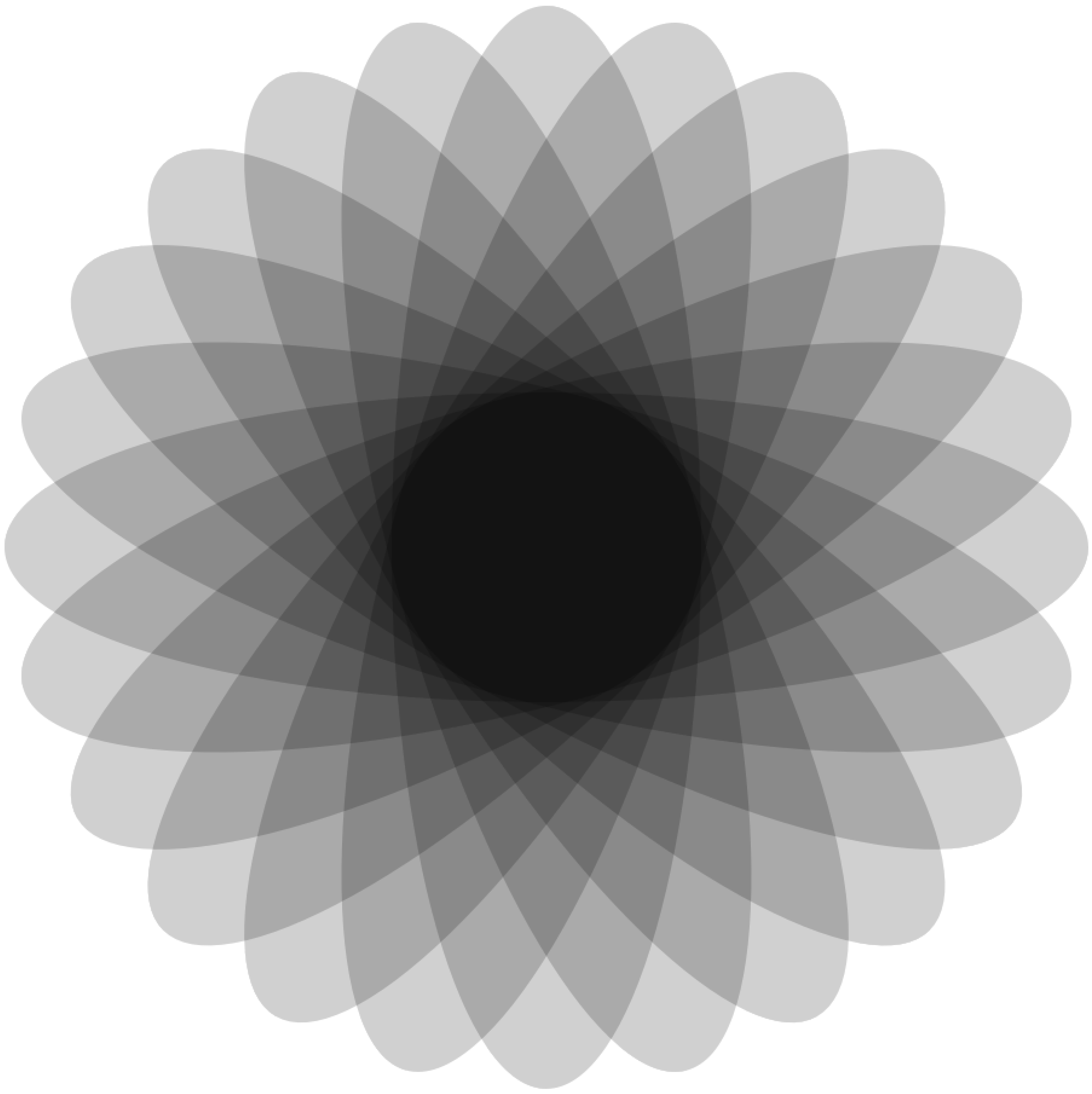
Radians(degrees float64) float64
Degrees(radians float64) float64
LoadImage(path string) (image.Image, error)
LoadPNG(path string) (image.Image, error)
SavePNG(path string, im image.Image) error

Separator

Another Example

See the output of this example below.

```
package main

import "github.com/fogleman/gg"

func main() {
 const S = 1024
 dc := gg.NewContext(S, S)
 dc.SetRGBA(0, 0, 0, 0.1)
 for i := 0; i < 360; i += 15 {
  dc.Push()
  dc.RotateAbout(gg.Radians(float64(i)), S/2, S/2)
  dc.DrawEllipse(S/2, S/2, S*7/16, S/8)
  dc.Fill()
  dc.Pop()
 }
 dc.SavePNG("out.png")
}
```



Ellipses