Digital Illumination

Dr Alun Moon

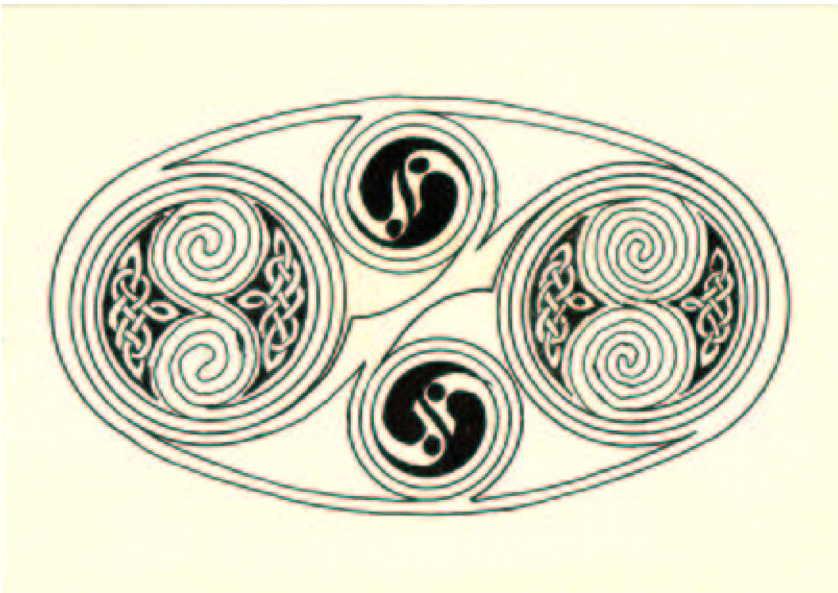
July 19, 2003

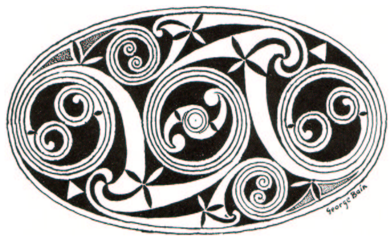
Celtic artwork

- From about the 7 century BC through to the 7 century AD
- Metalwork
- Jewelry
- stone carving
- Illuminated manuscripts
  - Lindesfarne Gospels
  - Book of Kells

Example by hand

Compare a scan of one of my pieces with a sketch from the Lindesfarne Gospels.





Main elements

- Knotwork
- Keypatterns
- Spirals

- a highly developed artistic style, with very fine intricate detail
- high degree of geometry and geometrical construction in their work

Knotwork

- one of the most recognisable elements of Celtic artwork.
- once paths are defined
  - find all intersection points
  - sort into order
  - draw curves
  - draw crossings, path goes over first crossing then alternates

Getting global intersection time

Problems with `intersectiontimes` operator

- points are found on successive subpaths starting just beyond last point.
- length of subpaths is always integer.
  - path $z_0..z_1..z_2$ has length 2
  - subpath $[.75, 1.25]$ has length 2
- how to get intersection-time on original path

Algorithm

For subpath starting at time t_s on original path. Intersection time on subpath is t_f

$$t = \begin{cases} t_f[t_s, \lceil t_s \rceil] & t_f < 1 \\ t_f + \lfloor t_s \rfloor \end{cases}$$

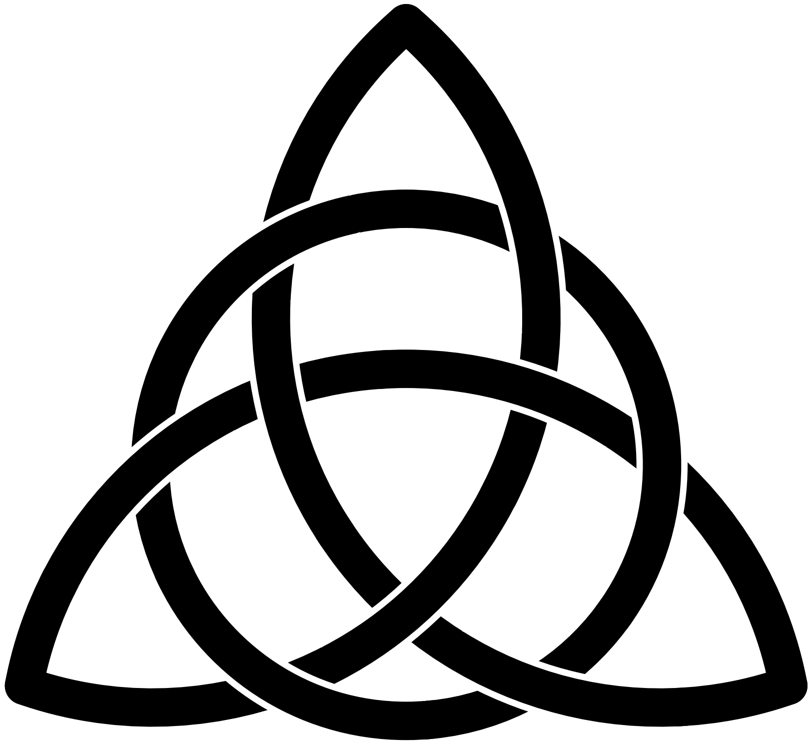
- if $t_s < 1$ use t_s to interpolate between the beginning of the subpath (a) and the next point on the curve (ceiling of a).
- if $t_s >= 1$ then add it to the last point on the curve before the subpath (floor a)

crossings

```
vardef crossings@#(text others) =
  save lastpt, tmp;
  p@#t[0]:=0;
  p@#t#:=0;
  forsuffixes $=others:
    numeric lastpt;
    lastpt := epsilon;
    forever:
      numeric tmp;
      (tmp,whatever)=
        subpath (lastpt,length(p@#)-epsilon)
        of p@#
      intersectiontimes p$;
      exitif (tmp<=0);
      p@#t[incr p@#t#] := if(tmp<1):
        tmp[lastpt,ceil(lastpt)]
      else:
        floor(lastpt)+tmp
      fi;
      lastpt := p@#t[p@#t#]+epsilon;
```

Trefoil – intersections

Trefoil



Some people claim it symbolises the Holy Trinity, or wholeness (I like it because it is the motif used for my wedding).

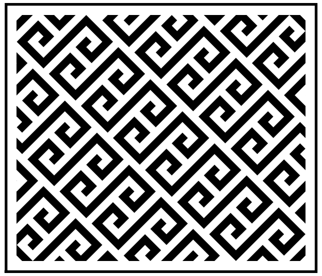
Keypatterns

Keypatterns are often based on a tessellating square spiral (straight lines and rectangles). George Bain uses a simple notation to characterise the spiral, a sequence of “arc” lengths.

S-spiral macro

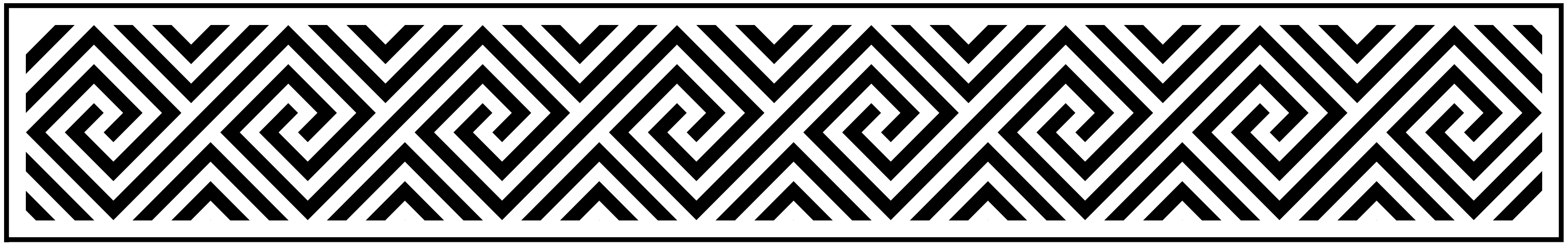
```
def keySspiral(text tail) :=
  begingroup
    save direct,lastpoint,maxlength;
    pair direct,lastpoint;
    direct := up rotated -90;
    lastpoint := origin;
    maxlength := 0;
    origin
    for p=tail: --
      begingroup
        direct := direct
          rotated if (maxlength<=p):
            begingroup maxlength := p;
            90 endgroup
          else:
            -90
          fi;
        lastpoint := lastpoint + direct*p;
        lastlength := p;
        lastpoint
```

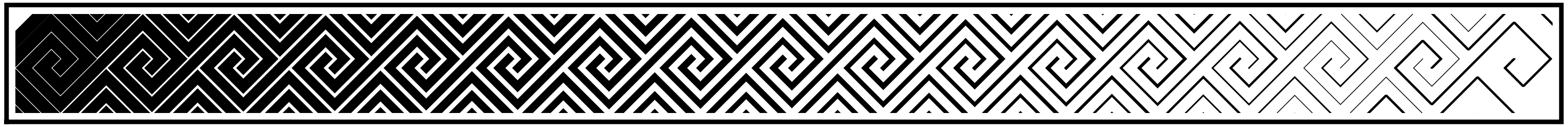
Keypattern – tessellating

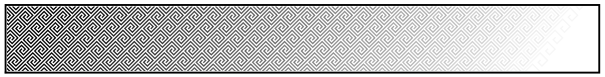
sequence of $(1,2,3,4,8,4,3,2,1)$ 

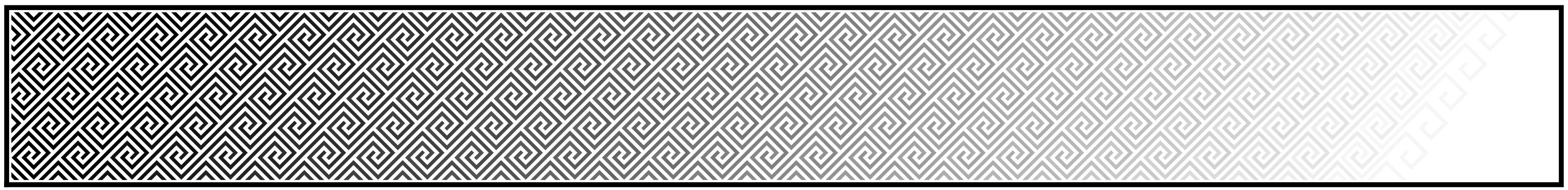
Interlocking

sequence $(1,2,3,4,9,4,3,2,1)$







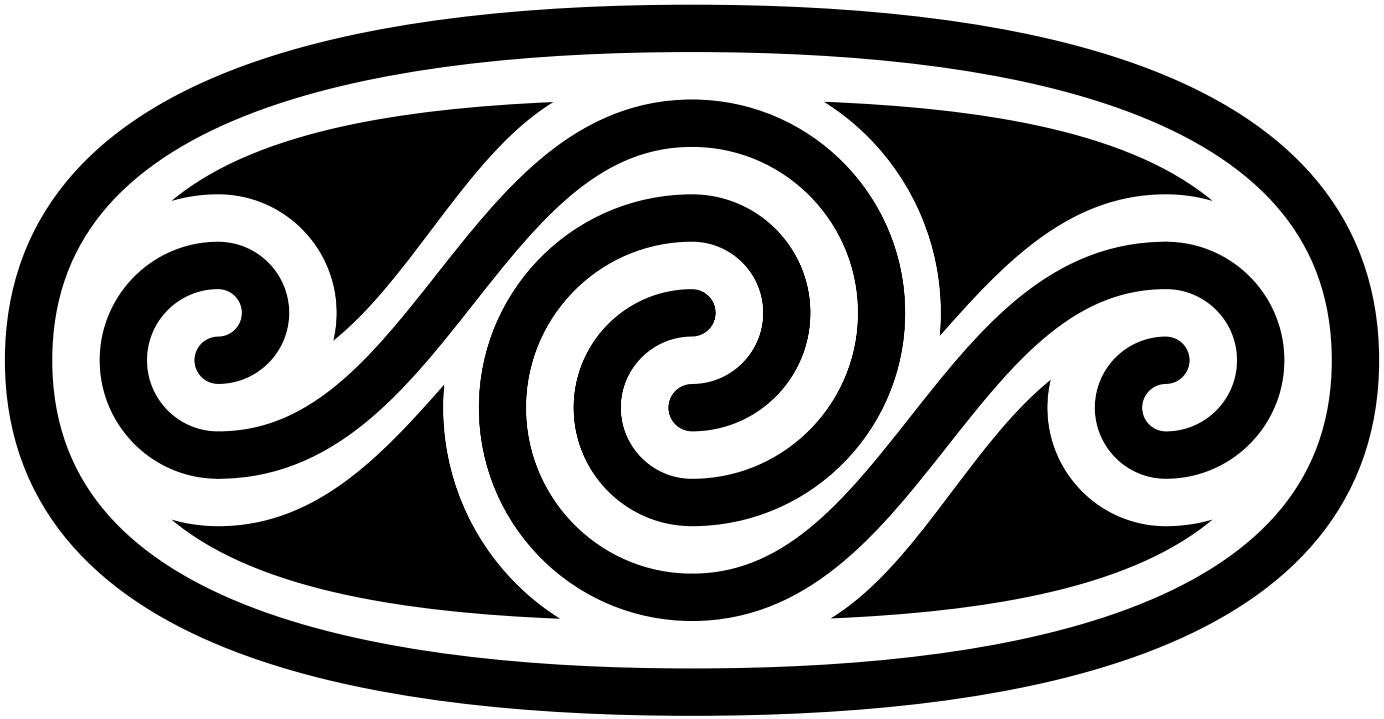


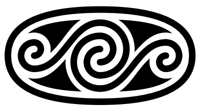
Spirals

Given an initial point, a pair of centres, and a number of turns, the `spiral` macro is very simple recursive function (figure **??**). Although it could be just as simple with a loop, swaping the centres over is easier to do with the recursive call.

```
def spiral(expr a,b,$)(expr turns) =
  $
   .. $ rotatedaround(a, 90)
   .. $ rotatedaround(a, 180)
  if( turns>1 ):
    & spiral(b,a,
             $ rotatedaround(a, 180))
             (turns-1)
  fi
enddef;
```

Spirals – cartouche



1in  – 1cm  – 1pt . !

Spiral developments

- links between spirals should start and end on tangents
  - Need a macro to find common tangent and tangent points to paths
- geometry rapidly becomes complex
- often need sets of parallel curves.