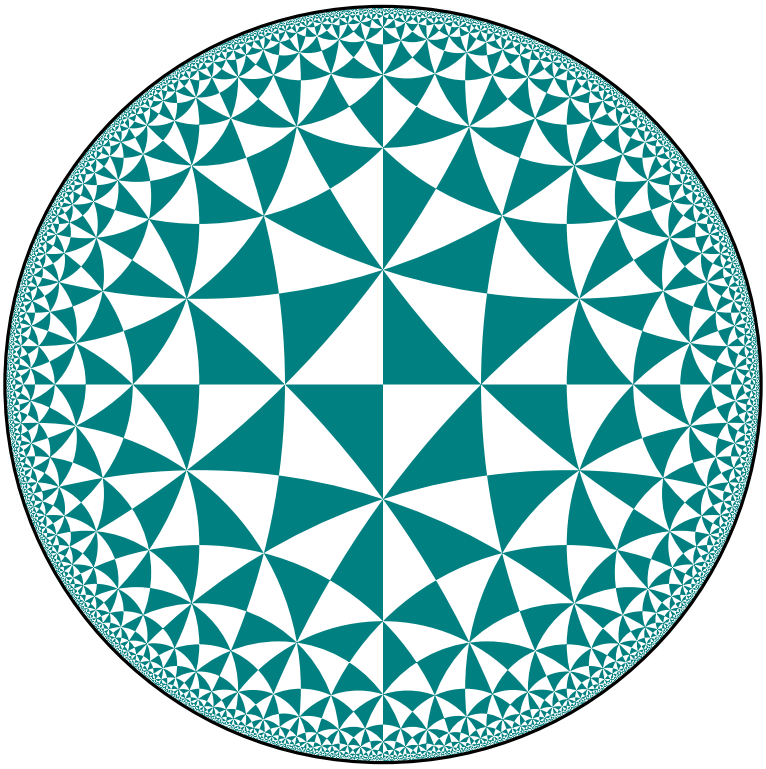
The **luahyperbolic** LaTeX package

Damien Mégy



`\hyperbolicTiling{2}{4}{5}[scale=5,color=teal,depth=24]`

This is version 2026/11/13. It may not be the same as the one available on CTAN.
Get the latest version of the `.sty` file, documentation and examples at: github.com/dmegy/luahyperbolic.

Contents

1 Introduction **2**
1.1 What you can do with this package . . . 2
1.2 What this package cannot do for you . . 2
1.3 Efficiency 2
1.4 How to load the package 3
1.5 Dependencies 3
1.6 Exporting the pictures 3
1.7 (Advanced users) Build from source . . . 3

2 Drawing and labeling elements **4**
2.1 Draw and label points 4
2.2 Draw/label/mark segments/lines/rays . 4
2.3 Circles, semicircles, arcs 8
2.4 Horocycles and hypercycles 10
2.5 Printing custom strings to TikZ 10
2.6 Modifying default styles 11

3 Defining points **11**
3.1 Random points with uniform density . . 11
3.2 Endpoints, midpoint, interpolate 11
3.3 Intersections 12
3.4 Reflections and projections 14
3.5 Exponential map 15
3.6 Automorphisms (direct isometries) . . . 15
3.7 Orbits under automorphisms 17
3.8 Triangle geometry (CAUTION) 18

4 Tilings **21**
4.1 Draw Tilings 21
4.2 Fill tilings 22
4.3 Tilings without any Lua code 23

5 Overview of the lua libraries **24**
5.1 complex 24
5.2 luahyperbolic 24

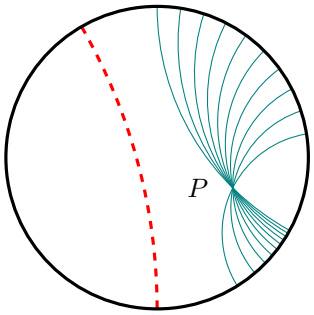
6 Changelog **25**

1 Introduction

1.1 What you can do with this package

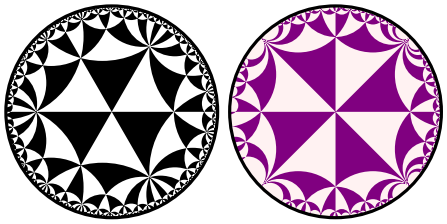
1. Draw hyperbolic geometry pictures, high precision (PDF output).
2. Use an easy syntax. At least, easier than with LaTeX macros and bloody PGF.
3. Export generated pictures to external tikz files if you want.
4. Read the source code and learn hyperbolic geometry.
5. Compile **without** installing python packages or lua packages etc.
6. Compile **without** "shell-escape" mode.
7. Compile everything in a single pass. No (Leeloo Dallas) multipass. Want to illustrate Euclid's fifth postulate? Do it right there in your TeX file without having to include external files:

```
\begin{luacode*}
hyper.tikzBegin("scale=2")
local P = complex(0.5,-0.2)
local A = complex.exp_i(math.pi/20)
for k=1,10 do
  hyper.drawLine(P, A^k, "teal")
end
hyper.labelPoint(P, "$P$", "left=.2cm")
local style = "very thick, dashed, red"
hyper.drawLine(complex.J,-complex.I,style)
hyper.tikzEnd()
\end{luacode*}
```



8. Draw Tilings in one line, without any Lua code:

```
\hyperbolicTiling{3}{4}{7}[scale=1.4]
\hyperbolicTiling{4}{4}{inf}[scale=1.4,
    ↪ color=violet, backgroundcolor=pink
    ↪ !20, depth=6]
```



1.2 What this package cannot do for you

1. Really intensive computations, millions of geodesics etc. Source code is not optimized to keep maximum readability. All computations involve automorphisms. See other remarks below.
2. Coffee.

1.3 Efficiency

For heavy computations, please use a ~~Python~~ C library. Nevertheless, this library is still reasonably efficient and usable:

- The first tiling illustration of this documentation (hyperbolic 2-4-5 tiling with 2361 geodesics) just compiled in 3.35 seconds on a 2020 Intel MacBook Air laptop. (Other users report 1.48 second on a MacBook Air M1.)
- This entire documentation including all pictures compiles in approx. 20 seconds on the same 2020 Intel MacBook Air, the majority of the time spent on the tilings. The package is suitable for producing lecture notes in hyperbolic geometry, which was the reason for all of this. For an entire book with dozens of pictures, you may need to externalize some of the pictures.

- Tests show that the bottleneck is Ti*k*Z, not Lua. In a future version, some future maintainer may add other frontends to the library, including drawing directly to the PDF with PDF primitives which are several orders of magnitude faster than Ti*k*Z.

1.4 How to load the package

Local install

Simply put the `luahyperbolic.sty` file in your working directory and add the following line to your .tex preamble:

```
\usepackage{luahyperbolic}
```

The single file `luahyperbolic.sty` contains everything needed, including the necessary library for complex numbers. It can be sent to colleagues by email, etc.

It will declare two Lua globals: `complex` and `hyper`, and define a `hyperbolicTiling` command for convenience.

If the .sty file is not in the same directory as your .tex file, you can write `\usepackage{../luahyperbolic}`.

1.5 Dependencies

The package luahyperbolic requires the following packages:

- tikz
- luacode

It loads them if they are not loaded already.

The LaTeX files must be compiled with the **lualatex** engine.

1.6 Exporting the pictures

1.6.1 When finishing a picture

Simply add a filename to the closing `hyper.tikzEnd()` instruction, as below:

```
hyper.tikzEnd("myfile.tikz")
```

This will save the current picture (including `\begin{tikzpicture}[options]` and `\end{tikzpicture}`).

1.6.2 In the middle of the tikzpicture

It is also possible to export the current picture at any point with `hyper.tikzExport(optional-filename)`. If no filename is provided, the package will name it `hyper_picture_N` with `N` the picture counter. Luahyperbolic keeps count of how many pictures have been produced in the document. This allows exporting images in loops, or drawing several steps of the same picture.

1.7 (Advanced users) Build from source

The `luahyperbolic.sty` is pre-generated from a template .sty file and four .lua modules :

1. `complex.lua`
2. `luahyperbolic-core.lua`
3. `luahyperbolic-tikz.lua`
4. `luahyperbolic-tilings.lua`

These files are in the `lua/` subdirectory of the package repo.

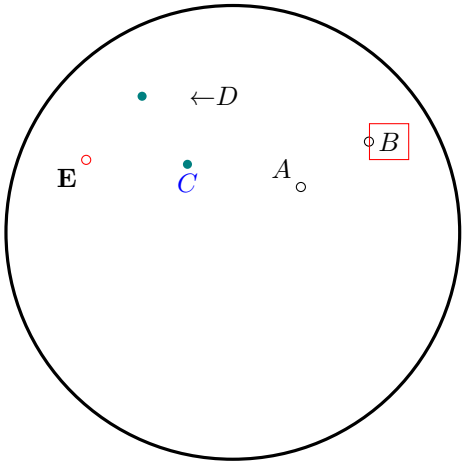
To build the `luahyperbolic.sty` from .lua source files, execute `lua build.lua` in the `lua` subdirectory of the package. This will build and update (replace) the existing .sty file.

2 Drawing and labeling elements

2.1 Draw and label points

2.1.1 Draw and label points

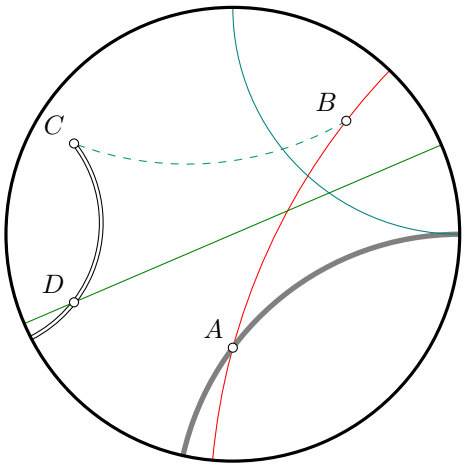
```
\begin{luacode*}
local A = complex(0.3,0.2)
local B = 2*A
local I, J = complex.I, complex.J
local C, D, E = I*A, I*B, J*B
hyper.tikzBegin()
hyper.drawPoints(A, B)
hyper.drawPoints(C, D, "teal")
hyper.drawPoint(E, "white, draw=red")
hyper.labelPoint(A,"$A$")
-- (default options = "above left")
hyper.labelPoints(B,"$B$", "right, draw=red")
hyper.labelPoints(C,"$C$", "below, blue")
hyper.labelPoint(D,"←$D$","right=.5cm")
hyper.labelPoint(E,"\\textbf{E}","below left")
hyper.tikzEnd()
\end{luacode*}
```



2.2 Draw/label/mark segments/lines/rays

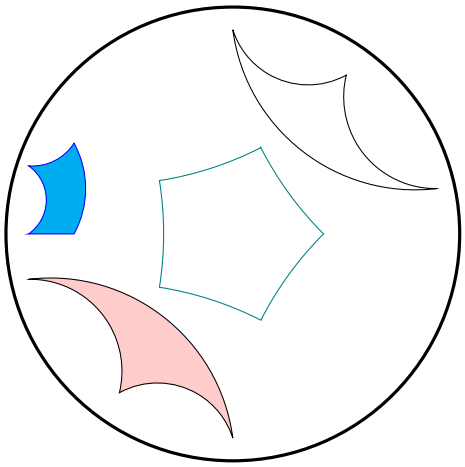
2.2.1 Segments lines and rays (through two points)

```
\begin{luacode*}
local A = -complex.I/2
local B = (1+complex.I)/2
local C = complex(-0.7,0.4)
local D = complex(-0.7,-0.3)
hyper.tikzBegin()
hyper.drawLine(A, B, "red")
hyper.drawSegment(B,C,"dashed, ForestGreen")
hyper.drawRay(C,D,"double,double distance=1pt"
    ↪ )
hyper.drawLine(D,-D, "green!50!black")
hyper.drawLine(A, 1, "gray, line width=2pt")
hyper.drawLine(1, complex.I, "teal")
hyper.drawPoints(A, B, C, D)
hyper.labelPoints(A, "$A$", B, "$B$", C, "$C$"
    ↪ , D, "$D$")
hyper.tikzEnd()
\end{luacode*}
```



2.2.2 Triangles and polygons

```
\begin{luacode*}
local A = complex(.9,.2)
local B = complex(.5,.7)
local C = complex(0,.9)
hyper.tikzBegin()
hyper.drawTriangle(A, B, C)
hyper.drawTriangle(-A, -B, -C, "fill=red!20")
hyper.drawPolygon(-.9, -.7, complex(-.7,.4),
    ↪ complex(-.9,.3), "blue, fill=cyan")
local points={}
for k=1,5 do
  table.insert(points, complex.polar(.4,k*2*
      ↪ math.pi/5))
end
hyper.drawPolygonFromTable(points,"teal")
hyper.tikzEnd()
\end{luacode*}
```



2.2.3 Triangles and ideal triangles with prescribed angles

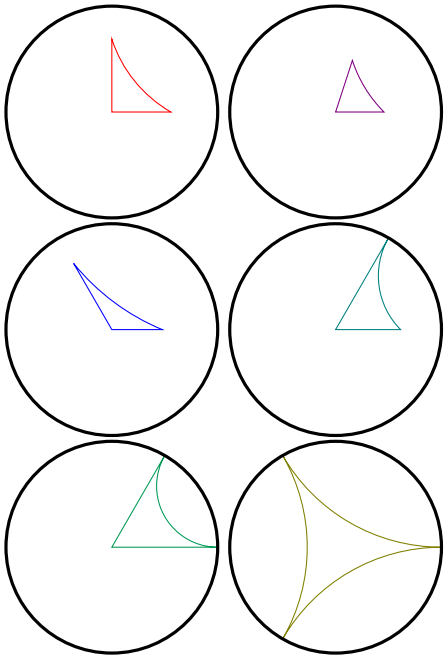
```
\begin{luacode*}
local function myTriangle(a, b, c, color)
  hyper.tikzBegin("scale=1.4")
  local A, B, C = hyper.triangleWithAngles(a,
      ↪ b, c)
  hyper.drawTriangle(A, B, C, color)
  hyper.tikzEnd()
end

local PI = math.pi

myTriangle(PI/2, PI/6, PI/10, "red")
myTriangle(2*PI/5, PI/4, PI/5, "red!50!blue")
tex.print("\\par")

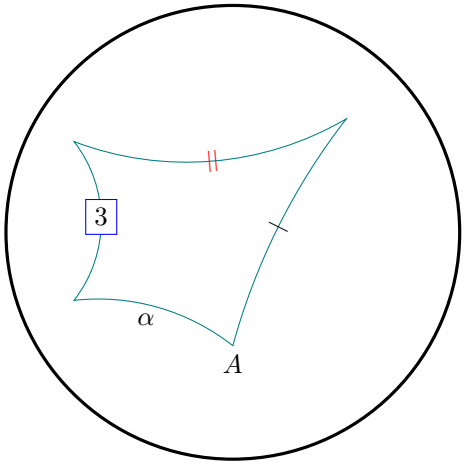
myTriangle(2*PI/3, PI/8, PI/20, "blue")
myTriangle(PI/3, PI/4, 0, "blue!50!green")
tex.print("\\par")

myTriangle(PI/3, 0, 0, "ForestGreen")
myTriangle(0, 0, 0, "green!50!red")
\end{luacode*}
```



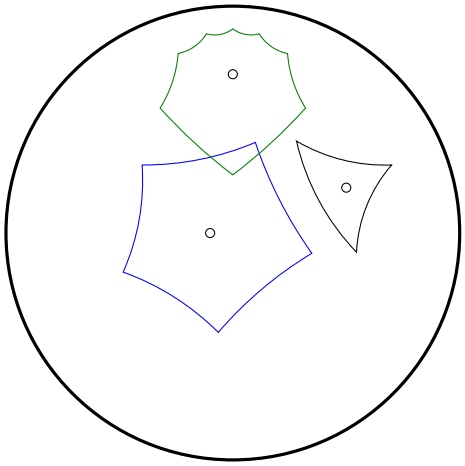
2.2.4 Label and mark segments

```
\begin{luacode*}
local A = -complex.I/2
local B = (1+complex.I)/2
local C = complex(-0.7,0.4)
local D = complex(-0.7,-0.3)
hyper.tikzBegin()
hyper.drawPolygon(A, B, C, D, "teal")
hyper.markSegment(A,B,"|")
local marking = "\\textcolor{red}{||}"
hyper.markSegment(B,C,marking)
hyper.labelSegment(C,D,"$3$", "fill=white,
    ↪ draw=blue")
hyper.labelSegment(D, A,"$\\alpha$", "below")
hyper.labelPoint(A, "$A$", "below")
hyper.tikzEnd()
\end{luacode*}
```



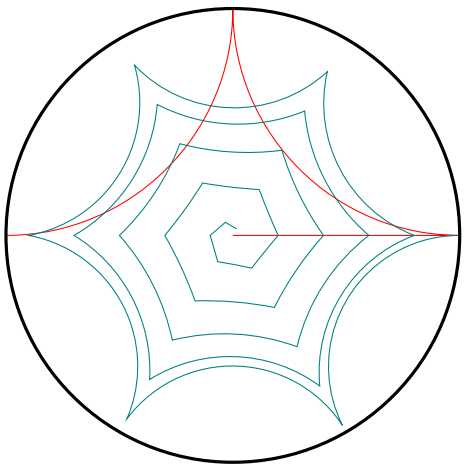
2.2.5 Regular polygons

```
\begin{luacode*}
local O1 = complex(.5,.2)
local A1 = complex(.7,.3)
local O2 = complex(-0.1,0)
local A2 = complex(-.4,.3)
local O3 = complex(0,.7)
local A3 = complex(0,.9)
hyper.tikzBegin()
hyper.drawRegularPolygon(O1, A1, 3)
hyper.drawRegularPolygon(O2, A2, 5, "blue")
hyper.drawRegularPolygon(O3, A3, 8, "green!50!
    ↪ black")
hyper.drawPoints(O1,O2,O3)
hyper.tikzEnd()
\end{luacode*}
```



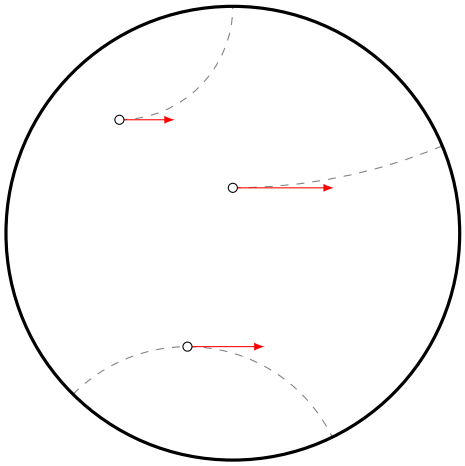
2.2.6 Polygonal lines

```
\begin{luacode*}
hyper.tikzBegin()
-- drawPolyline :
hyper.drawPolyline(0, 1, complex.I, -1, "red")
-- drawPolylineFromTable :
local points={}
local N=30
for k=1, N do
  table.insert(points, complex.polar(k/N, k*
      ↪ math.pi/3))
end
hyper.drawPolylineFromTable(points,"teal")
hyper.tikzEnd()
\end{luacode*}
```



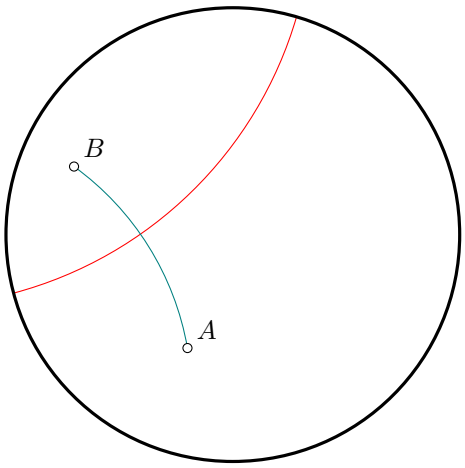
2.2.7 Lines and rays (point and tangent vector)

```
\begin{luacode*}
local A = complex(-0.2,-0.5)
local B = complex(-0.5,0.5)
local C = complex(0,0.2)
local v =  complex(1,0)
hyper.tikzBegin()
myVec = ">=latex, red"
myLine = "dashed, gray"
hyper.drawLineFromVector(A, v, myLine)
hyper.drawRayFromVector(B, v, myLine)
hyper.drawRayFromVector(C, v, myLine)
hyper.drawVector(A,v,myVec)
hyper.drawVector(B,v,myVec)
hyper.drawVector(C,v,myVec)
hyper.drawPoints(A, B,C)
hyper.tikzEnd()
\end{luacode*}
```



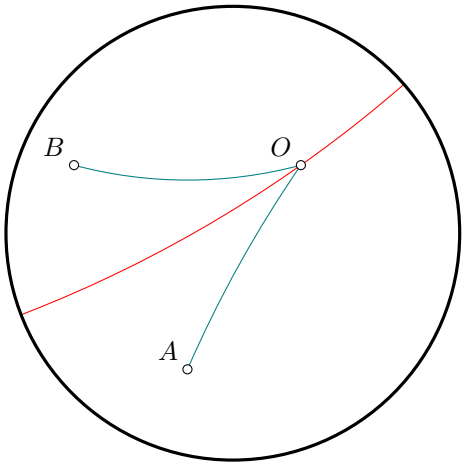
2.2.8 Perpendicular bisector

```
\begin{luacode*}
local A = complex(-0.2,-0.5)
local B = complex(-0.7,0.3)
hyper.tikzBegin()
hyper.drawPerpendicularBisector(A,B,"red")
hyper.drawSegment(A, B, "teal")
hyper.drawPoints(A,B)
hyper.labelPoints(A,"$A$", B, "$B$", "above
    ↪ right")
hyper.tikzEnd()
\end{luacode*}
```



2.2.9 Angle bisector

```
\begin{luacode*}
local A = complex(-0.2,-0.6)
local O = complex(0.3,0.3)
local B = complex(-0.7,0.3)
hyper.tikzBegin()
hyper.drawAngleBisector(A,O,B,"red")
hyper.drawPolyline(A,O,B, "teal")
hyper.drawPoints(A,O,B)
hyper.labelPoints(A, "$A$", O, "$O$", B, "$B$"
    ↪ )
hyper.tikzEnd()
\end{luacode*}
```

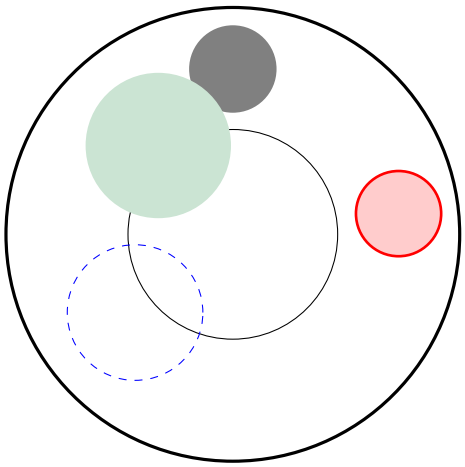


2.3 Circles, semicircles, arcs

2.3.1 Center and radius: drawCircle and fillCircle

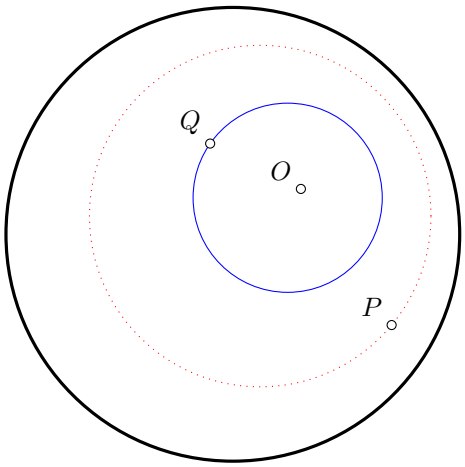
```
\begin{luacode*}
local A = complex(0,0)
local B = complex(-0.5,-0.4)
local C = complex(0.8,0.1)
local D = .8*complex.I
local E = complex.polar(.6,math.rad(130))

hyper.tikzBegin()
hyper.drawCircle(A,1) -- default : black
hyper.drawCircle(B,1,"dashed, blue")
hyper.drawCircle(C,1,"thick, red, fill=red!20"
    ↪ )
hyper.fillCircle(D, 1) -- default : gray
hyper.fillCircle(E, 1,"ForestGreen!20")
hyper.tikzEnd()
\end{luacode*}
```



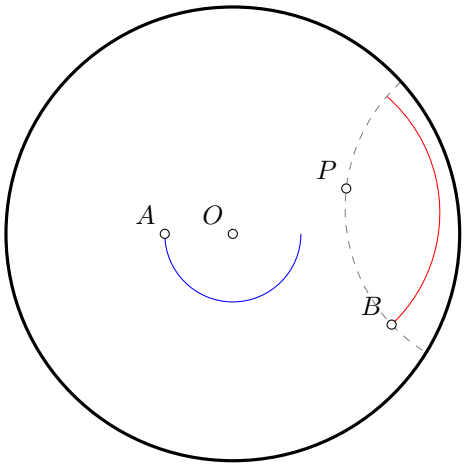
2.3.2 Circle through point

```
\begin{luacode*}
local O = complex(0.3,0.2)
local P = complex(0.7,-0.4)
local Q = complex(-0.1,0.4)
hyper.tikzBegin()
hyper.drawCircleThrough(O, P, "dotted, red")
-- or draw by hand :
hyper.drawCircle(O, hyper.distance(O,Q), "blue
    ↪ ")
hyper.drawPoints(O, P, Q)
hyper.labelPoints(O, "$O$", P, "$P$", Q, "$Q$"
    ↪ )
hyper.tikzEnd()
\end{luacode*}
```



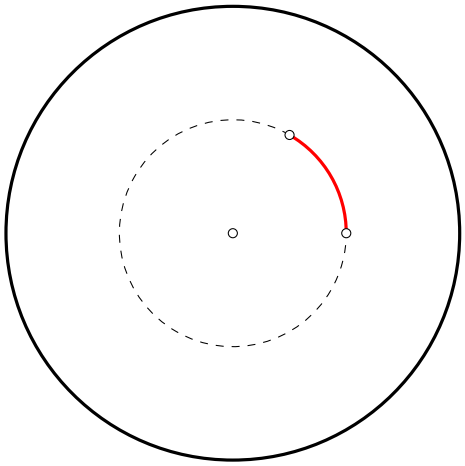
2.3.3 Semicircles

```
\begin{luacode*}
local O = complex(0,0)
local A = complex(-0.3,0)
local P = complex(0.5,0.2)
local B = complex(0.7,-0.4)
hyper.tikzBegin()
hyper.drawSemicircle(O, A, "blue")
hyper.drawSemicircle(P, B, "red")
hyper.labelPoints(O, "$O$", A, "$A$", P, "$P$"
    ↪ , B, "$B$")
hyper.drawLine(P,B, "gray, dashed")
hyper.drawPoints(O, A, P, B)
hyper.tikzEnd()
\end{luacode*}
```



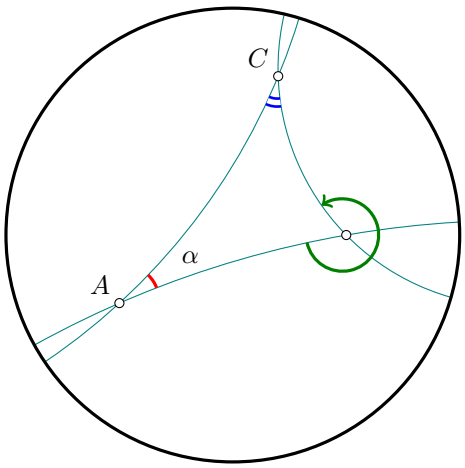
2.3.4 Arc (center and endpoints)

```
\begin{luacode*}
local O = complex(0,0)
local A = complex(0.5,0)
local B = complex.polar(0.5,math.pi/3)
hyper.tikzBegin()
hyper.drawArc(O, A, B, "very thick, red")
hyper.drawArc(O, B, A, "dashed")
hyper.drawPoints(O,A,B)
hyper.tikzEnd()
\end{luacode*}
```



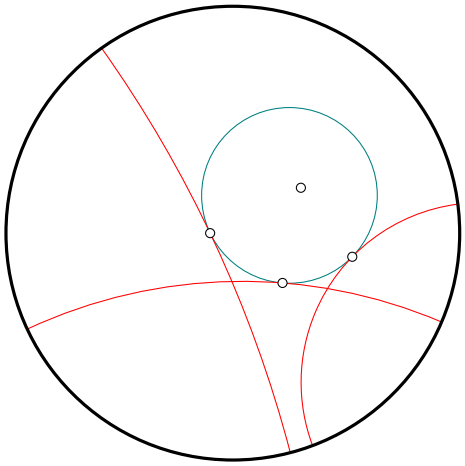
2.3.5 Angles

```
\begin{luacode*}
local A = complex(-.5,-.3)
local B = complex(0.5,0)
local C = complex(0.2,.7)
hyper.tikzBegin()
hyper.drawLines(A, B, B, C, C, A, "teal")
hyper.drawAngle(B, A, C, "red, very thick")
hyper.drawAngle(A, C, B, "thick, double,
    ↪ double distance=2pt, blue")
hyper.drawAngle(A, B, C, "very thick, ->,
    ↪ green!50!black")
hyper.drawPoints(A, B, C)
hyper.labelPoints(A, "$A$", C, "$C$")
hyper.labelPoint(N, "$B$", "above right")
-- poor man's labelAngle:
hyper.labelPoint(A,"$\\alpha$", "above=.6cm,
    ↪ right=.7cm")
hyper.tikzEnd()
\end{luacode*}
```



2.3.6 Tangent (line) at point

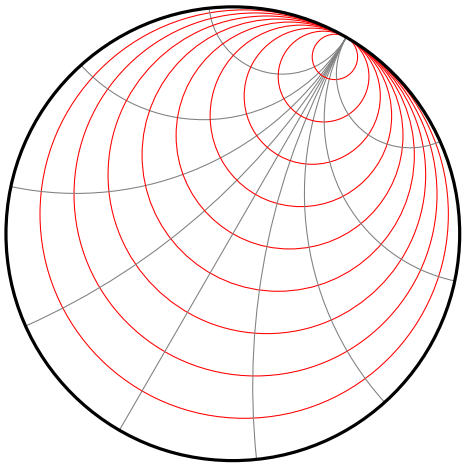
```
\begin{luacode*}
local O = complex(0.3,0.2)
local P = complex(-0.1,0)
local phi  =  hyper.rotation(O,math.pi/4)
local Q =phi(P)
local R = phi(Q)
hyper.tikzBegin()
hyper.drawCircleThrough(O, P, "teal")
hyper.drawTangentAt(O,P,"red")
hyper.drawTangentAt(O,Q,"red")
hyper.drawTangentAt(O,R,"red")
hyper.drawPoints(O, P, Q, R)
hyper.tikzEnd()
\end{luacode*}
```



2.4 Horocycles and hypercycles

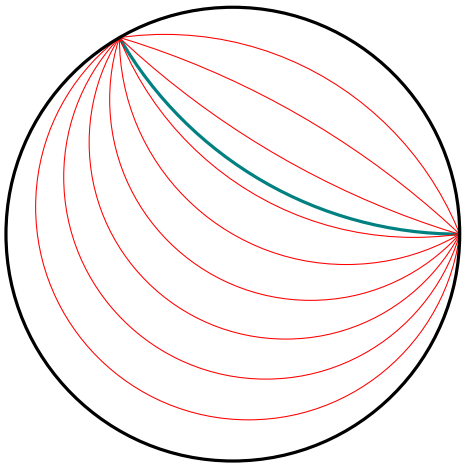
2.4.1 Horocycles

```
\begin{luacode*}
local theta=math.pi/3
local idealPoint = complex.exp_i(theta)
local N = 10
hyper.tikzBegin()
for k=1,N-1 do
  local point = idealPoint * (2*k/N-1)
  hyper.drawHorocycle(idealPoint,point,"red")
  -- and an orthogonal geodesic :
  local endPoint = complex.exp_i(theta+2*k*
    ↪ math.pi/N)
  hyper.drawLine(idealPoint,endPoint,"gray")
end
hyper.tikzEnd()
\end{luacode*}
```



2.4.2 Hypercycles

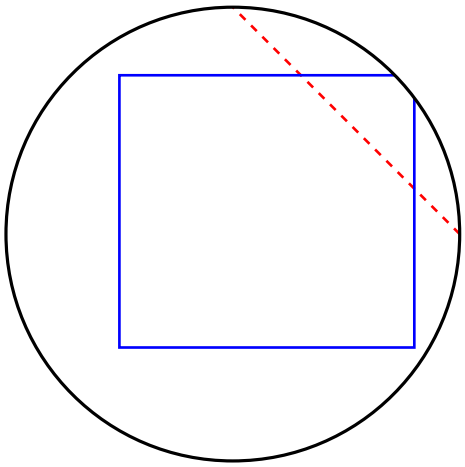
```
\begin{luacode*}
hyper.tikzBegin()
local A = 1
local B = complex.J
hyper.drawLine(A,B,"very thick, teal")
local N = 10
for k=1,N-1 do
  local P = (2*k/N-1)*(-B^2)
  hyper.drawHypercycleThrough(P, A, B, "red")
end
hyper.tikzEnd()
\end{luacode*}
```



2.5 Printing custom strings to Ti*k*Z

In case you want to insert custom Ti*k*Z instructions into the Ti*k*Z picture, the library provides a `tikzPintf()` function, which is essentially `tex.print` composed with `string.format()`, but also saves the printed string to the Ti*k*Z buffer.

```
\begin{luacode*}
local A = complex.ONE
local B = complex.I
local C = (-A-B)/2
hyper.tikzBegin()
hyper.tikzPrintf(
  "\\draw[%s] (%f,%f) -- (%f,%f) ;",
  "red, thick, dashed", A.re, A.im, B.re, B.im
)
hyper.tikzPrintf(
  "\\draw[%s] (%f,%f) rectangle (%f,%f) ;",
  "blue, thick", C.re, C.im, 0.8,0.7
)
hyper.tikzEnd()
\end{luacode*}
```



The package doesn't expose Euclidean drawing functions. Users should write TikZ directly in the picture as above with `tikzPrintf`, or use some other library, for example the amazing library `tkz-euclide`.

2.6 Modifying default styles

The drawing default styles are defined in Lua constants that the user can modify. Keys and default values include:

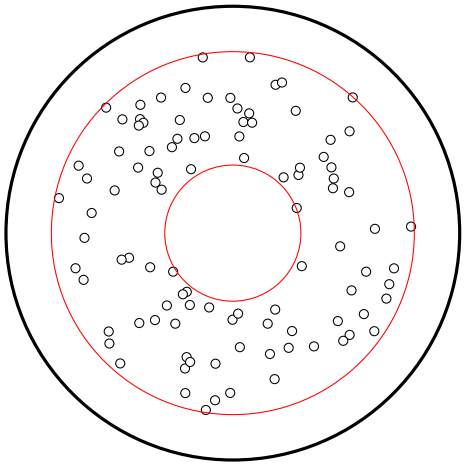
```
hyper.LABEL_STYLE = "above␣left"
hyper.DRAW_POINT_RADIUS = 0.02
hyper.DRAW_POINT_STYLE = "white,␣draw=black"
hyper.BOUNDARY_CIRCLE_STYLE = "very␣thick,␣black"
```

For a complete list, see the source file `luahyperbolic-tikz.lua`

3 Defining points

3.1 Random points with uniform density

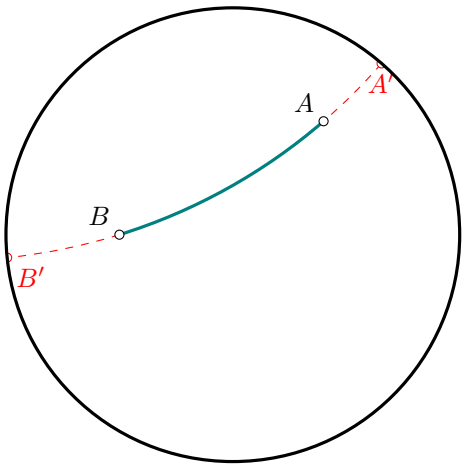
```
\begin{luacode*}
local rMin, rMax = .3, .8
hyper.tikzBegin()
for k=1,100 do
  local M = hyper.randomPoint(rMin,rMax)
  hyper.drawPoint(M)
end
local fmt = "\\draw[red] (0,0) circle (%s);"
hyper.tikzPrintf(fmt, rMin)
hyper.tikzPrintf(fmt, rMax)
hyper.tikzEnd()
\end{luacode*}
```



3.2 Endpoints, midpoint, interpolate

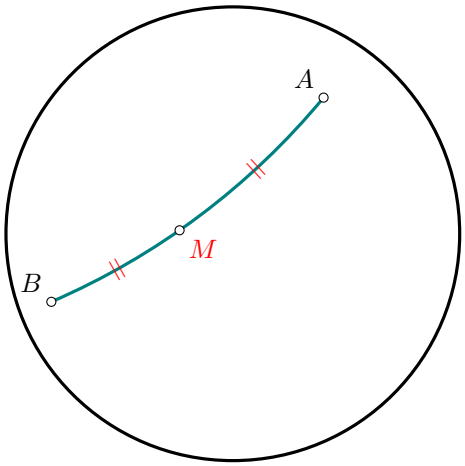
3.2.1 Endpoints

```
\begin{luacode*}
local A = complex(.4,.5)
local B = complex(-.5,0)
local AA, BB = hyper.endpoints(A,B)
hyper.tikzBegin()
hyper.drawSegment(A,B, "very thick, teal")
hyper.drawSegments(AA,A, B, BB, "dashed, red")
hyper.labelPoints(A, "$A$", B, "$B$")
hyper.labelPoint(AA, "$A'$", "red, below")
hyper.labelPoint(BB, "$B'$", "red, below right
    ↪ ")
hyper.drawPoints(A, B)
hyper.drawPoints(AA,BB,"white, draw=red")
hyper.tikzEnd()
\end{luacode*}
```



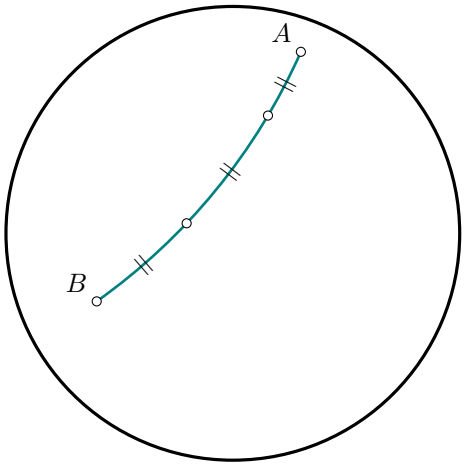
3.2.2 Midpoint

```
\begin{luacode*}
local A = complex(0.4,0.6)
local B = complex(-0.8,-0.3)
local M = hyper.midpoint(A,B)
hyper.tikzBegin()
hyper.drawSegment(A,B, "very thick, teal")
hyper.drawPoints(A, B, M)
hyper.labelPoints(A, "$A$", B, "$B$")
hyper.labelPoint(M, "$M$", "below right, red")
local marking = "\\textcolor{red}{||}"
hyper.markSegment(A,M,marking)
hyper.markSegment(M,B,marking)
hyper.tikzEnd()
\end{luacode*}
```



3.2.3 Interpolate and barycenter

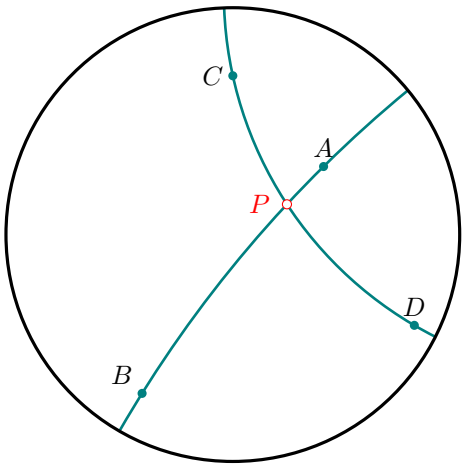
```
\begin{luacode*}
local A = complex(0.3,0.8)
local B = complex(-0.6,-0.3)
local P1 = hyper.interpolate(A, B, 1/3)
local P2 = hyper.interpolate(A, B, 2/3)
hyper.tikzBegin()
hyper.drawSegment(A,B, "thick, teal")
hyper.labelPoints(A, "$A$", B, "$B$")
hyper.markSegments(A, P1, P1, P2, P2, B, "||")
hyper.drawPoints(A, B, P1, P2)
hyper.tikzEnd()
\end{luacode*}
```



3.3 Intersections

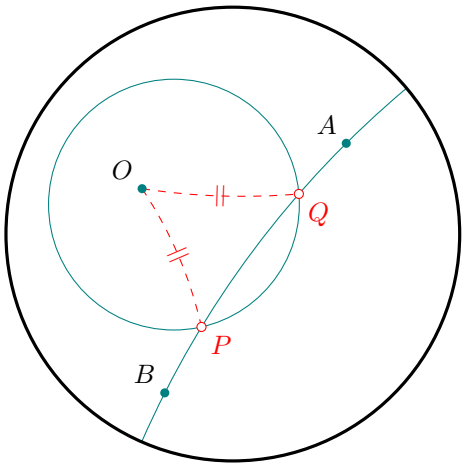
3.3.1 Line-Line (LL) intersection

```
\begin{luacode*}
local A = complex(0.4,0.3)
local B = complex(-0.4,-0.7)
local C = complex(0,0.7)
local D = complex(0.8,-0.4)
local P = hyper.interLL(A, B, C, D)
hyper.tikzBegin()
hyper.drawLines(A, B, C, D, "thick, teal")
hyper.drawPoints(A, B, C, D,"thick, teal")
hyper.drawPoint(P,"white, draw=red")
hyper.labelPoint(A, "$A$", "above")
hyper.labelPoint(B, "$B$")
hyper.labelPoint(C, "$C$", "left")
hyper.labelPoint(D, "$D$", "above")
hyper.labelPoint(P, "$P$", "left=.1cm, red")
hyper.tikzEnd()
\end{luacode*}
```



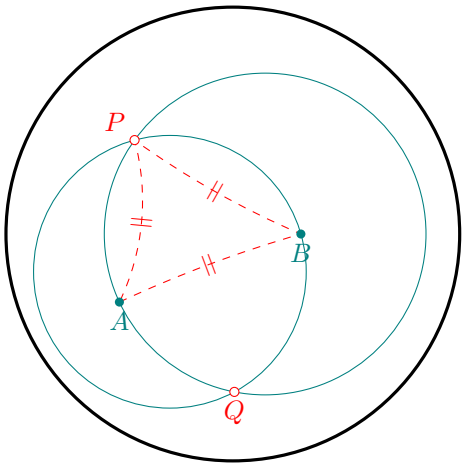
3.3.2 Line-circle (LC) intersection

```
\begin{luacode*}
local A = complex(0.5,0.4)
local B = complex(-0.3,-0.7)
local O = complex(-0.4,0.2)
local radius = 1.5
local P, Q = hyper.interLC(A, B, O, radius)
hyper.tikzBegin()
hyper.drawLine(A, B, "teal")
hyper.drawCircle(O,radius, "teal")
hyper.drawSegments(O, P, O, Q, "red, dashed")
hyper.drawPoints(A, B, O, "teal")
hyper.drawPoints(P, Q, "white, draw=red")
hyper.labelPoints(A, "$A$", B, "$B$", O, "$O$"
    ↪ )
hyper.labelPoints(P, "$P$", Q, "$Q$", "red,
    ↪ below right")
local mark = "\\textcolor{red}{||}"
hyper.markSegments(O, P, O, Q, mark)
hyper.tikzEnd()
\end{luacode*}
```



3.3.3 Circle-circle (CC) intersection

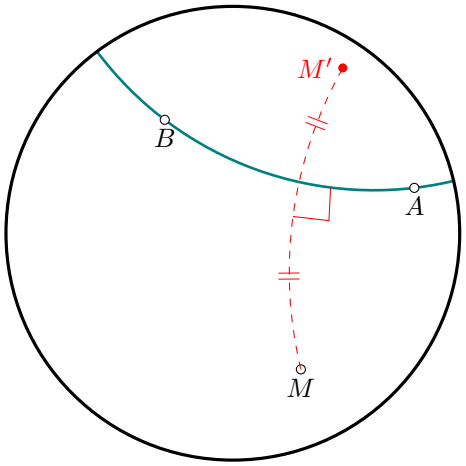
```
\begin{luacode*}
local A = complex(-0.5,-.3)
local B = complex(0.3,0)
local r = hyper.distance(A, B)
local P, Q = hyper.interCC(A, r, B, r)
hyper.tikzBegin()
hyper.drawCircle(A, r, "teal")
hyper.drawCircle(B, r, "teal")
hyper.drawPolygon(A, B, P, "dashed, red")
hyper.drawPoints(P, Q, "white, draw=red")
hyper.drawPoints(A, B, "teal")
hyper.labelPoints(A, "$A$", B, "$B$", "below,
    ↪ teal")
hyper.labelPoints(P, "$P$", "above left, red")
hyper.labelPoint(Q, "$Q$", "red, below")
local mark = "\\textcolor{red}{||}"
hyper.markSegments(A, B, B, P, P, A, mark)
hyper.tikzEnd()
\end{luacode*}
```



3.4 Reflections and projections

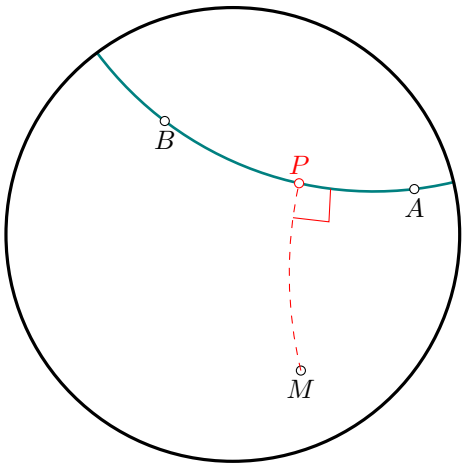
3.4.1 Reflection across a geodesic line

```
\begin{luacode*}
local A = complex(0.8,0.2)
local B = complex(-0.3,0.5)
local M = complex(0.3,-0.6)
local Mp = (hyper.reflection(A,B))(M)
hyper.tikzBegin()
hyper.drawLine(A, B, "thick, teal")
hyper.drawPoints(A, B, M)
hyper.drawPoint(Mp,"red")
hyper.drawSegment(M,Mp,"dashed,red")
hyper.labelPoint(Mp,"$M'$", "left, red")
local P = hyper.interLL(A, B, M, Mp)
local mark="\\textcolor{red}{||}"
hyper.labelPoints(A, "$A$", B, "$B$", M, "$M$"
    ↪ , "below")
hyper.markSegments(M, P, P, Mp, mark)
hyper.drawRightAngle(M, P, A, "red")
hyper.tikzEnd()
\end{luacode*}
```



3.4.2 Projection onto a geodesic line

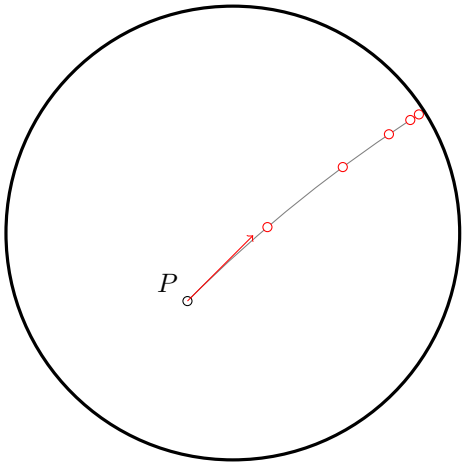
```
\begin{luacode*}
local A = complex(0.8,0.2)
local B = complex(-0.3,0.5)
local M = complex(0.3,-0.6)
local P = (hyper.projection(A,B))(M)
hyper.tikzBegin()
hyper.drawLine(A, B, "thick, teal")
hyper.drawPoints(A, B, M)
hyper.drawPoint(P,"white, draw=red")
hyper.drawSegment(M,P,"dashed,red")
hyper.labelPoint(P,"$P$", "above, red")
hyper.labelPoints(A, "$A$", B, "$B$", M, "$M$"
    ↪ , "below")
hyper.drawRightAngle(M,P,A,"red")
hyper.tikzEnd()
\end{luacode*}
```



3.5 Exponential map

3.5.1 Image of a vector by exponentiel map at a point

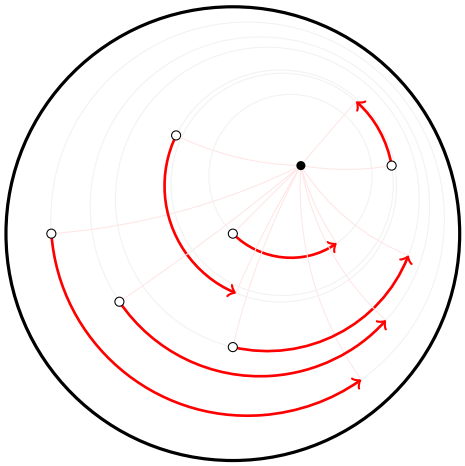
```
\begin{luacode*}
local P = complex(-.2,-0.3)
local vect = complex.exp_i(math.pi/4)
hyper.tikzBegin()
hyper.drawRayFromVector(P, vect, "gray")
for k=1,5 do
  local A = hyper.expMap(P)(k*vect)
  hyper.drawPoint(A, "white, draw=red")
end
hyper.drawPoint(P)
hyper.labelPoint(P,"$P$")
hyper.drawVector(P,vect, "red")
hyper.tikzEnd()
\end{luacode*}
```



3.6 Automorphisms (direct isometries)

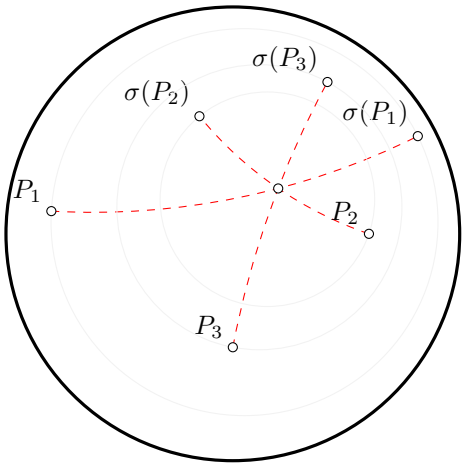
3.6.1 (Elliptic) Rotation by 60°

```
\begin{luacode*}
local O = complex(0.3,0.3)
local r = hyper.rotation(O,math.pi/3)
local points = {0, -.8, complex.J/2, -complex.
    ↪ I/2, complex(-.5,-.3), complex(.7,.3)}
hyper.tikzBegin()
for _,P in ipairs(points) do
  local rP = r(P)
  hyper.drawCircleThrough(O,P, "gray!10")
  hyper.drawArc(O,P,rP,"->, red, thick")
  hyper.drawPolyline(P,O,rP, "red!10")
  hyper.drawPoint(P)
end
hyper.drawPoint(O,"black")
hyper.tikzEnd()
\end{luacode*}
```



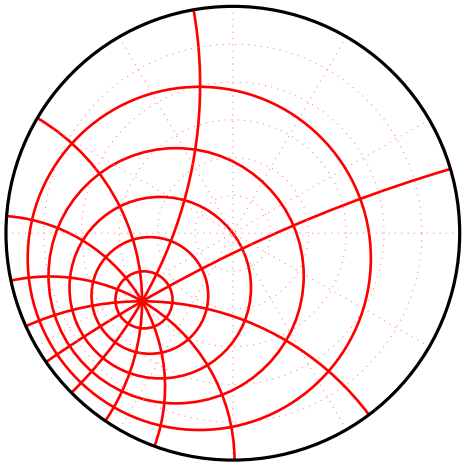
3.6.2 Symmetry around a point (rotation by 180°)

```
\begin{luacode*}
local O = complex(0.2,0.2)
local s = hyper.symmetry(O)
local points={complex(-.8,.1), complex(.6,0),
    ↪ complex(0,-.5)}
hyper.tikzBegin()
for k, P in ipairs(points) do
  hyper.drawCircleThrough(O, P, "gray!10")
  local sP = s(P)
  hyper.drawSegment(P,sP, "red, dashed")
  hyper.drawPoints(P,sP)
  hyper.labelPoint(P, "$P_{".. k .."}$")
  hyper.labelPoint(sP, "$\\sigma(P_{".. k .."
      ↪ })$")
end
hyper.drawPoint(O)
hyper.tikzEnd()
\end{luacode*}
```



3.6.3 Hyperbolic automorphism

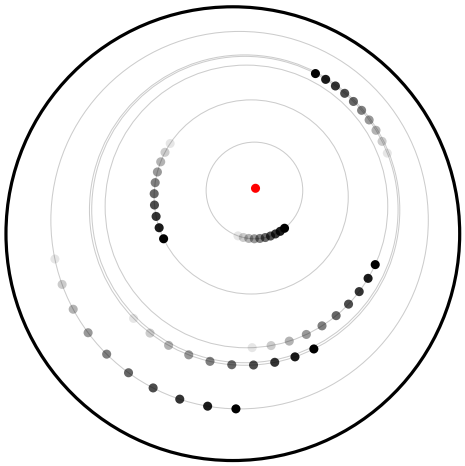
```
\begin{luacode*}
local A = complex(0.4,0.3)
local phi = hyper.automorphism(A,0)
local N=6
hyper.tikzBegin()
style1 = "dotted, red!50"
style2 =  "thick, red"
for k=1,N do
  local P = complex.polar(k/N, k*math.pi/N)
  local Q = phi(P)
  hyper.drawLine(0,P,style1)
  hyper.drawLine(-A,Q,style2)
  if k < N then
    hyper.drawCircleThrough(0,P, style1)
    hyper.drawCircleThrough(-A,Q,style2)
  end
end
hyper.tikzEnd()
\end{luacode*}
```



3.7 Orbits under automorphisms

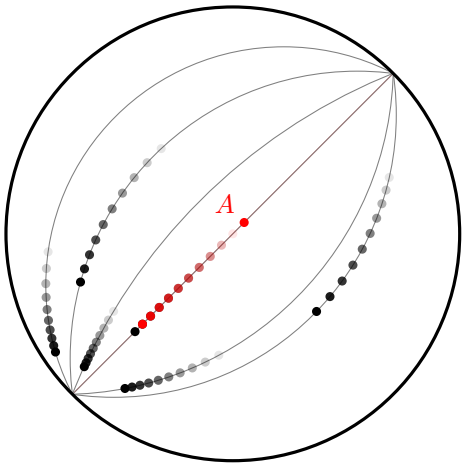
3.7.1 Orbit under elliptic automorphism (rotation)

```
\begin{luacode*}
local nbIterations = 10
local O = complex(0.1,0.2)
local points = {0, -.8, complex.J/2, -complex.
    ↪ I/2, complex(-.5,-.3), complex(.7,.3)}
local rot = hyper.rotation(O,math.pi/30) --
    ↪ angle in radians
hyper.tikzBegin()
hyper.drawPoint(O,"red")
for _,P in ipairs(points) do
  hyper.drawCircleThrough(O,P,"draw opacity
      ↪ =0.2")
  hyper.drawPointOrbit(P, rot, nbIterations)
end
hyper.tikzEnd()
\end{luacode*}
```



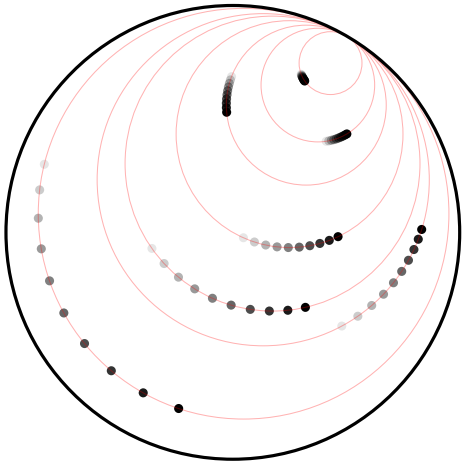
3.7.2 Orbit under hyperbolic automorphism

```
\begin{luacode*}
local points = {0, -.8, complex.J/2, -complex.
    ↪ I/2, complex(-.5,-.3), complex(.7,.3)}
local nbIterations = 10
local A = complex(0.05,0.05)
local phi = hyper.automorphism(A,0)
local X, Y = hyper.endpoints(A,0)
hyper.tikzBegin()
hyper.drawLine(A,0,"red!50")
for _,P in ipairs(points) do
  hyper.drawHypercycleThrough(P,X,Y,"gray")
  hyper.drawPointOrbit(P, phi, nbIterations)
end
hyper.drawPoint(A,"red")
hyper.labelPoint(A,"$A$", "above left, red")
hyper.drawPointOrbit(A, phi, nbIterations, "
    ↪ red")
hyper.tikzEnd()
\end{luacode*}
```



3.7.3 Orbit under parabolic automorphism

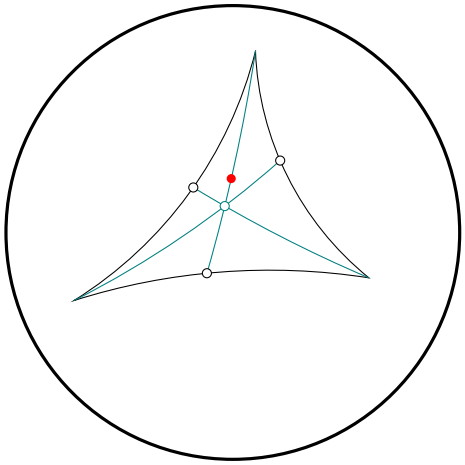
```
\begin{luacode*}
local nbIterations = 10
local limit = complex.exp_i(math.pi/3) -- on
    ↪ circle
local phi = hyper.parabolic(limit,math.pi/30)
local points={
  complex(0,0), complex(0,.7),
  complex(.4,.4), complex(.3,.7),
  complex(-.8,.4), complex(-.4,0),
  complex(.4,-.45)}
hyper.tikzBegin()
for _,P in ipairs(points) do
  hyper.drawPointOrbit(P, phi, nbIterations)
  hyper.drawHorocycle(limit, P, "red, draw
      ↪ opacity=.3")
end
hyper.tikzEnd()
\end{luacode*}
```



3.8 Triangle geometry (CAUTION)

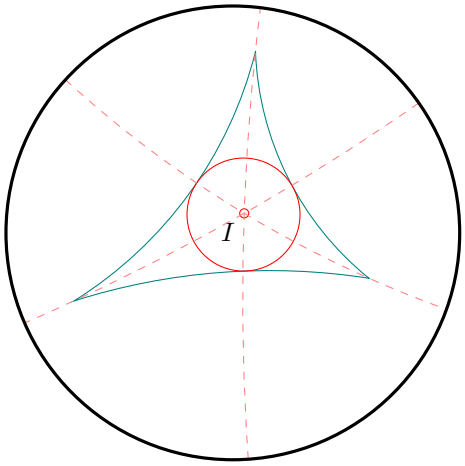
3.8.1 Centroid (intersection of medians)

```
\begin{luacode*}
hyper.tikzBegin()
local A = complex(0.1,0.8)
local B = complex(-0.7,-0.3)
local C = complex(0.6,-0.2)
local AA = hyper.midpoint(B,C)
local BB = hyper.midpoint(C,A)
local CC = hyper.midpoint(A,B)
local G = hyper.triangleCentroid(A,B,C)
hyper.drawTriangle(A,B,C)
hyper.drawSegments(A, AA, B, BB, C, CC, "teal"
    ↪ )
hyper.drawPoints(AA,BB,CC)
hyper.drawPoint(G,"white, draw=teal")
-- CAUTION :
local wrong = hyper.interpolate(A,AA,2/3)
hyper.drawPoint(wrong, "red")
hyper.tikzEnd()
\end{luacode*}
```



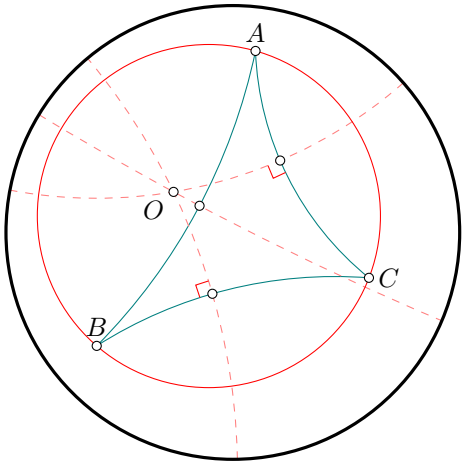
3.8.2 Incenter and incircle

```
\begin{luacode*}
hyper.tikzBegin()
local A = complex(0.1,0.8)
local B = complex(-0.7,-0.3)
local C = complex(0.6,-0.2)
local I = hyper.triangleIncenter(A,B,C)
hyper.drawTriangle(A,B,C,"teal")
hyper.drawPoint(I,"white, draw=red")
hyper.labelPoint(I,"$I$", "below left")
hyper.drawIncircle(A,B,C,"red")
-- show intersection :
local mystyle = "dashed, red!50"
hyper.drawAngleBisector(A,B, C, mystyle)
hyper.drawAngleBisector(B,C,A, mystyle)
hyper.drawAngleBisector(C,A,B, mystyle)
hyper.tikzEnd()
\end{luacode*}
```



3.8.3 Circumcenter and circumcircle (CAUTION: does not always exist)

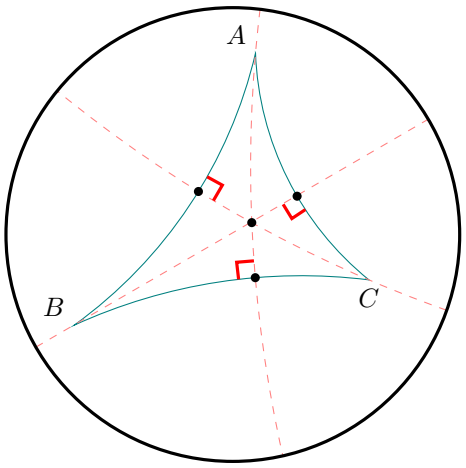
```
\begin{luacode*}
local A = complex(0.1,0.8)
local B = complex(-0.6,-0.5)
local C = complex(0.6,-0.2)
local O = hyper.triangleCircumcenter(A,B,C)
local BC = hyper.midpoint(B,C)
local CA = hyper.midpoint(C,A)
local AB = hyper.midpoint(A,B)
hyper.tikzBegin()
local mystyle = "dashed, red!50"
hyper.drawTriangle(A,B,C,"teal")
hyper.drawPerpendicularBisector(A, B, mystyle)
hyper.drawPerpendicularBisector(B, C, mystyle)
hyper.drawPerpendicularBisector(C, A, mystyle)
hyper.drawCircumcircle(A,B,C,"red")
hyper.labelPoint(O,"$O$", "below left")
hyper.drawRightAngle(O, CA, C,"red",1/8)
hyper.drawRightAngle(O,BC,B,"red",1/8)
hyper.drawPoints(O,A,B,C,AB,BC,CA)
hyper.labelPoints(A, "$A$", B, "$B$", "above")
hyper.labelPoints(C, "$C$", "right")
hyper.tikzEnd()
\end{luacode*}
```



3.8.4 Orthocenter (CAUTION: does not always exist)

```
\begin{luacode*}
local A = complex(0.1,0.8)
local B = complex(-0.7,-0.4)
local C = complex(0.6,-0.2)
local H = hyper.triangleOrthocenter(A,B,C)
local AA = hyper.projection(B,C)(A)
local BB = hyper.projection(C,A)(B)
local CC = hyper.projection(A,B)(C)

hyper.tikzBegin()
local mystyle = "dashed, red!50"
local style2 = "very thick, red"
hyper.drawTriangle(A,B,C,"teal")
hyper.drawPerpendicularThrough(A,B,C,mystyle)
hyper.drawPerpendicularThrough(B,C,A,mystyle)
hyper.drawPerpendicularThrough(C,A,B,mystyle)
hyper.drawRightAngle(A,AA,B,style2,1/15)
hyper.drawRightAngle(B,BB,C,style2,1/8)
hyper.drawRightAngle(C,CC,A,style2,1/12)
hyper.labelPoints(A, "$A$", B, "$B$")
hyper.labelPoints(C, "$C$","below")
hyper.drawPoints(AA,BB,CC,H,"above")
hyper.tikzEnd()
\end{luacode*}
```

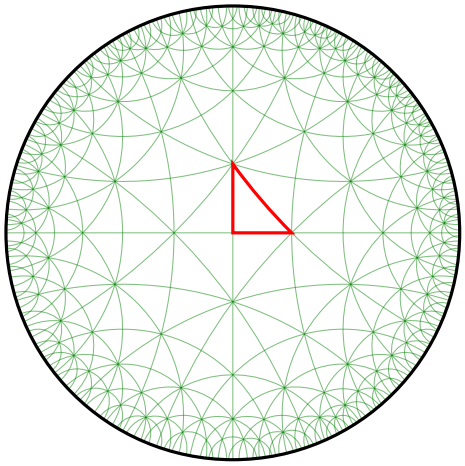


4 Tilings

4.1 Draw Tilings

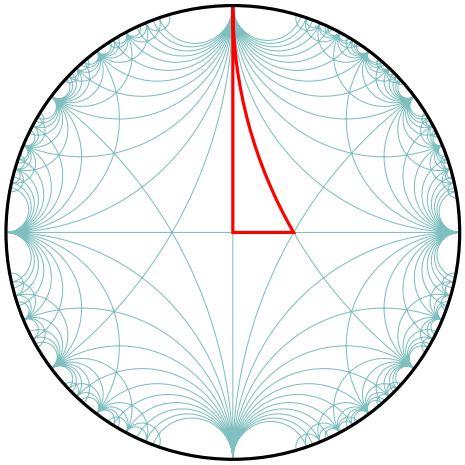
4.1.1 Triangle tiling (almost) by hand

```
\begin{luacode*}
local PI = math.pi
local A, B, C = hyper.triangleWithAngles(PI/2,
    ↪  PI/4, PI/5)
local geodesics = {
  {hyper.endpoints(A,B)},
  {hyper.endpoints(B,C)},
  {hyper.endpoints(C,A)}
}
geodesics = hyper.propagateGeodesics(geodesics
    ↪ , 12) -- depth = 12
hyper.tikzBegin()
hyper.drawLinesFromTable(geodesics, "green!50!
    ↪ black, draw opacity=.5")
hyper.drawTriangle(A,B,C,"red, very thick")
hyper.tikzEnd()
\end{luacode*}
```



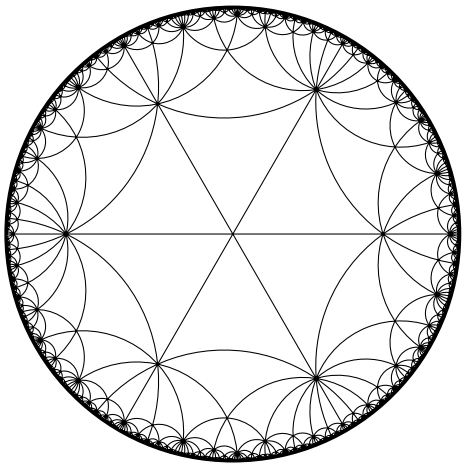
4.1.2 Triangle tiling with less code

```
\begin{luacode*}
local PI = math.pi
local DEPTH = 12
local A, B, C = hyper.triangleWithAngles(PI/2,
    ↪ PI/3,0)
hyper.tikzBegin()
hyper.drawTilingFromTriangle(A, B, C, DEPTH, "
    ↪ teal, draw opacity=.5")
hyper.drawTriangle(A,B,C,"red, very thick")
hyper.tikzEnd()
\end{luacode*}
```



4.1.3 Triangle tiling with even less code

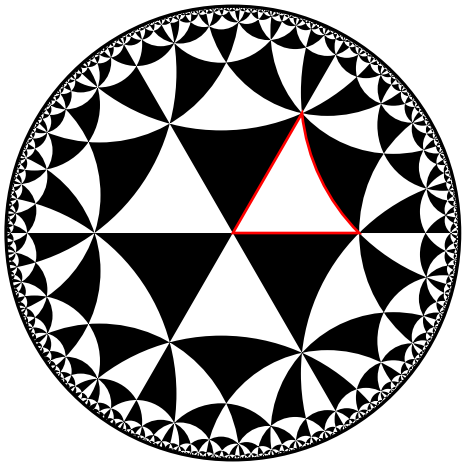
```
\begin{luacode*}
local PI = math.pi
hyper.tikzBegin()
hyper.drawTilingFromAngles(PI/3,PI/5,PI/7,12)
hyper.tikzEnd()
\end{luacode*}
```



4.2 Fill tilings

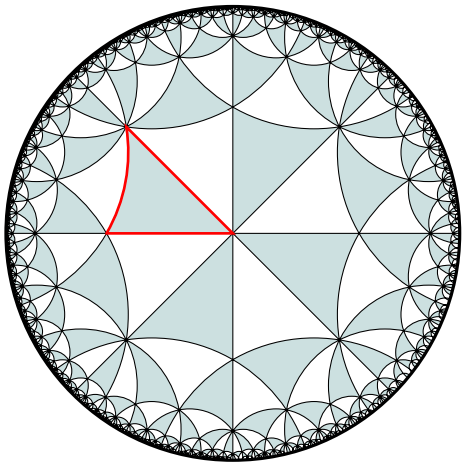
4.2.1 The 3–4–5 triangle tiling

```
\begin{luacode*}
local PI = math.pi
local DEPTH = 10
local A, B, C = hyper.triangleWithAngles(PI/3,
    ↪  PI/4, PI/5)
hyper.tikzBegin()
hyper.fillTilingFromTriangle(A, B, C, DEPTH)
-- default color is black
hyper.drawPolygon(A, B, C, "thick, red")
hyper.tikzEnd()
\end{luacode*}
```



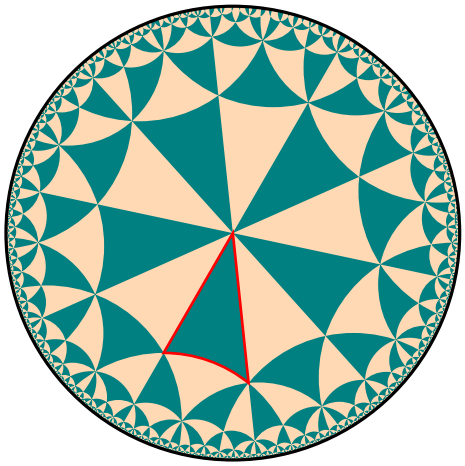
4.2.2 Same 3–4–5 tiling centered on point B

```
\begin{luacode*}
local PI = math.pi
local DEPTH = 10
local A, B, C = hyper.triangleWithAngles(PI/3,
    ↪  PI/4, PI/5)
local phi = hyper.automorphism(B)
A, B, C = phi(A), phi(B), phi(C) -- put B in
    ↪ the center
hyper.tikzBegin()
hyper.fillTilingFromTriangle(A, B, C, DEPTH, "
    ↪ fill opacity=.2, teal!80!black, draw=
    ↪ black")
hyper.drawPolygon(A, B, C, "thick, red")
hyper.tikzEnd()
\end{luacode*}
```



4.2.3 Same 3–4–5 tiling centered on point C

```
\begin{luacode*}
local PI = math.pi
local DEPTH = 10
local A, B, C = hyper.triangleWithAngles(PI/3,
    ↪  PI/4, PI/5)
local phi = hyper.automorphism(C)
A, B, C = phi(A), phi(B), phi(C) -- put C in
    ↪ the center
hyper.tikzBegin()
hyper.tikzPrintf("\\fill[orange!30] (0,0)
    ↪ circle (1);")
hyper.fillTilingFromTriangle(A, B, C, DEPTH, "
    ↪ teal")
hyper.drawPolygon(A, B, C, "thick, red")
hyper.tikzEnd()
\end{luacode*}
```



4.3 Tilings without any Lua code

The package provides a TeX macro for users who want to completely avoid Lua.



The syntax is `\hyperbolicTiling{p}{q}{r}[options]` with p, q, r positive integers or `inf` satisfying $\frac{1}{p} + \frac{1}{q} + \frac{1}{r} < 1$. The associated hyperbolic triangle has angles $\frac{\pi}{p}$, $\frac{\pi}{q}$, and $\frac{\pi}{r}$.

The above tilings were obtained with the following commands:

```
\hyperbolicTiling{3}{4}{5}[scale=4]
\hyperbolicTiling{3}{3}{4}[scale=4,depth=12]
\hyperbolicTiling{2}{3}{7}[scale=2.5, color=teal, depth=16]
\hyperbolicTiling{2}{5}{9}[scale=2.5, color=gray]
\hyperbolicTiling{4}{4}{4}[scale=2.5, depth=6, color=purple,backgroundcolor=pink!30]
\hyperbolicTiling{2}{5}{inf}[scale=2, depth=6, color=brown]
\hyperbolicTiling{2}{inf}{inf}[scale=2, depth=6, color=olive]
\hyperbolicTiling{3}{inf}{inf}[scale=2, depth=6, color=orange]
\hyperbolicTiling{inf}{inf}{inf}[scale=2, depth=5, color=violet]
```

Default values are : `scale=3`, `depth=8`, `color=black`, `backgroundcolor=white`.

5 Overview of the lua libraries

The Lua files are in the `lua/` subdirectory of the repo.

To build the `luahyperbolic.sty` from the Lua source files, run `lua build.lua` on the `lua/` directory. (Lua must be installed.)

The Lua source files are not needed to compile a LaTeX document using luahyperbolic, only the .sty file is necessary.

5.1 complex

Functions

__add	__unm	det	isClose	isNot	rotate
__div	abs	distinct	isColinear	isReal	rotate90
__eq	abs2	dot	isComplex	isUnit	scal
__mul	arg	exp	isImag	log	toPolar
__pow	clone	exp_2ipi	isInteger	new	unit
__sub	coerce	exp_i	isNear	nonzero	
__tostring	conj	invert	isNearInteger	polar	

Constants (numbers and complex numbers)

EPS, EPS_INV I, J, ONE, ZERO, __index

5.2 luahyperbolic

The luahyperbolic library consists of three submodules : luahyperbolic-core, luahyperbolic-tikz and luahyperpolic-tilings. These three submodules are flattened in a single `hyper` global module. (The submodules are still available in three tables hyper.core, hyper.tikz and hyper.tilings if necessary.)

Functions beginning with underscore are available in the module but should not be needed for normal use. Don't use them, they could become private in the future and not be available to users anymore.

Functions

_assert	drawAngleBisector	drawSegments
_assert_in_closed_disk	drawArc	drawSemicircle
_assert_in_disk	drawCircle	drawTangentAt
_circle_to_euclidean	drawCircleThrough	drawTilingFromAngles
_coerce_assert_in_closed_disk	drawCircumcircle	drawTilingFromTriangle
_coerce_assert_in_disk	drawHorocycle	drawTriangle
_ensure_order	drawHypercycleThrough	drawVector
_error	drawIncircle	endpoints
_geodesic_data	drawLine	endpointsAngleBisector
_in_closed_disk	drawLineFromVector	endpointsPerpendicularBisector
_in_disk	drawLines	expMap
_metric_factor	drawLinesFromTable	fillCircle
_midpoint_to_origin	drawPerpendicularBisector	fillTilingFromAngles
_on_circle	drawPerpendicularThrough	fillTilingFromTriangle
_on_line	drawPoint	interCC
_on_segment	drawPointOrbit	interLC
_radial_half	drawPoints	interLL
_radial_scale	drawPolygon	interpolate
_warning	drawPolygonFromTable	labelPoint
angle	drawPolyline	labelPoints
automorphism	drawPolylineFromTable	labelSegment
automorphismFromPairs	drawRay	logMap
automorphismSending	drawRayFromPoints	markSegment
barycenter2	drawRayFromVector	markSegments
distance	drawRegularPolygon	midpoint
distanceToLine	drawRightAngle	mobiusTransformation
drawAngle	drawSegment	parabolic

pointOrbit
projection
propagateGeodesics
randomPoint
reflection
rotation
rotationFromPair
symmetry
symmetryAroundMidpoint
tangentVector
tikzBegin
tikzClearBuffer
tikzDefineNodes
tikzEnd
tikzExport
tikzGetFirstLines
tikzPrintf
tikz_shape_closed_line
tikz_shape_euclidean_segment
tikz_shape_segment
tikz_shape_segment_continuation
triangleCentroid
triangleCircumcenter
triangleIncenter
triangleOrthocenter
triangleWithAngles

Constants

Numbers: DRAW_ANGLE_DIST, DRAW_POINT_RADIUS, EPS, QUANTIZATION_SCALE, TIKZ_MAX_EXTENT, tikzNbPicturesExport
Strings: ANGLE_STYLE, BOUNDARY_CIRCLE_STYLE, CIRCLE_STYLE, DRAW_POINT_STYLE, GEODESIC_STYLE, HOROCYCLE_STYLE, HYPERCYCLE_STYLE, LABEL_STYLE, MARKING_SIZE, MARKING_STYLE, TIKZ_BEGIN_DISK, module, tikzpictureOptions
Tables: core, options, tikz, tikzBuffer, tilings

6 Changelog

Luahyperbolic is still in an early development phase. Changes and bug fixes are being made regularly.

- 2026-11-13: new version (v. 2026-11-13) submitted to CTAN.
- 2026-11-13: Bezier approximation for geodesics that are almost diameters.
- 2026-10-26: added `logMap`.
- 2026-10-23: fixed bug: using `\hyperbolicTiling` inside a `\begin{center}` used to throw an error.
- 2026-09-17: fixed bug in `automorphism(a,theta)` function and added coercion.
- 2026-09-05: changed the way polylines and polygons are drawn : a single tikz shape is now used. This allows to fill polygons.
- 2026-04-12: added function `fillCircle(center, radius, options)`
- 2026-03-23: expMap now takes only one argument (a point), and returns a function. New syntax is `newPoint = hyper.expMap(Point)(vector)`.
- 2026-03-16: first version (v.2026-03-16) uploaded to CTAN.

Index of functions used in this documentation

automorphism, 16

complex
    exp_i, 2

drawAngle, 9
drawAngleBisector, 7
drawArc, 9
drawCircle, 8
drawCircleThrough, 8
drawCircumcircle, 19
drawHorocycle, 10
drawHypercycleThrough, 10
drawIncircle, 19
drawLine, 4
drawLineFromVector, 7
drawLinesFromTable, 21
drawPerpendicularBisector, 7
drawPerpendicularThrough, 20
drawPoint, 4
drawPointOrbit, 17
drawPoints, 4
drawPolygon, 5
drawPolygonFromTable, 5
drawPolyline, 6
drawPolylineFromTable, 6
drawRay, 4
drawRayFromVector, 7
drawRegularPolygon, 6
drawRightAngle, 14
drawSegment, 4
drawSemicircle, 8
drawTangentAt, 9
drawTilingFromAngles, 21
drawTilingFromTriangle, 21
drawTriangle, 5
drawVector, 7

endpoints, 11
expMap, 15

fillCircle, 8
fillTilingFromTriangle, 22

interCC, 13
interLC, 13
interLL, 12
interpolate, 12

labelPoint, 4
labelPoints, 4
labelSegment, 6

markSegment, 6
midpoint, 12

parabolic, 18
projection, 14
propagateGeodesics, 21

randomPoint, 11
reflection, 14
rotation, 15

symmetry, 16

tikzBegin, 2
tikzEnd, 2
tikzExport, 3
tikzPrintf, 10
triangleCentroid, 18
triangleCircumcenter, 19
triangleIncenter, 19
triangleOrthocenter, 20
triangleWithAngles, 5