Chapter 1

Code for Diagram Generation

Included in Appendix 1 is the Python code used to generate the various figures. All files should be saved in the same folder, and only hyperbolictilinggenerator.py needs to be run. The code will create (or modify) a document named texcode.tex. This document uses the standalong package, and should be able to be compiled as is, imported into another document, or simply gutted for the tikz code. A word of warning, the code can generate exceptionally large tikz pictures. There is no check to determine how long the program will run with given inputs, and there is certainly no way of telling how long your LaTeXcompiler will take to actually compile the document.

```
import math
import cmath
import copy
from hyptransformations import *


class Vertex:
    def __init__(self,z,t):
        self.pos=z
        self.type=t
        if t not in ["s","t","u"]:
            print("Not a valid vertex type")
            exit(1)

    def reflect(self,p1,p2):
        return Vertex(reflect(p1,p2,self.pos),self.type)

class Panel:
    def __init__(self,v1,v2,tolerance):
        self.start=v1.pos
        self.end=v2.pos
        if v1.type==v2.type:
```

```
            print("Two vertices on the same panel cannot share a type")
            exit(1)

        types=["s","t","u"]
        types.remove(v1.type)
        types.remove(v2.type)
        self.cotype=types[0]
        self.tolerance=tolerance


    def liesondiameter(self):
        return abs(self.start.real*self.end.imag-self.start.imag*self.end.rea

    def drawingcenter(self):
        if self.liesondiameter():
            return None
        denom=self.start.real*self.end.imag-self.start.imag*self.end.real
        a=(self.start.imag*(abs(self.end)**2)-self.end.imag*(abs(self.start)*
        b=(self.end.real*(abs(self.start)**2)-self.start.real*(abs(self.end)*
        return (-a/2)+1j*(-b/2)

    #This function only makes sense when liesondiameter returns false
    def drawingradius(self):
        if self.liesondiameter():
            return None
        return abs(self.drawingcenter()-self.start)


    def gettexcode(self):
        if self.liesondiameter():
            return "\\draw("+str(self.start.real)+","+str(self.start.imag)+")
        else:
            center=self.drawingcenter()
            return Panel.texcodearc(self.start,self.end,center)

    def getwalltexcode(self):
        if abs(self.start-self.end)<self.tolerance:
            return ""

        if self.liesondiameter():
            step=abs(self.end-self.start)
            startbottom=0
            starttop=3/step
            endbottom=0
            endtop=-3/step
```

```
            while abs(starttop-startbottom)>self.tolerance or abs(endbottom-e
                startmid=(startbottom+starttop)/2
                endmid=(endbottom+endtop)/2
                startguess=self.start+startmid*(self.end-self.start)
                endguess=self.start+endmid*(self.end-self.start)
                if abs(startguess)>=1:
                    starttop=startmid
                else:
                    startbottom=startmid

                if abs(endguess)>=1:
                    endtop=endmid
                else:
                    endbottom=endmid

            return "\\draw("+str(startguess.real)+","+str(startguess.imag)+")


        else:
            center=self.drawingcenter()
            r=self.drawingradius()
            c=abs(center)
            x=(r**2-1-c**2)/(-2*c)
            ypos=math.sqrt(1-x**2)
            yneg=-ypos
            start=(x+1j*ypos)*cmath.exp(1j*cmath.phase(center))
            end=(x+1j*yneg)*cmath.exp(1j*cmath.phase(center))
            return Panel.texcodearc(start,end,center)

    def texcodearc(start,end,center):
        startangle=cmath.phase(start-center)
        endangle=cmath.phase(end-center)
        radius=abs(center-start)

        if abs(endangle-startangle)>cmath.pi:
            if endangle<0:
                endangle+=2*cmath.pi
            else:
                startangle+=2*cmath.pi


        return "\\draw("+str(start.real)+","+str(start.imag)+")␣arc␣("+str(ma
```

```
class Chamber:

    def __init__(self,v1,v2,v3,tolerance):
        self.vertices={v1.type:v1,v2.type:v2,v3.type:v3}
        if len(self.vertices)<3:
            print("A chamber must have vertices of 3 different types.")
            exit(1)
        self.tolerance=tolerance

    def center(self):
        ans=0
        for k in self.vertices.keys():
            ans+=self.vertices[k].pos
        return ans/3

    def getpanel(self,cotype):
        types=["s","t","u"]
        types.remove(cotype)
        return Panel(self.vertices[types[0]],self.vertices[types[1]],self.tol

    def translate(self,coxeterword):
        ans=copy.deepcopy(self)
        for c in reversed(coxeterword):
            if c in ans.vertices.keys():
                fixed=list(ans.vertices.keys())
                fixed.remove(c)
                ans=Chamber(ans.vertices[c].reflect(ans.vertices[fixed[0]].po


            else:
                print("Invalid letter in coxeter word.")
                exit(1)
        return ans
```