STUDY OF ALGORITHMIC GENERATION OF THE PENROSE-STAIRCASE

FERDINAND LEHR

Abstract. In this paper we discuss the algorithmic generation of Penrose-Stairs. Furthermore we will work out several proves, type-classifications and investigate numerical and geometrical behaviour of those staircases under different preconditions. Graphics and coding-examples in different programming-languages are being added to illustrate our subjects.

1. Introduction

In the year 1937 the swedish graphic artist Oscar Reutersvärd created the drawing of a fours-sided staircase, whose stairsteps lead all the way down, ending finally exactly at their starting position again, to form a 2D-parallel projected closed geometrical object that looks like being constructable in 3D-space, but factual is an illusion that's impossible to exist in 3D-space. Later in the year 1958 two english mathematicians, Lionel Penrose and his son Sir Roger Penrose independently discovered and made popular the "impossible staircase".

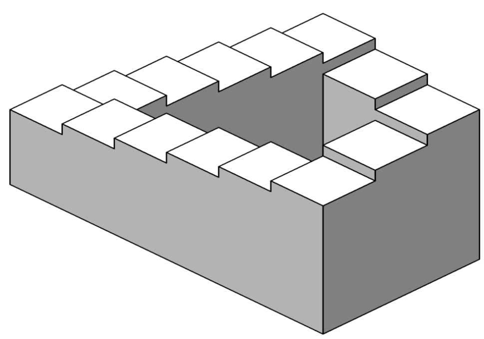

Figure 1. The original Penrose-Staircase.

Though many interesting articles has been written since and lots of beautiful graphics are created still nowadays, there seems to be the lack of a serious investigation how to mathematically and algorithmically obtain a perfect looking Penrose-Staircase in parallel projection graphics.

2. Coordinate System

The well-known original image of the Impossible-Staircase (1) is actually isometric, although back in 1958 the picture had been squeezed in y-size (perhaps to obscure its isometric character). Later we will see and discuss its exact derivation.

In order to obtain a perfect parallel-projected Impossible-Staircase, we start with a basic isometric coordinate-system. We determine, that in flat isometry the coordinates are made out of equilateral triangles and we declare the sidelength of one of those triangles being one single unit u=1.

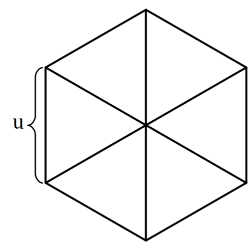

FIGURE 2. Isometric grid with triangle sidelength of one single unit.

Since the original Penrose-Staircase doesn't fit perfectly in integer isometric-coordinates and since we don't have an algorithm yet, we use a slightly different staircase that we easily found by trial-and-error:

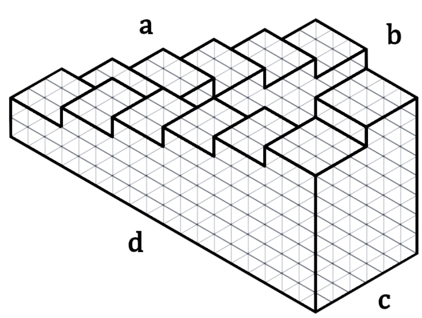

FIGURE 3. Impossible staircase that fits perfectly into an isometric grid

We assert the count of stairsteps for each side a=6 b=2 c=2 d=6, and the side-length of each single stairstep l=3 isometric units, following written as a/b/c/d(l) e.g. 6/2/2/6(3). We define $a, b, c, d \in \mathbb{N}_{>0}$

3. Walking down the steps

Next we need a walking-routine and an ISO-position-cursor, so we define the walkable directions we can go from an ISO-point to another like this:

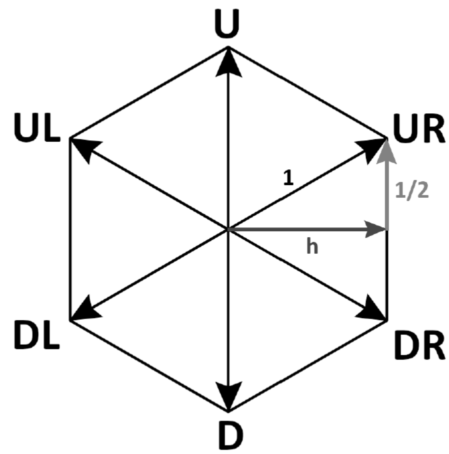

Figure 4. Walkable ISO-directions.

Note that height h in an equilateral triangle is determined by cos(30°)=0,866025403... To project our walking functions into cartesian coordinates, which can be represented by a computer-screen, we define them as follows:

$$U(n) : y = y + n$$

$$D(n) : y = y - n$$

$$UR(n) : x = x + nh; \quad y = y + n\frac{1}{2};$$

$$DR(n) : x = x + nh; \quad y = y - n\frac{1}{2};$$

$$DL(n) : x = x - nh; \quad y = y - n\frac{1}{2};$$

$$UL(n) : x = x - nh; \quad y = y + n\frac{1}{2};$$

Using that functions walking downstairs, we get our path which starts at point S, going all the way down and around the staircase until it finally ends at point S again, defining the shape of our 6/2/2/6(3) staircase:

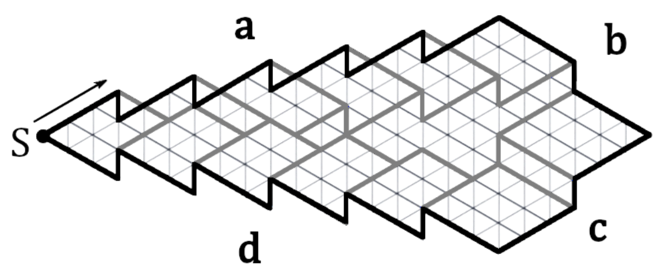

FIGURE 5. The walking path from startpoint S down the staircase and back to S.

$$\begin{aligned}
\text{Side a}: &\ UR(la); \quad D(a-1);\\
\text{Side b}: &\ DR(lb); \quad D(b-1);\\
\text{Side c}: &\ DL(lc); \quad D(c-1);\\
\text{Side d}: &\ UL(ld); \quad D(d-1);
\end{aligned}$$

Firstly, by extracting and joining the walking-formulas for x we get $x = ah + bh - ch - dh$ which is $x = h(a+b-c-d)$. Our x/y-path must start and end at (0,0) so we can say $0 = h(a+b-c-d)$. Because of distance $h > 0$ always, that equation only gets zero if $(a+b-c-d)$ is zero, so our final equation for the x-direction is:

$$0 = a + b - c - d$$

Secondly, by extracting and joining the walking-formulas for y we get

$$y = \frac{1}{2}la - (a-1) - \frac{1}{2}lb - (b-1) - \frac{1}{2}lc - (c-1) + \frac{1}{2}ld - (d-1)$$

Which is simplified

$$y = l\left(\frac{a-b-c+d}{2}\right) - a - b - c - d + 4$$

Same as above with x, we need to set y to zero, in order to end up at the starting-point S again, and then we solve for the single-stairside-length l, so we get a more expressive value in result than just zero:

$$l = \frac{a+b+c+d-4}{\frac{a}{2} - \frac{b}{2} - \frac{c}{2} + \frac{d}{2}}$$

Division by zero occurs here for staircases with all sides having the same count of steps: a=b=c=d, and if a=c and b=d. To avoid this, we declare $a \neq c$.

Also, our result, the length l, has to be always greater than zero to make sense, thus we can assume that if $l <= 0$ the staircase is not valid, as well as for $l = \infty$.

4. Constraints

Since one side a,b,c, or d of the staircase has to contain at least two stairsteps to assure that every single stairstap in the whole staircase has two different directions for going up and down

5. Various

The length l of one single stairstep can be a float-number.
The smallest penrose-staircase is: 3/2/2/3(6).
The original penrose-staircase is 6/3/3/6(4.666666666...)
The growth of the amount of total stairsteps in penrose-stairs seems to be logarithmic

6. Code Examples

Penrose-Staircase-Database CSV-File Generator for PYTHON:

```
###############################################
# PENROSE-STAIRCASE-GENERATOR v1.0            #
# This program calculates ratios for          #
# the "Impossible-Staircase".                 #
# (c) 2022 by F. Lehr                         #
# https://www.ferdinandlehr.de                #
# ferdinand@ferdinandlehr.de                  #
###############################################

import csv

#Calculates the number of staircases for a given (always even) stairsum.
#def P(g):
#    return int((1/8)*g**2-(3/4)*g+1)

#Calculates the number of staircases for a given (always even) stairsum.
def P2(g):
    c=0
    p=1
    k=2
    i=6
    while(i<=g):
        #print("i,g,c",i,g,c)
        c=0
        for j in range(p):
            c+=1
        #print(c)
        p+=k
        k=k+1
        i+=2
    return c


#Set of all PStairs to an upper limit : nth Stairsum
def tz(n):
    r=0
    for i in range(1, n+1):
        r += P2(2*i+4)
    return int(r)

#Calculates the nth stairsum
def NG(n):
    return(n*2+4)

#Calculates the stairsum g for the nth Pstair
def SNP(n):
```

```
    k=1
    while(tz(k)<n):
        k=k+1
    return NG(k)

#calculates how many stairsums occur up to a given g (including stairsum for g)
def SBG(g):
    n=1
    while(NG(n)<g):
        n=n+1
    return n

#Calcs l of the nth PStair.
def LONP(n):
    g=SNP(n)
    #print("Stairsum =",g)
    rpos=P2(g)
    #print("Number of stairs with that stairsum p =", rpos)
    m=int((g-4)/2) #calculate how many times d=l for a given g
    #print("Position within that stairsum range rpos =", rpos)
    #print("m =", m)
    tpos = tz(SBG(g))#n+rpos-1
    #print("Cursor position total tpos = ", tpos)
    u=m
    for j in range(m+1):
        d=1 #divisor
        l=-1 #length
        for i in range(u):
            l=g/d
            d=d+1
            tpos-=1
            #print ("l =", l, "tpos=", tpos)
            if (tpos+1)==n:
                return l
        u=u-1

# returns 3,4,4,5,5,5,6,6,6,6,...
def AINC(p):
    n=1000000000
    c=0
    for i in range(1,n+1):
        for k in range(1,i+1):
            c+=1
            if(c==p):
                return i+2


#calcs a of the nth pstair
```

```
def A_of_NP(n):
    g=SNP(n)
    rpos=P2(g)
    tpos = tz(SBG(g)-1)+1
    #print("rpos,tpos,",rpos,tpos)
    return(AINC(n-tpos+1))

#calcs b of the nth pstair
def B_of_NP(n):
    a=A_of_NP(n)
    g=SNP(n)
    return int(((g+4)/2)-a)

# returns 2,2,3,2,3,4,2,3,4,5,2...
def CINC(p):
    n=100000000
    c=0
    for i in range(1,n+1):
        for k in range(1,i+1):
            #print(i,k)
            c+=1
            if(c==p):
                return k+1

#calcs c of the nth pstair
def C_of_NP(n):
    g=SNP(n)
    rpos=P2(g)
    tpos = tz(SBG(g)-1)+1
    #print("tpos=",tpos)
    return(CINC(n-tpos+1))

#calcs d of the nth pstair
def D_of_NP(n):
    a=A_of_NP(n)
    b=B_of_NP(n)
    c=C_of_NP(n)
    return a+b-c

#calculate and print the nth pstair
def PStair_nth(n):
    a=A_of_NP(n)
    b=B_of_NP(n)
    c=C_of_NP(n)
    d=a+b-c
    g=SNP(n)
    l=LONP(n)
    print(a,b,c,d,g,l)
```

```
#calculate and print 1st pstair to nth pstair
def PStairs_to_n(n):
    for i in range(1,n+1):
        PStair_nth(i)

#calculate and write 1st pstair to nth pstair to CSV-file
def PStairs_to_n_CSV(n):
    f = open('pstairs.csv', 'w', encoding='UTF8', newline='')
    writer = csv.writer(f,delimiter=',')
    for i in range(1,n+1):
        a=A_of_NP(i)
        b=B_of_NP(i)
        c=C_of_NP(i)
        d=a+b-c
        g=SNP(i)
        l=LONP(i)
        #print(a,b,c,d,g,l)
        #data = [a,b,c,d,g,str(l).replace('.', ",")]
        data = [a,b,c,d,g,l]
        writer.writerow(data)
    f.close()

#Main-Program
print("PENROSE-STAIRCASE GENERATOR v1.0")
n=10
print("Calculating the first", n, "Penrose-Stairs")
print("a b c d sum len")
PStairs_to_n(n)
```

Viewer for “Context Free Art”:

```
//--------------------------------------------------//
// THE INFINITY-STAIRCASE VIEWER (CFDG)             //
// (c) and written 2022 by F. Lehr                  //
// For more info visit https://www.ferdinandlehr.de    //
// github.com                                       //
// https://www.contextfreeart.org                   //
// This code is Creative Commons licensed:          //
// Creative Commons Attribution-NonCommercial-      //
// ShareAlike 4.0 International:                    //
// https://creativecommons.org/licenses/by-nc-sa/4.0/  //
//--------------------------------------------------//
CF::Impure=1
//----------------- PLAYAROUND-AREA -------------------//
COLOR1 = [b 0.007 sat 0.5 h 0]
COLOR2 = [b 0.5 sat 0.5 h 150]
COLOR3 = [b 1 sat 0.2 h 200]
// R = 3,2,2,3,6             // Smallest Staircase
   R = 6,3,3,6,4.66666       // Penrose-Staircase 1958
// R = 24,6,14,16,5.6        // For Reutersvard and Escher
// R = 15,2,5,12,3           // Castle of the Wizards
// R = 17,17,10,24,9.14285  // Large Antique Arena
// R = 100,66,67,99,9.93939 // Chinese Wall
// For more ratios visit https://www.ferdinandlehr.de
//--------------------------------------------------//
U = 1
H = 0.866025404
L = R[4]
A=R[0]
B=R[1]
C=R[2]
D=R[3]
WH = D*2
startshape PAGE
shape PAGE {
    PSTAIR[skew 2 6 r -30]
}
shape PSTAIR {
    loop i=A-1,-1,-1 [] {

        S(L)[x ((U*0.5*L)*i+(U/2)*i) y (((H*L)-H)*i)]
    }
    loop m=1,D-1,1 [] {
        xx1 = ((U*0.5*L)*(A-1)+(U/2)*(A-1)+(L*1+U/2)*(B-1))
        yy1 = (((H*L)-H)*(A-1)-(B-1)*H)
        xxx1 = (xx1-L*U*0.5*(C-1)+(U/2)*(C-1))
        yyy1 = (yy1-(C-1)*H*(L+1))
        S(L)[x (xxx1-L*U*m+(U/2)*m) y (yyy1-H*m) z (-m+A+B+C+D+1)]
```

```
    }
    loop j=1,B,1 [] {
        S(L)[x ((U*0.5*L)*(A-1)+(U/2)*(A-1)+(L*1+U/2)*j) y (((H*L)-H)*(A-1)-j*H)]
    }
    loop k=1,C,1 [] {
        xx1 = ((U*0.5*L)*(A-1)+(U/2)*(A-1)+(L*1+U/2)*(B-1))
        yy1 = (((H*L)-H)*(A-1)-(B-1)*H)
        S(L)[x (xx1-L*U*0.5*k+(U/2)*k) y (yy1-k*H*(L+1)) z (A+B+C+D+2)]
    }
    inner_wall2[]
    front_wall[z (A+B+C+D+3)]
    mid_wall2[]
    right_wall[]
    loop n=2,B,1 [] {
        px = ((U*0.5*L)*(A-1)+(U/2)*(A-1)) + L*U*n + (n-1)*(U/2)
        py = (((H*L)-H)*(A-1))-H*(n-1)
        stair_rect_C[x px y py]
    }
    loop o=1,C-1,1 [] {
        px = ((U*0.5*L)*(A-1)+(U/2)*(A-1)) + L*U*B + (B-1)*(U/2) - L*U
        py = (((H*L)-H)*(A-1))-H*(B-1)
        stair_rect_D[x (px-L*U*0.5*o+(U/2)*o) y (py-(L+1)*H*o)]
    }
}
path stair_rect_D {
    MOVETO(0,0)
    LINETO(U*L,0)
    LINEREL(U/2,-H)
    LINEREL(-U*L,0)
    CLOSEPOLY()
    FILL()[trans COLOR1]
    STROKE(0.1, CF::RoundJoin)[]
}
path stair_rect_C {
    MOVETO(0,0)
    LINETO(U*L*0.5,H*L)
    LINEREL(U/2,-H)
    LINEREL(-U*L*0.5,-H*L)
    CLOSEPOLY()
    FILL()[trans COLOR2]
    STROKE(0.1, CF::RoundJoin)[]
}
path stairlinesC {
    MOVETO(0,0)
    MOVEREL(L*U*0.5+U*2*A+L+(L/3)+U/2,H*L+H*2*(A-1)-H)
    loop i=0,(B-2),1 [] {
        MOVEREL(U/2,-H)
        MOVEREL(-L*U*0.5,-H*3)
```

```
        MOVEREL(-U/2,H)
        MOVEREL(L*U*0.5,H*L)
        MOVEREL(U*0.5+L,-H)
    }
    MOVEREL(-L-U*1.5,-H*L)
    MOVEREL(U/2,-H)
    MOVEREL(-U*0.5*L,-H*L)
    loop j=0,(C-2),1 [] {
        LINEREL(U*0.5,-H)
        MOVEREL(-L*U*0.5,-H*L)
    }
    FILL()[b 1]
    STROKE(0.1, CF::RoundJoin)[]
}
path stairlinesB {
    MOVETO(0,0)
    MOVEREL(L*U*0.5+U*2*A+L+(L/3)+U/2,H*L+H*2*(A-1)-H)
    loop i=0,(B-2),1 [] {
        LINEREL(U/2,-H)
        MOVEREL(-L*U*0.5,-H*3)
        LINEREL(-U/2,H)
        MOVEREL(L*U*0.5,H*L)
        MOVEREL(U*0.5+L,-H)
    }
    FILL()[b 1]
    STROKE(0.1, CF::RoundJoin)[]
}
path mid_wall2 {
    MOVEREL(L*U+L*U*0.5+U/2,H*L-H)
    loop n=1,A,1 [] {
        LINEREL(L*U*0.5,H*L)
        LINEREL(U*0.5,-H)
    }
    LINEREL(-L*0.5*U,-H*L)
    LINEREL(U*WH*0.5,-H*WH)
    LINEREL(-U*0.5*L*(C-2),-H*L*(C-2))
    CLOSEPOLY()
    FILL()[trans COLOR2]
    STROKE(0.1, CF::RoundJoin)[]
}
path inner_wall2 {
    MOVEREL(L*U+L*U*0.5+U/2,H*L-H)
    loop n=1,A,1 [] {
        MOVEREL(L*U*0.5,H*L)
        MOVEREL(U*0.5,-H)
    }
    MOVEREL(-L*0.5*U,-H*L)
    loop m=1,B-1,1 [] {
```

```
        LINEREL(L*U,0)
        LINEREL(U*0.5,-H)
    }
    LINEREL(L,0)
    LINEREL(U*0.5,-H)
    LINEREL(-L,0)
    LINEREL(U*WH*0.5-(B*U*0.5),-H*WH+H*B)
    LINEREL(-L*U*(B-2)+U*0.5,-H)
    CLOSEPOLY()
    FILL()[trans COLOR1]
    STROKE(0.1, CF::RoundJoin)[]
}
path right_wall {
    MOVETO(0,0)
    loop i=1,(D),1 [] {
        MOVEREL(L,0)
        MOVEREL(-U*0.5,H)
    }
    MOVEREL(L,0)
    loop i=1,C,1 [] {
        LINEREL(L*U*0.5,H*L)
        LINEREL(-U*0.5,H)
    }
    LINEREL(L*U*0.5,H*L)
    loop j=1,C,1 [] {
        LINEREL(U*0.5,-H)
    }
    LINEREL(U*WH*0.5,-H*WH)
    LINEREL(-U*0.5*L*C,-H*L*C)
    CLOSEPOLY()
    FILL()[trans COLOR2]
    STROKE(CF::RoundJoin)[]
}
path front_wall {
    MOVETO(0,0)
    loop i=1,(D),1 [] {
        LINEREL(L,0)
        LINEREL(-U*0.5,H)
    }
    LINEREL(L,0)
    LINEREL(U*WH*0.5,-H*WH)
    LINEREL(-U*D*L,0)
    CLOSEPOLY()
    FILL()[trans COLOR1]
    STROKE(CF::RoundJoin)[]
}
path S(1) {
    MOVETO(0,0)
```

```
    LINETO(U*0.5*l,H*l)
    LINETO(U*0.5*l+U*l,H*l)
    LINETO(U*l,0)
    CLOSEPOLY()
    FILL()[trans COLOR3]
    STROKE(CF::RoundJoin)[]
}
```