Experiments with integer vectors

The algorithm described in [1], [2] and [3] is originally intended for integer vectors. Therefore, the programming language Python is well suited to represent large integers. In the following two Python source codes „EightPoints03.txt“ and „EightPoints03-graph.txt“ an example is discribed. The algorithm is derived exclusively from position and displacement vectors. In this example eight points are pairwise coupled so that four bivectors emerge. Within the bivectors the points are displaced by the step width h. The bivectors are coupled by charges. For the displacement step width pull caused by the charges shall apply

$$1 << \text{pull} << h.$$

This is described by the reciprocal coupling factor reci:

$$h = \text{reci} * \text{pull}.$$

If one chooses reci close to 1, the coupling is so strong that chaos can be caused. If reci >> 100 is chosen, the coupling between the bivectors is very weak, so that the interactions are very slow.

Python source code EightPoints03.txt without graphic.

```
"""
EightPoints03
Experiments with integer vectors
Four bivectors coupled by charges

@author: Wolfhard Hoevel

https://opus4.kobv.de/opus4-ohm/frontdoor/index/index/docId/34
https://opus4.kobv.de/opus4-ohm/frontdoor/index/index/docId/126
https://opus4.kobv.de/opus4-ohm/frontdoor/index/index/docId/253

"""

import random


def iSqrt(n): # Newton
    if n < 0:
        n = 0
    x = n
    y = (x + 1) // 2
    while y < x:
        x = y
        y = (x + n // x) // 2
    return x

def randomVector(factor, dim):
    temp = []
    i = 0
    while i < dim:
        temp.append(random.randint(-factor,factor))
        i += 1
    return temp

def multiplay(factor, vect, dim):
    temp = []
    i = 0
    while i < dim:
        temp.append(factor*vect[i])
        i += 1
    return temp

def add(vecta, vectb, dim):
    temp = []
    i = 0
    while i < dim:
        temp.append(vecta[i] + vectb[i])
        i += 1
    return temp

def subtract(vecta, vectb, dim):
    temp = []
    i = 0
    while i < dim:
        temp.append(vecta[i] - vectb[i])
        i += 1
    return temp

def dotProduct(vecta, vectb, dim):
    c = 0
    i = 0
    while i < dim:
        c = c + vecta[i] * vectb[i]
        i += 1
    return c

def magnitude(vecta, dim):
    return iSqrt( dotProduct(vecta, vecta, dim))

def hVector(vecta, h, dim):
    temp = []
    r = magnitude(vecta, dim)
    if r == 0: r = 1
    i = 0
    while i < dim:
        temp.append((h*vecta[i])//r)
        i += 1
    return temp

def changeLength(vecta, z, n, dim): # first multiply by z, then divide by n
```

```
    temp = []
    if n == 0: n = 1
    i = 0
    while i < dim:
        temp.append((z*vecta[i])//n)
        i += 1
    return temp

def drCharge(vecta, pull, h, qa, qb, dim):
    temp = []
    r = magnitude(vecta, dim)
    if r == 0: r = 1
    i = 0
    while i < dim:
        temp.append((-pull*qa*qb*vecta[i])//r)
        i += 1
    return temp

def reflect(rVector, draVector, h, s, dim):
    rq = dotProduct(rVector, rVector, dim)
    if rq < 1: rq = 1
    r = iSqrt(rq)
    k = dotProduct(draVector, rVector, dim)
    radi = ((rq * h * h)//(s * s) - h*h - (k * k)//(s *s) + (k * k)//rq)
    x = (-k)//r + iSqrt(radi)
    xVector = changeLength(rVector, x, r, dim)
    yVector = add(draVector, xVector, dim)
    drb = hVector(yVector, h, dim)
    return drb

pull = 100    # magnitude of the radial displacements by charges
reci = 100     # reciprocal coupling factor
h = reci*pull   # step width inside bivectors

dim = 4    # dimension of Euclidean space

s01 = 5*h    # spin of particle01 s01 <= e01
e01 = 5*h    # extension of particle01

s23 = 10*h    # spin of particle23 s23 <= e23
e23 = 10*h    # extension of particle23

s45 = 10*h    # spin of particle45 s45 <= e45
e45 = 10*h    # extension of particle45

s67 = 10*h    # spin of particle67 s67 <= e67
e67 = 10*h    # extension of particle67

q0 = 1       # charge of point r0
q1 = 1       # charge of point r1

q2 = 1       # charge of point r2
```

```
q3 = -1       # charge of point r3

q4 = -1       # charge of point r4
q5 = 0       # charge of point r5

q6 = -1       # charge of point r6
q7 = 0       # charge of point r7

a = 20        # initial condition factor
nFrame = 180   # nFrame loops

t = 0  # time

# initial conditions, points
r0 = randomVector(a*h, dim)
r1 = randomVector(a*h, dim)
r2 = randomVector(a*h, dim)
r3 = randomVector(a*h, dim)
r4 = randomVector(a*h, dim)
r5 = randomVector(a*h, dim)
r6 = randomVector(a*h, dim)
r7 = randomVector(a*h, dim)
# initial conditions, displacement vectors
dr01 = randomVector(a*h, dim)
dr23 = randomVector(a*h, dim)
dr45 = randomVector(a*h, dim)
dr67 = randomVector(a*h, dim)


# calculation for one step t --> t+1
for n in range(0, nFrame, 1):

   # distance vectors
   r01 = subtract(r1, r0, dim)
   r02 = subtract(r2, r0, dim)
   r12 = subtract(r2, r1, dim)
   r03 = subtract(r3, r0, dim)
   r13 = subtract(r3, r1, dim)
   r23 = subtract(r3, r2, dim)
   r04 = subtract(r4, r0, dim)
   r14 = subtract(r4, r1, dim)
   r24 = subtract(r4, r2, dim)
   r34 = subtract(r4, r3, dim)
   r05 = subtract(r5, r0, dim)
   r15 = subtract(r5, r1, dim)
   r25 = subtract(r5, r2, dim)
   r35 = subtract(r5, r3, dim)
   r45 = subtract(r5, r4, dim)
   r06 = subtract(r6, r0, dim)
   r16 = subtract(r6, r1, dim)
   r26 = subtract(r6, r2, dim)
   r36 = subtract(r6, r3, dim)
```

```
r46 = subtract(r6, r4, dim)
r56 = subtract(r6, r5, dim)
r07 = subtract(r7, r0, dim)
r17 = subtract(r7, r1, dim)
r27 = subtract(r7, r2, dim)
r37 = subtract(r7, r3, dim)
r47 = subtract(r7, r4, dim)
r57 = subtract(r7, r5, dim)
r67 = subtract(r7, r6, dim)

# displacment vectors by charges
if( q0*q1 != 0 ): drC01 = drCharge(r01, pull, h, q0, q1, dim)
if( q0*q2 != 0 ): drC02 = drCharge(r02, pull, h, q0, q2, dim)
if( q1*q2 != 0 ): drC12 = drCharge(r12, pull, h, q1, q2, dim)
if( q0*q3 != 0 ): drC03 = drCharge(r03, pull, h, q0, q3, dim)
if( q1*q3 != 0 ): drC13 = drCharge(r13, pull, h, q1, q3, dim)
if( q2*q3 != 0 ): drC23 = drCharge(r23, pull, h, q2, q3, dim)
if( q0*q4 != 0 ): drC04 = drCharge(r04, pull, h, q0, q4, dim)
if( q1*q4 != 0 ): drC14 = drCharge(r14, pull, h, q1, q4, dim)
if( q2*q4 != 0 ): drC24 = drCharge(r24, pull, h, q2, q4, dim)
if( q3*q4 != 0 ): drC34 = drCharge(r34, pull, h, q3, q4, dim)
if( q0*q5 != 0 ): drC05 = drCharge(r05, pull, h, q0, q5, dim)
if( q1*q5 != 0 ): drC15 = drCharge(r15, pull, h, q1, q5, dim)
if( q2*q5 != 0 ): drC25 = drCharge(r25, pull, h, q2, q5, dim)
if( q3*q5 != 0 ): drC35 = drCharge(r35, pull, h, q3, q5, dim)
if( q4*q5 != 0 ): drC45 = drCharge(r45, pull, h, q4, q5, dim)
if( q0*q6 != 0 ): drC06 = drCharge(r06, pull, h, q0, q6, dim)
if( q1*q6 != 0 ): drC16 = drCharge(r16, pull, h, q1, q6, dim)
if( q2*q6 != 0 ): drC26 = drCharge(r26, pull, h, q2, q6, dim)
if( q3*q6 != 0 ): drC36 = drCharge(r36, pull, h, q3, q6, dim)
if( q4*q6 != 0 ): drC46 = drCharge(r46, pull, h, q4, q6, dim)
if( q5*q6 != 0 ): drC56 = drCharge(r56, pull, h, q5, q6, dim)
if( q0*q7 != 0 ): drC07 = drCharge(r07, pull, h, q0, q7, dim)
if( q1*q7 != 0 ): drC17 = drCharge(r17, pull, h, q1, q7, dim)
if( q2*q7 != 0 ): drC27 = drCharge(r27, pull, h, q2, q7, dim)
if( q3*q7 != 0 ): drC37 = drCharge(r37, pull, h, q3, q7, dim)
if( q4*q7 != 0 ): drC47 = drCharge(r47, pull, h, q4, q7, dim)
if( q5*q7 != 0 ): drC57 = drCharge(r57, pull, h, q5, q7, dim)
if( q6*q7 != 0 ): drC67 = drCharge(r67, pull, h, q6, q7, dim)

# reflection inside the four bivectors
if (magnitude(r01, dim) > e01 ): dr01 = reflect(r01, dr01, h, s01, dim)
if (magnitude(r23, dim) > e23 ): dr23 = reflect(r23, dr23, h, s23, dim)
if (magnitude(r45, dim) > e45 ): dr45 = reflect(r45, dr45, h, s45, dim)
if (magnitude(r67, dim) > e67 ): dr67 = reflect(r67, dr67, h, s67, dim)

# displacement by reflect().
r0 = add(r0, dr01, dim)
r1 = subtract(r1, dr01, dim)
r2 = add(r2, dr23, dim)
r3 = subtract(r3, dr23, dim)
r4 = add(r4, dr45, dim)
```

```
r5 = subtract(r5, dr45, dim)
r6 = add(r6, dr67, dim)
r7 = subtract(r7, dr67, dim)


# superposition of the displacement vectors influenced by charges
if( q0*q1 != 0 ):
   r0 = add(r0, drC01, dim)
   r1 = subtract(r1, drC01, dim)

if( q0*q2 != 0 ):
   r0 = add(r0, drC02, dim)
   r2 = subtract(r2, drC02, dim)

if( q1*q2 != 0 ):
   r1 = add(r1, drC12, dim)
   r2 = subtract(r2, drC12, dim)

if( q0*q3 != 0 ):
   r0 = add(r0, drC03, dim)
   r3 = subtract(r3, drC03, dim)

if( q1*q3 != 0 ):
   r1 = add(r1, drC13, dim)
   r3 = subtract(r3, drC13, dim)

if( q2*q3 != 0 ):
   r2 = add(r2, drC23, dim)
   r3 = subtract(r3, drC23, dim)

if( q0*q4 != 0 ):
   r0 = add(r0, drC04, dim)
   r4 = subtract(r4, drC04, dim)

if( q1*q4 != 0 ):
   r1 = add(r1, drC14, dim)
   r4 = subtract(r4, drC14, dim)

if( q2*q4 != 0 ):
   r2 = add(r2, drC24, dim)
   r4 = subtract(r4, drC24, dim)

if( q3*q4 != 0 ):
   r3 = add(r3, drC34, dim)
   r4 = subtract(r4, drC34, dim)

if( q0*q5 != 0 ):
   r0 = add(r0, drC05, dim)
   r5 = subtract(r5, drC05, dim)

if( q1*q5 != 0 ):
   r1 = add(r1, drC15, dim)
```

```
    r5 = subtract(r5, drC15, dim)

if( q2*q5 != 0 ):
    r2 = add(r2, drC25, dim)
    r5 = subtract(r5, drC25, dim)

if( q3*q5 != 0 ):
    r3 = add(r3, drC35, dim)
    r5 = subtract(r5, drC35, dim)

if( q4*q5 != 0 ):
    r4 = add(r4, drC45, dim)
    r5 = subtract(r5, drC45, dim)

if( q0*q6 != 0 ):
    r0 = add(r0, drC06, dim)
    r6 = subtract(r6, drC06, dim)

if( q1*q6 != 0 ):
    r1 = add(r1, drC16, dim)
    r6 = subtract(r6, drC16, dim)

if( q2*q6 != 0 ):
    r2 = add(r2, drC26, dim)
    r6 = subtract(r6, drC26, dim)

if( q3*q6 != 0 ):
    r3 = add(r3, drC36, dim)
    r6 = subtract(r6, drC36, dim)

if( q4*q6 != 0 ):
    r4 = add(r4, drC46, dim)
    r6 = subtract(r6, drC46, dim)

if( q5*q6 != 0 ):
    r5 = add(r5, drC56, dim)
    r6 = subtract(r6, drC56, dim)

if( q0*q7 != 0 ):
    r0 = add(r0, drC07, dim)
    r7 = subtract(r7, drC07, dim)

if( q1*q7 != 0 ):
    r1 = add(r1, drC17, dim)
    r7 = subtract(r7, drC17, dim)

if( q2*q7 != 0 ):
    r2 = add(r2, drC27, dim)
    r7 = subtract(r7, drC27, dim)

if( q3*q7 != 0 ):
    r3 = add(r3, drC37, dim)
```

```
        r7 = subtract(r7, drC37, dim)

    if( q4*q7 != 0 ):
        r4 = add(r4, drC47, dim)
        r7 = subtract(r7, drC47, dim)

    if( q5*q7 != 0 ):
        r5 = add(r5, drC57, dim)
        r7 = subtract(r7, drC57, dim)

    if( q6*q7 != 0 ):
        r6 = add(r6, drC67, dim)
        r7 = subtract(r7, drC67, dim)

    # Testing variables
    print(' ')
    print('t = ',t)
    # Position of the points
    print('r0 = ',r0)
    print('r1 = ',r1)
    print('r2 = ',r2)
    print('r3 = ',r3)
    print('r4 = ',r4)
    print('r5 = ',r5)
    print('r6 = ',r6)
    print('r7 = ',r7)

    t += 1
```

Python source code EightPoints03-graph.txt with graphic.

```
"""
EightPoints03
Experiments with integer vectors
Four bivectors coupled by charges

@author: Wolfhard Hoevel

https://opus4.kobv.de/opus4-ohm/frontdoor/index/index/docId/34
https://opus4.kobv.de/opus4-ohm/frontdoor/index/index/docId/126
https://opus4.kobv.de/opus4-ohm/frontdoor/index/index/docId/253
"""

import tkinter as tk
import random

class App(tk.Tk):
    def __init__(self):
        tk.Tk.__init__(self)
        self.title("EightPoints03")

        self.c = tk.Canvas(self, bg="black", width=850, height=870)
        self.c.pack()

# Parameters------------------------------------------------------------------

        self.pull = 100    # magnitude of the radial displacements by charges
        self.reci = 100    # reciprocal coupling factor
        self.h = self.reci*self.pull   # step width inside bivectors

        self.dim = 4    # dimension of Euclidean space

        self.s01 = 5 * self.h    # spin of particle01 s01 <= e01
        self.e01 = 5 * self.h    # extension of particle01

        self.s23 = 10 * self.h    # spin of particle23 s23 <= e23
        self.e23 = 10 * self.h    # extension of particle23

        self.s45 = 10 * self.h    # spin of particle45 s45 <= e45
        self.e45 = 10 * self.h    # extension of particle45

        self.s67 = 10 * self.h    # spin of particle67 s67 <= e67
        self.e67 = 10 * self.h    # extension of particle67

        self.q0 = 1     # charge of point r0
        self.q1 = 1     # charge of point r1

        self.q2 = 1     # charge of point r2
        self.q3 = -1     # charge of point r3

        self.q4 = -1     # charge of point r4
        self.q5 = 0     # charge of point r5

        self.q6 = -1     # charge of point r6
        self.q7 = 0     # charge of point r7


        self.zoom =20
        self.a = 20    # initial condition factor

        self.t = 0    # time
        self.tFrame = 0
        self.tFrameMax = 180  #  8 * points / frame

#---------------------------------------------------------------------------


        def randomVector(factor, dim):
            temp = []
            i = 0
            while i < dim:
```

```
            temp.append(random.randint(-factor,factor))
            i += 1
        return temp

    # initial conditions, points
    self.r0 = randomVector(self.a*self.h, self.dim)
    self.r1 = randomVector(self.a*self.h, self.dim)
    self.r2 = randomVector(self.a*self.h, self.dim)
    self.r3 = randomVector(self.a*self.h, self.dim)
    self.r4 = randomVector(self.a*self.h, self.dim)
    self.r5 = randomVector(self.a*self.h, self.dim)
    self.r6 = randomVector(self.a*self.h, self.dim)
    self.r7 = randomVector(self.a*self.h, self.dim)
    # initial conditions, displacement vectors
    self.dr01 = randomVector(self.a*self.h, self.dim)
    self.dr23 = randomVector(self.a*self.h, self.dim)
    self.dr45 = randomVector(self.a*self.h, self.dim)
    self.dr67 = randomVector(self.a*self.h, self.dim)



    self.pixels = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]



def calculatePixels(self):

    def iSqrt(n): # Newton
        if n < 0:
            n = 0
        x = n
        y = (x + 1) // 2
        while y < x:
            x = y
            y = (x + n // x) // 2
        return x

    def multiplay(factor, vect, dim):
        temp = []
        i = 0
        while i < dim:
            temp.append(factor*vect[i])
            i += 1
        return temp

    def add(vecta, vectb, dim):
        temp = []
        i = 0
        while i < dim:
            temp.append(vecta[i] + vectb[i])
            i += 1
        return temp
```

```
def subtract(vecta, vectb, dim):
    temp = []
    i = 0
    while i < dim:
        temp.append(vecta[i] - vectb[i])
        i += 1
    return temp

def dotProduct(vecta, vectb, dim):
    c = 0
    i = 0
    while i < dim:
        c = c + vecta[i] * vectb[i]
        i += 1
    return c

def magnitude(vecta, dim):
    return iSqrt( dotProduct(vecta, vecta, dim))

def hVector(vecta, h, dim):
    temp = []
    r = magnitude(vecta, dim)
    if r == 0: r = 1
    i = 0
    while i < dim:
        temp.append((h*vecta[i])//r)
        i += 1
    return temp

def changeLength(vecta, z, n, dim): # first multiply by z, then divide by n
    temp = []
    if n == 0: n = 1
    i = 0
    while i < dim:
        temp.append((z*vecta[i])//n)
        i += 1
    return temp

def drCharge(vecta, pull, h, qa, qb, dim):
    temp = []
    r = magnitude(vecta, dim)
    if r == 0: r = 1
    i = 0
    while i < dim:
        temp.append((-pull*qa*qb*vecta[i])//r)
        i += 1
    return temp

def reflect(rVector, draVector, h, s, dim):
    rq = dotProduct(rVector, rVector, dim)
    if rq < 1: rq = 1
```

```
            r = iSqrt(rq)
            k = dotProduct(draVector, rVector, dim)
            radi = ((rq * h * h)//(s * s) - h*h - (k * k)//(s *s) + (k * k)//rq)
            x = (-k)//r + iSqrt(radi)
            xVector = changeLength(rVector, x, r, dim)
            yVector = add(draVector, xVector, dim)
            drb = hVector(yVector, h, dim)
            return drb


        # distance vectors
        self.r01 = subtract(self.r1, self.r0, self.dim)
        self.r02 = subtract(self.r2, self.r0, self.dim)
        self.r12 = subtract(self.r2, self.r1, self.dim)
        self.r03 = subtract(self.r3, self.r0, self.dim)
        self.r13 = subtract(self.r3, self.r1, self.dim)
        self.r23 = subtract(self.r3, self.r2, self.dim)
        self.r04 = subtract(self.r4, self.r0, self.dim)
        self.r14 = subtract(self.r4, self.r1, self.dim)
        self.r24 = subtract(self.r4, self.r2, self.dim)
        self.r34 = subtract(self.r4, self.r3, self.dim)
        self.r05 = subtract(self.r5, self.r0, self.dim)
        self.r15 = subtract(self.r5, self.r1, self.dim)
        self.r25 = subtract(self.r5, self.r2, self.dim)
        self.r35 = subtract(self.r5, self.r3, self.dim)
        self.r45 = subtract(self.r5, self.r4, self.dim)
        self.r06 = subtract(self.r6, self.r0, self.dim)
        self.r16 = subtract(self.r6, self.r1, self.dim)
        self.r26 = subtract(self.r6, self.r2, self.dim)
        self.r36 = subtract(self.r6, self.r3, self.dim)
        self.r46 = subtract(self.r6, self.r4, self.dim)
        self.r56 = subtract(self.r6, self.r5, self.dim)
        self.r07 = subtract(self.r7, self.r0, self.dim)
        self.r17 = subtract(self.r7, self.r1, self.dim)
        self.r27 = subtract(self.r7, self.r2, self.dim)
        self.r37 = subtract(self.r7, self.r3, self.dim)
        self.r47 = subtract(self.r7, self.r4, self.dim)
        self.r57 = subtract(self.r7, self.r5, self.dim)
        self.r67 = subtract(self.r7, self.r6, self.dim)

        # displacment vectors by charges
        if( self.q0*self.q1 != 0 ): self.drC01 = drCharge(self.r01, self.pull, self.h, self.q0, self.q1,
self.dim)
        if( self.q0*self.q2 != 0 ): self.drC02 = drCharge(self.r02, self.pull, self.h, self.q0, self.q2,
self.dim)
        if( self.q1*self.q2 != 0 ): self.drC12 = drCharge(self.r12, self.pull, self.h, self.q1, self.q2,
self.dim)
        if( self.q0*self.q3 != 0 ): self.drC03 = drCharge(self.r03, self.pull, self.h, self.q0, self.q3,
self.dim)
        if( self.q1*self.q3 != 0 ): self.drC13 = drCharge(self.r13, self.pull, self.h, self.q1, self.q3,
self.dim)
        if( self.q2*self.q3 != 0 ): self.drC23 = drCharge(self.r23, self.pull, self.h, self.q2, self.q3,
self.dim)
        if( self.q0*self.q4 != 0 ): self.drC04 = drCharge(self.r04, self.pull, self.h, self.q0, self.q4,
self.dim)
        if( self.q1*self.q4 != 0 ): self.drC14 = drCharge(self.r14, self.pull, self.h, self.q1, self.q4,
self.dim)
        if( self.q2*self.q4 != 0 ): self.drC24 = drCharge(self.r24, self.pull, self.h, self.q2, self.q4,
self.dim)
        if( self.q3*self.q4 != 0 ): self.drC34 = drCharge(self.r34, self.pull, self.h, self.q3, self.q4,
self.dim)
        if( self.q0*self.q5 != 0 ): self.drC05 = drCharge(self.r05, self.pull, self.h, self.q0, self.q5,
self.dim)
        if( self.q1*self.q5 != 0 ): self.drC15 = drCharge(self.r15, self.pull, self.h, self.q1, self.q5,
self.dim)
        if( self.q2*self.q5 != 0 ): self.drC25 = drCharge(self.r25, self.pull, self.h, self.q2, self.q5,
self.dim)
        if( self.q3*self.q5 != 0 ): self.drC35 = drCharge(self.r35, self.pull, self.h, self.q3, self.q5,
self.dim)
        if( self.q4*self.q5 != 0 ): self.drC45 = drCharge(self.r45, self.pull, self.h, self.q4, self.q5,
self.dim)
        if( self.q0*self.q6 != 0 ): self.drC06 = drCharge(self.r06, self.pull, self.h, self.q0, self.q6,
self.dim)
        if( self.q1*self.q6 != 0 ): self.drC16 = drCharge(self.r16, self.pull, self.h, self.q1, self.q6,
self.dim)
        if( self.q2*self.q6 != 0 ): self.drC26 = drCharge(self.r26, self.pull, self.h, self.q2, self.q6,
self.dim)
        if( self.q3*self.q6 != 0 ): self.drC36 = drCharge(self.r36, self.pull, self.h, self.q3, self.q6,
self.dim)
        if( self.q4*self.q6 != 0 ): self.drC46 = drCharge(self.r46, self.pull, self.h, self.q4, self.q6,
self.dim)
        if( self.q5*self.q6 != 0 ): self.drC56 = drCharge(self.r56, self.pull, self.h, self.q5, self.q6,
self.dim)
        if( self.q0*self.q7 != 0 ): self.drC07 = drCharge(self.r07, self.pull, self.h, self.q0, self.q7,
self.dim)
        if( self.q1*self.q7 != 0 ): self.drC17 = drCharge(self.r17, self.pull, self.h, self.q1, self.q7,
self.dim)
        if( self.q2*self.q7 != 0 ): self.drC27 = drCharge(self.r27, self.pull, self.h, self.q2, self.q7,
self.dim)
        if( self.q3*self.q7 != 0 ): self.drC37 = drCharge(self.r37, self.pull, self.h, self.q3, self.q7,
self.dim)
        if( self.q4*self.q7 != 0 ): self.drC47 = drCharge(self.r47, self.pull, self.h, self.q4, self.q7,
self.dim)
        if( self.q5*self.q7 != 0 ): self.drC57 = drCharge(self.r57, self.pull, self.h, self.q5, self.q7,
self.dim)
        if( self.q6*self.q7 != 0 ): self.drC67 = drCharge(self.r67, self.pull, self.h, self.q6, self.q7,
self.dim)

        # reflection inside the bivectors
        if (magnitude(self.r01, self.dim) > self.e01 ): self.dr01 = reflect(self.r01, self.dr01, self.h,
self.s01, self.dim)
        if (magnitude(self.r23, self.dim) > self.e23 ): self.dr23 = reflect(self.r23, self.dr23, self.h,
self.s23, self.dim)
        if (magnitude(self.r45, self.dim) > self.e45 ): self.dr45 = reflect(self.r45, self.dr45, self.h,
self.s45, self.dim)
        if (magnitude(self.r67, self.dim) > self.e67 ): self.dr67 = reflect(self.r67, self.dr67, self.h,
self.s67, self.dim)

        # displacement by reflect().
        self.r0 = add(self.r0, self.dr01, self.dim)
        self.r1 = subtract(self.r1, self.dr01, self.dim)
        self.r2 = add(self.r2, self.dr23, self.dim)
        self.r3 = subtract(self.r3, self.dr23, self.dim)
        self.r4 = add(self.r4, self.dr45, self.dim)
        self.r5 = subtract(self.r5, self.dr45, self.dim)
        self.r6 = add(self.r6, self.dr67, self.dim)
        self.r7 = subtract(self.r7, self.dr67, self.dim)


        # superposition of the displacement vectors influenced by charges
        if( self.q0*self.q1 != 0 ):
            self.r0 = add(self.r0, self.drC01, self.dim)
            self.r1 = subtract(self.r1, self.drC01, self.dim)

        if( self.q0*self.q2 != 0 ):
            self.r0 = add(self.r0, self.drC02, self.dim)
            self.r2 = subtract(self.r2, self.drC02, self.dim)

        if( self.q1*self.q2 != 0 ):
            self.r1 = add(self.r1, self.drC12, self.dim)
            self.r2 = subtract(self.r2, self.drC12, self.dim)

        if( self.q0*self.q3 != 0 ):
            self.r0 = add(self.r0, self.drC03, self.dim)
            self.r3 = subtract(self.r3, self.drC03, self.dim)

        if( self.q1*self.q3 != 0 ):
            self.r1 = add(self.r1, self.drC13, self.dim)
            self.r3 = subtract(self.r3, self.drC13, self.dim)

        if( self.q2*self.q3 != 0 ):
            self.r2 = add(self.r2, self.drC23, self.dim)
            self.r3 = subtract(self.r3, self.drC23, self.dim)

        if( self.q0*self.q4 != 0 ):
            self.r0 = add(self.r0, self.drC04, self.dim)
            self.r4 = subtract(self.r4, self.drC04, self.dim)

        if( self.q1*self.q4 != 0 ):
            self.r1 = add(self.r1, self.drC14, self.dim)
            self.r4 = subtract(self.r4, self.drC14, self.dim)

        if( self.q2*self.q4 != 0 ):
            self.r2 = add(self.r2, self.drC24, self.dim)
            self.r4 = subtract(self.r4, self.drC24, self.dim)
```

```
if( self.q3*self.q4 != 0 ):
    self.r3 = add(self.r3, self.drC34, self.dim)
    self.r4 = subtract(self.r4, self.drC34, self.dim)

if( self.q0*self.q5 != 0 ):
    self.r0 = add(self.r0, self.drC05, self.dim)
    self.r5 = subtract(self.r5, self.drC05, self.dim)

if( self.q1*self.q5 != 0 ):
    self.r1 = add(self.r1, self.drC15, self.dim)
    self.r5 = subtract(self.r5, self.drC15, self.dim)

if( self.q2*self.q5 != 0 ):
    self.r2 = add(self.r2, self.drC25, self.dim)
    self.r5 = subtract(self.r5, self.drC25, self.dim)

if( self.q3*self.q5 != 0 ):
    self.r3 = add(self.r3, self.drC35, self.dim)
    self.r5 = subtract(self.r5, self.drC35, self.dim)

if( self.q4*self.q5 != 0 ):
    self.r4 = add(self.r4, self.drC45, self.dim)
    self.r5 = subtract(self.r5, self.drC45, self.dim)

if( self.q0*self.q6 != 0 ):
    self.r0 = add(self.r0, self.drC06, self.dim)
    self.r6 = subtract(self.r6, self.drC06, self.dim)

if( self.q1*self.q6 != 0 ):
    self.r1 = add(self.r1, self.drC16, self.dim)
    self.r6 = subtract(self.r6, self.drC16, self.dim)

if( self.q2*self.q6 != 0 ):
    self.r2 = add(self.r2, self.drC26, self.dim)
    self.r6 = subtract(self.r6, self.drC26, self.dim)

if( self.q3*self.q6 != 0 ):
    self.r3 = add(self.r3, self.drC36, self.dim)
    self.r6 = subtract(self.r6, self.drC36, self.dim)

if( self.q4*self.q6 != 0 ):
    self.r4 = add(self.r4, self.drC46, self.dim)
    self.r6 = subtract(self.r6, self.drC46, self.dim)

if( self.q5*self.q6 != 0 ):
    self.r5 = add(self.r5, self.drC56, self.dim)
    self.r6 = subtract(self.r6, self.drC56, self.dim)

if( self.q0*self.q7 != 0 ):
    self.r0 = add(self.r0, self.drC07, self.dim)
    self.r7 = subtract(self.r7, self.drC07, self.dim)
```

```
if( self.q1*self.q7 != 0 ):
    self.r1 = add(self.r1, self.drC17, self.dim)
    self.r7 = subtract(self.r7, self.drC17, self.dim)

if( self.q2*self.q7 != 0 ):
    self.r2 = add(self.r2, self.drC27, self.dim)
    self.r7 = subtract(self.r7, self.drC27, self.dim)

if( self.q3*self.q7 != 0 ):
    self.r3 = add(self.r3, self.drC37, self.dim)
    self.r7 = subtract(self.r7, self.drC37, self.dim)

if( self.q4*self.q7 != 0 ):
    self.r4 = add(self.r4, self.drC47, self.dim)
    self.r7 = subtract(self.r7, self.drC47, self.dim)

if( self.q5*self.q7 != 0 ):
    self.r5 = add(self.r5, self.drC57, self.dim)
    self.r7 = subtract(self.r7, self.drC57, self.dim)

if( self.q6*self.q7 != 0 ):
    self.r6 = add(self.r6, self.drC67, self.dim)
    self.r7 = subtract(self.r7, self.drC67, self.dim)

# Output to the graphic
self.pixels[0] = 425 + (self.zoom*self.r0[0])/self.h
self.pixels[1] = 425 + (self.zoom*self.r0[1])/self.h

self.pixels[2] = 425 + (self.zoom*self.r1[0])/self.h
self.pixels[3] = 425 + (self.zoom*self.r1[1])/self.h

self.pixels[4] = 425 + (self.zoom*self.r2[0])/self.h
self.pixels[5] = 425 + (self.zoom*self.r2[1])/self.h

self.pixels[6] = 425 + (self.zoom*self.r3[0])/self.h
self.pixels[7] = 425 + (self.zoom*self.r3[1])/self.h

self.pixels[8] = 425 + (self.zoom*self.r4[0])/self.h
self.pixels[9] = 425 + (self.zoom*self.r4[1])/self.h

self.pixels[10] = 425 + (self.zoom*self.r5[0])/self.h
self.pixels[11] = 425 + (self.zoom*self.r5[1])/self.h

self.pixels[12] = 425 + (self.zoom*self.r6[0])/self.h
self.pixels[13] = 425 + (self.zoom*self.r6[1])/self.h

self.pixels[14] = 425 + (self.zoom*self.r7[0])/self.h
self.pixels[15] = 425 + (self.zoom*self.r7[1])/self.h

return self.pixels
```

```
    def next_frame(self):

        if self.tFrame > self.tFrameMax:
            self.c.delete('all') # clear canvas
            self.tFrame = 0

        # plot pixel
        pixel = self.calculatePixels()
        self.c.create_line(pixel[0], pixel[1], pixel[0]+1, pixel[1], fill='white')
        self.c.create_line(pixel[2], pixel[3], pixel[2]+1, pixel[3], fill='green')
        self.c.create_line(pixel[4], pixel[5], pixel[4]+1, pixel[5], fill='cadetblue1')
        self.c.create_line(pixel[6], pixel[7], pixel[6]+1, pixel[7], fill='yellow')
        self.c.create_line(pixel[8], pixel[9], pixel[8]+1, pixel[9], fill='red')
        self.c.create_line(pixel[10], pixel[11], pixel[10]+1, pixel[11], fill='pink')
        self.c.create_line(pixel[12], pixel[13], pixel[12]+1, pixel[13], fill='cyan2')
        self.c.create_line(pixel[14], pixel[15], pixel[14]+1, pixel[15], fill='tomato')

        self.t += 1
        self.tFrame += 1

        self.c.after(1, self.next_frame) # call after 1 ms

if __name__ == "__main__":
    app = App()
    app.next_frame()
    app.mainloop()
```

----

[1] https://opus4.kobv.de/opus4-ohm/frontdoor/index/index/docId/34
[2] https://opus4.kobv.de/opus4-ohm/frontdoor/index/index/docId/126
[3] https://opus4.kobv.de/opus4-ohm/frontdoor/index/index/docId/253