Instituto Politécnico Nacional
Escuela Superior de Cómputo
Evolutionary Computing
7 Particle Systems
Martínez Coronel Brayan Yosafat
Rosas Trigueros Jorge Luis
22/10/2021
5/11/2021

Theorical framework

In this practice two problems about particle systems are reviewed, both talk about interactions between particles, one about physics interactions and the other size interaction. Thus, let us define some concepts, for example, particle systems and interactions.

One important thing about particle systems should be considered into trying to define it, and it is the fact that is considered a technique in Computer graphics, even if we do not talk about computers graphics, we use it, not with the same intentions, we just try to look at how particles interact. With that settled, we could borrow the definition from Computer graphics:

Particle systems are a graphical technique that simulates complex physically based effects. Particle systems are collections of small images that when viewed together form a more complex "fuzzy" object, such as fire, smoke, or weather. These complex effects are controlled by specifying the behavior of individual particles using properties such as initial position, velocity, and lifespan [1].

Now, in our situation, we just assign initial position and velocity in the second problem, and we define a size interaction with contiguous neighbors in the first problem. But, before we move onto that, we define an interaction as:

An occasion when two or more people or things communicate with or react to each other [2]. Now we understand some basics about tis practice, but, about the second problem, we use a pair of interactions borrow from classic physics about particle interactions in a 2D space, we calculate the force applied in a particular particle and repeat for each particle in the system, then we iterate over a discrete time, which means the object fields had been modified.

We repeat this process until we reach a certain number of iterations, for example, in the second problem it is set to 2000, but in the first, 300.

Material and equipment

In this practice, Google Colab with Jupiter Notebooks, Python were used for development.

Practice development

Neighbor interactions: size

Code given by professor show us how particle interacts over the y dimension, if the neighbor is at a greater y position, then its neighbor will increase its y position, except at the end of that system. Here is a picture of the result from that code at frame 231.

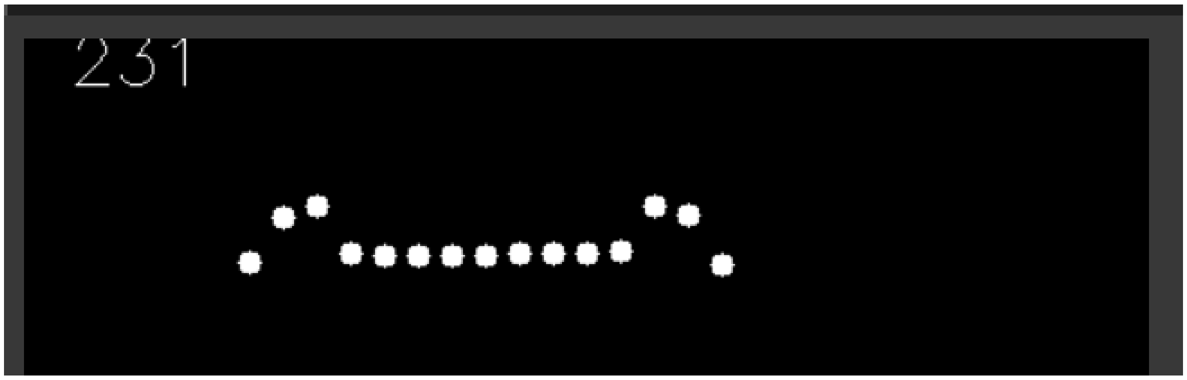

Fig. 1 Neighbor interactions in y dimension

When they get to certain value at y position, then it stops to increase its y value, and decrease over the discrete time. When a particle reaches 0 at y, again has the possibility to increase it y value. Something similar happens in our code. Rather than interacting on y dimension, we code it on size dimension, the reason for this is that we require a 2D system and interacting on y would have been more difficult.

```
import cv2
import numpy as np
from IPython import display as display

import ipywidgets as ipw
import PIL
from io import BytesIO
```

Code 1. Imports for both problems

```
NUM_PARTICULAS = 20
MAX_X = 450
MAX_Y = 450
X0 = 50
Y0 = 50

class System:
    def __init__(self):
        self.x = np.linspace(X0, MAX_X, NUM_PARTICULAS)
        self.y = np.linspace(Y0, MAX_Y, NUM_PARTICULAS)
        self.r = np.ones((NUM_PARTICULAS, NUM_PARTICULAS))
        self.canIncrease = np.ones((NUM_PARTICULAS, NUM_PARTICULAS))
        self.img = np.zeros((500, 500, 3), dtype="uint8")

        self.wIm = ipw.Image()
        display.display(self.wIm)

        self.r[np.random.randint(NUM_PARTICULAS), np.random.randint(NU
M_PARTICULAS)] = 3
        self.frame = 0
        self.graficar()

    def actualizar_particulas(self):
        for i in range(NUM_PARTICULAS-1):
            for k in range(NUM_PARTICULAS-1):
                if 0<i<NUM_PARTICULAS and 0<k<NUM_PARTICULAS and self.
canIncrease[i, k]==1:
```

```
                    self.r[i, k] += 0.05*self.r[i-
1, k] + 0.05*self.r[i+1, k]
                    self.r[i, k] += 0.05*self.r[i, k-
1] + 0.05*self.r[i, k+1]

                if self.canIncrease[i, k]==1 and self.r[i, k] >= 8:
                    self.canIncrease[i, k] = 0
                if self.canIncrease[i, k]==0 and self.r[i, k] <= 3:
                    self.canIncrease[i, k] = 1

                #decay
                self.r[i, k] *= 0.9


    def graficar(self):
        r=1
        color = (255,255,255)

        for i in range(len(self.x)):
            x = self.x[i]
            for k in range(len(self.y)):
                y = self.y[k]
                cv2.circle(self.img, (int(x), int(y)), int(r+self.r[i,
 k]), color, -1)


    def iterar(self):
        self.frame += 1
        self.actualizar_particulas()
        self.img[:] = (0,0,0)
        cv2.putText(self.img, str(self.frame), (20,20), cv2.FONT_HERSH
EY_SIMPLEX, 1, (255,255,255))

        self.graficar()
        pilIm = PIL.Image.fromarray(self.img, mode="RGB")
        with BytesIO() as fOut:
            pilIm.save(fOut, format="png")
            byPng = fOut.getvalue()

        # set the png bytes as the image value;
        # this updates the image in the browser.
        self.wIm.value=byPng
```

Code 2. Particle class definition

```
s = System()

for _ in range(300):
    s.actualizar_particulas()
    s.iterar()
```
Code 3. Run code for neighbor interaction

So, taking the idea from the code, we make then bigger, and we particles reach some radius we stop and decrease over discrete time. When they reach its original radius, then, again can increase depending on the radius of their neighbors. They just interact with left, right, up, and bottom particles.

Here is a link with a video of the results from this part of the practice:

https://github.com/YosafatM/ESCOM-Evolutionary-Computing/blob/master/L7%20Particle%20Systems/Near/Near.mp4

Physics interactions

Just as the same before, a code was given for a demonstration, two particles hit each other during the 2000 frames, and when they do, its velocity changes, over time this changes its position on x and y coordinates.

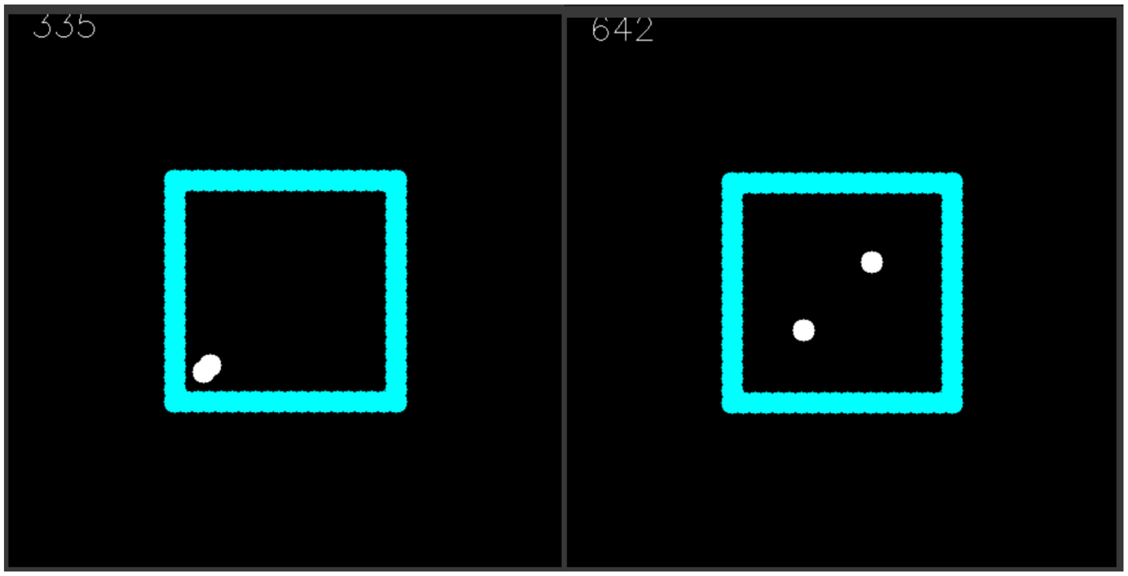

Fig. 3 and 4 Two frames from results of the physics particle code

Following the same framework, we took the idea from the code, and put more particles, and change some functions just to make it easier to solve both problems that are required in this practice: a bunch of particles inside the box try to exit from it, in the first problem it is just the window, and in the second is the window and an obstacle.

```python
RADIO = 10

class Particle:
    MaxV = np.sqrt(2) # Raíz de 1 +1

    def __init__(self, x, y, vel_x, vel_y, WallParticle=False):
        self.r = np.array([float(x), float(y)])
        self.v = np.array([float(vel_x), float(vel_y)])
        self.WallParticle = WallParticle
        self.F = np.array([0.0,0.0])

    def normalize_vector(self, x):
        norm = np.linalg.norm(x)
        if norm==0:
            return x*np.inf
        return x/norm

    def r12(self, r2):
        return np.linalg.norm(r2-self.r)

    def calculateF(self, r2):
        if self.WallParticle==True:
            return np.array([0.,0.])

        r12 = self.r-r2
        r12magnitude=np.linalg.norm(r12)

        if r12magnitude <= 2.5 * RADIO:
            return self.normalize_vector(r12) / (r12magnitude**2)

        return np.array([0.,0.])

    def update_r(self):
        if self.WallParticle==True:
            return
        self.r += self.v

    def update_v(self):
        if self.WallParticle==True:
            return

        self.v += self.F

        vmag = np.linalg.norm(self.v)

        if vmag > self.MaxV:
            self.v = self.normalize_vector(self.v) * self.MaxV

    def graph(self,x0,y0,img):
        if self.WallParticle==True:
            color=(0,255,255)
        else:
            color=(255,255,255)

        cv2.circle(img, (int(x0+self.r[0]), int(y0-
self.r[1])), RADIO, color, -1)
        return
```

Code 4. Particle class definition

```
particles=[]
NUM_PARTICULAS = 25
X_CAJA = 150
Y_CAJA = 100

def lineOfWallParticles(x1, y1, x2, y2, N, is_open=False):
    global particles
    x=np.linspace(x1,x2,N)
    y=np.linspace(y1,y2,N)

    index = range(N)

    for i in index[:int(N * 1/3)]:
        particles.append(Particle(x[i], y[i], 0, 0, WallParticle=True)
)

    if not is_open:
        for i in index[int(N * 1/3):int(N * 2/3)]:
            particles.append(Particle(x[i], y[i], 0, 0, WallParticle=T
rue))
```

```
    for i in index[int(N*2/3):N]:
        particles.append(Particle(x[i], y[i], 0, 0, WallParticle=True)
)



wIm = ipw.Image()
display.display(wIm)
maxX=500
maxY=500
x0=int(maxX/2)
y0=int(maxY/2)


img = np.zeros((500, 500, 3), dtype="uint8")

# Horizontales
lineOfWallParticles(-X_CAJA, Y_CAJA, X_CAJA, Y_CAJA, 50)
lineOfWallParticles(-X_CAJA, -Y_CAJA, X_CAJA, -Y_CAJA, 50)

# Verticales
lineOfWallParticles(-X_CAJA, Y_CAJA, -X_CAJA, -Y_CAJA, 30)
lineOfWallParticles(X_CAJA, -Y_CAJA, X_CAJA, Y_CAJA, 30, is_open=True)

for _ in range(NUM_PARTICULAS):
    x_particula = np.random.randint(-X_CAJA+10, X_CAJA-10)
    y_particula = np.random.randint(-Y_CAJA+10, Y_CAJA-10)

    vel_x = np.random.rand()
    vel_y = np.random.rand()
    particles.append(Particle(x_particula, y_particula, vel_x, vel_y))

MaxIterations = 2000

NumParticles=len(particles)

for count in range(MaxIterations):
    img[:]=(0,0,0)

    for i in range(NumParticles):
        for j in range(NumParticles):
            if i!=j:
                Fij=particles[i].calculateF(particles[j].r)
                particles[i].F+=Fij
    for p in particles:
        p.update_v()
        p.update_r()
        p.graph(x0,y0,img)
        p.F[:]=0

    cv2.putText(img, str(count), (20,20), cv2.FONT_HERSHEY_SIMPLEX, 1,
(255,255,255))

    pilIm = PIL.Image.fromarray(img, mode="RGB")
    with BytesIO() as fOut:
        pilIm.save(fOut, format="png")
        byPng = fOut.getvalue()

    # set the png bytes as the image value;
    # this updates the image in the browser.
    wIm.value=byPng
```

Code 5. Code for physics problem, first case

```
# Obstáculo
lineOfWallParticles(70, 0, 85, 30, 10)
lineOfWallParticles(70, 0, 85, -30, 10)
lineOfWallParticles(85, 30, 100, 0, 10)
lineOfWallParticles(85, -30, 100, 0, 10)
```

Code 6. Obstacle code, second case

Links for video results are listed below, the first one is for just window case, and the second for window and obstacle case, both were edited for speed because it took longer than 4 minutes:

https://github.com/YosafatM/ESCOM-Evolutionary-Computing/blob/master/L7%20Particle%20Systems/Collision/Colisiones%20Hueco.mp4

https://github.com/YosafatM/ESCOM-Evolutionary-Computing/blob/master/L7%20Particle%20Systems/Collision/Colisiones%20obstaculo.mp4

Conclusions and recommendations

Both problems were interesting in terms of complexity, I remember when I was scared about the first practice, but now I am in love with non-conventional computing, it’s interesting watching both cases having the same number of particles in the box at the end, but we see it’s faster when there is an obstacle, of course these particles are not really intelligent, so, in all likelihood, the second case would be a lot more efficient. Now, in the first problem, I loved to see how it works, and recalling my opinion about art, I don’ t know, but that looks pretty good to me.

References

[1] Cesium. *Introduction to Particle Systems*. Cesium JS. Accessed on: Nov. 1, 2021. [Online] Available: cesium.com/learn/cesiumjs-learn/cesiumjs-particle-systems/

[2] Cambridge. *Interaction.* Cambridge dictionary. Accessed on: Nov. 1, 2021. [Online] Available: dictionary.cambridge.org/es/diccionario/ingles/interaction