Final Project - Physics Fluid Simulation

Martin Alvo and Pablo Sabater

Iteration 04

Github Repository: https://github.com/pablosabaterlp/EECE2140FinalProject.git

March 2024

1 Iteration 2 - Project Objectives

- Utilize smooth particle hydrodynamics approach for fluids
- Simulate a fluid randomly flowing
- Implement an interactive element (i.e click to place fluid, manipulate fluid by clicking it)

2 Iteration 2 - Basic Functionalities

- Fluid is made of a lot of small circles which represents a particle of the fluid
- Each particle is affected by gravity, collisions, and pressure
- Each particle has velocity and viscosity(?) properties to affect its motion
- The fluid has a density property that is a representation of how closely packed particles are in a certain space (the more particles there are in a smaller space, the more dense the fluid is)

3 Iteration 3 - Commits/Implementations

- Added particle class with basic information for each particle in our simulation.
- Added method to draw particles, and a method to clear screen.
- Added wall collision detection, which included systems to set velocity to 0 after it's velocity gets below a threshold to avoid particles appearing like they vibrate,
- Implemented reflection of velocity upon impact with the wall, and added support for multi-particle creation
- Added density field class to calculate the density field of the particles in the screen. Process was re-designed a couple of times to increase efficiency.
- Added buttons for user input to activate different systems including drawing the density field, drawing the particles, using time delay in the loop, activating gravity, or activating random velocity addition.
- Created a method for adding random velocity to particles, can be activated by key press
- Implemented vector field calculations, as well as a method to update a particles velocity with the velocity vector fields. Also implemented vector field visualization, as well as re-designing particle positioning so that all particles have exact center.

4 Iteration 3 - Challenges Faced While Coding

- The main challenge faced was definitely optimization. We could have just created checks to see if the particles would be inside each other, and then make them just bounce back, but that would run really slow.
- To fix this, we implemented the density and vector fields previously mentioned. This makes it easier to check the pixels themselves to and update a particles velocity accordingly. We used numpy arrays to get this to work.
- Another challenge was figuring out how to do 3D and 4D arrays in order to store all the necessary information for the vector calculations. It just took a long time to understand and learn to manipulate.
- Lastly, there were a couple things we did to make it look better and smoother, which weren't much of a challenge they were just other things to think about. For example like previously mentioned, implementing a way to stop the particles from jiggling, or making it so they don't all get constrained to the floor, etc.

5 Iteration 4 - Final Additions

- Added user interaction for picking up water and repelling it
- Simulated particle friction/surface tension as a velocityDeceleration method for particles
- Cleaned up the code, and implemented add particles feature for user interaction
- Started creation of slides for presentation
- Cleaned up git repository and added about, usage, and download instructions

6 Screenshots - Fluid

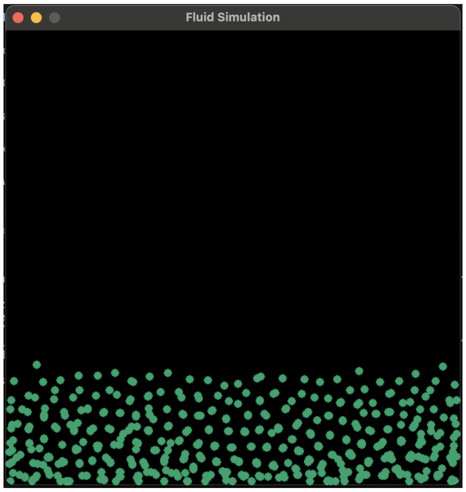

Figure 1: Fluid Stabilizing after a couple seconds.

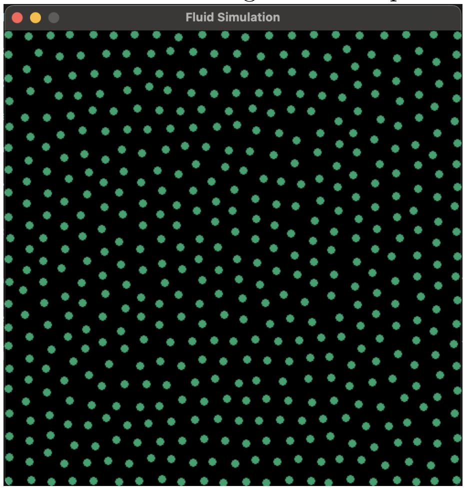

Figure 2: Fluid without the affects of gravity.

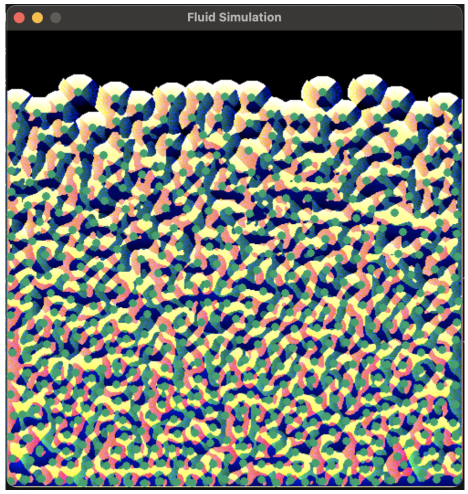

Figure 3: Vector field visualizer for the direction of the fluid particles.

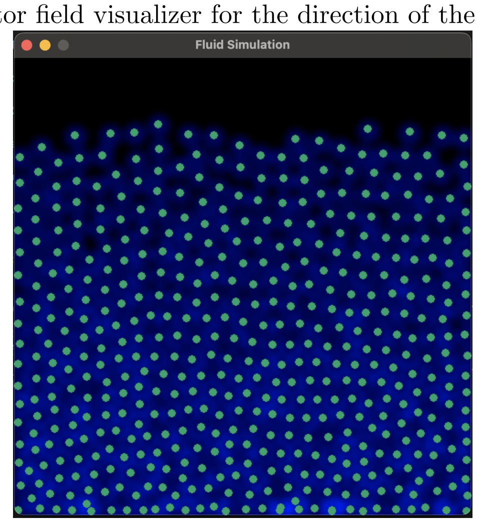

Figure 4: Density field visualizer for the makeup of particles on the screen.

7 Screenshots - Relevant Code Screenshots

```python
import pygame as py
import math
import numpy as np
import time



py.init()
class scene:

    def __init__(self, width, height):
        self.color = (0, 0, 0)
        self.caption = "Fluid Simulation"
        self.width = width
        self.height = height

        py.display.init()
        self.screen = py.display.set_mode((self.width, self.height))
        py.display.set_caption(self.caption)
        self.screen.fill(self.color)

    def updateScreen(self):

        py.display.update()

    def clear(self):
        self.screen.fill(self.color)

```

Figure 5: Library imports and scene class initialization.

```python
class densityField():

    def __init__(self, width, height, radius):
        self.fieldWidth = width
        self.fieldHeight = height
        self.field = np.zeros((height, width))
        self.smoothingRadius = radius

        self.radiusDensity = np.zeros((2*radius+1, 2*radius+1))
        self.initializeRadiusMatrix()

    def initializeRadiusMatrix(self):
        pixelx = -self.smoothingRadius
        pixely = -self.smoothingRadius

        for i in range(0, round((2*self.smoothingRadius+1)**2)):
            distance = (pixelx**2 + pixely**2)**0.5
            if distance <= self.smoothingRadius:
                self.radiusDensity[pixely+self.smoothingRadius, pixelx+self.smoothingRadius] += (1.0001 - (distance/self.smoothingRadius)**3)**6

            pixelx += 1
            if pixelx > self.smoothingRadius:
                pixelx = -self.smoothingRadius
                pixely += 1

    def updateField(self, particle):
        px = round(particle.posx)
        py = round(particle.posy)

        xmin, rxmin = (max(0, px-self.smoothingRadius), max(0, self.smoothingRadius-px))
        xmax, rxmax = (min(self.fieldWidth, px+self.smoothingRadius), min(2*self.smoothingRadius, self.smoothingRadius + self.fieldWidth-px))
        ymin, rymin = (max(0, py-self.smoothingRadius), max(0, self.smoothingRadius-py))
        ymax, rymax = (min(self.fieldHeight, py+self.smoothingRadius), min(2*self.smoothingRadius, self.smoothingRadius + self.fieldHeight-py))

        self.field[ymin:ymax, xmin:xmax] += self.radiusDensity[rymin:rymax, rxmin:rxmax]

    def normalizeField(self):
        self.field /= self.field.max()
```

Figure 6: Density class initialization.

```python
    while run:

        for event in py.event.get():
            if event.type == py.QUIT:
                run = False

            if event.type == py.KEYDOWN:
                if event.key == py.K_SPACE:
                    animate = not animate
                if event.key == py.K_p:
                    drawParticles = not drawParticles
                if event.key == py.K_f:
                    drawField = not drawField
                if event.key == py.K_t:
                    addTimeDelay = not addTimeDelay
                if event.key == py.K_g:
                    useGravity = not useGravity
                if event.key == py.K_r:
                    useRandom = not useRandom
                if event.key == py.K_v:
                    calcVField = not calcVField
                if event.key == py.K_b:
                    drawVField = not drawVField

        if animate:
            scene1.clear()
            densityField1.clearField()

            for p in particles:

                densityField1.updateField(p)

            densityField1.normalizeField()
            if drawField:
                densityField1.drawDensityField(scene1)

            if calcVField:
                vectorField1.updateVectorField(densityField1.field)

            if drawVField:
                vectorField1.drawVectorField(scene1)
```

Figure 7: Checks for key press in the main function.

```python
class vectorField():

    def __init__(self, width, height):
        self.vectorWidth = width
        self.vectorHeight = height
        self.vectorField = np.zeros((height, width, 2))
        self.xGrid3D, self.yGrid3D = vectorField.initializeVectorField(height, width)

    def initializeVectorField(h, w):
        xGrid3D = np.stack([np.full((h, w), -1), np.ones((h, w)), np.zeros((h, w)), np.zeros((h, w))], axis=2)
        yGrid3D = np.stack([np.zeros((h, w)), np.zeros((h, w)), np.ones((h, w)), np.full((h, w), -1)], axis=2)

        return xGrid3D, yGrid3D


    def updateVectorField(self, dField):

        densityLEFT = np.hstack((dField[:,1:], np.ones((self.vectorHeight,1))))
        densityRIGHT = np.hstack((np.ones((self.vectorHeight,1)), dField[:,:-1]))
        densityDOWN = np.vstack((np.ones((1,self.vectorWidth)), dField[:-1,:]))
        densityUP = np.vstack((dField[1:,:], np.ones((1,self.vectorWidth))))

        densityStack = np.stack([densityLEFT, densityRIGHT, densityDOWN, densityUP], axis=2)
        densityPicks = np.argsort(densityStack, axis=2)

        xVectors = np.take_along_axis(self.xGrid3D, densityPicks, axis=2)[:,:,3]
        yVectors = np.take_along_axis(self.yGrid3D, densityPicks, axis=2)[:,:,3]

        xVectorsWDensity = xVectors * dField
        yVectorsWDensity = yVectors * dField

        self.vectorField = np.stack([xVectorsWDensity, yVectorsWDensity], axis=2)



    def drawVectorField(self, scene):

        normalField = self.vectorField / self.vectorField.max()

        py.surfarray.blit_array(scene.screen, normalField[:,:,1].T*255 + normalField[:,:,0].T*255**2)
```

Figure 8: Vector class initialization.

```python

class particle:

    dampingcoeff = 0.7

    def __init__(self, radius, x, y, color):
        self.radius = radius
        self.color = color
        self.posx = x
        self.posy = y

        self.velocity = [0, 0]


    def draw(self, scene):
        py.draw.ellipse(scene.screen, self.color, (self.posx-self.radius, self.posy-self.radius, 2*self.radius + 1, 2*self.radius + 1))

    #Velocity Methods
    #==================================
    def initializeRadiusMatrix(radius):
        pixelx = -radius
        pixely = -radius

        vectorRadiusDensity = np.zeros((2*radius+1, 2*radius+1))

        for i in range(0, round((2*radius+1)**2)):
            distance = (pixelx**2 + pixely**2)**0.5
            if distance <= radius:
                vectorRadiusDensity[pixely+radius, pixelx+radius] += (1.0001 - (distance/radius)**3)**6

            pixelx += 1
            if pixelx > radius:
                pixelx = -radius
                pixely += 1

        return vectorRadiusDensity
```

Figure 9: Particle class initialization.

8 Screenshots - Iteration 4

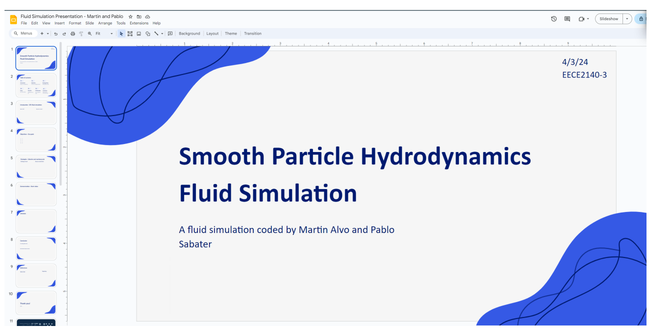

Figure 10: Screenshots of the start of the presentation.

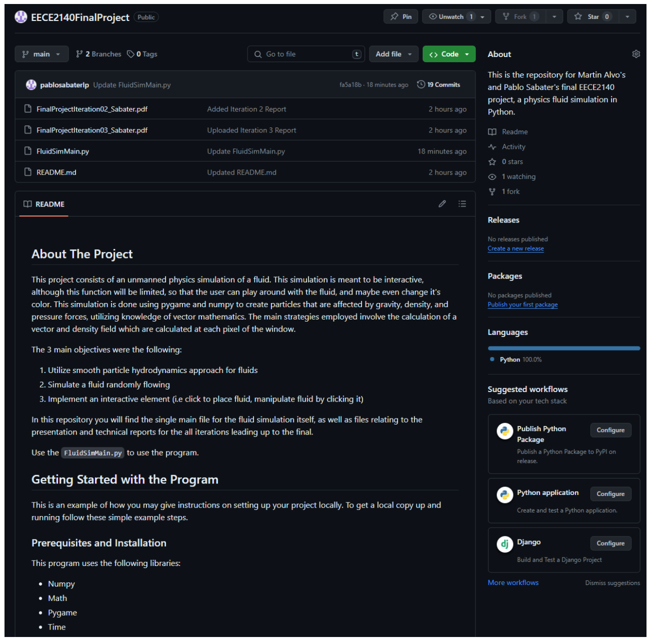

Figure 11: Screenshot of the first half of the git repository.

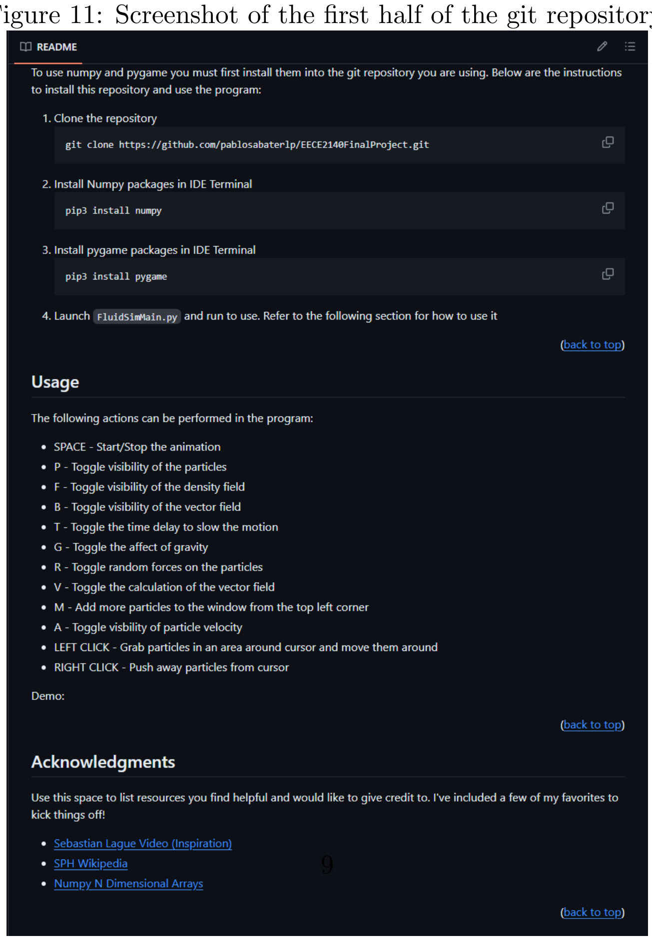

Figure 12: Screenshots of the second half of the git repository.