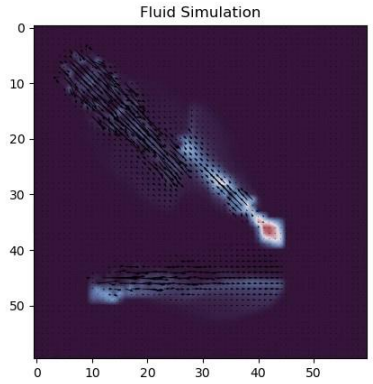
PROJECT II: FLUID SIMULATION

This code is based on Jos Stam's paper "*Real-Time Fluid Dynamics for Games*". This publication presents an easy and quick implementation of a fluid dynamics solver for game engines. The algorithms are based on the Navier-Stokes equations, which are the physical equations of fluid flow.

The following document explains the challenges I faced when developing a solution for this code in order to be able to achieve the given tasks for the project.

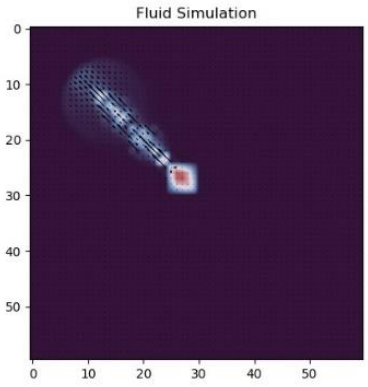
1. *Create multiple sources for velocity and density without modifying the source code.*

   To be able to do accomplish this task, I first needed to understand the paper and the code implemented to know where exactly each emitter was created and all the properties it has. What I need to understand first was where the fluids were instantiated. Once I understood this, I find where to change the velocity and density. I tried with a for loop to instantiate more emitters and change their density and velocity values, and it worked!

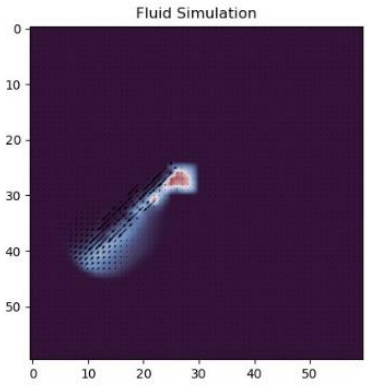


(3 sources with different velocities, densities and positions)

It was not really hard to find where to give those values to the fluid, but at that moment, the behaviors of each emitter were constant. It was just like a simple line that could go in any direction, but without variations. I noticed that the direction of the vectors was kind of inverted. If I putted an X velocity equals to -2 and in the same way for the Y velocity, the fluid was moving up and to the left, when it should be moving down to the left. So, this means that the Y velocity was inverted. I decided to multiply it by -1 to put it in the correct way.



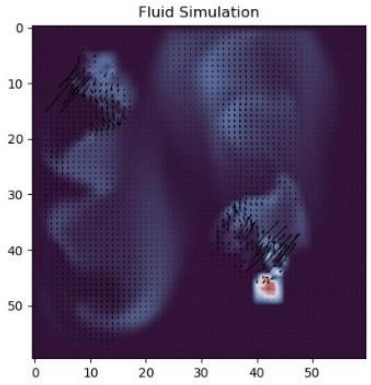
(Original movement)



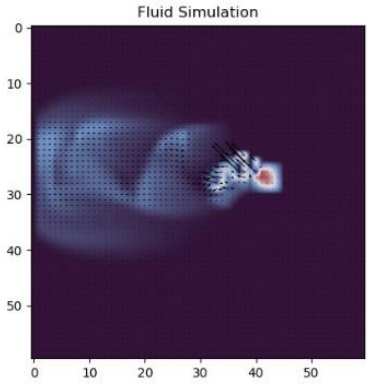
(Modified movement)

2. *Animate the velocity forces (At least 2 behaviors).*

I remembered that in other projects when I wanted that something moved with variations, I had been used trigonometric ratios. So, I started with a simple cosine multiplied by the Y-axis of the velocity of the emitter. By doing this I could create a simple vertical curly behavior. I replicate this but know multiplying the cosine with the X-axis of the velocity and I got a horizontal curly behavior.



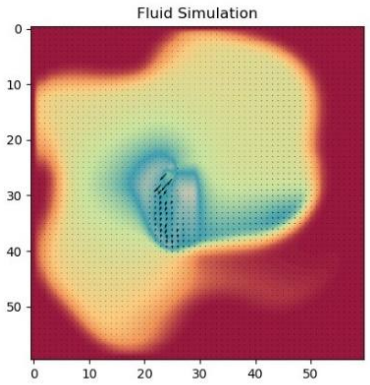
(Vertical curly)



(Horizontal curly)

I also created a variable called *factor,* which I multiplied inside of the cosine by the value of the current frame in the simulation. With this implementation, I can change the period of the movement. The bigger your factor is, the faster the fluid will move.

Then, I tried to create a swirling movement. To achieve this, I implemented the circle function; for the X-axis I multiplied the X velocity by the cosine, and for the Y-axis I multiplied the Y velocity by the sine. Finally, I added the factor inside the sine and the cosine, and I multiplied these by the value of each frame in the simulation. As it happened with the curly movement, the larger your factor is, the quicker the fluid will rotate.



(Swirl with color scheme #8)

3. *Create color schemas for the simulation.*

As you could see in the last image of this document, we have a colorful color scheme for the fluids. This combination of colors was completely chosen by me. To accomplish this goal, I made some research on the functions that matplotlib has for color schemes.

Matplotlib has several built-in colormaps. “Matplotlib colormaps are divided into the following categories; sequential, diverging, and qualitative.” (Sahakyan, 2019). I found that each category has a list of names that are stop words that matplotlib recognizes as a color scheme. Those are the names of the scheme from every category:

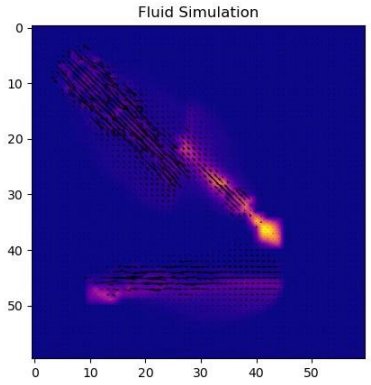
```
Perceptually Uniform Sequential
['viridis', 'plasma', 'inferno', 'magma']

Sequential
['Greys', 'Purples', 'Blues', 'Greens', 'Oranges', 'Reds', 'YlOrBr', 'YlOrRd',
'OrRd', 'PuRd', 'RdPu', 'BuPu', 'GnBu', 'PuBu', 'YlGnBu', 'PuBuGn', 'BuGn',
'YlGn']

Sequential (2)
['binary', 'gist_yarg', 'gist_gray', 'gray', 'bone', 'pink', 'spring', 'summer',
'autumn', 'winter', 'cool', 'Wistia', 'hot', 'afmhot', 'gist_heat', 'copper']

Diverging
['PiYG', 'PRGn', 'BrBG', 'PuOr', 'RdGy', 'RdBu', 'RdYlBu', 'RdYlGn', 'Spectral',
'coolwarm', 'bwr', 'seismic']

Qualitative
['Pastel1', 'Pastel2', 'Paired', 'Accent', 'Dark2', 'Set1', 'Set2', 'Set3',
'tab10', 'tab20', 'tab20b', 'tab20c']

Miscellaneous
['flag', 'prism', 'ocean', 'gist_earth', 'terrain', 'gist_stern', 'gnuplot',
'gnuplot2', 'CMRmap', 'cubehelix', 'brg', 'hsv', 'gist_rainbow', 'rainbow', 'jet',
'nipy_spectral', 'gist_ncar']
```

(Sahakyan, 2019)

For example, those are the color schemes that belong to the sequential category. I can send the name of the color as a parameter inside a matplotlib function.

To understand this better, I will show you how it works in the code. I sent the color *plasma* to the cmap argument inside the image I was using to show the fluid simulation. The result is shown below. As you can see, the colors on the simulation have changed and they belong to the ones that are on the *plasma* color scheme above.

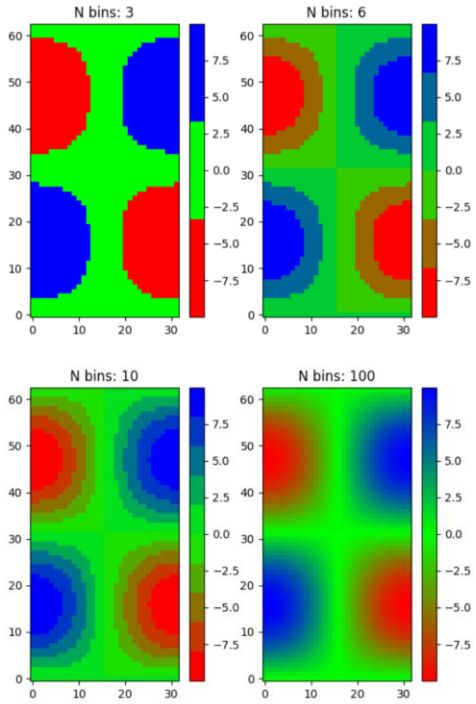
```
# plot density
im = plt.imshow(inst.density, vmax=100, interpolation='bilinear', cmap='plasma')
```



Once I understood this, I started searching if it was possible to create my own color scheme from scratch. I found that is possible to do this. "Creating a colormap from a list of colors can be done with the LinearSegmentedColormap.from_list method. You must pass a list of RGB tuples that define the mixture of colors from 0 to 1." (The Matplotlib development team, 2021). I created arrays with all the colors I wanted on each scheme. Here is an example:

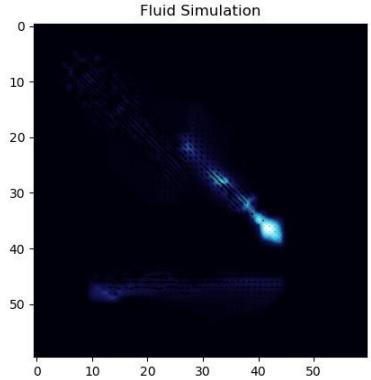
```
27        colors2 = [(0, 0, 0.05), (0.1, 0.1, 0.4),(0.2, 0.8, 0.9), (1, 1, 1)]
28        cmap = LinearSegmentedColormap.from_list(cmap_name, colors2, N=100)
```

You need to use the LinearSegmentedColormap.from_list method as it was mentioned before, and you send as parameters the cmap name, the array of colors you want, and the N bins. N bins can take the value of 3, 6, 10, or 100. This argument helps you to "mix" the colors. The bigger the value is, the higher the mix is.

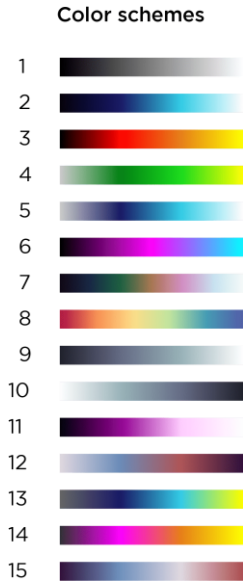


(The Matplotlib development team, 2021)

The colors I added to the array were black, blue and white, the result on the simulation was the following one:



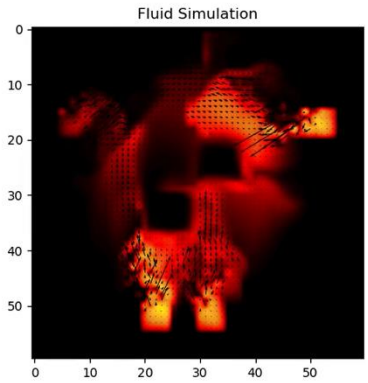
Following these steps, I created a group of color schemes. To identify and assign them on the simulation, I use the number that belongs to each one, and I send it as a parameter to a function that returns me the cmap that belongs to the number previously sent. The color scheme I used above on the simulation belongs to the #2.



4. Simulate the presence of objects in the simulation.

The final task of the simulation was to add an object to the simulations. This object should be able to interact with the fluids. First, I tried to create an emitter with a big density and with 0 velocities so that in this way, it didn't move. I thought that as a dense object, the fluid won't be able to move it, but I was wrong.

I spent a lot of time figuring out how to accomplish this task. I thought I need something that could detect the collisions on the objects. I believed that If I am able to detect these collisions, I could be able to simulate the interactions between them. Maybe that could be a good solution, but later I realized that there was another easier way to add objects.



I noticed that on the setBoundaries() function is where the simulation detects the limits of the grid, so after a lot of testing, I found that if I added a 0 to a certain position on the table, it won't simulate anything on the grid. So, I implemented this functionality with the user's input, and voila! I had objects on the scene.

This project was really interesting and taught a lot about fluid simulations. I had the opportunity to work in Realflow a few semesters ago, and I got really impressed with the power this software has. I am completely amazed, and I really appreciate the work that a lot of people do every day to find better ways to simulate real-life in movies or video games.

References

Sahakyan, E. T. (2019, June 04). *How to Use Colormaps with Matplotlib to Create Colorful Plots in Python*. Retrieved April 08, 2021, from BetterPrograming: https://betterprogramming.pub/how-to-use-colormaps-with-matplotlib-to-create-colorful-plots-in-python-969b5a892f0c

The Matplotlib development team. (2021, March 31). *Choosing Colormaps in Matplotlib*. Retrieved April 08, 2021, from Matplotlib: https://matplotlib.org/stable/tutorials/colors/colormaps.html

The Matplotlib development team. (2021, March 31). *Creating a colormap from a list of colors*. Retrieved April 08, 2021, from Matplotlib: https://matplotlib.org/stable/gallery/color/custom_cmap.html#sphx-glr-gallery-color-custom-cmap-py