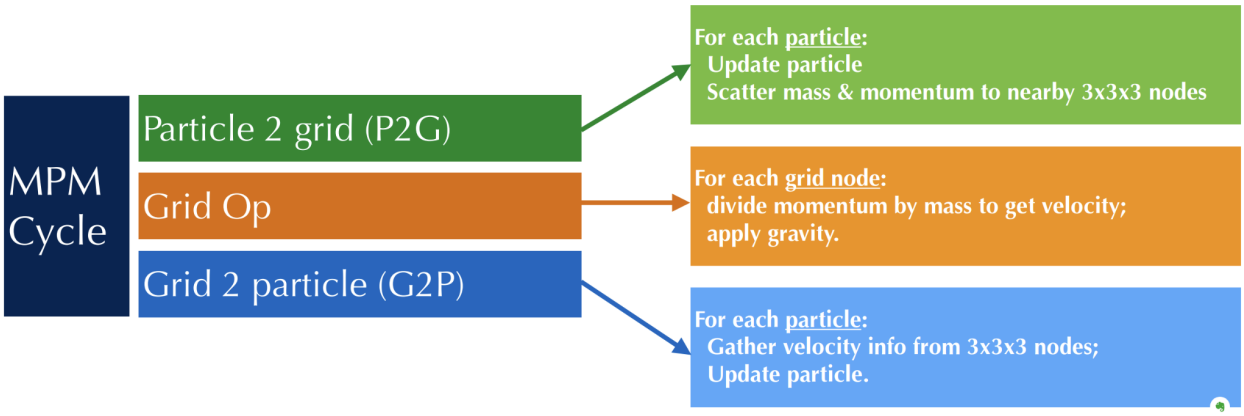
Members: Remus Wong and Christopher Chen

Final Project: L-Systems in Taichi

Overview

For our final project, we wanted to explore the powerful functionality of a high-performance graphics programming language such as that of Taichi all while learning how to implement L-systems. In addition, we used the Taichi-elements library that defines an MLS-MPM (Moving least squares material point method) solver that we use as the basis for our physical simulation. It was interesting learning more about how to utilize MLS-MPM for fast calculations. The operations are concisely described in this graph here.



Video Demo of Final results:
https://drive.google.com/file/d/1hUD6QQpeTeFTpUZJmduJTr0Hf0Qn-rY-/view?usp=sharing

How to Run

- python3 tai_rbd.py (or tai_rbd_3d.py)
- Run any of the treeX_XD.py

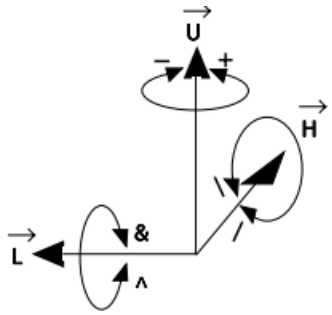
Features

- L-Systems
- Environment Controls (control gravity and substance in which the tree is in)
- Projectile spawning

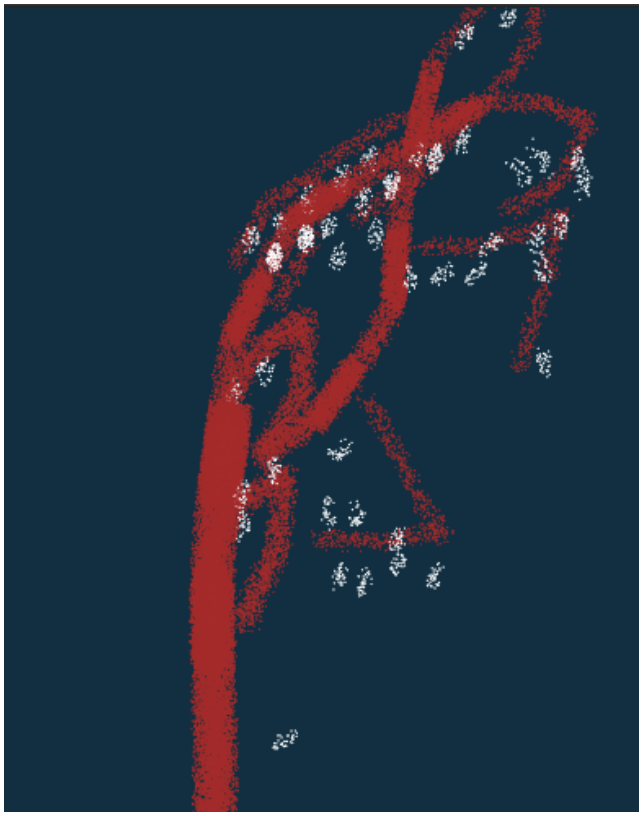
L-Systems

In the current state of our project, we were able to implement an L-system in Tai Chi to create a tree-like botanical structure using an axiom and a set of rules used to expand on the axiom. We were also able to decrease the particle density of a “branch” the further we traversed down the tree in an effort to create the tapering effect that trees have. Furthermore, we also included fruit-like objects to be added onto the branches as a way for us to add realism and to explore more into L-systems.

In 2D, we used Paul Bourke’s L-system User Notes as a basis for rules and axioms to create the botanical structures and built off of it to better adjust to the semantics of Taichi. The core functionality of the translation of the L-system string to usable information for Taichi can be seen here.
In 3D, we define a set of rules that correspond with this image from *The Algorithmic Beauty of Plants* to define how we rotate branches in 3D space.



Most of the code is maintained from the 2D L-system as the only thing that really needs to change is the way rotation is managed. To rotate, we use rotation matrices that correspond with each axis and rotate the orientation of a branch (each branch maintains its own orientation like a bone).
Because this was in 3D, this solved a problem in 2D in which branches would be created on top of each other. With the extra dimension, the tree is able to freely sway without being created on top of itself during generation. Because of this, we added the fruits only in 3D so that there would be more room to support them. You can see the white ellipsoids on the image below representing the fruits that will fall off if enough force is applied and break.



Fruits are only spawned on branches that have a certain horizontal threshold which is easily calculated by taking the dot product of the horizontal component of the orientation of a branch.

```
horizon = [0,0,0]
for i in range(3):
    if dir[i] > 0:
        horizon[i] = 1
    else:
        horizon[i] = -1
horizon[1] = 0
dot = horizon[0]*dir[0] + horizon[2]*dir[2]
if dot > .6:
    lc[1]-=.04
    mpm.add_ellipsoid(center=lc,
        radius=[0.01, .02, .01],
        material=MPMSolver.material_snow,
        sample_density=16)
```

Note that it was hard getting good looking 3D trees because there are no examples we could find that we could cross validate with so we had to make our own. In addition, there wasn't any helpful documentation on how the GUI handles the camera so there is little depth perception to observe the tree.

Controls

Controlling Gravity
We give the user the ability to control the gravity so that they can observe the tree from different angles and see the physics in motion. The control scheme is as follows

- A,D - X axis
- W,S - Y axis
- Q,E - Z axis

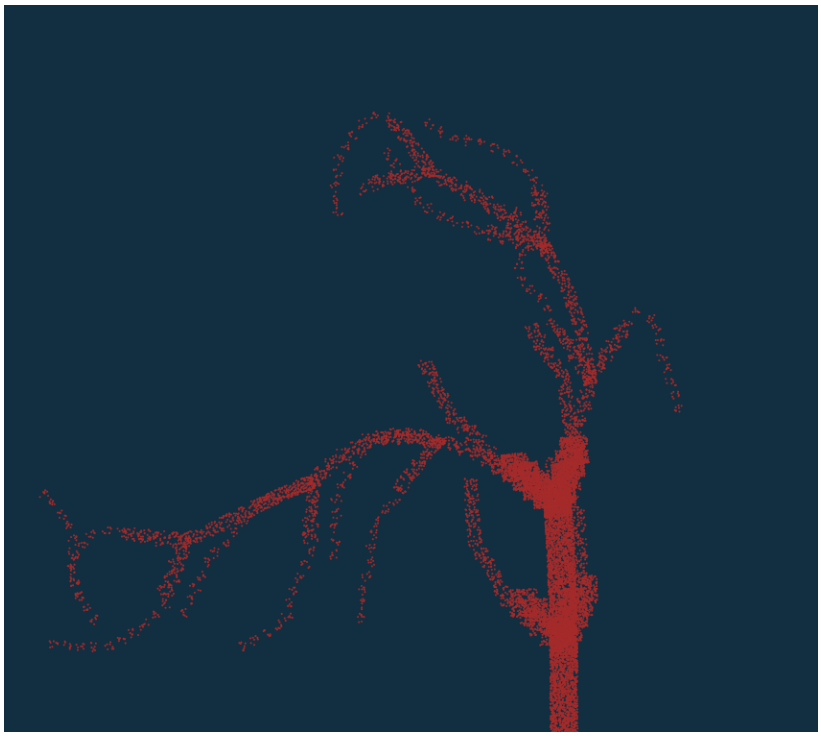
Projectile spawn
To see some more action, one can press ‘f’ which will spawn a projectile at the mouse’s position and fly across the screen. This way, users can try to break the tree and observe more behaviors.
Note that this can severely lag the simulation if too many projectiles are spawned.

Challenges

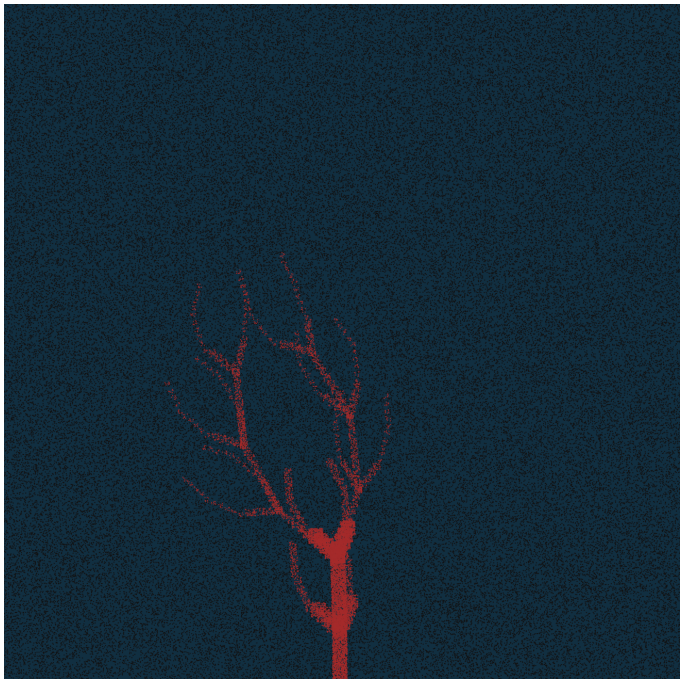
One of the biggest challenges that we ran into during the course of this project was trying to figure out a way to increase the rigidity of the material we used for the tree structure, so that the structure would not topple from its top-heaviness, as seen in some pictures below. The lack of documentation for Taichi made it extremely difficult for us to pinpoint the method to fix this. However, it was more so the underlying math that was related to it that was difficult to understand. We looked into the Neo-Hookean solid model for hyperelastic materials but we didn’t have enough experience to modify the equations to fit our own behavior.
Other issues that we wanted to address in this project follow this same pattern of lack or resources for assistance, and we quickly came to realize that the programming language Taichi was a lot more complex (and mysterious) than we had originally thought. Nonetheless, we tried our hardest given the time we had to find workarounds to the bugs we encountered, such as decreasing the material density of the branch as we get higher up the tree.
There were some initial difficulties handling the collision model of the tree. What would happen was that upon rendering the structure, some branches would spawn on top of each other and create unintended interlocking between the branches. We got the branches to interact properly after generation such that non connected branches would collide with each other.

Gallery

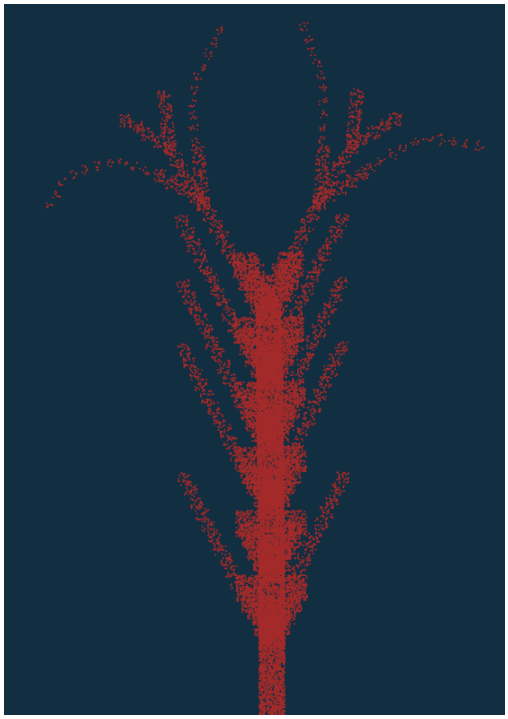
2D Tree1



2D Tree1 in Water

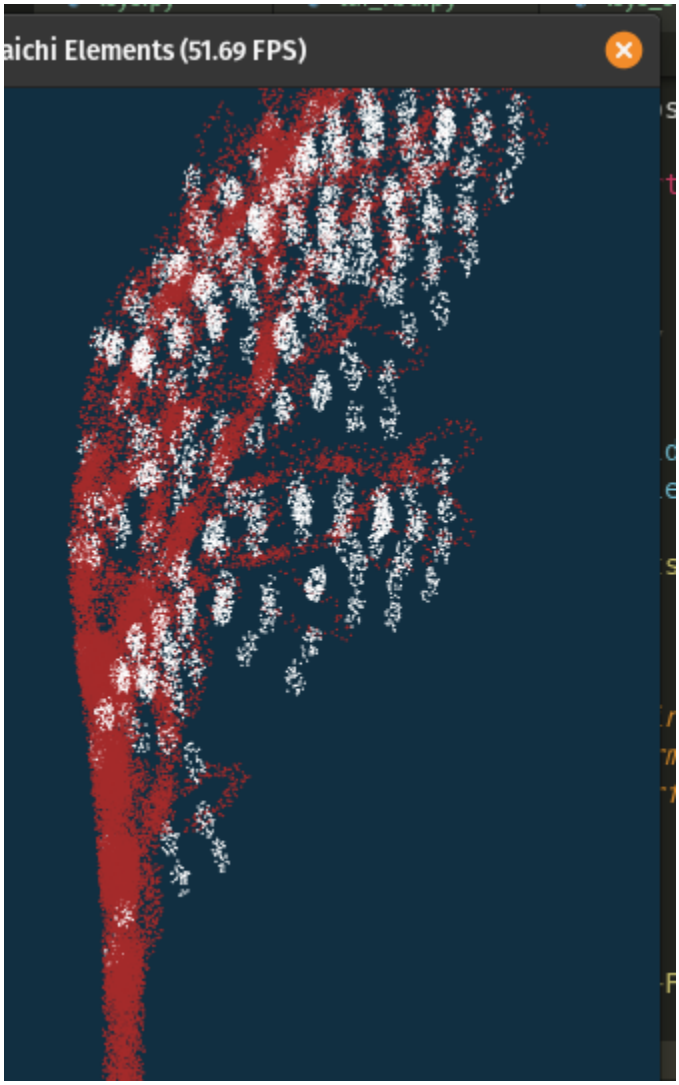


2D Tree2

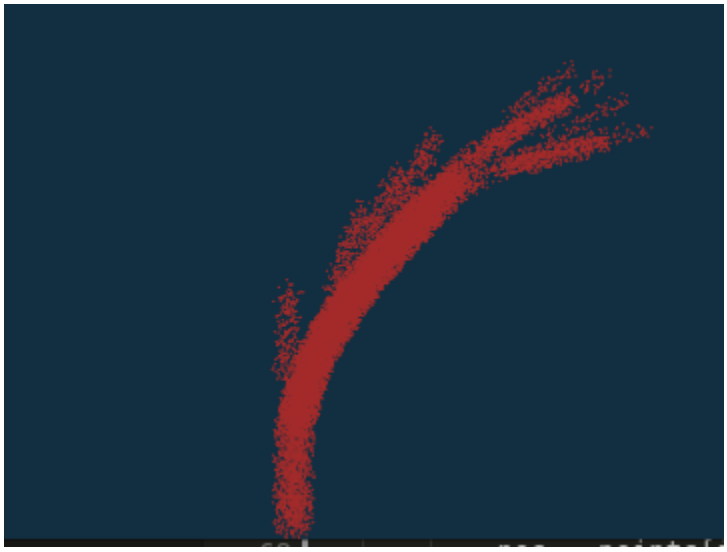


2D Tree3

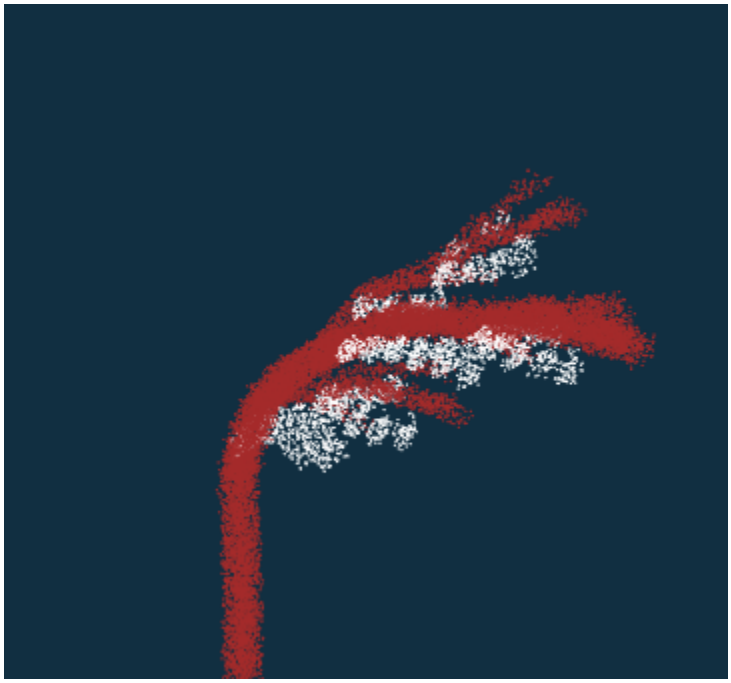
3D Tree1



3D Tree2



3D Tree3



The depth perception is a little hard to see but if you run the program and play around with it you'll be able to notice the swaying of the tree in 3D space.