Let's Get This Party Started

Partris is an interactive particle based tetris-esque game that runs in real time. Partris connects to the course via its particle/fluid simulation. The state-of-the-art water simulation technique most related to Partris is Eulerian noncompressible fluid simulation. Partris' fluid simulation relates to Eulerian noncompressible fluid simulation because Partris simulates water (a noncompressible fluid) by passing/flowing information through a grid. My implementation is possibly the hackiest implementation that could possibly exist and is quite frankly very bad. It is hardly even based in real physics.

The Interaction Between Gameplay Design Choices and Simulation Planning Techniques

One way that gameplay design choices affected the simulation planning I used for Partris was that I wanted different particles to move and interact in different interesting ways (ex: stone falls into piles, water/lava flows, plant doesn't move but grows, etc), so I had to give the individual particles different simulation techniques. Water, lava, and (surprisingly) stone all move in a similarly simulated manner, but plant and fire had to be given entirely different movement techniques.

On the other hand, by choosing the method that I did to simulate liquids, I had to make the design choice to not increase the particles per block from 10x10 to 30x30 (one particle per pixel) as the liquid movement technique I went with does not hold up when scaled up.

How it Works, Problems I Ran Into, And Why It Can't Be Scaled Up Easily

There are technically two main game loops running. One manages each tetromino as they are manipulated/fall to the floor, and the other manages the particles. The first loop is very non intensive as the tetromino stuff doesn't take much processing. The second loop however needs to make all the particles in the particle grid move and interact with all of their neighbors. It does this by looping through every particle, checking its type, then moving the particle according to its own movement scheme (dictated by particle type), and then looping through every particle again to check its particle type, then looping through that particle's neighbors, checking their particle types, then making the correct interaction happen between the particle and its neighbors. After each particle has been moved and has been checked for interactions, the board recalculates which blocks are full (any block more than 2/3 full of particles counts as full) and clears any full rows.

Now, with standard settings there are 10 x 20 blocks, with each block having 10 x 10 particles, and 30 rounds of movement and interaction checks a second. That puts 20,000 particles in the grid that all have to move 30 times every second and need to check for interactions with their 8 neighbors 30 times a second. That is up to 600,000 movements per second and 4.8 million interactions per second. This ends up a *little bit laggy*, so I set it up so that if a particle doesn't move or interact in any given tick it goes to sleep. If a particle is asleep, then it is not checked for movement or interactions (although other particles may try to interact with it). When a particle moves into a new location (including row clears!) or interacts with another particle, that new location or that particle wakes up all the particles around it. This fix SIGNIFICANTLY reduced lag as very few particles are ever truly active at any time. I don't have any real numbers for how much this reduces lag, but the lag when the board was full of

particles before the fix was pretty significant and I've since upped the particle count up to 30x30 per block instead of the default 10x10 while testing and there was zero lag even while the board was fairly active (many particles). I call that a success.

Another thing I worried about was that if I store all these particles in a grid and loop through them left to right row by row for movement/interactions was that it would end up asymmetrically updating the particles. The way I responded to this worry was that when the particle grid is created and loaded with particles, the particles are all added to a list. This list is then shuffled, and then the particles are all moved/interacted in that order for the duration of the game. I'm realizing as I write this that the particles could still randomize into an order that still leads to asymmetrical movement/interactions, but with the number of particles in the grid I feel that the randomness works.

I've tried scaling Partris up to see where it runs into problems, and there are three main limiting factors.

1. Obviously after a certain point there are too many particles that want to be considered every second. I tried scaling up from 10x10 particles per block to 100x100 and it is just unfathomably too many for my poor laptop to try to process efficiently. It WILL run at that scale, but it is not fast or smooth. The awake/asleep system is not enough to make it work, but maybe some other cleverer setup could help make them run a little quicker.

2. In retrospect this is unsurprising but shuffling all the particles in the beginning actually takes a long time. It took several minutes for my laptop to even load up the game at the 100x100 density. I tried loading it up without shuffling and it loaded up in less than 30 seconds, so there could be something to be said about the tradeoff between startup time and visual artifacts from asymmetrical movement/interactions.

3. This one is not technically a computational limitation, but my hacky liquid simulation does not look good when scaled up. Again, this was the factor that made me decide on 10x10 particles per block instead of something like 30x30 (which would be one pixel per particle). 30x30 runs without any noticeable lag on my laptop, but at that scale the water/lava behaves in a very noticeably non liquid manner.

Addressing Feedback from Peers During the Progress Report

During the progress report the two biggest pieces of feedback I got from my peers were that I could very easily come up with an interesting way to render the game in 3d as an easy way to make it look nicer, and that it was worth really thinking about how the particles would interact with one another. As for rendering the game in 3d: I would have loved to do that, but I ran out of time. As for thinking about how the particles would interact: I feel that I did a good job. All particles are able to interact with all particles easily because all types are standardized to the particle grid, and all possible interactions between all particle combinations have been considered.

The Relationship Between My Work and The State-Of-The-Art

Most standard state-of-the-art Eulerian fluid models track things like density, gravity, pressure, and viscosity to update velocities which are used to update whether cells do or do not have water in them. My liquid system also updates whether cells in a grid do or do not have water in them, but it does so in a way that doesn't track velocities or interesting/meaningful statistics about the cells' neighbors. My system loops through every liquid particle and checks if the space below the particle is open. If it is open, then it moves there. If not, then it checks the spaces to the left and to the right of that space. If either space is the only one open, the water moves there. If both are open, then one is chosen at random. If neither are open, then the spaces to the left and right of the water particle are checked for openness. If either space is the only one open, the water moves there. If both are open, then one is chosen at random. If neither are open, then the water does not move.

Limitations of The Current Versions and How I Might Get Past Them If Given More Time

There are several things I wanted to do with this project that I just didn't have time for. I feel that if I had another week to work on this that I would have been able to complete all these things. First, I *really* wish I had had time to finish implementing a proper Eulerian fluid system. The current water/lava system was meant to be a placeholder until I made a generalized system that could be used for multiple particle types by changing viscosity (lava would be thicker than water) and compressibility (I was thinking about adding steam and explosive gas, and those would be compressible!), but I ran out of time and ended up stuck with my hacky system. Secondly, I wanted to make fire give off smoke so that it would look cooler and be more noticeable. Currently fire just takes up a single particle layer of whatever it is burning, which is hard to see at high particle densities. If the fire gave off smoke, then the smoke could take up more space (thus more noticeable) and change color based on how old it is (thus looking cooler). Plus, the smoke could be plugged into the Eulerian system which would have been pretty sweet. Thirdly, there are some more particles I would have liked to add. I've mentioned steam, smoke, and explosive gas, but I would also like to add acid (which could burn through everything it touches) and dirt/mud (dirt interacts with water to form mud, and then I would rework plant to only grow in mud/dirt or something along those lines). Finally, I would like to have rendered the simulation in an interesting 3d way, with some nice lighting. All the coolest looking projects do that.

Key Binds

Up: Rotate Tetromino

Left: Move Tetromino Left

Right: Move Tetromino Right

Down: Move Tetromino Down

Space: *Slam* Tetromino

R: Restart

C: Save Tetromino

P: Toggle Pause

Debug Mode

Konami Code: Activate Debug Mode (Border will turn red)

1 – 9: Set Tetromino Shape/Type

`: Toggle 1-9 Mode (Shape or Type?)

\: Cycle Through Particle Types

Enter: Cycle Through Tetromino Shapes

A: Toggle Draw Border Around Awake Particles

D: Toggle Row Clearing

F: Toggle Automatic Tetromino Falling

G: Toggle Display Ghost Block

L: Move Tetromino Up

Z: Force a Particle Update (Only If Auto Update Particles Is Disabled)

X: Toggle Auto Update Particles

/: Disable Debug Mode