Coding Adventure: Simulating Fluids

47 MIN · YOUTUBE · HTTPS://YOUTU.BE/RSKMYC1CQHE?SI=L7PZF-TVRMJAHE82
https://youtu.be/rSKMYc1CQHE?si=L7pzf-tvRmjaHE82

SUMMARY

The episode explores the development of a fluid simulation using particle-based methods, focusing on the implementation of physical forces such as gravity and pressure to create realistic fluid behavior. The creator discusses challenges faced in simulating fluid dynamics, including particle density management, collision handling, and the introduction of viscosity to enhance realism.

- *Introduced gravity and collision damping to simulate particle movement.*
- *Expanded from a single particle to an array to represent fluid behavior.*
- *Implemented a density function to estimate fluid density based on particle distribution.*
- *Developed a pressure calculation to correct density differences in the fluid.*
- *Optimized calculations using spatial partitioning to improve performance.*
- *Added viscosity to smooth out particle velocities and reduce chaotic behavior.*
- *Explored the transition to a GPU-based simulation for better performance.*
- *Discussed future improvements, including rendering techniques and stability enhancements.*

[Music] hello everyone and welcome to another episode of coding Adventures today I'd like to dive into the world of fluid simulations so to begin with let's draw a [Music] circle very nice the circle represents a tiny bit of water or whatever fluid we want to imagine and it's going to move around in response to various forces for instance gravity is probably a good place to start so back in the code I'll

add in a gravity variable and let's also keep track of the particle's position and velocity each time step we'll then want it to be accelerated downward by gravity and to then move according to its new velocity let's take a [Music] look okay not bad but it is falling off the screen which is a bit annoying so I've added a tiny function that just checks if the particle has moved outside of a box and and if so we shove it back

into the box and send it bouncing off in the opposite direction I'd actually also like to multiply the velocity by some Collision damping factor between 0er and one here so that we can control how much energy it loses with each bounce all right so we can set up our little bounding box and turn on the gravity again and off we go bouncing by the way I've also made it so we can control the display size of the

particle here okay let's then also try out our Collision damping setting and we can see the particle now bounces lower and lower each time until it comes to a stop believe it or not though one particle is not enough particles to simulate a fluid so I've quickly upgraded our position and velocity variables to instead store a whole array of positions and velocities which we can then Loop over and just do the same update as before to

each of them I've also created a simple function that runs at the beginning to set up the particles in a little grid Arrangement just so that they're not all on top of one another so let's make a bunch of particles and I'll quickly tweak the size and spacing over here to something more reasonable and then we can let this run okay there's a pretty clear problem though which is that the particles are all just collapsing on top of one

another so I guess we need some sort of force to push them apart I'm curious to learn how this is typically handled in in the fluid simulation world so I'm going to do some reading I found a bunch of intriguing particle-based fluid papers and I've just spent the last few hours trying to work my way through those honestly most of the maths has gone well over my head as it often does but the broad ideas at least are

encouragingly simple so I think my goal for today is not to try and make some amazingly accurate simulation but just to build a rough starting point from which we can delve deeper into the maths and physics in the future when I'm hopefully a little bit smarter anyway the first step to fixing that overlapping particle problem we were having is to be able to estimate the density of our fluid at any point so I've just scattered our particles around

randomly for this example and of course since we're imagining that this represents some kind of fluid in reality there should be way more particles in this but we're always going to be limited by what our computers can handle so to approximate reality we can just cheat a little bit by blurring or smoothing out the few that we have so that it appears more as a continuous field than a bunch of individual points this simple idea is the basis of smooth

particle hydrodynamics a technique introduced back in the 70s to help solve astrophysics problems and further our understanding of the universe which today we'll be using for the equally lofty goal of making some little pixels go Splish Splash for our Amusement all right so to see how we're going to calculate this sort of density field we have here let's zoom in on a single particle and Define a smoothing radius which gives us this circle of influence around the particle where it

will have maximum influence at the center falling off to no influence at all at the Outer Edge Let's draw a little graph of this Behavior so on the x-axis we'll have the distance from the center of the particle and because negative distance doesn't make a huge amount of sense I'll just make the left side of the graph be a mirror of the positive side then the y- AIS will represent the influ of the particle at any given distance and let's

say for now that our smoothing radius is just one so a super simple function we could use would be something like this just subtracting the distance from the radius and clamping it to never go below zero and here's what that gives us this is not very smooth though so we could take that straight line and Cubit for example which will ease it out as it approaches zero another option if we wanted it to be smooth at the start start as well

would be to also Square the radius and distance before subtracting them and here's how that comes out obviously we could also try different powers or different functions alog together I think it's mostly a case of just playing around and seeing what works best but let's go with this one for now so I've used that to write this little density function which takes in the point we want to find the density at and then for each particle it gets the

distance to

that sample point which determines how much influence the particle has at that location and then it simply increases the density by the particle's mass multiplied by the influence value and the mass I've just defined to always be one for Simplicity so let's test this quickly with an evenly spaced grid of particles and I'll just try sampling the density at the center here with a radius of 0.5 for example now at the moment the density is coming to about

0.48 but if we squish the particles closer together we can see that the density value goes up which makes sense and if we move them further apart it goes down of course but what about if we increase the smoothing radius well our density value has just shut up through the roof which is very concerning because making the radius bigger should only make the result more blurry which for this uniform grid of particles should actually have no effect

on the density at all so let's think about this a bit for each particle we're calculating an influence value which we could draw as as a height and here we can of course see the shape of our smoothing function emerge now we're effectively just adding all these Heights together to create our density value but it's kind of helpful to note that if we were to First also multiply the heights by the width and breadth of these little boxes I've drawn here what

we'd actually be doing is estimating the volume of the smoothing function so with that in mind I think it's reasonable to say that if we want the density to stay the same as we change the smoothing radius then what we'll need to do is make sure that the volume of our smoothing function Remains the Same when we change the smoothing radius that means we're going to need to calculate its volume or make wfrm calculated for us at any rate and that

is come out to Pi * the smoothing radius to the^ h / 4 we can then just go back to our smoothing function and calculate the volume in here and then simply divide the output by the volume which means that now the new volume of the function will always be one let's quickly make sure this is working so the density is 187 at the moment and now if we change the smoothing radius that should stay the same which it does

of course if we make the radius too small the results will get a little dodgy since they just aren't enough particles but apart from that we now have a nice way of determining the density at any point with whatever smoothing radius we choose to use okay so let's return to our random arrangement of particles and we can now properly visualize the density values here as we increase the smoothing radius now we're claiming that this represents a fluid so one would probably

assume it's a guess at the moment because we have all these regions with different densities but I'm more interested in simulating liquids today which in Practical terms are incompressible meaning their molecules are packed together as tightly as they can be and so we'd expect the density to be the same everywhere for our simulation to behave at least somewhat like a liquid then we're going to need to rapidly correct these density differences by moving particles from areas of high density towards areas of

low density so we need to figure out how to calculate that but I started to get a bit confused at this point while I was doing my research so I'd like to take a step back for a moment and just play around a bit with an abstract example to try and wrap my head around some stuff first and then we'll come back and apply what we've learned to our actual problem so here's a simple little function that takes in a point in 2D

space and outputs a single value which looks like this and what it represents is nothing at all it's just a madeup function that we're going to try and represent with particles to hopefully gain a better understanding of this whole smooth particle business we're working with today so I've now added some code that spawns in a bunch of particles at random positions and each of these just looks up the value of the example function at its location and

stores that in this vaguely named particle property is array now we're going to pretend that we no longer have access to that example function for whatever reason so we only know the values at the particle positions and our first goal is to Simply approximate the missing values to do that we'll use the same smoothing idea from when we calculated the density earlier so I've made this little calculate property function that takes in a point in space

Loops over all the particles and just adds up the values of their properties multiplied by the smoothing function since again that just tells us how much influence the particle has at the current point and then also multiplied by the mass since that effectively scales how much influence the particles have let's see how that comes out so here's the original function again just for reference and here's our approximation obviously the shape isn't perfect but we could simply use more

particles to improve that what's more concerning though is that the values are clearly being greatly exaggerated now we could try to correct this by reducing the particle Mass but that just reveals a deeper problem which is that the values are being particularly exaggerated in regions of high particle density since obviously more values are being added together there than in regions of low density so to fix that all we actually need to do is calculate the density at each

particle using the function we wrote earlier of course and then divide each particle's contribution by its density now our approximated result looks like this this without needing to make any adjustments to the mass which is a whole lot better and it's reasonably close to the original function so what we've arrived at here is actually one of the core equations of this whole technique which says that to calculate some property a at any position X we just need to Loop over all

the particles and add together the value of that property that's stored in each particle multiplied by the particle's mass divided by its density and finally multiplied by the smoothing function given the distance between the particle and the sample Point what's interesting to note here is say that the property we want to calculate with this equation is the particle density in that case we replace a with the density which then cancels out with the density over here leaving us with

just mass times the smoothing function which is exactly what we came up with in the beginning so that bit of

math seems to check out at least okay that's nice and all but what we're more interested in right now than calculating the value of a property at any point is calculating in which direction it's most rapidly changing since that's essential to our problem of correcting the density in our fluid so I've started writing this little calculate gradient function to do that

and all this does is Define a tiny step size and then figure out how much the value changes if we take that tiny step along the X and Y AIS using the calculate property function we just wrote then the estimated gradient is just those two changes each divided by the size of the step that we took let's try it out so I've drawn in some little arrows to visualize the gradients at different points and just visually it

looks like these are all correctly pointing along the direction where the values are most rapidly increasing so that's great what's less great is that it's taking almost 20 seconds to calculate all of these which is ridiculously slow fortunately though there is a more efficient approach imagine we have just a single particle and I'll draw in the smoothing radius here as well and we're trying to calculate the gradient of whatever property at this point over here well first of all the direction in

which that property will most rapidly be increasing is either directly towards the particle or directly away from it if the property is negative so that's easy enough the gradient doesn't only tell us the direction though but also how fast the property is changing and that depends purely on our smoothing function at the current distance we can see that the smoothing function isn't very steep which means that the property will be changing quite slowly over here whereas of course if our sample point

was over here for example then it would be changing a lot more rapidly so after struggling to remember how basic calculus works for a few minutes I finally figured out the equation for the slope of the smoothing function which I've just translated into code over here and that means that we can now easily look up the slope value at any distance so let's return to our fast attempt at the gradient function and I'll delete the old code and replace it

with our calculate property code since that's almost exactly what we need except since we want to know the gradient now we'll multiply not by the smoothing function but by the slope of the smoothing function and then also by the direction towards the current particle by summing up all these individual gradients we should logically get the overall gradient then if we return to our little visualization and just run it again it should look exactly the same as

before which it doesn't I guess I got the direction back to front so I'll just stick a minor sign in there quickly I come from the trial and error School of mathematics but now it does look the same as before this optimization has taken us from 20 seconds down to about five which is still uselessly slow but head it in the right direction at least okay I've just been having another look at our gradient function and we

definitely need to stop calling calculate density all the time I somehow forgot already that that's also looping over all the particles no wonder this is so slow so what we can do is just create an array of density values and then pre-calculate those for each particle so that we can just use those cached values in our gradient function

that brings our computation time down from 5 Seconds to 18 milliseconds I probably should have started with that anyway it's still not

fantastic but it's at least usable for now so let's return at last to our little density test over here and see if we can apply this gradient stuff to make the density be the same everywhere so in the code I've defined a Target density that we want to aim for along with a pressure multiplier which is just how strongly we're going to push the particles to try and reach that density then I've also added this little

function for converting the density to a kind of pressure value and this just looks at how far away the density is from what we want it to be and then multiplies that by the pressure multiplier from what I understand this isn't really a super realistic way to calculate pressures in a liquid it more so describes the behavior of gases but it still seems to be a popular choice for its simplicity so let's stick with it for now at least I would like to

quickly visualize these values so I've set up three different colors over here one for the regions where it's negative just meaning that the density is lower than we want it to be another for where the value is positive meaning of course that the density is higher than we want it to be and finally one for the boundary between them where the density is just right I'll also change the particle color to Black so that it stands out a bit better here okay so

let's finally get these particles moving along the pressure gradient and for that we can just use the gradient function we wrote which I'll rename to calculate pressure force and then and then the property we're interested in here is of course the pressure so let's substitute in our little pressure calculation then our simulation update Loop now looks like this we still have the gravity position and collision stuff from before but I've added in the density caching we decided to do we

still need to actually apply the pressure forces here though so I'll make another loop quickly to calculate those for each of the particles and then we know that Force equals mass time acceleration so acceleration is force over Mass so my first thought was to just calculate the acceleration like this but actually we're thinking about the movement of tiny volumes of fluid here and density is the mass per volume so it's in fact the density that we want

to use instead all right all that's left then is to just increase the particles velocity by this acceleration and we can finally try it out this has been a long time coming so let's get a little drum roll going [Music] ah the Cur of the drum roll continues okay the positions are all not a number I see so most likely we're dividing by zero somewhere oh of course we're being given the position of a particle here but then we're also

looping over all the particles and finding the distance between the two and that's where everything's going wrong I guess what I'll do is just have this function Tak in the particle index instead of the position and that way we can very easily just skip over the case where the two particles are the same okay I just need to fix this up quickly and I suppose it is technically possible for two different particles to be in the same position so if that edge

case occurs let's just pick a random Direction then all right let's try this out again well at least everything hasn't blinked out of existence but the particles are getting more dense which is the opposite of what we want I guess I need to stick another minus sign in there somewhere let's see if this works now the third time okay that's that was looking promising for a moment there for a brief instant I thought it was working but

there's still a lot more red areas than I'm hoping to see ideally the whole screen should turn white since that represents our Target density so to try figure out what's going on I want to see what happens if instead of adding the acceleration to the velocity we just assign it directly so we're removing any inertia from the particles they're just purely moving based on the current pressure Force okay if we run this now nothing happens but that's fine we aren't

accumulating velocity anymore so I guess we just need to turn the pressure multiplier up really high all right that looks interesting I am a bit surprised by how close together some of these particles are although they seem to gradually be pushing each other apart and I actually remember one of the papers mentioning this potential problem with with the smoothing function we're using since its slope becomes very shallow as the distance becomes small meaning that our pressure force will

also be small when the particles are close together that seems odd so let's maybe ditch this nice smooth curve for the spiky version instead since of course the slope of this one just gets steeper towards zero so I had to do the volume and derivative calculations again but ended up with these two functions here plugging those in we can see our little map looks just ever so slightly different and then I'm going to turn up the pressure multiplier again now and

see what happens okay that's looking a lot better actually the question now though is what thises will look like if we put the acceleration back to how it's supposed to be so I've changed it back and I'm going to try running this again let's maybe try increasing the pressure multiplier a bit so that the particles can react more quickly that's looking reasonbly good I think although maybe I'm imagining this but it seems to be getting worse over

[Music] time okay I'm definitely not imagining it so another thing we need to think about is Newton's third law of motion every Force has an equal and opposite reaction force so when we were adding on this pressure force between the current particle and some other particle we want to make sure that the other particle experiences the same Force just in the

other direction I've seen a bunch of different suggestions on how to actually do this but an i simple version is to calculate this shared pressure which is literally just the average of the pressure values calculated at both particles so let's try that out quickly and I'll just increase the pressure multiplier again and this does seem to have made a pretty big difference I guess maybe that Newton guy was onto something now we're only dealing with a

few hundred particles at the moment which is not very many so let's ramp this up to a few thousand instead and this is running at 5 frames per second so we'd better start optimizing and by far the most critical place to do that is when we're calculating the densities and pressure forces we should really avoid looping over all the particles

that lie outside of the smoothing radius since those don't contribute anything and they're slowing us down

immensely to do this we're going to need to chop space up into a grid and we'll choose the size of the grid cells to be the same as our smoothing radius since that means if we imagine there's a bunch of particles on here that means that to find the particles inside of the smoothing radius we only need to consider the 3X3 grid of cells around the center of our Circle and in that way way of course we cut out a huge amount

of unnecessary work now to actually implement this we could say that each cell has its own list that grows or shrinks to hold however many particles are currently inside of it but we're probably going to want to convert the whole simulation to a comput shador at some point to run on the GPU and there we need to be able to specify ahead of time how much memory we're going to use so I'd like to experiment with a different GPU friendly

approach which I've been reading about in this paper here I'm going to modify it very slightly though so that we don't need to know the dimensions of the grid ahead of time meaning that particles can travel anywhere in the world and it'll still work so what we'll do is create a single array with at least as many entries as we have particles so here it has 10 entries meaning in this case we could have at most 10

particles then for each of these particles we're going to calculate the coordinate of the cell that it's in so for example particle Z which happens to be this one over here is in the cell 2 comma 0 we need to turn that cordinate into a single number to make it easy to work with though so we can just do something like multiply the X and Y by two different prime numbers and then add them together to get some arbitrary hash

value we can then wrap that around the length of the array so that it becomes a valid index three in this case and let's call that our cell key not to be confused with the seal people so since this was point0 we'll store the cell key over here at index0 in the array then the next point a turns out has a key of six and so we'll record that in the next place and so on and so

forth for all of the particles that we have now we want the points that share a cell to be next to one another in this array so that we can efficiently Loop over them of course if they're in the same cell they're going to have the same cell key so we can simply sort the list based on those keys to do that and now we can easily see from this array that particles 2 5 and 7 are all

together in the same cell particle zero is in a Cell all by itself and so on anyway let's call this array our spatial lookup because that sounds nice and fancy and then the final thing we need to do is create a second array of start indices which looks like this to understand this second array let's just do a quick example so say we want to know which points are in this cell over here we would first calculate the cell's

key like before which is N9 in this case and then we'd proceed to look up the ninth element in the array of stat indices which is this last one over here the number six that lets us know that we need to head over to index six in

the spatial lookup in order to find the first entry with the cell key that we're interested in we can then simply Loop over all of those to get the indices of

the particles that are in that cell unfortunately it is possible for different cells to end up mapping to the same key which would mess with these results but we're anyway going to need to do distance checks to see which points are actually inside the smoothing radius and so that'll get rid of any mistakes obviously having to check extra particles from some other random cell that just happens to have the same key does waste time but that's what we get

for trying to implement an infinite grid with a ly non- infinite amount of memory anyway turning this concept into code didn't go as smoothly as it possibly could have but after a bit of frustration here's what I finally ended up with we have a function for updating the lookout whenever the points have moved and this just calculates the cell key for every particle and records that along with the particle index the array is then sorted based on those keys and

lastly the stat indices are calculated simply by testing if each key is is the same as the key that came before it because if not then it must be the first occurrence of that key and we can record its index as the start index here are the little helper functions by the way for calculating the cell coordinate hash and key then finally there's the function that allows us to actually find all of the points within the radius of some

given sample point this works as we've seen by just looping over the 3X3 block of cells around that sample point and calcul deleting each of their keys each key is then used to look up the start index for that cell so we can Loop over all the points in the cell and of course once we reach a point that has a different key we just exit out of the loop all it remains then is to do a

quick distance check to make sure the point is actually inside the circle and then we can do whatever we want with it trying this out now we've gone from barely five frames per second up to 120 so that was a reasonable success but I'd be a lot happier about about it if our simulation wasn't in total chaos over here one Cluny way I found to improve this is to Simply start off with a really low pressure multiplier this way

the particles don't have such a huge initial burst of acceleration and they can spread out a little and then we just gradually increase the multiplier and it seems to work a bit better at least that's not really a usable solution though so I've just been implementing an idea I read about where we B basically predict what the next position of each particle is going to be simply based on the current velocity and use those predicted positions when

calculating the densities and pressure forces I guess this could help the particles to better react to upcoming situations and maybe compensate a bit for the fact that time is obviously not continuous in a computer simulation but rather broken up into discrete steps okay let's try it out I honestly don't see it making a big difference though never mind it's actually making a pretty massive difference I'd say well that's a nice

surprise just for fun I've quickly gone into the particle rendering code I have here and just set it up so that we can visualize the speed of the particles with a color then here's a little gradient I made for that so the slowest particles will appear blue fading to Red for the fastest [Music] particles I also added in some simple controls for things like pausing the simulation stepping through frame by frame and resetting [Music]

it all right I've been playing around with this some more and one thing I've noticed is that it behaves very inconsistently at different simulation frame rates here's a little grid of simulations I set up to observe this problem so we're going from 60 simulation steps per second on the top left to almost 1 th000 steps per second on the bottom right and as we can see here if the number of steps is higher meaning the time step is smaller and so

we're predicting less F to the Future the particles take longer to settle down so even though this feels a bit wrong to me I'm going to try just removing the Delta time here and use a constant look ahead Factor instead let's then run the same comparison again and interestingly it is actually behaving a lot more consistent ly now well okay I guess we'll go with that then now I think it'd be fun if we could interact with the particles in some way

so I've written this little function that basically just pulls nearby particles in towards the mouse or pushes them away if the input strength is negative which is controlled by left or right clicking so let's give it a shot I'm going to start by pushing these particles outwards and then let's try slicing through the fluid all right I feel like this is slowly starting to get somewhere clearly we haven't really

succeeded in our goal of making this fluid incompressible it's certainly compressing and expanding all over the place but the density does even out over time so that's something at least and we can look into fancier methods to try and solve this better in the future anyway now that this seems to be somewhat working at any rate I think it's time we brought gravity back into the mix so I'll reset this quickly and let's bring in our settings window I feel like

the density could maybe be set a bit higher so I'll turn that up and then let's get the simulation going again okay I guess that's a little too high now so I'll dial that setting back down a bit then let's turn on the gravity wait that's up upside down let me make it go the other way instead the particles are quite wild at the moment so I'll also increase the pressure multiplier to try

and drain them in a [Music] little okay let's try picking up a bowl of [Music] water and dropping it back in Splash this is actually working surprisingly well I'd say I mean I realize I've been droning on for over half an hour already but all that we've really done is made a bunch of points that don't like to be too close together but not too far apart either and while

this is obviously far from being super realistic or anything I think it's still quite fascinating that this fluidlike Behavior has already Arisen from just the few little things we've [Music] implemented anyway let's see if we can still improve move this at least a little bit more today and perhaps even Venture into the third dimension so a few issues are jumping out to me at the moment for example there's the fact that the particles are

really tightly squeezed together along the edges here and that's causing a gap between the rest of the particles since they're trying to get away from that overly dense region and a similar effect seems to be happening along the surface of the fluid as well another thing is that when we have fast moving particles such as when starting up the ulation for instance the fluid seems to be overly chaotic for example if we zoom in on a

single frame here we can see from these colors how the velocities are all over the place even between nearby regions for this last problem at least I think it would be a good idea to try add a bit of friction between the particles in the fluid more commonly known as viscosity so let's actually take a moment to look at the famous Navia Strokes equations for incompressible fluid flow which under pin everything that we've been doing first of all this equation here

just says that the density of the fluid must remain the same everywhere and I mean we're trying then the other equation tells us that each tiny little volume of fluid is accelerated down the pressure gradient and that it responds to external forces such as gravity and mice in our case so we've done both of those terms but then this slightly scary looking term here is the viscosity and essentially what it does is caused the velocities of nearby regions of fluid to

become blurred together now we could implement it this way but for today at least I actually want to go with a different approach I've seen that seems much simpler but still achieves the same sort of thing so I've added this little function here which just takes in the index of a particle and Loops over all the other particles within the smoothing radius for each of those it then calculates the difference between the velocities of the two particles and adds

that on to the viscosity Force meaning that over time each particle's velocity will become more like its neighbors and nearby neighbors have more influence as usual which is done using this viscosity kernel for which I've just repurposed that function that we were originally using for the pressure Force okay let's try it out oh my settings have been reset so we're back to no gravity at the moment but let's see what that looks like currently our viscosity is at zero so

let's turn it up I'm not sure what a good value would be let's just try five maybe no definitely not that looks very strange my guess is that the viscosity is just way too high at the moment and so some of these particles are almost exactly matching one another in velocity and that's causing them to Clump together weirdly so let's reduce it to maybe 0.5 instead and see how that goes okay this is looking pretty good

actually we can see how the particle velocities are more smooth down out and so we no longer have those few particles with super high velocities shooting off on their own I want to see how this looks with gravity enabled again so I've set up a quick comparison here with a range of different viscosity values and let's see how it [Music] goes I think the one on the top right looked best to me but let's see that

again quickly and I'll freeze it here actually because this gives us a nice view of the increasingly smooth out results we get with this new artificial viscosity ter obviously we don't want to smooth out too much detail

though so I'm going to keep that value pretty low okay now I'm not quite sure yet how to tackle that boundary problem I mentioned earlier so I'm going to just ignore that for now and instead work on something else

that's bothering me if we lift up a bunch of water and then let it go we can see how it quickly splits up into these little droplets of just a few particles each which looks a little strange we could get rid of this Behavior by simply clamping the pressure values that they can't go below zero meaning that the particles won't pull each other together anymore so let's try that out but now the particles are just raining down individually which doesn't

look right either I guess allowing that negative pressure was giving us a very crude kind of surface tension effect and so I think until we Implement a more accurate version of that in a future video perhaps we should probably hang on to it so I'm going to undo that change and instead I want to try an interesting work around one of the papers suggested which is to Simply have a second pressure Force purely for pushing apart

particles that get too close together so this is the shape of the smoothing function we're using for our density calculation at the moment and what the paper recommends is to use another even spikier version to calculate what the they call the near density from this near density we calculate the near pressure simply by multiplying it by some constant meaning that this will be a purely repulsive Force so let's try it out quickly and just for fun I'm going to see what

happens if we make the near pressure multiplier negative okay they just collapse in on one another which makes sense so let's then try a positive value and now I want to try picking up a ball of fluid again and dropping it and as we can see this time it's able to hold its shape a lot better since the particles are no longer getting pulled into those tiny clusters we saw before so I think we've succeeded in

improving the fluid a bit with these last two changes although I'm certainly not entirely happy with how it's behaving yet for example something that bothers me quite a lot is how it often appears to sort of Bounce more like a jelly than a liquid this is happening pretty much all the time but we can see a dramatic example if I just let the fluid come to a rest and then change the target density for example now we could get less jiggly

results simply by increasing the pressure multiplier which is why it's often actually called the stiffness constant by the way so here I've set up a little test where it has a value of a th000 and if I change the density now we can see that it does settle down a lot more quickly but that doesn't come for free though because the greater the forces in our fluid are the more simulation steps we need to run per frame to avoid things

devolving into chaos for example if I just lower the number of steps here slightly we can see that already we're on the brink of [Music] pandemonium anyway I'm sure we'll learn about ways to overcome or at least improve this problem in the future but for right now what I'd like to do is finally convert the whole simulation to a computor so that we can run it on the GPU which excels at doing loads of tiny tasks in parallel which should be a

perfect fit for our particle calculations I doubt that'll take very long so I'll see you in a [Music] minute [Music]

okay I kept finding new and creative ways to mess everything up but this finally seems to be running properly in a computat I'll still need to test the performance with more particles of course but just glancing at the FPS counter looks promising so far hovering at around 500 frames per second by far the trickiest part was translating just one line of code the array. sort from our neighborhood search I ended up spending quite a while trying to figure

out how to implement a parallel sorting algorithm called bonic merge sort to replace it in particular I was trying to generate this pattern of lines and generalize it for any number of inputs the inputs here are represented by the horizontal lines and we have 16 of them in this case meaning that this network can sort 16 values let's actually assign a random value to each input so we can see how this goes then each of these little vertical

lines represents a pair of inputs that we're going to compare and potentially swap around so to start with we're going to look at each of the eight pairs that we have over here and let's say we want to sort from high to low so in this first pair we have five on top and three on the bottom which we're happy with so that can stay unchanged in the next pair though we can see seven on top and eight

on the bottom so the bottom is a higher number which is not okay and we'll need to swap them around so we can look at all these pairs in parallel figure out which need to be swapped and then swap them of course this alone is unlikely to sort the list so once that's done we'll need to continue to the next stage here the pairs are arranged changed a little differently but the operation is still exactly the same we can see this first

pair has five on Top seven on the bottom so it will need to swap as will this pair with three on top and eight on the bottom and so [Music] on this pattern has been carefully devised by some clever person such that following it will do all the comparisons required to guarantee that the result is fully sorted by the end for anyone interested I'll quickly show the implementation I came up with so this part runs on the CPU and is

responsible for simply looping through each of those patterns and telling the GPU to sort the pairs and then here's the code that actually does that pair wise sorting so it starts by just figuring out which pair of numbers it's actually looking at and then it Compares them to see if they need to be swapped and if they do it of course swaps them all right I've been doing some testing with different amounts of particles so here is 100 particles for example and I

think it's kind of cute watching these little droplets wobble about then after that I tried 100,000 particles but my computer was not happy about that so there's definitely a lot of room left for optimization here's a test with about 40,000 particles though which seems to be running okay although I'll have to fine-tune the settings of course because at the moment there are some weird tendrils shooting out of the liquid and I really don't know why or perhaps we

could call this a a speculative simulation of how liquids might behave on an alien planet the settings are definitely quite finicky at the moment though so that's something else I want to improve here's another little test I've set up by the way and this one has an obstacle over here with a gap beneath it because I just want to see if the simulation is able to keep the height of the liquid the same on both sides so I'm just going to start

bucketing some of this liquid over to the other end here and then let's grab another blob and another and then let's just wait a bit for this to settle down but already we can see the height gradually leveling out so even though our simulation is far from being super realistic it's nice to see at least that it's not entirely unrealistic either okay now as always there's so much more I still want to do but to end with for today let's see if

we can get this working in the third dimension this basically just means replacing a bunch of float twos with float threes in the computor as well as updating the scaling factors of the various smoothing functions and their derivatives and making sure our neighborhood search is aware of this brand new dimension as well of course I've also updated the Collision function to work in 3D and I tweaked it to account for the bounding box being moved or rotated as well as scaled simply by

transforming the points and velocities to its local coordinate system then resolving those the same as before and then finally transforming them back into World coordinates my first attempt at running this did not go particularly well but after some trial and error with the settings I managed to get this rather goopy looking result and with a few more tweaks from there I was finally able to get something that I was reasonably happy [Music] with so let's just play around with this

a bit I'm going to try squeezing it together and what's nice about how we implemented the neighborhood search stuff is that that we UNC constrained to any predetermined bonds so we can stretch this out however much we want as well and it should still work those little waves were looking quite nice I think so let's actually smoosh this together again and then try that out once [Music]

more now another thing to add to my list of a million things I want to improve is how the fluid is actually rendered these little balls are good for seeing what's going on of course but it would be nice to make it actually look more fluid like and I guess some sort of Ray marching is probably a good way to approach that but we'll have to see so let me actually make a quick note here of my sort of wish list for this

project I want the simulation to be more stable and performant so that we can have many more particles and I'd like the parameters to be less finicky so it's easier to get good results and also for the particles to behave better along the boundaries because it looks really odd at the moment another thing that would be really nice is to be able to do stuff like put little boats or rubber ducks in the water and just watch them

Bob about then finally of course there's the rendering stuff I just mentioned as well so plenty of work for the future but until then I hope you've enjoyed following along with the process so far okay that's all for today let me know if you have any suggestions for this project or for anything else you'd like to see all right thanks for watching and [Music]

goodbye