2/3/25

Today I would like to add another volume calculation like the marching cubes method (something with less assumptions about the object shape, perhaps with some over cost to this) as well as run a larger test where I can view each graph side-by-side to see the effect of convergence and importantly non-convergence too.

```
(base) james-paget@james-paget-HP-255-15-6-inch-G9-Notebook-PC:~/Desktop/SH_DDA$
 python SimulationVaryRun.py
^[[Apoints =  5192
volume =  155.33333333333334
Usage: python <RUN_TYPE>
(base) james-paget@james-paget-HP-255-15-6-inch-G9-Notebook-PC:~/Desktop/SH_DDA$
 python SimulationVaryRun.py
points =  5244
volume =  147.33333333333334
Usage: python <RUN_TYPE>
(base) james-paget@james-paget-HP-255-15-6-inch-G9-Notebook-PC:~/Desktop/SH_DDA$
 python SimulationVaryRun.py
points =  5247
volume =  -6.0
Usage: python <RUN_TYPE>
(base) james-paget@james-paget-HP-255-15-6-inch-G9-Notebook-PC:~/Desktop/SH_DDA$
```

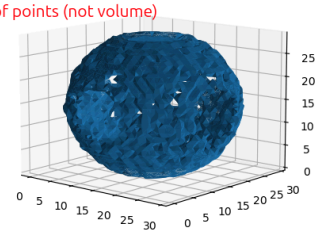
While trying to implement this algorithm using the module ***skimage.measure.marching_cubes()*** my test on a volume filled with points at resolution 50 voxels total gives essentially random volumes.

```
(base) james-paget@james-paget-HP-255-15-6-inch-G9-Notebook-PC:~/Desktop/SH_DDA$
 python SimulationVaryRun.py
points =  5221
volume =  -160.0
Usage: python <RUN_TYPE>
(base) james-paget@james-paget-HP-255-15-6-inch-G9-Notebook-PC:~/Desktop/SH_DDA$
 python SimulationVaryRun.py
points =  5285
volume =  -196.0
Usage: python <RUN_TYPE>
(base) james-paget@james-paget-HP-255-15-6-inch-G9-Notebook-PC:~/Desktop/SH_DDA$
 python SimulationVaryRun.py
points =  5255
volume =  -222.0
Usage: python <RUN_TYPE>
(base) james-paget@james-paget-HP-255-15-6-inch-G9-Notebook-PC:~/Desktop/SH_DDA$
```

N=500, not the 50 used before

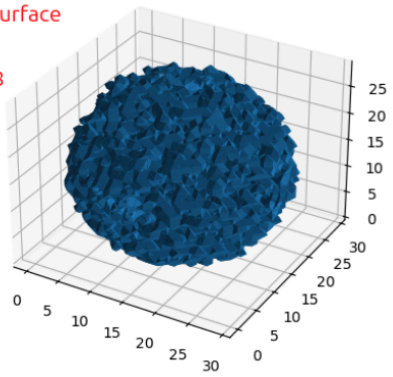
By tuning the level set parameter you can pick out better surfaces, as well as by reducing your resolution if points are not dense enough. Even for plots below which appear to have found the correct bounding surface, the marching cubes still gives negative volumes consistently (likely due to the surface seen having a backwards face as well which sits within the interior). This makes the volumes found here unusable, and given the amount of tuning required for each setup this may be even more

30x30x30
1.0 level surface
surface of points (not volume)



30x30x30 lattice
0.1 level surface
N=10000

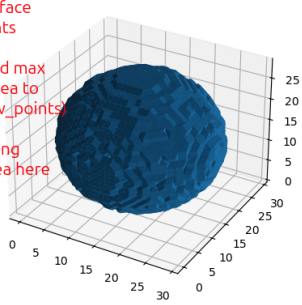
Vol=-7458



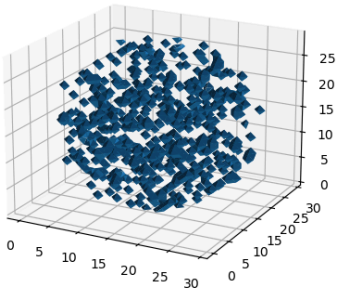
30x30x30
0.1 level surface
surface points

-> Also raised max boudning area to 1.1*max(raw_points)

=> Still getting negative area here

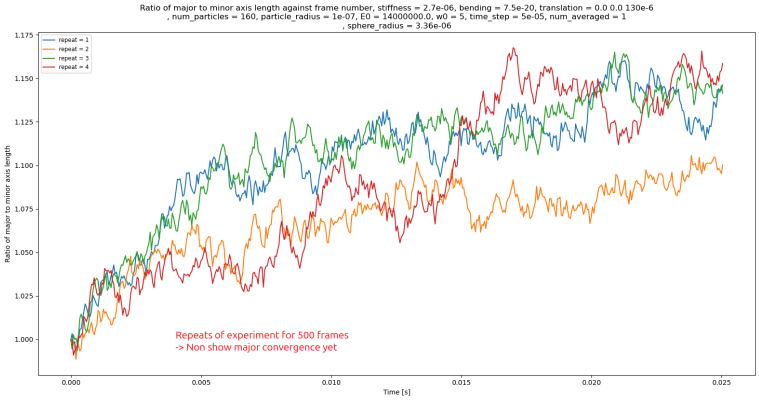


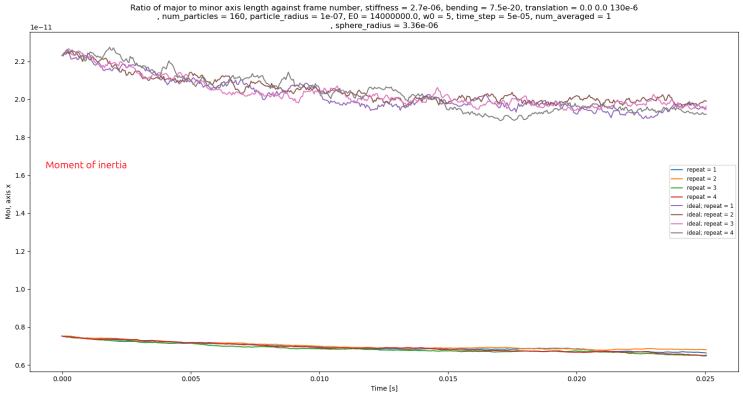
30x30x30
0.1 level surface
N=1000



difficult to implement in the final dynamics plot. Hence I will leave this for now and return if the ConvexHull approach appears to give significantly incorrect values.

I am also in the process of writing up my final report.





1/3/25

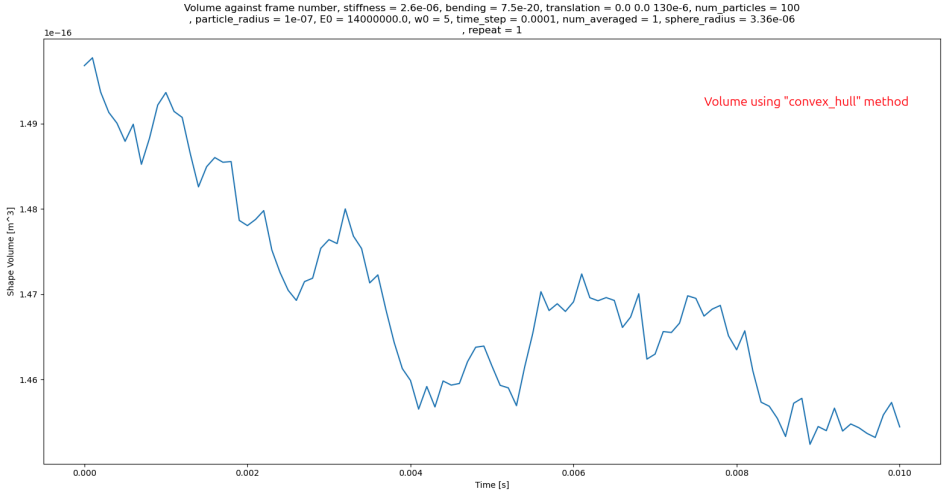
My plan for today to focus on the dynamics section of work more as, after talking to our supervisor, it was showing more convergence and the analysis of the other method did not work particularly well as despite a set of points having zero force each individual particle just wanted to step out of this stability. Therefore now we want to see whether the dynamically deforming mesh in the trap is conserving its volume, possibly through some shoelace theorem type calculation (however this would be quite difficult since you would have to ensure all faces considered were outward pointing, which with quite jumbled vertices from deformation could prove difficult but maybe possible), and what its shape is (which we could possibly find by calculating its moment of inertia and comparing that to an ellipsoid's MOI). If we can show its volume conserving then this will correspond with real deformation observed for particles of this scale (conservation of mass but on small scales, e.g. no density changes either), and if we can find the shape it converges to then we can directly compare this to observed shapes to ensure/measure how good of match we have.

```
(base) james-paget@james-paget-HP-255-15-6-inch-G9-Notebook-PC:~/Desktop/SH_DDA$
 python SimulationVaryRun.py vol
points =  [[0. 0. 0.]
 [2. 0. 0.]
 [0. 2. 0.]
 [2. 2. 0.]
 [0. 0. 3.]
 [2. 0. 3.]
 [0. 2. 3.]
 [2. 2. 3.]]
volume =  12.0
```

```
(base) james-paget@james-paget-HP-255-15-6-inch-G9-Notebook-PC:~/Desktop/SH_DDA$
 python SimulationVaryRun.py vol
points =  [[0.  0.  0. ]
 [2.  0.  0. ]
 [0.  2.  0. ]
 [2.  2.  0. ]
 [0.  0.  3. ]
 [2.  0.  3. ]
 [0.  2.  3. ]
 [2.  2.  3. ]
 [1.  1.  1. ]
 [1.  1.5 0.5]]
volume =  12.0
```

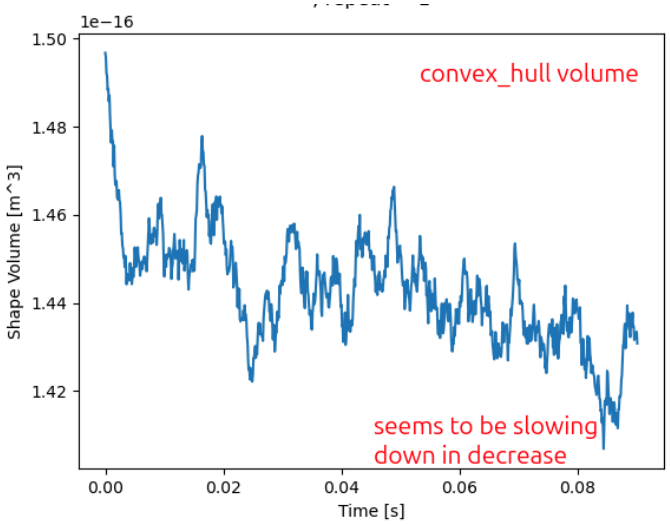
interior points to try break volume calc, but still finds correct value

My partner has started working on calculating the MOI, hence I will attempt to find the volume and apply it to a plot. Using ***Scipy's spatial.ConvexHull()*** function with a test set of points generates the correct volume by finding the outer surface of the points, which can be seen by me introducing interior points and not causing problems.

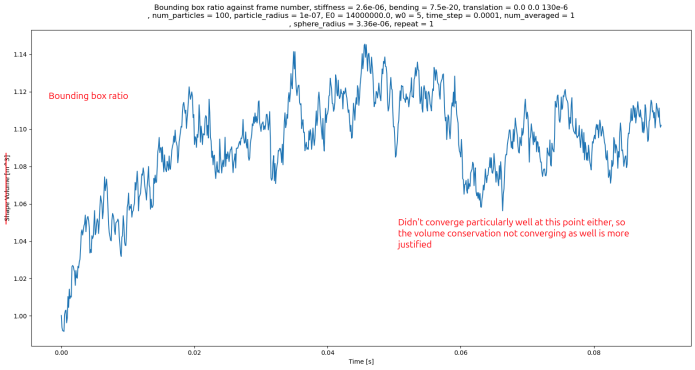


Running a test where I place ***N=10000 (=> roughly 50% or spherical checks)*** points in a cube of radius/half-width 1.0 (full width 2.0), I get roughly a ***volume of 7.9***, which is good match meaning the algorithm doesn't require surface points only, which is actually better for our case (however should not be too relevant, it just means that the solid case could also tested and any deformations on the

membrane won't break the calculation severely). Similarly testing a half-width of 20 gives a volume of ~63e3~(40^3) as it should. Similar results are also seen for spherical objects as expected, and results are even better when points are restricted to the surface alone (rad=1.0, N=10000, volume allowed => ~3.8 vol, surface only => ~4.18 vol, where 4.188 is the exact volume).



Using this method on the dynamics N-sphere membrane, I got the graph above. This sphere membrane has a radius of 3.36e-6m, hence has a volume of ~1.56e-16m^3. The plot shows an initial volume of ~1.5e-16m^3, which is pretty close to reality, however the main concern is how much the volume calculated will change from this initial value, and so far from the 100 frames tested the shape's volume dropped to about 1.45e-16m^3, which is only about a 3.3% volume drop. The next stage would be to test how this changes once the shape reaches its convergence too, to see if the main volume changes occur just as it is transitioning or not.

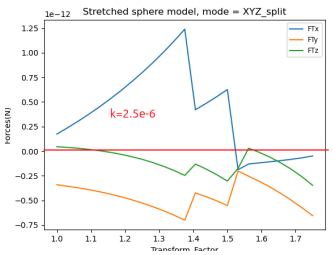


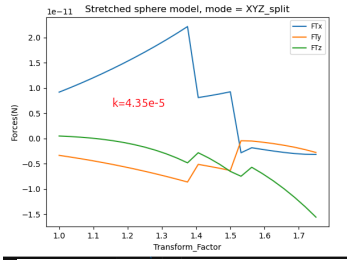
The plot for the bounding box ratio doesn't show great convergence at this point either yet, hence this more varied volume is not too problematic. I need to find a time where good convergence occurs in the bounding box then just perform a plot for this value. Note as well that the height/width ratio changes by ~10% here (to its stable-ish value), and the volume changes by roughly 4 to 5%, which is good since if the height increased without any width change then ***w^2*h~V=w^3*(h/w), where w^3~V_original, hence a ratio change here would correspond to a direct volume change, hence you may expect the same volume percentage change as the height/width ratio, but here we see that the volume change currently is about half the height/width ratio.*** This is not a perfect situation (there is still volume losses after all, when we would hope for no loss), however the loss is definitely less significant than the deformation seen could have gone to, hence some realism is preserved even on this fairly inaccurate data point tested.

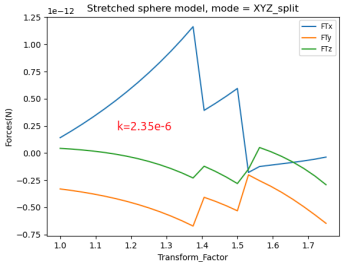
28/2/25

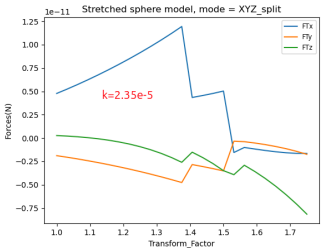
. Previous tests were not correctly using the equation in the paper, but were still interesting in their own right

- Not able to get zero force for actual paper version, partially due to the Z stretching not allowing too much stretching (vol conservation limitation likely)
- Other one did get crossing where Z moved much more freely
- Hence important factors are
    (1) Radial scaling with dimple in centre, expand outwards then scale back in after
    (2) Sufficient ratio of width to height so X,Y,Z trapping all coincide
- Non-conserving model also had some twisting effect, however this was small for the transformations tested (T<2, generally between [1.0, 1.6])

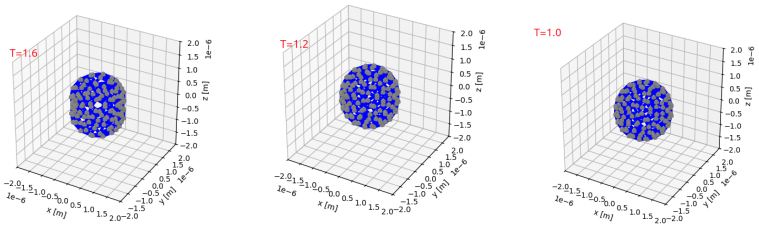




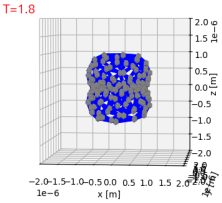




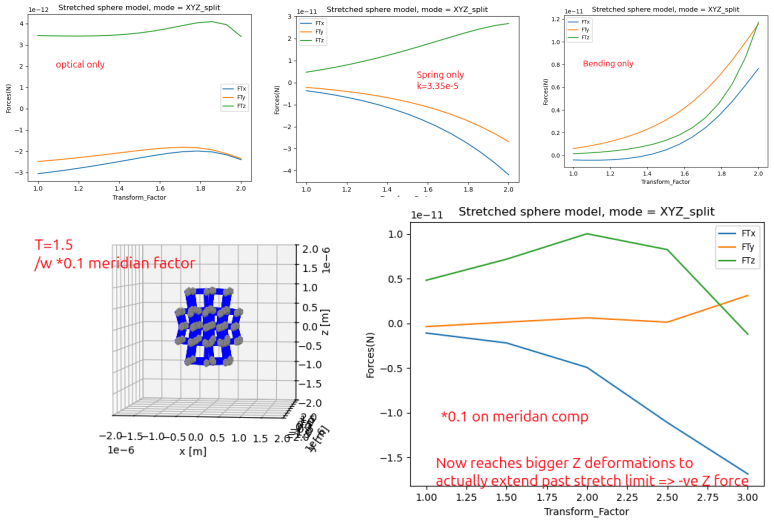
Attempted to get inverse area transformation working better, and so performed some tests for varying parameters to reach a zero force state, which was close to occurring but never quite occurred.



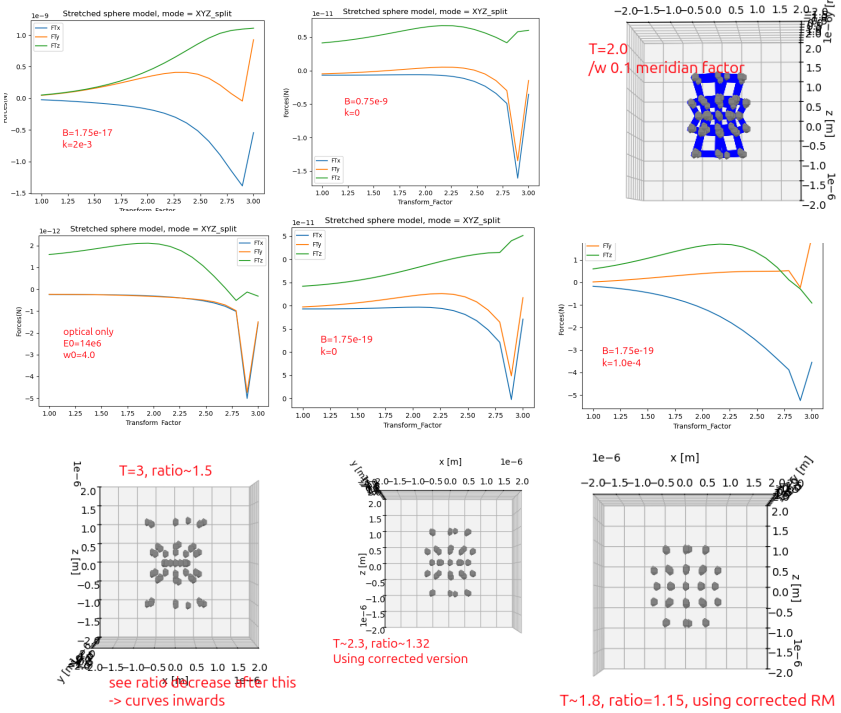
Revisiting the equation for deformation given in the paper we are considering, this was not correctly implemented first time around (meridian deformation added to Z component as opposed to theta hat direction when working in spherical coordinates).



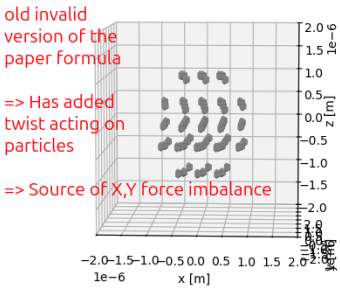
This was tested by considering the influence of each force contribution so it would be easier to find zero force locations, however trying to combine these functions for this effect proved to work particularly well.

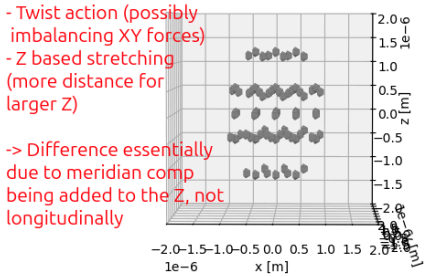


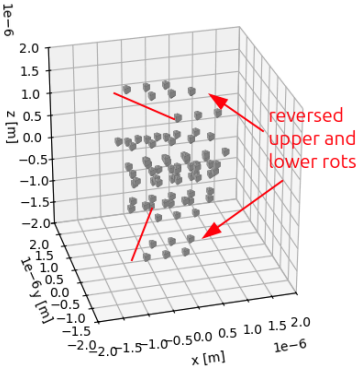
This new implementation now seems to working quite well, however zeros have not been seen so far, and when viewing the equation for large transformation factors you observe and extreme pinch in the centre of the object, which did not appear to applicable to cases observed in the paper, hence this seems like a low deformation regime only application. The Z stretches observed for low transform factor, T, here did not give the zeros hoped for and showed only very small height/width ratios.



I also went back to see how the previous implementation of the radial-meridian (RM) method I implemented was different to the corrected version, which essentially came down to a larger Z stretch seen in mine, with an equivalent radial stretch, however without the theta hat extension this lead to a twisting effect seen in the particles, which may have done work to balance/unbalance the X-Y forces. The key point is that a large enough Z stretch and a radial XY dependence is important to an optimal stretching shape.

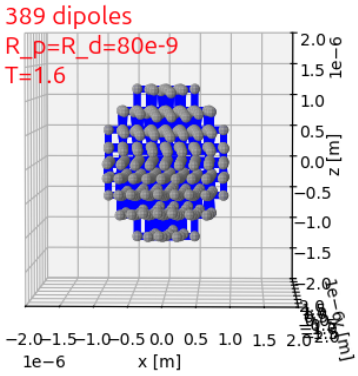


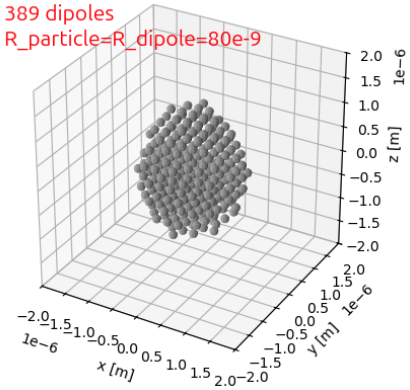




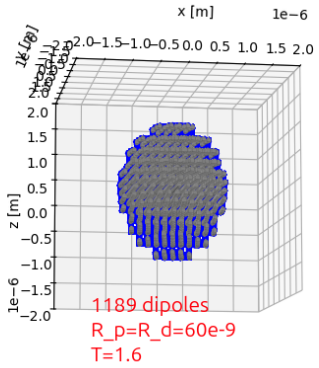
27/2/25

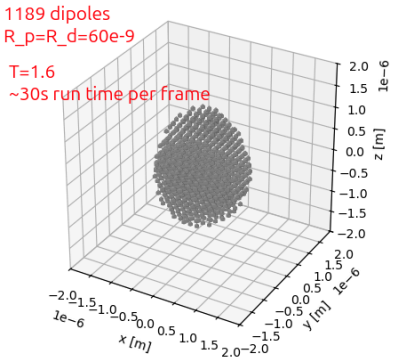
I wanted to test the T=1.6 system for higher particle/dipole refinements to see if this resulted in less initial motion when the particle jumped to a lower stretching factor.



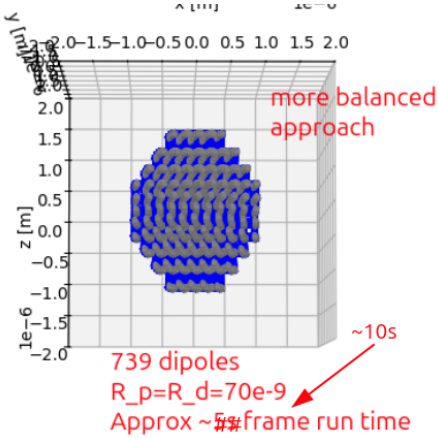


Testing at 80 and 60 nm particles and dipoles the cost to computational time was very large (as you introduce roughly 3-4x more dipoles), so a balance of about 70e-9 particle/dipole size gave a much more balanced computational time and simulation accuracy tradeoff.

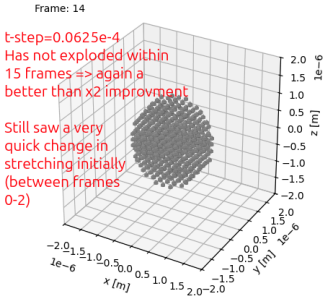


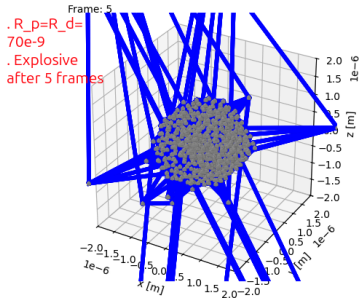


Testing the 70e-9 system at T=1.6 for time-step=0.125e-4 results in quick explosive behaviour again, as expected due to the particles now being far more compact again, taking only ~5 frames to explode. Testing this for 0.0625e-4 (6.25e-6) time-step results in …. Note however that we must be careful not to go below the approx 1e-7 lower bound, as past this we would expect more predictable ballistic motion for our brownian motion.



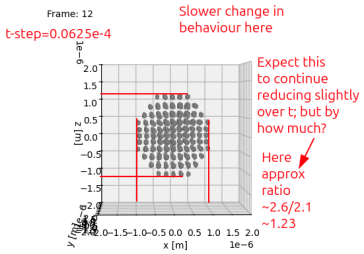
A more careful approach to seeing how this refinement would be to slowly increase the refinement and observe any improvements or different behaviour, which I will try if the 70e-9 case causes troubles when evolved for larger total times, however for now it appears to be fairly stable.

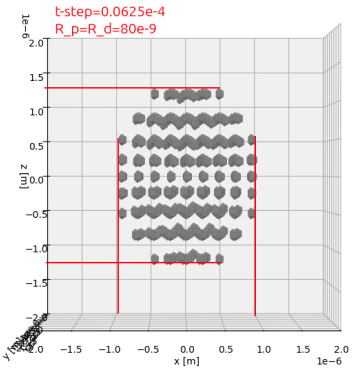


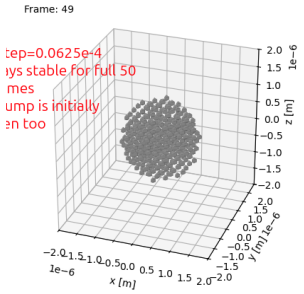


Considering time-step=0.0625e-4 for the T=1.6 case shows no explosive behaviour now for the

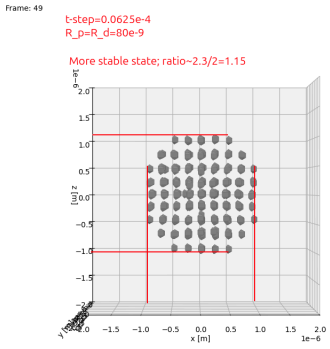
first 15 frames observed. We still see a jump here however, but it is perhaps less significant than before, but this is still very undecided as only a short time frame has been viewed. Considering this case but for 50 frames shows that explosion actually occurs just after the prior test, at ~23 frames in, so I would require a smaller step to consider this case further, therefore instead I will lower the particle/dipole size to 80e-9 and test this case to see if its behaviour is any better, as tests for smaller time-step now would be quite close to the limit and take very long times to run.





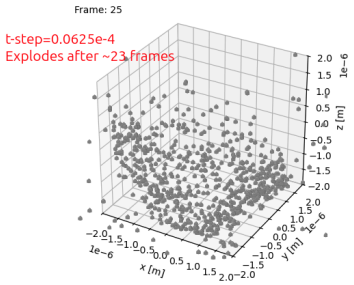


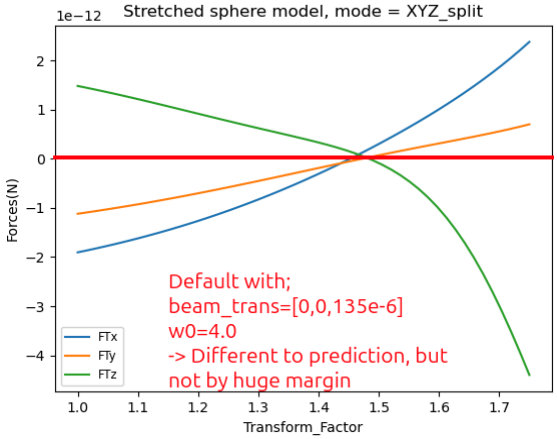
For 80e-9 particle sizes, we get stability for the 50 frames tested (at t-step=0.0625e-4s), and some initial jump again, which has resulted in a change of ratio from roughly 1.5->1.15. This is a similar result to what was seen in the 100e-9 case, which makes me believe this is not a refinement issue. It would be good to see how this stability changes for even larger times, however the program written by my partner considering dynamics did show plateauing regions at roughly 1.15 ratio too, which would stay stable for hundreds of frames, so it wouldn't be unreasonable to assume this is the behaviour seen here. That being said, the system he tested was for dynamics of a spherical shell, NOT a solid volume shape, so this could result in different behaviour but equally it is a good sign that there is agreement currently. Because of this, I will test this setup for much larger time steps to ensure I am getting the same behaviour, which would show that the two models (filled and membrane) do show some agreement here for a membrane-like particle being modelled (RBC). The springs constants are different here (2e-5 vs 3.35e-5) however so this would only imply the same behavior could occur for differently spring systems using the shell and solid models, and the particles were different sizes by a factor of 3x (3x larger for shell), ut this may not affect the stretching significantly, which should be tested.



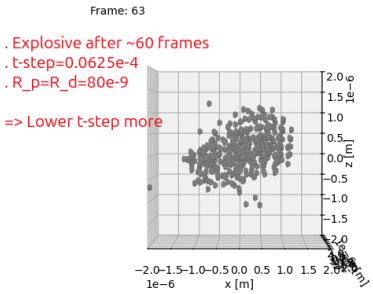
Also, going back to the previous force graph found I just wanted to make a comment about its stability, which is that the X,Y and Z forces are restoring, which is a requirement alongside the forces just simply meeting at a zero point, e.g. if they were perturbed they would return to the original state, as this transformation broadly acts to bring particles inwards (compression) in the X,Y axes and outwards (stretching) in the Z plane. Therefore a negative Z force for larger transformations implies it will compress after being stretched too far, and equally the opposite is true for small transformations. The X and Y forces also follow the same rule but in reverse, hence their opposite signs to the Z force make physical sense and are restoring too. However,

this analysis is slightly flawed in that compression and stretching does have radial dependence, so it is not as clear cut as increasing T results in compression along the X,Y axes, so there is room for unstable motion, however as an overall measure the average X and Y position should drift inwards and the Z average position should drift outwards with increase T, therefore I still think it is fair to argue this restorative force, so long as it is also acknowledged that will vary in the accuracy of its application for different points in the system.





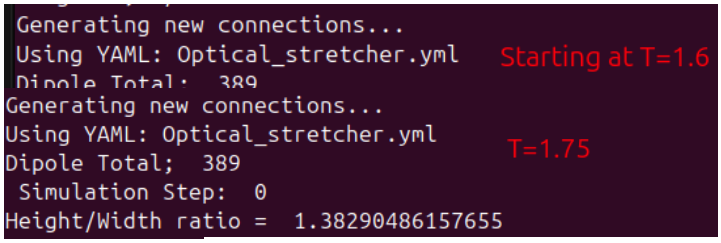
I have now written a function to get the bounding box for my transformed object and so get the width to height ratio (by linearly averaging the X and Y axes for the 'width', then just the Z axis for the 'height'). Doing this gives a ratio of ***1.29 for T=1.6, 1.38 for T=1.75 and 1.15 for T=1.28***. Using this, I would like to view the dynamics for the T=1.28 case and see if it jumps initially or not (since the other system jumped and settled to this value).

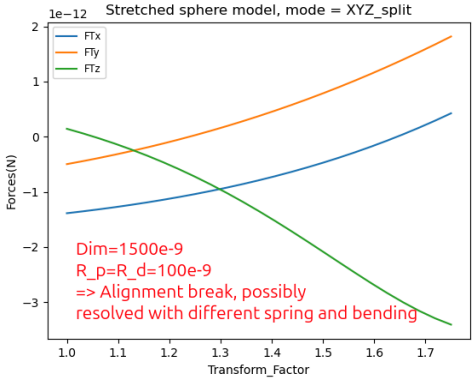


Changing the size of the object without changing any other parameters does break the alignment found, which at first seems unusual since the spring properties of the system should get better with refinement, which could equivalent be considered as the system be reduced in size (assuming the)

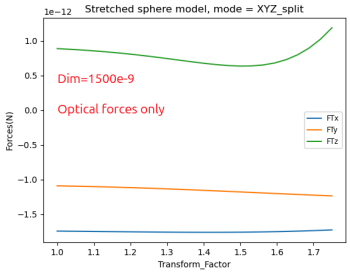
(1)

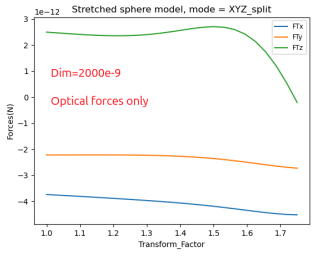
(2) Try running from starting point of stretch factor =1.15, → Need to find the T this is for

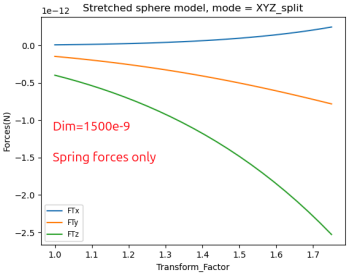


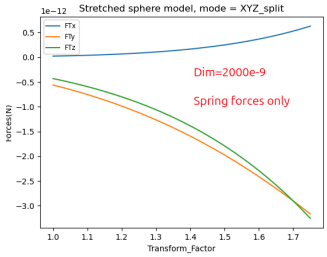


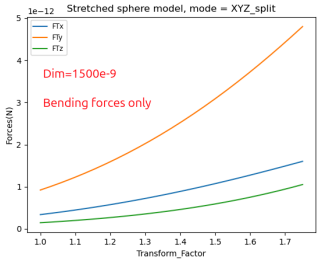
Considering a different sized object, I get the following plots for each force source;

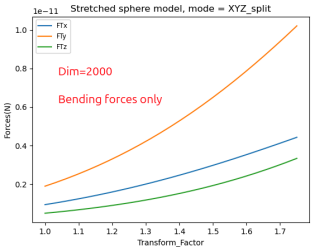


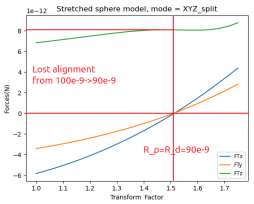


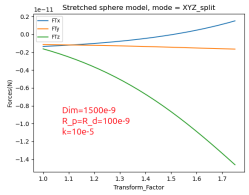


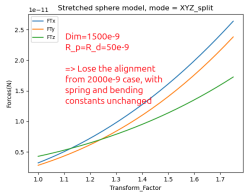


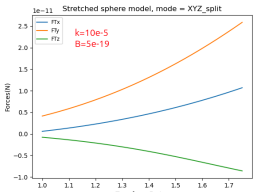


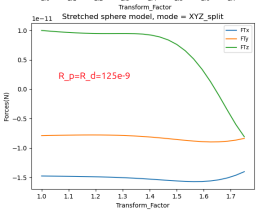


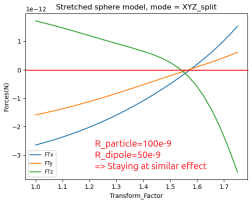




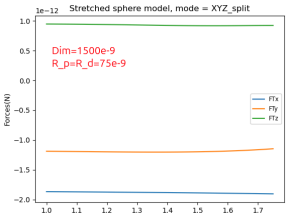


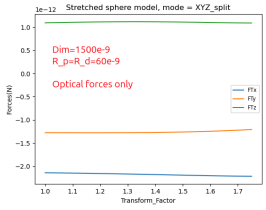


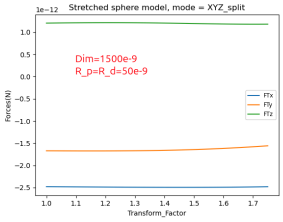




The issue I was having was that smaller object sizes changes (particles and dipoles unchanged) had quite different behaviour to the originally tested 2000e-9 diameter mesh. Considering the force components on the left (generated for B=0.75e-19, k=1.5e-5, R_p=R_d =100e-9), we see that the bending is has the same behaviour but just scaled magnitude (more dominant in the large system), and the spring forces have somewhat similar behaviour, with the exception that the Y component of force becomes more aligned with the X in the larger system (this may be due to the larger system have greater resolution, as the dipole/particle sizes were not changed between these two). The optical forces however had very different behaviour for the two systems, which suggests the scattering has some significant difference for different sizes. It is true that the particle/dipole size not being changed between the two would also affect this behaviour, however since the Gaussian beams are very diverging and far apart here the E gradients should be shallow and certainly no fringes are present, hence this loss in resolution shouldn't change the behaviour observed this drastically I would presume (81 dipoles in smaller case, 179 in higher), which can be tested by running the smaller case with smaller dipoles/particles again to match the 179 seen above, where we see that for

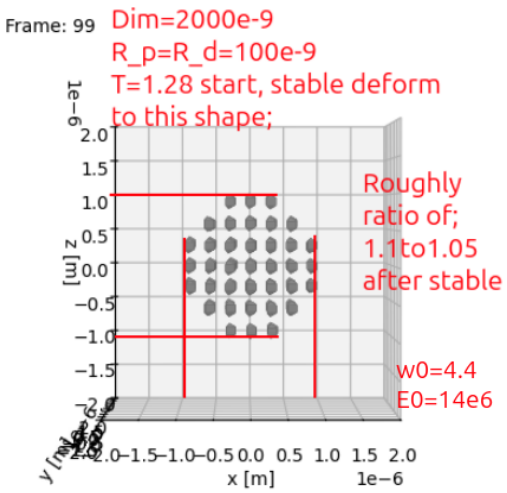


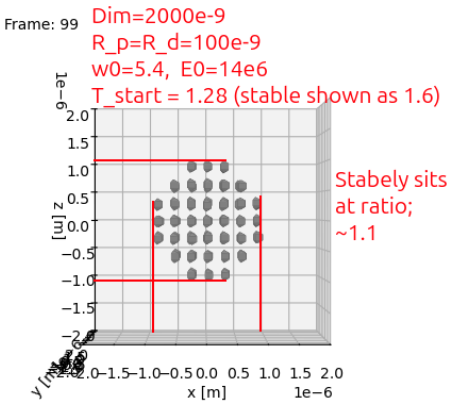


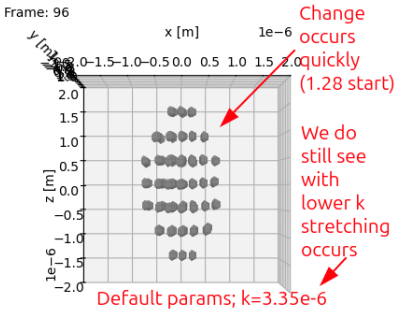


R_particle=R_dipole=75e-9 we get 179 dipoles for Dim=1500e-9 we see the behaviour flattens out but is resistant to changing its turning point as seen in the different sizes.

Setting T=1.28 for default parameters with w0=4.4, E0=14e6, which gives a zero at the transform factor=1.28, I see a small jump again to a lower stretching factor, but it tends to a value slightly smaller than that of the previous example (w0=5.4, E0=14e6, starting at T=1.6). If instead I start it at T=1.28, but in the system where T=1.6 was supposed to be stable (but shifted to T=1.28 equivalent factor after dynamics), then we continue to see this stretch factor jump down to roughly 1.15 again. Considering these two cases, we see both particles tend towards approximately the same deformation despite the stable point shown being very different. Time-steps used here are 0.0625e-6. To check nothing is broken with the stretcher I ran the simulation with k/10, and did see the stretching occur at a pretty fast pace ***(from 1.28 to 2=3/1.5 within 100 frames, hence 0.0072 stretching factor increase per frame, as oppose to the decrease before of roughly 1.28->1.1 =>-0.0018 per second, which is 4x slower)***, hence this slow pace seen normally implies that values chosen are close to equilibrium, with the exception of the initial jumps which suggest they are far from equilibrium but not too far as to cause explosion from too rapid movement.







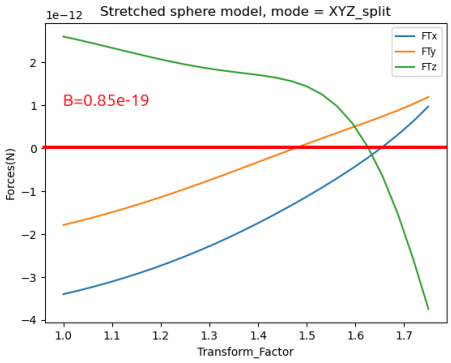
Another key point to mention here too is that the equilibrium state is given as the cubic lattice forming the sphere, and this new transform factor introduces a lot of ‘shearing’ effects where non-axial transformation occurs (bunch.spacing radially), hence when allowed to dynamically deform it would be reasonable to assume the shearing effects could quickly be removed from many connections within the particle (e.g. its solid nature) resulting in structure quickly deforming back out to a more cubic shape, and so here having a different (more linear style) force plot. These plots tell us the total force is zero however internal forces may be cancelling but also suggest particles back towards the lattice structure, as either end of the region measured will have unbalanced surface forces which could cancel. For this reason it may be the case that predicted flexible stable spots work in principle, but not under a dynamics simulation

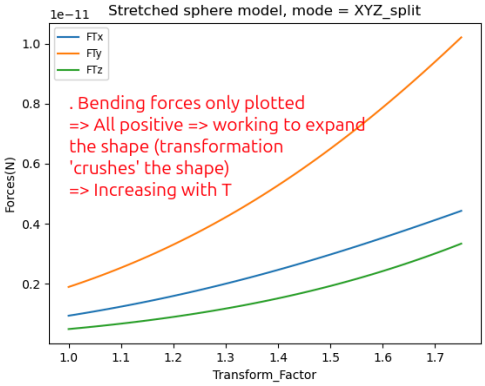
modelled with a solid internal structure as these forces create an overwhelming push back towards the lattice structure, which breaks the deformation quickly and so results in the linear deformation being observed again. This suggests that perhaps the membrane approach has significant advantages over the solid approach for situations in which a membrane is expected, especially since physically it would be impossible for opposite sides of the membrane to instantaneously transfer forces, which is not strictly true for the solid simulation run here (it would still have to pass to intermediate connections, but this still allows a far shorter path than the alternative of travelling around the membrane surface).

1. ***Test for more relaxed spring constants, see if you can reach a value where it reaches the expected stability (height/width ratio=1.33) for 110mW beam, then consider the plot and see how it looks, see if the force does have a zero meeting point.***

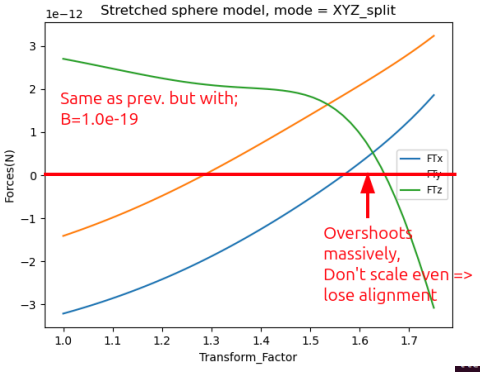
26/2/25

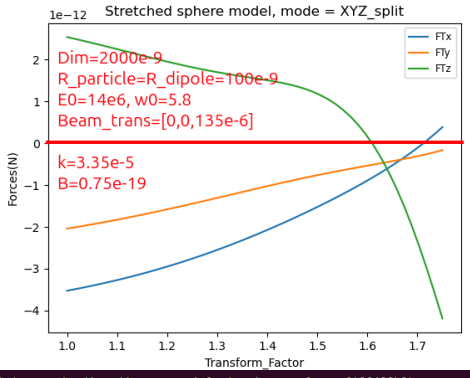
Work was done on the poster yesterday this morning, but has now been submitted, hence for the final few days of practical work my partner and I can continue to investigate this deformation and look for further agreement between the simulated values and real observations.

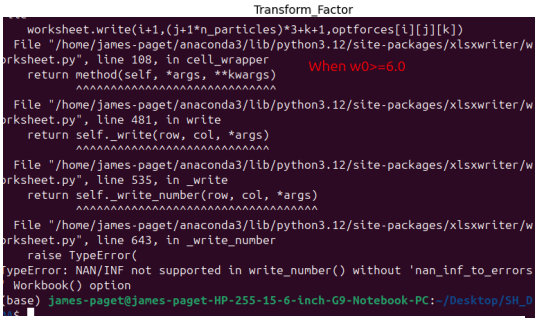




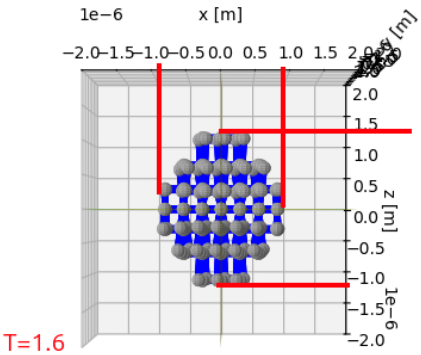
With this new deformation regime, I want to now see what effects the bending forces have, as this is not a simple linear scaling only along the XYZ axes, hence the bending force will be non-zero here. Doing this shows the bending is strictly positive here, as the transform compresses the shape in all axes overall (considering interior forces from connections, hence Z still positive [but less so than others] despite the shape still resulting in stretching).



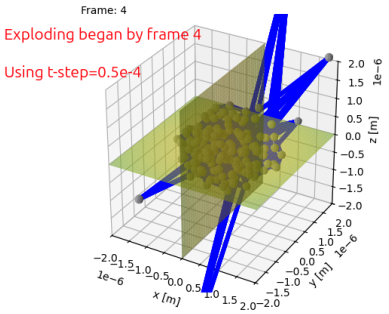


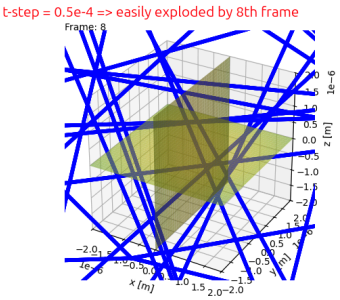


The previous tests showed that for very high transformations the forces would meet but at some negative force instead, hence this could help resolve this issue, however the scaling of the and X and Y forces are not equal, hence the Y force would be inclined to very quickly overtake the X force, so a meeting point at zero is not likely if this bending id changed further, which is supported by the following plots seen whereY gets too large too quick. It may be possible to remedy this with a larger w0 which appears to possibly scale X faster, however for w0 values >= 6.0 the simulation crashes due, which I

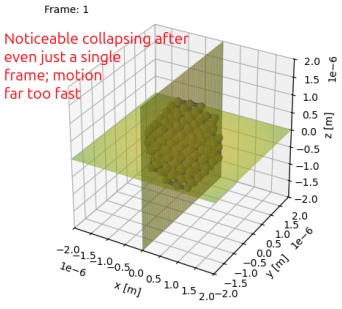


believe is due to some extreme forces breaking certain particles at high transformations, hence you get NAN errors.

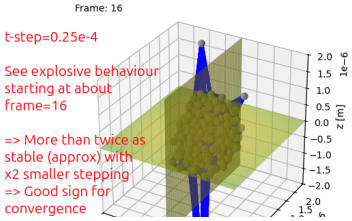


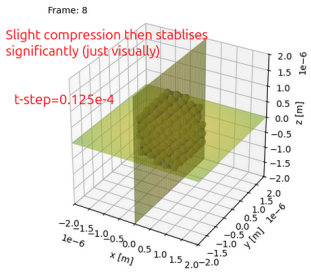


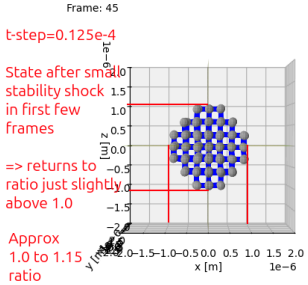
Therefore, considering the largest value we could hit zero at (~T=1.6) we get a ***height/width ratio of ~2.8->3.0/2.0 = 1.4->1.5***, which is actually just over 1.33 we were looking for. Looking back at the ***T=1.75*** reading I took, I believe I may have underestimated its height to width ratio (not measuring the height from the very edge as I did for the width) and also interestingly this plot appears to have less stretching than the 1.6 case. In any case, this value approximately where it should be for what was given as realistic deformation, hence I would like to view the dynamics for this system, perhaps with some extra refinement and low time-step since the object is solid with particles, hence will very easily explode.



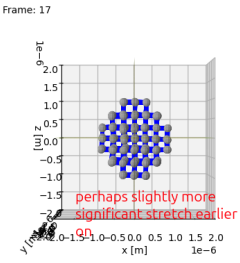
At time-step=0.5e-4, we see very fast collapsing and explosive behaviour very quickly (~ 4 frames). For 0.25e-4 time-step, this only begins at 16 frames, hence it appears to show greater stability. Note that these tests were performed with original resolution e.g. 179 particles & dipoles, hence this could also be refined for more accuracy. At time-step=0.125e-4 the system did not explode in the 50 frames tested (more than double the 16 exploded before, hence last past 32 frames for good stability, as seen), and visually appear very stable throughout this motion, bar the original jump seen in the first few frames which lowered the ratio to roughly 1.0 to 1.15.







These tests used default params; E0=14e6, w0=5.4, Dim=2000e-9, R_particle=R_dipole=100e-9, beam_trans=[0,0,135e-6], k=3.35e-5, B=0.75e-19.



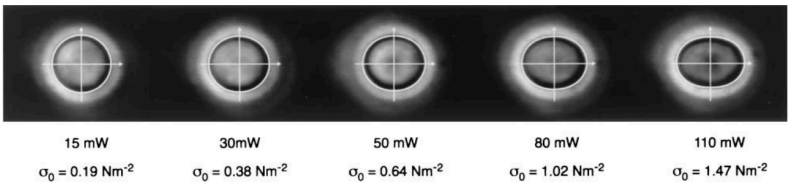
(1) Run dynamics tests at T=1.6 for higher dipole/particle refinement
(2) Keep testing for smaller time-steps
(3) See if it will ever tend to a slightly larger stable point here

24/2/25 - 25/2/25

If we want to calculate the beam power used, we can consider;
I=0.5*c*permitivity_free*(E_0)^2
P=0.5*I_peak*(Pi*w0^2)
Then with E0s of the order 10^6 (as used in our program), then we get roughly milli-watt powers out (w0=lambda*program_w0, as a relative version is used here, with lambda~10^-6), hence we can see that for roughly E0=1.4e7 we get 100mW power, which is the power range used for recent experiments (20e6 used in my tests yesterday and for my partner's tests of dynamics yesterday, which both seemed to be giving very reasonably stretched deformation as their stable points). This lets us ensure we are using beam values similar to that of the experiments we are trying to replicate.

https://phys.libretexts.org/Courses/Bowdoin_College/Phys1140%3A_Introductory_Physics_II%3A_Part_1/07%3A_Electromagnetic_Waves/7.04%3A_Energy_Carried_by_Electromagnetic_Waves

The paper gave the following deformations in lower-power regimes, where each deformation had approximate ***width/height ratios of 1.03, 1.05, 1.09, 1.22, 1.33 for each power rating***, or alternatively were predicted through the use of the equation given (overlayed in white in figure).



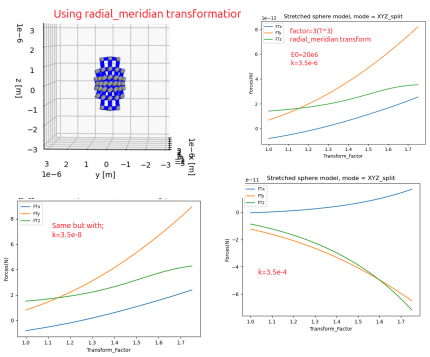
Using Eqs. 10, 11, and 12 and the explicit form of $\sigma_r(\theta)$, we find the following expressions for the radial and the meridional deformations of the membrane,

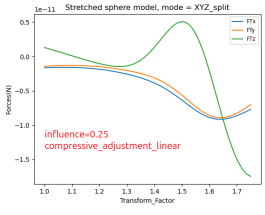
$$u_r(\theta) = \frac{\rho^2\sigma_0}{4Eh}[(5+\nu)\cos^2(\theta) - 1 - \nu] \qquad (13a)$$

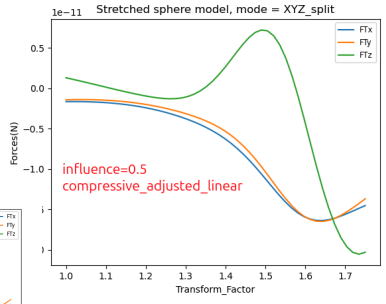
and

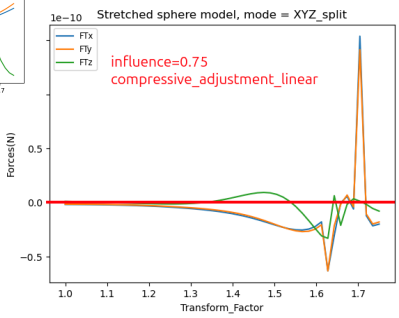
$$u_\theta(\theta) = \frac{\rho^2\sigma_0(1+\nu)}{2Eh}\cos(\theta)\sin(\theta). \qquad (13b)$$

An important feature of these shapes is that the shape appears to bend more into a cuboid shape, e.g. less deformation at the high radius points in the perpendicular plane, hence this adjustment should also be tried for a volume conserving transformation if possible. I also would like to explicitly use the equation form they gave too to see how this performs (should perform the best according to these results).

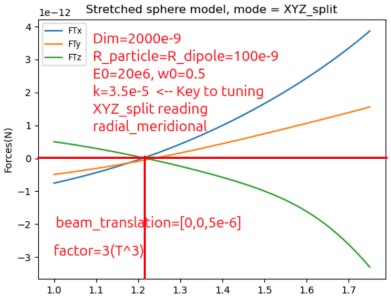


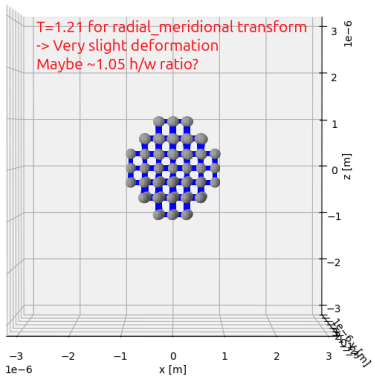




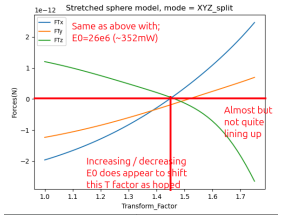


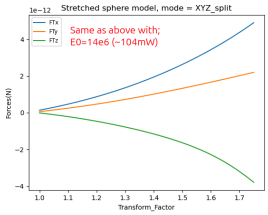
Using the transform from the paper, with beam strength E0=20e6 => beam power P~208mW, where a zero is seen at a transform_factor=T~1.21 which would give a const (=stress0)=3(1.21^3)=5.31. Note as well however, that we are considering a particle of width=2e-6, compared to the paper's original 6e-6 (unstretched diameter). This change would result in a surface area change of 4piR1^2/4piR2^2=(R1/R2)^2=9x difference, and since stress=F/A, then for the paper's shape S_paper=F/A=9*S_sim if the forces are the same (same beam power used). This makes the stress used for T=1.21 => stress0=5.41 now when scaled more like 0.6, which is much closer to the values seen in the paper (ranged between 0.19 to 1.47), which would align with an approximately 50mW laser. Here I was using what equated to a 208mW laser, but this may be able to be corrected through a change in the spring constant.



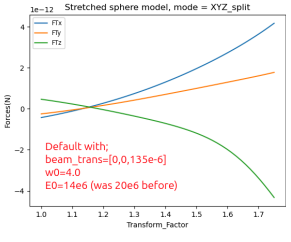


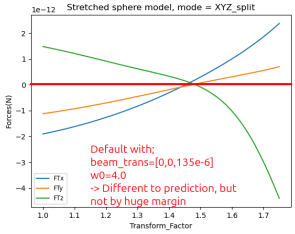
Beam foci may be at +-60 micrometres OR (more likely) 100-130 micrometres due to the diverging beam starting at w0=2.7 micrometres, then doubling to 5.4 micrometres at the cell, which is 60-70 micrometres away from each optical fibre, hence the focus can be linearly interpolated back to 130 micrometres from the cell in each direction. Therefore I want to try to move this beam centre out to this range, which should have the effect of lowering the effective beam power too.

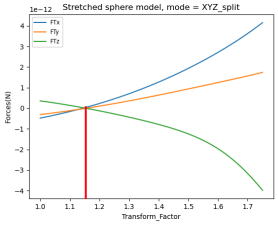


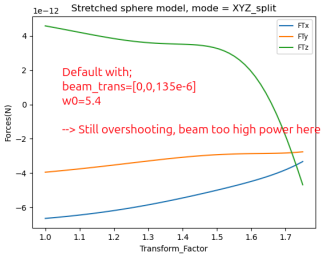


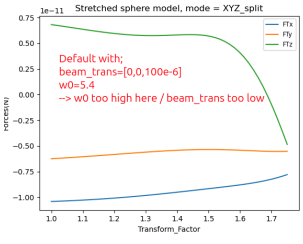
When moving the foci out to 15e-6m, we see that this has the effect of reducing the E0 of the beam and that by increasing w0 as well we can get the same zero force location and total force.
→ Specifically we saw an x3 beam_trans => ~>x2 w0 to balance out the zero force transformation point. With this, going from beam_trans=15e-6m sep -> 100e-6m would require w0=1.0->6.66*0.6=4.0, which is much closer to the w0 and beam_trans expected. In fact, If we wanted to get to 20=5.4, this would require a multiplier of 5.4/0.6=9 => 15*9=135e-6 beam translation separation, which



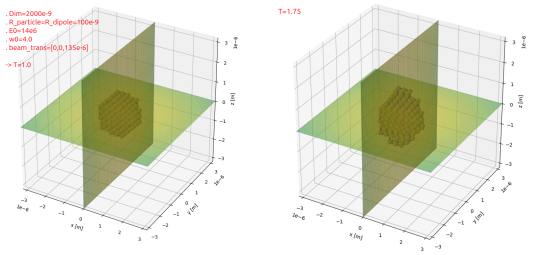




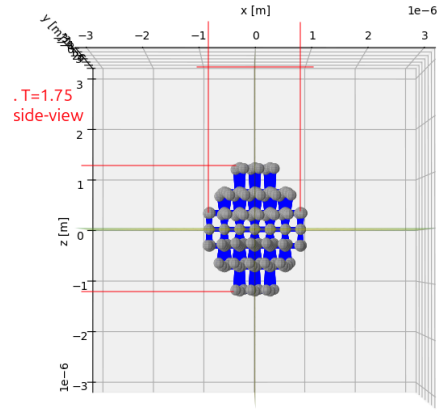


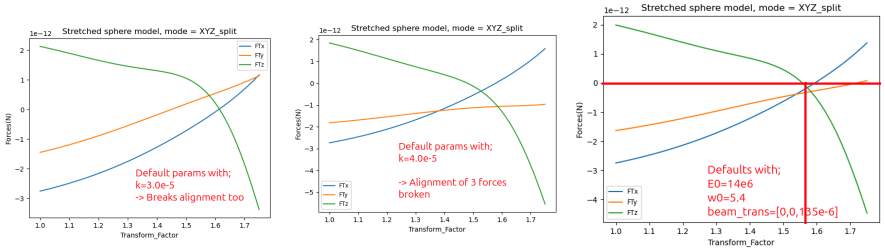


was the prediction made if we interpolated the beam focus backwards (of course here I have assumed the w0 and beam_trans share a ***linear*** scaling). When running this case, we see the scaling is not quite linear, hence it works for roughly w0=4.0 not 5.4, and with the beam strength E0 being reduced to the required 14e6 (for~100mW) rather than prior 20e6 this brings the beam back to the original zero point.

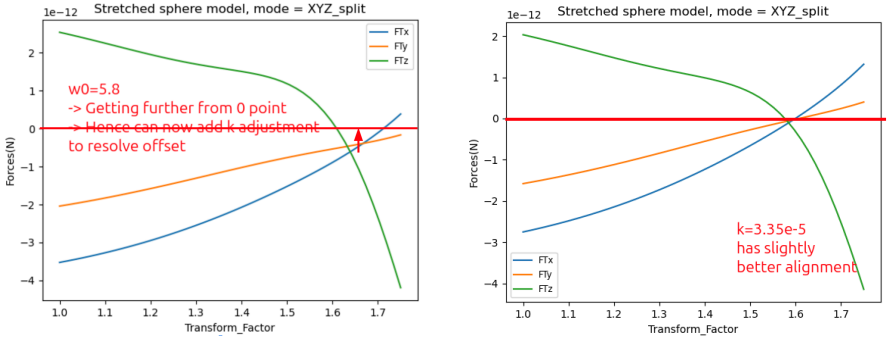


Now I want to tuning the spring constant ***k and the beam_trans, w0*** to match the transform factor to that seen for the 110mW beam, where it had a ***width/height ratio=1.33*** (opposite here as the stretched along height, Z, not width, X, as done in their experiment). Considering the 1.75 case, this has an approximate height/width ratio of 2.4/1.9=1.26, so I would want a cross at a value just above here (or slightly larger, with care that the stress scales as T^3, hence a small increase in T would give much larger results) in order to match the 1.33 deformation seen for the experimental particle as a sort of calibration of the system to this case.





k=3.4e-5 aligns X and Y forces more, slightly off from Z but close.

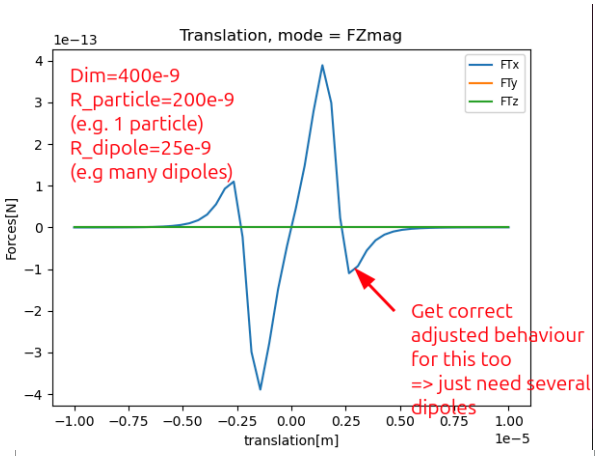


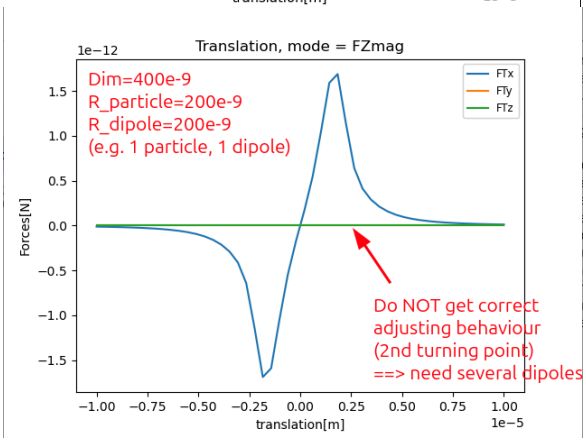
(1) Test bending force values effect.

(2) How is this transformation different from an affine transform; More of an explanation of just additional radial scaling.

23/2/25

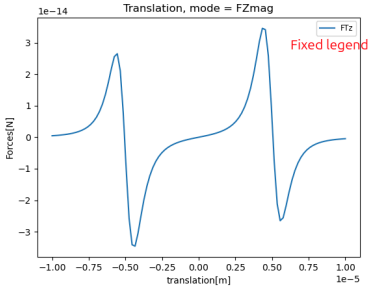
Yesterday I saw a single dipole show slightly incorrect behaviour, which was then fixed through a small collection of particles moving. I now want to check to make sure everything calculated there was correct, then extend this to bigger groups of particles to see if it still applies, and if so then I will try it for stretched sets of particles to see if it continues with this behaviour OR if the stretching causes something new to occur. If it does continue with the stretching, then I should be able to simply get a plot out for the forces cancelling (Z forces at least, then try to extend to X and Y) once stretched to some critical factor, then being pulled back after that. With this information I should find that with a certain aspect ratio of deformation I get closest to equality in the forces, which would mean I had found the equilibrium state for stretching here, which I can then compare to the values found in the paper we were considering. We can compare how this matches to these values for varying beam powers and spring constants, which will inform which constants gave realistic results (calibrating our simulation), which could then be further investigated through comparisons of other tests involving RBCs from different sources to make sure they agree.

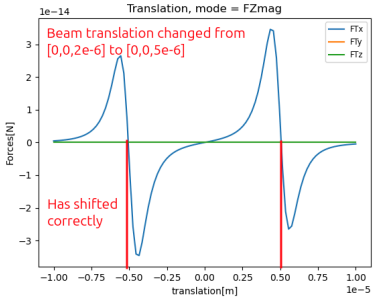




Considering a different range of particles and dipoles, it becomes obvious that the only requirement is that there is more than 1 dipole present for this effect to be seen, and that it becomes stronger with more dipoles present (larger secondary turning points).

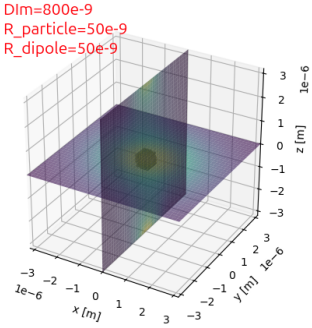
NOTE; The legends in these figures are wrong, this is due to me making quick change to add the "FZmag" forces reading. This is fixed by removing the labels we usually had for "FTx" and "FTy", which were just using zeroed data, hence the flat lines. Now that we have observed that this is true for

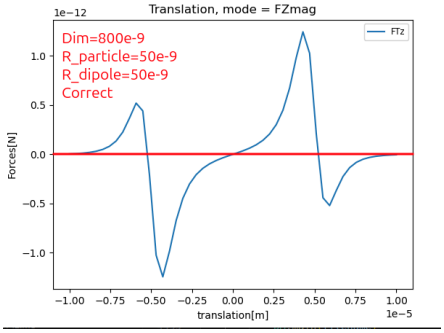





```
case "FZmag" | "CZmag":
    output = np.zeros(3)
    for p in range(int(len(value_list)/3)):
        output += [value_list[3*p+0], value_list[3*p+1], value_list[3*p+2]]
    output = output[2]
    data_set[params_i*num_expts_per_param + expt_i, 1, i] = output
```

Takes just Z reading

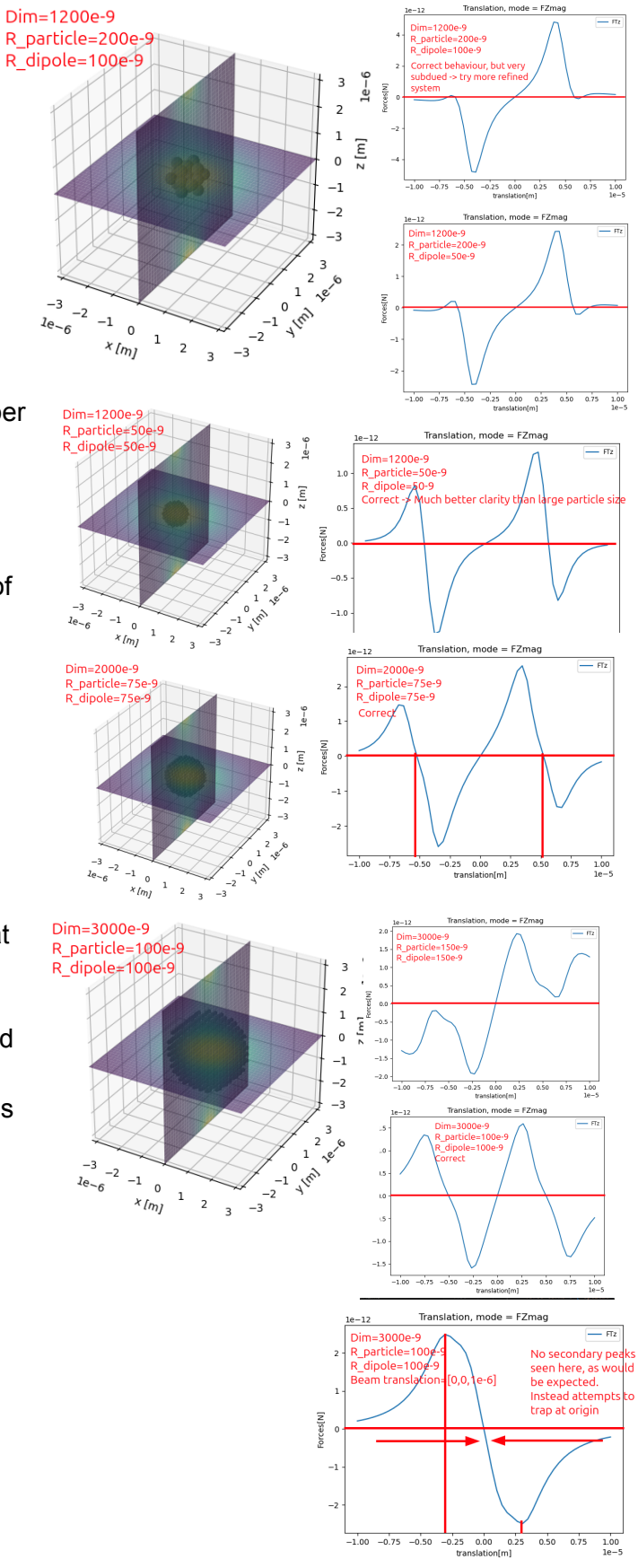




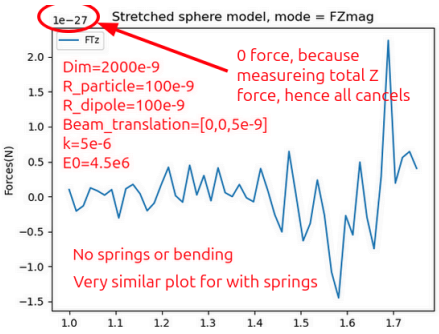
smaller systems, I will do the same for increasingly larger systems to observe when a change in behaviour may occur. Using ***beam translation=[0,0,5e-6]*** for following graphs.

This last graph for the ***Dim=1200e-9*** case really highlights the importance of having enough resolution for the sphere to be accurately portrayed, as just having enough dipoles is not enough if their placement is not representative, hence we need an equally small particle number in order to see proper secondary peaks (even if that means having 1 dipole per particle, which wasn't a problem as seen in the clean results).

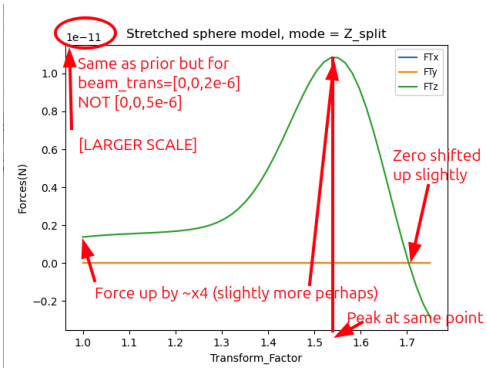
Now I want to consider how stretching is affected, given that we know that particles of essentially any size in a trap like this (given that the particle is smaller than the trapping width, from beam offsets) will be pulled towards the Gaussian it approaches, and pulled after surpassing t too (hence force reversal). The prediction is that, like a large particle, a stretched particle will want to be kept in at the origin, but locally should have parts of it which are attracted to the Gaussian, which should reach equilibrium at some point. I will test this first for ***Dim=2000e-9, R_Particle=R_dipole=100e-9***, which should give a good mixture of being quite accurate for translation already and quite fast, but has room to make the simulation more accurate for some slower but possible tests.

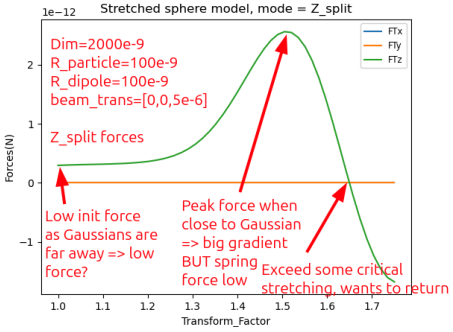


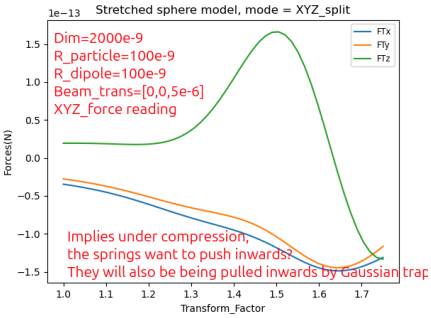
Measuring the Z force on the upper half of the stretched sphere, we see a curve that makes sense with what we have observed, but the main question is as to why the force on the particle in its unstretched state is so small compared to the critical point later on. I theorised that this might be due to the particle having points further from the Gaussians here, hence less field gradient is present, but I will need to test this by moving the offset inwards to see the impact.

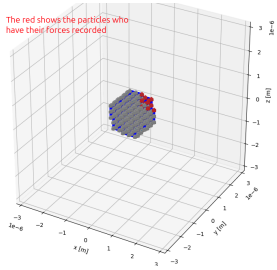


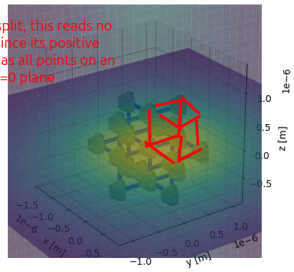
Changing the beam offset from Z=+-5e-6 to +-2e-6 resulted in the forces becoming approximately 4x bigger and the zero point becoming slightly larger. The first point could be attributed to the inverse square law for roughly 2x closer Gaussians giving roughly 4x larger forces, and the second point implies that the system becomes stable for a longer stretch than before, which makes sense as the force has become larger with a power of 2 relationship, whereas the springs are still affected by a linear (power 1) relationship, hence equilibrium is reached a bit closer the beams' foci (in the second case the particle would stretch to 2000e-9*1.7=3400e-9 => 1700e-9 radii, for 2000e-9 beam placement, hence just shy of beams, but in first case would have been at a radii of roughly 1650e-9 with foci at 5000e-9, hence much further away, so surprisingly little change from such a dramatic shift, which implies the spring constants are quite strict for motion here as they dominate over E field term, unless the E field is just very constantin its Z axis which it didn't appear to be when running the small particle translation tests).

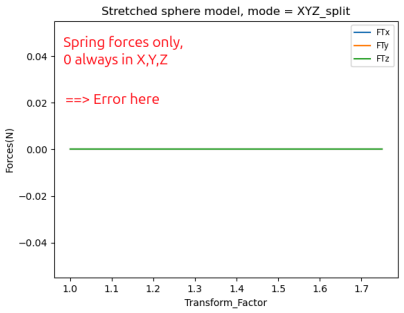






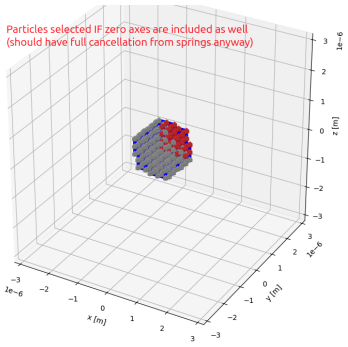


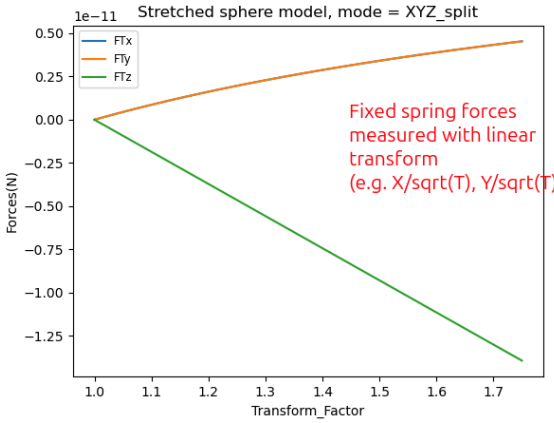




Testing the XYZ forces on the purely positive octant of the sphere gives the result seen, where the X,Y forces are actually purely negative for the full region tested, which implies the Gaussian trapping is stronger than the spring forces always. To test this, I did a plot for spring forces and

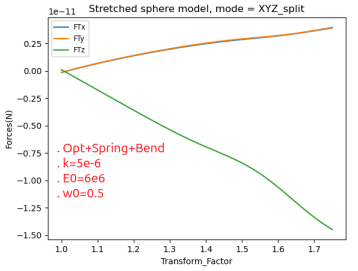
found it had been set to zero for some reason. I found the same result when testing it with singular stretching too (Transform=[x*factor, y, z]). One possible answer to this is that for the particle set I test each will feel an equal and opposite force from the other, hence they could all cancel, however you may then expect the outer particle to give the imbalance in results. Following this back to the main simulation() function, the spring forces here also so [0,0,0] at all times, which showed that the matrix arguments were not being written to the YAML, hence the default value was being used (to make every system in equilibrium).

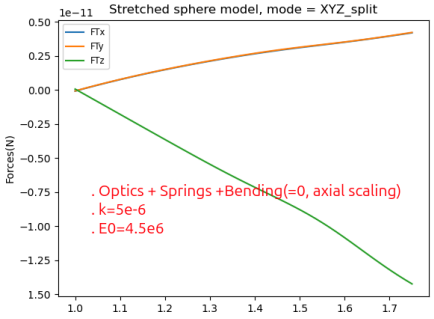


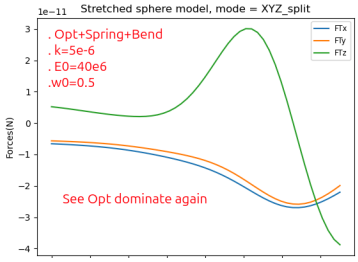


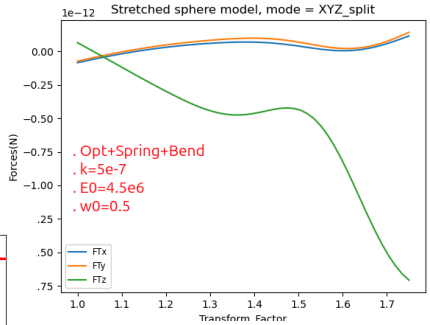

```
Progress; 0/50
Generating new connections...
Using YAML: Optical_stretcher.yml
Dipole Total;  179
 Simulation Step:  0
========
     for at p=89: [0. 0. 0.]
     for at p=90: [0. 0. 0.]
     for at p=91: [0. 0. 0.]
     for at p=92: [0. 0. 0.]
     for at p=96: [0. 0. 0.]
     for at p=97: [0. 0. 0.]
     for at p=98: [0. 0. 0.]
     for at p=99: [0. 0. 0.]
     for at p=102: [0. 0. 0.]
     for at p=103: [0. 0. 0.]
     for at p=104: [0. 0. 0.]
     for at p=106: [0. 0. 0.]
     for at p=107: [0. 0. 0.]
     for at p=126: [0. 0. 0.]
```

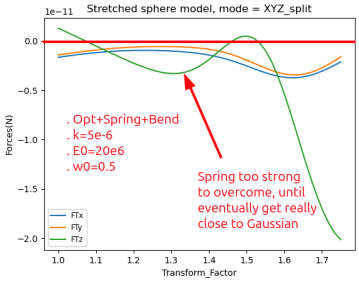
Now that these spring forces are fixed, I will remake the plots above with the corrected forces; Note as well that this means the plots above were for optical forces only, and this plot here was for spring forces only, hence it is clear that the spring forces here far outweigh/dominate the optics (10^-11 Vs 10^-13), so I should lower the spring constant.

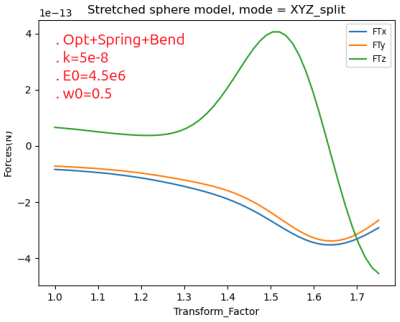


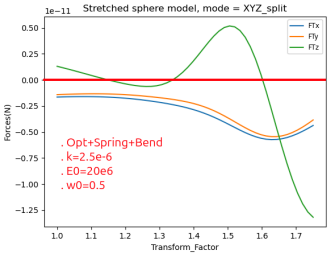






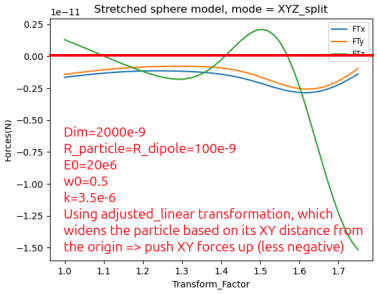




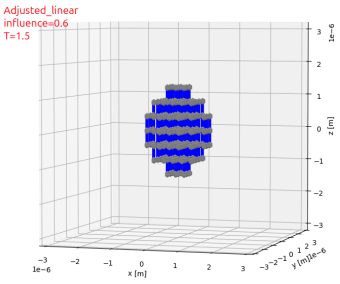


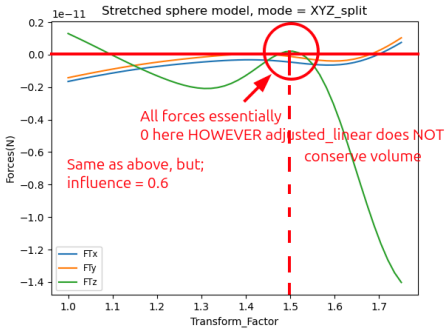
Doing this process does lead to solutions closer to a zero, but despite careful tuning does not appear to show signs of the Z and XY forces equilibrating at the same time, which is perfectly within reason and expected an incorrectly guessed transformation (e.g. this linear volume conserving regime), hence we must search for another volume conserving scaling to investigate. From this experiment, we

can see that when the Z force nicely reaches 0 force, the XY forces still acted negatively (e.g. compressing) from the Gaussian trapping overcoming the outward pressure of the springs, hence we should consider wider XY parts to reduce influence of Gaussian beam pulling (which will also increase spring pull, however this should be of lower order => net positive movement up/larger XY force force).



Considering an ***adjusted_linear*** transformation, which simply adds additional XY transformation to points based on their XY distance from the origin (not Z dependent), hence is non-volume conserving, results in a point in which all forces are extremely close zero (could essentially be matched through further slight tuning).





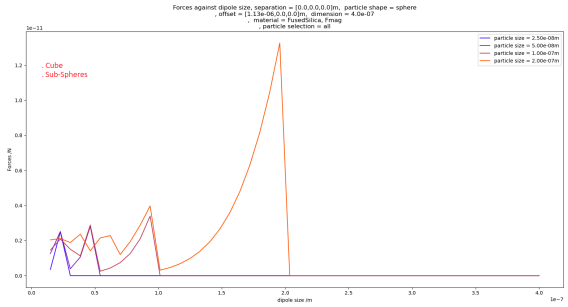

```
tch transform_type:
  case "linear":
      # Linear transform
      transformed_coords_list = coordinates * [1/np.sqrt(transform_factor), 1/np.sqrt(transform_factor), transform_factor]
      return transformed_coords_list

  case "adjusted_linear":
      # Linear transform with some alteration
      # NOTE; This does NOT preserve volume, you could add this though by scaling the Z down by rad_factor^2, but this would likely cause more problems
      #   --> Need a function with positive and negative integral contribution => Takes away and awards volume in XY plane without influencing Z plane
      mesh_unstretched_radius = 1000e-9   ##### HARD CODED FOR NOW -> Fits current test, see if works then implement fully ######
      influence = 0.6     # Acts to inflate the transformation in the XY plane
      transformed_coords_list = []
      for coord in coordinates:
          rad_factor = 1.0 +influence*(transform_factor-1.0)*(np.sqrt( pow(coord[0],2) + pow(coord[1],2) ) / (mesh_unstretched_radius))
          transformed_coords_list.append( [rad_factor*coord[0]/np.sqrt(transform_factor), rad_factor*coord[1]/np.sqrt(transform_factor), coord[2]*transf
      return np.array(transformed_coords_list)
```

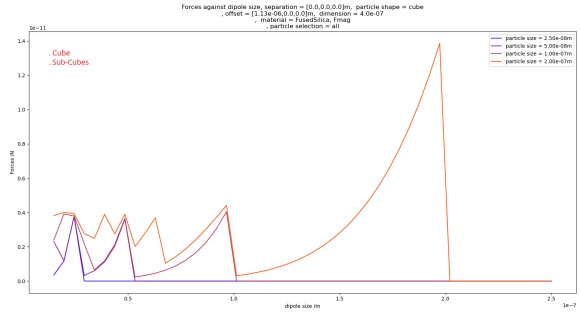
I should now;

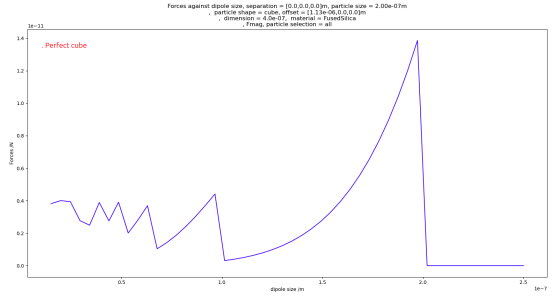
1. ***Revisit the paper to see exactly what stretching ratios were stable for them, construct volume-conserving functions that pass through these values, then evaluate how they perform.***
2. ***Vary the E0 and k values to test whether each of these systems works better for different or similar k values, and how the stretching wanted changed with E0 (can I stretch regime give agreement for all E0s chosen).***

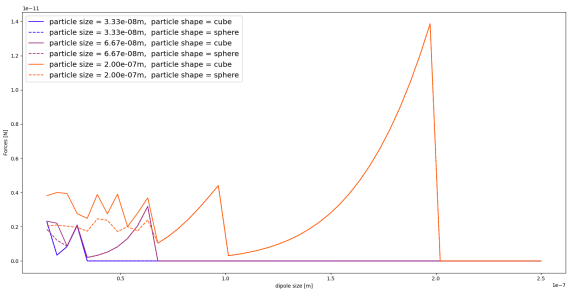
22/2/25

I have continued working on one half of the poster, covering some topics from both verifying that the bead-and-spring model works (cube refined into sub-cubes and sub-spheres with varying dipole numbers, as well as single dipole testing using RR/LDR) and some information on the dynamics simulations we could perform with this validity known (e.g. meshes made of shells with different patterns interconnected, lattice of connections, how the time stepping worked in terms of low Re number regime with drag DE formed + Brownian motion), as well as working on the conclusion of our project overall.









. Force experienced by a cubic mesh simulated using spherical and cubic particles of varying size

. Using 'squish' regime,, therefore place as close together, leave gaps on outside if space remains from particle size

. Sat at 1.13e-6 in LG beam (on high intensity)

. Total force on mesh

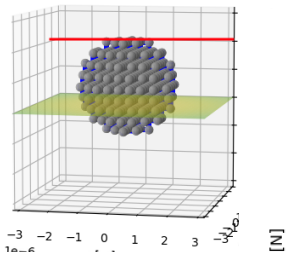
. ~2197 dipoles in largest test, lambda=1e-6 => well within lambda/10 limit

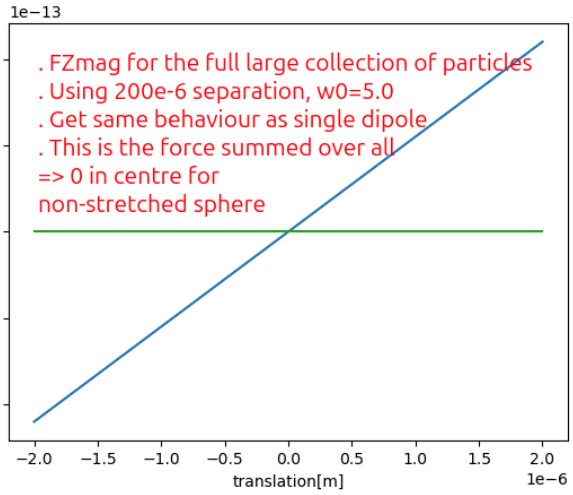
To continue with the main project goal, I need to understand why this shape appears to want to stretch and compress in an oscillating manner as seen in the plots, rather than just stretch only then compress only after that (after the critical stretching is reached such that the springs are now more influential than the beam), which I presumed was due to the fringes from the beams' interference before, but now that this has been removed through an X and Y linearly polarised Gaussian beam this cannot be the reason.

First off, when checking that the Gaussian beam can actually only be X polarised I see that the Gaussian beam has a default value for X polarisation in its YAML parameters, but looking through the ***create_beam_collection() and make_beam()*** functions it does not appear as though the ***GAUSS_CSP*** would not be allowed to use LCP polarisation (there is no check for beam type when the jones vector is pulled from the polarisation state, and before it is added to the YAML), however this being said we did not specifically require the LCP (as the polarisation

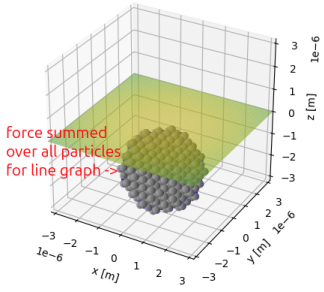
used in the paper was not specified), so a more simple X or Y polarisation should be fine and hopefully simplify the situation.

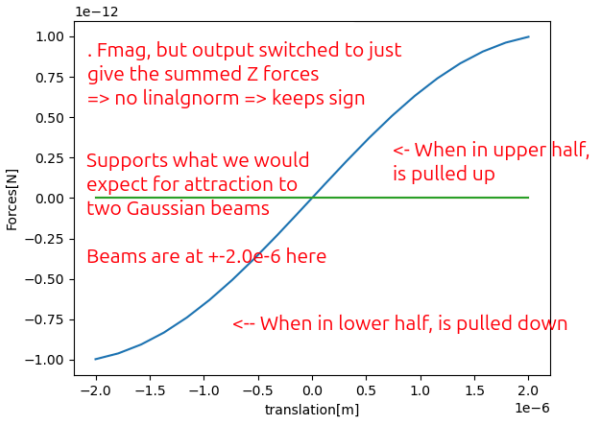
Next, I want to move a single particle through the beam to ensure that it does uniformly have a beam force either directly in the positive Z direction for Z>0, and vice versa.

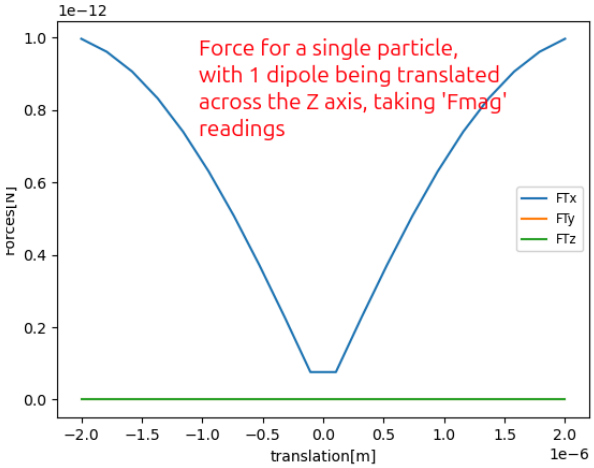




When considering this for the large sphere of sub particles and for a single dipole being translated, both have the same expected relationship of stretching

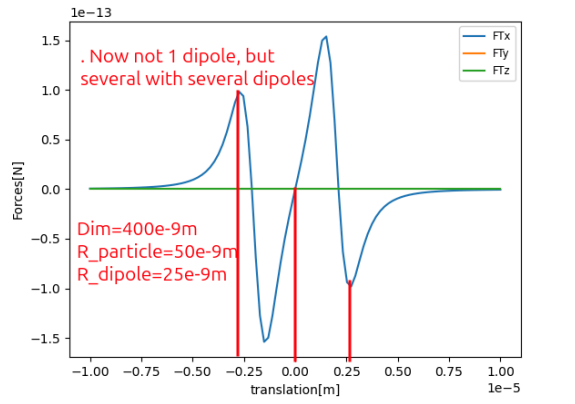




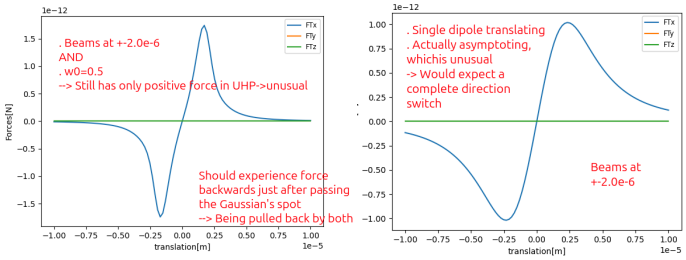


downwards when in -Z region and upwards when in +Z region (e.g. when closer to either Gaussian, get pulled to it). Note as well that for the large particle collection tested, the Gaussians were at +-200e-6 (very very far away), hence we saw no turning point to the curve, whereas that was seen for the single dipole case when the beams were at +-2e-6.

Only when considering not just a single dipole single particle setup, but having several dipoles and particles do I see the expected behaviour, where about the +-2.0e-6 point the Z force rapidly switches to the opposite sign then asymptotes as you travel away. Now that I have seen this on a small scale, I need to see how it is influenced as I scale the mesh up, and whether this is occurring the cases I have been testing or not, as it

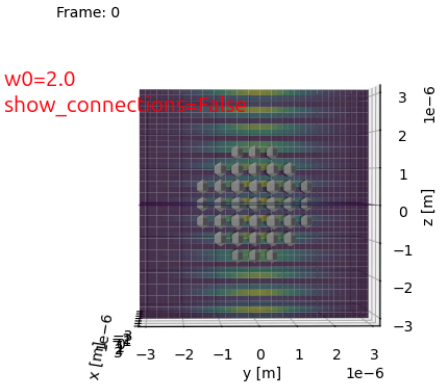


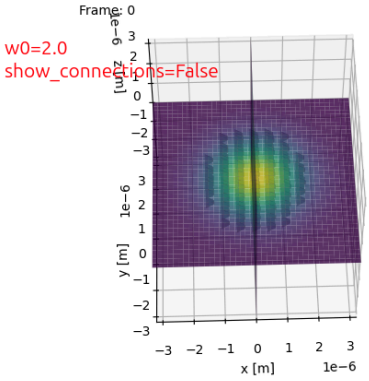
should be happening in order for the stretching to occur as expected (pulled to these Gaussians, then possibly constrained by the springs if far enough; hence for very far away Gaussians they will require the springs to reach some equilibrium).



21/2/25

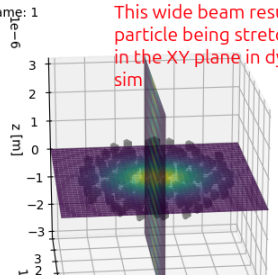
To start these more realistic tests, I will first do some dynamics tests with the stretcher foci offset in order to see some better stretching. If this looks reasonable, then I will do a sweep for different scaling.

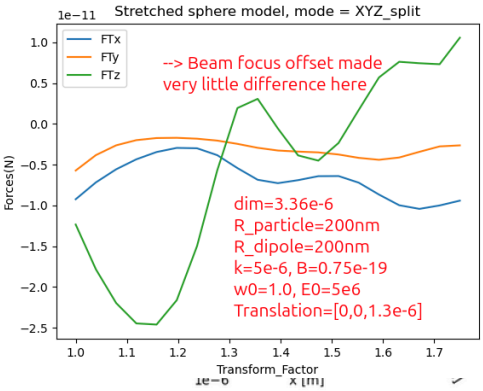


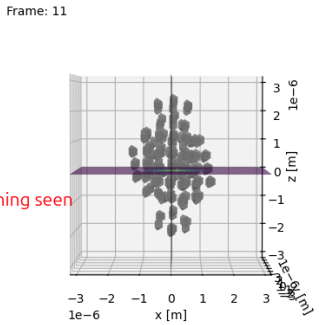


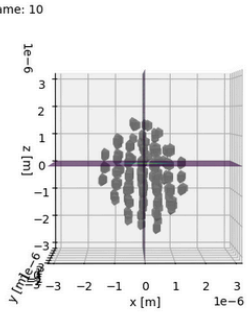
Vary power, see how stable equilibrium stretch value changes

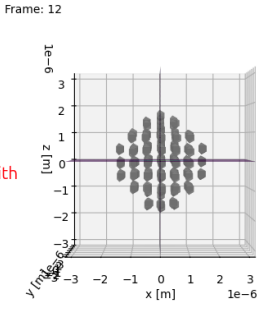
In their experiments they used base sphere of R=3.36 micrometers, but this would require too many dipoles if tested at this size whilst also keep the dipole size < 100 nm (< wavelength/10, which we could exceed but generally this accuracy is considered where DDA works

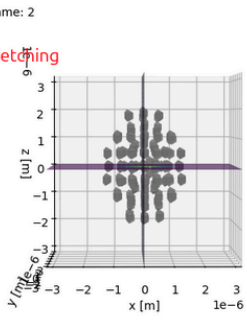


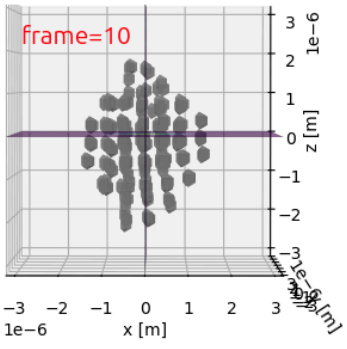


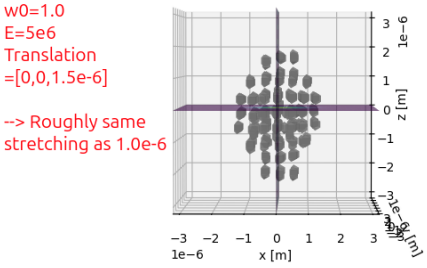


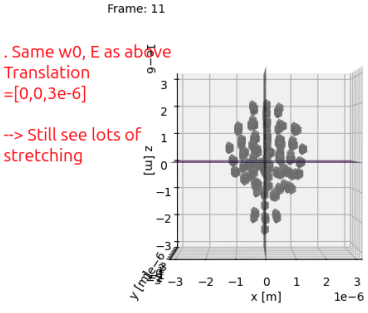


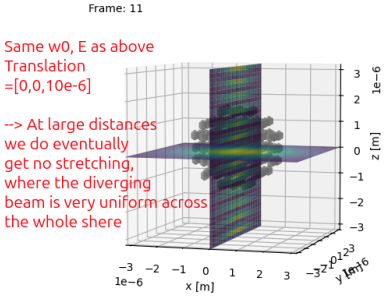






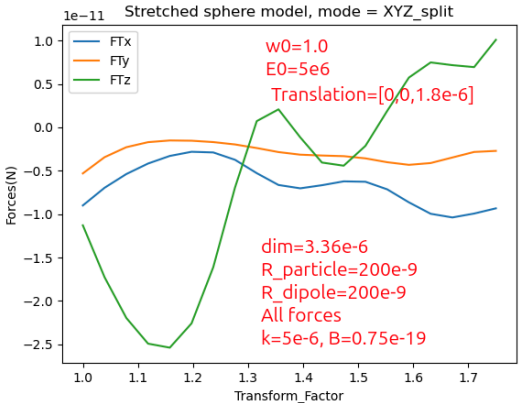




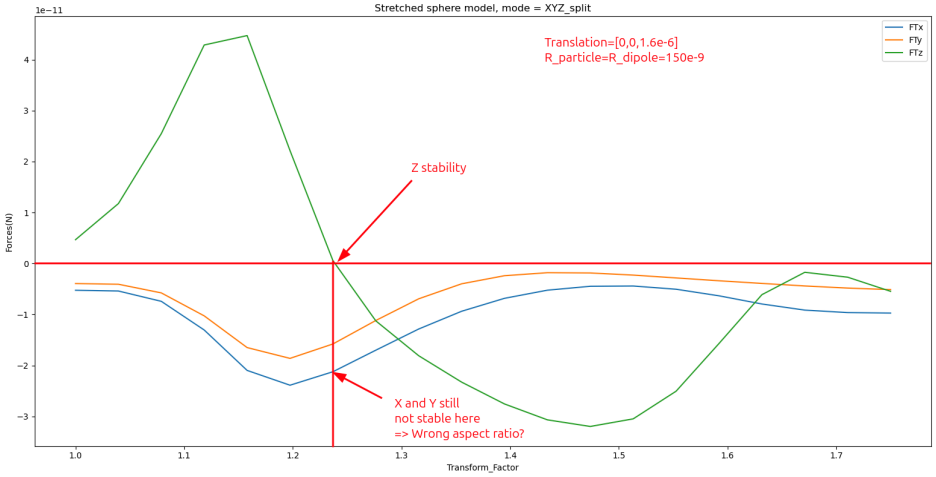


best), hence I will consider a setup of half the size therefore also half the beam widths and powers too. Translation 1.65e-6 used for the first 3 figures tested. After, a grid of varying translations is tested (as well as considering some of the difference from measuring at different powers).

The particle reaches a steady-state where it wants to stretch and stay stretched (seen in dynamics), but the first frame has compression initially before it bounces back into a stretch which is then sustained.

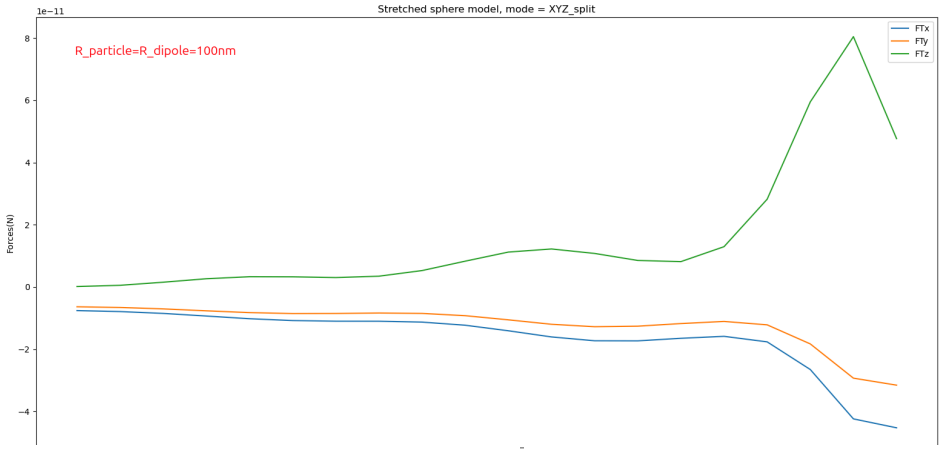


Fringes are very tight, hence you must have lots of particles to accurately map which fringe you are on, otherwise a tiny change in separation of particles will give a vastly different Z force.

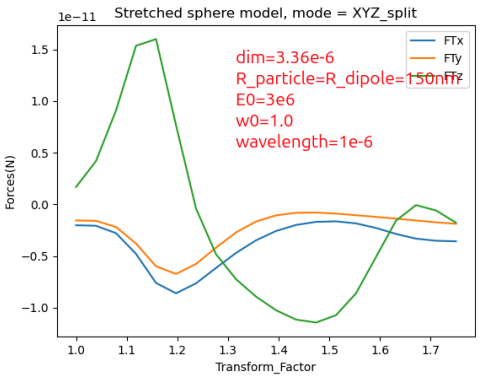


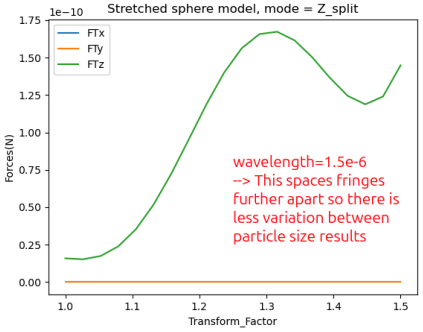
Still unsure why X and Y forces vary.

As separation between particles/dipoles tend to zero, we should see the plot stabilize in shape (no fringes skipped or similar), this can be improved further by considering a smaller range of transformations, so less space will be required between elements.

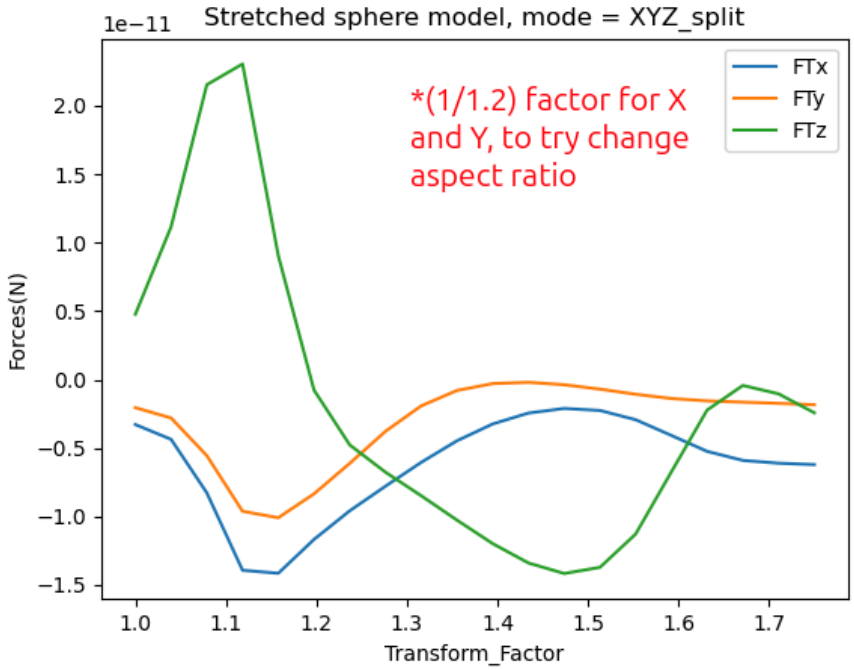


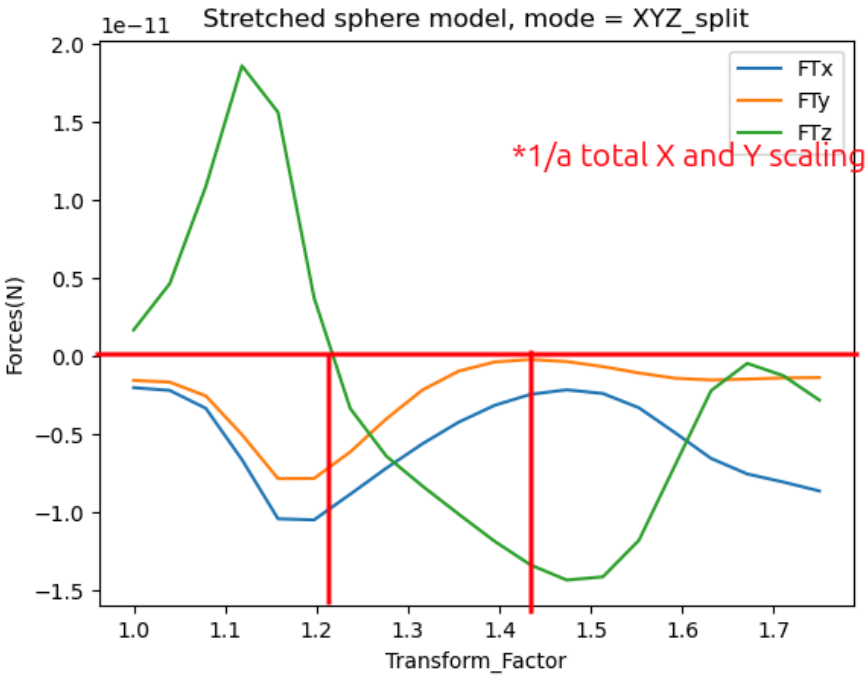
After speaking with our supervisor he said that;
. Only X linear polarisation is implemented for Gaussian, therefore





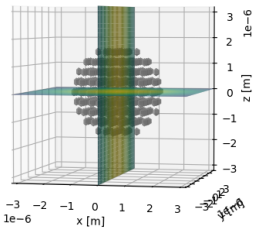
don’t worry about LCP/RCP
-> Do 90 deg rotation to remove fringes
. X force different to Y likely from X polarisation





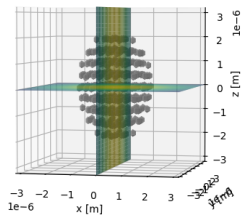
Frame: 0

w0=5.0
translation=[0,0,20e-6]



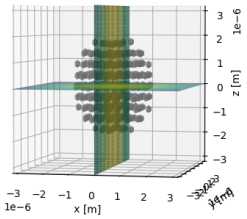
Frame: 4

Stretches out

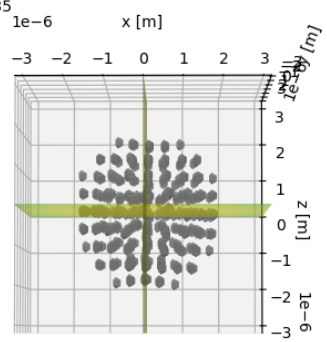


Frame: 6

w0=5.0
translation=[0,0,20e-6]
--> Bounces back a bit

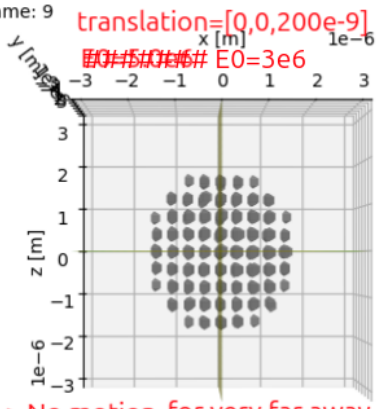


Frame: 35



Frame: 9

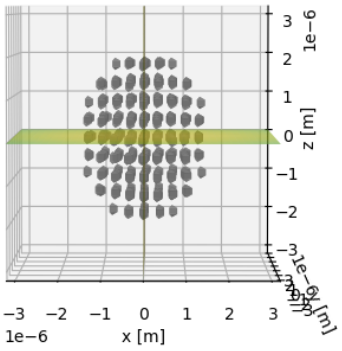
w0=5.0
translation=[0,0,200e-9]
E0=3e6



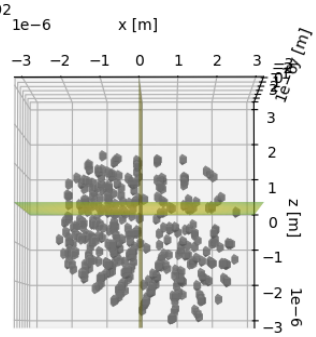
-> No motion for very far away beam foci

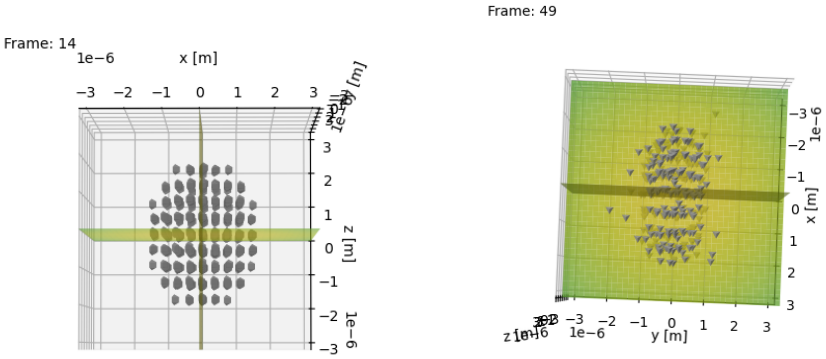
Frame: 9

E0=5e6
w0=5.0
Transaltion=[0,0,200e-9]



Frame: 92



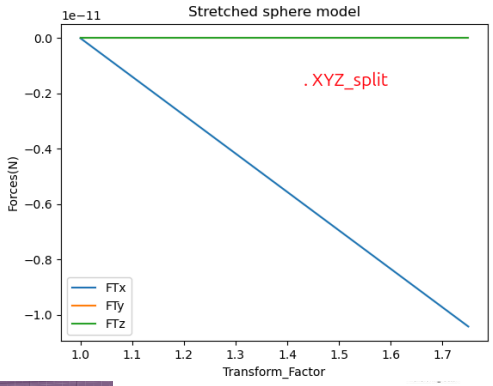


Squished in the Y axis for long times here. This happens for varying E0 values, possibly Y oscillations too large. Need to do some simpler checks to ensure polarisability is ok.

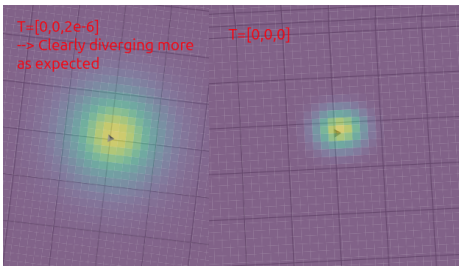
20/2/25

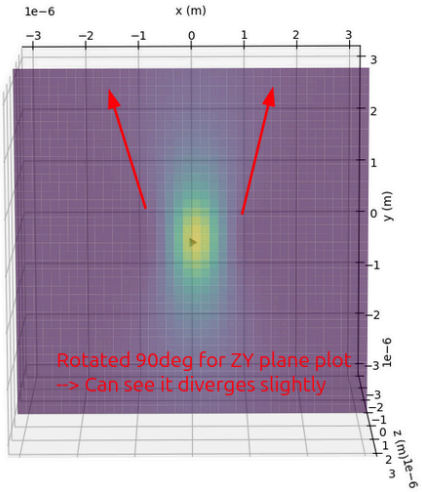
. Spring XY is fine, it was just that Z_split summed opposite contributions therefore cancelled, XYZ split works

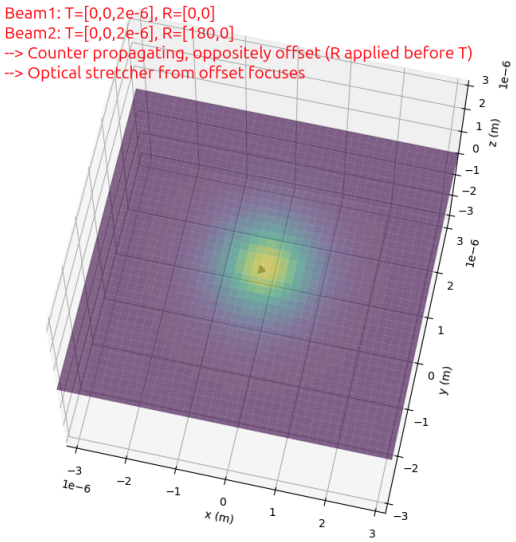
. Bending having no effect makes sense as is stretched linearly, never bent, only affected



ALl the forces seem more trustworthy now, so the next priority is to match the simulation parameters to those seen in paper ***"The Optical Stretcher: A Novel Laser Tool to Micromanipulate Cells"***, which uses unfocused laser beams, hence the particles need to sit just beyond the focus for both beams, which may not be the case here currently. We need to be careful when placing the beams as the rotation appears to also rotate the translation, so we should visualise each separately then visualise the combination to ensure the beams are correctly set up. It also mentions that ***" This condition is fulfilled if the refractive index of the object is larger than the refractive index of the surrounding medium and if the beam sizes are larger than the size of the trapped object"***, hence we should probably increase the beam width being used here.

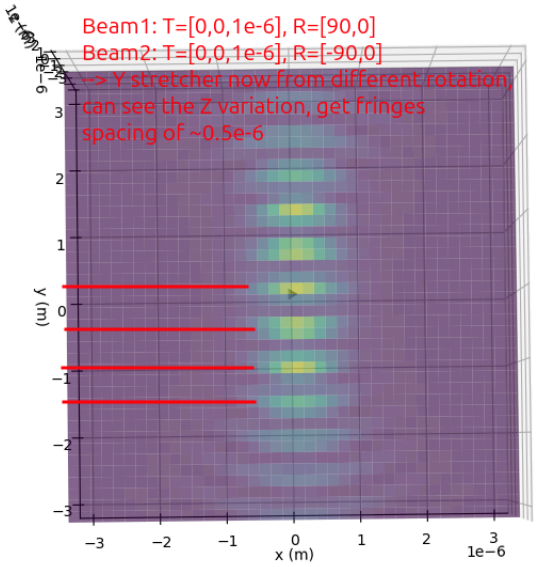




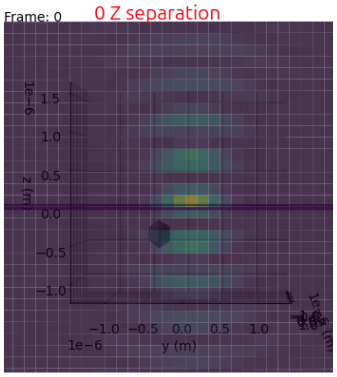


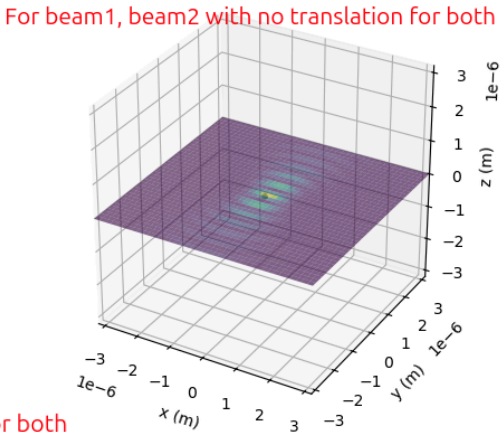

```
beam_0:
  beamtype: BEAMTYPE_GAUSS_CSP
  E0: 8000000.0
  order: 3
  w0: 0.5
  jones: POLARISATION_LCP
  translation: 0.0 0.0 2.0e-6
  translationargs: None
  translationtype: None
  rotation: 0.0 0.0
beam_1:
  beamtype: BEAMTYPE_GAUSS_CSP
  E0: 8000000.0
  order: 3
  w0: 0.5
  jones: POLARISATION_LCP
  translation: 0.0 2.0e-6
  translationargs: None
  translationtype: None
  rotation: 180 0.0 0.0
```

This paper also justifies why a membrane of particles should be sufficient for the elastic properties of a RBC: ***"he only elastic component of RBCs is a thin membrane composed of a phospholipid bilayer sandwiched between a triagonal network of spectrin filaments on the inside and glycocalix brushes on the outside (Mohandas and Evans, 1994)."***

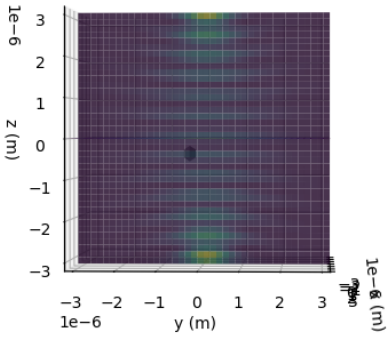


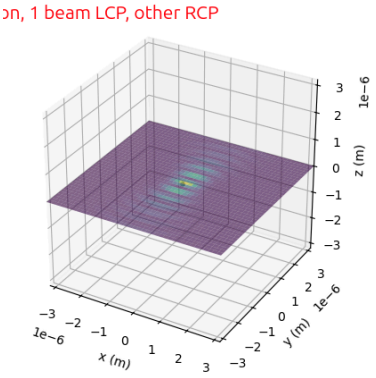
Since the Gaussian beams do diverge here, we want to offset their focus from the centre so the main bright spots are either side of the particle in the Z axis, which should make the stretcher work more effectively. However, this requires the beams to have rotation 0deg ([0,0,1] propagation direction) and 180deg ([0,0-1] propagation direction), which therefore means both have equal positive offset (since the 2nd will be rotated before applying, therefore offset in opposite direction).



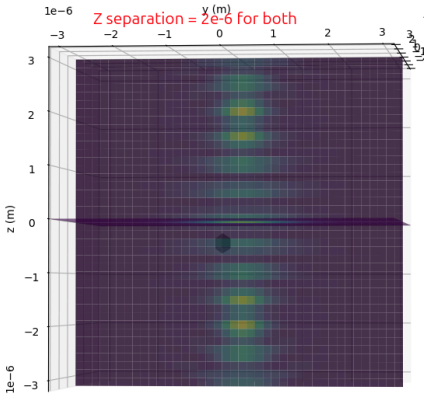


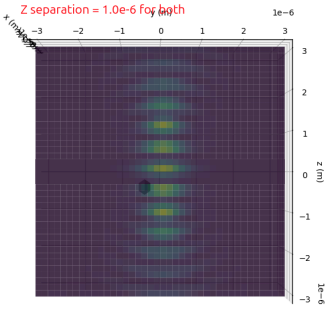
Testing for different polarisations made no difference, but this offset changed where the high intensity parts of the beam occurred in this interference (shifted outwards for both when both shifted outwards). Hence by having some offset in our system we would hopefully see more pulling.





We should have the beam radius be slightly larger than the particle radius for optimal trapping, and it seems as though the





offset of the beams should be about the radius of each particle so the two high intensity spots sit at the particle's sides, hence pulling each side strongly into each trap whilst keeping it centered on the 0 of the axes. The particle they tested was originally ~3 micrometres radius (sphere), which deformed to a radius of ~6 micrometers, hence a diameter of 6 and 12 micrometres which is much larger than we have tested before. Our wavelength is 1 micrometer (0.785 micrometres for the experiment technically), hence we would want dipoles of size less than roughly 0.1 micrometres to be valid in DDA (100 nm max), so for a 12 micrometre shape we would have 120 dipoles stacked along the width as a minimum, which could potentially be very slow to run if any more than 1 dipole per particle is used (next best would be 2x2x2 per particle, hence 8* as many, therefore ~1000 dipoles along the width alone not accounting for height and depth).

Therefore;

. The spring force problem was resolved, and the offset Z force to what was expected was also resolved (scattering resulted in interference bands in Z axis, therefore different separations resulted in varying force directions as it should do)

. The bending force problem was also resolved (irrelevant as stretching applied here did not change angles between the particles, e.g. all layers shifted together and got space in the X and Y equally).

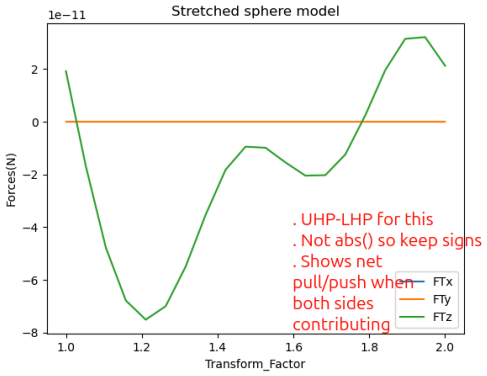
. The only remaining question is as to why the X and Y forces did not match, since the transform should be identical for both).

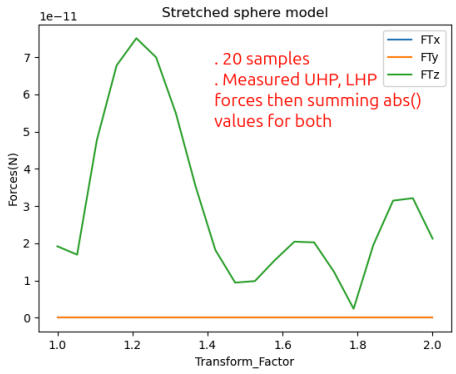
. Hence the next course of action is to match the paper's experimental setup (using offset focuses for better stretching + still trapping, and correct mesh sizes). Since a RBC was being investigated it would also be worth considering just a membrane to model the behaviour (relying on bending forces mainly to prevent it collapsing and so give correct deformation behaviour). I would also like to continue investigating the filled in version too as it more easily would generalise to other particles.

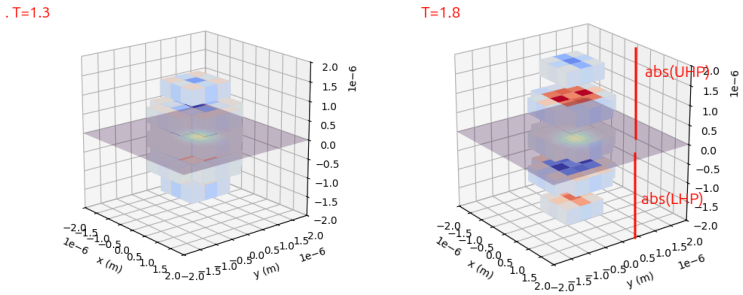
I also have been discussing and reproducing some of the graphs we will need for our poster presentation, which have begun assembling. Me and my partner also agreed on the title of project being ***"Applying a bead and spring framework to light-driven deformation"*** as a new working title, due to the Abraham-Minkowski controversy being a far more secondary part of the project.

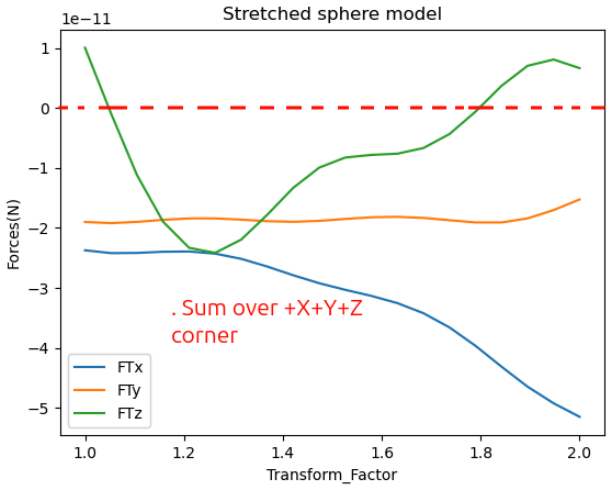
19/2/25

Today I want to get the force calculation working on different regions of the graph, as yesterday I did a measure of the full set of particles and found that they all had very small force (O~10^-25, when we would expect O~10^-12), which interesting still showed a nice sinusoidal curve for these low values, which is go and expected since the Gaussian beam and stretched sphere are both symmetric about the Z axis. By considering forces on particles in the positive Z and negative Z regions (***UHP, LHP***) we can then sum the ***magnitudes*** of the forces here and so see how strong the force couple is that acts to stretch or compress the sphere in the Z axis (which is what we are actually interested in). SImilarly, this should be extendable by considering the 8 octants of the bounding cube for the sphere mesh in order to find the force couple for the X, Y and Z forces, which we could also simplify by just considering the force on one of the couples and assuming the symmetry this would have over all 8 octants (e.g. measure the force on only 1 octant and multiply the force found by 8 in each axis, however this would not give a nice average value as there will be some imbalance in force).



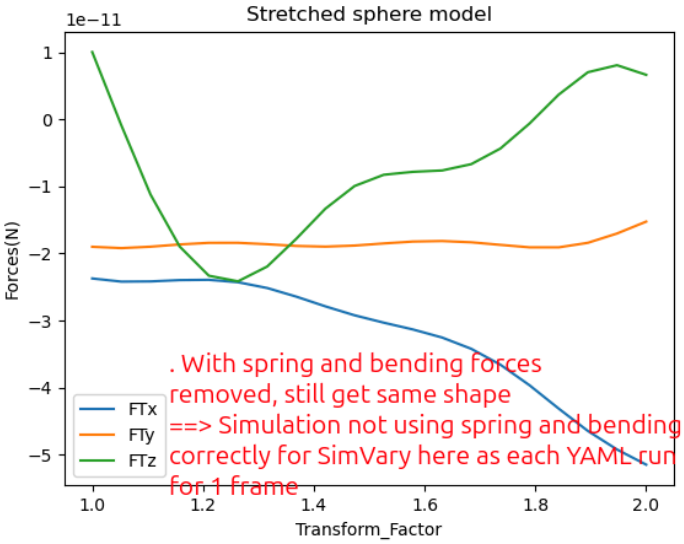




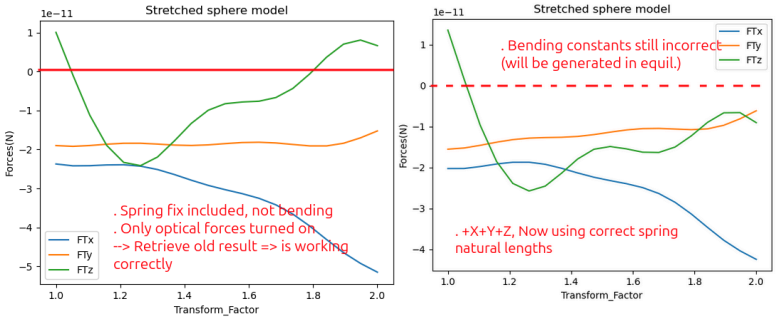


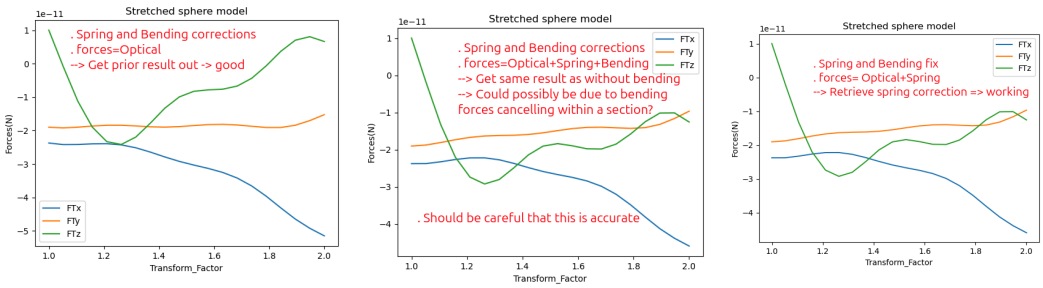
Performing this gives the graphs above, where we see turning points in the Z forces where we approach a correct shape for equilibrium but then either overshoot or have invalid aspect ratio between the width and height, meaning this turning point never reaches 0 force. Also, we see that the X and Y forces (in the 1 octant case/+X+Y+Z case) has X and Y forces with no clear turning points, implying that the forces cancel over the total shape from . ***This was performed on a sphere model with parameters;***

R_sphere=2.4e-6m,
k=5e-6
B=0.75e-19
R_particle=200e-9m
R_dipole=100e-9
E0=7e6
w0=0.5
=> Dipole total=648

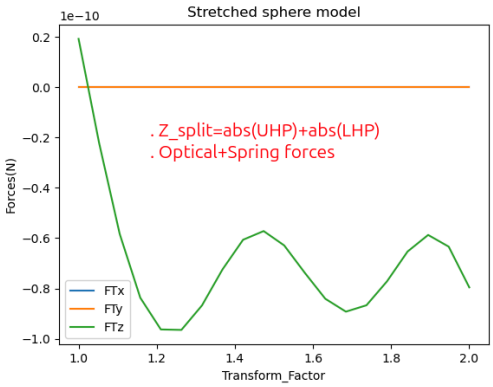


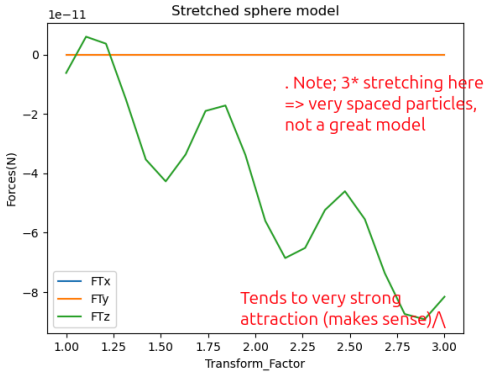
When running this same setup but with no spring forces, I found that the graph was identical. This is due to the fact that, despite setting the connections correctly inside the YAML generation, the natural length matrix for the springs is fixed using the particles specified in the final YAML, e.g. it will use the stretched positions as the locations for the natural length. This is not a huge problem however as the dynamics I ran yesterday are still valid as they ran for several frames all beginning at the unstretched sphere, hence had their spring natural lengths set correctly (so the stable values found are valid). ***This being said, the previous graphs generated for varying transform factor will not be correct and so will need re-running*** after the natural length generator is fixed. My plan for this is to parse in an optional value to be used to overwrite any of the non-zero matrix elements, which should be fine for systems where the natural length is the same for all connections, e.g. nearest neighbour connections on a cubic lattice as is done here. In future, this could be expanded to just parse in the full matrix of elements desired so the connections can be arbitrary. This also explains the glaring problems in the graphs above such as why the Z force is negative for most of the time considered, despite expecting positive Z force for most of it, as well as why the X force never tends towards 0 force (no restoring force, just more and more repulsion).





Fixing the bending constants will be more challenging in this same method, hence it would be best if I just calculated the natural length and bending coefficient matrices inside the YAML generation, then parsed them into main() to just be copied, as the bending matrix will have very different values for each term.



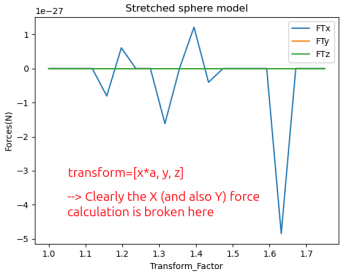


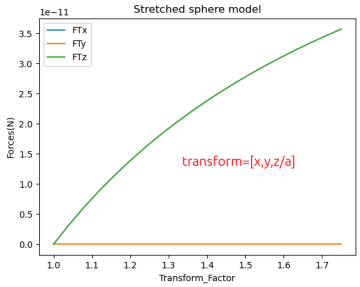
The change I made for this is actually to parse in the initial positions of particles into the YAML, which can then be read to generate the spring matrix or bending matrix from just this list. This keeps all the spring and bending calculations localised in the main program, which is cleaner. However, when testing this it seems as though the bending force does not change the profile observed, which is unusual since the dynamics clearly differs when it is changed. This could be chalked up to cancellation of the equal and opposite force terms when viewing a segment of the particles, however I would expect at least some small change still for the outer shell of particles. Ignoring the bending for now until I come to a good conclusion for this, I see a similar but shifted Z behaviour to the purely optical case, which makes sense as particles are oriented symmetrically about the Z-axis, hence any contribution from either side would cancel, but the Z is being measured separately to its opposing side (that being said, the same should be true for the X and Y components however they appear to have not changed). It is also weird that the Z forces are purely negative here (better than mixing between positive and negative, since I would expect a gradual change in the same direction, only switching once after some critical distance), but this could be due to the force calculator actually performing the LHP-UHP calculator instead. Either way, the core idea of the experiment is to find when its magnitude tends to zero, which can still be done here. The next step for this would be to test for other configurations, e.g. differently scaling linear functions or nonlinear functions.

I will now test for a larger range of scaling, which will unfortunately come at the cost of accuracy if the particle sizes are kept the same, as it will have to space the initial placement further apart.

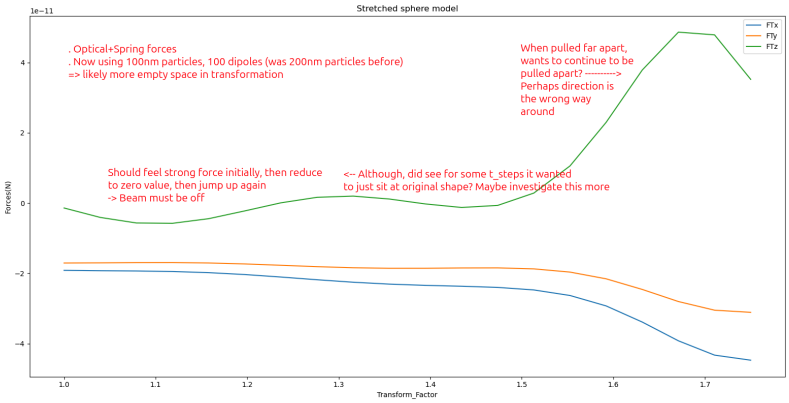
I would like to test scaling more, 1 example is given in a plot where it can be seen the accuracy change gives dramatically different results. This also highlights a key point that in order for Hooke's law to apply AND for particles to have less space between them (therefore greater accuracy) you must test for only smaller total scaling amounts, which also makes sense with the low levels of bending observed in dynamics before an equilibrium was reached. It would also be good to test this with a more realistic mesh size and beam strengths.

Tests for transformations of (1) no change and (2) no X or Y change, but Z/factor instead were performed and (1) gave results as expected, (2) gave a nonlinear curve, which is unusual since the force just uses ***F=kx***, therefore linear scaling would also be expected, however this may be correct behaviour (1/x curve to be considered). Considering (3) scaling in the X*factor, but not in the Y or Z lead to a clearly broken reading (essentially 0 still), hence this just be fixed. This is likely an error in the format the original shape is parsed into the YAML and then into main() again, leading to the wrong constants being found and the wrong positions/connections being considered.





As another note, the oscillating seen in Z_split graphs seems to show oscillating, which I think is from optical binding, since it occurs repeatedly at these even intervals (=> even spaced stretching in a linear stretching regime). Also, the growing X (only slightly, if at all, seen for Y) makes sense with the spring force as the particles are being very tightly bound/compressed, hence wants to spring out, however even with this you would expect the beam to be pulling them back in with equal force. This suggests that the scaling for the two are off, e.g. for the X value, it quickly surpasses its stable extension, whereas the Z is only just starting to get to its stable extension, hence both will never reach stability together perhaps?



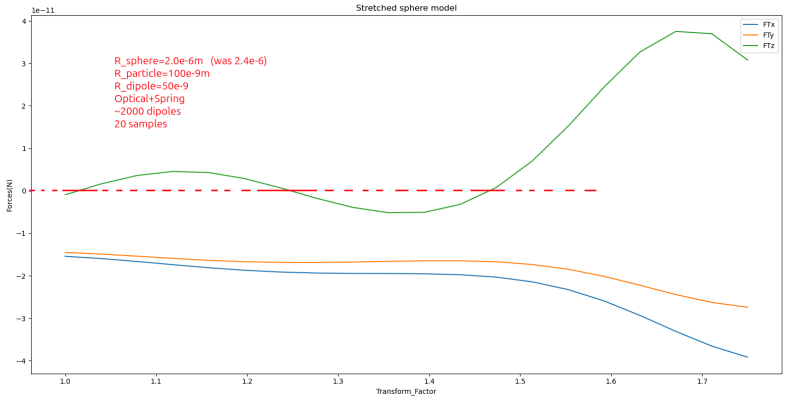
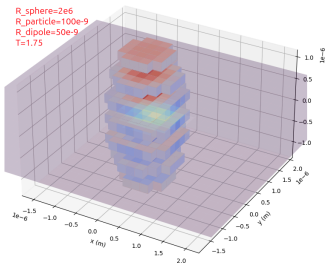


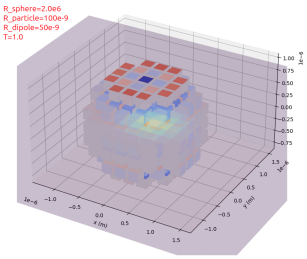
Diagram shows strong outwards force for the top layer, but perhaps is cancelled by lower layers?

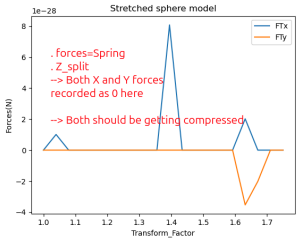
Major problems of note;
. (0) X/Y force calculation is broken for springs in these cases.

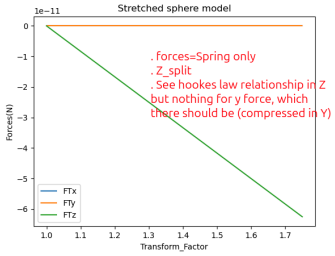
. (1) Z force 0 at original (*1.0 scaling) shape + Z forces wrong direction for high stretching in many cases

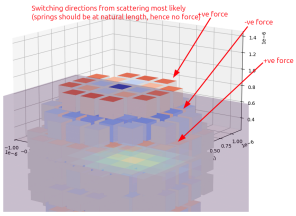
Next fixes to test;
. (0) FInd error in X force calculation
. (1) Test for more realistic mesh size + beam setups, as seen in papers
. (2) Ensure the forces on the particle are as expected for simple/obvious cases such as spring forces on original position, + spring on easily deformed shape, + optical on known cases, + combination to ensure this combining is not causing problems.

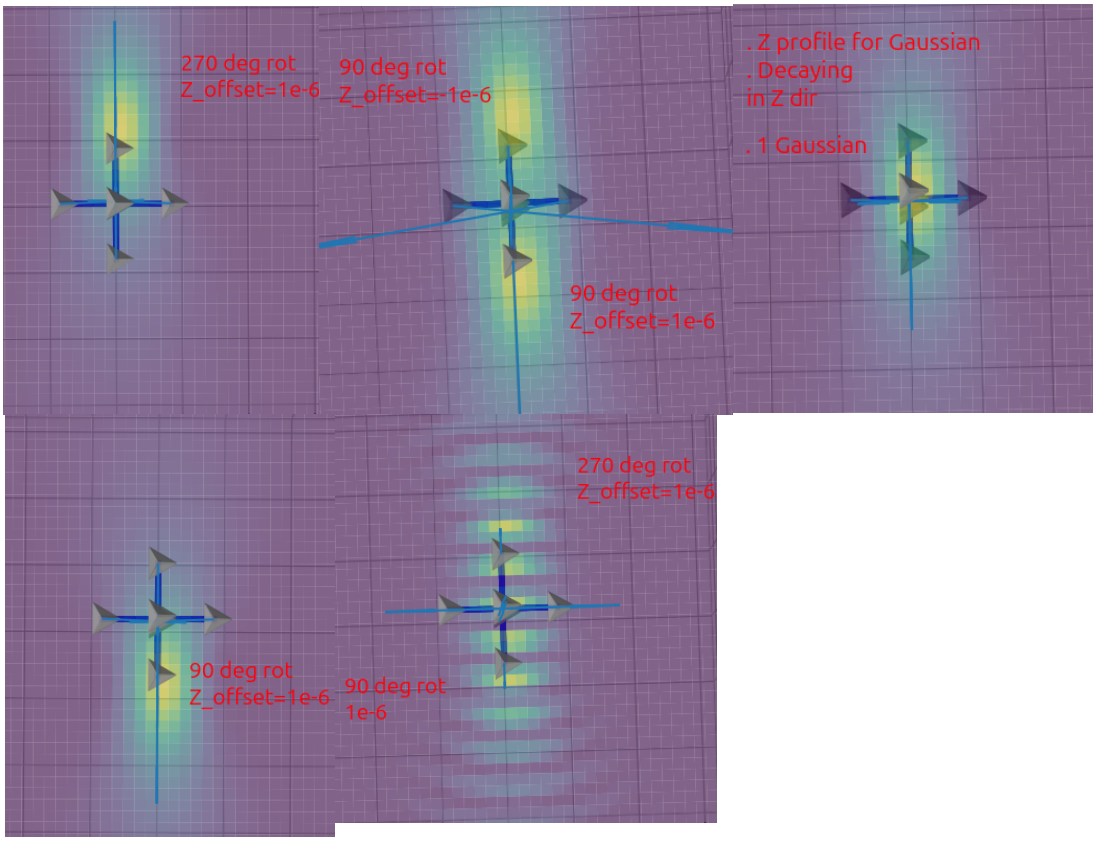






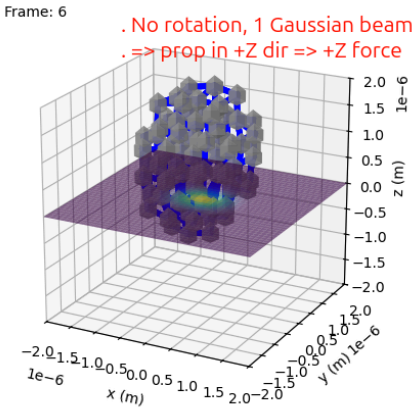






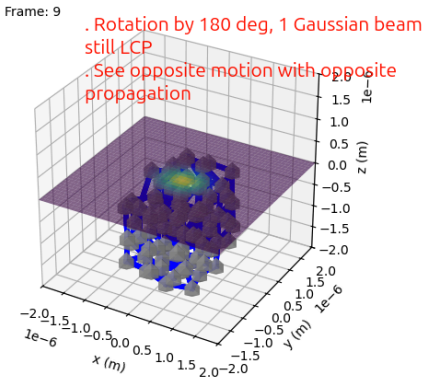
18/2/25

Currently I am testing some different parameters to see if I can make the simulation behave better, rather than the one sided motion and quick explosion of the setup yesterday. Turning off the spring and bending force I see pretty similar motion to before, which suggests the beams are the main cause for the problem, likely from being too powerful.

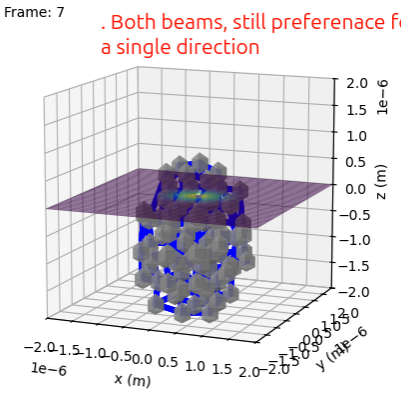


Setting ***k=5e-8, B=5e-20***, made the system less prone to explosion, but still had a tendency to drift upwards when it should sit in the center, hence I tried disabling the upwards facing beam **(as a thought, the polarisation maybe needs to be reversed too when counterpropagating the beams, will have to check this)**.

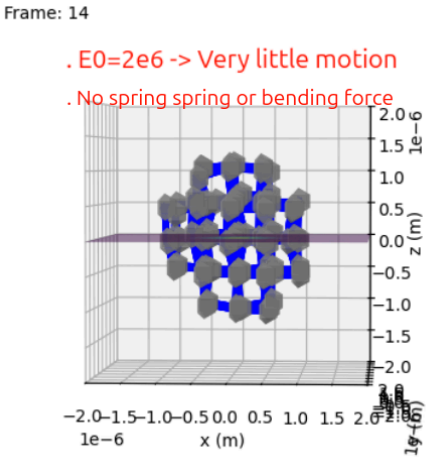
When running a test for the beams separately, I saw that there was a problem with the rotation where it was in terms of radians not degrees. After fixing this, the two beams separately moved the beam in the correct direction as expected (along the propagation vector). Testing this for left and right circularly polarised light (LCP/RCP) made no difference to this gradient, so I will continue using the default LCP for now. Re-running the main program with both forces now still gave a preference for a single direction.

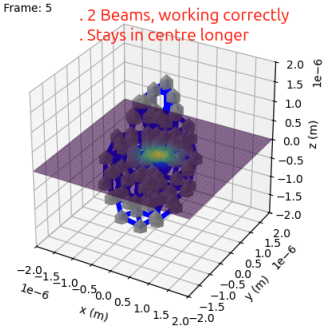


Offsetting the particle still gave the same downwards preference.
Switching the order changes the preference => YAML generation problem
Fixing this we see the particle have less of a preference so possibly just a stability fix through parameters?

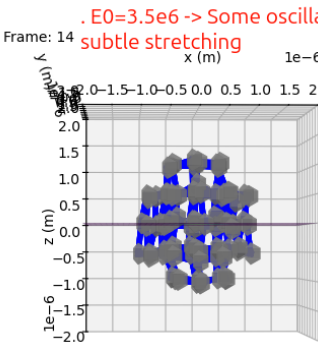


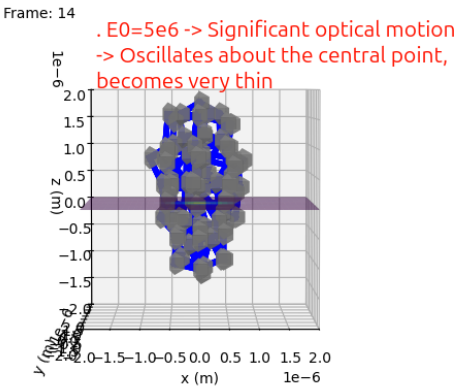
My approach to fixing this stability will be to lower all the values to be far more gentle in the system (e.g. beam forces, springs, bending, time step, etc) so everything moves

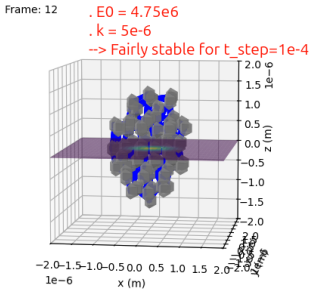


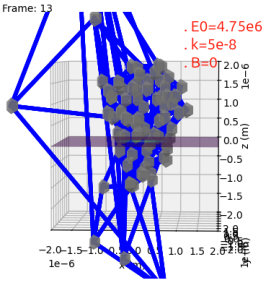


at a slower less sporadic pace to reduce chances for explosions. The simulation can then be run for longer to test for stability, and the parameters increase to an appropriate amount that physically makes the most sense, whilst maintaining this stability (which can also be further corrected through smaller time steps). Doing this, it seems like a power ***E0~(4.5->5.0)e6*** is a good range which is fairly stable but also has the ability to stretch, so I will now start introducing a spring force to try and prevent the collapsing structure in the XY plane, which will then likely allow the base intensity to raised again.

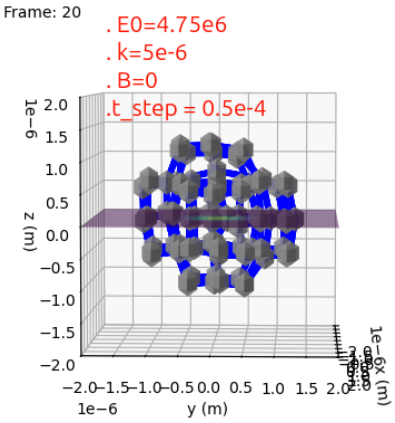


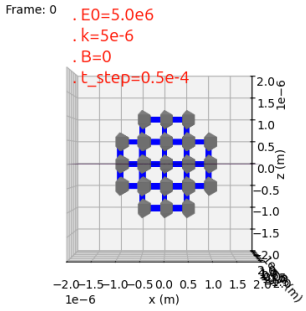


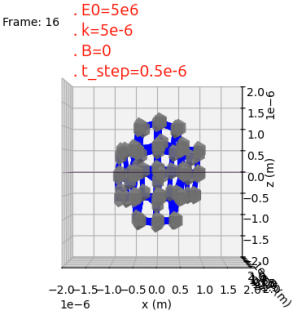


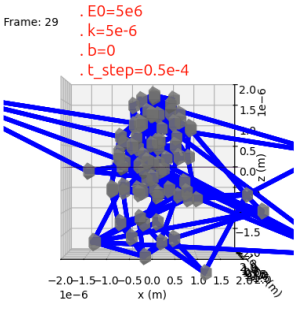


When testing the spring constants it can be seen that extremely large or extremely small values explode, where the sphere other compresses too rapidly from the spring pulling it inwards after small fluctuations, or the sphere compresses from the beams' gradient pulling the particles inwards (and the spring constant is too low to repel this), hence an intermediate value is required correct stretching where explosive behaviour does not occur. When choosing k=5e-6, with a time step of 1e-4 we see some small stretching and a fairly stable system, but for the 0.5e-4 time step we see a sphere with very little stretching and what actually looks like some expansion in the XY



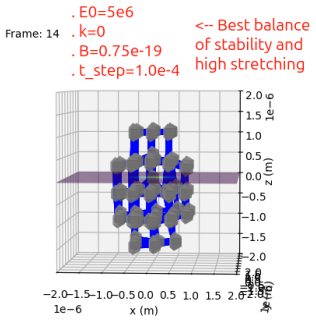


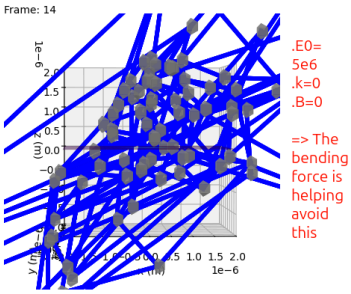




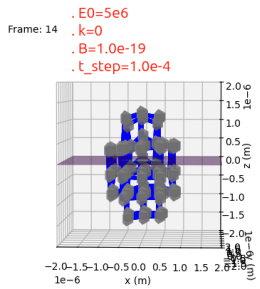
plane. This implies the spring constant is too large, hence is unable to move properly. Note for this smaller time step I increased the frames to view the motion for the same amount of time.

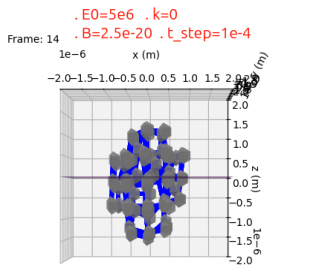
When increasing the field strength (E0) for this t_step=0.5e-4 case, the stretching is retrieved again, which suggests that on the lower accuracy scales the equilibrium spring constant (withstand the force of the beams) is reached for a lower field strength. I now want to look at the effect of the bending constant on this motion.

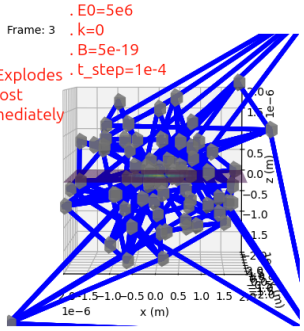


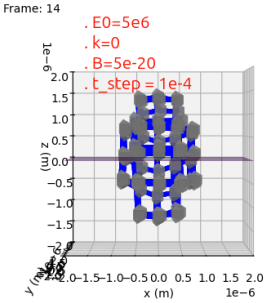


From the observations of just the bending force on the right, it appears to give far more stable results that just with spring forces, and still prevents some of the XY plane deformation seen when ***k=0, B=0,*** so it has benefits over these other two regimes. However, with purely bending forces it is possible that volume may not be conserved as well as occurs when spring forces are present, as a uniformly scaling sphere in all directions will maintain its angles but increase in size, and so will stay in equilibrium according to just bending forces, hence I want to use both of these and will likely need a more impactful bending force with a smaller spring contribution for smaller corrections. Currently, it seems like ***B~0.75e-19 and k~5e-5*** are the best (most stable whilst being on the border of allowing more/less stretch) values individually, hence I will have to see how they perform together (will both likely need reductions).

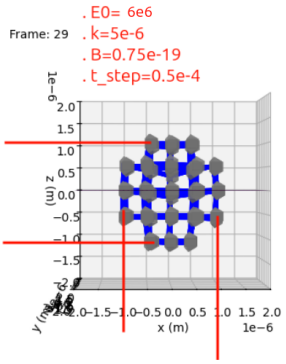


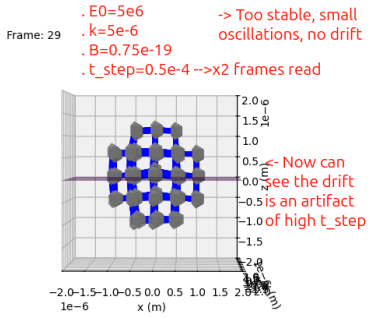


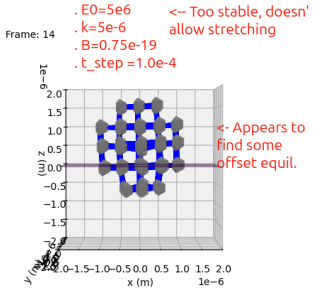




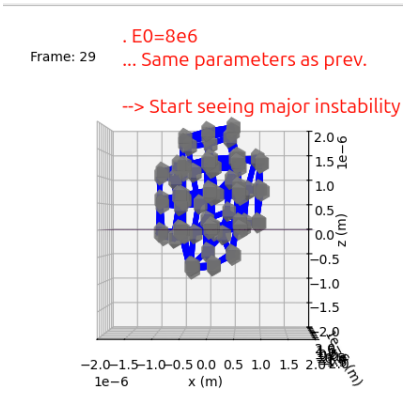
Testing this system shows that these two values combined are too stable and so do no stretch at all (as predicted),

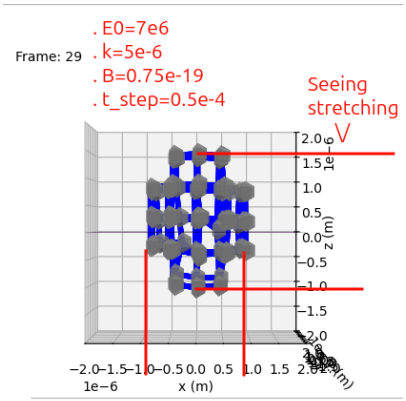


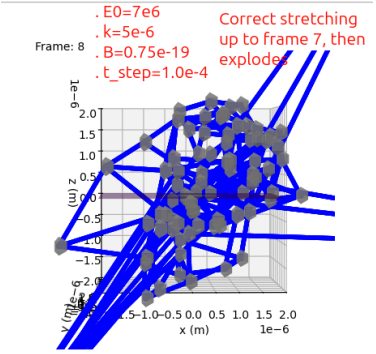




so I will now try just increasing the power of the beams to find the point at which this overcomes the spring forces and reaches variable equilibrium configurations based on how strong this beam is. This could act as a way to calibrate the constants (experimentally find an equilibrium with beam power variation, then associate this to the constants ***k and B***, however this would assume that the relationship between ***k and B*** is correct).

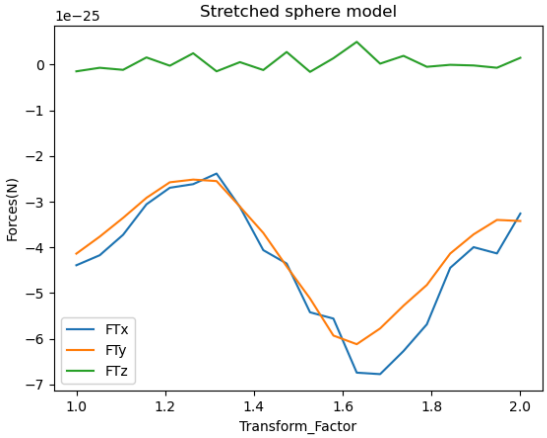






For the small time step (0.5e-6s) I find a stable looking system (when observing the dynamics for 30 frames) for ***k=5e-6, B=0.75e-19, E0~[5,8]e6***, where it ranges from very little to unstable stretching between this E0 range (and obviously stable for E0 less than this).

WIth this system appearing quite stable (at least for before significant time-stepping occurs, so should be good for a first frame review) I will now look at how the total forces vary as I scale the shape's scaling 'linearly' ([x,y,z]->[x/sqrt(a), y/sqrt(a), za] for a scale factor a, which will be varied) I should hopefully be able to see where the scaling moves the whole particle into a state closer to equilibrium, or further away from it. These rough dynamics simulations also gave an idea as to what the stretching should tend to in each case, for example with the ***E0=7e6 case,*** the original dimensions in the XZ plane were ***(2.4,2.4)e-6,*** which then transformed into ***~(2.0, 2.75)e-6,*** so the X and Z scaling is ***(0.83,1.15)***, which means the X is scaled by ***1.0/sqrt(1.45) and the Z just scaled by 1.15.*** This does not match the 'linear' prediction made of ***X*1.0/sqrt(a) and z*a,*** but this prediction would not strictly conserve the volume of the particle (volume changes by ***1.15*(0.83)^-2=0.79 != 1.0***), however perhaps this is not a problem as the stretched material becomes less dense as it stretches, hence you do not have to conserve volume particularly (mass can be constant with changing volume). I will try and perform this test for a system that scales like this as well to see if it performs better. This system does also make sense physically as there is a stronger force at the equator of the sphere, hence it would be more likely to be pulled towards the origin (as observed with trapping), and so you may expect more compression at the equator than at the ends along its length.

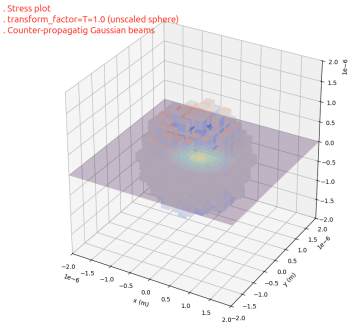


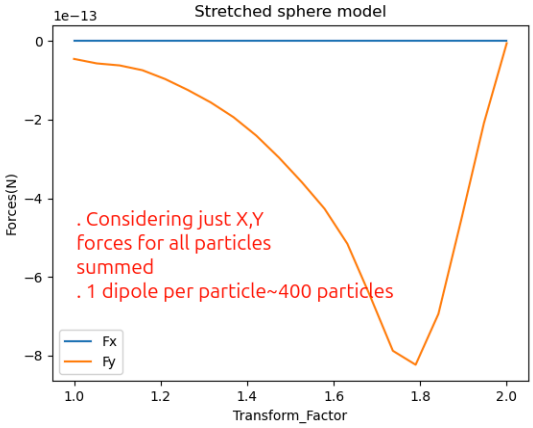
. CONSIDER FORCE TOTAL'S ACROSS A COUPLE OF THE WEIRD SYSTEMS THAT CHANGED WITH TIME STEP -> SEE HOW THIS COMPARES ON FIRST FRAME
. TRY DIFFERENT MESH SIZES IN THIS SYSTEM -> MATCH RBC
*. TEST THIS NEW SCALING SEEN AS FUNCTION TOO → X=*0.83, Z=*1.15*

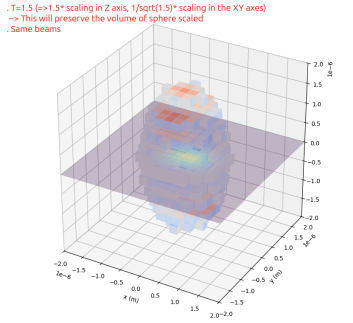
. I WANT TO PULL DATA FROM THE TOP / BOTTOM HALF SEPARATELY => SUM MAGNITUDE OF FORCE THE EXPERIENCE TOWARDS/AWAY FROM CENTRE
—> YOU WOULD EXPECT FROM SYMMETRY ALL CASES TO CANCEL TO ZERO FORCE

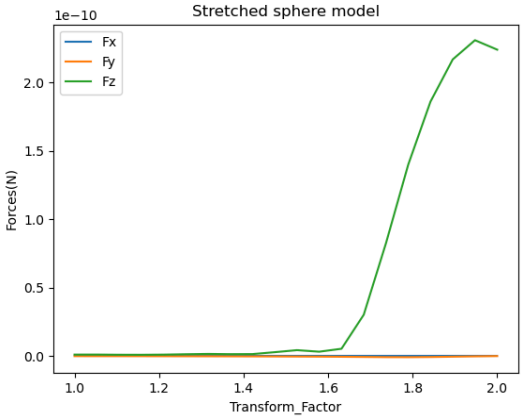
17/2/25

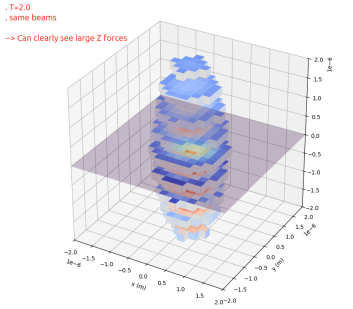
The focus for today is to start considering different flexible deformation examples and seeing if the bead and spring with DDA setup can be used to understand this deformation and prove its validity.

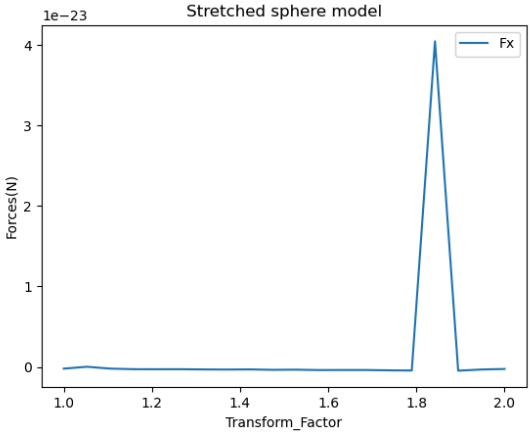








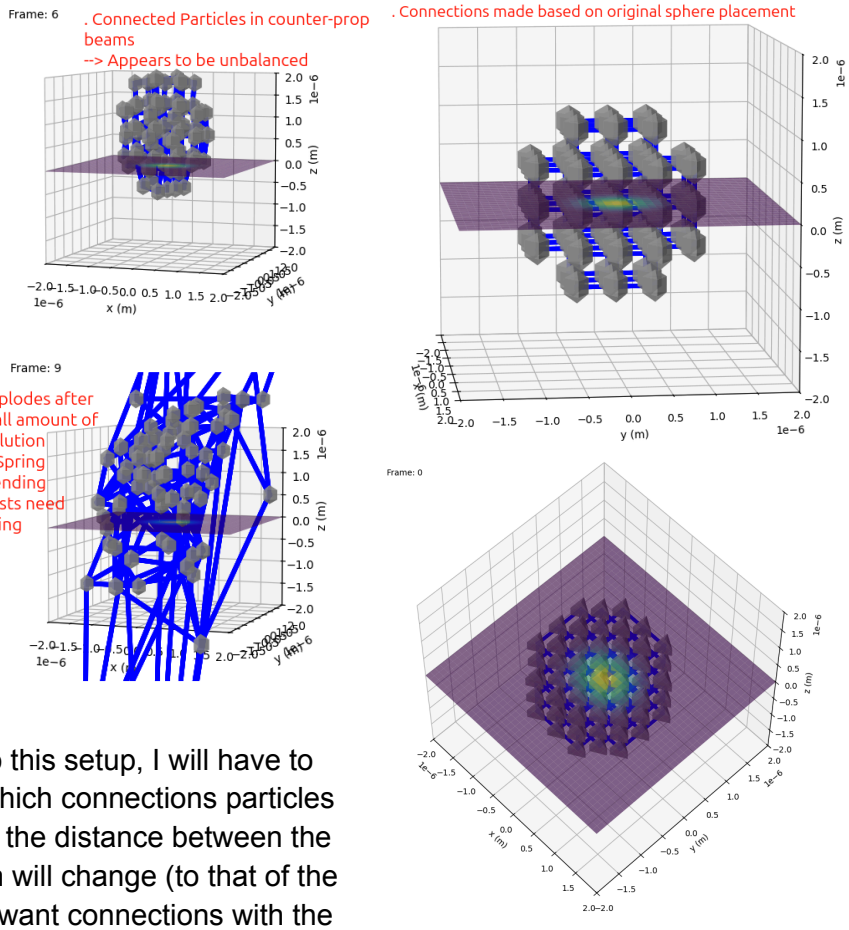




To do this, first I will use the stress plot written before to get the forces on each particle in a set of counter-propagating beams (Gaussian) acting to stretch a particle (will consider a sphere for simplicity, but this was experimentally tested on RBCs). First I will assume the stretching occurs in each axis according to some fixed function (for now, a linear stretch in the propagating axis, and a ***sqrt*** stretch in the X-Y axes, which will preserve the volume of the sphere if considered as a series of summed rectangular dipoles). I can then examine the force felt in each of the axes for each scale of transformation used in order to see if a certain transformation results in a lower total force on the particle (which would imply equilibrium is being reached, hence a stable stretching degree). In order for an equilibrium to be reached, an opposing/restoring force will be needed, which will be the spring/bending force already implemented. To start off I will need to write a function to place these particles in a grid which then allow a certain amount of stretching and compressing (which I will specify through a ***critical_transform_factor*** will puts a bound on the scaling allowed), with particles of a given size placed into a formation for a mesh of some total overall radius. As a test, I will consider this setup without springs first (where we should expect

the particles to be infinitely pulled in the Z axis by the Gaussian beam, as there is no force to oppose this, e.g. draw out into a thin wire essentially).

. 20 Samples used for each graph
. Set critical transform factor to 2.0
. NO spring forces used in each
=> Would not expect this to reach a nice equilibrium
→ This system would be happy to stretch out into an infinitely thin rod, which may be seen here as the XY plane feels no net force but the Z plane feels an increasing force (maybe would expect the upper and lower contributions of this to cancel normally however).



In order to introduce springs into this setup, I will have to add another connection mode which connections particles based on an index I parse in, as the distance between the particles in each scaled situation will change (to that of the original sphere), however I only want connections with the original sphere's nearest neighbours, so it will have to remember the indices to connect to and parse this into the YAML for correct connections at any stretching degree.

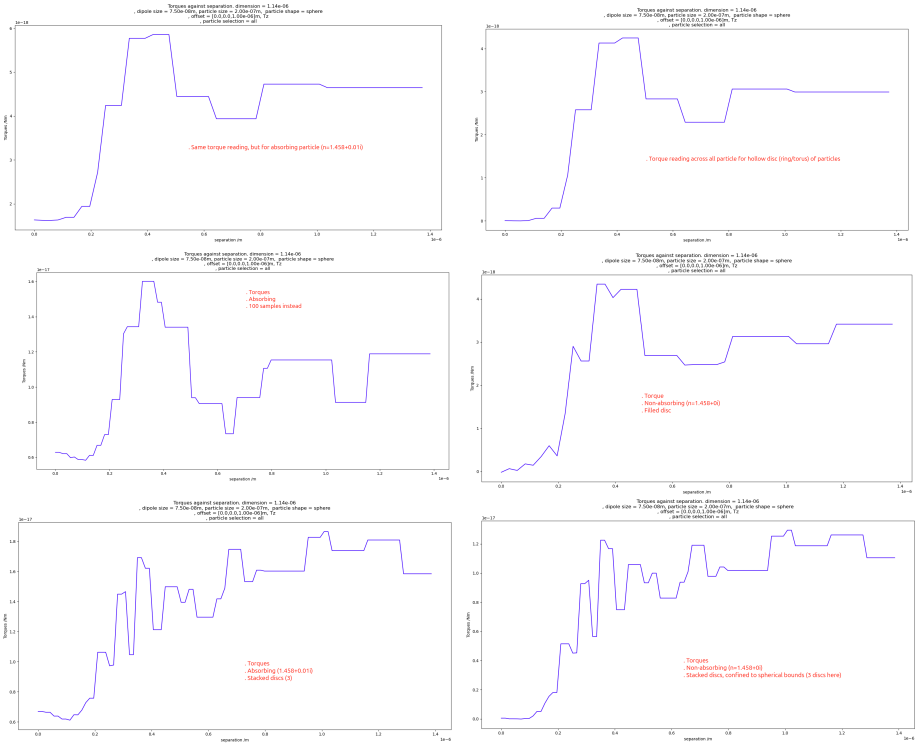
Using this and briefly checking the dynamics to make sure the behaviour is not unreasonable, we see that the system has a tendency to explode and to drift upwards. The exploding should be fixable through tuning of arguments, but the drifting is more tricky and I believe could be from the system having an exact central particle layer, hence if this moves one side (through brownian motion, say) then the whole system may be influenced to follow this. However, this may be a problem to do with the Gaussian beams having their focus at the centre, NOT offset, so the beams don't attempt to keep particles in the centre as well.

. Test see if a single particle can be trapped in the optical stretcher centre, make sure this makes physical sense
. Try to make the stretcher have offset focus points to possibly resolve this.
. Tune the constants to prevent exploding

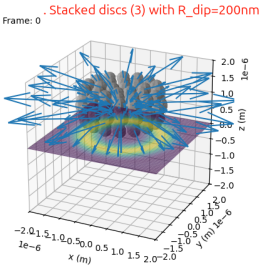
. Get plots for forces at various stretches with spring forces present
. Add main() arguments to YAML instead, and ensure is backward compatible / fix old implementations if not possible
. Think of other ways to test the flexibility / possibly go back to the bending rod and try see the variability in flexibility allowed there + investigate the difference between the spring forces & spring+bending forces (if bending is required/helpful)

14/2/25

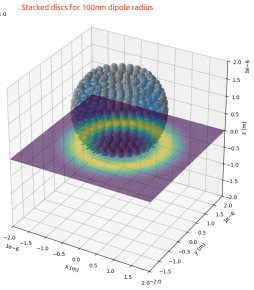
Samples on the disc and torus/ring were 50, and 100 samples used for the stacked discs.

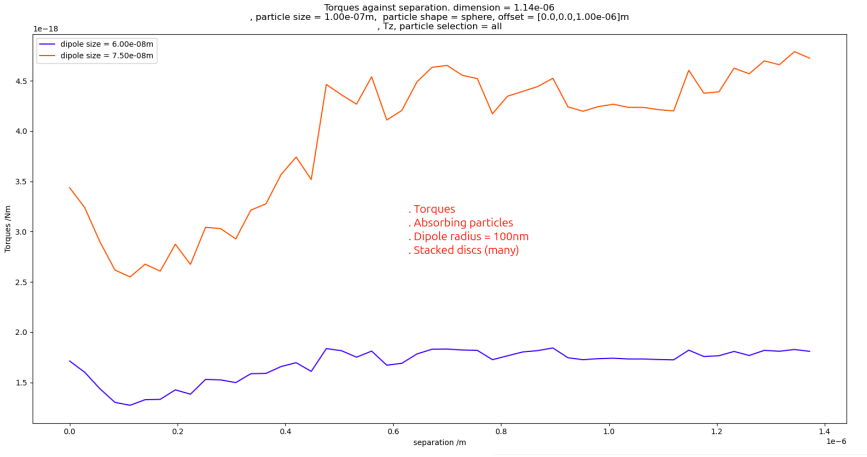


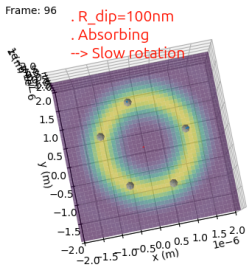
The tests here were made to confirm that the behaviour seen on the far right particle (measuring just force) do apply to the full shape in terms of torques seen. It can clearly be seen that this is true here as these plots replicate the behaviour for both absorbing and non- -absorbing particles.

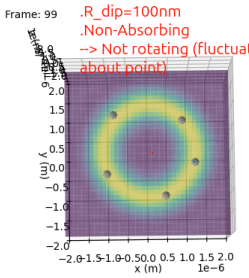


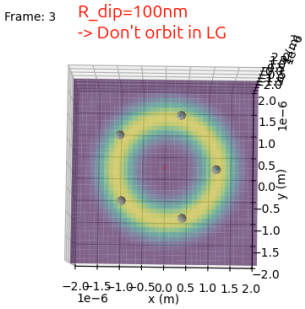
Further, testing with 100 nm dipole particles instead, which do NOT orbitally trap in a LG beam, we see the exact same behaviour, showing that when the particles are compacted together, we do see the behaviour of larger particles (e.g. get this orbiting effect in the limit of many particles). This also allowed the stacked discs example to be taken further as more layers could comfortably fit into the given radius, hence a much more accurate representation of sphere is generated, and with







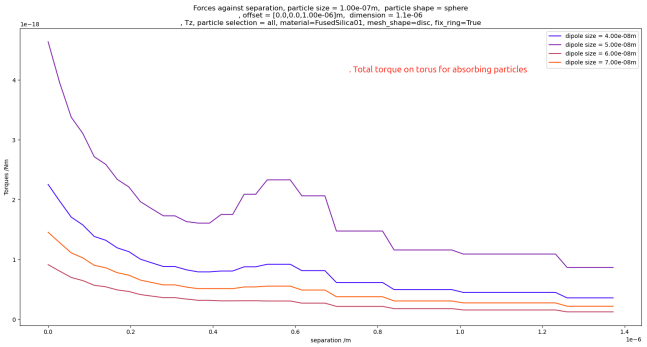


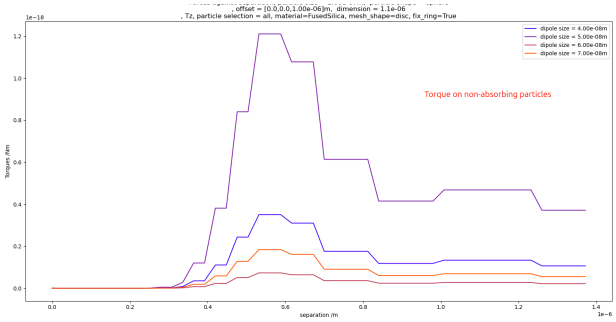


that we see an equally good asymptotic relationship for absorbing and non-absorbing particles (however, the absorbing particles do show a slight ‘lip’ at 0 separation, but this is likely small overlapping occurring due to my assumption in particles placement that the arc length covered by a sphere is its diameter, which is not strictly true, hence will lead to small errors for exact 0 separation when the particle sizes fit exact into the radius given, e.g. requires 0 separation and perfect particle size to be observed).

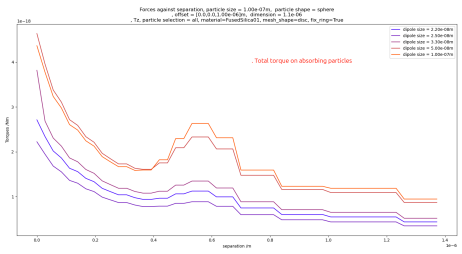
A bug was just found in the program where the read parameter “all” was not working due to reading the wrong particle number, but has been fixed now. This means the absorbing plot for dipoles of radius 100 nm is not the same now (200 nm radius particles still have the same behaviour).

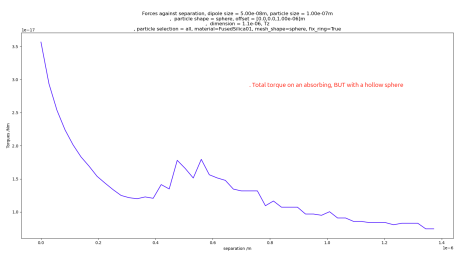
Considering just the forces on the 0th particle (far right, central disc) we see that force absorbing and non-absorbing plots you get the same behaviour but offset, however this means that the total torque actually increases with N for small separations then, because each particle now has a value tending to zero+some flat offset from the absorption, which means that as the number of particles increase the torque linearly increases too, hence we see total torque plots that look as though they tend to some increasing value (note however that the actual torque is still quite low here), whereas the non-absorbing plots do tend to zero as expected.





It is possible that this problem is a result of the number of dipoles used, hence with increasing dipoles the force on each particle will drop to prevent this infinite scaling. When testing this for different dipole numbers, we do not see much difference in the behaviour (even for very large numbers of dipoles, in the largest case ~3000 dipoles total), and so this behaviour is slightly unusual in that it leaves particle torques somewhat unbounded. This being said when this is tested at 0 separation, this means the particles will try to pack as tightly as possible without spaces UNLESS they cannot fill the space in an integer number of particles, in this case spacing themselves evenly with as many particles as

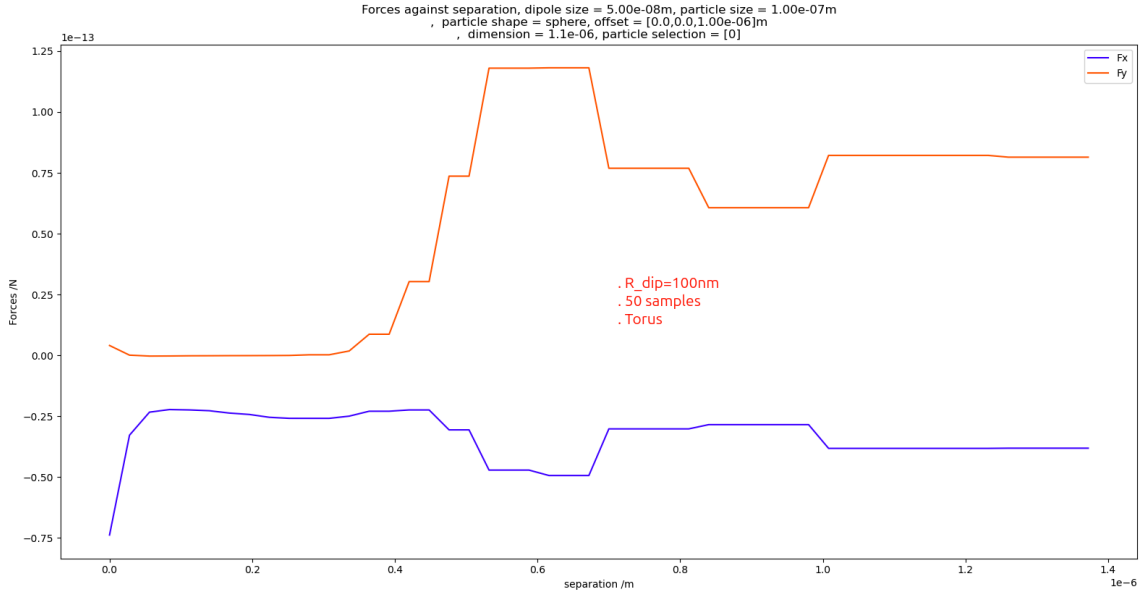


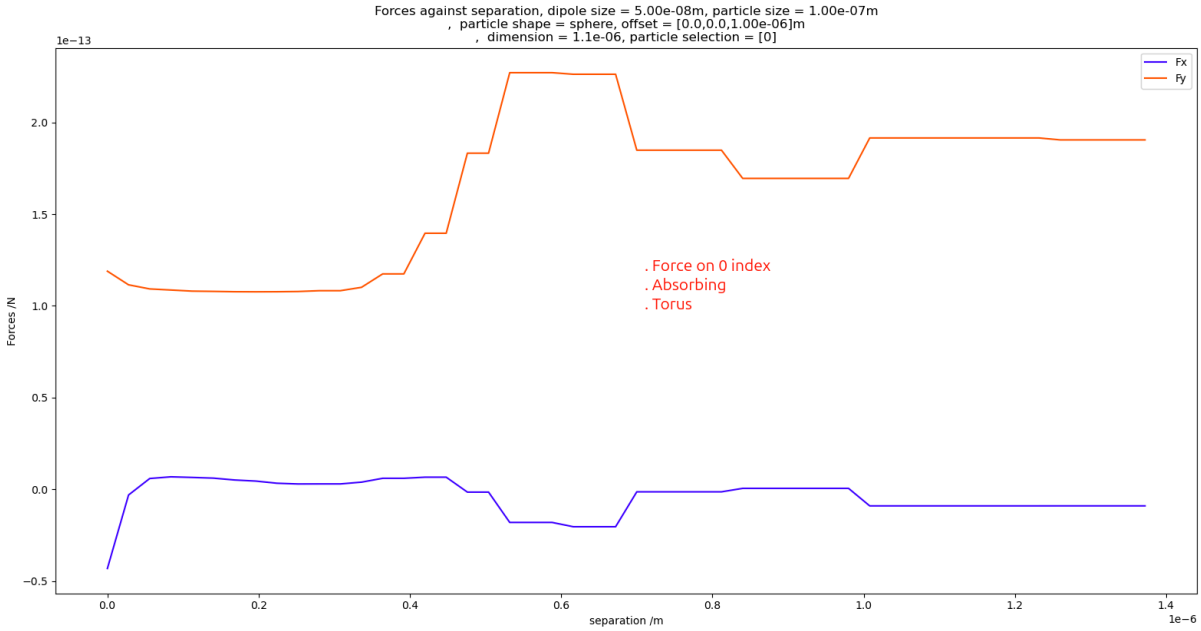


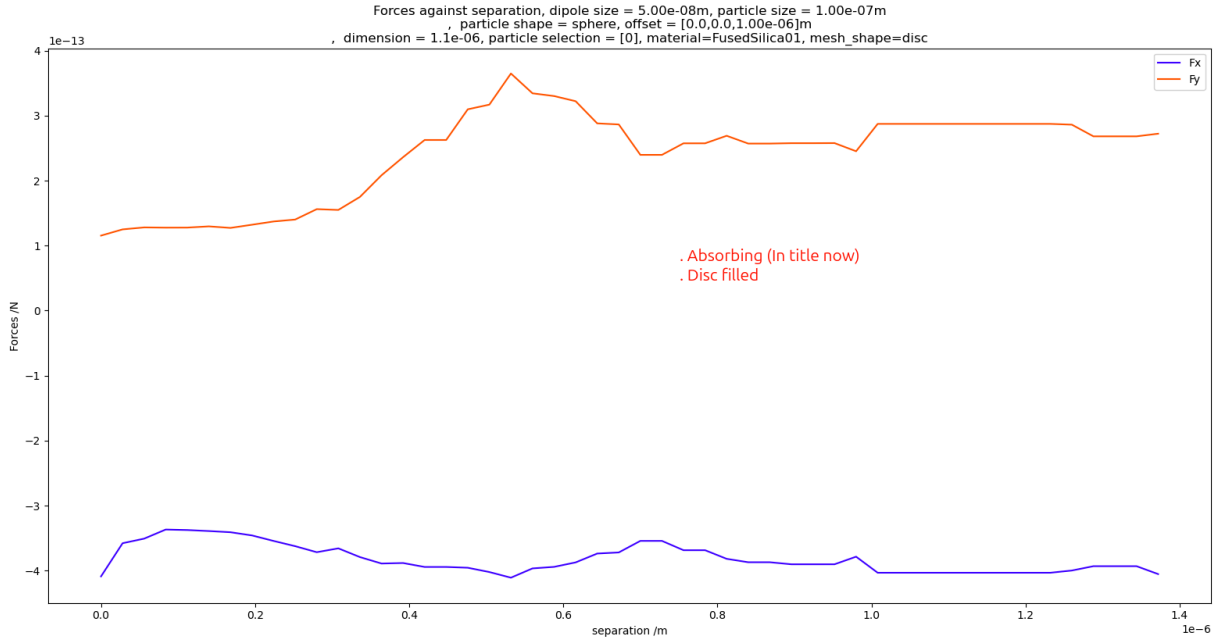
possible. This indirectly does set a limit, however it is strange for the torque to approach this value with such a large gradient.

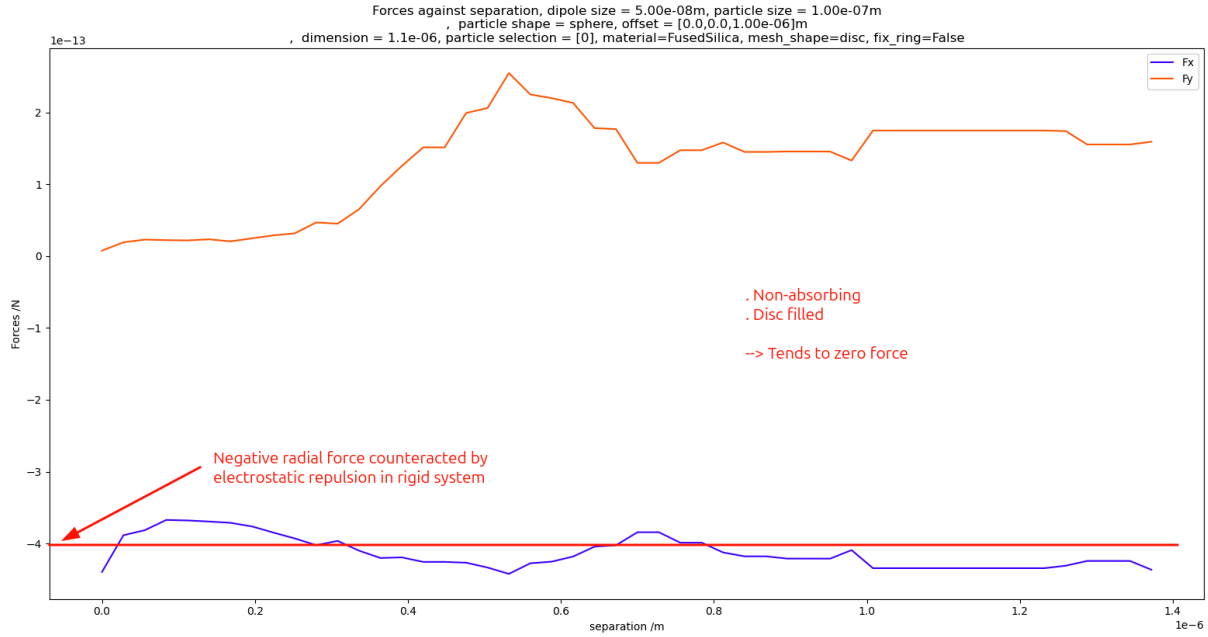
When testing the forces and torques for a hollow sphere, we see the same pattern as before but just with lower magnitude (~10x less), which suggests that the inner parts simply follow the same pattern as the outer parts (with greater radial force too that is cancelled by opposing sides), hence the torque regime is essentially unchanged.

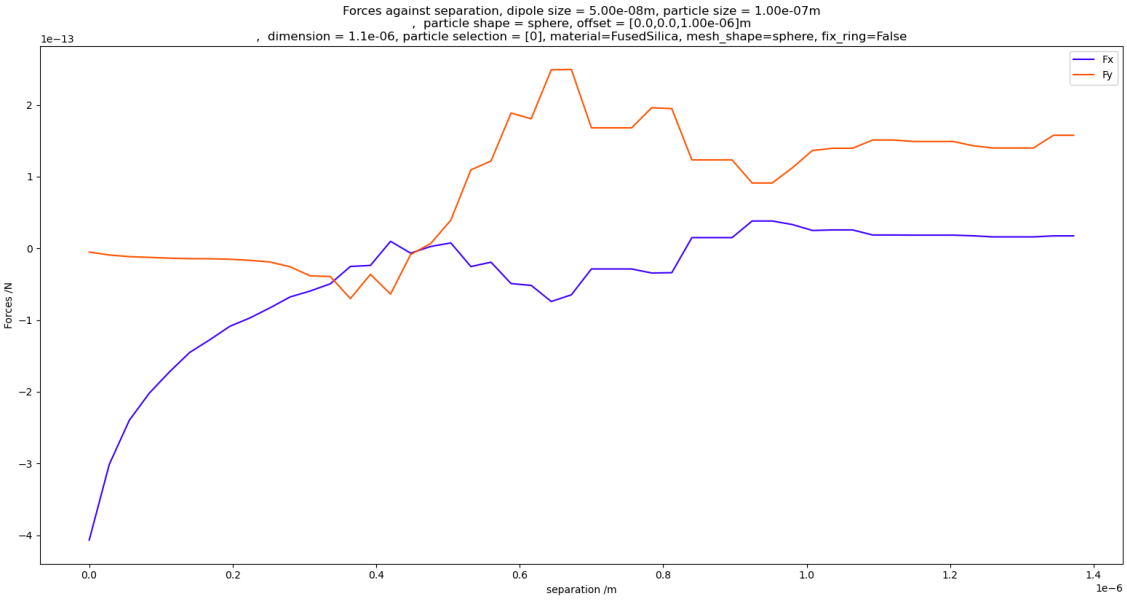
A bug we were encountering before was also, when doing torque plots, that the dipoles stored in the XLSX file exceeded the maximum number of columns allowed (16384), which caused remaining data to be defaulted to zero. We could increase this cap by storing the data in row format instead (which has a tolerance of ~1 million rows allowed), however unless very large systems are tested this is not a big issue, but would be good to fix at some point.

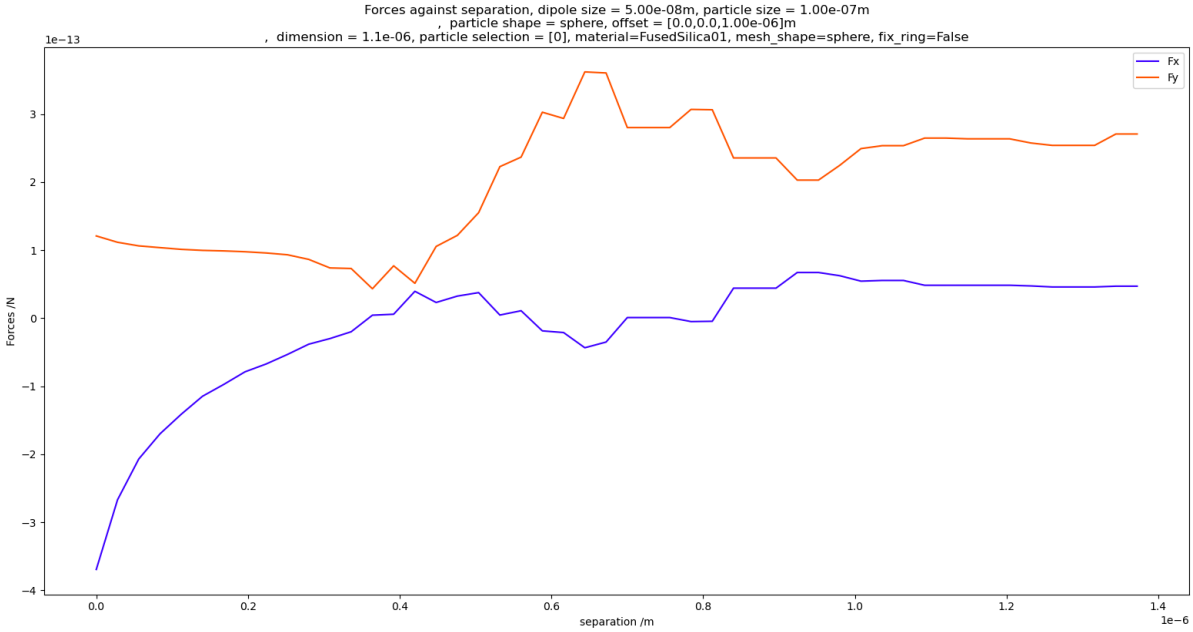

Forces against separation, dipole size = 5.00e-08m, particle size = 1.00e-07m
, particle shape = sphere, offset = [0.0,0.0,1.00e-06]m
, dimension = 1.1e-06, particle selection = [0]
1e−13
Forces /N
separation /m
1e−6
Fx
Fy
. R_dip=100nm
. 50 samples
. Torus

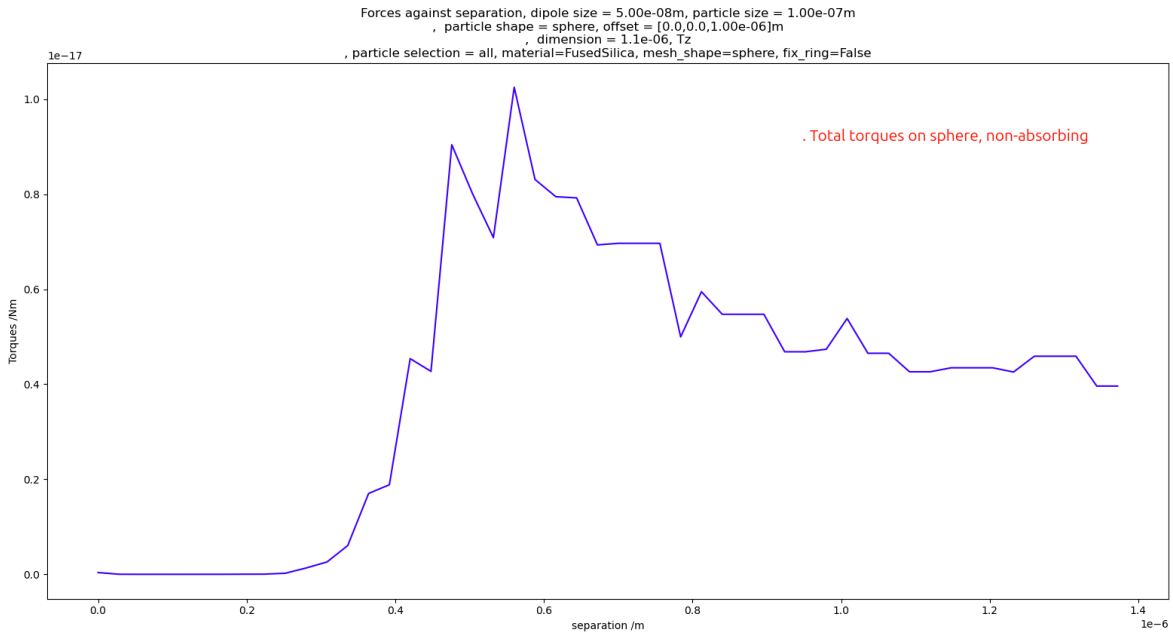

Forces against separation, dipole size = 5.00e-08m, particle size = 1.00e-07m
, particle shape = sphere, offset = [0.0,0.0,1.00e-06]m
, dimension = 1.1e-06, particle selection = [0]
1e−13
Forces /N
separation /m
1e−6
Fx
Fy
. Force on 0 index
. Absorbing
. Torus

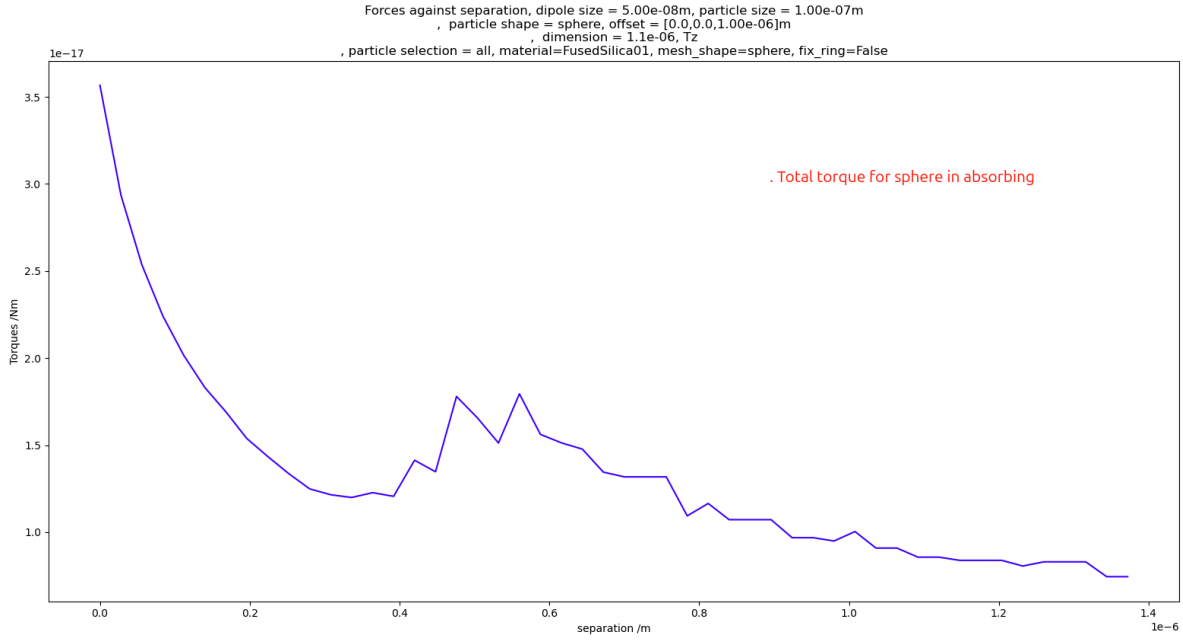

Forces against separation, dipole size = 5.00e-08m, particle size = 1.00e-07m
, particle shape = sphere, offset = [0.0,0.0,1.00e-06]m
, dimension = 1.1e-06, particle selection = [0], material=FusedSilica01, mesh_shape=disc
1e−13
Fx
Fy
. Absorbing (In title now)
. Disc filled
Forces /N
separation /m
1e−6


Forces against separation, dipole size = 5.00e-08m, particle size = 1.00e-07m
, particle shape = sphere, offset = [0.0,0.0,1.00e-06]m
, dimension = 1.1e-06, particle selection = [0], material=FusedSilica, mesh_shape=disc, fix_ring=False
1e−13
Fx
Fy
. Non-absorbing
. Disc filled
--> Tends to zero force
Negative radial force counteracted by
electrostatic repulsion in rigid system
Forces /N
separation /m
1e−6


Forces against separation, dipole size = 5.00e-08m, particle size = 1.00e-07m
, particle shape = sphere, offset = [0.0,0.0,1.00e-06]m
, dimension = 1.1e-06, particle selection = [0], material=FusedSilica, mesh_shape=sphere, fix_ring=False
1e−13
Fx
Fy
Forces /N
separation /m
1e−6

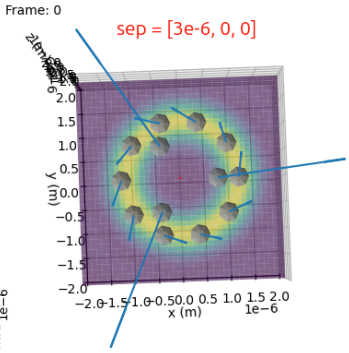

Forces against separation, dipole size = 5.00e-08m, particle size = 1.00e-07m
, particle shape = sphere, offset = [0.0,0.0,1.00e-06]m
, dimension = 1.1e-06, particle selection = [0], material=FusedSilica01, mesh_shape=sphere, fix_ring=False
1e−13
Fx
Fy
Forces /N
separation /m
1e−6

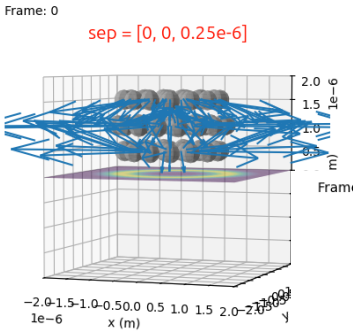

Forces against separation, dipole size = 5.00e-08m, particle size = 1.00e-07m
, particle shape = sphere, offset = [0.0,0.0,1.00e-06]m
, dimension = 1.1e-06, Tz
, particle selection = all, material=FusedSilica, mesh_shape=sphere, fix_ring=False
1e−17
Torques /Nm
separation /m
1e−6
. Total torques on sphere, non-absorbing

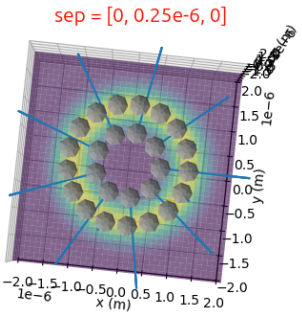

Forces against separation, dipole size = 5.00e-08m, particle size = 1.00e-07m
, particle shape = sphere, offset = [0.0,0.0,1.00e-06]m
, dimension = 1.1e-06, Tz
, particle selection = all, material=FusedSilica01, mesh_shape=sphere, fix_ring=False
1e−17
Torques /Nm
separation /m
1e−6
. Total torque for sphere in absorbing

13/2/25

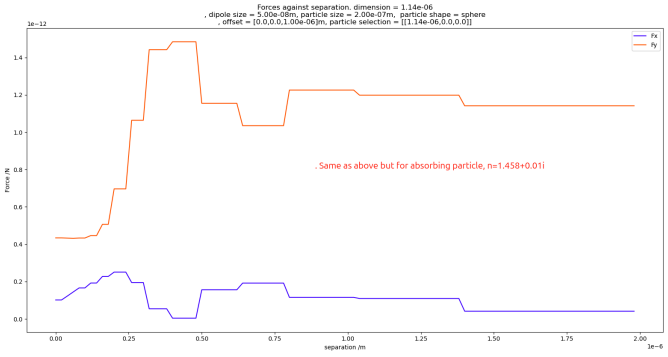
With the new particle generation model setup to perform some force tests inside a Laguerre beam, I will now try to convert the simulation function I made to use the new force calculating system my partner was working on at the end of yesterday (cleaner than previous version which allows new force tests to be added easily). The main goal is to assess what the tangential component of force is acting on a particle at the edge of the mesh, and then to consider the torque acting on the body as a whole. This will be considered for absorbing and non-absorbing particles, which should hopefully give evidence as to whether the model is working correctly.



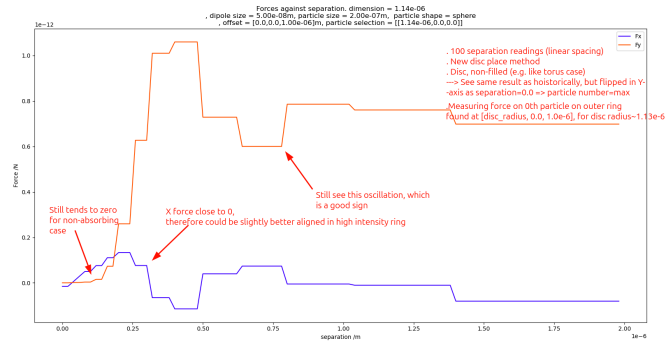




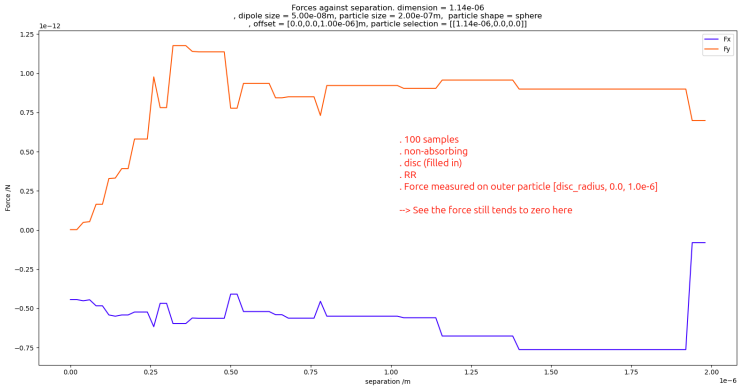
My approach is to first test a ring of particles on the intensity ring, get it to a position as before where they circularly orbit and trap (and we see that tangential forces tend to zero when the ring is filled), then I can perform this with the disc filled in, then extend to N layers for a sphere. Currently the separation of particles in the ring is not working, which is crucial in order to have more spaced out rings to test how it behaves as it converges. The figures show 1 version of separation chosen, where each ring has the same conserved total separation across its layers, but this meant that inner rings had a major disadvantage (smaller separation for outer ring is huge for inner rings), so I switched to using a raw offset between each particle in each layer, which gives more evenly spaced results. When toggling off the inner section, I get a force graph for the rings as seen on the right, which shows the same behaviour as before when considering a torus in a LG beam but with some jumpiness caused by different separations resulting in the same particle number/position as it is transitioning to the next value where another particle is lost, as expected (good confirmation that this is still working for this model however). Similarly we get that the force tends to some fixed offset value when the particle is made absorbing too.

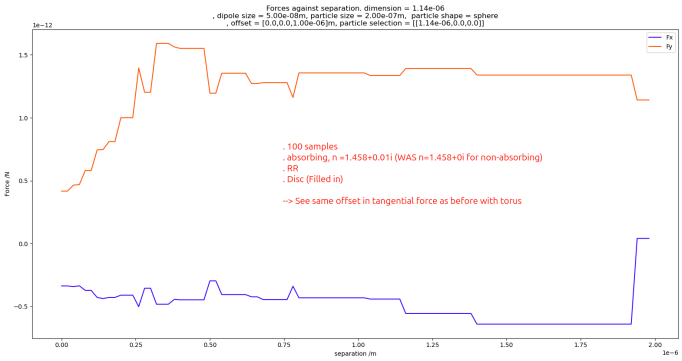


Now I can consider the case where the disc is filled in, wherein measurements are still taken at the outer edge particle on the rim of the disc. Doing this gives a plot quite similar



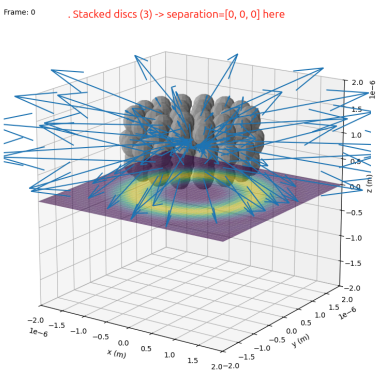
to the torus case where the tangential force tends to zero for a closely packed disc, but here the X force sits at a more constant negative value (particle being pulled in towards the large mass of particles scattering with it) and we also the tangential force for more separated particles sits a quite stable value just under 1.0e-12N (possibly the optical binding fluctuations smoothed out by the larger number of particles either side of the particle). Therefore the trajectory of this particle points towards an inward spiral from purely optical effects, which in reality would be repelled by inter-particle forces. This would not be true for the tangential force seen. Repeating this for an absorbing particle gives the same offset in tangential force observed as in the torus case, and the X force is entirely unaffected.

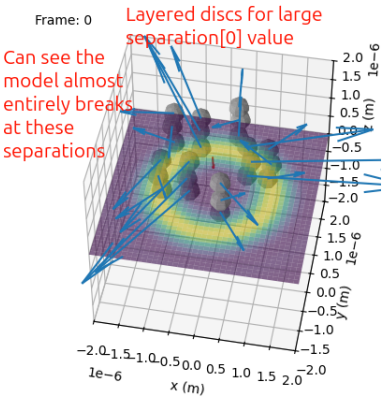




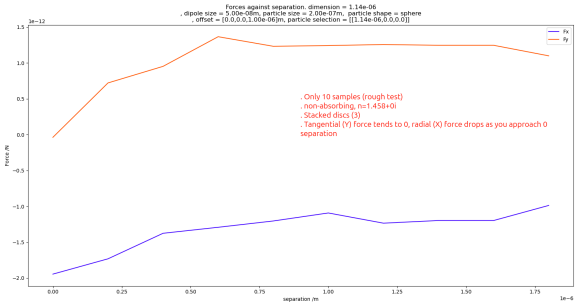
Something to note about this however is that this is just considering the separation component which only varies the separation of particles within each ring of the disc (e.g. distance between subsequent disc layers and stacked discs is kept to zero), so we could also consider the effect of these other parameters on the model.

Also, for the sphere case, with particles of this size we can only fit 3 layers into the sphere model, meaning the particle loses some of its ‘spherical resolution’ in this case, however the argument of stacking discs still applies.

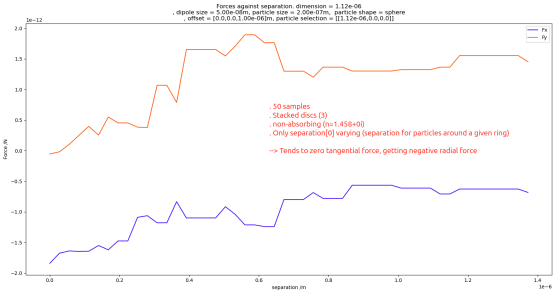




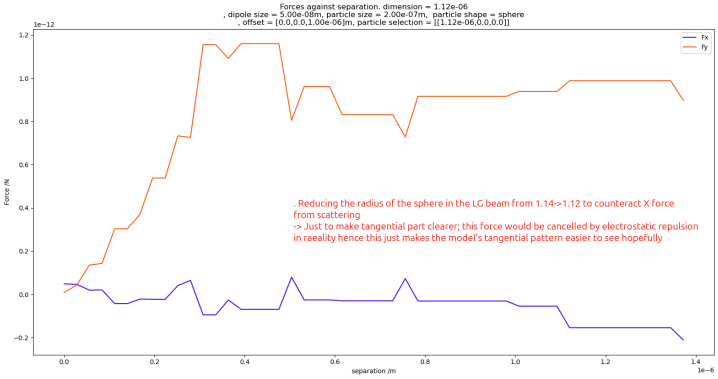
Looking at stacked discs (with reducing radius to approximate a sphere) we see for this case, where only 3 stacks can be created, generated symmetrically about our offset (e.g. every layer above is mirror with a lower layer, prioritising a central layer to be formed always, hence no even stack numbers allowed) that, in a rough model, tangential forces tend to

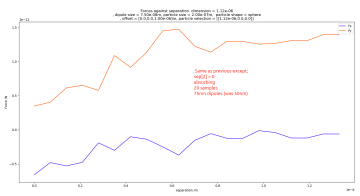


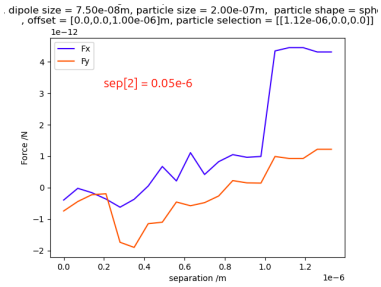
zero still, but now with a more negative radial force (likely due to the increased number of particles scattering and so pulling the particle inwards, as this is measured at [disc_radius, 0.0, 1.0e-6] again). To try and correct the X-force seen in these I will try reducing the radius of the particle to sit slightly within the beam (not exactly on the ring), so the opposing gradient from the beam can combat the pull from the other particles scattering, as this is purely an effect seen.

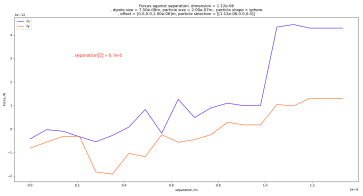


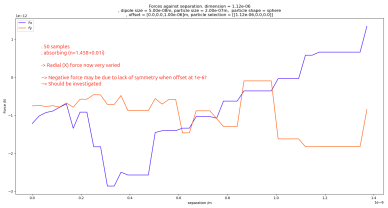
Now that the X force has been reduced to zero here, I will try the sphere case again and perform another correcting radius change. However, it is harder to experiment in the sphere case as it takes far longer to run (~3x particle number for 3 layers, therefore flat 3x longer + additional times for N^N-1 additional calculations introduced). NOTE; These 2 sphere graphs also had some separation[2] component (0.1e-6m) in an attempt to reduce problems when separation[0] was large.









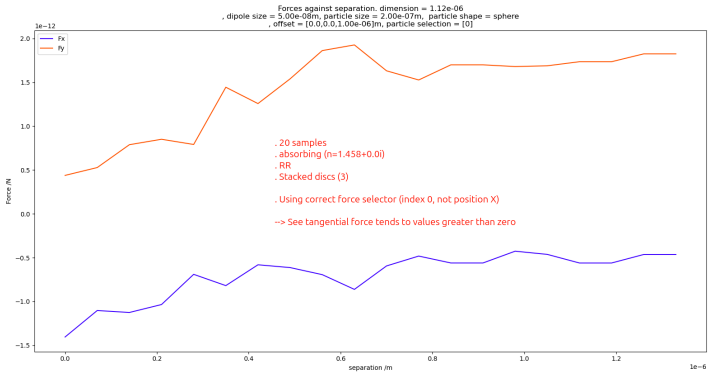


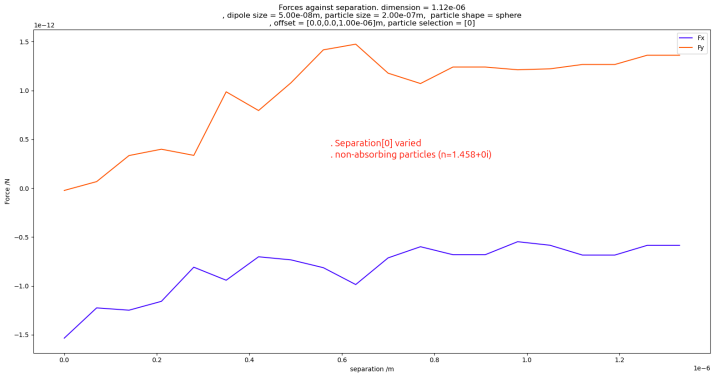
A difficult bug was just found, demonstrated by the figures on left, where in my force calculator which pulled the forces on a particle closest to a point was giving 2 separate results when it was ***accidently reading the oth index and 4th index particle <u>nondeterministically</u>***. I believe this was caused by a float error where these two particles were equidistant from the point I attempted to sample at, which caused a tie in position that, and so a particle would win this tie based on a floating point error (some of these graphs were found by simply changing the particle from absorbing or non-absorbing, or from changing the separation[2] from one value to another minutely different one, or even simply changing the names/positions of variables and prints). Now I will pull forces from the particle just from its index, as for this particle on the far right edge it is already located at index 0, hence it is very easy to switch to this method. When doing this, I get the following behaviours on the right for stacked discs showing an offset and no offset for absorbing and non-absorbing particles respectively.

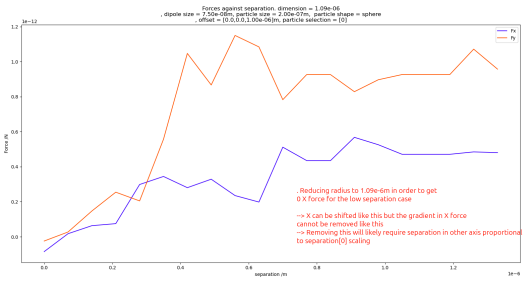
Further areas to explore some more;

(1) Consider effect of separation[1],[2] on the forces, as well as separation[0,1,2] all at once -> May show morning valid results for the less accurate system, however the main point of interest is the accurate system so not massively important

(2) Would be good to do plots for both the total forces on the sphere -> Hope for zero hence it rigidly stays still in centre (trapped), but then also evaluate the torque on the sphere to see if it is rotating and by how much -> extra confirmation for the results seen, as it has been observed to want to rotate for this particle on the edge of the radius, but this will confirm that this is generally true across the perimeter (not just via symmetry arguments).

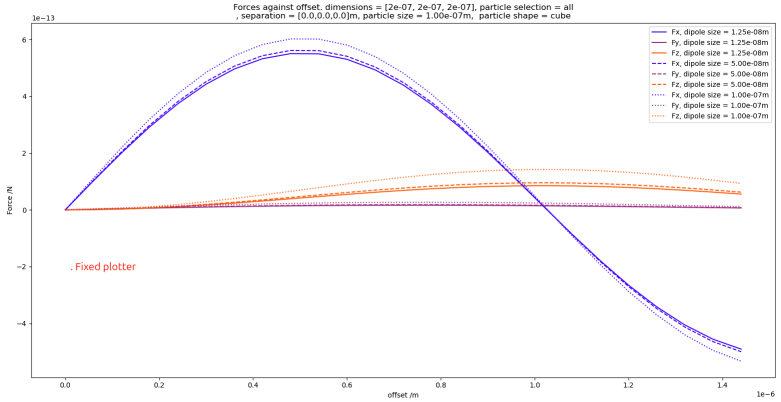




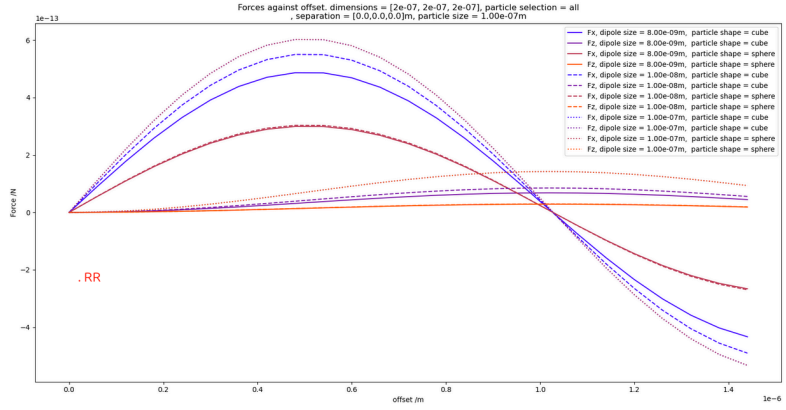


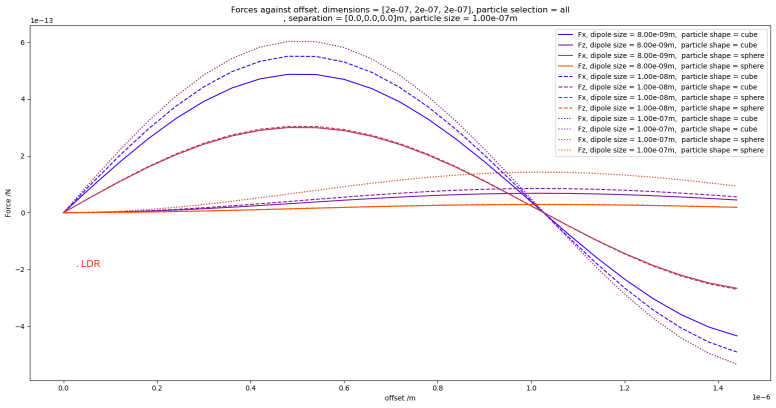
12/2/25

Me and my partner looked at the plotter again this morning to check it was working correctly, and found a problem where it summed values on subsequent plots for a given measured value (e.g. Fx reading in a given problem). This was not too difficult to solve and now the plots should be correct, however it should be noted that previous plots which plotted multiple graphs will have errors where subsequent graphs have values from the previous summed, meaning the first line plotted for a given value will be correct, but later plots will be offset by the previous, and plots with just a single line for each parameter would be unaffected.

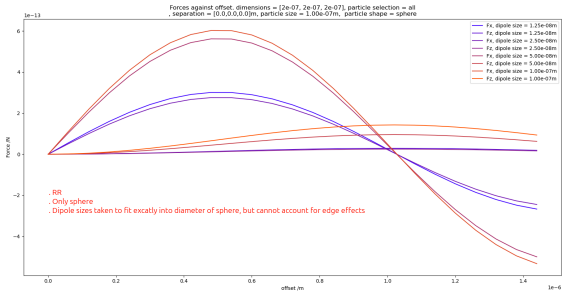


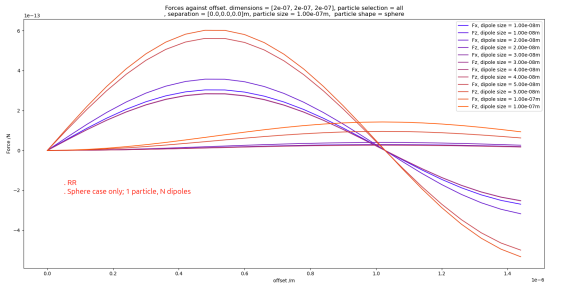
Redoing the plot from yesterday now gives a more promising result that for dipole radius 100e-9m you get the same plot as the single dipole case (as should be the case), and then following plots show a fairly small reduction in force. Note as well that the forces here are the sum across the full particle (all dipoles) to match the full force seen on the single dipole. This case of a single particle split into many dipoles should tend to the perfect result for DDA, so the fact that this is quite similar to the Z forces from just a single dipole when using RR polarisability shows that whilst single dipoles are less accurate, but still does a decent job at modelling the situation, and importantly does not completely disregard the Z forces. This means that for a bead-and-spring model you could refine the system down to single dipoles and not encounter a sudden discontinuity in the model, it would simply just lose accuracy as the dipole size becomes large relative to the particle. Interestingly though the Z forces for spherical primitives seem to drop to zero almost completely for a refined system, which while I would expect a lower force as it has less dipoles than the cube, it is

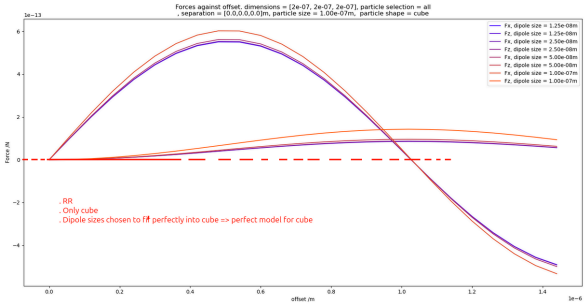


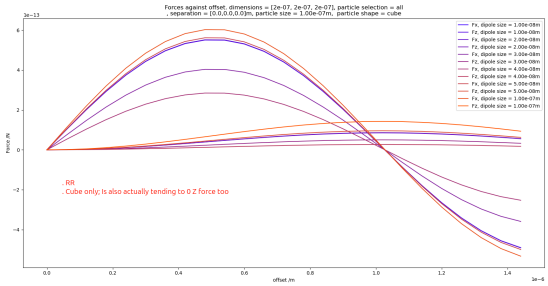


unusual that it would essentially experience no radiation pressure forces, so I will investigate this more (Note that here a sphere particle shape means that the 1 particle in the system is a



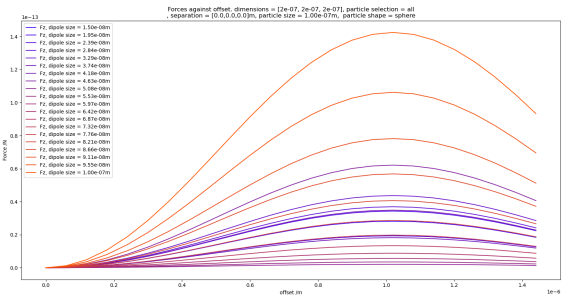




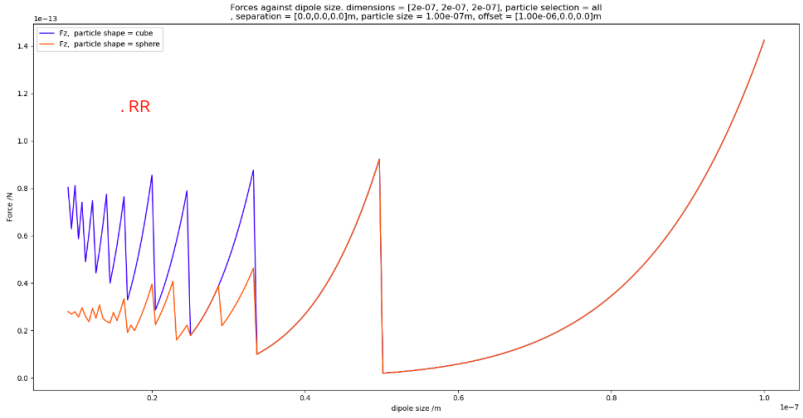


sphere instead of cube, and then that is split into many dipoles, so we are simulating two separate shapes for a sphere and cube but are just interested in if the Z forces are non-zero and comparable to the single cubic dipole).

Clearly here the sphere tends towards having zero net radiation pressure force as the system becomes more accurate, whereas the cube does tend to non zero value when dipole sizes are chosen to fit the particle well (you do see some forces which are much smaller than this if this is not considered however, and dipoles are arbitrarily smaller).

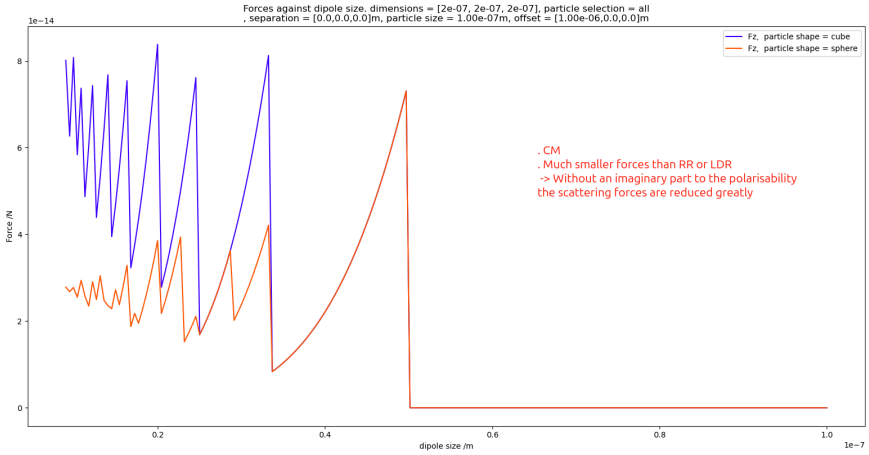


Varying the dipole size for a single spherical particle shows that the ‘optimal’ dipole sizes for a sphere are less obvious, as by simply reducing the size you encounter increasing and decreasing



These sphere and cube refinements converge to a different force than is seen with a single dipole, hence there is error in the approximation (obviously as you have only 1

dipole), but the radiative behaviour is not excluded and so the single dipoles do not have some sharp cutoff in behaviour when, for example, particles are refined down to single dipole sizes and split for us in a bead-and-spring model, they will instead show a standard and expectation reduction in accuracy until the model is refined again (to have smaller dipole size relative to the particles involved).

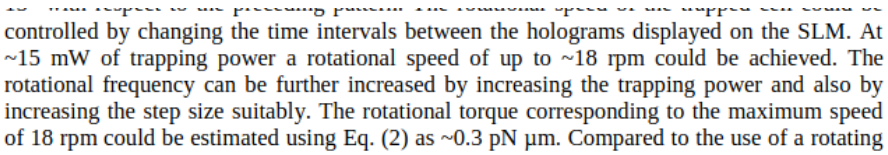


This all suggests that a single dipole, or small collection of dipoles, or large collection of dipoles all include all forces expected, with an increasing accuracy as the dipole size is made smaller (and the particle number is increased), and that there is not in fact any disruptive behaviour when dividing down to a single dipole particle as long as you are using the RR or LDR (or some other self-interacting correction) polarisability.

Now that we have more confidence in single dipole particles not causing problems as we subdivide particles, I want to now consider the large sphere/torus in a Laguerre beam again for further evidence that splitting a DDA system still gives valid optical forces as more particles are added into the system to better approximate the overall sphere/torus. Whilst there doesn't appear to be experimental testing of a torus in a Laguerre beam, there is evidence for (1) a large sphere in a LG beam, and (2) many small spherical particles in a beam, both with absorbing particles. Hence if we could adapt the torus case we tested to be a large sphere instead, this would have more evidence supporting it and so the validity of a split model (note that there does not seem to be any evidence for non-absorbing particles since these are likely harder materials to encounter in reality).

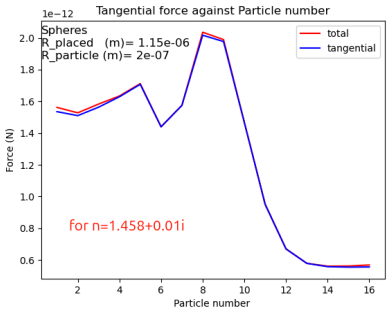
Replotting the ***spheresInCircle*** plot with some absorption, as seen in the paper above, we see that the torque (/tangential force component) now tends to some small value and not strictly 0 (~0.6e-12N), which does agree with the sentiment of the experiment. However, this is when considering a torus modelled with spheres, not a sphere modelled with spheres, hence there could be some difference here so we should test a sphere case as well now OR try to find where this has been tested for a torus.

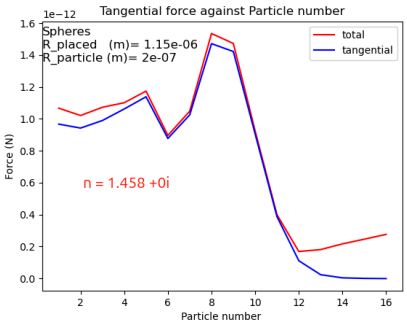
NOTE; Interestingly the paper ***"Optical orientation and rotation of trapped red"*** cites roughly the same torque as we observed with the torus on these bean-shaped particles tested, which

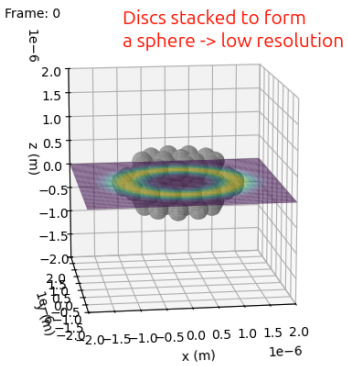

controlled by changing the time intervals between the holograms displayed on the SLM. At ~15 mW of trapping power a rotational speed of up to ~18 rpm could be achieved. The rotational frequency can be further increased by increasing the trapping power and also by increasing the step size suitably. The rotational torque corresponding to the maximum speed of 18 rpm could be estimated using Eq. (2) as ~0.3 pN µm. Compared to the use of a rotating

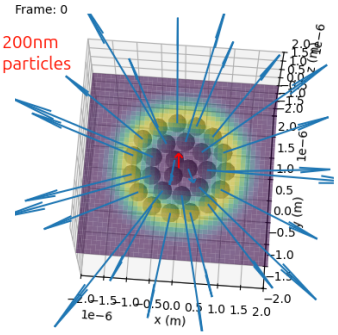
were of sizes between ~2.5 and 6 micrometers.

To start testing this for a sphere, there is the worry that it will be too large to be able to use reason dipole numbers without extremely high computational times, hence I will first find what the torque on flat disc is (fill in the centre of the torus), then consider stacking the discs, which will then approximate a sphere when stacked enough. The hope is for this to also show a small torque on a large sphere, which was what was experimentally tested (have not been able to find evidence for a torus being experimentally tested). To do this, I am modifying a previous YAML that generates a grid of particles of a triangular lattice to now be constrained to a circular region, so I can generate particles of the given size in a lattice in this region, with a given separation between each. This should give a uniform arrangement of particles throughout the disc and hopefully still have enough resolution at the corners to be curved here and get the corresponding correct torque (I may have to generate this using a more circular arrangement if this fails).

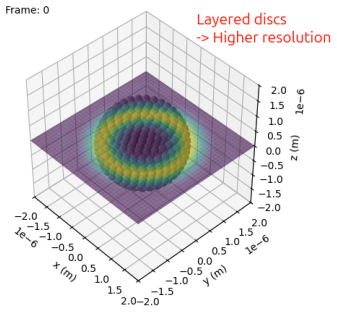


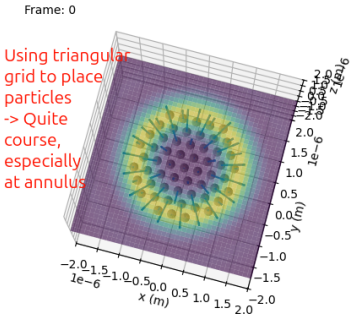


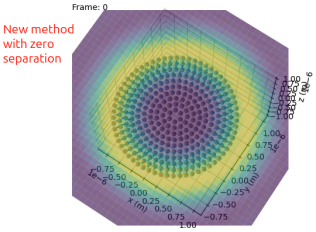


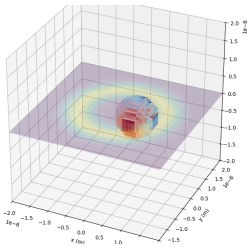


Testing this, the triangular lattice did appear to be too jagged, as it seems likely that the particles on the high intensity ring will dominate circular motion, and the others inside will simply to to move radially towards the intensity ring from purely E gradients, therefore I need to ensure that the outer layer of particles on the ring are nicely distributed as they will likely have the largest effect. Changing to a method that tries to evenly place particles in concentric rings, we get a better looking distribution, which gives discs that can be stacked to give









spherical shapes. As expected, for a filled in sphere this requires many particles/dipoles, hence is very slow to run so I will try to run tests on just the discs first to gauge the motion, then slowly introduce more layers.

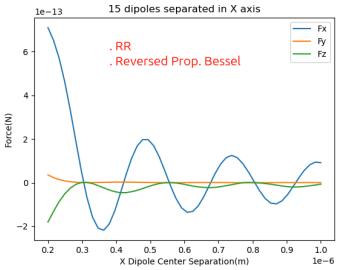
. Could try to find the exact rotation of particle with the torque seen (paper found sphere of ~8 micrometers to have angular velocity of 0.2 rad s^-1, which we may be able to calculate using physical parameters of the sphere such as a moment of inertia, approximate mass, etc).

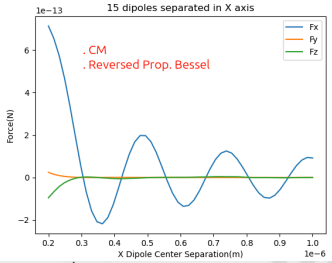
11/2/25

I want to;

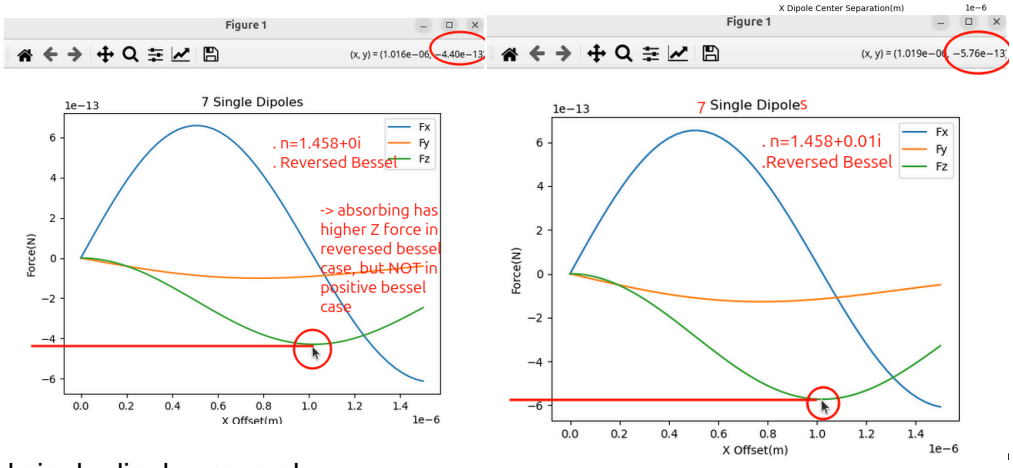
(1) Test different polarisabilities to see if other corrections work to include the radiation pressure like RR
(2) Try reversing the beam to see why the radiation pressure seems to be in the opposite direction to the beam propagation (or maybe the beam propagation is opposite to the direction I believe it is in)
(3) Ensure that the Z radiation pressure I am getting is accurate (e.g. compare it to a perfect sphere/cube case and see if it tends to the correct value)

The main goals of these tests are to ensure I have tested a range of options as to how these single dipoles can behave, to make sure they all agree, and to try and replicate the errors Simon had seen in previous work, which required BOTH the RR term to be included as well as several dipoles to be included in order to get a correct radiation pressure, as currently I only require the RR too start seeing these pressure forces (and including additional dipoles helps prevent this effect for CM, but since it was already seen for RR for a single dipole then no difference is seen here). Therefore the two cases I am considering here are either that the RR term allows single dipoles to work correctly (correct radiation pressure force), so single dipoles are valid in bead-spring models as they have not lost any influence that other particles felt when grouped together, OR that even with RR single dipoles are not correctly modelled (when separated form the group at least) hence care should be taken if refining a bead-and-spring model to this degree. The force difference plot I did actually showed that for dipoles with low separation, the CM and RR term behaved very similarly (e.g. radiation pressure terms were correctly induced by surrounding dipoles, even though all were solitary), hence both models work for low dipole separation, so a bead and spring model would work for low dipole separation in either model.





First I will test the beam to be propagated in the opposite direction. This is performed by changing the ***rotation*** argument in the YAML, which performs the rotation on the beam once its fields are computed. Rotating the separated case and single dipole cases show reversed Z forces, as would be expected with a radiation pressure. However, when the particles



involved are made absorbing with imaginary refractive index (n=1.458 +0.01i, instead of just n=1.458+0i), we see the Z force increases, which is the opposite to what occurred in regular propagation Bessel case. Revisiting that result, it appears the program had been running the wrong set of parameters as now the absorbing case always had a larger magnitude of force compared to the non-absorbing case (for the regular and reversed Bessel beam), hence this is acting as expected, where a higher absorption leads to more radiation pressure, and the beam flipping does result in this force flipping. Therefore this means the Bessel beam is actually propagating from underneath the sample, e.g. in the (0,0,1) direction, NOT the (0,0,-1) direction (a convention ADDA uses I believe). Now that the direction of this Z force is understood, and so the RR term can be seen to be emulating the radiation pressure, I would like to see how well it performs this (e.g. if the radiation pressure it is generating is accurate when the dipole is on its own OR in groups). We expect the force to be different when the RR term is used in a group as it is a corrective factor to the CM in general, but I would like to see how the force on a single dipole particle corresponds to a refined particle made of many dipoles. We will consider a single refined particle later.

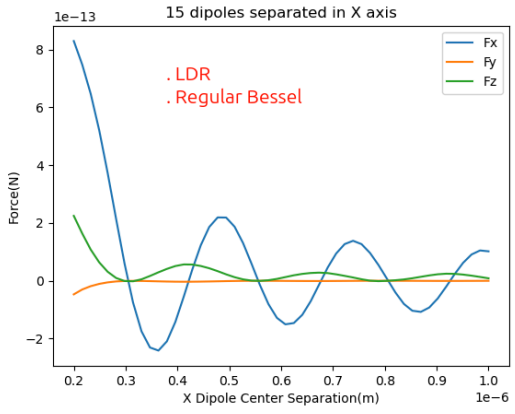
```
polarizability: LDR, [1.1775525e-33+2.06185014e-36j 1.1775525e-33+2.06185014e-36
j
 1.1775525e-33+2.06185014e-36j 1.1775525e-33+2.06185014e-36j]
81  dipoles generated
81  dipoles generated
81  dipoles generated
81  dipoles generated
```

As a quick note, this tests above have shown that the force in the Z axis is a radiation pressure (not just part of the field gradient as you enter the intense annulus of the beam) because it is not visible in the CM case with a single dipole, but clearly this dipole still experiences forces from field E gradients as it moves in the X axis.

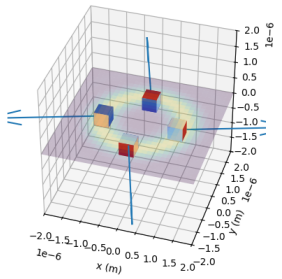
To compare what Z forces we are getting in a model and ideal case, I have added the LDR polarisability too as this is supposed to have the same radiation pressure / self-interaction corrections, so it would be good to see if the radiation pressure is seen here too, how strong it is and if it tends to some true value at a different speed to RR. LDR is formulated with the same component as RR in its denominator, just with an additional term included too. As can be seen in the screenshot above, this is a small correction to RR, but will increase if S is made non-zero (currently propagating in the Z axis, circular polarisation in XY => ***a***.***e***=S=0) meaning LDR Will have more impact.

```
polarizability_LDR =  [1.19682728e-33+2.12990163e-36j 1.19682728e-33+2.12990163e-36j
 1.19682728e-33+2.12990163e-36j 1.19682728e-33+2.12990163e-36j]
polarizability_RR  =  [1.1775525e-33+2.06185014e-36j 1.1775525e-33+2.06185014e-36j
 1.1775525e-33+2.06185014e-36j 1.1775525e-33+2.06185014e-36j]
```

```
a0 = (4 * np.pi * 8.85e-12) * (dipole_radius ** 3) * ((ep1 - epm) / (ep1 + 2*epm))
polarizability_type="LDR"
match polarizability_type:
    case "CM":
        polarizability = a0
    case "RR":
        #a = a0 / (1 - (2 / 3) * 1j * k ** 3 * a0/(4*np.pi*8.85e-12))
        polarizability = a0 / (1 - (2 / 3) * 1j * k ** 3 * a0/(4*np.pi*8.85e-12))
    case "LDR":
        # From ADDA Manual; https://github.com/adda-team/adda/blob/master/doc/manual.pdf
        # Works best for lower imaginary refractive index components
        # NOTE; Currently assumes only 1 beam present
        b1=1.8915316
        b2=-0.1648469
        b3=1.7700004
        n = ref_ind[0]      # NOTE; Assumes all particles have the same material
        d = dipole_radius   # dipole separation
        for beamname in beaminfo.keys():    # Read propagation and polarisation for a single beam (will default ot reading last beam, should only use with 1
            beam_prop = beaminfo[beamname]["rotation"]
            beam_pol  = beaminfo[beamname]["jones"]
        S = np.dot(get_propagation_vector(beam_prop), get_polarisation_vector(beam_pol))     # a=Propagation direction, e=Polarisation vector (for beam), ca
        polarizability = a0 / ( 1.0 - ((b1 +b2*np.dot(n, n) +b3*S*np.dot(n, n))*((k**2) / d) +((2 / 3) * 1j * k ** 3))*(a0/(4*np.pi*8.85e-12)) )
    case _:
        polarizability = np.ones(n_particles)
        print("polarizability not recognised, defaulting to RR: "+str(polarizability_type))
```

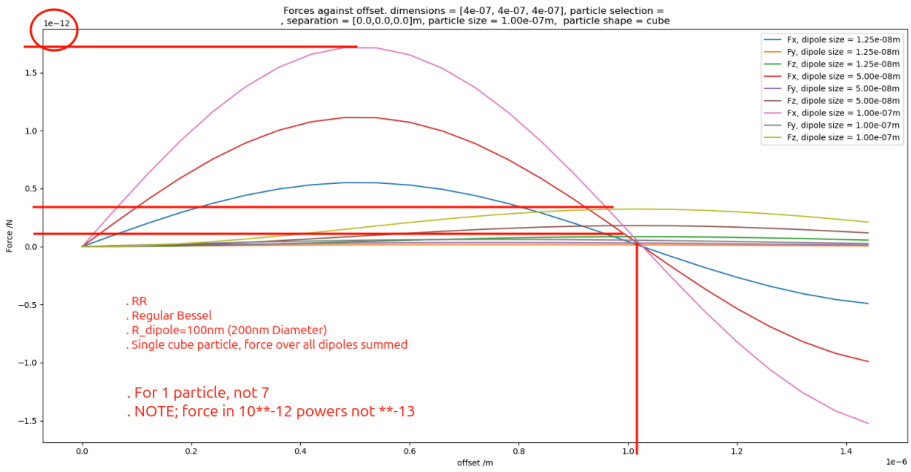


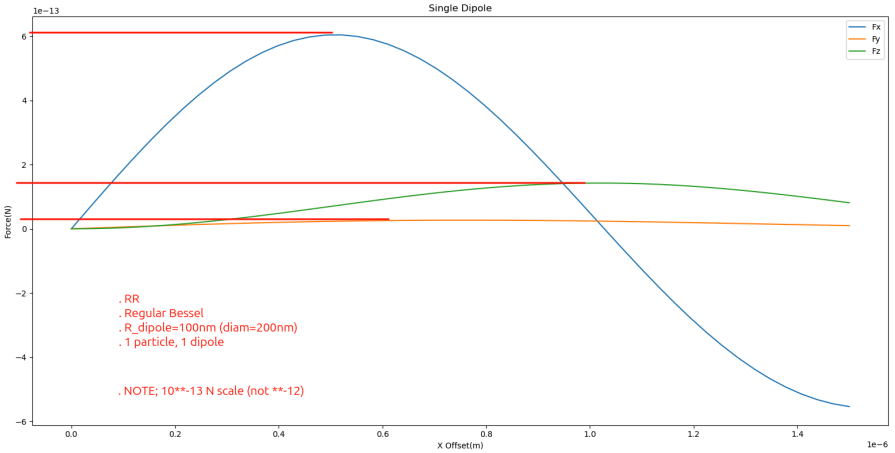
My partner has also been working on getting a plot of the stresses on the particles in a system working. This colours faces of the cubic bounding box for the primitive (spheres and cubes tested so far) with the force magnitude travelling through that face, with blue for high negative flux and red for high positive flux (away from and along the normal of the face). This is to be used to show how a flexible system may want to deform, when only considered optically, in some beam and so could be stepped according to these stresses to deform the particle.



Notes for tomorrow;

- Looks as though the refined cube is reading to the level of forces seen in a single dipole case, but perhaps may overshoot to 0 force. Hence I need to test these smaller dipole sizes mainly to focus the plot on this region of interest.

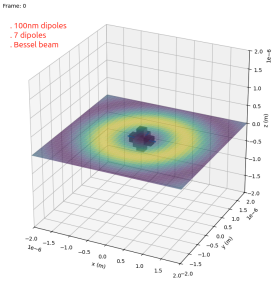


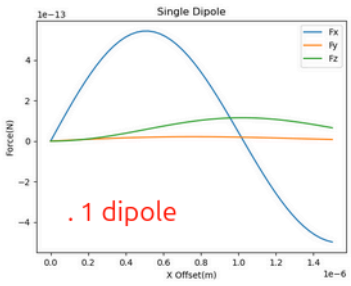


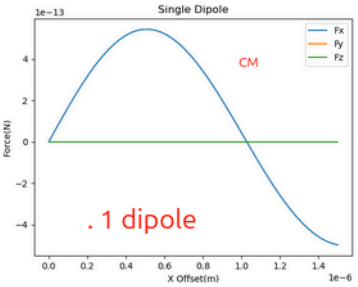
- --> Need a very convincing argument that the RR or LDR is doing exactly what is required / is over or under shooting by how much => how well will it model reality? Is this improved by having several side by side? If so, is each individual dipole working well or do forces need to be considered as a whole?
- Make sure the plotter is working properly for this mode, as it had some issues on a previous plot. Also, should redo the force filtering system to make it easier to use.

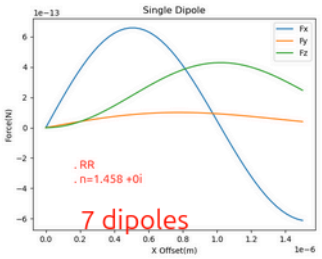
10/2/25

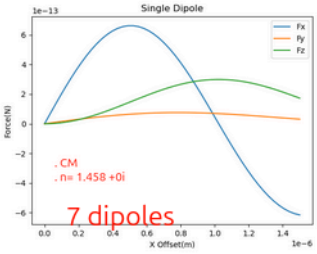
I am now considering larger particles in an attempt to make the force differences more prominent between the CM and RR tests. I also switched to using a Bessel beam as this is where Simon originally encountered this problem.

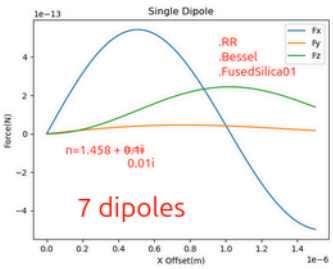


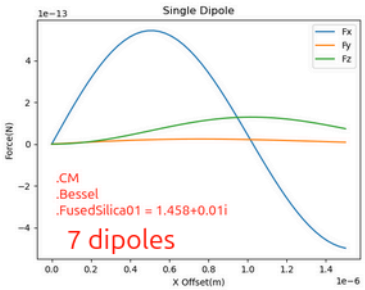










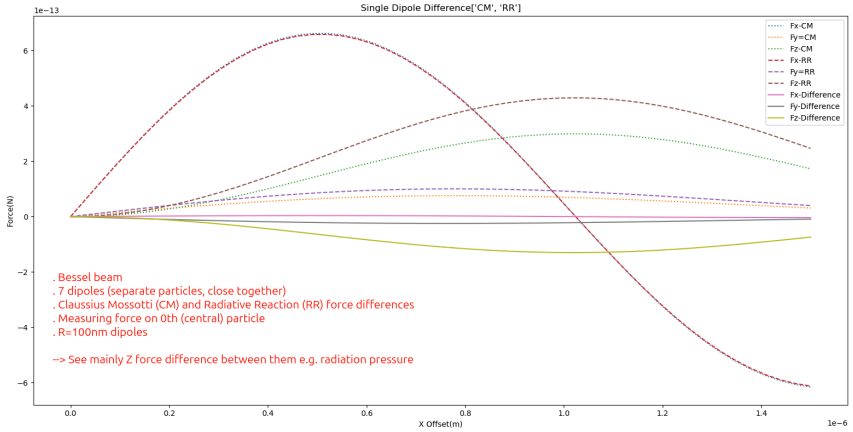


Forces on a system of 7 dipoles (6 surrounding a single dipole in a ‘cross’ shape) show this effect but increased further, and now introduces a scattering force in the Z direction on the 0th (central) dipole.

```
python SimulationVaryRun.py single_dipole_exp
Using YAML: SingleLaguerre.yml
polarizability: CM, [1.83993141e-32+0.j 1.83993141e-32+0.j 1.83993141e-32+0.j
 1.83993141e-32+0.j]
 Simulation Step:  0
```

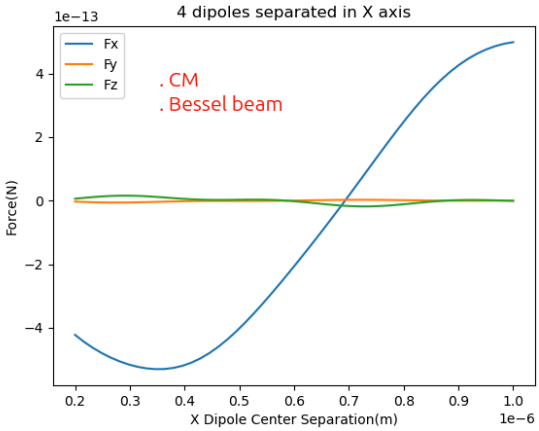
```
python SimulationVaryRun.py single_dipole_exp
Using YAML: SingleLaguerre.yml
polarizability: RR, [1.83855525e-32+5.03006424e-34j 1.83855525e-32+5.03006424e-3
4j
 1.83855525e-32+5.03006424e-34j 1.83855525e-32+5.03006424e-34j]
 Simulation Step:  0
```

Considering this for absorbing dipoles also shows a change in Z force, which highlights that the previous Z forces have all

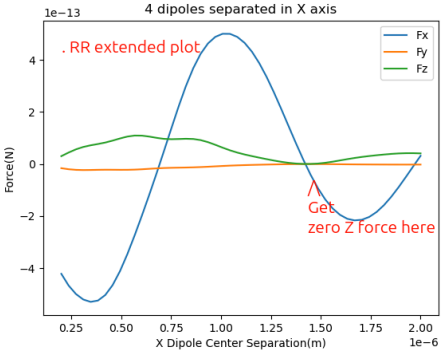


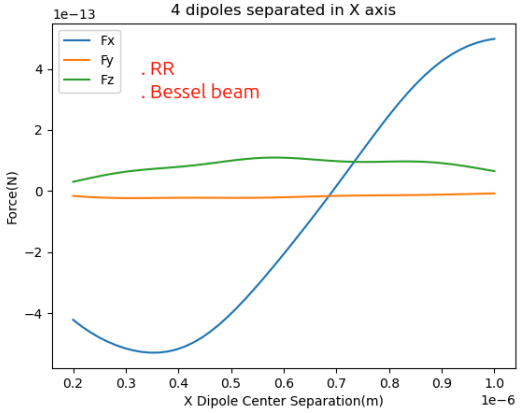
been positive for a beam propagating in the negative Z direction, hence the absorption now present actually reduces the Z force experienced by the 0th dipole.

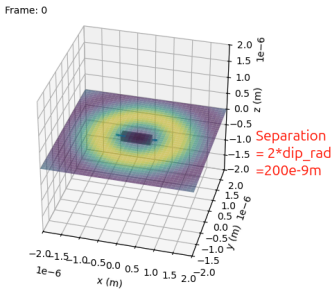
A plot for the difference in CM and RR force components is then done to highlight this difference, where it can be seen that these changes have mainly affected the Z forces.

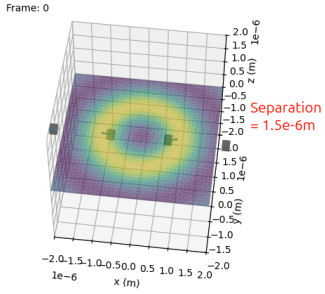


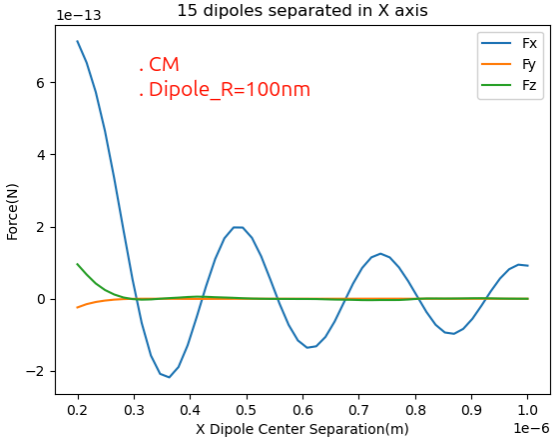
Doing plots for separated single dipoles of R=100 nm, it can be seen that the CM form has Z&Y force~0, whereas the RR term has a maximum value reached when the particles are separated by roughly 0.6e-6m (therefore 3 dipole diameters between each particle), and has a zero again at ~1.5e-6m (therefore 7 to 7.5 dipole diameters) which intersects with a zero X force too (e.g)

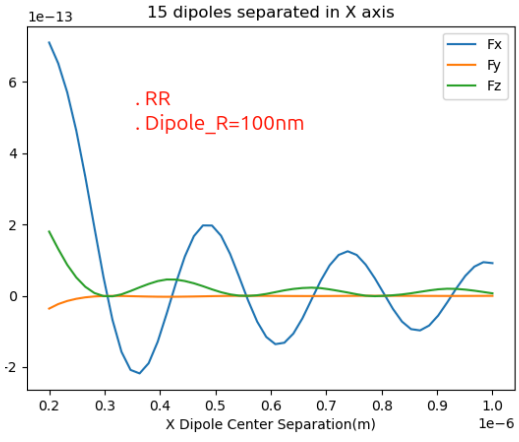












8/2/25 - 9/2/25

Having spoken with my supervisor yesterday, the next things to test are;

(1) Test systems with small dipole numbers, particularly for single dipoles, which are known to show radiative forces (worsened by using bad polarisability regimes, such as not using 'Radiative Reaction' over Claussius Mossotti). This is a big concern for splitting the mesh into many smaller particles, however may only be a problem at the extreme 1 particle limit, and may also not be a problem for separated, but sufficiently close, single dipoles. Rayleigh Hypothesis/LImit.

(2) Find papers which talk about the torques from optical tweezers on particles. There is supposedly a paper on the torque on a large sphere in a Laguerre beam, which did not rotate when non-absorbing and rotated for absorbing. When testing the Laguerre situation from before, we could instead consider segments from a sphere rather than torus and consider the same thing, which may now be better supported (as when considering a torus we found some on the torus, however there may have been problems with absorbing vs non-absorbing material here).

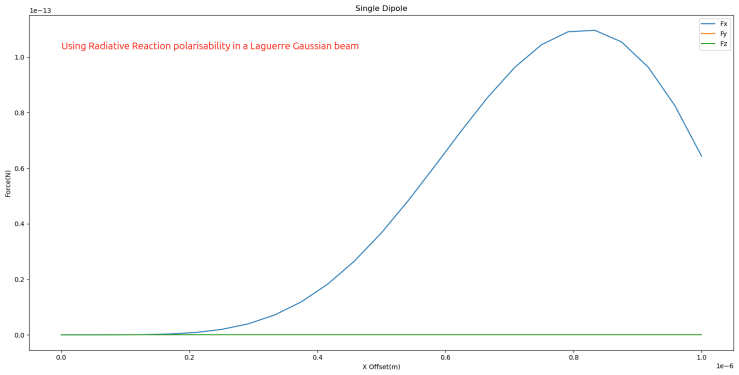
(3) Linking to this single particle testing, it would be good to test the forces and stresses on individual particles (when brought up to the 1 dipole per particle limit particularly) to see if there are problems with forces here, and to see if deformation can be found through these means (stress forces across the particle, which will show points of high/low stress which may want to deform if time stepped in theory).

It also should be noted for completeness that the torque calculator in SH_DDA does refer to the spin momentum of the particle, not its orbital motion, hence is not applicable to the torus tests my partner was considering yesterday (MST was evaluated individually at point son a torus and the torque found from this, which was slightly different to the torque found in SH_DDA, however this shows that this was fine as the torque referred to a different motion).

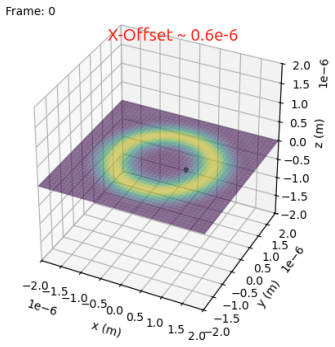
I would like to test this single dipole case again, as it was a problem Simon encountered in another project. Some details of exactly how this occurred should be repeated to ensure there are no false assumptions or misconceptions. To do this I will consider first a single particle within a beam, refined several times to view it at 1 dipole and then for several. I will then repeat this but for dipoles spaced apart to see the changes in force. I will also repeat using the ***Radiative Reaction*** and ***regular CM polarisability*** to hopefully see a difference in force, which will in part be due to radiative force loss if this idea is correct (which should be seen in 1 dipole systems but not many dipoles).

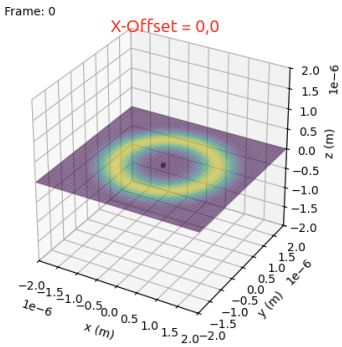
Also, Simon said the experiment in which the sphere in a Laguerre beam was tested my have been from a book written by ***MIles Padgett***, and it appears that he has only written one book called ***Optical Angular Momentum (2003),*** ***https://www.gla.ac.uk/schools/physics/staff/milespadgett/#publications,books,*** so I will check to see if this does have information on an experiment like this.

This book seems to talk a lot about observation of the spin and orbital angular momentum aspects of small particles trapped in a Laguerre Gaussian beam, but the paper ***"Observation of a single-beam gradient force optical trap for dielectric particles"*** considers trapping a larger particle (~10 micro m), however it did not consider the torques that occurred when this happened. Another paper however, called ***"Direct observation of transfer of angular momentum to absorptive particles from a laser beam with a phase singularity"*** also considers a Laguerre Gaussian beam and attributes the spin felt by the particles here to their absorption from an imaginary refractive index, when they are trapped in the central minimum of the beam (e.g. centred at the minimum centre). Therefore you could perhaps consider these two together to argue that the large particle would only spin when trapped like this if it were absorbing (which it was not when we tested it), however the 2nd paper only considered this affect for small particles, so perhaps bigger particles do not follow this rule (Rayleigh to Mie regime difference perhaps). Note in the Mie regime the scattered force is mainly in the direction of beam propagation (Z axis in our experiments), which should be a significant contribution to the total force.

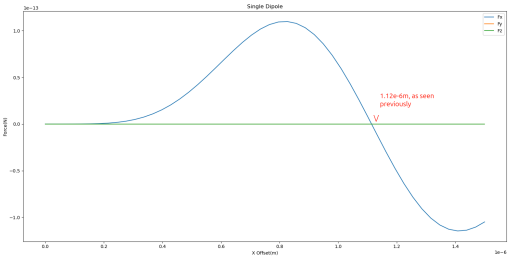


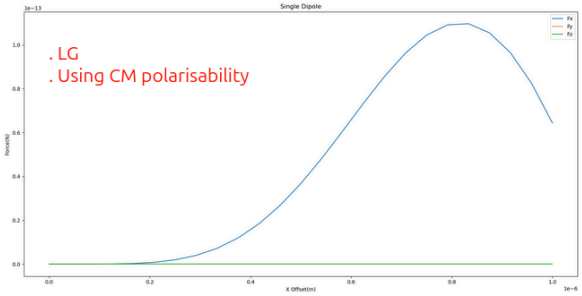
I am finding it difficult to find information on the forces experienced by single dipoles in a DDA system not having a radiation term, other than the fact that there are no other particles to scatter with, hence the 'scattering force' would be zero, which would lead to no force from say Mie fields through the dipole, just gradient forces. If this is the reason then I can see why it may be a problem, but it could be another effect that was being talked about. This theory would also mean that single dipoles separated would be fine so long as they are still calculating scattering between them, but perhaps for large distances this would break.



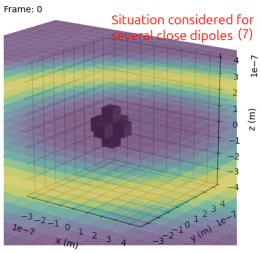


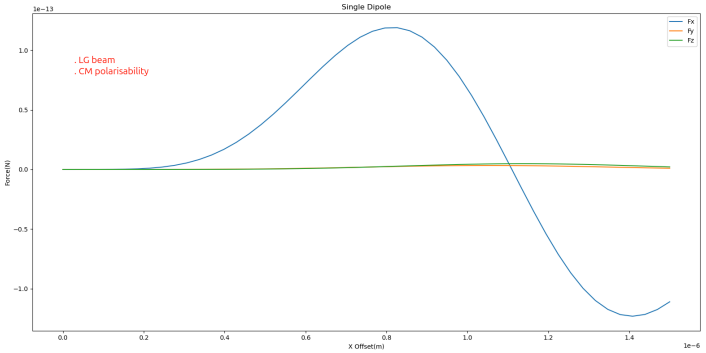
Testing the force on a dipole as it moves across a beam using different polarisability prescriptions

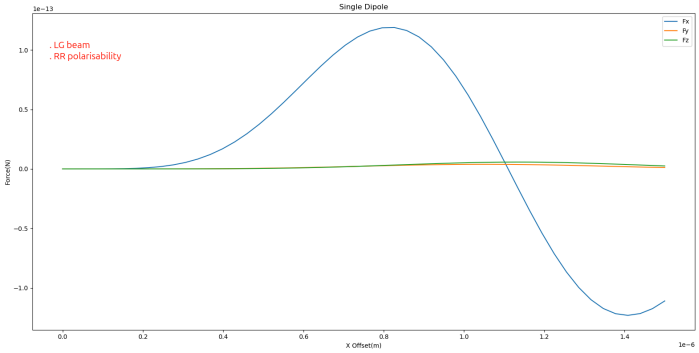


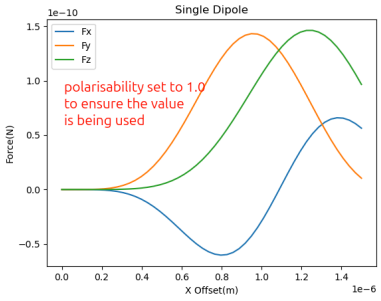


(RR and CM currently) we see an equal force for both, where it peaks as we approach the high intensity ring (at ~0.82e-6m), and if we extend the graph we see a zero force at exactly the intensity ring as expected. Considering this for 7 close dipoles instead, we get essentially the same effect with no difference in forces. This is unusual as we would expect the more accurate/realistic RR prescription to give different resultant forces, however this could be because the dipoles are still very small or because the 0th (central) particle is being measured only so forces may cancel.



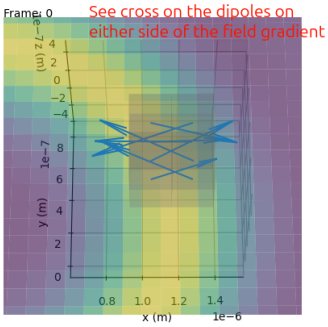


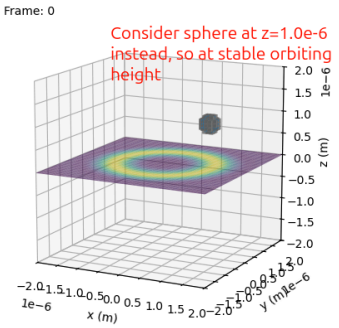


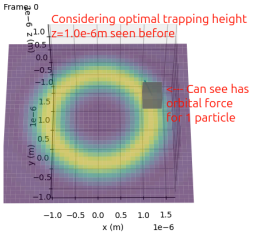


7/2/25

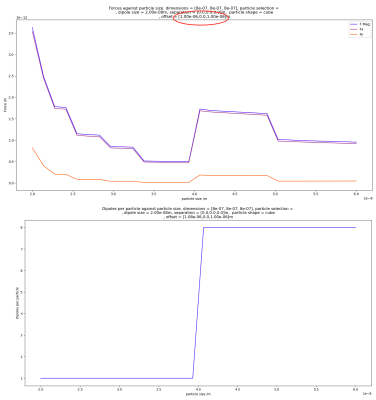
When considering the particle at the stable height we can see the forces at each dipole follow the beam gradient as expected, so we have a large tangential component as well as fairly large X components two (that oppose each other on either side of the total mesh).

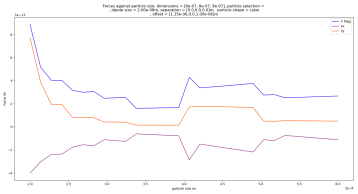


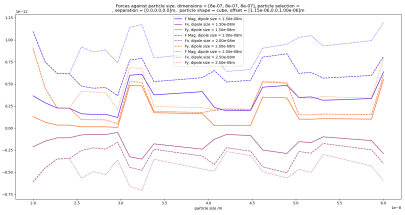


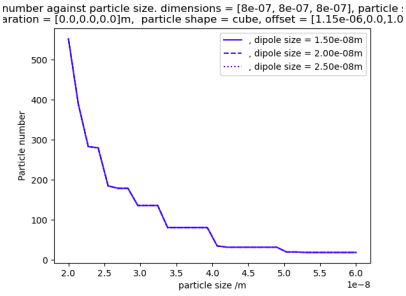


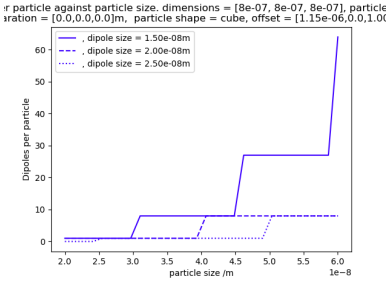
If the particle is then moved inwards slightly (in an attempt to remove X forces from the particle being improperly placed in the intensity ring gradient), then the X component of force









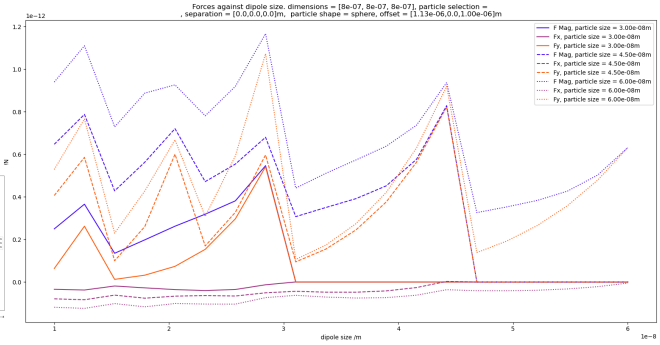


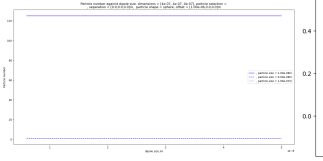
across the total shape can be reduced, and so we see essentially purely tangential forces (with some small Z force too).

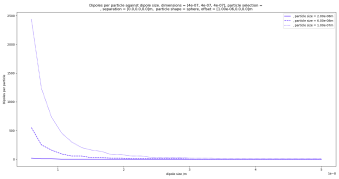
Check plots multi←2 main jumps can also be seen in the forces observed here, which are due to the number of dipoles available per particle jumping up as the particle size increased (seen in the particle number and dipoles per particle plots), hence this effect would disappear for smaller dipoles tested.

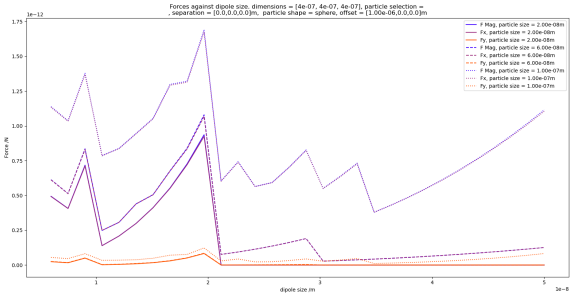
is

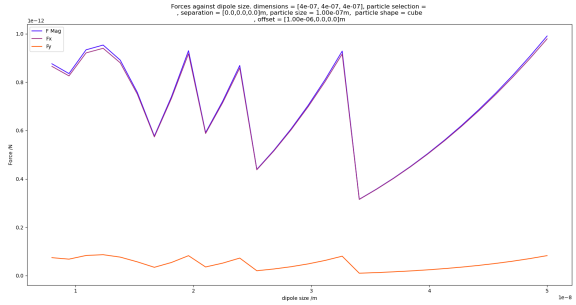
NEXT SECTION CONSIDER; -> cube modelled with Looking at both cubic and sphere primitives to model a perfect cube, we see that the cubes tend towards the true value (seen when considering the the cubic primitives for particle sizes that match the desired dimensions for integer multiples). For non-integer multiple cubes and spheres, we see that cubes tend towards the true value faster for increasing particle numbers (smaller sizes) and smaller dipole sizes, e.g. both are required for the best model, which is good and aligns with the principles of a bead and spring model.



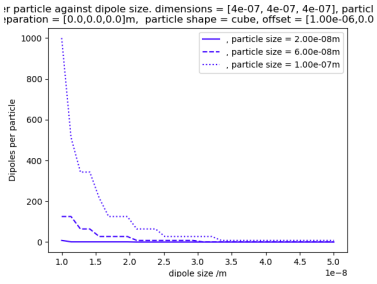


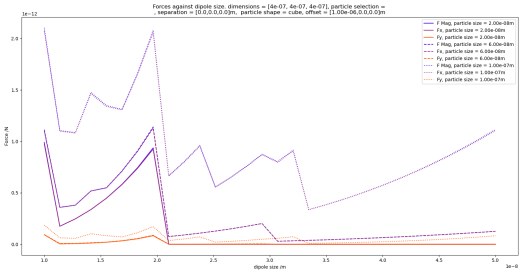






It would be good to test these for even smaller dipole sizes to see if the models do eventually tend perfectly to the case, but this would require use of the supercomputer, which currently does not support the XLSX writer module used in python to record our data, hence I would I have to write a new data storing function (e.g. to store the data in a text file, then convert to an XLSX when visualised locally), which is not the biggest priority currently.

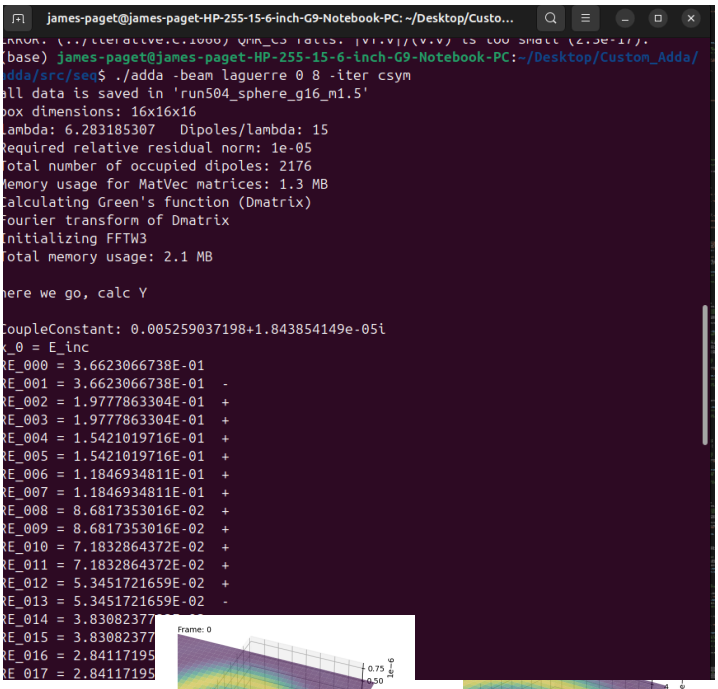


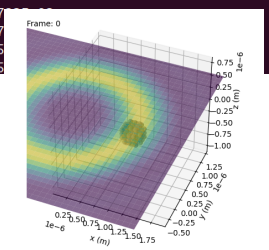


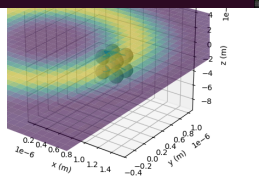
6/2/25

I have spent the morning attempting to fix the previous laptop with no success. Because of this I have had to switch out my old laptop for a new one, which has required me to set up all the python environments and gcc versions again. I have only lost minimal work due to our work being constantly saved over Github (to share with my partner), and the programs (SHDDA and ADDA) are now running on the new computer. The quicker processing sketches I had made (e.g. to test the different particle generation forms) have been lost as they were not backed up, however this is fairly inconsequential and could have been far worse.

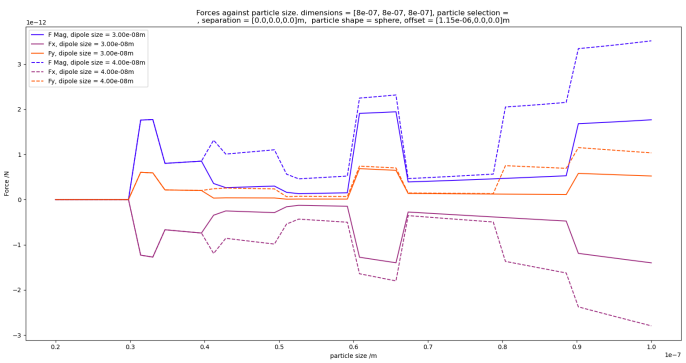
Picking up where I left off, I would be interested in performing more area based checks for dipole forces, however I am unsure as to how useful these plots were, as they did show error generated from a spherical model increasing as more spheres were added without increasing dipole sizes (which makes sense for the regions considered, where for big particles they would have checked areas of mostly filled space, hence more particles almost purely increases these gaps). Therefore I am interested in trying the current method for rods, but on spheres instead (modelling spheres with smaller sphere/cube primitives) so I can more directly look at the resultant forces, where in a Laguerre beam we know these should equate to orbital motion (for ~200e-9m radius sphere mesh), which would also give interesting results as to whether the divided system follows the ***purely optical*** behaviour of the larger (orbiting is purely optical). Before we tested if a series of these form a torus, but never if the unit sphere itself could be correctly modelled similarly, hence I think this would be a good test to show optical effects are refined alongside the model.

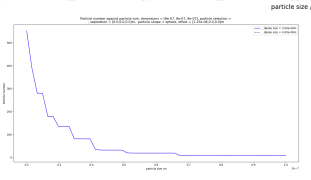


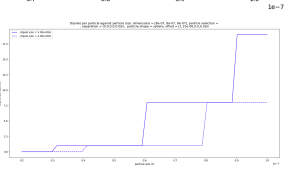




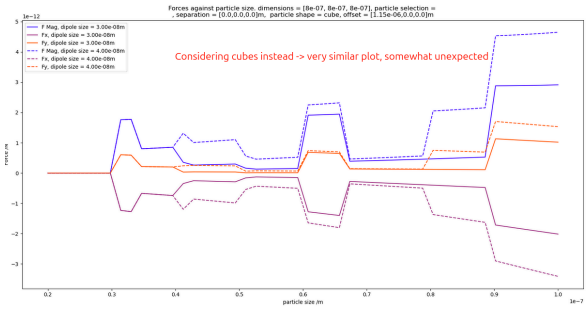
When this is considered, using the same placed regime ideas as for the rod, we can see that (1) the beads are feeling a next negative X force and net positive Y force in general, which does correspond to counter-clockwise rotation, as observed here previously. However, the X magnitude is larger than the Y here which

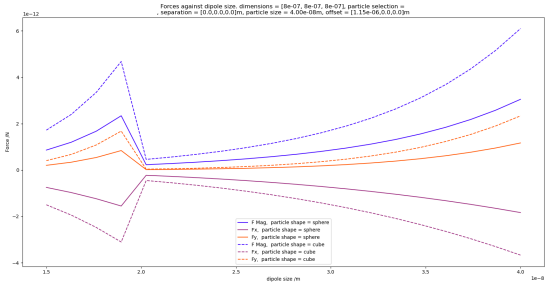






is opposite to the expected for this placement (on [DimX/2.0, 0.0, 0.0], hence expect largely Y motion here), but this being said the 30->40 nm dipole transition seems to be working to correct this (large X change, small Y change for any given point, tending towards smaller X for smaller dipoles, as expected). Here, the X force corresponds to a radial force, and the Y to a tangential force. It can also be seen that (2) that spikes in force occur where spikes in dipoles per particle occur. A direct tangential and radial force plot (force of an arbitrarily placed particle) would be good here to further verify how these X and Y forces are related to each other, and so give a clear comparison as to whether the refined system is working towards the orbital motion hoped for.





To probe the forces for the varying particle size graph a bit force, I will also try measuring the forces across the dipoles along the centre (along X axis, so we can continue labelling the X force as the radial part and Y as the tangential part). I also want to move the particles up by 1 micrometre so they don't have a large Z force and will be at a stable height to just orbit.

Key Ideas;

. Rigid particle of R=200 nm => orbital motion,

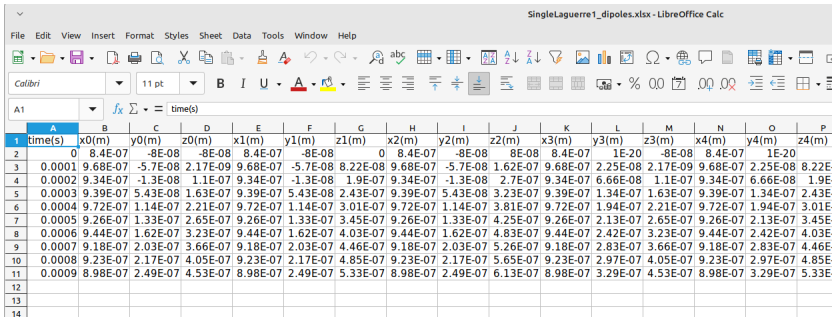
. "" "" of R=100 nm => rotational motion OR not coupled at all

. Therefore will a divided/refined particle of overall radius 200 nm => orbital motion? Or will the individually smaller particles just behave as smaller particles? Will optical binding effects cause the optical forces of the system to behave as its overall counterpart? Will the system get better or worse at this as refined?

5/2/25

Another approach to try to measure how the forces change as the model is refined is to sum the forces on dipoles within a certain region, as opposed to over a total particle/set of particles, so we can view how the region's force is affected as the model is refined.

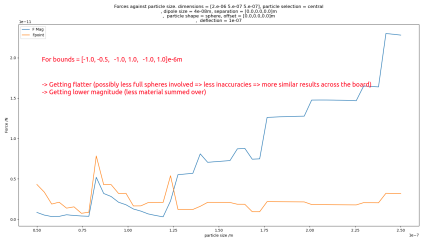
In order to measure this, the forces on each dipole needs to be output instead of the entire particle's forces summed, which requires the ***Dipoles.cpp*** and its wrapper to be extended to include more returns, as well as several other locations where the list now needs to be initialised and setup in the [frame, particle, component] format to then be read by the excel writing function (which needs to be adjusted to deal with just dipole positions and forces).

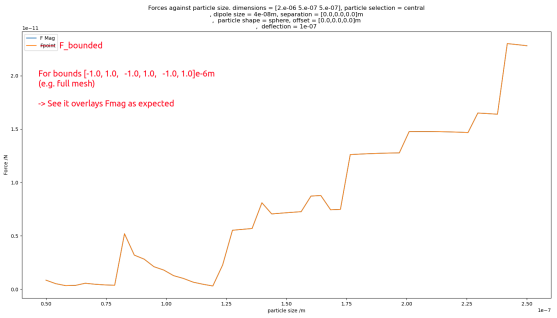

SingleLaguerre1_dipoles.xlsx - LibreOffice Calc

File Edit View Insert Format Styles Sheet Data Tools Window Help

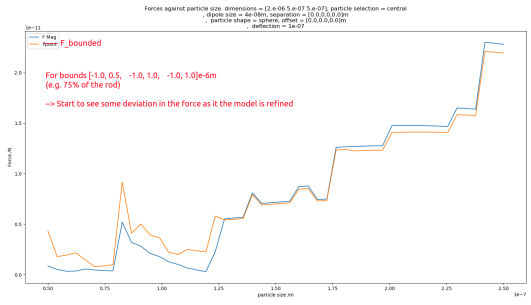
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	time(s)	x0(m)	y0(m)	z0(m)	x1(m)	y1(m)	z1(m)	x2(m)	y2(m)	z2(m)	x3(m)	y3(m)	z3(m)	x4(m)	y4(m)	z4(m)
2	0	8.4E-07	-8E-08	-8E-08	8.4E-07	-8E-08	0	8.4E-07	-8E-08	8E-08	8.4E-07	1E-20	-8E-08	8.4E-07	1E-20	
3	0.0001	9.68E-07	-5.7E-08	2.17E-09	9.68E-07	-5.7E-08	8.22E-08	9.68E-07	-5.7E-08	1.62E-07	9.68E-07	2.25E-08	2.17E-09	9.68E-07	2.25E-08	8.22E
4	0.0002	9.34E-07	-1.3E-08	1.1E-07	9.34E-07	-1.3E-08	1.9E-07	9.34E-07	-1.3E-08	2.7E-07	9.34E-07	6.66E-08	1.1E-07	9.34E-07	6.66E-08	1.9E
5	0.0003	9.39E-07	5.43E-08	1.63E-07	9.39E-07	5.43E-08	2.43E-07	9.39E-07	5.43E-08	3.23E-07	9.39E-07	1.34E-07	1.63E-07	9.39E-07	1.34E-07	2.43E
6	0.0004	9.72E-07	1.14E-07	2.21E-07	9.72E-07	1.14E-07	3.01E-07	9.72E-07	1.14E-07	3.81E-07	9.72E-07	1.94E-07	2.21E-07	9.72E-07	1.94E-07	3.01E
7	0.0005	9.26E-07	1.33E-07	2.65E-07	9.26E-07	1.33E-07	3.45E-07	9.26E-07	1.33E-07	4.25E-07	9.26E-07	2.13E-07	2.65E-07	9.26E-07	2.13E-07	3.45E
8	0.0006	9.44E-07	1.62E-07	3.23E-07	9.44E-07	1.62E-07	4.03E-07	9.44E-07	1.62E-07	4.83E-07	9.44E-07	2.42E-07	3.23E-07	9.44E-07	2.42E-07	4.03E
9	0.0007	9.18E-07	2.03E-07	3.66E-07	9.18E-07	2.03E-07	4.46E-07	9.18E-07	2.03E-07	5.26E-07	9.18E-07	2.83E-07	3.66E-07	9.18E-07	2.83E-07	4.46E
10	0.0008	9.23E-07	2.17E-07	4.05E-07	9.23E-07	2.17E-07	4.85E-07	9.23E-07	2.17E-07	5.65E-07	9.23E-07	2.97E-07	4.05E-07	9.23E-07	2.97E-07	4.85E
11	0.0009	8.98E-07	2.49E-07	4.53E-07	8.98E-07	2.49E-07	5.33E-07	8.98E-07	2.49E-07	6.13E-07	8.98E-07	3.29E-07	4.53E-07	8.98E-07	3.29E-07	5.33E
12																
13																
14																

With these modifications done, I can now read this file (still using my original ***pull_file_data() function***) to check which dipoles are within the region of interest and then plot their summed forces, which should not be difficult to do.

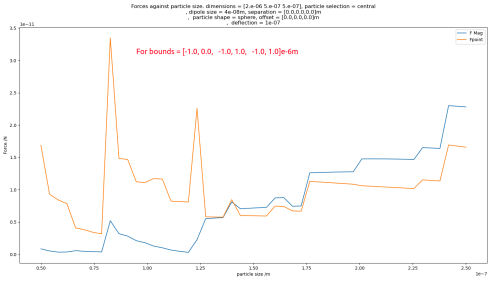




After performing this I get the following plots, where I initially do a test plot to ensure the force when bounding the entire rod is the default ***Fmag*** value, which it is (overlapping each other). Varying the area I view the forces over gives a change in these forces which tends towards a more 'perfect' answer (flat as model is refined, e.g. model is not losing accuracy from gaps introduced by spheres).



After these tests, when running a simulation for a cube and spheres, my laptop had a serious error (possibly from overheating or just old hardware) which has caused the HDD to become unrecognised, and so I cannot access any of my work and the laptop cannot be booted. Luckily our

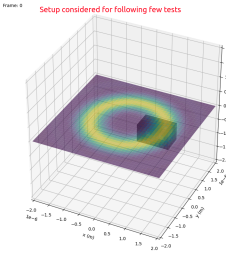


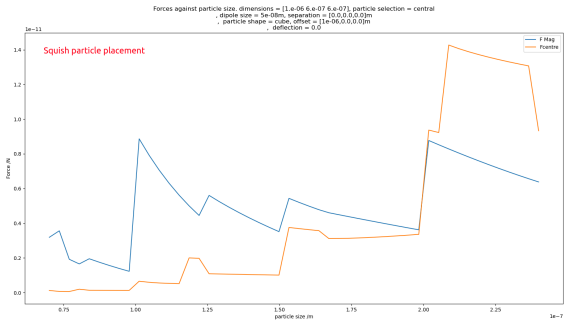
procedures with GitHub mean the work is mostly backed up (except for the most recent work on setting up the bounded area plots as I did not commit this before the computer failed), so the work should not be set back too much, however I will need to setup everything required for the software to run again which could take a couple days to fully fix.

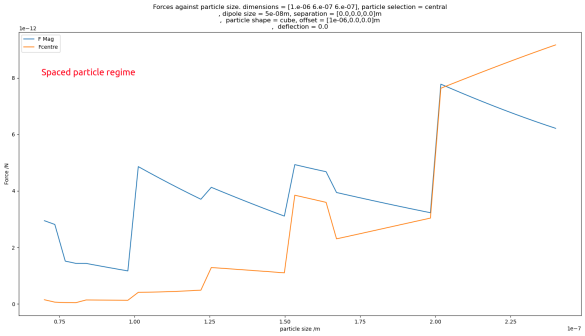
Note; The force on the deflected particle found yesterday should be redone as it may not have been measuring from the deflected part => may have gotten the wrong particle to measure at (one of the lower ones as opposed the the bit at the deflected part)

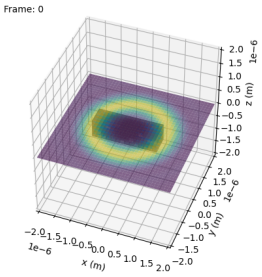
4/2/25

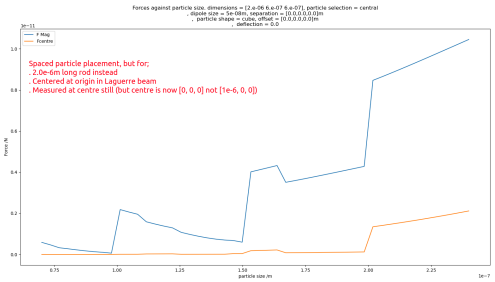
Revisiting the observations of yesterday and re-running some of our initial experiments, it actually looks like the slightly difference in particle placement method was not the problem (which I suspected was the case), and it was actually a difference in (1) the beam, location and dimensions of the rod we were measuring, (2) the exact location on the rod we were measuring and (3) the force we were considering (central particle vs 0th particle vs total mesh force magnitudes). This does still leave some confusion as to why the models did not match in our testing, however it is possible we just missed one of these factors. The models now agree, but the results I had before do not appear to be replicated, which makes me believe the previous model was using another different arrangement for particles (which may have possibly been incorrect, however when viewing its particle placement it appeared fine, however it is possible there was overlap/extra spacing).

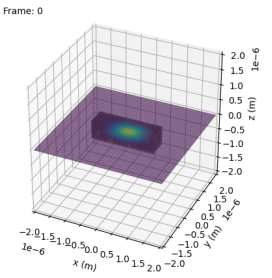


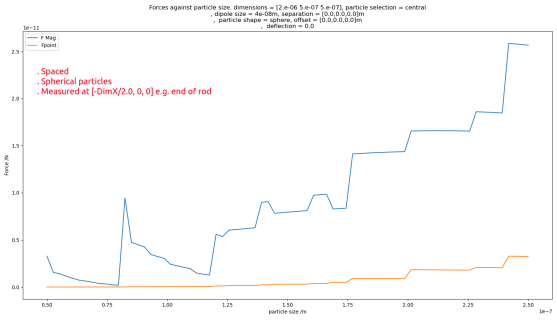




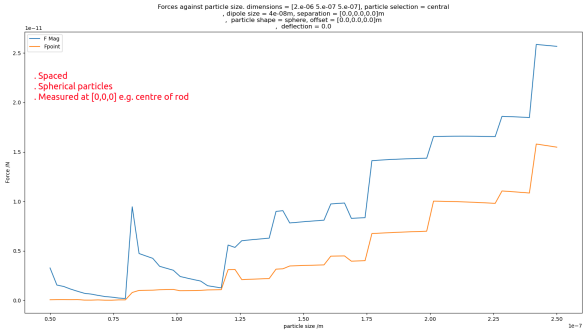








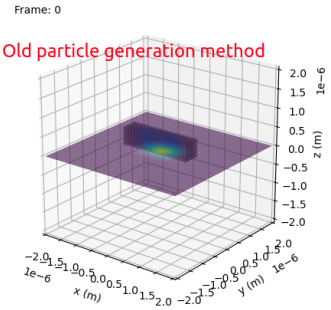
When copying the previous function for generating the YAML, I do get the previous data, which means that the changes here are purely from the generation array, which means this method is actually distinct to the ‘squish’ and ‘spaced’ methods.

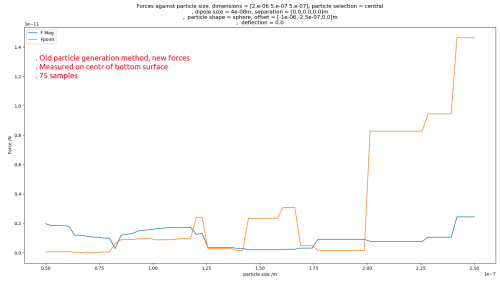


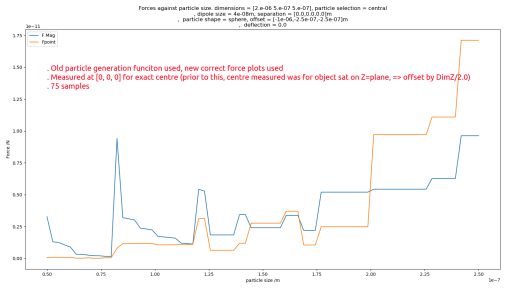
Plotting and comparing the new and old generation methods I can see that the old method appears to add some extra padding to

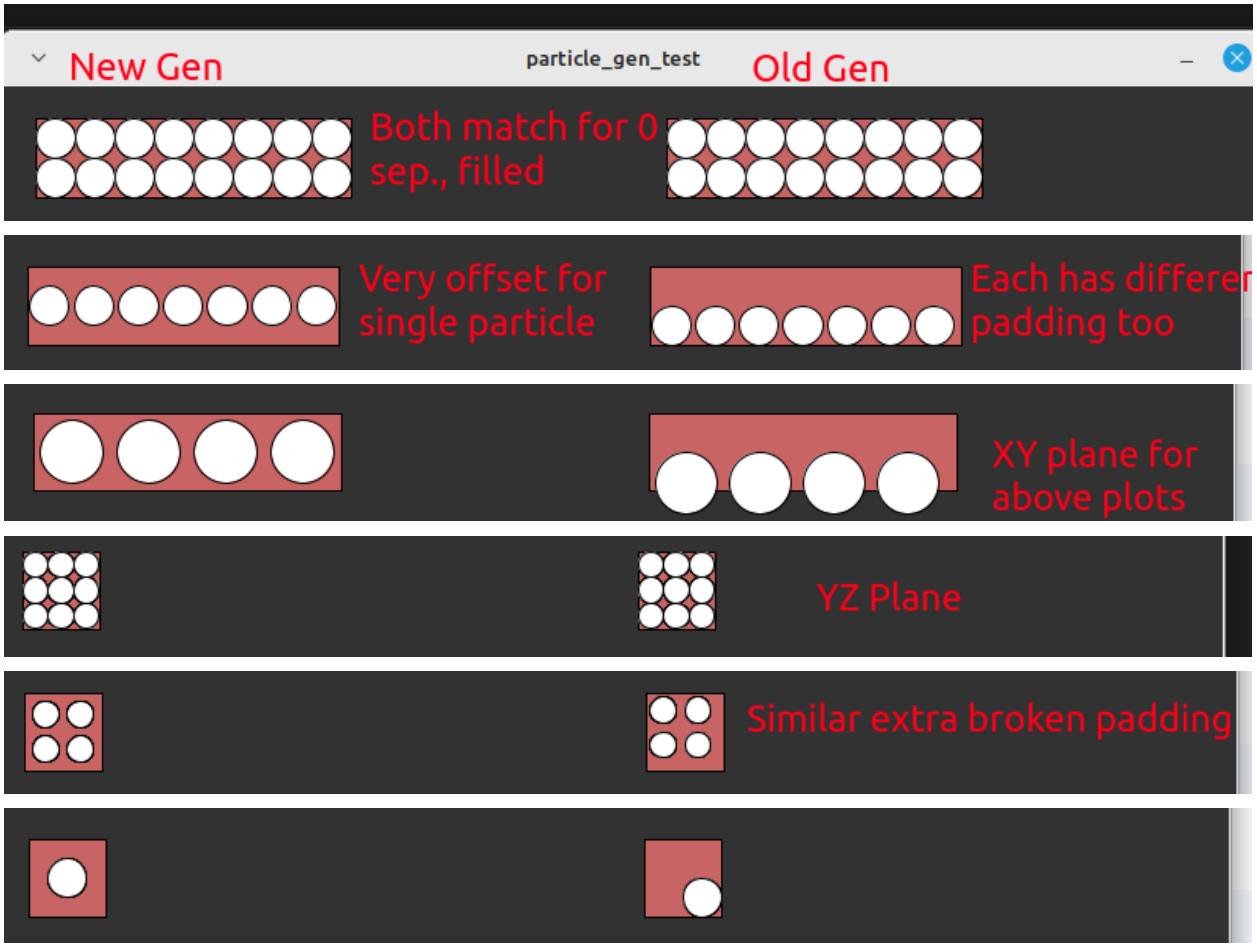
one side over the other, and some additional offset for 1 particle cases (which occur at the bumps observed before, which is likely the cause for this sudden zero force seen).

With this observation I am happy to leave this older approach and focus now on the better implemented behaviours, which also show a similar trend towards zero force as the particles become infinitesimally small, except now without a glaring discontinuity, supporting the usage of bead-and-spring approaches.





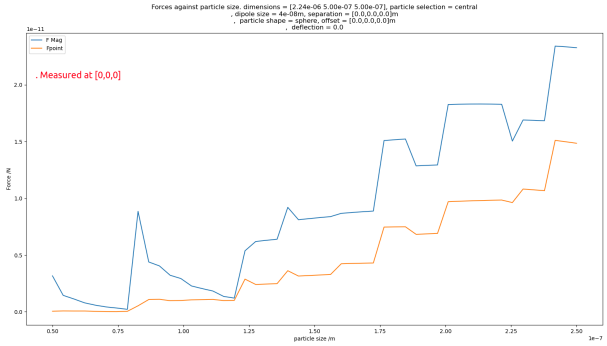


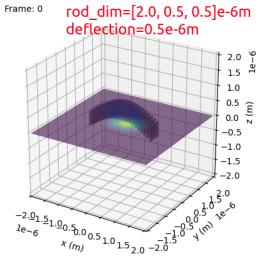


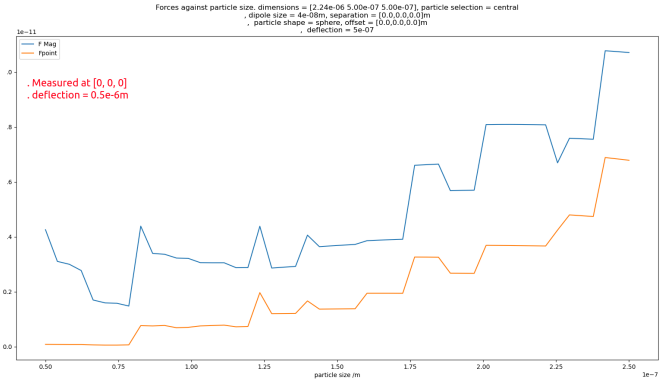
When considering measurements at different sections of the material we saw that the magnitude of the force at the particle in question dropped for all particle sizes. I could consider here;

(1) The rate at which this drops for each segment
(2) How it drops for different dipole sizes considered
(3) How the force varies as the rod is deflected -> plot a continuous stream of forces as it is moved

I also need to check other situations to ensure continuity is preserved for a split system, e.g. valid for bead-and-spring models in high refinement.

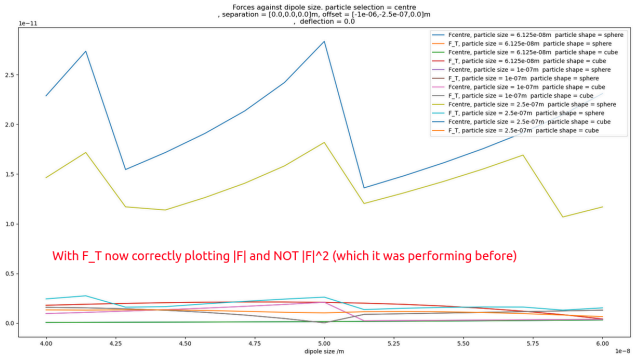




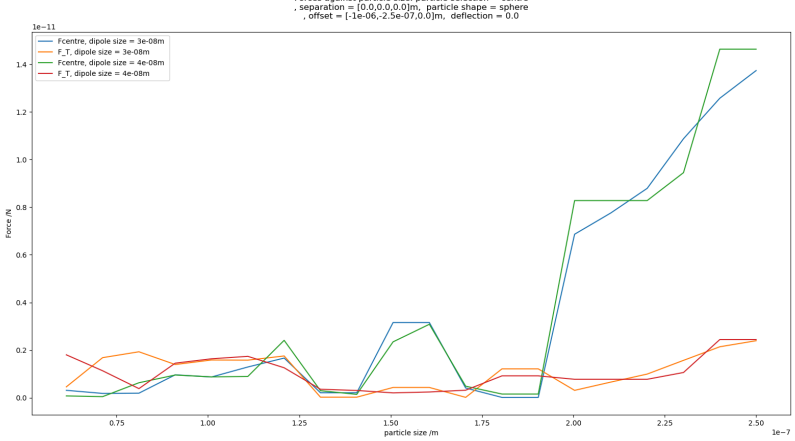


3/2/25

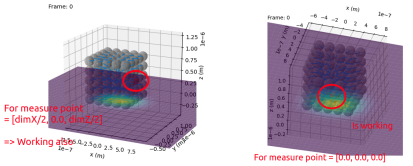
I found a bug with the program where it initially reads some data from the xlsx file made after a simulation run, however I wanted to read the number of particles here from the YAML instead of the xlsx file, but through some reordering this was and should have only caused problems for the first data point generated. I also realised the total force across the mesh was actually plotting ***|F|^2, not |F|***, which could explain why some of values where so small

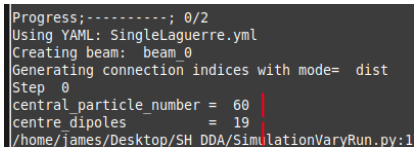


I now want to continue fixing the issue mentioned yesterday and analysing the graphs found (specifically the zero force drop seen on the particle length plot yesterday). I also need to replot that graph in light of the F_T correction.

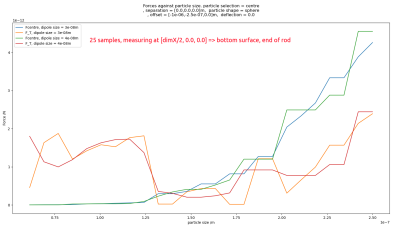


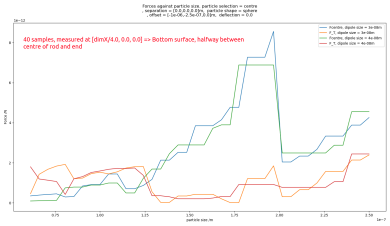
Doing this we see a substantially larger F_T, and we still see the zeros. To further investigate the zeros, I want to try measuring at different points (not just the centre) and see if this region shifts. If so, that could imply it is a product of a superposition of E fields at this specific point. Testing the altered force point selector (arbitrary point) this works well, so I can reproduce this graph for several points. Considering the forces at the end of the rod instead (slightly more samples used too 25vs20) these zeros definitely have changed, and appear to actually disappeared altogether.

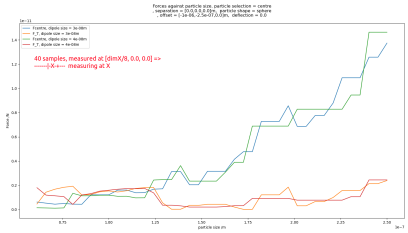


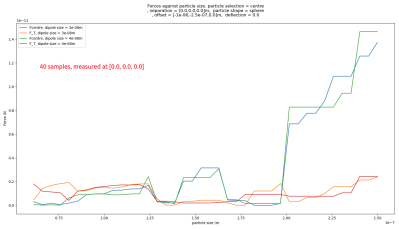


When comparing these plots to the previous method used to generate cuboids, me and my partner saw that they had quite different behaviours. The process of fixing this was quite time consuming as I initially believed one of the force calculations was incorrect due

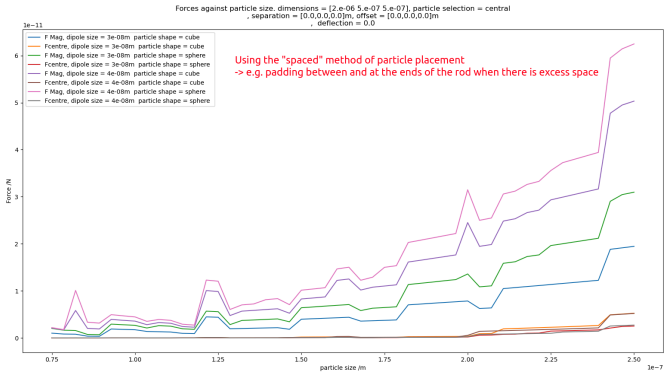








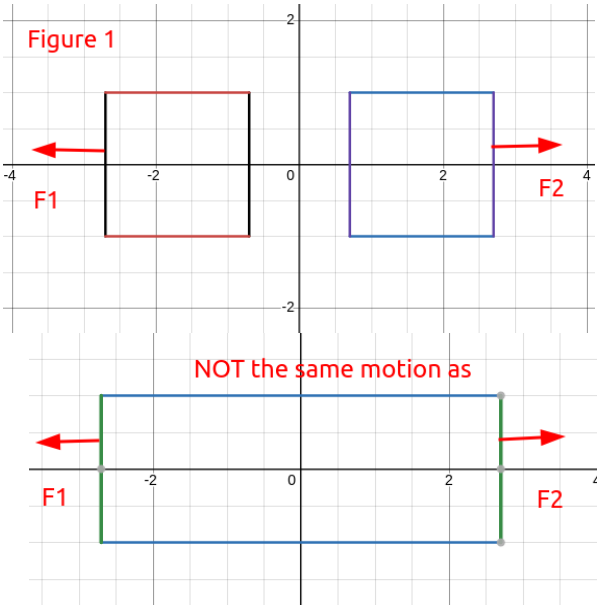
to the fact that, visually, the two scenarios tested looked identical (when viewing the first few and last few meshes) with different results occurring, which led me to believe there must have been a more impactful parameter causing this change (like the force calculator, which has several steps of data collection, storing, retrieving and filter before it is plotted, hence my concern with this part of the process). After this it was found that the difference in the particles was either (1) generation was close together, where additional space that could not be filled was ignored and the remaining particles centered side-by-side at the centre of the origin specified. The other method, which was used for the figures above, was (2) to space particles evenly (with space at the sides of the rod too) using the left over space (prioritising separation values given, e.g. left over after this was used for padding). This additional separation resulted in an interesting spectrum of highly ‘spikey’ behaviour (for (1)) to a more continuous behaviour (for (2)), most likely due to the padding reducing the force experienced by each dipole before the split/jump to a new number of dipoles present occurs, softening this boundary. Both cases are worth investigating so I added a switch between the two in my arch_prism generating function.



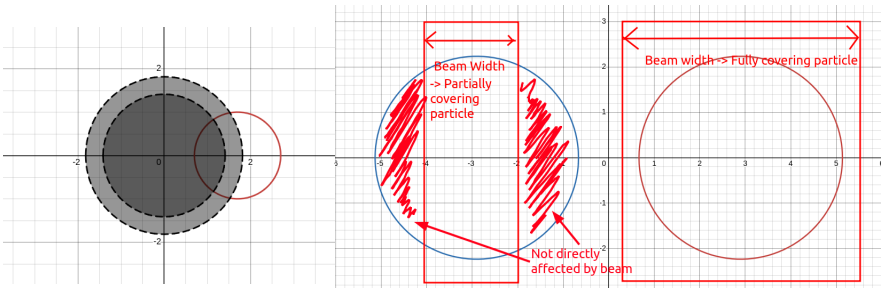
Going back to the series of data points investigated earlier for the varying particle length, we can see that this graph is significantly affected by where the measurement occurs as you would expect when contributions are largely given by scattering from nearby dipoles, hence lower forces + some distribution change at the edges.

2/2/25

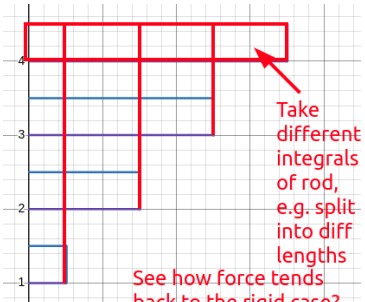
The main goal currently is to assess whether a DDA simulation can be applied in bead and spring models. The Laguerre example we explored showed that a large torus particle was well approximated by a system of particles separated but brought to a continuous limit. This property is not generally true however, as by splitting up the particle and evolving each section separately you would expect in many cases the motion to be different e.g figure 1, however it worked in this case as each particle was under rotationally symmetric forces in a rotationally symmetric system. For the rod case we have been considering this constraint is not imposed (e.g. each segment of the rod is experiencing non-symmetric forces from the field present on it, as the left and right halves feel left and right force respectively from the Laguerre beam edge, hence already want to travel differently immediately). Therefore by splitting the particle into segments we would NOT expect, say, each segment to travel as the complete rod would (they would need some way to transfer their energy and come to a resultant force for that to occur, e.g. a system connected with springs). Instead we want to see some continuity in the forces experienced at these sub particles.



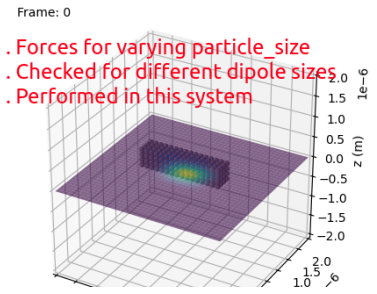
This discussion also makes me believe that going back to look at spheres trapping in Laguerre and gaussian beams may also be better as we know the behaviour for both flexible and rigid particles (both want to trap, as flexible bubbles do in ***"Trapping and manipulation of microscopic bubbles with a scanning optical tweezer"***, and as previously seen when observing ***calcite crystals fragments***). With this, we could then verify that the tangential force is converging to the same value as the system becomes more refined (considering all particle forces in the refinement mesh summed, and for individual particles within that mesh), and see if this is affected by varying the beam size the particle is trapped in (where trapping still occurs, but not all particles are directly within the beam, e.g. figure 2).



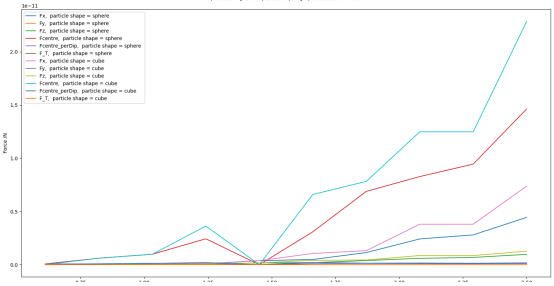
Another idea related to this is I could consider the effect of how the system is split on the forces retrieved. We have already argued

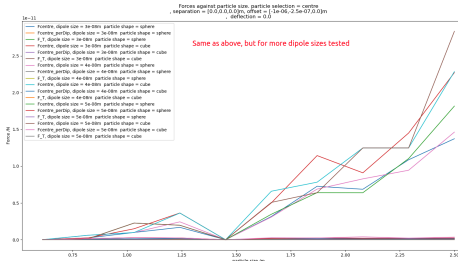


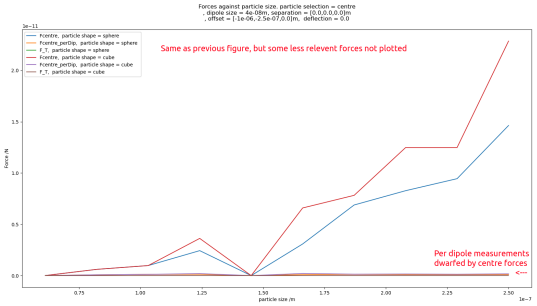
that the splitting of the system does give a fundamental difference in the force you would expect it to experience, but you would expect continuity in that force as you slow varying the size of region you are considering ‘together’/one rigid object, which as you tend to the size of the fully mesh would give you the rigid body motion. Considering how the force changes here could show us that perhaps at certain size proportions (relative to the full mesh size or perhaps to the dipole size) we see segments behave like the larger rigid body more so than at other points (e.g. some optimal length for segments?).

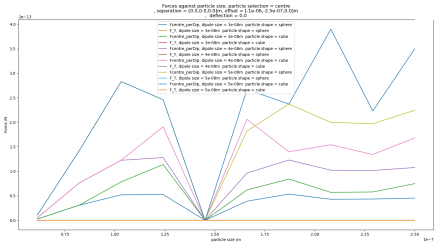


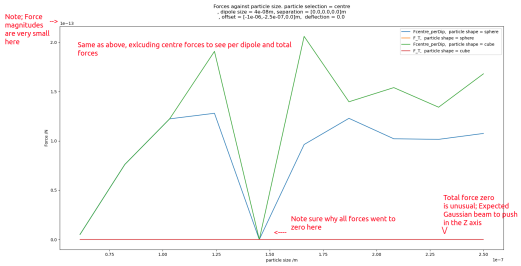
Performing some more tests on the rod again, I performed a force vs particle length plot, for a constant dimension rod, tested over several particle primitives (cubes and spheres). I expected the force on the central particle to get change slightly as the particle lengths changed (as more spheres e.g. smaller particle length => better approximation => tends to value at lower particle lengths, and for cubes the approximation should vary between better and worse as the particle length lines up exactly / slightly less than the full dimensions of the rod => oscillating behaviour with a maximum value that is never exceeded). My predictions seemed to match for the spheres, however the cubes also showed the same behaviour which I didn’t expect as they should react several points where they perfectly model the rod, and so that should give an absolute max force that is the same for all perfect matches (everything else would achieve continuous values less than this). The cause of this I believe is that the dipole size would not necessarily match this perfect particle length each time, so actually other values had more force due to being able to fit more dipoles in their volume, despite being at the perfect size. It also seemed that the force per dipole was not a great measure, as the smaller dipoles already had the force experienced by then reduced due to the polarisability scaling with volume, hence scaling it again by volume would be irrelevant, however the above realisation implies that simply stacking more dipoles in a volume is actually just increasing the force, which would imply





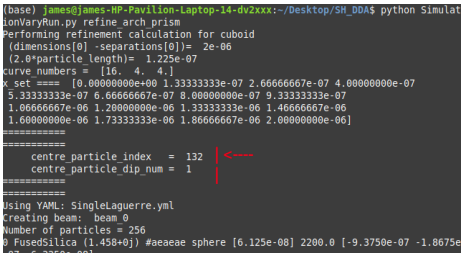


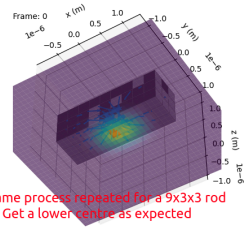


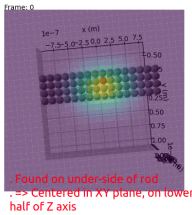


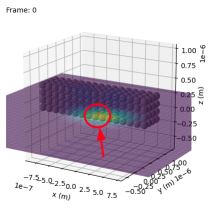
that the scaling is doing a useful role at normalising this force. Hence I will now test if simply stacking more and more of these dipoles seems to be infinitely increasing the force (which appears to be going against DDA's principles, but perhaps it takes longer than expected to converge. Note as well that the force per dipole appears to flattening, which suggests the force will continue to stack linearly for every additional dipole -> I must check that this calculator is correct and not reading the wrong values or some other issue). There are a few issues to solve so I will break each one down;

[a] Checking if the central particle is found correctly; To check this I ran the current simulation being considered with a print for parameters of interest, in a simple setup (single particle length, dipole size, etc => only 1 scenario setup, same every time), and adjusted the display function to colour this index particle so it can be visually seen where it is located. This confirmed that the point found is centered in the XY axis, but not exactly so in the Z axis (which makes sense, as it checks for points close to Z=deflection => purely the gap between the lowest dipole position and where the 'arch' of the bent rod begins). Repeating this for cube primitives of different sizes also confirms this.

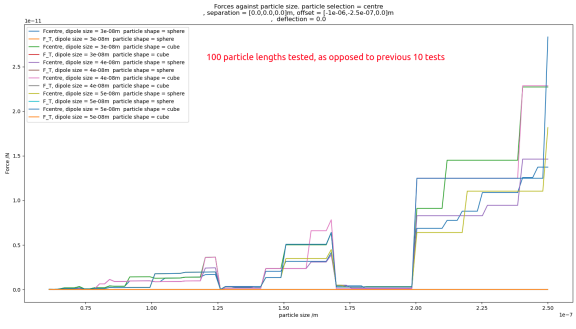






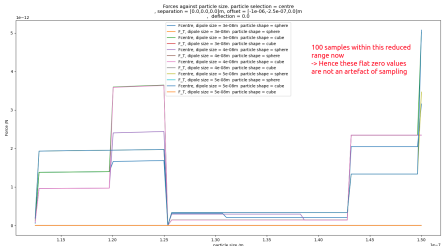


[b] Checking infinite stacking dipole force & 0 force data point above; First and foremost, the zero data point from above looks as though it could be from a lack of data points being samples (only 10 particle lengths tested above), hence it appears to have landed on a possible low value but missed it due to a large jump range. When testing for 100 particle lengths for 3 varied dipole sizes (largest test for spheres being ~2000 dipoles in system), we see that there is a region of zero force between roughly 1.25->1.4e-7m particle lengths, with another repeat of this between roughly 1.7->2.0e-7m. The zero point observed before sat at the latter half of the lower bound seen (~1.4e-7), however there was also a point in the previous data set which sat in the next region seen, but was not zero there (seen at ~1.875e-7m), hence this could be an error in the force calculation itself?

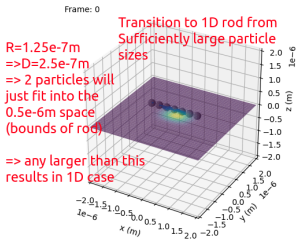


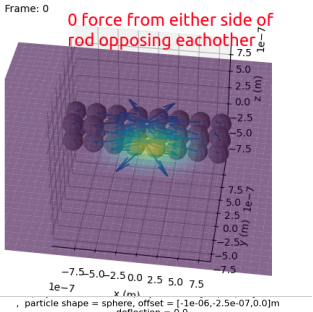
We do still also see the trend of a growing force, however the end of this data set signifies the largest values for which a particle can fit in the Y-Z axes, as the rod has dimensions [2.0e-6, 0.5e-6, 0.5e-6], hence a radius (or half-width) of 0.25e-6 signifies the largest particle able to be generated whilst still maintaining the bounds of the rod (spheres approximate volume as dipole

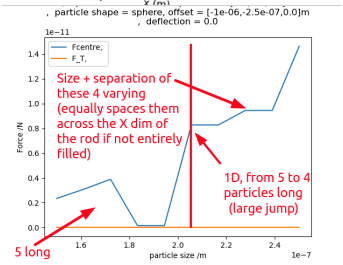
size tends to 0). Therefore further change in particle length would not be valid as it would then be modelling a different shape, and so this infinite case idea wouldn't make sense. Instead, this shows that the force for a very unrefined system has a large gradient (e.g. big change with refinement), where it then tends to some value later (~0N for each particle as they are all infinitesimal). ***Note; This poses an interesting question about the above mentioned integral over some subset of particles now, where you can then see perhaps how many particles are required to get the force in the same direction as the rigid body, or until some other force vector condition is met?***



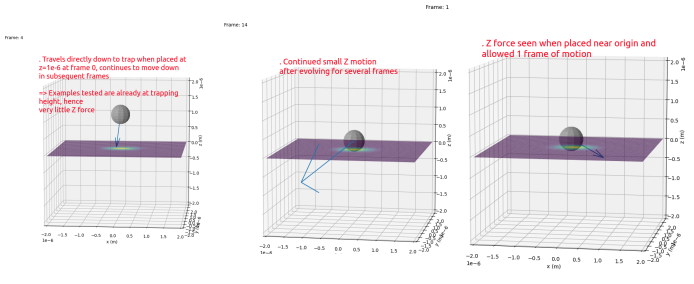
Considering the regions where these zeros occurred I did a closer plot to delve deeper into what is occurring there. For particle lengths in [0.1125e-6, 0.15e-6], we see the same picture, hence the 'zeros' (just small, not actually zero) are not a sampling issue, but could still be a calculation problem (however, the ***pull_file_data() function's output*** defaults to values of zero if not read correctly, hence it is unlikely that is the problem here as forces are just small not zero). The regions of very low force appear to be from a combination of an optimally small particle size (fewer dipoles inside) as well as the size not allowing particles to get closer (lower interaction forces) from the separation required to fill the rod's X dimension. *****It would be good to get a better understanding of this with a more rigorous explanation.***



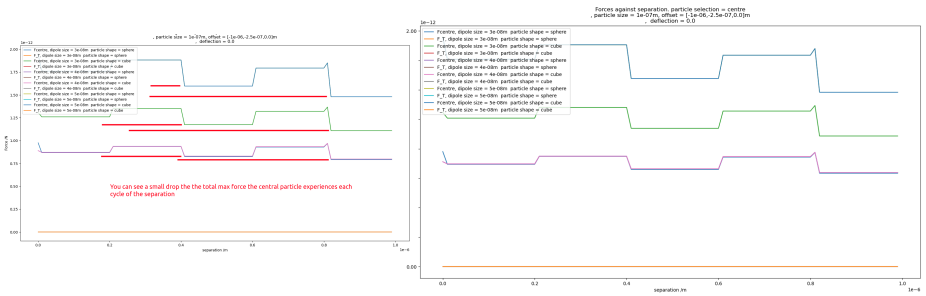




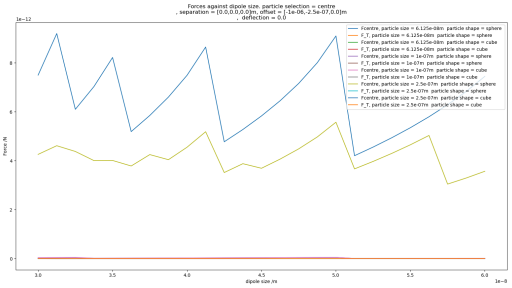
The total force is also 0 here as, when considering the force quiver plot, the forces are all in the XY plane and can be seen to oppose each other on each side of the rod. It is unusual that there is no Z force, but after some quick investigating it can be seen that the particles are started at the trapping height essentially (the origin, plus or minus Z particles will negate each other), so we are getting no force here either, which is fine.



[c] Perform other plots wanted (vary sep., dip.size, deflection, etc); When considering plots such as the forces for different separations (purely X separation not Y or Z, and



between 0 and 50% of the rod's width as separation), when considering systems with given dipole sizes, we see a pretty flat graph where separation results in oscillating high and low forces (as seen in other cases), but each cycle has less magnitude (lower interaction forces most likely, small in magnitude to the Gaussian the central particle is sat on top of).



A plot is also done again for dipole size varying, which was performed before but using a different system. The same results are produced again here as hoped.

[d] Add a generation tester that preserves total volume of dipoles, not of shape => spheres modelling rod with matching volume instead -> does this match the cubic cases now;

** This could imply that bead and spring models should have dipole volumes NOT shape volumes conserved for more accuracy.

-> This is supported by the graph above (100 steps), where the two centre force curves meet/tend to the same value for small particle sizes (e.g. for small particles, both spheres and cubes are accurately capturing the cuboid rod shape, which would be further improved with even smaller dipoles in addition to these small particles).

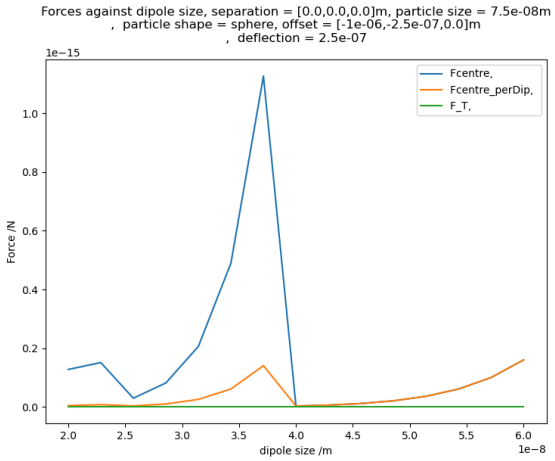
-> Can consider the packing fraction too; V_sphere = (4pi/3)r**3, V_cube=(2r)**3 => (V_sphere / V_cube) = pi/6 =0.52 => 52% of cubic volume filled, hence a cube can fit almost double the number of dipoles a sphere can hold (therefore big force difference for larger particle sizes, where the badly approximated space left by the sphere is most apparent).

[e] Add sphere in Laguerre/Gaussian beam refiner;
[f] Setup with BC4 for further testing power;

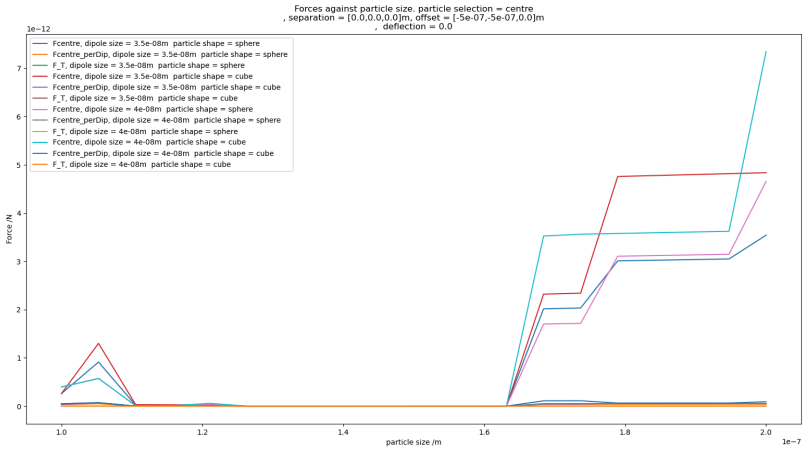
[g] Making the graph colouring more readable;
**** Add colouring options to choose what patterns of list maintain a colour / linestyle
[h] Add a dipole plotter option to see individual dipoles;

31/1/25

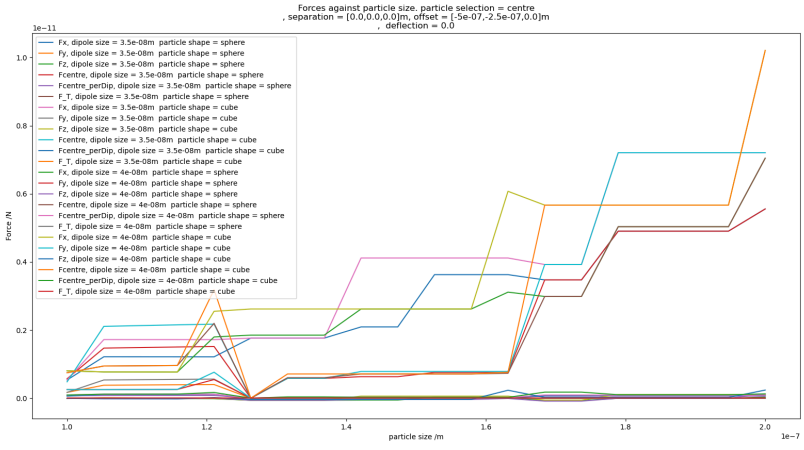
My partner has written a method to better iterate over (and then use the plotter) for a system where several parameters are varying at once (with a given chosen independent variable to plot on the X-axis). I am integrating my new force calculations for force per dipole and force across total mesh to this new.



We have been experimenting with both changing the dipole sizes for a system of fixed particle size(/particle number, as the total rod is always fixed dimensions), as well as changing the particle size with a fixed dipole size.



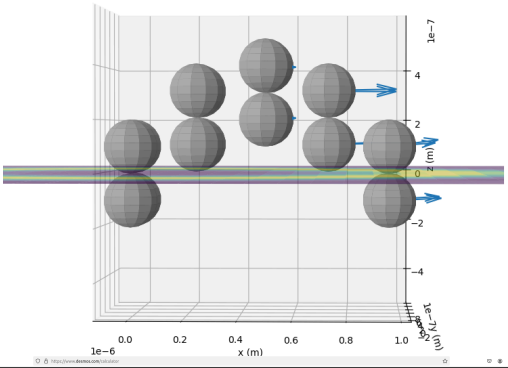
Changing the dipole size leads to the oscillating effect where the force on a given particle will increase then, drop, then repeat, as its dipoles try to fill the volume of the particle fully but can only meet this at some critical widths, which seemed to show a constant force on the object when they were reached.

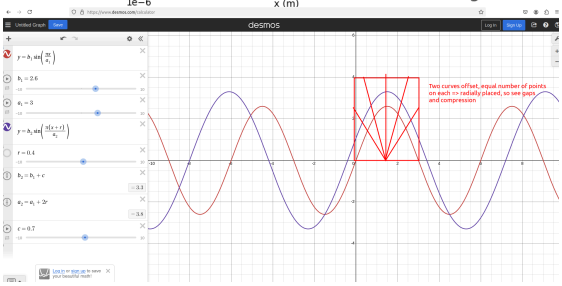


Varying the particle sizes has shown some very varied data for sphere and cube primitives, which could be due to errors in the calculation of the centre particle or some other factor, therefore I need to check this is working as expected.

30/1/25

When generating a curved lattice there are two main approaches, either (1) generate planes of unrotated particle sets (which can be done in practice either by having a single parameterised curve generating the plane at each point, or through several curves generating points at fixed lengths travelled along the curve) or by (2) generating points as you travel the curve, but with the plane of points rotated. Approach (1) essential fills the space fully with particles, whereas (2) leaves gaps and compressions (as you would see for a molecules in a bent rod), hence both could be useful for modelling a rod, as the second approach would more accurately model bending, bt on the discrete scale already taken would be including unrealistically large spacings, but equally (1) would not have a sense of more compression or extension.





The figures to the right show the first method in action at separate levels of refinement (varying particle size, trying to fill the space evenly and as much as possible with the size given) for a given deflection.

Approximation Formulas:

$P \approx \pi (a + b)$

$P \approx \pi\sqrt{2 (a^2 + b^2)}$

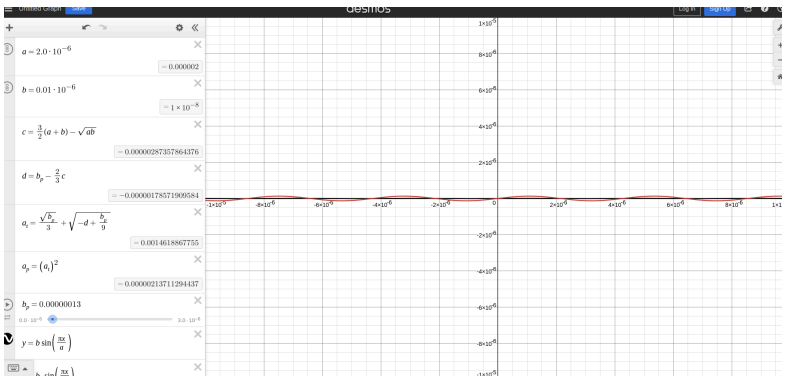
$P \approx \pi \left[\frac{3}{2}(a+b) - \sqrt{ab} \right]$ <----

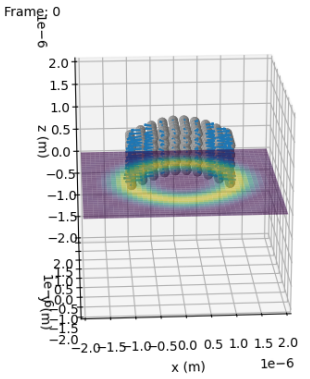
Ramanujan Formulas:

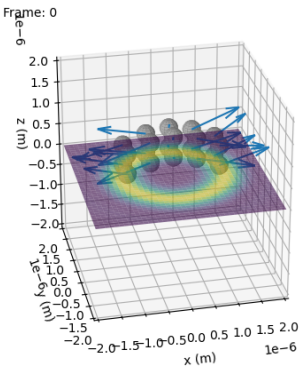
$P \approx \pi \left[3 (a+b) - \sqrt{(3a+b)(a+3b)} \right]$

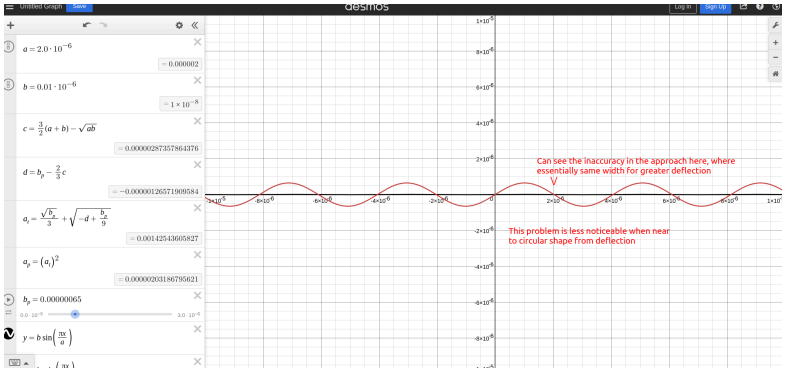
$P \approx \pi (a+b) \left(1 + \frac{3h}{10 + \sqrt{4 - 3h}} \right)$

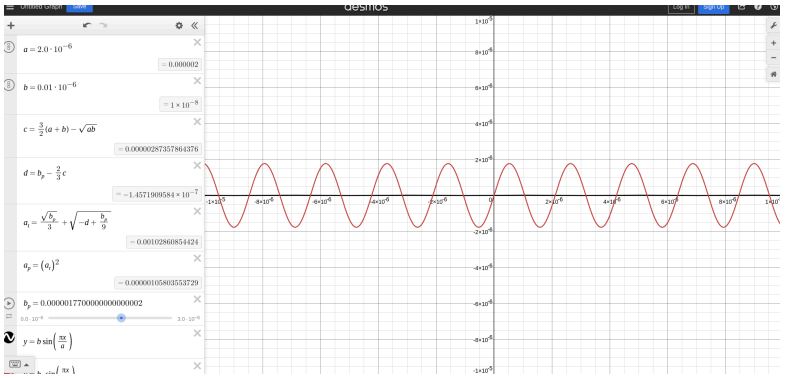
Using an ***ellipse perimeter approximation,*** https://www.cuemath.com/measurement/perimeter-of-ellipse/, I can obtain an estimate for the length of the arc of this sinusoidal section (path integral gives elliptical integral result). This will allow the arc length (perimeter/2) to be conserved as different deflection heights are considered (by reducing the width), e.g. conservation of mass, so all the bent rods are comparable.

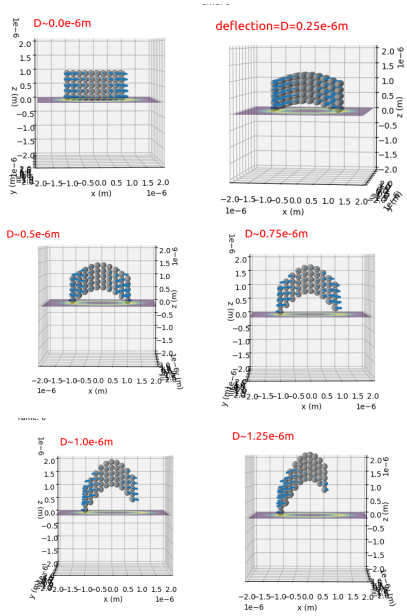




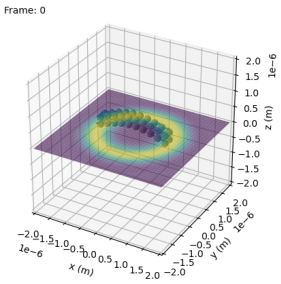


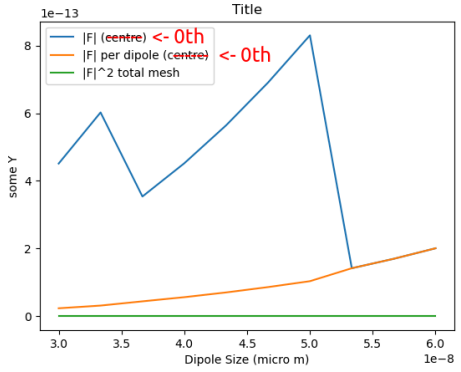


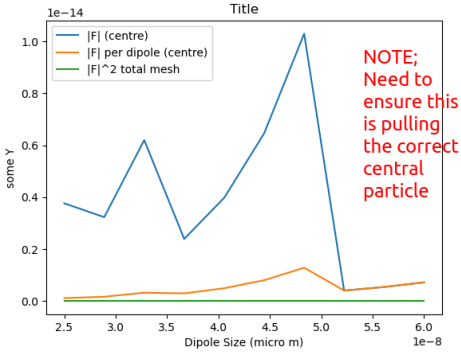




This setup for a deflected rod will let me test varying dipole sizes, particle sizes, particle separations, etc to refine the system. Now I need to decide which optical forces should be considered for the best comparison of systems. With this, I can then make those comparisons between deflected rods and see if further continuity is present. A good measure for each system may be the ***average force per dipole***, which is the force on a particle divided by the number of dipoles it has. This would prevent physically larger systems having a force magnitude advantage. This requires the number of dipoles for a primitive to be found (given by ***<name>_size() function*** for each primitive) and the force magnitude (pulled from xlsx). For now I will consider this force per dipole on the centre-most particle of the system, in order to keep the measurement fair for all systems (requiring a function to







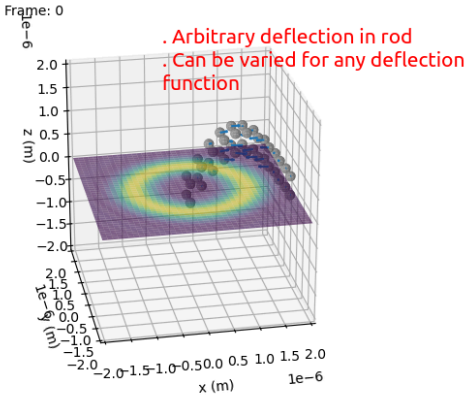
find the closest particle to a point).

```
def get_closest_particle(point, output_data):
    #
    # Looks through all particles in a given frame (from some output data), and finds the
    # index of the particles closest to the point given
    #
    # expects point as numpy array
    # output_data in the format of (x0,y0,z0, ...) for each particle, and no other readings
    #
    low_dist  = 1.0e10  # Distance of closest particle to point (easily beatable placeholder when initialised)
    low_index = 0       # Index of the particle that is closest to the point (placeholder of 0 when initialised)
    for i in range(0,len(output_data[0]),3):
        pos  = np.array([ output_data[0,i+0], output_data[0,i+1], output_data[0,i+2] ])
        dist = np.linalg.norm( point-pos )
        if(dist <= low_dist):
            low_dist  = dist
            low_index = int(np.floor(i/3.0))
    return low_index
```

29/1/25

When creating a cube primitive to use, my partner noticed that the sphere primitive's current method for dipole placement assumes the number of dipoles in the diameter is odd (as it compares and floors just the radius to the dipole width, hence with a radius R=1.5d, you get a sum from -1 to 1 => 3 dipoles), which has now been fixed.

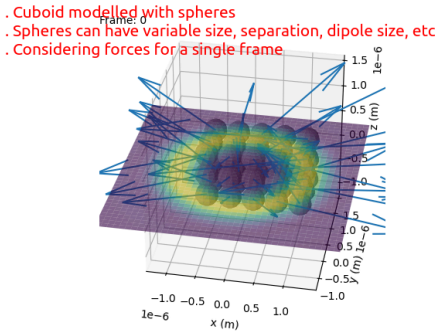
I am making a plotter to allow varied measurements more easily plotted with iverlayed figures if required. This will let me more easily see which parameters are having the most impact, and so begin refining these. I have also had to create a more easily accessible function to pull data from the ***_combined.xlsx*** file I generate (storing forces in an easier to use format) so it will be easy to vary the types of plots being generated (e.g. XY plane forces, total forces, torques, individual components of force, forces from specific subsets of particles, forces from particles at evolved frames if wanted, etc).

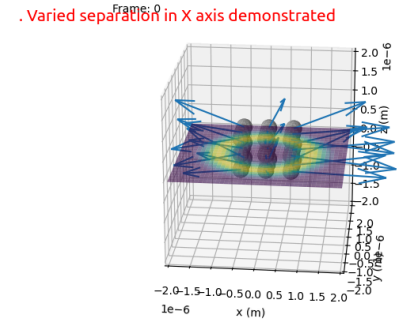


I am testing out a new function to generate meshes in a cubic-like formation (follow some parameterisation for a curve, which is replicated at regular intervals, then checked to ensure is within some distance function of the curves to give arbitrary cubic prisms) which should allow tests to be performed for shapes that are slowly varied (emulating the flexible motion you may have if the particles were joined with springs and deformed) to see whether or not the optical forces drastically change as you move between these 'quasi-static' states. If they continuously evolve between each other at varying refinements in resolution, then it would suggest that the deforming of particles modelled like this does not introduce any inconsistencies, and so is further evidence for flexibility being valid. The main influence for doing this is the lack of evidence to compare our optics results to (only comparison to each other as refinements occur), so we can only look at the continuity of the force and not the ground truth of whether it is accurate for that model. This approach does not give a ground truth, but gives further reason to trust the calculator performed, as it is closely reproduced when considered similar systems, suggesting again that there are no discontinuities, which is what we are interested in (if a certain level of refinement leads to breakages).

27/1/25

A large portion of the day was spent discussing what we would like to measure and how this may be experimentally found. The ultimate goal is to assess whether or not a rod can be modelled as a series of separate particles (both spherical and cubic/another primitive which matches the larger shape's geometry), as we would like to see a more 'ideal' case where the primitive leaves as little inaccuracy in the shape dimensions required, and a sphere case for a less realistic model, but more easily applicable to later dynamics that would want to be applied (and hopefully this matches the ideal case well). In order to do this, we will setup these particles will some constant parameter, e.g. dimensions of the total shape being modelled, the total separation across each axis of the shape, the number of particles present, the number of dipoles per particle, the size of dipoles in particles, etc, then vary these other parameters one at a time (where relevant/not conflicting with other parameters) to refine the model. At each refinement, we can consider the force on a particle particle, collection of dipoles, or the entire object. By considering the continuity of these values we will assess whether DDA has a certain 'break-down' resolution where it suddenly treats particles in the system under a different regime (this idea stems from that fact that it is acceptable to consider any size/shaped particle in DDA, however for a given particle the force on each dipole need not be correct, due to varying formulations of force density which will give different individual values, but the integral across the particle always appears to give the correct values, however at some point of 'refinement' of the system by increasing particles present you would expect to reach this single dipole type case -> which raises the questions s to whether the system can be refined through increasing number of particles as opposed to just increasing number of dipoles).





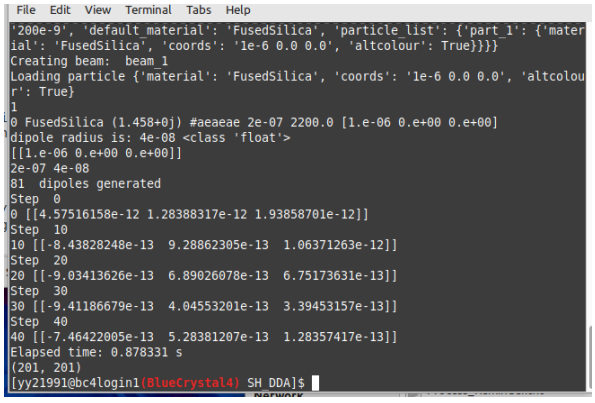
Following the vein of thought outlined above, you may expect a rod split into N cubic sections to have each section move in various directions, seemingly in opposition to the single particle (rigid) case. However, you could argue that this is not unusual as a separated particle is fundamentally different to the connected particle (which is somewhat indistinguishable in DDA, e.g. when particles are placed directly adjacent, you would not be able to say which particle each dipole belonged to directly, only through the external cutoff imposed when you select the bounding region to sum forces over for the particle's individual force). Therefore you would only expect the motion to follow this more rigid case with a spring system (e.g. transfer of force enabled without changing total energy of the system), but we can still consider the optical force on each and assess how the refinement affects this force (bear in mind DDA is numerically exact, so improvement is expected but a change in regime is what needs to be watched, to see if this correct behaviour is suddenly 'activated' at some critical resolution).

Therefore in SH_DDA I will generate a cuboid rod with varying particle numbers, dipole sizes, separations, etc, allowed where forces can be pulled from various parts of the system too.

26/1/25

After speaking with my supervisor about our progress with flexible particles, he suggested that the best approach for the current line of thought may just be to test various spring and bending constants to try and match the deflection seen in real samples, and hence try to associate these constants with a flexural rigidity. The problem with this is that there isn't an established way to connect this bead and spring model to real flexibility, as it would be very shape dependent (e.g. how the beads are connected), so for our case we may want to try a cubic arrangement for the bead and springs to model the crystal lattice.

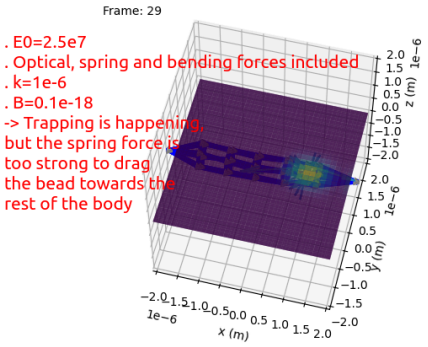
This being said, our supervisor suggested we instead focus our efforts back on the optical effects instead, as the main concern of the project is as to whether you are able to express a mesh as a bead and spring model, rather than actually computing the flexibility in this case (not to mention the uncertainty already involved in our Buckingham force calculation, giving more reason to not focus on the dynamics itself too much, but rather just the validity of the model). To act on this, we will model different systems with particles (spherical and other appropriate primitives to match the sample's shape) of increasing resolution, and observe if/where DDA breaks in its modelling (stemming from the fact that moving individual dipoles is traditionally disallowed in DDA, but moving particles is considered fine, hence you may expect a limiting case at a certain resolution). This approach will be very similar to the previous example, but now particles will also be considered of various sizes to further probe the system (not allowed before as trapping only occurred for specific sizes). The plan is to focus on the continuity of the forces experienced by each particle, and to focus on the fact that the integral of force densities across a shape is valid despite the possibility of having different formulations for that density (hence different individual forces at each dipole).

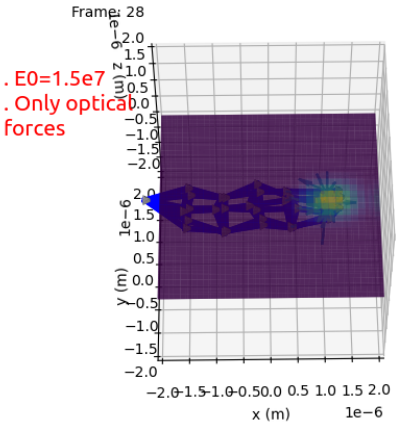


In order to test at the very high resolutions of interest, it is very likely the university supercomputer ***Blue Crystal 4*** will need to be used. This appears to still be working on an account I have set up (originally for another module). We have also been provided with additional codes to use other partitions of the supercomputer.

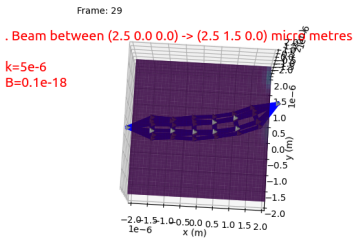
24/1/25

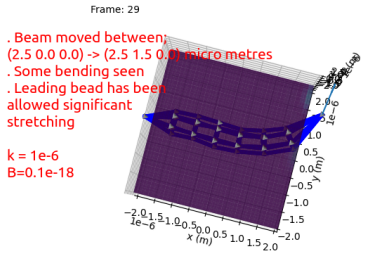
When testing the Gaussian beam on the system using beads at either end, with the rod particles being ***Red Blood Cell materials, n=1.41,*** and the beads on the ends being ***FusedSilica glass, n~1.45,*** then we see the beam pull particles when travelling from X=2.5e-6m to X=1.0e-6m across 30 frames, but it cannot overcome spring forces when ***k=1e-6, B=0.1e-18 for 6 radial and 6 angular particles used with a rod of length 3e-6m***. We expect to see resistance to follow the motion if (1) the spring constant is too high or (2) the beam is too weak (F_elastic > F_Egradient), so must change either of the these factors, but must be careful to not tune them too high in order to avoid clumping particles or exploding.



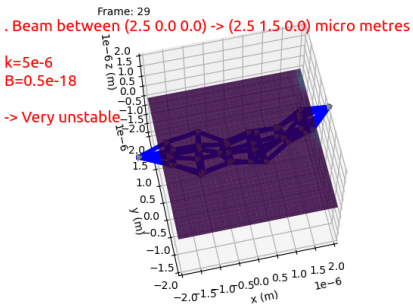


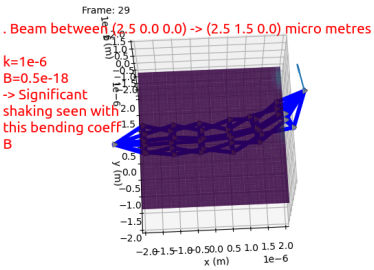
If we try instead moving the beam in a line perpendicular to the axis the rod is formed in (X, => move Y), then we observe move bending with less resistance. Note that here the other bead is not fixed, as in the experiments described in the prior paper.



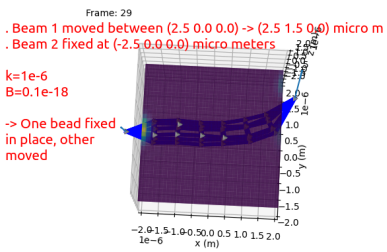


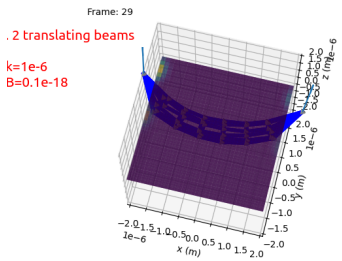
By testing the spring and bending coefficients, k and B resp., we see the model breaking with high B, but giving a much cleaner transition in behaviour as k is varied (varied by 5x its value here for a relatively small change in rigidity).



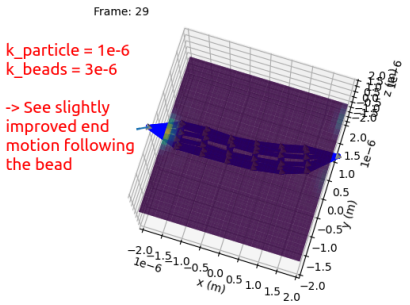


I now try fixing the other bead at a single point by trapping it with another beam. We could also consider the two ends being translated to get a slight curve,

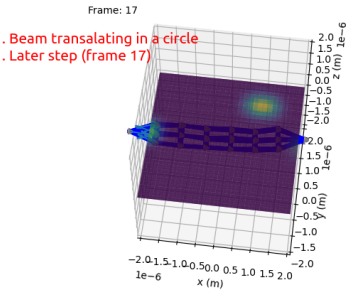


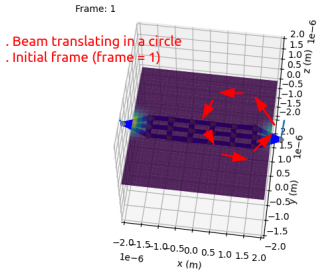


then bring them together, which will require the translation of the beam to be a function instead of simply a linear translation.



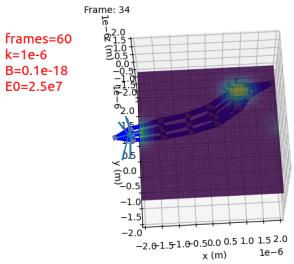
I have now generalised the stiffness and l=natural length matrix to have changes made from SimulationVary(), which now lets a non-uniform stiffness matrix be formed, where connections with beads can be a different stiffness to the default between each particle. This lets the beads have a much stronger adhesion to the mesh, whilst leaving the mesh properties along. You could also in theory adjust the natural lengths of the mesh as well now to be non-uniform (perhaps force curvature through smaller interior natural length?).

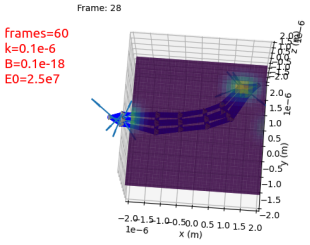


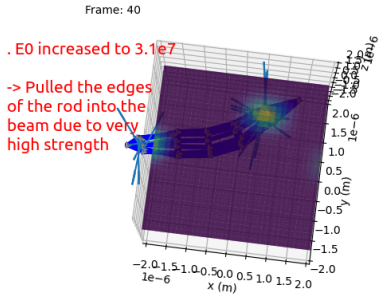


The beam translation has also been generalised to allow linear and circular translation. My hope with this is that by taking a circular route, we will give the rod an easier transition into bending, as opposed to simply compressing it and hoping for buckling to occur in the correct direction.

With the first circle test, the bead quickly lost contact with the beam due to the extreme bending angles (had ***E0=2.5e7, k=5e-6, B=0.1e-18***). By slowing down the rotation of the beam (interpolating over 60 frames instead of 30) and changing the spring constant, k, we see that slightly more bending is allowed for the smaller k, as you would hope, which is a good sign. However the other bead (meant to be fixed by the other beam) slid around quite a bit when the circularly moving bead got closer (more compression forces), hence I either need a stronger beam to keep it more stationary, or to manually fix it in place (e.g. don't calculate stepped positions in dynamics). I will try the beam approach as it is less 'forced' (and so hopefully offers more realism) than the alternative, however I must be careful that this beam is very focused on the bead alone, as interactions with







other particles may cause unintended side effects not present in the paper’s experiment.

23/1/25

It is described in the paper ***"Buckling of a Single Microtubule by Optical Trapping Forces: Direct Measurement of Microtubule Rigidity"*** that the stiffness is found from the experiment (compressive forces from two beads on a rod) by ***deflected curvature analysis***, which considers the deflection distance (at the centre of the rod) caused by buckling from the compression.

Hence for our simulation we could (1) repeat the experiment, find the stiffness using this analysis, then compare it the stiffness constant used in our simulation to is if they match, or (2) repeat the experiment using the stiffness they found for each rod, and compare the deflection of our simulation to the reality they observed (for a given compressive force).

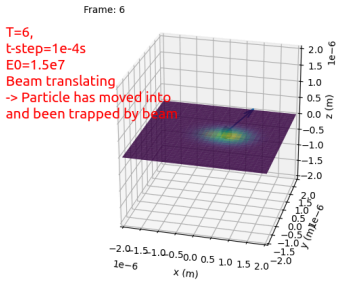
TABLE I. Comparison of Flexural Rigidity Evaluated From Critical Load and Deflected Length Measurements*

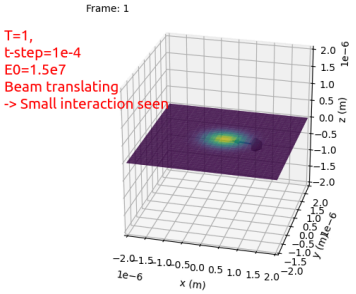
		Critical load meas.		Deflected length meas.		
Samples	Microtubule length (μm)	Critical load (pN)	Flexural rigidity ($\times 10^{-23}$ Nm^2)	Force (pN)	Deflected length (μm)	Flexural rigidity ($\times 10^{-23}$ Nm^2)
MAP-stabilized microtubule	10.5	3.0	3.4	4.9	6.0	4.3
	10.5			3.7	9.1	3.8
	19.9	1.5	6.0	3.1	9.3	9.1
	27.9	1.5	12.0	3.3	17.0	20.0
Taxol-stabilized microtubule	4.4	0.7	0.13	1.2	3.6	0.20
	11.8	0.4	0.56	0.5	9.5	0.59

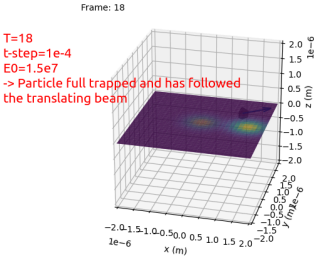
*Values are compared for MAP-stabilized and taxol-stabilized microtubules. Critical loads and deflected lengths are also listed.

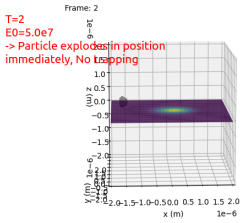
The main problem here is that we want to vary the spring and bending constants in our simulation to achieve different rigidities, but the data given (obviously) does not give a spring or bending coefficient (because it is not using a bead-spring model). Hence we would have to calculate the flexural rigidity of our shape and compare, vary the constants to give this same rigidity, then make an association between the flexural rigidity wanted for these coefficients controlling our simulation.

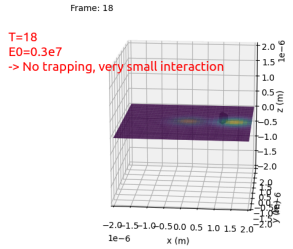
In any case, we will need to deform the particles through the beads being trapped and translated, hence I will try to perform this. When using the ***"GAUSS_CSP"*** beam, which is a complex source point Gaussian beam as used in the paper mentioned above, using an ***E0=1.5e7***, which is scaling the power of the beam, I then see trapping occur. Values <*10 this value do not appear to trap at all, and values >*10 this are pulled far too quickly by the beam and so explode. The particle used here has radius ***R=200e-9m***. When testing this with larger particles (600nm radius), trapping still occurs in the beam with E0=1.5e7. Note as well, the faint beam seen in the background of the translated beams is not another beam, but just shows where the beam initially started, and has no optical influence.



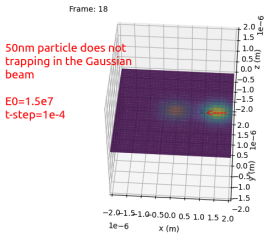


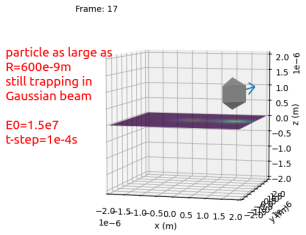


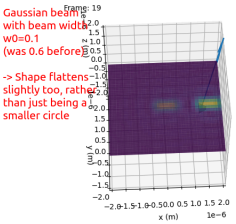


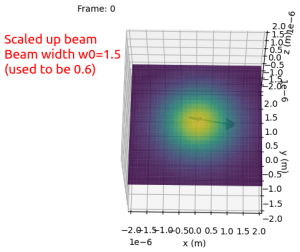


With this Gaussian beam trapping as hoped, the beam can now be placed at the bead located at the end of the rod models, and moved slowly to cause compression of the rod. I will also want to scale the beam to just be located about the particle, which can be seen in the figure on right where the beam does scale down slightly, but not with the same symmetry. Scaling up on the other hand appears to work better, hence we could consider the system as slightly bigger rather than the beam as smaller (will depend on the physical dimensions of the system, which for the paper above is larger by roughly 4-5 times, so this should be fine).



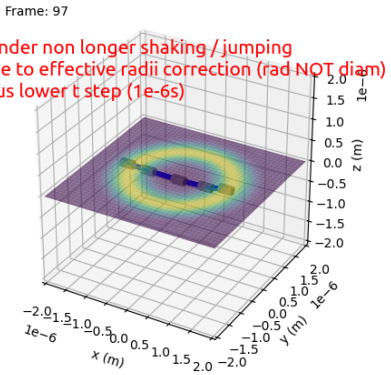






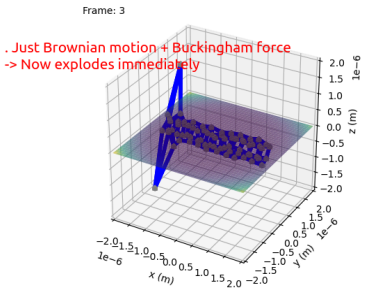
22/1/25

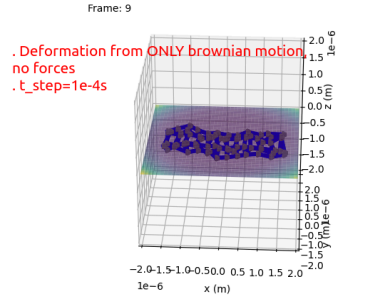
I have started converting the models made in SimulationVary to start using the new general call for the main() function inside DipolesMulti2024, so each can keep the constants it requires for stable motion separate (prior to this we would manually change the constants in the main function, which would affect each other simulation if not changed again). As I am performing this conversion I am searching for parameters which make the system stable, so after it can have beads attached to the ends of the rod and be deformed in a stable, more physical, manner.

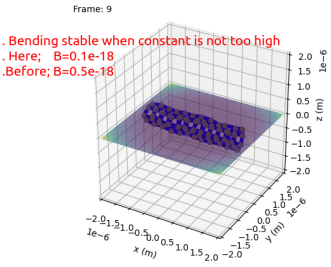


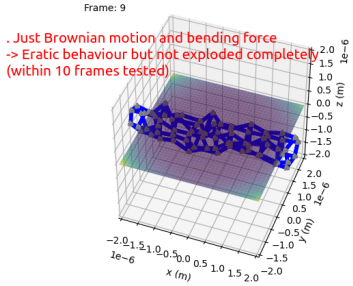
The ***effective radii*** of the cylinders was also corrected (now considers [total_length_1/2.0 + total_length_2/2.0]/2.0, before used the full total_length, which is more like a diameter for a cylinder (of the bounding sphere to the cylinder)).

When considering the hollow shell of spheres for the rod, this had a tendency to explode from the Buckingham term, and partially from the bending term (optical and spring forces were very stable for default spring value ***k=5e-6***). This gives further reason for the Buckingham force to have all its parameters rescaled again to be more gentle. When excluding the Buckingham force and including others, with the lower Bending constant we get a stable mesh. When considering beads attached with torques, we could possibly consider other more impactful Bending constants.

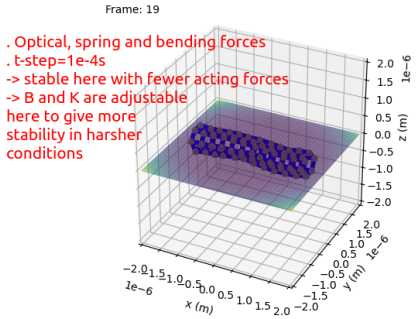


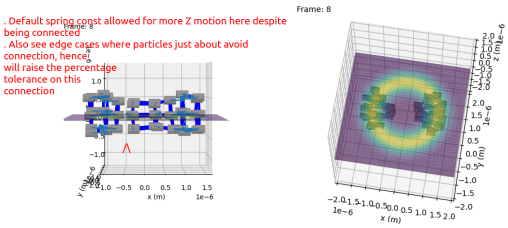




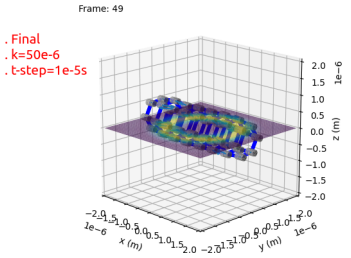


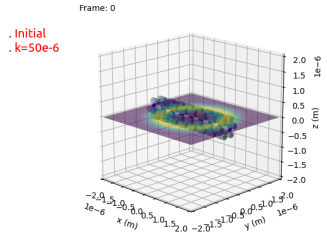
The cylinder variant of this model worked with similar results, only requiring a slightly higher spring constant (not observed in sphere case as particles sat outside the high intensity part of the beam). However, the cylinders here also had the problem that they continually inflated up to some roughly stable point (seen at ~frame 50 here), despite choosing very large spring constants. Despite this they did not explode and had only a small amount of wiggle which is good. By increasing the bending constant here the expansion occurred less, as I believe it was driven by the outer edges being pulled out, and so the inner edges loosely being pulled up to it by the bending force. A larger bending force means a better balance between the outer



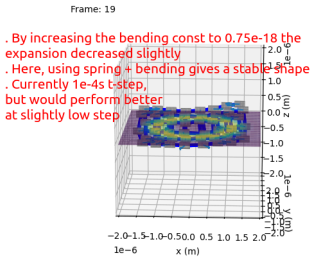


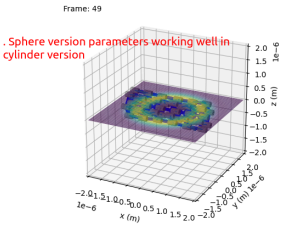
pull and inner unaffected part strikes a better balance with only a slight expansion.

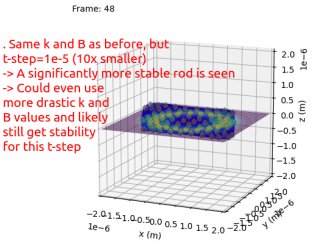


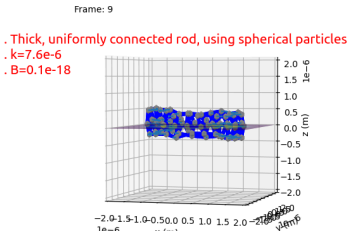


Testing the thick sphere based rod, the bending force had a much higher impact (due to the greater number of connections present, which seemed to bring about perhaps a bit more instability from the bending force at least with it). For this, I chose a larger spring constant and smaller bending force, and got a most correct plot seen on the right, with some unusual creasing at the edge directly brought about by the bending force. These parameters also worked well when cylinder particles were used instead of standard spherical particles.

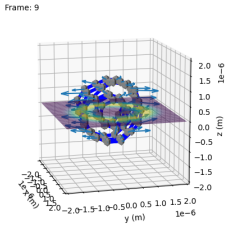


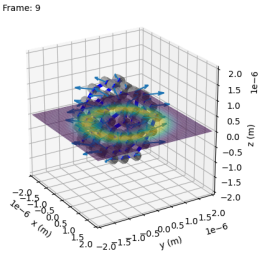




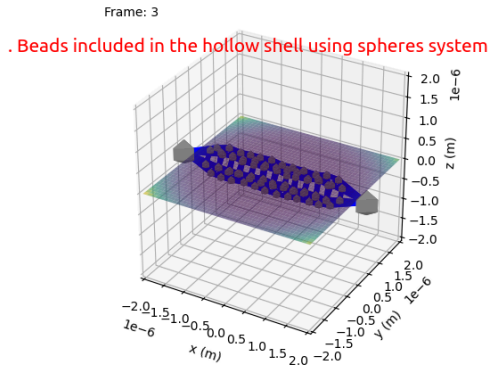


For the layered models, they do appear to be stable with the parameters of the previous system, however they are far larger than them, as the current ‘connect via distance’ function would connect the layers otherwise, hence I need to add another connection method that looks in plane about some radius to determine connections to allow the layers to be made closer together, and so model the same rod as the other systems.

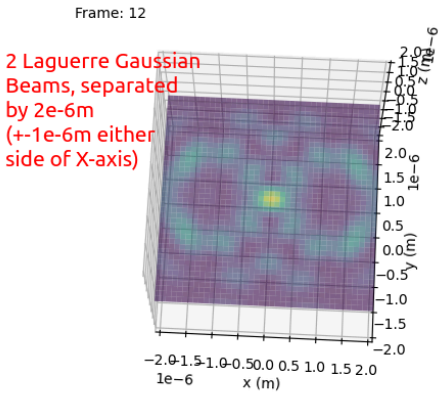




I have now also added a system to include beads in the models created. This generates N beads at chosen points, then connects the system by distance as before, but now includes an additional step that connects beads to any other particle (including other beads) in the system (note however that other particles cannot connect to beads in this first parse). My partner has worked on a system to translate the beam as the simulation occurs, which should let us stretch the particles out. These beads generated can have arbitrary material properties to the rest of the system (so we can have RBCs and silica beads together).



21/1/25

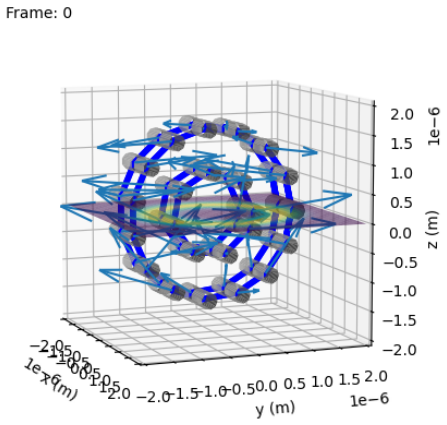


Today when thinking about how to confirm which model is correct I found this paper ***"Buckling of a Single Microtubule by Optical Trapping Forces: Direct Measurement of Microtubule Rigidity"***, which considers the measurement of the stiffness of microrods (which it also mentioned appear to have varying stiffness depending on the sample's length, which is an unusual property). This method also uses beads attached to the rod to manually trap (and so allow bending) of the ends of the rods. This paired with the red blood cells having the same approach (trapping beads attached to sample) implies I should try to implement this behaviour as soon as possible, which I will start working on now, as well as testing to ensure the rod models don't and so are physically sensible.

The first step with this is to ensure I can have multiple beams within a single system, so I can trap multiple beads at once. The figure above shows this with 2 Laguerre beams, which appear to be summing and interacting correctly when separated. Now I need to (1) find bead parameters that trap in other beams (like Gaussian beams, referred to as TEM_00 beams in several papers), and make sure that just the bead is trapping and not interacting too heavily with the other material itself (which could possibly be fine, however I feel as though this would would break lots of trapping).

20/1/25

My plans for today are to introduce another model for spheres/rods where you have unconnected membranes of particles to see if they can also model these scenarios without exploding/introducing other problems related to particles slipping through each layer, etc. I also want to continue making sure each of the other models I made do not immediately explode, then try to find comparisons in literature to try and assess which model is performing the best. Another small adjustment is to move the spring and bending coefficients into the ***simulation()*** arguments too so each adjustment won't affect the other systems.



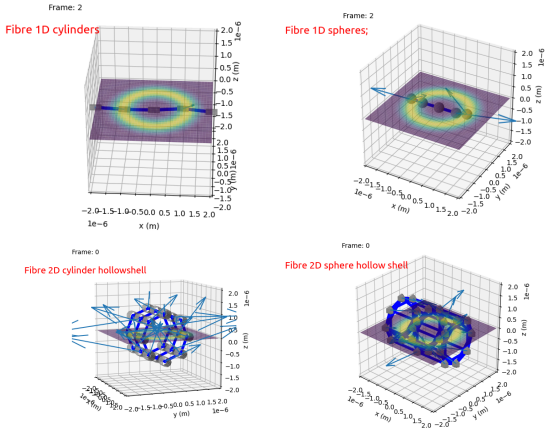
I have also now generalised the DipolesMulti2024 file to allow constants (e.g. for spring and bending constants) to be varied from this file (without having to manually adjust DipolesMulti2024), as well as allow the different forces to be selected from here too. This required lots of functions to be reformatted, as many variables were no longer global.

19/1/25

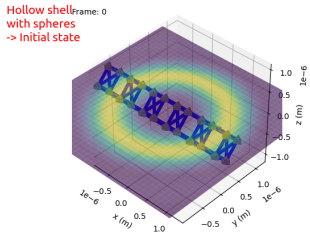
With the newly made cylinder primitive, I want to test the deformation for a series of objects;

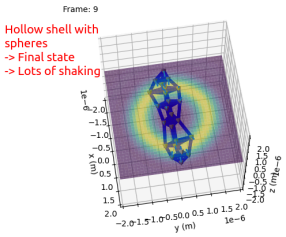
(1) 1D Line of spheres, AND line of cylinders
(2) 2D line of spheres, AND cylinders -> Hollow shell formed in a circle
(3) 2D line of spheres, AND cylinders -> Where they are formed by surrounding the 1D line with a circle of other 1D lines, each connected radially
(4) 2D line of spheres, AND line of cylinders -> But with connects made randomly (not formed in lines of radial connections specifically), as random N within some close range

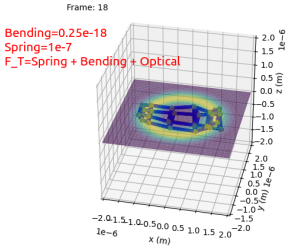
I can then compare the deformation of each of these cases to see what difference each of these models make in terms of the amount of flexibility provided. I can also then look at the forces experienced by each to see if, for example, the hollow shapes are missing out on optical force compared to the filled in shapes, which will inform the choice of shape wanted when trying to model the red blood cell. ***NOTE;*** equally, I should probably also test a ***water filled hollow shape*** as Simon tested in his simulation, however in our system we assume the particles are already surrounded in water medium, so could be that this actually models the system incorrectly (would have too much internal pressure).

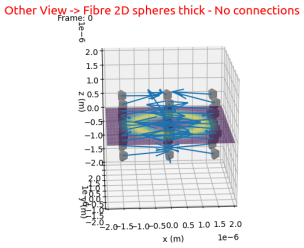


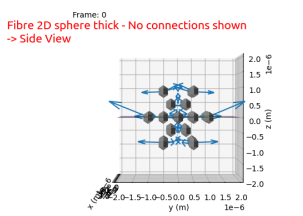
I decided to leave the random connection generations or now as it would introduce too much variance to be of use currently I believe. When testing a simple hollow shell cylinder made of spheres, it immediately shows exploding behaviour (with stiffness=1e-7, bending=0.5e-18) with a ***[spring, bending, optical, buckingham]*** force set used. When ***Buckingham and optical*** forces are ***removed***, I still see the shaking, meaning it is due to the constants chosen which can be varied. By choosing a smaller bending and reintroducing the optical force, I now see the shape compress, but not violently shake as before (some wobbling still seen from brownian

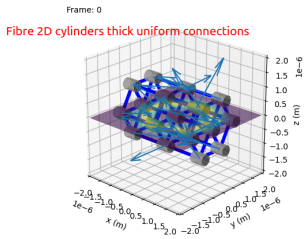


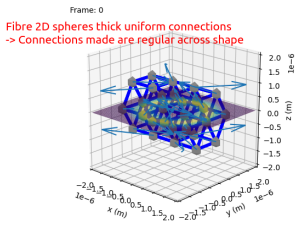






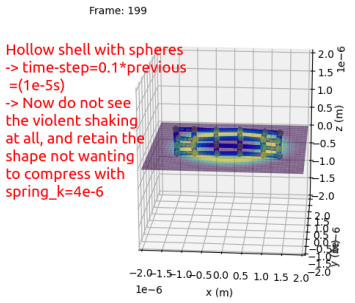


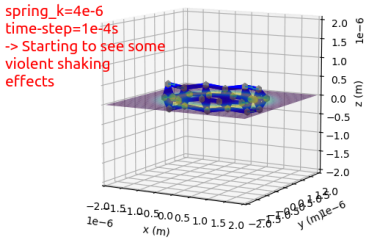


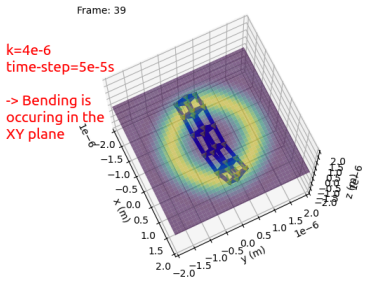


motion, which could likely be improved by running the simulation at a smaller timestep).

When running the simulation slower, with a string spring constant to prevent the shape being crushed in the beam, I was able to tune it to (1) deform only slightly (or allow more or less if wanted, by decreasing or increasing k) and (2) remove lots of the shaking behaviour whilst maintaining the qualities I wanted. This tunability is a good sign, but I now want to test that idea in a beam that will bend the rod, e.g. edges move up and center moves down, in order to more clearly see that effects of the changed parameters, e.g. how much exactly the bending constant is affecting the rigidity of the rod. My first thoughts are that this will require 3 bessel beams, 2 at the sides and 1 in the centre, pointing in different directions.



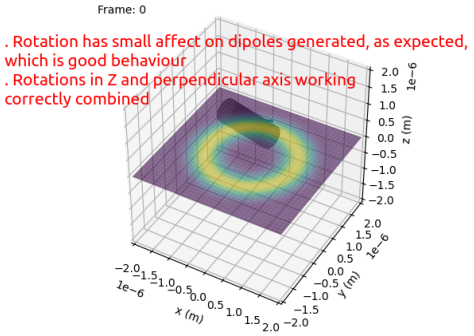


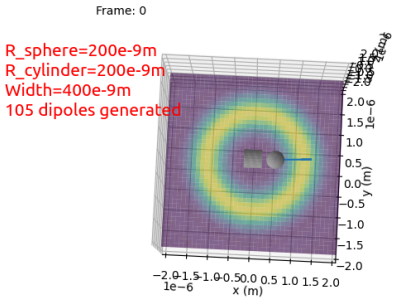


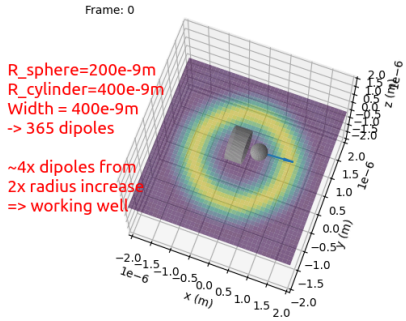
On the previous note, it is worth saying as well that if this 3 beam approach doesn't work, the Laguerre beam does appear to bending this rod in the XY plane too, so it could be used, however I am worried that this may be more telling of the particle sizes used as oppose to the larger rod dynamics, as Laguerre beams have a tendency interact differently with these different sized systems (at least they do here where the particles still are not particularly close together). Note as well that the inclusion of the Buckingham force did appear to help this bend occur, hence it may just be an artefact of some other small issue.

18/1/25

After talking to my supervisor yesterday, the continuous limit tests and changes to the spring and bending calculations seemed good, but if we wanted to expand the connected particle interactions we could also add a torsion term too (however we felt that this was likely unnecessary as the term is usually quite small, and for the cases we are testing with forces usually applied to stretch the mesh rather than twist it, this would provide little benefit). We also discussed how to model the red blood cells, where it was mentioned that he had previously modelled these cells as shells filled with water particles, which resulted in a naturally occurring biconcave structure when water was removed, so this approach could work, however it is possible something else may work better. For now, we decided to consider a more simple case of fibres being pulled, and want to observe how the optical/total/specific components of force varied as we changed our model, e.g. from a 1D fibre as a string of spheres, to a lattice of connected spheres to give a 2D cross section. This will hopefully give some insight as to how a more complicated 3D shape should be modelled to still get correct optical forces (as it being hollow would result in forces on dipoles within being ignored, hence this approach would not work).







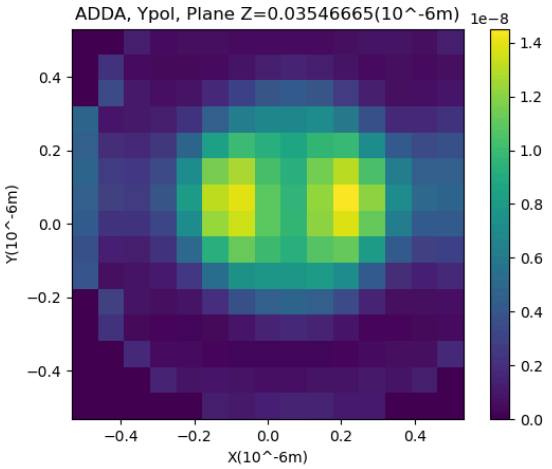
$$R = \begin{bmatrix} u_x^2\,(1-\cos\theta)+\cos\theta & u_x u_y\,(1-\cos\theta)-u_z\sin\theta & u_x u_z\,(1-\cos\theta)+u_y\sin\theta \\ u_x u_y\,(1-\cos\theta)+u_z\sin\theta & u_y^2\,(1-\cos\theta)+\cos\theta & u_y u_z\,(1-\cos\theta)-u_x\sin\theta \\ u_x u_z\,(1-\cos\theta)-u_y\sin\theta & u_y u_z\,(1-\cos\theta)+u_x\sin\theta & u_z^2\,(1-\cos\theta)+\cos\theta \end{bmatrix}$$

Therefore today I have written a new primitive into the SH_DDA code to generate ***cylinder primitives***, so we can create lattices for fibres using both spheres and cylinders as primitives within it (close to each other) to see if this models them better.

The cylinders added can be any radius and length, and can also be rotated to any angle (specified by a ***Z rotation*** followed by a ***perpendicular rotation***, which is a rotation in the axis perpendicular to the normal of flat circular face of the cylinder, acting about the COM of the cylinder). Note however that like with the torus sector this will not have any rotation in the dynamics simulation, so once it has been generated at some orientation it will only ever be translated with that same orientation.

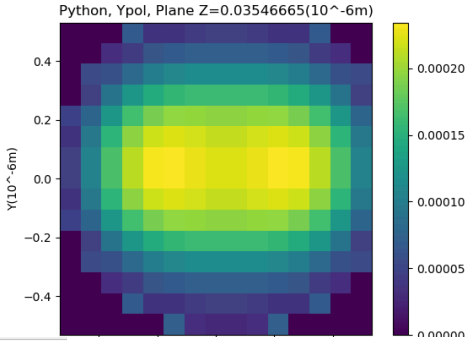
17/1/25

I have continued trying to get the ADDA force calculation working before meeting with our supervisor for the first time to discuss all the changes we made over Christmas. I now have the code from ***Kyle Hall*** working and generating a set of forces for each dipole now, however when I run this with a system I am considering (e.g. sphere16x16x16 with plane wave, 1.064lambda, 1.5+0j refractive index, 15 dpl, etc) I do not get agreement between his forces and ADDA's calculated forces.

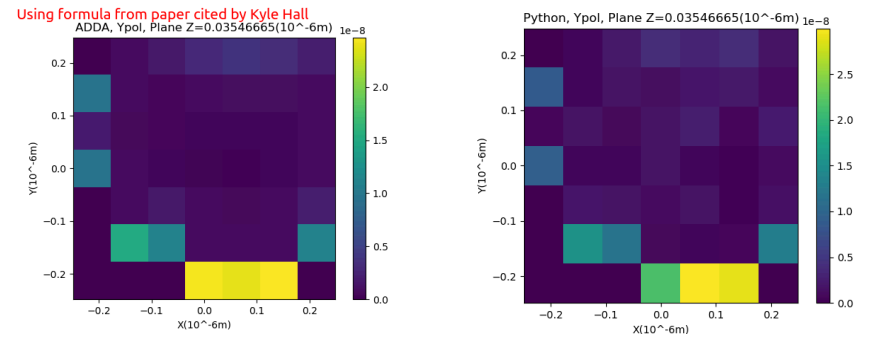


https://github.com/kyleashleyhall/Modelling-Light-Driven-Micro-Machines/blob/ADDAdevelopment/ADDA/Linux/FinalResults/Cone/TorqueForceAnglePosition/Fancyplot.py

The pattern does resemble ADDA's calculator, perhaps with an additional incident field overlaid? Testing this did not help, as the gradient of the plane wave is essential 0 (constant field across), and just considering a decaying term similar to the internal |E|^2 field being subtracted (at different scales, just to get a picture of the patterns able to be formed) did not result in a matching pattern either.



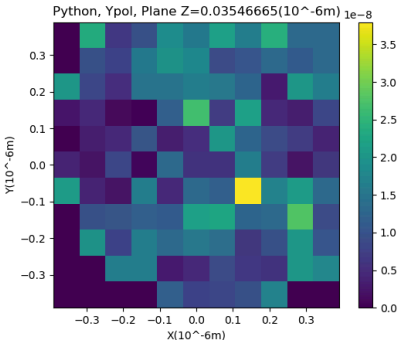
Using a formulation of the ***Aij*** interaction tensor from a paper I found cited by Kyle Hall (***Application of fast-Fourier-transform techniques to the discrete-dipole approximation***), and using a Y polarisation not X for the plane wave (hence why figures appear rotated) then I get a good match for the sphere8x8x8 case. This approach takes slightly longer than previous similar approaches (likely inefficient ***np.cross***, more so than some other np functions?). When tested on a sphere of size 12x12x12, the results for the pattern matching were not good,




```
def get_E_interaction(pos, dip_positions, dip_polarisations, k, pos_ind):
    number_of_dipoles = len(dip_polarisations)

    E_interaction = np.zeros(7, dtype=complex)
    E_interaction[:3] = pos

    E_joint = np.zeros(3, dtype=complex)
    for j in range(number_of_dipoles):
        if(pos_ind!=j):     # NOTE: For gradient, pos_ind is the point being considered about
            rij_3 = pos-dip_positions[j]
            rij_ = np.linalg.norm(pos-dip_positions[j])
            E_joint += -( (math.exp(1.0j*k*rij_))/(rij_**3) )*( (k**2)*(np.cross( rij_3, np.cross(rij_3,dip_polarisations[j]) )) + ((1.0-1.0j*k*rij_)/(rij_**2))*( (rij_**2)*(dip_polarisations[j]) - 3.0*rij_3*(np.dot( rij_3, dip_polarisa

    E_interaction[4:] = E_joint
    E_interaction[3] = np.dot( E_interaction[4:], E_interaction[4:].conjugate() )
    return E_interaction

print("Calculating time-averaged forces for "+beam_spec[0]);

dipole_forces = np.zeros( (len(dip_positions), 7), dtype=complex)
k = (2.0*np.pi)/(beam_spec[1])

for i in range(len(dipole_indices)):
    if(i % 20 == 0):
        print("Dipole "+str(i)+"/"+str(len(dipole_indices)))
    # For every dipole in particle of interest
    dip_ind = dipole_indices[i]
    pos = dip_positions[dip_ind]
    dip_force = np.zeros(7, dtype=complex)
    dip_force[:3] = pos

    # Gradient conj. of scattered field
    E_func_incident    = partial(Calculate_IncidentField_Point, beam_spec=beam_spec)        ### COULD MOVE OUT OF FUNC FOR MORE SPEED ###
    E_func_interaction = partial(get_E_interaction, dip_positions=dip_positions, dip_polarisations=dip_polarisations, k=k, pos_ind=dip_ind)

    # Gradient conj. of just the incident field
    for comp in range(3):
        # For each component, get gradient, multiply by factor
        E_conj_grad_inc   = conj_gradient(E_func_incident, pos, comp)       # Gradient of E inc field at the position given
        E_conj_grad_inter = conj_gradient(E_func_interaction, pos, comp)    # Gradient of E scat field at the position given

        dip_force[4+comp] += np.dot(dip_polarisations[dip_ind], (E_conj_grad_inc+E_conj_grad_inter)) # Set |Fx, Fy, Fz|
        dip_force[4+comp] = ( (0.5)*dip_force[4+comp] ).real                                        # Set |Fx, Fy, Fz|
    dip_force[3] = np.dot(dip_force[4:],dip_force[4:].conjugate())     # Set |F|^2
    dipole_forces[i] = dip_force
if(mode == 1):
    dipole_forces[:,2] = dipole_forces[:,3].real
    dipole_forces = np.array(dipole_forces, dtype=float)
return dipole_forces
```

however the magnitude of the forces were fine.

There also appears to be a problem with the plotter picking too many points when trying to extract a plane, which can cause some of the noise seen if it chooses two dipoles with the same X-Y position, but different Z positions, where it will then overwrite the higher value it finds. This needs fixing to ensure it does not cause any problems.



16/1/25

Now that the interim report has been submitted and successfully completed, me and my partner have started working on the main program again, where we were fixing some of the older functions we developed, which had suddenly started producing invalid graphs. We explored the “arbitrary plotter” which had somewhat recently been changed to allow total forces to be plotted too, which we thought may have been interfering with results previously made with just a single force in mind. This turned out not to be the problem, but was in fact due to the Laguerre not having a default ***Order*** value being set, as the Bessel beam did not utilise orders, and so we removed it to prevent any unexpected behaviour. This means the error only occurred for Laguerre plots after the Bessel force had been used (when testing spring forces), and had only affected the beam by essentially making it smaller by default, which is fine for the work being performed (only cared that some force was being applied, exact Laguerre properties or sizes were not important), meaning all our previous data is unaffected, and our new data is valid irrespective of this beam. With this fixed, the old plotting functions now work correctly.

I have also been working on the ADDA force calculator more as well, where I have tested some different formulations of the force (Green’s tensor from different sources with and without what I believe to be CGS units). I am currently using a (1) incident beam gradient term and (2) a scattered interaction field gradient term, calculated separately and added (once the gradient of each is found). This gives similar plots to ADDA for a plane wave for smaller numbers of dipoles, but as more are added it begins to look more significantly different. The current results look very similar to the near field calculations used before, as the equation uses the same factors (Greens tensor interaction + some incident amount by default), but work correctly with small positional gradient steps (hence I have gone back to using ***sys.float_info.epsilon*** as my step). I have also attempted to reverse engineer the calculation performed by SH_DDA’s E field gradient, however this method uses large matrix products, Eigen and Cpp principles so has difficult points when trying to convert it. If this was working in Python, then I could compare this

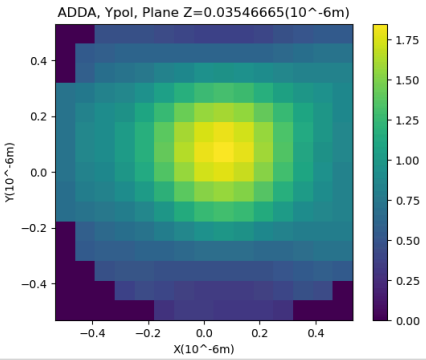
```
def scattered_force(p_array, r_array, kvec):
    grad_matrix_T = np.zeros((3*number_of_particles,3*number_of_particles), dtype=complex)
    Gradiiblock = np.zeros((3,3), dtype=complex)
    Gradijblock = np.zeros((3,3), dtype=complex)
    pvec = np.zeros(6, dtype=complex)
    rvec = np.zeros(6, dtype=complex)

    # Get Gradients
    gradEE = np.zeros(18, dtype=np.float64) # // to hold 9 complex values => 3x3 matrix
    #kvec = beam_collection->BEAM_ARRAY[0].k; # // (scalar) assuming same for all beams!
    for i in range(number_of_particles):
        ti = 3*i;
        # print("p_array[i]   = ",p_array[i])
        # print("p_array[i,0] = ",p_array[i,0])
        # print("p_array[i,0] = ",p_array[i,0].real)
        pvec[0] = p_array[i,0].real;
        pvec[1] = p_array[i,0].imag;
        pvec[2] = p_array[i,1].real;
        pvec[3] = p_array[i,1].imag;
        pvec[4] = p_array[i,2].real;
        pvec[5] = p_array[i,2].imag;
        for j in range(number_of_particles):
            tj = 3*j;
            if (i==j):  # // All zeros here
                grad_matrix_T[ti:ti+3,ti:ti+3] = Gradiiblock;
                # for i_off in range(3):
                #     for j_off in range(3):
                #         grad_matrix_T[ti +i_off,ti +j_off] = 0.0 +0.0j;
            else:       # // Compute the matrix
                rvec[0] = r_array[i,0] - r_array[j,0]
                rvec[1] = r_array[i,1] - r_array[j,1]
                rvec[2] = r_array[i,2] - r_array[j,2]
                gradEE = grad_E_cc(rvec, pvec, kvec)

                gradEE_complex = gradEE.view(np.complex128)
                Gradijblock  = gradEE_complex.reshape(3, 3)

                grad_matrix_T[tj:tj+3,ti:ti+3] = Gradijblock.T

    # Get Forces
    optical_force_array = np.zeros((number_of_particles, 3), dtype=complex)
    one_polar = np.zeros(3, dtype=complex)
    for i in range(number_of_particles):
        ti = 3*i
```

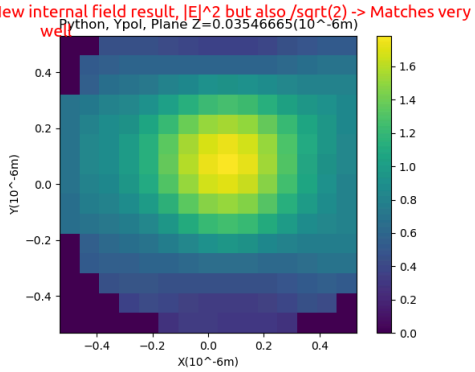
```
Generating sphere; 12x12x12
=== ADDA Log ===
WARNING: (../param.c:2424) Directory 'output_data' already exists
Calculating time-averaged forces for plane
dipole_forces =  [[0.+0.j 0.+0.j 0.+0.j ... 0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 0.+0.j ... 0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 0.+0.j ... 0.+0.j 0.+0.j 0.+0.j]
 ...
 [0.+0.j 0.+0.j 0.+0.j ... 0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 0.+0.j ... 0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 0.+0.j ... 0.+0.j 0.+0.j 0.+0.j]]
AFTER dipole_forces =  [[ 0. +0.j  0. +0.j  0. +0.j ...  0. +0.j  0. +0.j
.j]
 [ 0. +0.j  0. +0.j  0. +0.j ...  0. +0.j  0. +0.j  0. +0.j]
 [ 0. +0.j  0. +0.j  0. +0.j ...  0. +0.j  0. +0.j  0. +0.j]
 ...
 [nan+nanj nan+nanj nan+nanj ... nan+nanj nan+nanj nan+nanj]
 [nan+nanj nan+nanj nan+nanj ... nan+nanj nan+nanj nan+nanj]
 [nan+nanj nan+nanj nan+nanj ... nan+nanj nan+nanj nan+nanj]]
Dipole 0/338
Dipole 20/338
Dipole 40/338
```

method to what is currently being used and hopefully diagnose the problem (or just continue with this method). The E field used here is given in ***"CLASSICAL ELECTRODYNAMICS, 3RD ED"***.

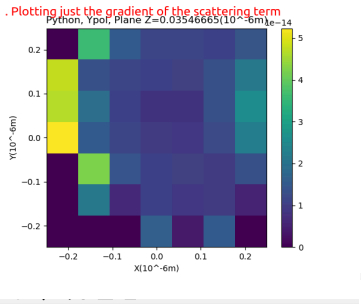
The figure shows a snippet of the conversion of the CPP code from SH_DDA to a Python version (implementing everything for the scattered force, beam force still found as usual in Python). When running the program, I get ***NaN*** values after a single cycle, so I will have to follow the process and see where it breaks. I have to cast from a ***complex128 -> complex*** at one point, which should be fine however I can see the lower precision value giving NaN when it is very small (however we should not have values that small, hopefully order 10^-12 to 10^-9 approximately).

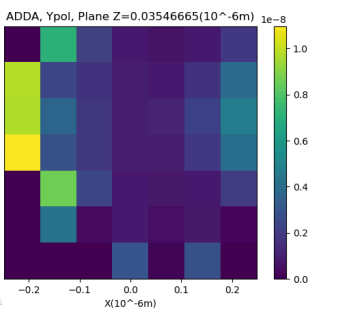
8/1/25

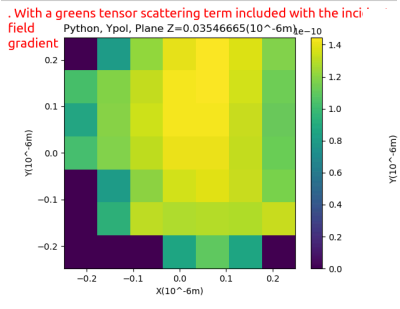


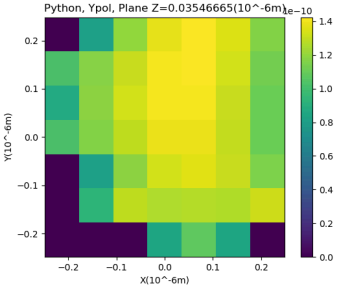


When talking to my partner about my ADDA progress, we realised that I had made quite a critical mistake in my internal field calculation, where I was using the near-field as a way of interpolating in-between dipole centers, however I now believe that the volume of the dipole is occupied by a uniform electric field across it (due to having a uniform polarisability for that dipole volume element), hence I should have been using the formula ***E=P/alpha*** instead to find these internal fields and should have entirely avoided performing near-field calculation here, as near-fields should be located outside the particle purely but sufficiently close. However, this does raise problems as to how gradient calculator of the E field should be found (when calculating forces), as now this implies the E field at a dipole is uniform across its radius, hence small steps will give 0 gradient (before, the near-field calculation was allowing this to have more continuity between dipole points, which was giving a result which appeared to be 'converging' towards the correct answer given by ADDA). I believe the near-field result looked correct before as the near-field may have tended towards this simpler linear response equation (which is what you would hope for a near-field, e.g. near the particle surface hence near the dipoles there hence look sufficiently similar hopefully). It is worth noting however this improved approach is actually giving a result with magnitude off by ***sqrt(2)** (too large) which I need to resolve, but I believe this may be a result of plotting |E|^2, not simply E or |E|. I will need to consider how this gradient calculation can be performed again.









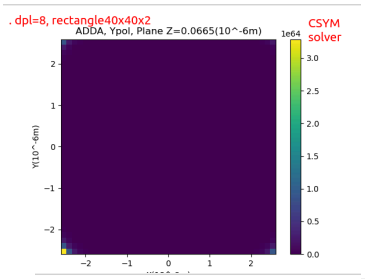
Considering Simon's program and talking with my partner it seems that the total E gradient is actually just the E gradient of the incident field (from the beam) + a scattering term because as we talked about before the E gradient across the dipole is just 0 due to it being constant across (internal from before, hence must be found fresh). Hence I just need to have my gradient calculation use the Laguerre/Plane wave beam instead of the near-field used before, and then use the polarisations already calculated to get the force. Currently when I perform this the result is not correct (however the magnitude is not too far off), but mainly the pattern retrieved is very different. The figures on the right show when the gradient step is changed back to ***float_info.epsilon***, as with

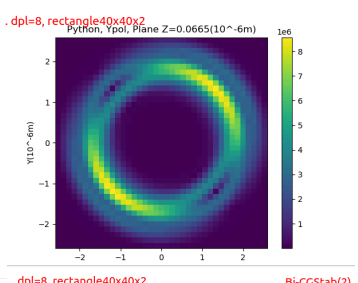
the new approach the gradient doesn't appear to get worse with smaller steps (appears to converge closer to the desired 10^-8 magnitude, being of order 10^-14 when step = 10^1). This being said, when just considering the gradient of the scattered/interaction term (not incident field gradient), then we get a plot which looks quite similar to the actual result, but with a much smaller magnitude.

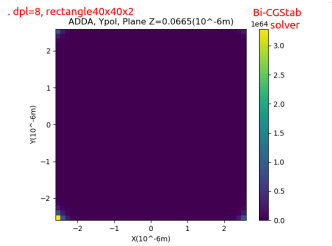
Therefore my next step is to figure out why the scattering calculation is wrong, which I will try to do by referring back to the ADDA manual for scattering quantities, and by comparing to SH_DDA to see what I am missing.

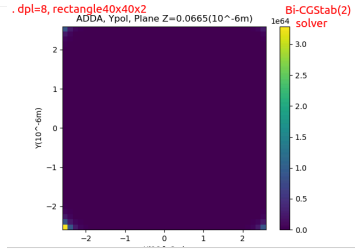
7/1/25

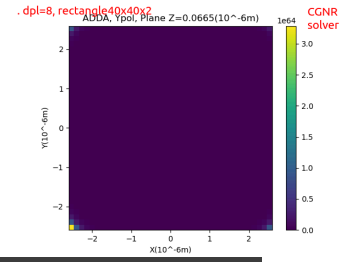
Testing the ADDA and Python version with dpl=8 on a 40x40x2 rectangle so we can see the full ring, we see that ADDA is not correctly showing this for the **CSYM, Bi-CGStab(2), Bi-CGStab and CGNR** iterative solvers, where all showed no ring formation (instead just showed extremely large peaks at the corners of the view, even when viewed on different scales). From this I need to look at ADDA's implementation and see where and why it is different to Python's version.











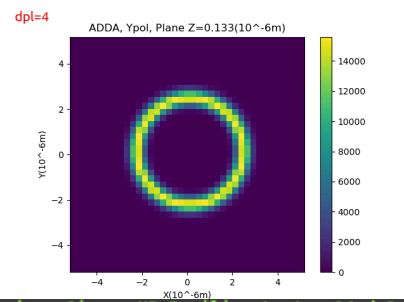
The variables being used in Python are given in the screenshot, where the integral is between 0 and k. In ADDA, t1 and t2 are alpha and beta respectively. These main constants (alpha, beta, k, z_R, l, p) all match, so I should check whether the function values match well now. T6, t7 and t8 are the E components for the first dipole checked, which give unusually high values compared to Python (~10^8 for ADDA, ~10^6 at peak for Python). When comparing values in the E_x, E_y, E_z component calculator for similar kappa values, I found that ADDA was evaluating many terms to 1+0i, which appears to be a problem with values found using the ***imExp()*** function, which seemed to be fixed after changing this to ***cexp()***. When running the program after this, I did get a very clear Laguerre Gaussian beam, however the iterative solver gave an error about the solution failing due to small values, as encountered before. When using the ***-iter csym*** solver I no longer get these errors and I get an identical E field plot.

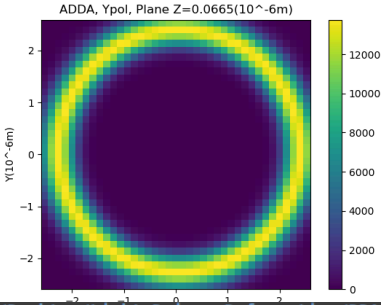
```
=====
Variables   Python
=====
l= 0
p= 8
k= 5.905249348852994
w_0= 0.6384
z_R= 1.203355650031034
alpha= 1.0
beta= 1.0
```

```
CoupleConstant: 0.000174748381+4.242413181e-06i
local_nvoid_Ndip= 2176
wavelength= 1.060000e+00
k = 5.927533e+00                    ADDA
w0= 6.360000e-01
integral_start= 0.000000e+00
integral_end  = 5.927533e+00
t1 = 1.000000e+00, 1.000000e+00
t2 = 1.000000e+00, -1.000000e+00
z_R= 1.198832e+00
t6 = -1.259314e+07, 2.112357e+08
t7 = 2.112357e+08, 1.259314e+07
t8 = -1.412758e+09, -8.324131e+07
x_0 = E_inc
RE_000 = 6.9542027407E-01
```

```
k = WaveNum;    //In micrometres, is a double, NOT dou
wavelength = (2.0*PI)/(k);
w0 = 0.6*wavelength;            //<--- Was defined els
integral_start = 0.0;
integral_end   = k;

t1  = 0.4*cpow(10,-8)*(1.0+1*I);        //alpha
t2  = 0.4*cpow(10,-8)*(1.0-1*I);        //beta
z_R = k*(w0*w0)/2.0;    //z_R
```

This just leaves me with fixing the Python version to match the ADDA version again, then force and scattering calculations can be performed.

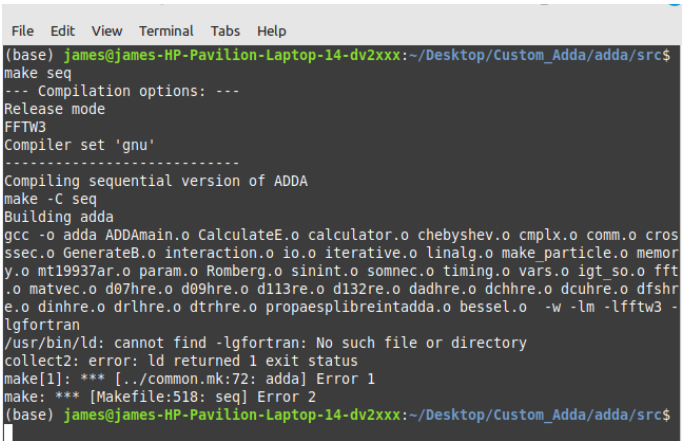




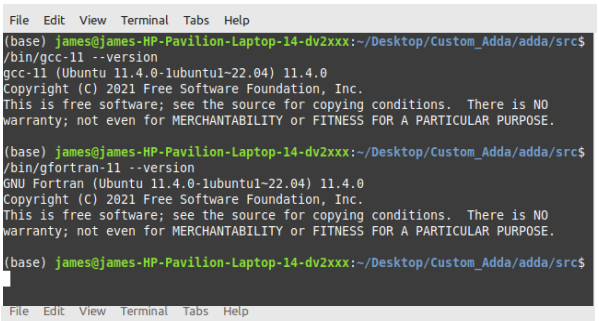

```
(base) james@james-HP-Pavilion-Laptop-14-dv2xxx:~/Desktop/Light-Driven-Deformation/DDA
programs/ADDA_programs/ADDA_force_extension$ python adda_force_ext.py plot_adda_Einc
Program Started
Generating rectangle; 40x40x2
=== ADDA Log ===
WARNING: (../param.c:2424) Directory 'output_data' already exists
ERROR: (../iterative.c:1066) QMR_CS fails: |vT.v|/(v.v) is too small (6.5e-18).
```

6/1/25

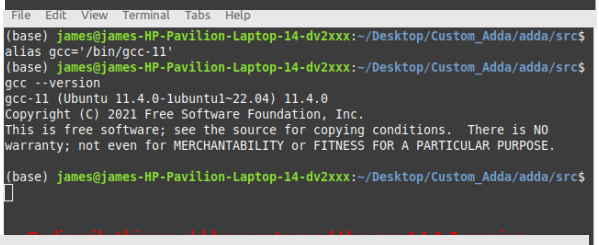
Today I plan on getting my older gcc version (needed for compiling ADDA, 11.4.0) working alongside my newer version (needed for SH_DDA, 14.1.0) so I can test if the Laguerre beam is generating the correct incident field. This would then allow calculations to be performed once I find a better gradient calculation (for force calculation in Python). After this if I have time I plan to simply continue writing my report.



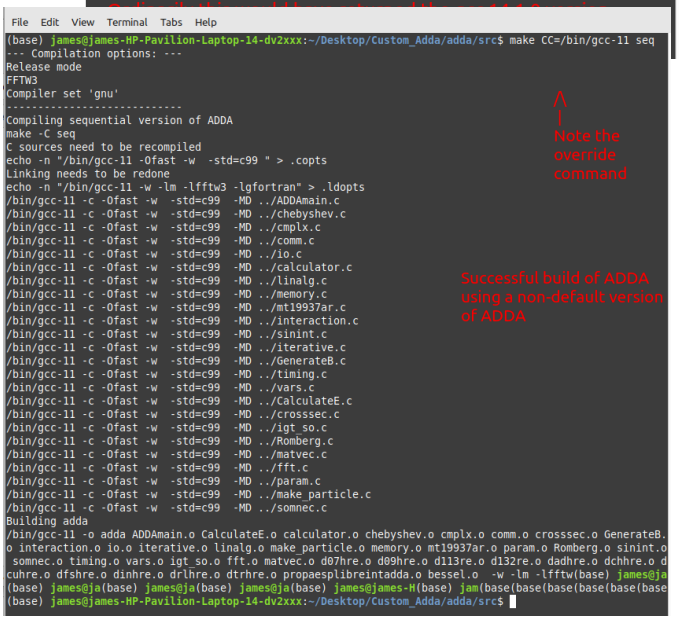
The problem I was encountering was that ADDA makes use of the ***gfortran*** compiler, but my system does not have gfortran 14.1.0 installed as I had to manually install gcc 14 in order to get it working on my Linux Mint XFCE computer, and could not find a way to install this version on gfortran on here manually, and the automatic update cannot detect updates for this (hence the "-lgfortran" missing error). Therefore my solution is to go back to gcc 11 and gfortran 11 version I previously had installed and try to use them, but without removing the gcc 14 version I currently have which is needed for SH_DDA and was quite tricky to get installed in the first place. The error above is the error I get when trying to compile my previously working ADDA code.



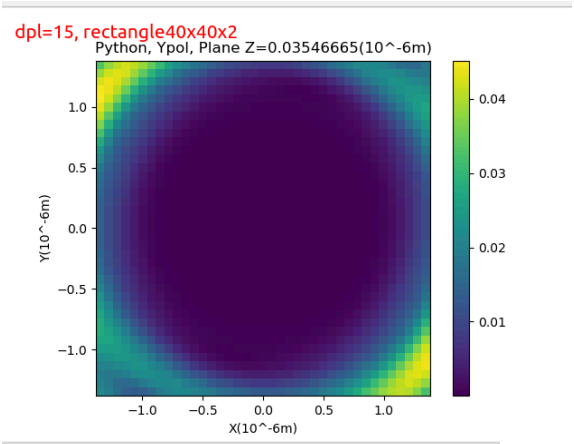
By running ***"alias gcc='/bin/gcc-11"*** I tried then running the makefile with ***"make seq"***, however this did not work as I believe the makefile specified it wanted to use the default version within it, hence I actually needed an override command;

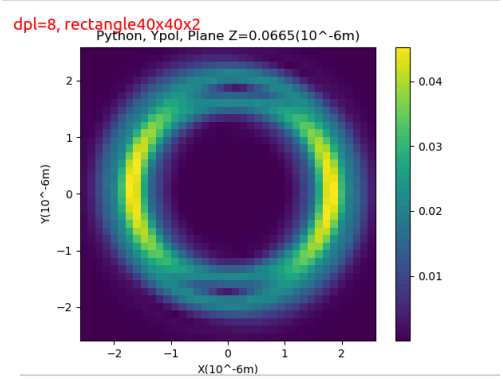


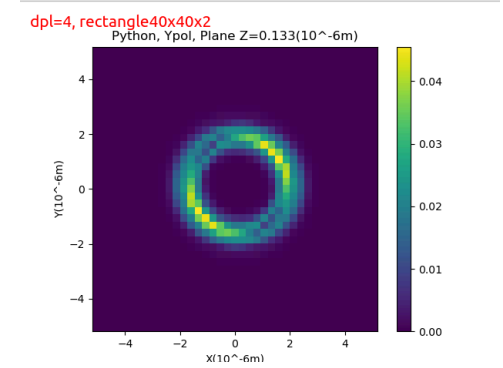
"make CC=/bin/gcc-11 seq", which worked and built the program correctly. Note that this will work for any gcc version installed on the system, default or not. This process was explained to me within the "MakeFile" inside the adda/src/ folder. To be explicit this means the alias was not needed, but was an interesting and possibly useful tool for the future.



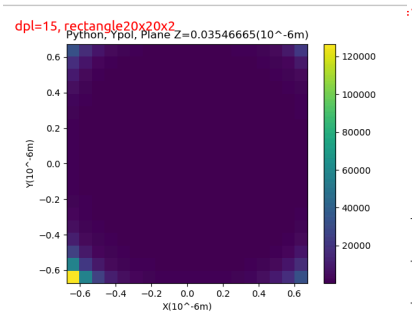
I have now quickly written a function to generate arbitrarily sized geometry files for cubes and spheres, as in order to see the beam incident field generated by ADDA I will have to get the incident field at each dipole. Hence in order to get a better reading I will want a higher density of dipoles. My plan is to increase the number of dipoles in a rectangular ‘sheet’ of dipoles, visualise this layer for Python and ADDA and continue increasing the size of the grid until the full ring pattern can be seen, then I can increase the density of the grid. From this I can compare ADDA to the ‘ground truth’ Python version, which when tested before gave a good Laguerre beam. Running this for a rectangle of dimensions AxAx2 (2 chosen over 1 because spheres I tested were even integer widths, so I was reading at Z=lattice_spacing/2.0, hence is easier to choose this as 2 layers to also get ½ lattice spacing reading, otherwise would have to change the read position between different runs). The figures to the right show the incident field from the beam at each of these dipole positions (e..g no scattering, just looking at the incident beam).

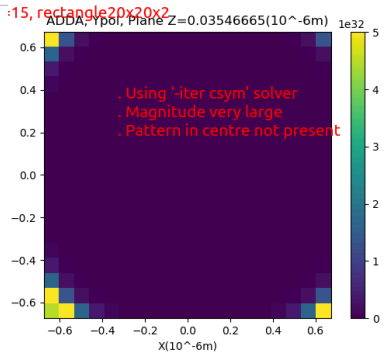






This seems to show a previous version of the Laguerre beam in which gaps are seen. I remember encountering this when first generating the beam, and believe it was due to certain components in the equation being assumed real when they were actually complex, however I will need to revisit my old notes when generating this beam to see where this came from and resolve it here. Once this is fixed I can run the ADDA version and compare the electric fields. It is worth noting here as well that for these plots ADDA was also generating plots too, however each of these failed due to the iterative solver, hence I need to run this with the ⅔ iterative solvers I found in ADDA that did not give errors, then see if they are generated correctly. I should also check the ADDA code to ensure the error on these Python plots (broken rings) are not repeated in ADDA, as I can see this causing problems with convergence at the opposite ring ends where the E field appears to disappear quickly.



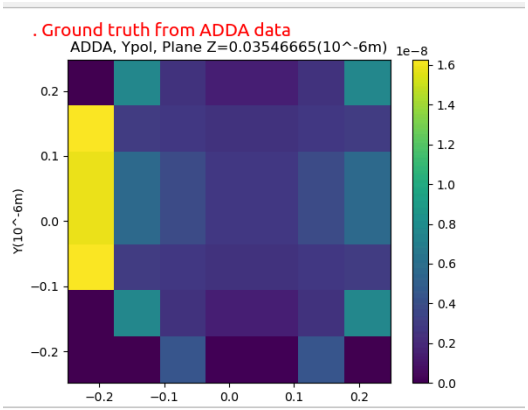


When initially testing the ADDA plot with the ‘-iter

csym' iterative solver (which is supposedly very accurate for symmetrical systems, and so I hoped would work well for the Laguerre beam -> however the circular polarization may be causing problems), which does look similar to the Python version here, however does have some regions of concern where it is reading a strong field not present in Python. I need to test whether a pattern will form in the centre for a more 'zoomed out' picture, or if the iterative solver problems will cause just this edge effect and nothing more, as I have seen in other plots previously in the project.

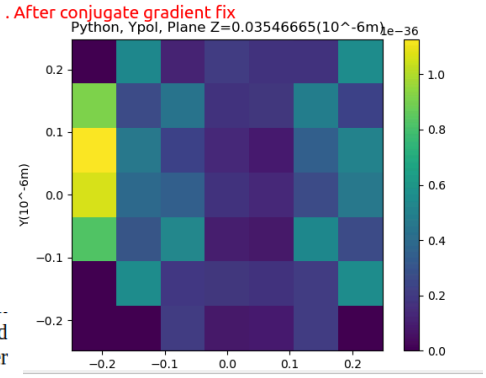
5/1/25

Today I plan to continue writing for my interim report and try to fix the issues related to the force calculation in Python. It was showing some progress yesterday with different patterns, but the magnitude looks as though it is overpowering the intricacies that make up the pattern.

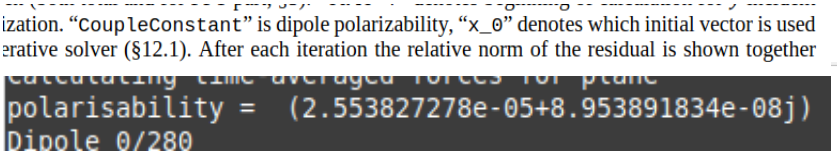


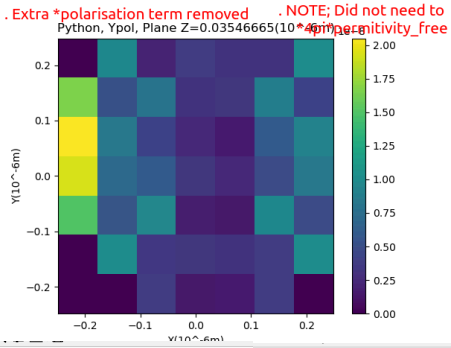
After fixing the gradient calculation, the magnitude of the force found is much more reasonable (10^-36 rather than 10^-101), which was due to incorrect division of the sys.float_info.epsilon value.

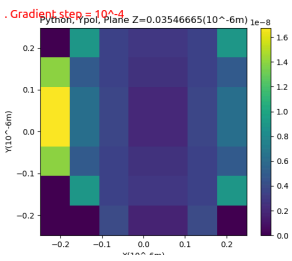
I have also found an additional *polarisability term in the calculation that was present in the previous method (when multiplying E by polarisability in order to the polarisation). When this is removed (as we are using polarisation directly instead here) we get the correct order of magnitude expected, as we have a slight offset due to the slight offset in the scattered field we have. This is great news for the force magnitude, however the pattern still does not match.

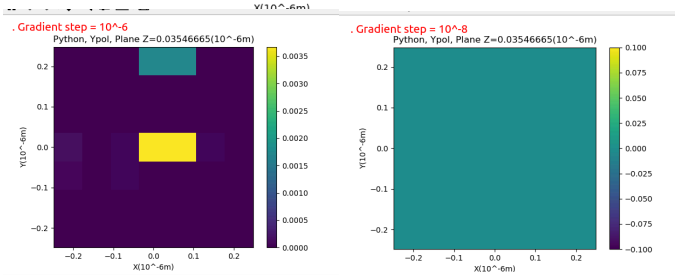


I have also realised now that the ***'couple constant'*** given in the ADDA log is the dipole polarisability, hence this can be used instead of having to calculate it. When generating data with this, the pattern is still offset as before and the magnitude the same, due to polarisability not being needed in this function as polarisation is already given (note however that the order is now ~10^-5, which is much larger, than before which could be in part due to ADDA scaling).

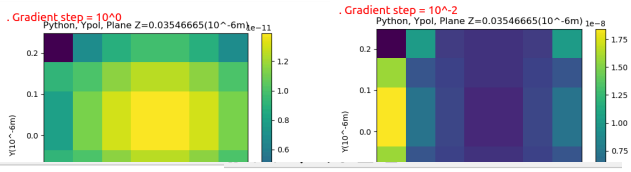


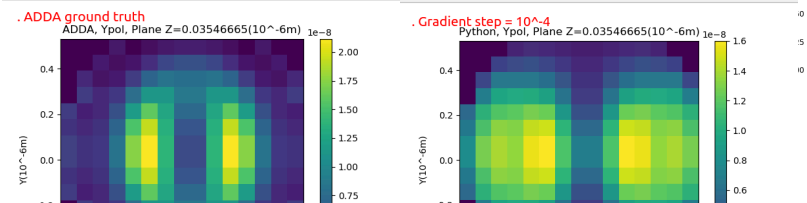




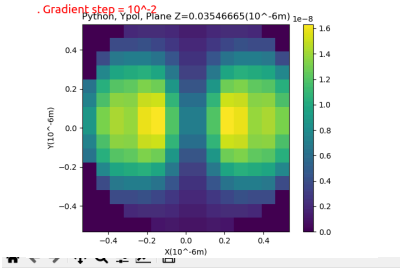


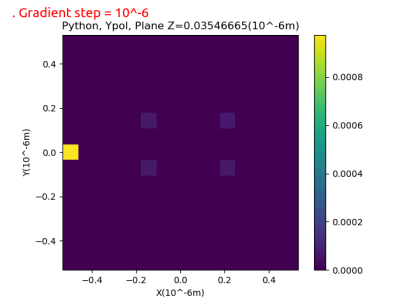
After adjusting all values in the calculation, it appears that it must be the gradient calculation that is causing the problem, hence I tested different steps in my central





difference partial derivative calculation (as seen above). An interesting note here is that the derivative is shown to be inaccurate for high values (>10^0), but also for small values (<10^-6 / 10^-8), however when used for step=sys.float_info.epsilon (which is 2.220446049250313e-16) we actually get a fairly good gradient calculation, which is perhaps due to this variable storing the small value in calculations better, hence allowing it to differentiate between 0.0 and some small value in +-*/ operations. The right figures show how the 16x16x16 force calculation looks using this new gradient step, which clearly is a much closer match to reality than before. Note as well that the positions used are already in micrometers, and so of the order of 10^0, hence gradient steps (in x,y,z position) of ~10^-5 (hence 10^-11m in reality) is a small jump and so is very much valid, it unusual however that the gradient appears to go to zero for smaller steps (you would hope accuracy to improve with smaller steps).





If I can now compute a more accurate derivative then that may lead to the final correct result, wherein after if I can get the previous gcc version again alongside gcc14, then I will be able to recompile and test the ADDA code with the Laguerre beam to ensure that is working too. If this all works fine then it would be possible for me to reproduce the SH_DDA particle tests and get tangential forces out from ADDA. This could potentially be time consuming however, so this may have to be skipped in order to progress the spring part of the project further.

3/1/25

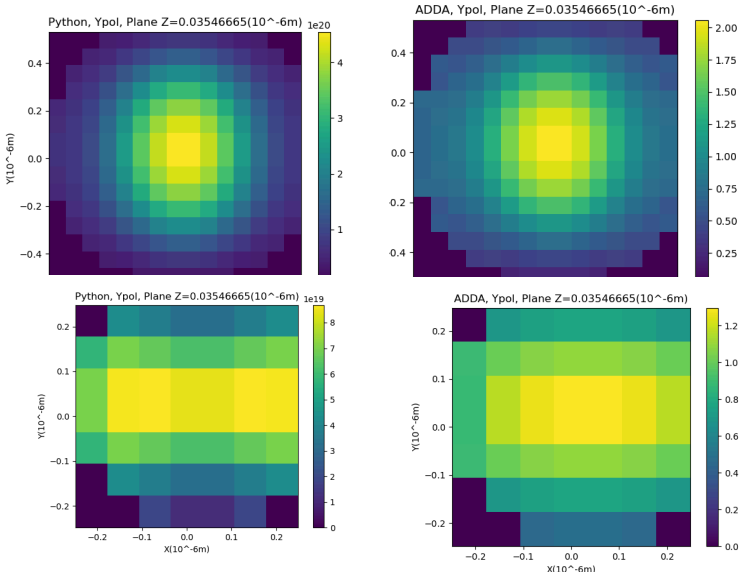
With yesterday's work still struggling to give the correct pattern of forces across the dipoles of the sphere (although there was progress in the right direction, with non-uniform patterns being observed), I am now going to try and debug the force calculation by comparing my code to that of ADDA's calculation and SH_DDA's calculation. With ADDA, force calculation is only supported for plane waves, hence I am expecting the code for its calculation to just be a form for a simplified case with the assumption it is only true for plane waves, which is not useful for me however there may be other useful information involved.

```
Eigen::MatrixXd optical_force_array=Eigen::MatrixXd::Zero(number_of_dipoles,3);
Eigen::Vector3cd one_polar;

for (i=0; i<number_of_dipoles; i++){
    ti = 3*i;
    one_polar = p_array.row(i);
    for (j=0; j<number_of_dipoles; j++){
        tj = 3*j;
        if (i!=j){ // Compute the matrix
            Gradijblock = grad_matrix_T.block<3,3>(ti,tj);
            optical_force_array.row(i) += 0.5*(Gradijblock*one_polar).real();
            // Need to check the ordering of above - has it computed correctly? - Yes
        }
    }
}
```

Considering SH_DDA, the process is as follows;

1. Polarisations are found in an array of complex vectors, which is 1D with each particle ordered in groups of 3 for each comp
2. This is unpacked into a list that has the real and imaginary Px, Py, Pz components
3. It then looks through every other dipole, finds displacement vectors between the two dipoles, finds the gradient of the complex conjugate of the electric field at the other dipole (is original-other for displacement vec => from other to original) then stores these in a matrix of conjugate gradients
4. It then performs matrix operations to calculate sets of the forces at once using ***F=0.5*Re[p.Grad(E*)]***.
5. After this another similar operation is performed, but for each beam present, hence using the incident E field from the beam (so he has split the calculation into just the scattering parts and beam parts done separately and combined with a +=)

As far as I can tell this matches the procedure I perform, except perhaps I am computing ***[Grad(E)]* rather than Grad(E*)***. After accounting for this with a new ***conj_gradient()*** function I still get the same error.



By checking and adjusting my Green's tensor calculation, I switch out some np.sqrt(np.sum(pow(...,2))) terms with np.linalg.norm() functions, changed math.pi s to np.pi s, and

ensured all constants used in the calculation were floats (e.g. used 1.0 not 1 values), this appeared to fix the scattering calculator (bar the units error, giving a 10^20 factor different in magnitude, but the correct pattern), only very minor differences are seen in the scattering calculated to the ADDA ground truth, which now also works for the 8x8x8 sphere (lower figure pairing) as opposed to only 'working' for the 16x16x16 just above.

$$\bar{\mathbf{G}}_{ij} \overset{\text{def}}{\equiv} \bar{\mathbf{G}}(\mathbf{r}_i, \mathbf{r}_j) \overset{\text{def}}{\equiv} \frac{\exp(\mathrm{i}kR)}{R}\left[k^2\left(\bar{\mathbf{I}} - \frac{\mathbf{R}\otimes\mathbf{R}}{R^2}\right) - \frac{1-\mathrm{i}kR}{R^2}\left(\bar{\mathbf{I}} - 3\frac{\mathbf{R}\otimes\mathbf{R}}{R^2}\right)\right], \quad (44)$$

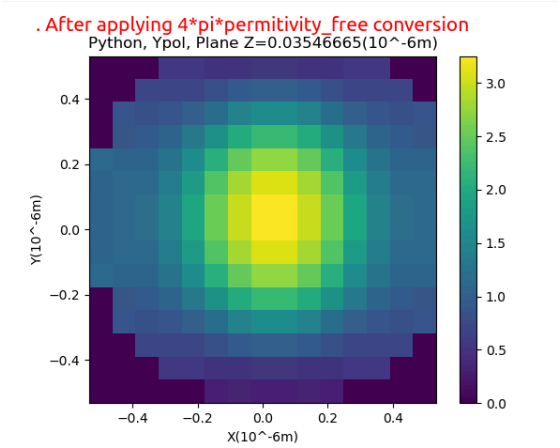
$$\bar{\mathbf{G}}(\mathbf{r},\mathbf{r}') = \bar{\mathbf{G}}(\mathbf{R}) = \frac{\exp(\mathrm{i}kR)}{4\pi R}\left[\left(\bar{\mathbf{I}} - \frac{\mathbf{R}\otimes\mathbf{R}}{R^2}\right) + \frac{\mathrm{i}kR - 1}{k^2R^2}\left(\bar{\mathbf{I}} - 3\frac{\mathbf{R}\otimes\mathbf{R}}{R^2}\right)\right]$$
$$\boldsymbol{R} = \mathbf{r} - \mathbf{r}'$$

As a side note, the ADDA manual and presentation given by Yurkin (top and bottom equations respectively) give the Green's tensor in non-CGS and CGS forms, so be wary when selecting one to use.

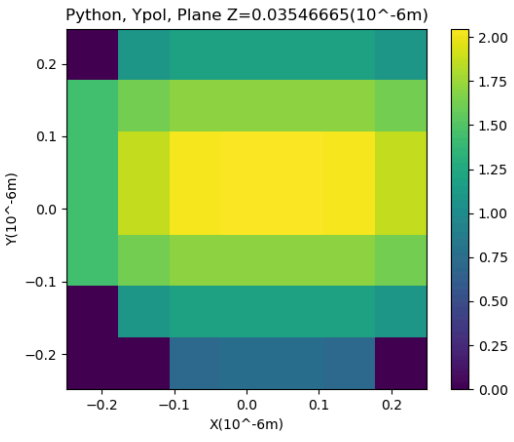
The ADDA manual explains that the saved values are dimensionless, hence I would need to be de-dimensionalised my result to match the E field produced by my python version, hence I need to find the factor required for this.

> Symbol used in the first row for column labels is different for different fields. It is "E" for the internal field, "P" for the dipole polarization, and "Einc" for the incident field. Electric fields are considered relative to the amplitude of the incident field, i.e. the values in the files are effectively dimensionless. The unit of the dipole polarizations is then the same as of volume (μm³).

When using the conversion mentioned between CGS and SI units for the E term in the force calculation (overall is a *4*pi*permittivity*) for the section of the calculation that is different to ADDA's expected results (just the scattering part, not the beam incident section), then I get a slightly different pattern that much better resembles the ADDA result, and that has almost the exact right magnitude (appears to peak at roughly 3 dimensionless units, rather than the 2 ish seen with ADDA). This is a big step in the right direction, however there still seems to be a small difference in the factor required compared to what has been tested, however this is definitely a close enough match to new test the force calculation again, where I will likely need to

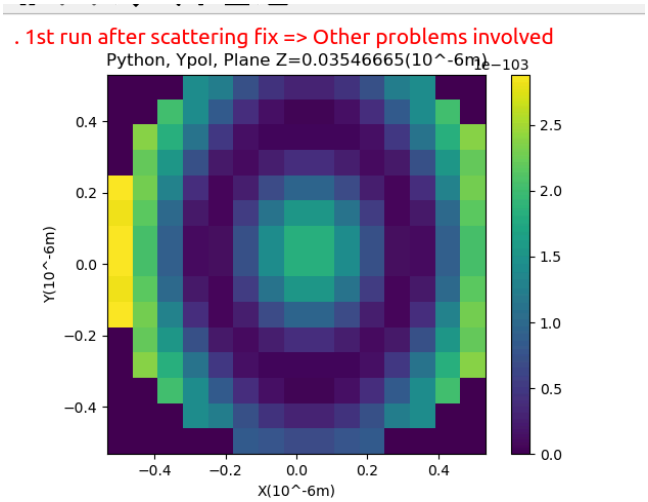


> plane beams (§9.2), $\mathbf{E}_0$ and I_0 are considered in the focal point of a beam. The formula connecting w_{rad} and $\mathbf{E}_0$ is given for Gaussian-CGS system of units, which is used throughout ADDA. In SI $1/(8\pi)$ should be replaced by $\varepsilon_0/2$. However, the relation between w_{rad} and I_0 is the same for both systems. It is important to note, that Eq. (80) explicitly contains the refractive index of the surrounding medium m_0. Hence, $\mathbf{C}_{\text{pr}}$ calculated for equivalent scattering problem of particle in the vacuum (§4) is the



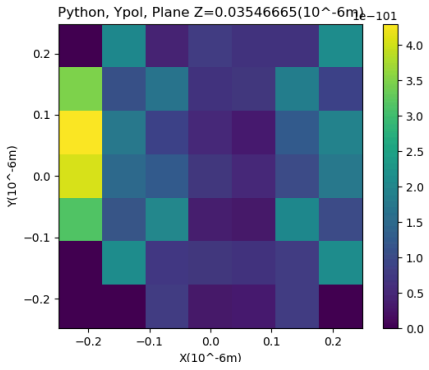
apply this conversion again (as the force equations will not account for this). Similarly, the 8x8x8 sphere agrees much better now too (1.2 peak vs 2.0 peak with matching patterns).

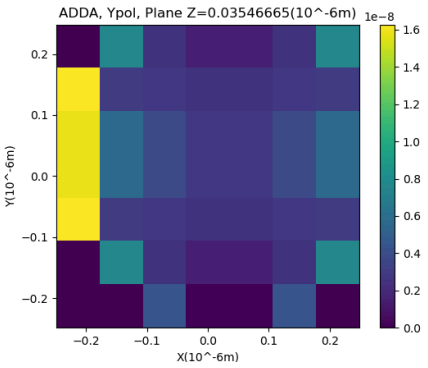
Running the forces immediately getting this scattering fix working, this is the force calculation produced, hence other problems must be present too.

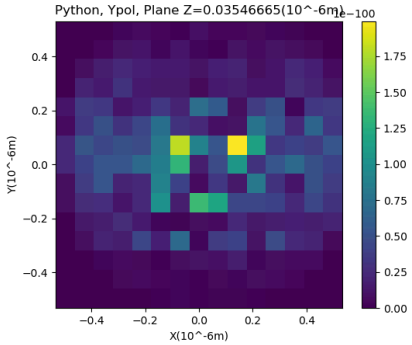


Looking at the values given, the polarisability seems to have a large real part but approximately correct imaginary part (SH_DDA had value ~10^-33), and the polarisation looks correct (pulled straight from data), but the gradient looks extremely low and far more importantly it has 7 arguments, not the 3 expected, as I had not adjusted this section code since combining the positions into the total near field calculation. With this fixed, I get the figures on the right for the 8x8x8 case, which have very similar patterns, and a highly offset magnitude. For the 16x16x16 case a very noisy result is produced, but this could be due to the large magnitudes involved, perhaps overwhelming the pattern we would expect from beam contributions?

```
===
    polarisability =  (9.289981477856267e-16+1.1848221328832673e-28j)
    E_conj_gradient =  [ 0.00000000e+00+0.00000000e+00j  0.00000000e+00+0.00000000e+00j
 4.93038066e-32+0.00000000e+00j  4.68386162e-31+0.00000000e+00j
-3.32800694e-31+8.01186857e-32j  1.25185446e-32-1.15555797e-32j
-1.92592994e-31-2.70400564e-31j]
    dip_polarisations[dip_ind] =  [-2.12413170e-05+3.53520242e-05j -1.84972186e-07+1.13414863e-06j
 2.43822434e-06+9.07648907e-07j]
    dip_force[3] =  (2.3977886335972185e-103+0j)
```

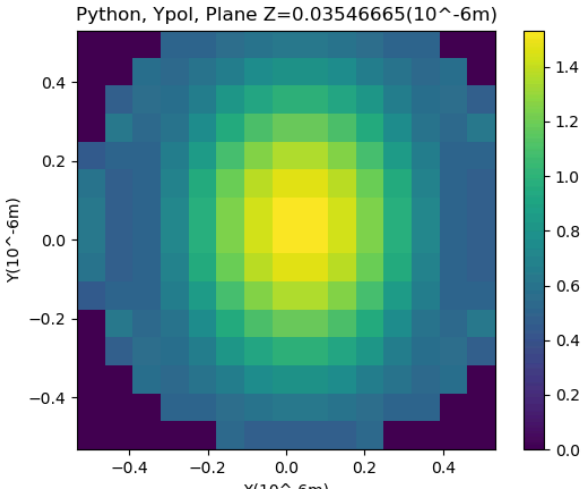




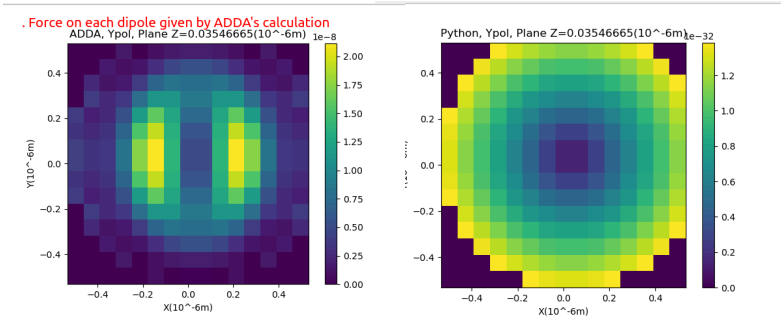


2/1/25

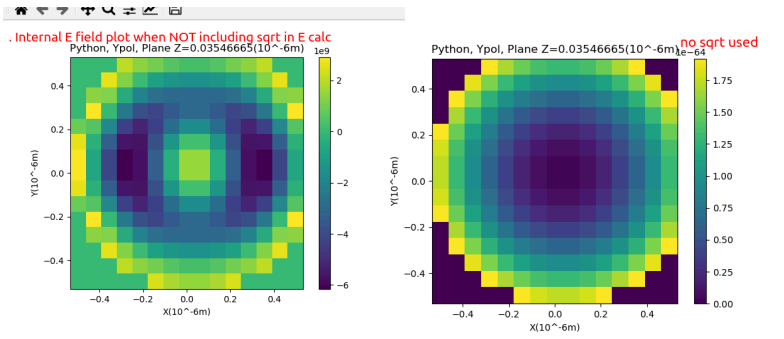
Looking again at ADDA for a bit, I switched the internal force calculation to be found on an arbitrary point, not just dipole points, which now allows the gradient to be found through a central difference method, hopefully allowing the forces to be found to be found without issue. The right shows the internal E field found through this method, which has been rescaled by 10^-10 for now to account for the unit scale issues I was having, which is temporary for now to allow the forces to be tested (this is still slightly off from ADDA's internal field, but should be close enough for the force calculation to be tested).



I have been rewriting the force calculation, still according to the paper ***"Time-averaged total force on a dipolar sphere in an electromagnetic field",*** but with extra care to have all the sources of electric field accounted for and with the units from the previous calculation scaled to not throw off this calculation. Currently, I am getting forces of the order of 10^-63, all equal values.



After changing the equation to use the E field * the E gradient (not both E gradient) I get the following plot, which doesn't have equal values across the entire shape which is better than before, however the pattern is still not correct, which may be a ***np.sqrt*** issue when plotting (e..g plotting |E|^2 vs |E|). When plotting without a square root, I get the figure below …. The scale is also off what is to be expected too (expect order 10^-8, got order 10^-32), however I am not too concerned about this as the internal field calculation was similarly unscaled due to what I believe is unit problems (ADDA often used micrometers and CGS units, the



$$\alpha_{\mathrm{LDR}} = \frac{\alpha^{(0)}}{1 + (\alpha^{(0)}/d^3)[(b_1 + m^2 b_2 + m^2 b_3 S)(kd)^2 - (2/3)i(kd)^3]},$$

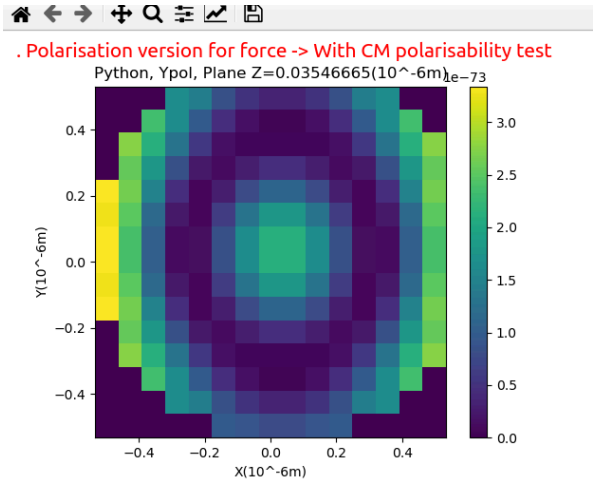
$$\alpha_0 = a^3 \frac{\epsilon - 1}{\epsilon + 2},$$

equations being used do not use this scheme, and also the wavelength used in in terms of micrometers) and was also off by ~20 orders of magnitude, as roughly seen here, hence when I fix this for either case I believe the issue will be solved for both. Also, for simplicity I have been using the form of polarisability given in the paper referencing the time-averaged force equations (purely Clausius-Mossotti), however ADDA is technically using the LDR formulation, which is different and so could result in a significantly different pattern being observed, however from the papers I have read discussing their differences it seemed that the differences were quite small and preferred the option of testing with a simple case to reduce errors to mainly be from the force calculation itself, rather than an misunderstanding / input of constants for the polarisation used. However if the pattern continues to not match I will also try the LDR version.

$$F_u(\mathbf{r}_i) = \sum_{v=1}^{3} \mathrm{Re}\left(p_v(\mathbf{r}_i) \frac{\partial E_v^*(\mathbf{r}_i)}{\partial u} \right),$$

Also, considering the implementation used in DDA, the formula on the right seems to match it best (dipole moment * E gradient), which is from the paper ***"Review; The Discrete Dipole Approximation: A Review" by Patrick Chaumet***, so it would be good to use the dipole moment explicitly here, in opposition to the other formula used.

When testing the LDR formulation the result was essentially the same as when just the CM relation was used (as I had read before).



WAS NOT USING POLARISABILITY CORRECTLY, HOWEVER RESULT WAS STILL ESSENTIALLY THE SAME AS EXPECTED

THE HOPED FOR CASE FOR THE FORCES DID LOOK LIKE THE INVERTED VERSION OF THE INTERNAL FIELD /W / /WO A SQRT → HENCE SQRT IS PROBABLY GOOD TO LOOK AT

+ COMPARE MORE TO SIMON'S VERSION

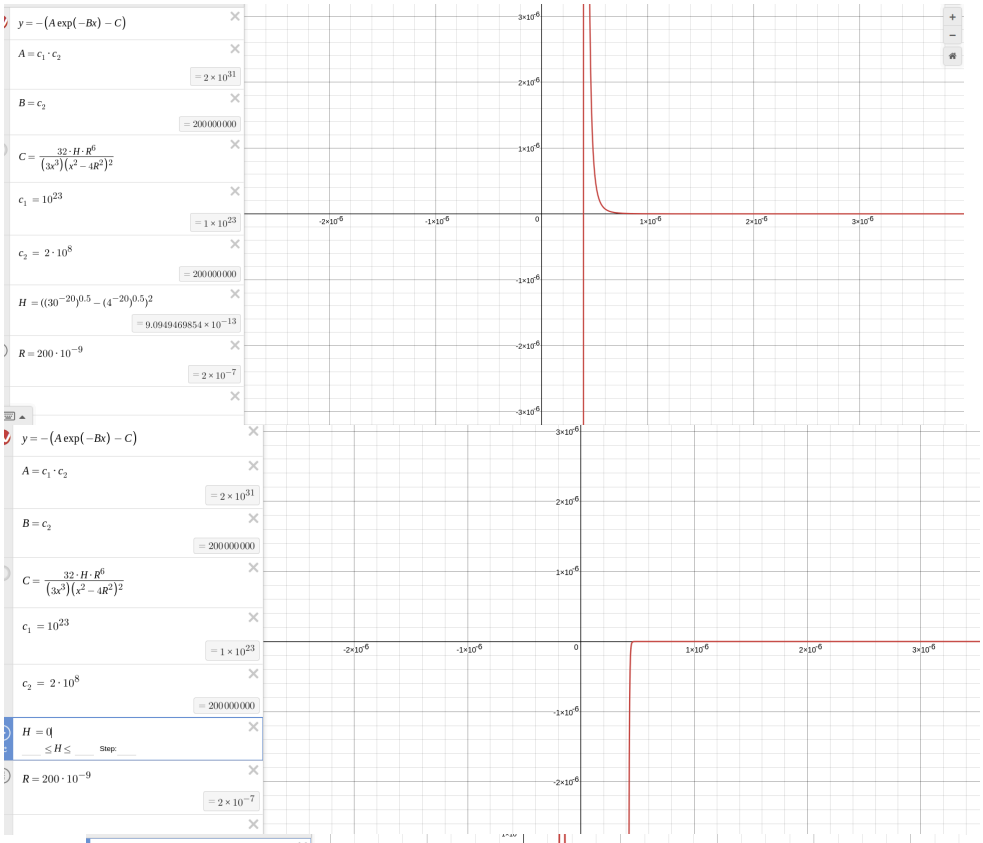
1/1/25

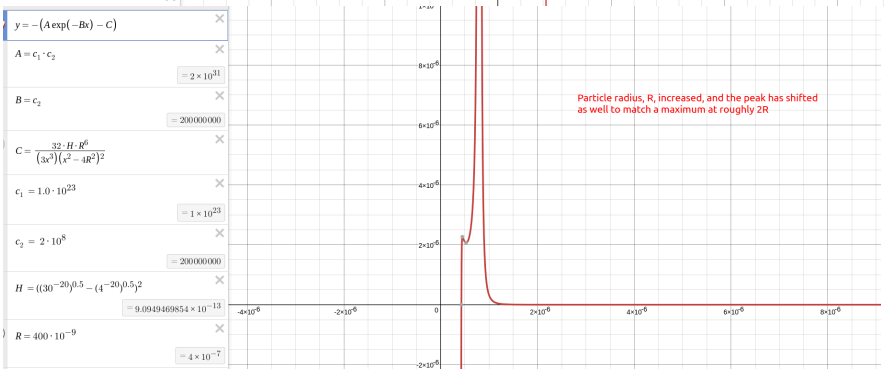
I would now like to try to fix the Buckingham force scaling so we can have particles much closer together. The formulation provided for this Buckingham force is different to the version I had found, mainly in that its second term scaled roughly with ***r^-5 not r^-6***. I believe this is from the '***Hamaker'*** term included which is supposed to account for inter-particle molecular forces when there is a dielectric medium between the two particles. Below shows the force profiles for the two Hamaker values given in the program (with and without, H=Const and H=0). We see with its inclusion there is a strongly repulsive force at 2R (R=particle radius), e.g. when the particles are touching. Closer than this results in the force becoming attractive, however in the program in the distance between the particles is less than 1.1*2R then the distance is fixed to this value and the force evaluated. Obviously when H=0 then the second term is removed and only this exponential is left, which when the negative is taken gives a strongly attractive force at 2R instead. To get the force vector this value is multiplied by each component of the

$$\Phi_{12}(r) = A\exp(-Br) - \frac{C}{r^6}$$

```
force = np.array(
    [
        -(
            constant1 * constant2 * np.exp(-constant2 * r_abs)
            - (
                (32 * Hamaker * (radius_avg ** 6))
                / (3 * (r_abs ** 3) * (r_abs ** 2 - 4 * (radius_avg ** 2)) ** 2)
            )
        )
        * (r[i] / r_abs)
        for i in range(3)
    ]
)
```

Original Buckingham force



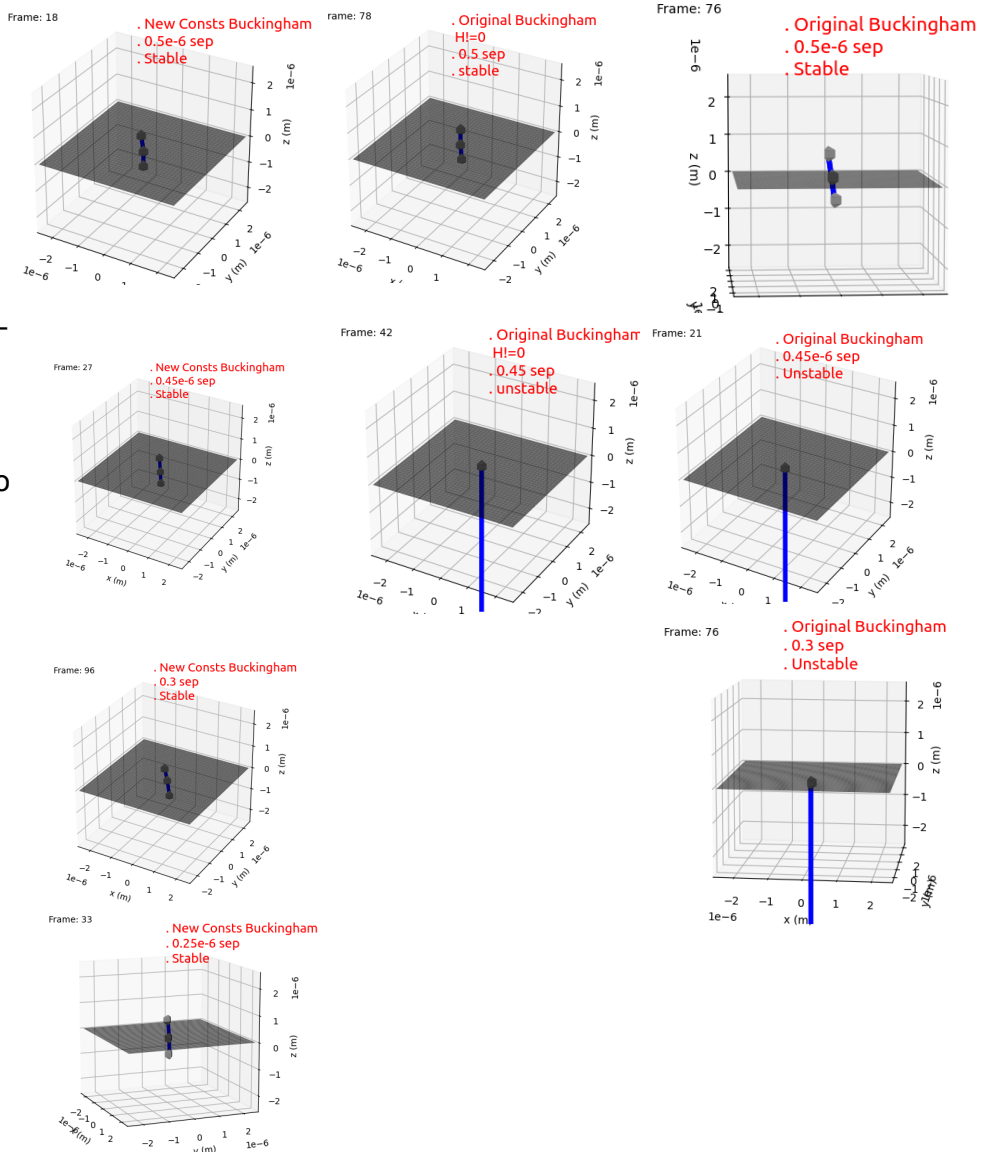


displacement vector between the two particles, which will give an overall repulsive force from the difference vector direction taken, hence for the Hamaker != 0 case we would actually have an attractive force which then becomes repulsive (and for H=0 is purely repulsive). It is important to note that these functions are not tending to +-infinity at 2R, but have had constants chosen such that their growth occurs at roughly 2R. This being said, when H=0 there is no R dependence, hence for smaller or larger particles, they will still feel the same string force at 2*(200e-9) distance anyway, which for larger particles means the Buckingham may not fully push the particle away before volume overlap occurs. The program tries to mitigate this by including a difference in the ***r_abs*** value and the radius (so distance are measured from the surface not the centres of the particles), however this is not a factor when H=0, which is the regime we have been dealing with thus far.

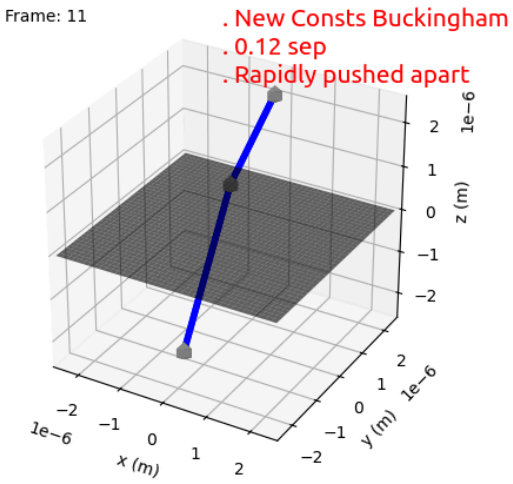
With this understanding, it can be seen that the force (with H=0) can be made to grow quickly at smaller separations by reducing the constant C1 or increasing C2. Also, by decreasing C2 we see a shallower curve (meaning a more gentle force as you approach the peak).

Considering the behaviour of close particles for an original regime (C1=1e23, C2=2e8, H=0 or

H=(np.sqrt(30e-20) - np.sqrt(4e-20))**2) and a new version (with C1*=1e-34, C2*=2e-1, chosen to make the curve shallower and have growth come into effect at closer distances). From this we see the new version is more stable at small

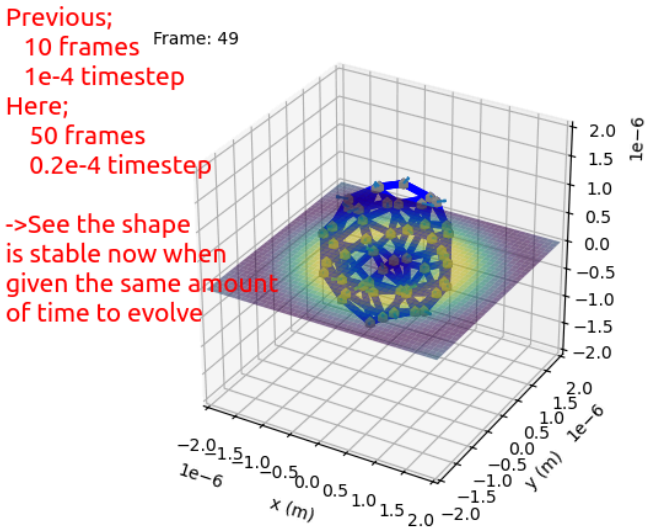


Distances, even when the Hamaker term is included in the original. We also see that at even smaller separations, the new formulation does still push particles apart, which occurs sharply here, but perhaps could be reduced in sharpness through smaller timesteps.



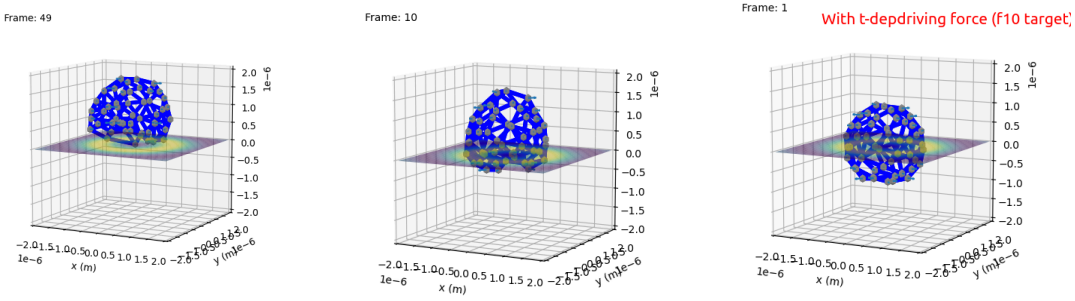
31/12/24

Using smaller timesteps as suggested yesterday has resulted in a much more stable shape when the animation is viewed as it has far less extreme deformation (and did not explode, which the other usually did at the 10th frame).



Now to better observe how the sphere deforms I modified the driver force to be frame dependent, where the force is applied only for frames less than some target.

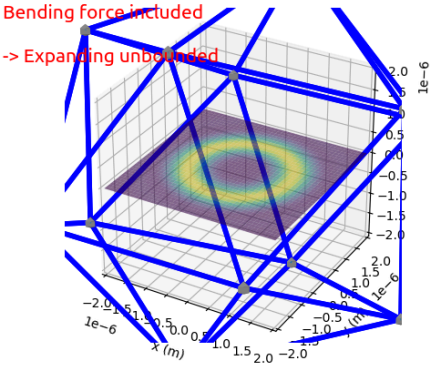
When I simulate this, I see the shape begin as sphere, be offset into a more ellipsoid shape within the first 10 frames (force only applied for first 10 frames, magnitude 5e-12 in the Z axis only), then be bent back to spherical shape afterwards (observed as spherical at frame 50, but reached this point before the 50th frame). The rate of this transition (both parts) will be dependent on the bending and spring constants. Optical effects were also present here, but did not affect the particles as strongly as the driving force chosen. It would now be interesting to see how this affects a much larger surface (force present on a small section) to see if ripples will occur, and also to find a beam type which can perform this deformation to bring the simulation back to the real scenario of a physical beam of light causing this deformation. I also would like to focus more on having particles closer together in order to take advantage of the accuracy of the situation in these situations that we proved earlier with the Laguerre beam (this would likely require the new scaled hydrodynamic force)

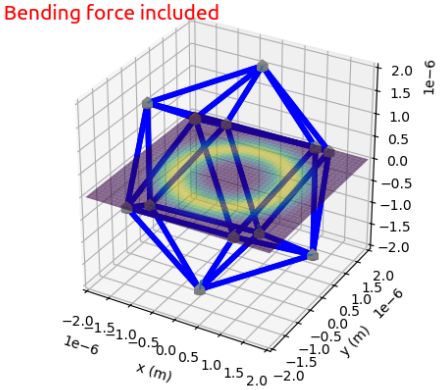


. Scale hydrodynamic force
. Bring particles closer
. Find beam to cause this deformation
. Observe if ripples (or another pattern) occurs on the surface of a sphere (or plane as a simplified paraxial-like case)

30/12/24

My partner and I attempted to combine the bending force with the spring force and saw an equilibrium slightly bigger than the original shape was formed when tested on an icosahedron. When the bending force was tested without the spring force,



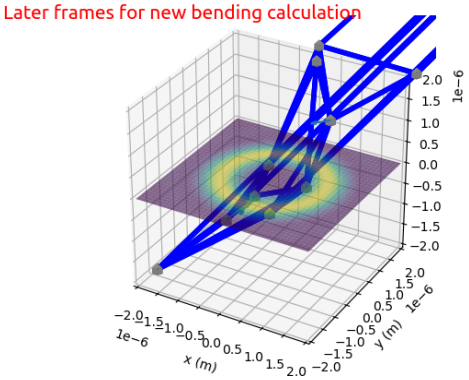


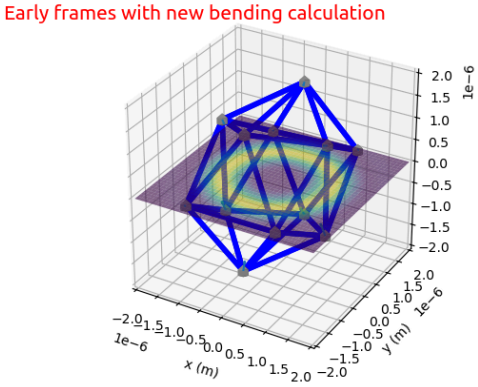
unbounded expansion is seen, which after some discussion we found to be from a mistake in the generalisation of the bending force from always having an equilibrium at angle 0 radians to any target angle.

To fix this, we used the working bending force that assumed particles sat in a line (equilibrium angle=0), then rotated one of the vectors to be offset by the bending force to give the desired target location. I have implemented this in the program by using the ***Rodrigues rotation formula*** to rotate by this equilibrium angle, but just in the plane that the 3 points (involved in the bending, forming a plane) are involved in.

$$\mathbf{v}_{\text{rot}} = \mathbf{v}\cos\theta + (\mathbf{k}\times\mathbf{v})\sin\theta + \mathbf{k}\,(\mathbf{k}\cdot\mathbf{v})(1-\cos\theta)\,.$$

Having used this formula, the behaviour of the points is slightly better (not massive expansion, but a slow crushing which eventually explodes).



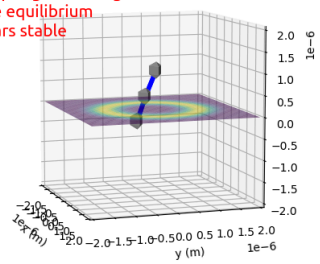


It is quite hard to debug a complex system like this, so I will instead test the bending on a 3 particle system at different equilibrium angles. My partner did manage to get a stable looking

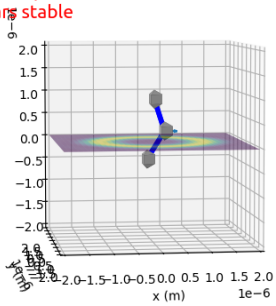
shape by changing some of the signs involved. But I would still like to test simpler cases to ensure everything works fully as expected.

When a 3 particle case is tested, various equilibrium angles were used and all appeared to work as expected.

Forces; Spring + Bending
0 degree equilibrium
-> Appears stable



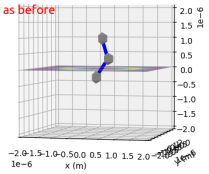
Forces; Spring + Bending
45 degree equilibrium centre
-> Appears stable



Now that the bending force has shown itself to be reliable, I want to test it for more complex shapes like spheres and see how it deforms, which will hopefully imitate a solid sphere when connected around a sphere (given the bending constant is chosen large enough).

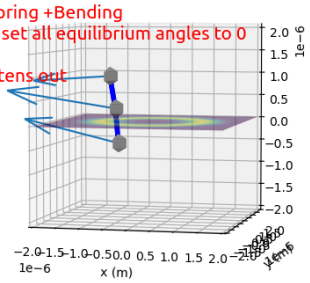
Frame 50
BendingConst = 1e-20 not the previous 1e-18

-> Weaker hence we see it doesnt shift to the flat state as before

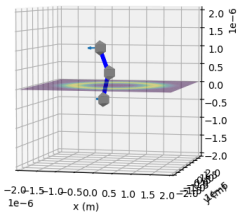


Frame ~25
Forces; Spring +Bending
Manually set all equilibrium angles to 0

->Straightens out

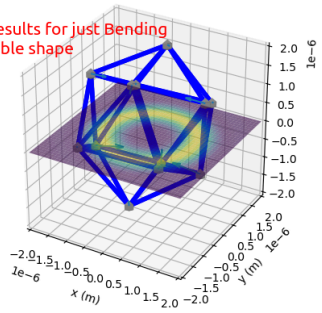


Forces; Spring + Bending
Manually set equilibrium angle to 0
Starting at 45 degree angle arms



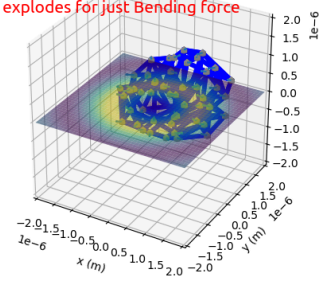
Forces; Bending+Spring+Optical

-> Similar results for just Bending
-> See a stable shape



When initially testing a sphere shell with this improved bending force (1e-18 ‘BENDING’ constant), however the system still breaks. I should try this with a re-scaled Buckingham force and at smaller time steps.

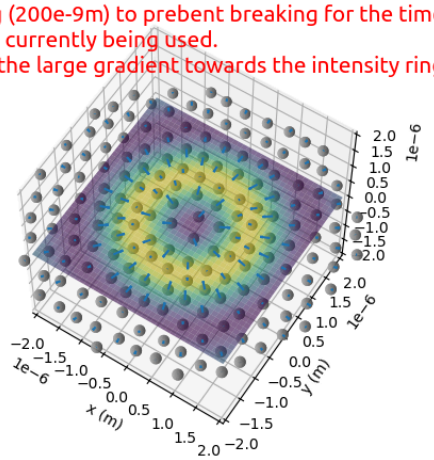
50 points in sphere shell
Connections to adjacent points
-> Bending + Spring + Optical still resulting in exploding
-> Also explodes for just Bending force



29/12/24

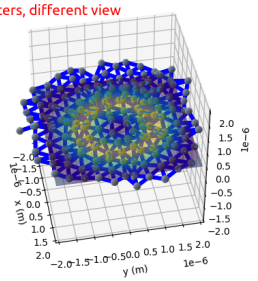
Running the large mesh of spheres (bounding radius of 3e-6m) the simulation exploded within approximately 2 or 3 frames because of particles being driven by a large force from the E field gradient (towards the high intensity ring) + their brownian motion + the spring forces between particles results in the particles overlapping and breaking (this is also due to the Buckingham force having to be removed, allowing situations like this more easily).

More spacing (200e-9m) to prebent breaking for the time step (order 10^-4) currently being used.
Can also see the large gradient towards the intensity ring here

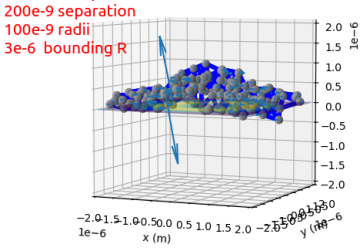


Implementing some code found by my partner I wrote a plotter for spheres of a given size placed on the surface of a sphere of given radius. These obey the natural length and stiffness rules implemented, and so we see a stable shape when just the spring force is included. When the driving force was included too, a similar effect to before is seen where the shape is dragged in the Z-axis by this force and its shape is roughly maintained through the spring interaction.

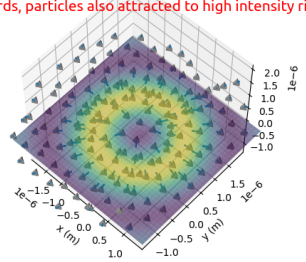
~30 frames into simualtion
Same parameters, different view



~30 timesteps into simulation
200e-9 separation
100e-9 radii
3e-6 bounding R



frame 50, NOT displaying connections (still connected)
Edges are drawn inwards, central particles drawn upwards, particles also attracted to high intensity ring



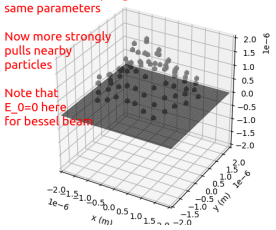
Currently when I try to use the bending and spring modifications (generalisations for any connections) together, I get problems with the linear equation solver in Eigen, which is usually a result of particles overlapping too much or exploding. I need to investigate why this occuring by doing smaller incremental tests.

I also should try and scale the hydrodynamic force differently, which may help prevent this linear eq. solver error. When these are all working I should be able to create a sphere shell (which will act rigid through the bending forces included), which I can then deform with a beam or manual motion with the driver (time dependent this time, so I can trigger some initial motion and see how it evolves on its own after that) to observe a flexible sphere being deformed.

Forces; Driver + Spring
same parameters

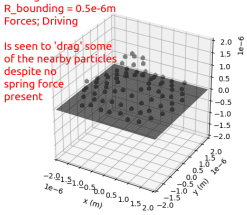
Now more strongly pulls nearby particles

Note that E_0=0 here for bessel beam

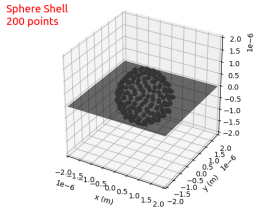


Driving force 5e-12 in Z axis
R_bounding = 0.5e-6m
Forces; Driving

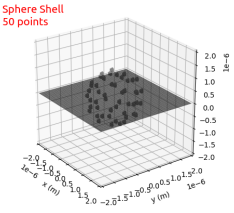
Is seen to 'drag' some of the nearby particles despite no spring force present



Sphere Shell
200 points

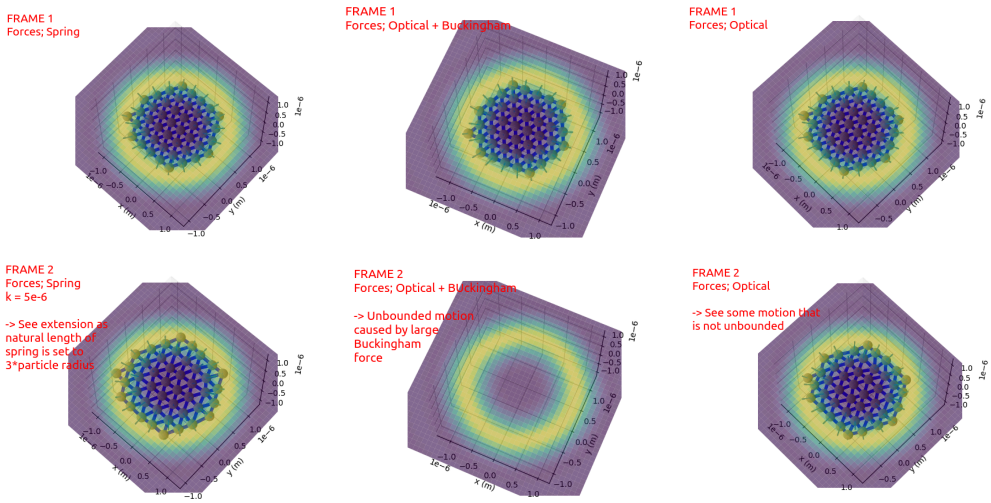


Sphere Shell
50 points



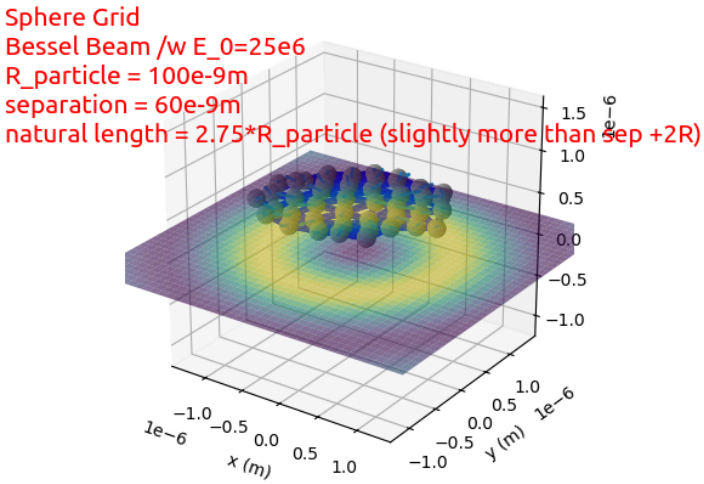
I need to start looking at how the different regimes for the momentum of light (Abraham and Minkowski) could be tested separately here too with their individual momentum density fields. Stephen Barnett argued that both should be present for an accurate description, but one will be dominant in different scenarios, and another paper argued that no specific consideration was needed to observe these effects too (demonstrated in an example where light enter a block of refractive material was carefully followed and analysed in terms of classical forces).

28/12/24



Considering the dynamics of this sphere grid, we see that the spring force works as expected (and would require the modification that natural lengths are individually set to the initial separations in order to have springs that try to preserve this original shape). Note as well that all these cases tested above also have Brownian motion every step as well. We could modify the Buckingham force to scale with the initial distances in order to prevent this huge force for the tight distances required for DDA to accurately model larger particles.

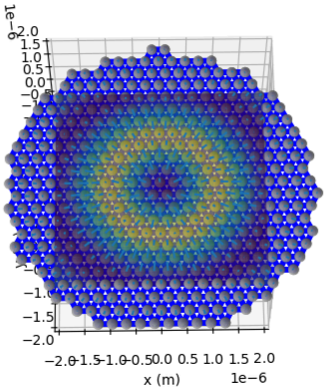
Now I would like to deform just the centre of this mesh and see how the other particles respond with just spring and optical forces (Buckingham removed for now until it is scaled down appropriately), without bending forces too for now. I will apply a Bessel beam to the mesh to give this axial force (Z axis).



When performed (seen on the right) the entire mesh moves as the beam width is large (a large E_0 was chosen in order to see any effect from this beam, this may be some normalisation issue, however the beam itself is irrelevant, we just want to cause some motion and see if a stable reaction is produced). I can fix this in a few ways, first I could make the beam width smaller so it only interactions the central few particles (however, we briefly experimented with the beam width argument in the YAML file ‘w0’ did not affect the bessel beam as this beam had no w0 scaling, however this is likely an error with the beam’s implementation and may be fixed fairly easily). Another fix would be to have a larger grid of particles so they extend further beyond the high intensity ring, however this would be bad for performance if made too large unless I made the dipole size much larger so fewer dipoles are used per particle (this seems like the most reliable solution). Another solution would be to ignore the Bessel beam and just apply a driving

force to the central particles to move them manually, then we can observe the motion after (this however is not quite in the spirit of the measurements we want in the future, as there would be no optical interactions, and so the inter-particle optical effects would not be considered at all).

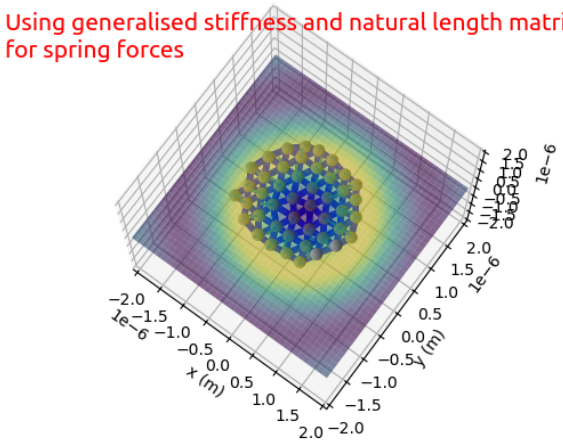
Sphere Grid /w bounding radius=3e-6m



When performing this calculation, I initially saw the system become very unstable and explode after just a few frames.

I also would like to generalise the natural length and stiffness coefficient implementation to allow each connection to have a unique stiffness and natural length (in preparation for more complex shapes). This should not be too difficult to do as a matrix for each of these parameters can be made to state the parameter's value from one particle index on another (would would also have the added benefit of allowing directional springs e.g. not symmetric for a on b and b on a, which is not physically accurate but perhaps could have some uses at some point). These matrices would be generated from an initial copy of the structure of particles (and would work similarly for setting a bending equilibrium angle too).

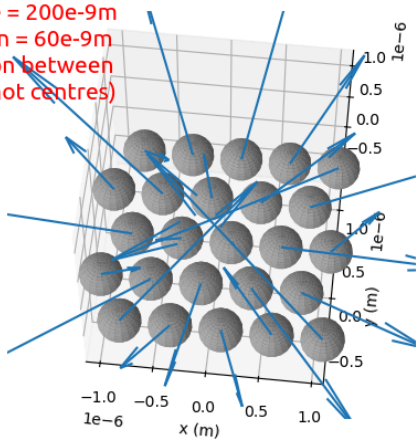
Using generalised stiffness and natural length matrix for spring forces



27/12/24

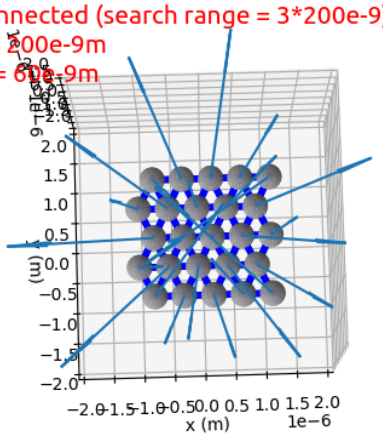
Now that the spring forces have been seen to work as expected I can start testing them in larger systems and for different particle sizes (200e-9m not needed specifically anymore as Laguerre trapping is not necessarily required). First I want to observe the motion of a flat web of connected particles in the presence of a Bessel beam interacting with just the particles in the centre (see how the rest of the web responds to this deformation, to see if perhaps some ripple effect occurs as maybe expected). After this I want to try the same situation for a connected shell of particles on a sphere surface and assess its validity. I am expecting the particle to collapse once this occurs (essentially losing volume, which is not wanted) as there is no restoring force to prevent this continual collapse, and so I would also like to connect some/all of the particles with a natural length of the separation between each of those particle sets, so any compression on the surface will push back (which will hopefully give rise to some ripple effect).

Flat sphere mesh
R_particle = 200e-9m
Separation = 60e-9m
(separation between surfaces not centres)

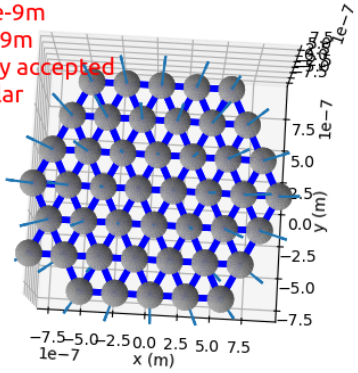


For the bending force, it is currently only implemented for a line of particles (2 connections per particle exactly), so for our general connection version we will instead find all the connections for a given particle, find all the bending forces for each particle from this set in groups of 3 (e.g. if particle 0 has 4 connections, it will have 4 combinations of sets of 3 particles to consider, which can be summed for a total effect). This can be repeated for all particles to get a bending force for each particle. Note that this bending force will consider the angle subtended by these 3 points (2 targets and the origin particle) relative to some base angle desired (found from the original system setup specified), and similarly the natural lengths of the springs can be pulled from this original system setup too. This approach is more complicated than is required for shapes like spheres, platonic solids, etc, but will allow less symmetric shapes like a red-blood cell to work.

Sphere Grid
Particles connected (search range = 3*200e-9)
R_particle = 200e-9m
separation = 60e-9m



Sphere Grid
connection range = (3*100e-9)
R_particle = 100e-9m
separation= 60e-9m
Particles also only accepted if on within circular bounding box

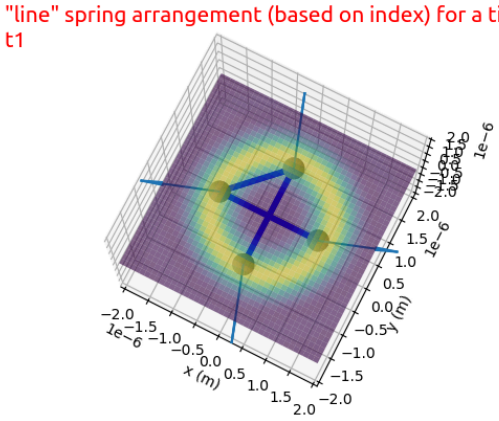


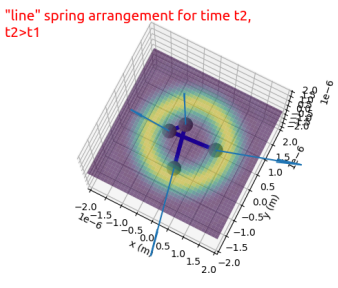
26/12/24

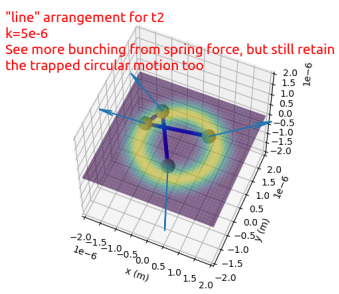
My plan for today is to go back to the SH_DDA work for a bit to start understanding how the current implementation of the spring force works and assess if it needs to be reworked to apply to the set of cases we will want to experiment with (e.g. any combination of particles springed with any of set of particles). After this I plan on checking the force calculation in my Python version against the ADDA calculation for a plane wave. If I have time after this I would then like to check the incident beam on each dipole in ADDA and see if the Laguerre Gaussian beam is working as hoped (which appeared to work correctly when transferring the plane wave between Python and ADDA, so hopefully the Laguerre equation which was transferred in the same way should have few problems to resolve).

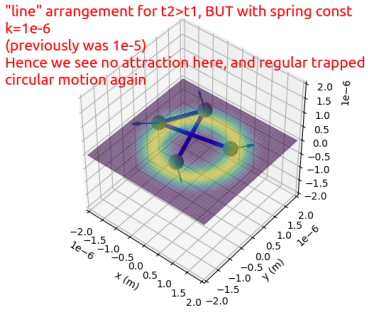
Looking at the original implementation of the spring forces, particles were generated and optical forces found as usual, then the spring force was found afterwards from a set of particle centre positions parsed in (seemingly mislabelled as dipole positions from older versions of code which dealt with particles that were just 1 dipole). This process had not been altered when we refactored the code to generalised particle version (more than just spheres, and can vary size). A matrix of forces is then found by pairing up particles and calculating the force between them, which appears to connect particles in a line as the force between adjacent particles are found (e.g. components near matrix diagonal found only). The total force on each particle is then found through a sum along ***"axis=1"***. We would want this to be changed to allow any particles (not just adjacent in index) to be linked.

Considering the changes my partner has already made to this method, the main difference is now there is a function ***generate_connection_indices()*** which accepts the particle center positions and type of connections wanted as arguments. This then returns the indices of groups of particles (where each group is the link between particles). Currently the particles can be linked in a line according to the index (as with the previous version), linked by the N closest particles or linked to every other particle within a set distance. The old dipole naming convention was replaced here to reflect the new approach. The radius of each particle still needs to be taken into account (currently just uses 1st effective





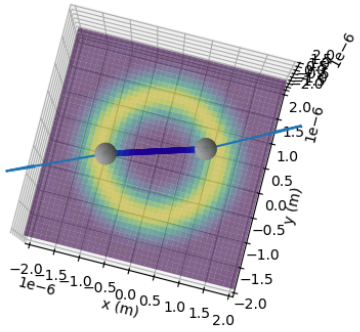




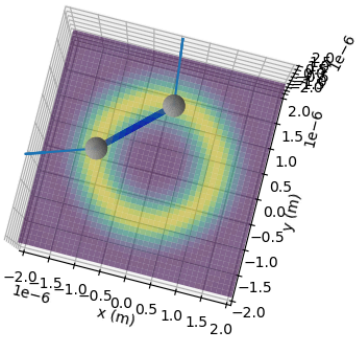
radii -> assuming all are the same) which is used to find the ***spring_force()***. A bending force is also found here however the ***BENDING*** constant here is set to 0 by default. This force considers the relative position of a target particle and the adjacent (in index) particles, populates a matrix with forces found from this, then returns the matrix. This means that bending force will need to be modified to match the connection indices chosen.

When plotting the connections between these particles using the “line” arrangement (the same as was originally implemented in the program) and time stepped for 10 frames including optical, buckingham and spring forces (no bending), we get the above plots clearly showing the additional force is working as intended.

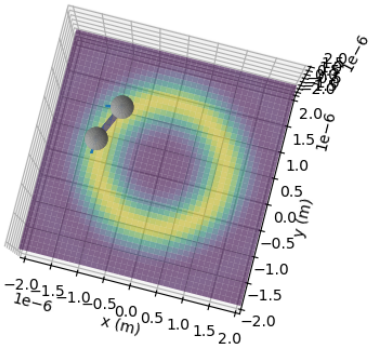
Springed particles; t1, k=5e-6



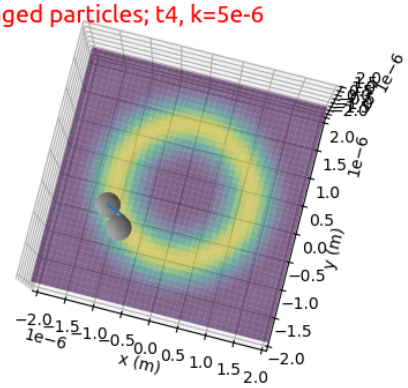
Springed particles; t2, k=5e-6



Connected particles; t3, k=5e-6

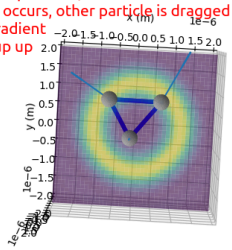


Springed particles; t4, k=5e-6



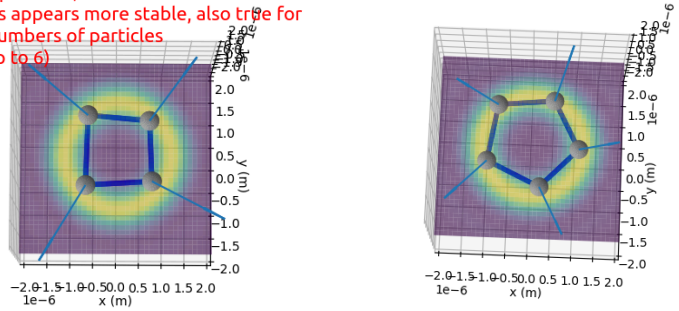
When considering a connection between adjacent particles in a ring, we continue to see circular motion but also the particles being brought together as would be expected (when the spring constant is deliberately chosen to not be overpowering or overly mild, e.g.k=5e-6). This shows that care needs to be taken with the spring constant chosen in or else the particles may be given enough force to jump the section of the Laguerre ring (e.g. overcome the electric field gradient in whatever shaped field that have been placed in, worth considering if you are trying to constrain the

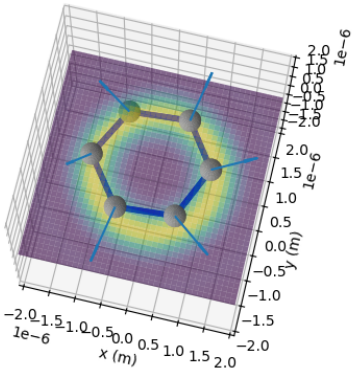
Springed particles; k=5e-6
Pairing occurs, other particle is dragged through field gradient to group up as well



particle). It is worth noting that the systems tested here appeared to be much more stable than their non-spring counterparts in terms of keeping particles trapped and evenly spaced (tested up to 100 frames, remained stable throughout).

Springed particles; k=5e-6
4 Particles appears more stable, also true for greater numbers of particles (tested up to 6)

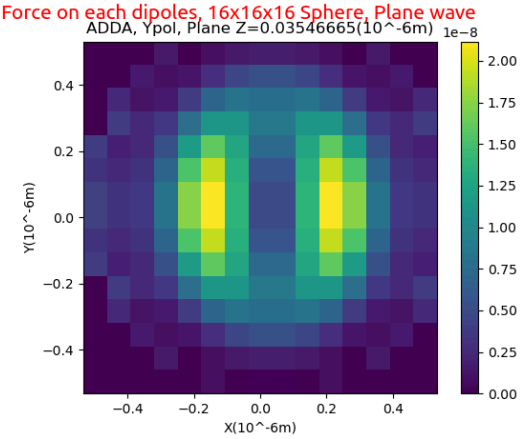




For ADDA I have also realised now that I cannot recompile the ADDA code anymore due to installing g++14 (was previously using g++11), as I was unable to install ***gfortran 14*** alongside it, hence i get the error that ***"lgfortran" cannot be found***. This means I cannot run the laguerre ADDA code until I have fixed this problem.

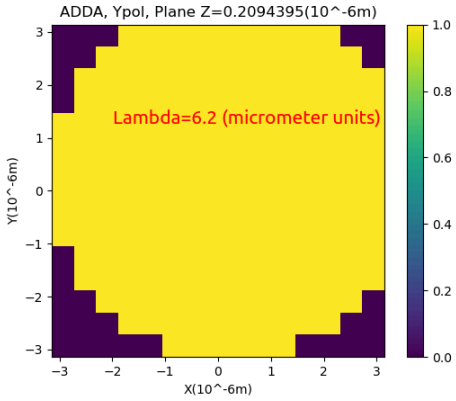
For the force plot, a plot from the ADDA data but the Python calculation is currently not working. To investigate this, I will look at each contribution to the force and see what appears unreasonably small / large, then investigate those further.

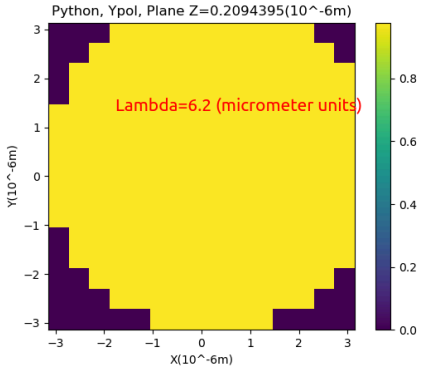
Force on each dipoles, 16x16x16 Sphere, Plane wave



24/12/24

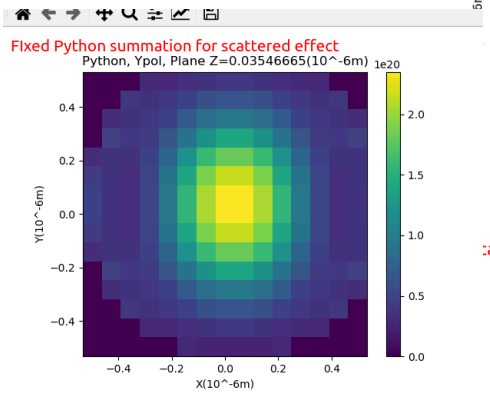
The ADDA data collected yesterday did not show a match between the ADDA and Python calculations, which today I realised was due to the wrong store from ADDA being called (previously called ***"store_int_field"***, now I am using ***"store_beam"***), which was giving the total internal (scattered + beam_incident) not just the beam_incident field (which store_beam does), and so now we see a match in the calculations.



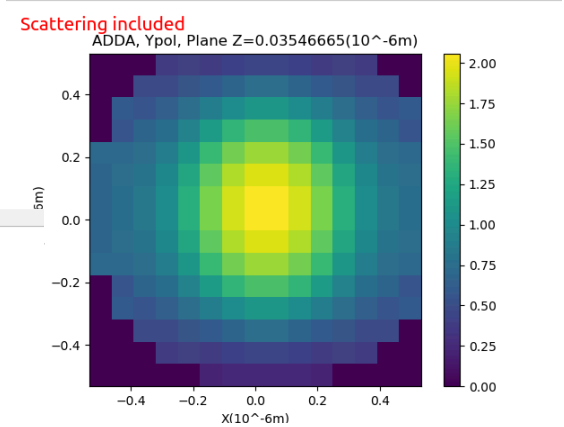


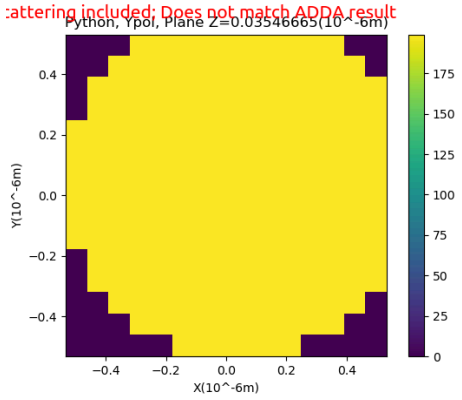
This is showing that the equations for the raw field at each point matches now, and so now I will attempt the Python scattering calculator and see if its plot matches the previous result as it should. This calculation has required quite a bit of reworking as it was slow before due to performing the scattering calculation for each point, and did make effective use of information that hadn't changed between points (lots of wasted recalculation of positions and values for polarisation), which now performs the calculator for a full set of positions and is more efficient in calculation.

Currently the calculation is not working, showing no variance in the E field at each dipole. When testing this function, I set all the near field forces to zero but still had the values change several magnitudes equally (seen below), showing some inherent problem with the combination of the forces.

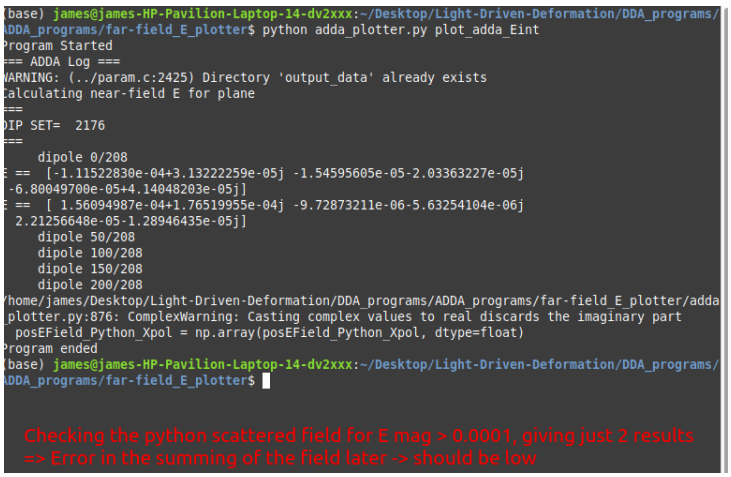


This error was due to the wrong values being summed when calculating the magnitude of the E field. When this summation was fixed we now get the correct pattern, but with a much larger magnitude of field (max at roughly





2.0e20 as opposed to true value of 2.0). I have excluded the dipole itself in the scattering calculation to try and avoid large spikes in intensity like this, hence this must be from another spiking effect, or the units supplied by ADDA already changed (however would be expecting small electric fields, not huge fields).



Looking at the scattered magnitudes directly, the values all appear to be very low (as expected) hence there is a mistake in the E mag calculation. This is definitely due to the electric field found having different units (in the beam calculation) which has not been accounted for in the scattered field. When this small value is summed over the ~2000 dipoles involved here (16x16x16 sphere) we would expected magnitudes of at most 1e-5 *2000 => values ~<1e-2, which when added to roughly 1.0 from the incident beam, you would expect orders similar to the original plot with the pattern seen (as also seen in the true ADDA calculation). The only other operation performed in this calculator is the multiplying of E by (k^2)/(epsilon_0), which would introduced a 10^-12 factor through the wavelength^2 (micrometer units), which is then later squared for a 10^-24 factor. This then leaves a 10^4 factor to be accounted for somewhere.

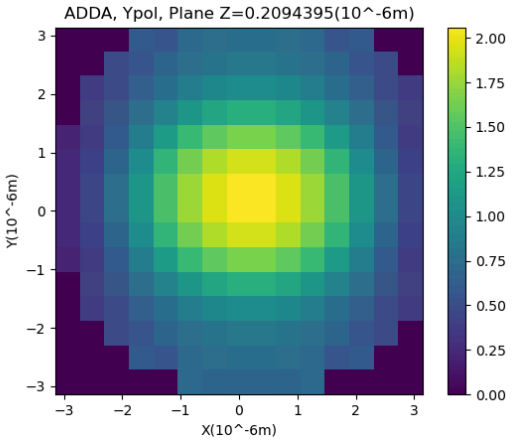
According to the ADDA manual ADDA uses units of statV/cm (CGS) for the electric field, which is approximately 3e4 V/m, however the equation I have used is perhaps already in CGS units as it was found from the ADDA manual, and produced the same raw incident field. If this were applied after the factor, this may account for the 10^4 factor missing (however in reality this situation would give a 9*10^8 factor for |E|^2 (V/m)^2). I am still unsure of the units used for Greens tensor, and as to whether they should use the micrometer units for wavelength, or be converted with a 10^-6 factor.

The scattering calculation appears to mostly be working (appears to just be a units problem affecting scaling) I will now consider the force calculation. This calculation can use ADDA data (total incident field at each dipole), and just requires the internal field near the dipole to be found for a gradient calculation (using central difference method), hence I will expect this gradient to have the wrong scale too, which I will need to account for when considering if the force is correct.

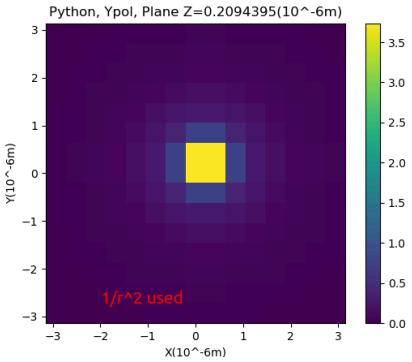
23/12/24

I plan to have a meeting with my partner later to see which problems he has tackled from the set we considered and compare to what I have done in order to see what remaining tasks must be done for this section of the research without repeating work the other has already done. Hence I will briefly look back at my ADDA code to start cleaning up its processes for easier usage and test whether the functionality I had written for it to calculate time averaged forces and far field scattering is accurate (by comparing to ADDA's values for a plane wave, which I can then similarly test for the Laguerre beam with comparison to OTT's and SH_DDA's values).

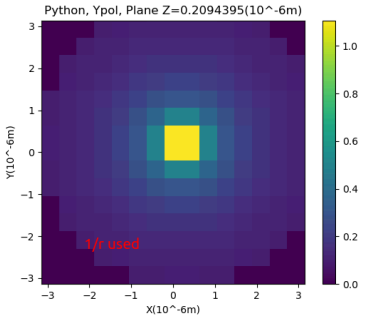
So far I have just generalised some of the data collection and plotting functions to let them be used easier and be applicable to both data generated from ADDA and data generated from pure python (equations for the field at any point I generate in python, compared to the incident fields ADDA gives at each dipole).



The plot on the right shows the new cross section plot for ADDA data generated through the (almost) extremal section of the sphere (16x16x16) with a plane wave moving in the Z axis present. This is plotting the |E|^2 of the incident field (not scattered) for a Y linearly polarised plane wave.



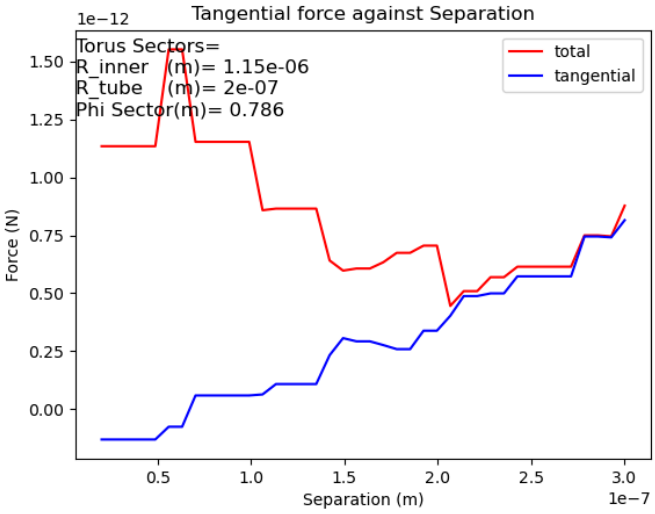
These plots show that the decay in intensity in my Python implementation is not correct, as the ADDA value shows a different incident field. The python version was evaluated at the same points as the ADDA version. This may be due to different units being assumed (ADDA assumes CGS and micrometers in several points, which may not have been accounted for here).



. Naive polar force -> show didn't work
. Then work on spring
-> See if matches red blood cell (perfect)
-> See what other work considers small flexible particles deformed through light, try to get their results

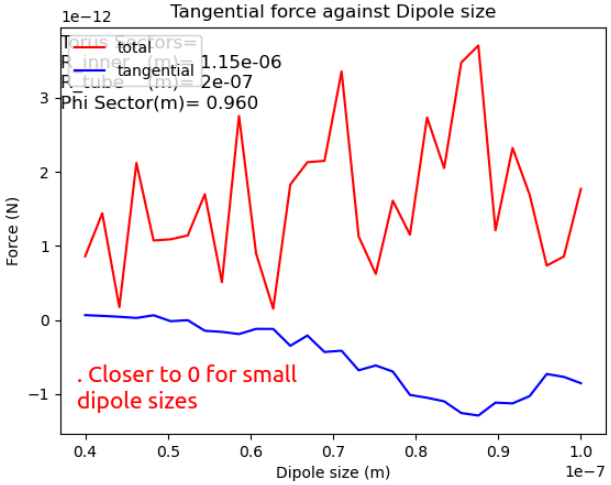
22/12/24

From the results seen yesterday the dipole size did appear to help improve the accuracy of the simulation by giving us the tangential force tending to 0 as hoped. WIth this in mind we close to the end of this first investigation of the limit in which DDA accurately models larger particles through a system of smaller particles, and so I will now get some of the plots my partner had worked on and record them here for posterity.



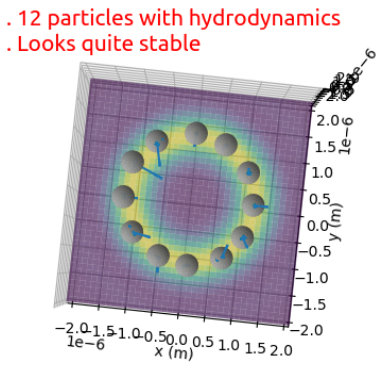
These graphs further support the ideas of force tending to 0 for small dipoles. The total force just oscillates at a larger magnitude than the tangential here, however this is not particularly of concern because this is the force on 1 particle (0th) on the first frame, and so for a torus we would expect some radial force (torus how a surface with normal in the radial direction, hence a normal force when surface integral for maxwell stress tensor is found), however for a full torus this would be cancelled by the other side of the material. Here each sector will feel the force but not cancel as they are different particles, so we expect some radial force to still be present.

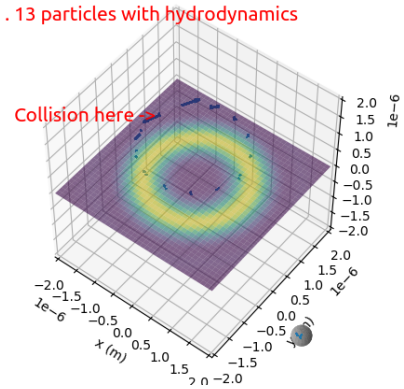
One other idea I wanted to test was why Simon had seen particles purely increase in speed as more were added, which we had observed up to 8 particles, but beyond this I believe the reason could be due to the hydrodynamic force included cause repulsion that was unphysical, and so the 0 force limit did not occur. I will test this by doing force plots with this force included.

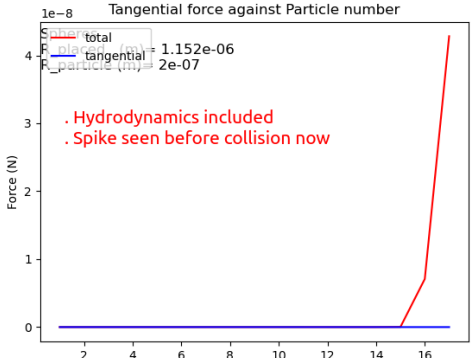


Currently the ***simulation()*** function calculates the optical forces, then additional forces, then applies this forces to the motion of the particle, but then only returns the optical force. This required me to return an addition ***totforce=total force*** parameter, which I then store in the standard excel file if forces are included (could have another argument for 'include_total_forces' but for now this is fine). This therefore requires the ***SimulationVary.py*** file to now extract the total force too if required, and adjustment was needed for the number of columns of data it expected to receive when calculating the number of particles present from the file structure. From this I could have the recorded data be the total forces instead of the purely

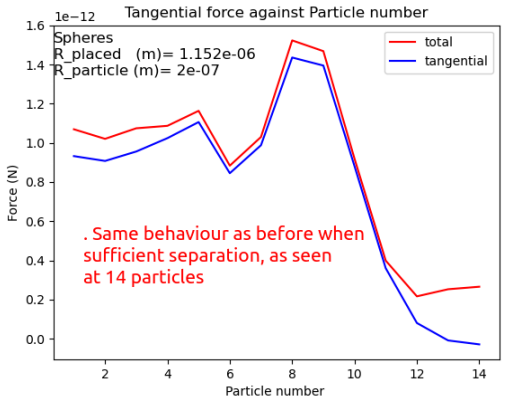
optical forces, resulting in the following plot. These plots show that for large particle numbers (but not overlapping) we get a sudden very large force, and for smaller particle numbers we observe the same behaviour as before (tangential force tends to 0). Note as well that when 30 frames of animation were viewed it was seen that the particles stayed roughly stationary for a large number of particles too (they would also sometimes clip into each other and break the simulation). I should be careful that the hydrodynamics is working correctly, as it is supposed to prevent behaviours like this, however for large numbers of particles this may simply be unavoidable (too closely packed) unless time steps are taken to be much smaller.

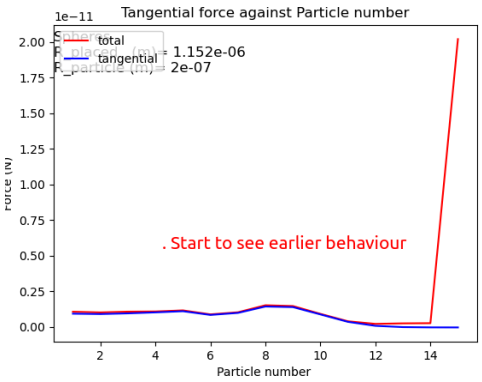






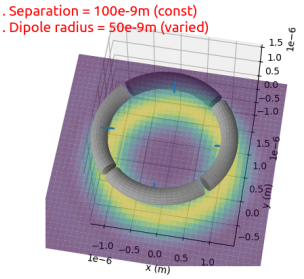
These results imply that the limit of many particles is still followed in the hydrodynamic case, however an extra precaution of not being too close to each other particle must be followed in order to prevent large force spikes. This would be more naturally avoided by having smaller time steps in a dynamics calculation too. This data also seems to suggest that perhaps the scaling of the repulsion should be modified to have a steadier gradient as separation varies (to prevent sudden breaking of system).

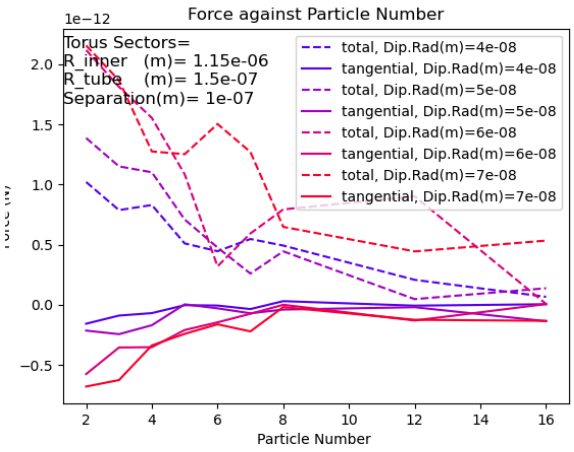


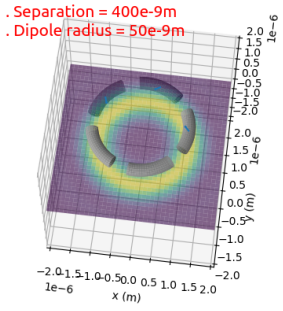


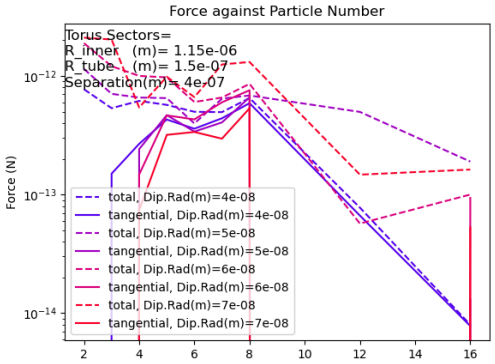
21/12/24

My lab partner has been working on the changes in force as the separation between torus segments is varied, for different dipole sizes, hence I will work on some wider tests to do with how many sections the torus is broken up into with the (1) same separation each time for each section and (2) total separation across all sectors. I will investigate this for different dipole sizes, which will show whether the whole torus is better approximated by many small particles or few larger particles, and if having smaller dipoles is strictly better (we would expect so, closer to reality).

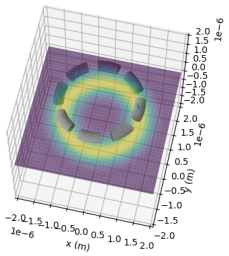


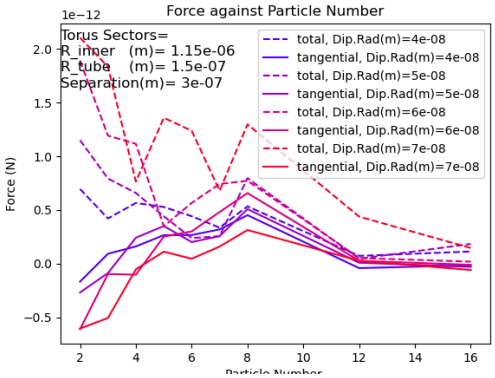




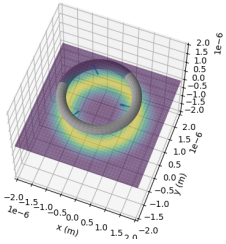


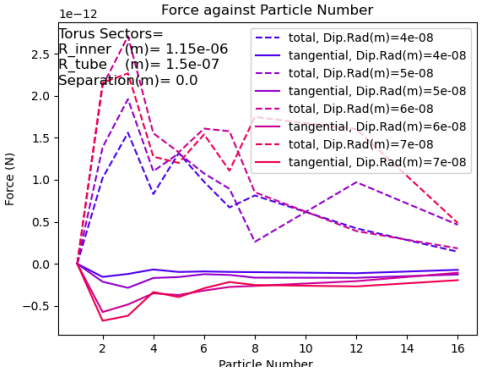
In this initial plot, we see (1) that for larger dipoles we start getting more and more negative tangential forces, lending to the error caused by this greater approximation. Equally we see the overall force tends to be larger in general. Note that this approach means for larger particle numbers there will be larger total empty space across the whole torus.





For a larger separation we also see the tangential forces have an increasing and decreasing behaviour as seen previously for spheres, which overall have very low magnitudes. Note as well



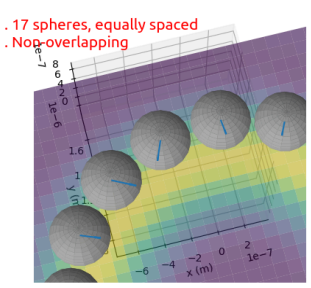


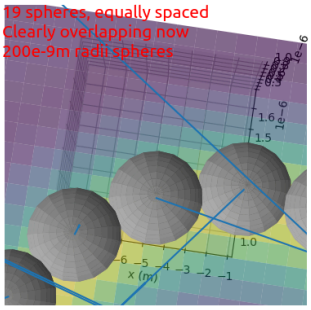
that, as hoped, for smaller separations the system stays at tangential forces close to zero for longer.

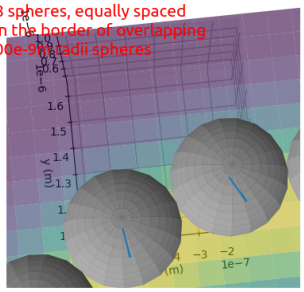
When considering a full torus split into different sections (e.g. separation=0) we see that tangential forces tend to stay close to zero, with some caveat when at low particle numbers, where there is a slight imbalance seen. This is likely due to some small number of missing dipoles near the edges of the torus when generated like this. This could be worked around by generating the full set of dipoles then assigning them to ensure all are accounted for, however this would likely result in missing parts before (now assigned) causing some particle sectors to have more dipoles. This case would be tended to in the limit of small dipole radii as well.

20/12/24

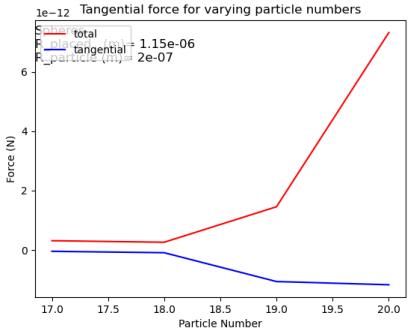
Here I tested to see the absolute limit to the number of 200e-9m radii spheres that would fit in the beam. In theory this would be circumference = 2*pi*1.15e-6m =7.22e-6m, which for spheres of radius 200-9m => you get 7.22e-6/0.4e-6=18.06 particles that could fit in the beam as a maximum. The reality is that dipoles are placed on a lattice which makes the radius of this sphere less exact, allowing some spill-over of dipole occupation, hence we would expect the 18th particle here to possibly have overlap problems.

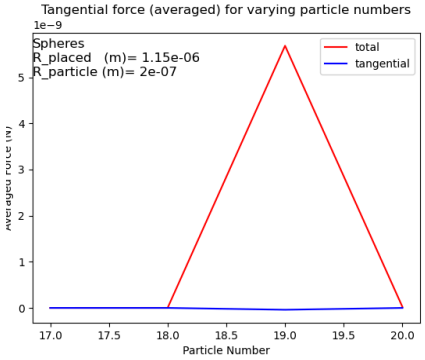


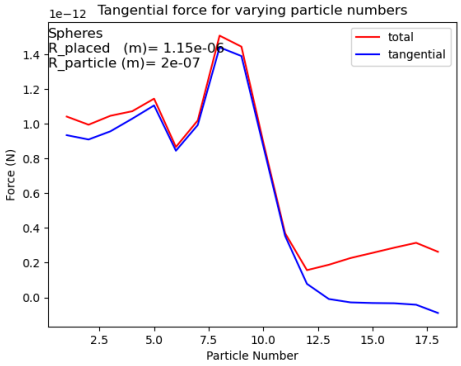


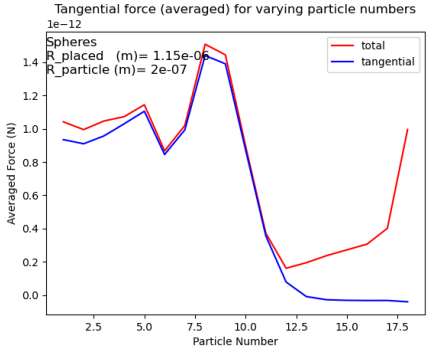


When plotting the forces here we see exactly this result, where we get the decrease up to 17 particles, with some discontinuity in the 0th and averaged force for the 18th due to random large forces from dipoles overlapping and behaving unphysically. This also appears to show that the minimum seen at the end here is not just another turning point (as may be expected for the previous wavelength theory, causing peaks roughly every 4 particles, where here the downwards trend is actually tending to zero). Therefore this shows that the zero value achieved at the end here does appear to be a true limit and not a continuation of oscillating behaviour.









Now, to continue the investigation of the early behaviour given that the latter appears to work as hoped I am interested in looking at the total electric field scattered from the particles. This

should naturally show constructive and destructive effects where high and low intensity fields are seen, but more easily allow me to visualise the additional inter-particle forces as well. My hope is that by viewing this field for an increasing number of particles in the smaller case (<8 particles) I should see how each additional particle affects the field near the target. I will try to consider both the total (incident + scattered) as well as just scattered. The main issue with this is in finding a clean way to retrieve these values from the current program, as they are currently tied into the ***optical_force_torque_array()*** method, and the resulting equations solved in Eigen, which may make retrieval difficult.

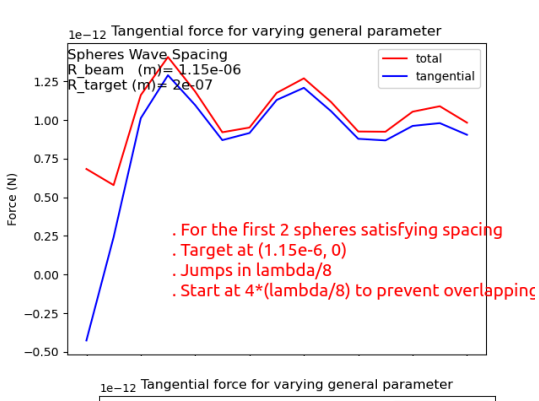
I can see that ***p_array = dipole_moment_array(...)*** finds the polarisation of each dipole, which is the most crucial step in the calculation. From here, the electric field seems to be found and stored in a ***Eigen::Map<Eigen::Matrix3cd> Gradijblock((std::complex<double>*)(gradEE), 3, 3)*** variable using Eigen (/values are stored in here initially, then acted on in to get the electric field in the end).

-> Key points to consider after talking to partner;
. Stepping as dipole radius changes -> jumps on thresholds
. complete torus test would be good
. Show tending is BETTER for smaller dipoles -> closer to reality (especially for torus cases)
→ ***Possible run on BC4 (maybe closed for christmas)***
. Wider test on torus' and their sizes -> test all cases of arclength+sep.
-> Do for all these but for diff. dip. Sizes
. Springs once this is all done

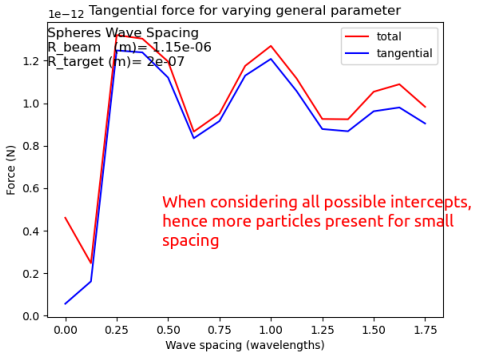
After talking to my partner, we realised that the analysis we have already done on the wavelength idea has shown some constructive and destructive effects occurring, and with a system as complex as this one (Laguerre beam scattering, multiple particles, all scattering with each other) we could very easily get tied down to this 1 idea while we already have a fairly satisfactory answer, and so it would be best to continue investigating other parts of the project (e.g. ensure the limit of the torus case with small dipoles and large dipole numbers, with small separation does continue to tend to have 0 angular momentum). Also, the idea proposed about considering the electric field seems as though it will more difficult to implement and test to ensure it works correctly (as currently only the E field gradient is found, not the E field directly itself, as this is all that is needed for the force to be found, hence the polarisation would have to be used to find the scattered E field, which was done in my work on ADDA but not test sufficiently to be trusted here yet. Once this is tested perhaps we can implement it here too).

19/12/24

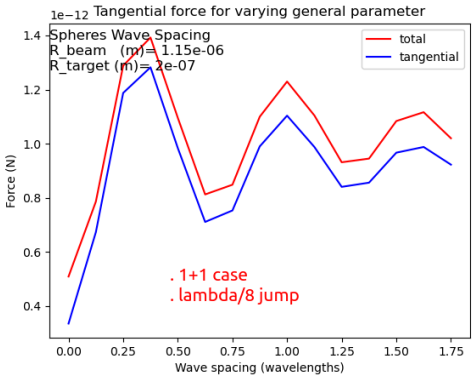
In an effort to try and better understand the lower half of the previous graphs seen, and test whether the wavelength theory is correct, I have programmed a system where radial waves about a target sphere are considered (with spacings between these waves swept through) and a sphere is placed where this wave intercepts the beam intensity ring.



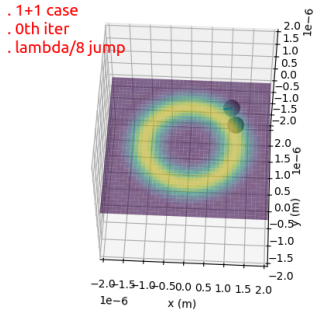
The function can be configured to either place particles at all possible intercepts for all possible jumps, or just for the first 2 intercepts found (for the first jump). When the jump is taken to be integer wavelengths we should expect constructive interference and when it is non-integer wavelength destructive interference should be seen. THe only caveat to this however is that when considering these systems of 2 particles + the target, there is another force experienced from the scattering between the 2 others on each other, as well as on the target, too. This means we should only expect to see exactly constructive interference at integer wavelengths if this contribution is small.

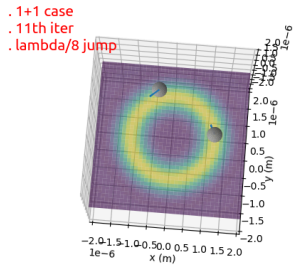


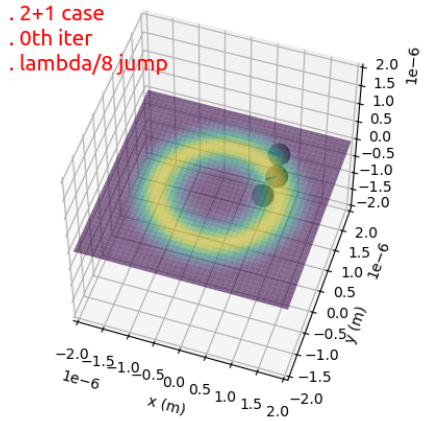
In the 2+1 particle case, we see something similar to this with oscillation as the 2 particles are moved apart, appearing constructive and destructive at different points, however these do not line up exactly on integer and non-integer differences as expected for a case with non-interacting forces. Therefore this approach requires we consider how impactful the other forces are, or just consider the 1+1 case (however even here there is inter-particle force between the target and the other particle, which will then affect the force felt by the target, however you would hope this would be a much smaller force than seen in the other approaches). Not as well that for all these cases the particle has also been placed in the beam, meaning it will feel a torque purely from the beam too. Hence for future tests I need to;

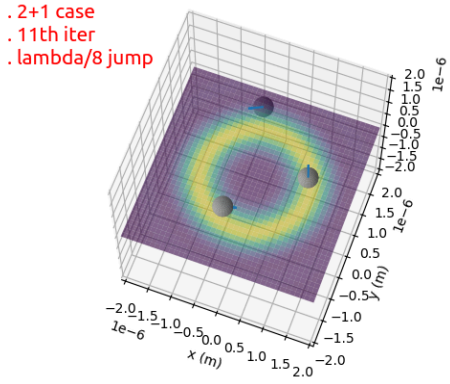


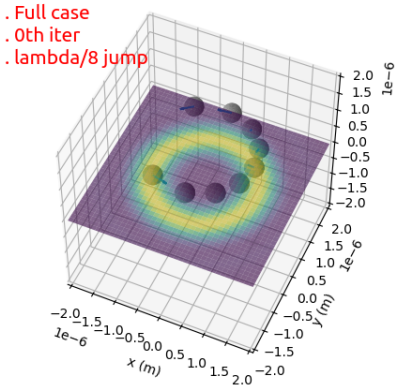
(1) Test the force on a target OUTSIDE the beam
(2) Assess the strength of the force felt by the other particles compared to the base force experience by the circularly polarised beam

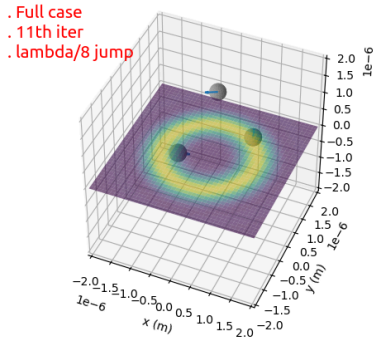




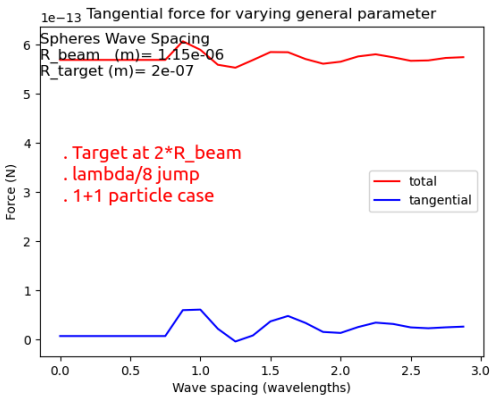


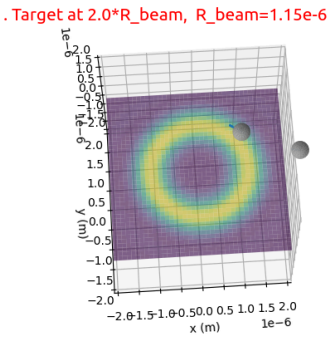






By performing this test when the target is outside the beam intensity ring we should hopefully just see significant forces from the other interactions (due to greater distance from each other we would also expect the magnitudes to drop, however the oscillation is of concern not

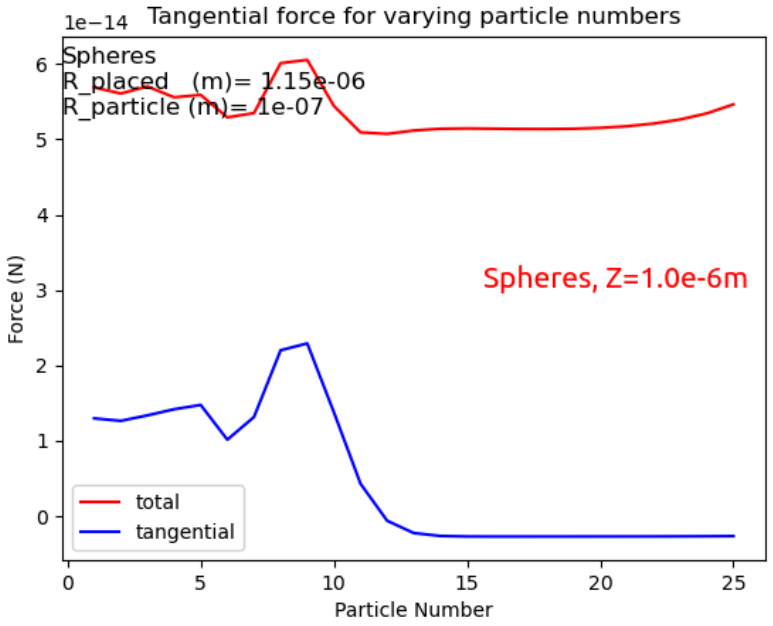


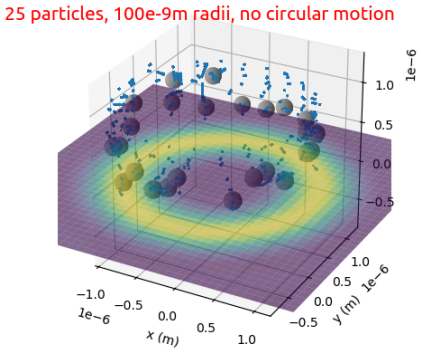


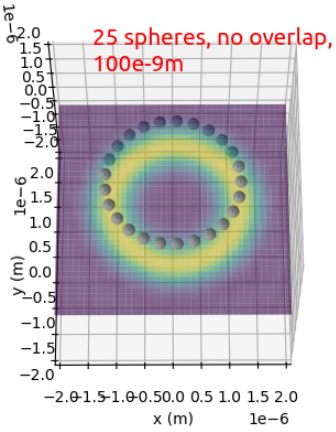
magnitude). In this plot, we see another oscillation, with a peak at roughly 1 wavelength, but then the next peak is significantly below 2 wavelengths. Perhaps instead this is howling that due to the more complicated nature of the Laguerre-Gaussian beam, and the fact multiple particles are present (interacting with each other), you actually should expect these peaks in intensity at values different to exact integer multiples of lambda (e.g. the slightly lower values seen), and that is where for this beam they appear strongest. This is also supported by the fact that the 2+1 and full case had very similar peaks, suggesting it is closer to a beam property (as a beam can be considered as a superposition of plane wave, which for each frequency would have different constructive spacings). This needs some more thought but does seem to justify the differences we are seeing. I need to compare this to the spacings observed yesterday that had constructive/destructive influence.

18/12/24

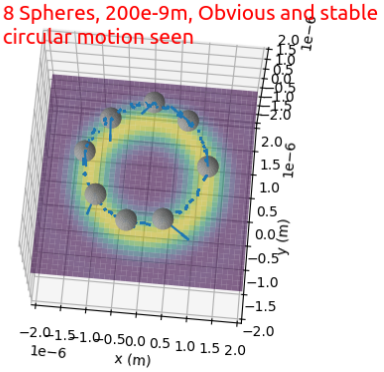
Looking at the data from yesterday, I was interested to see the effect of more particles in order to see if more oscillations occurred or the force continued to tend to 0N. The figure to the right shows that this is clearly tending to some value just under 0 (direction change with small magnitude), however for spheres this small there is no guarantee that circular trapped motion will be seen, as seen here, and so we cannot draw conclusions from this. THe same peak around 8 particles is seen however, indicating that this may not be a feature of the orbital motion, but more about the magnitude of forces involved increasing around this 1 wavelength point.

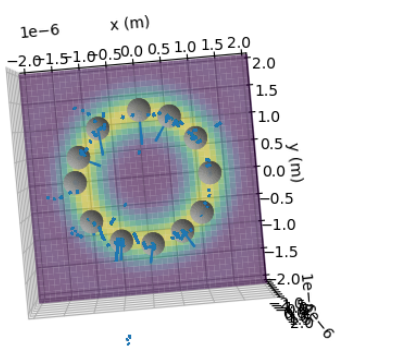




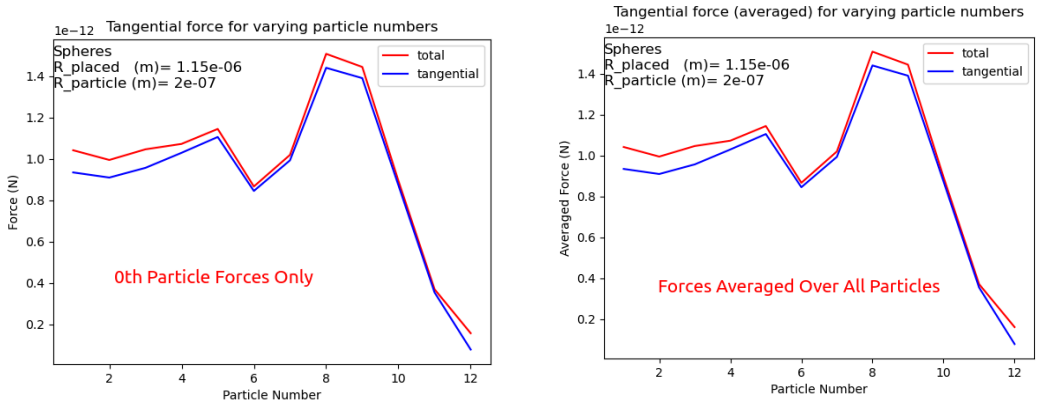


Seeing this made me revisit the first plot I performed to ensure it was making sense in terms of trapping actually occurring but being prevented purely through the large number of particles present. Calculating the dynamics for the 12 particle case for 50 frames, easily enough for circular motion to be observed before, (as seen on the





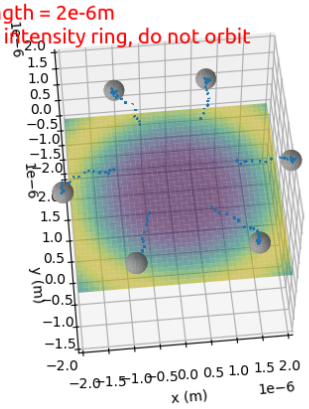
right) showed oscillation in place for approximately 30 frames, before 2 spheres make contact and cause singularities, rapidly (unphysically) jumping spheres out of palace as expected. Then when considering the 8 particle case it is seen that the circular motion is clear and stable for the full 50 frames. This does support the idea of the orbital motion being broken here through more particles present, and builds confidence in the data collected prior. The question about why the motion becomes more vigorous from 1->8 particles is still not entirely answered however (other than the vague idea that perhaps more constructive interference occurs for greater than 1 wavelength gaps between particles), and we still don't know if this regime has oscillating behaviour. The particles cannot be made smaller or else they will not trap and I have already observed that moving them to different placement radii just cause some disparity in tangential forces, as now a force component is dedicated to restoring forces (radial). Hence my options to test now are to consider ***(1) vary wavelength, look for trapping*** to then possibly allow more particles to be added to system for this larger beam width (unless trapping here also requires a larger particle, which is entirely possible, which would negate the extra width I introduced causing no difference), or ***(2) test different refractive indices***. I can also implement ***(3) the averaging of tangential forces***, not just the 0th particle forces, which would reduce the error in my results.



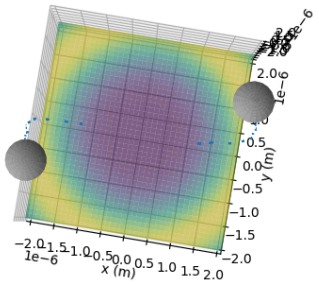
Considering the forces averaged over all particles Vs just the 0th particle did not seem to affect the results. This is to be expected however as on the 0th frame the particles are all at symmetrical positions, it is only once the Brownian motion step is introduced that large fluctuations are seen, and so for the cases considered here it is entirely irrelevant. However, if I did want to introduce a system were the particles are moved slightly about their origin then readings are taken for a better average then this would be necessary to use.

When looking at larger wavelengths again, I attempted doubling the wavelength (to 2e-6m) and keeping the particle radius the same (200e-9m). This resulted in the particle moving along the E gradient towards the high intensity ring, then oscillating at this point (no orbiting), which is as I predicted may occur. When doubling the particle radii (to 400e-9m) inside this x2 wavelength beam it is then seen that the particles now orbit again (after travelling radially to the beam's stable point). Hence this approach is identical to the original and so will not be able to be used to test a larger number of particles.

R=200e-9m
Wavelength = 2e-6m
Move to intensity ring, do not orbit

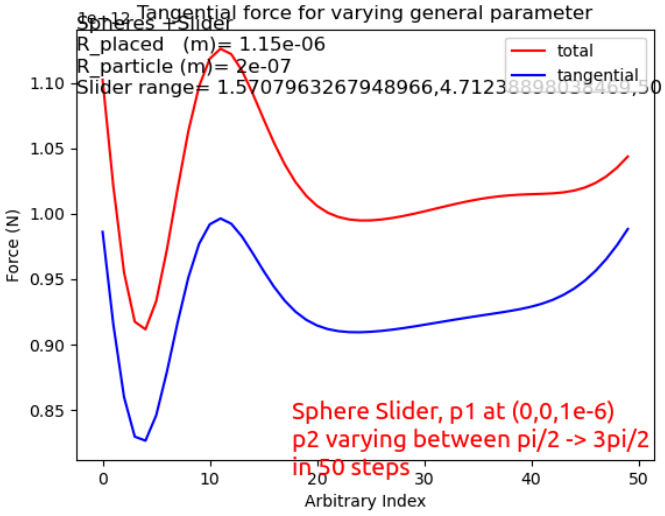


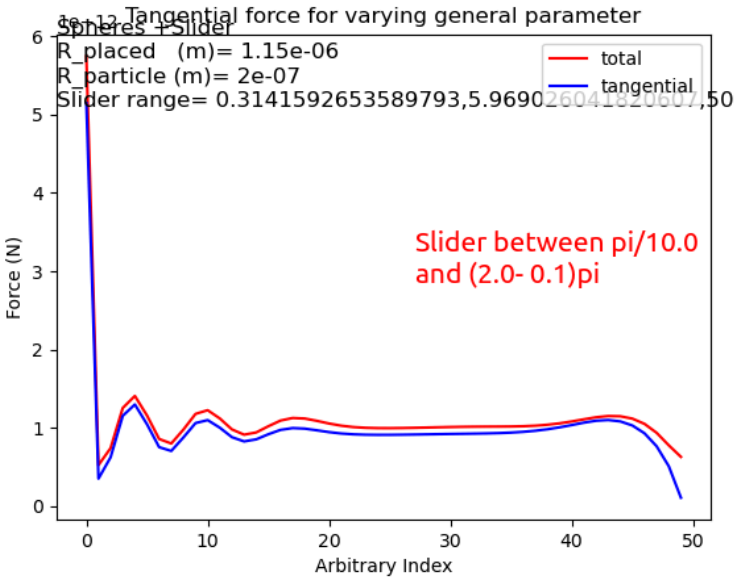
R=400e-9m
Wavelength = 2e-6m



These attempts have not allowed me to probe the lower limit of particles any further, so instead of trying to introduce ore particles to further test this I will try to move the particles to be less periodic (e.g. pair or triple particles up) in order to try and break the wavelength gap and see how this affects the results. I predict that by moving some particles closer and other further, I should get back the behaviour of smaller systems as the grouped particles behave as a new single particle-like object.

When testing this for 2 particles, measuring the force on the 0th (locked at [0,0,1e-6]m) the figure on the right is given. As expected the force at the magnitude at the start and end is equal due to the symmetry, and this oscillating pattern is observed. This change can only be due to the other particle's interaction (***it would be good to measure exactly how much impact this is, seemingly more than expected from this diagram***). This oscillation is maybe not as periodic as would be expected if the force is solely from constructive/destructive interference as the particle sweeps through being multiples of the wavelength away. This is none-the-less very interesting and I think very valuable in trying to understand the lower region of the previous plots found (in assessing if my assumption

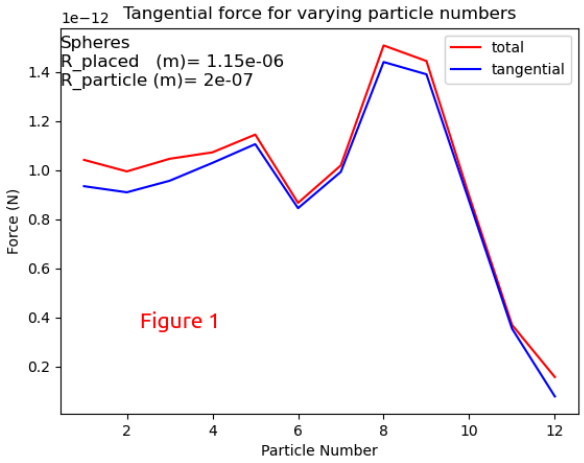




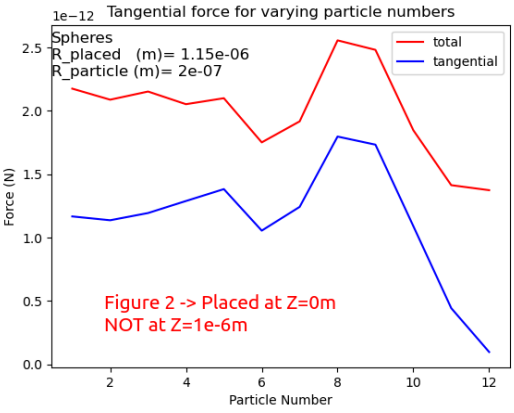
about the wavelength is actually correct or not).

17/12/24

A function has now been written to plot the data generated previously. On the right in **figure 1** the results for the purely tangential and total force on a series of spheres placed at **Z=1.0e-6m** (roughly their stable height) can be seen. The pattern seen here is very unusual, especially for a simulation (more deterministic), where between 1 and 8 particle there is an increase in force as particles are added, and after this (8-12) we see a drastic decrease in force (note as well that 12 particles is close to the limit of particles at this given radius before they begin to overlap and so behave unphysically). The wavelength used in these simulations was 1 micrometer



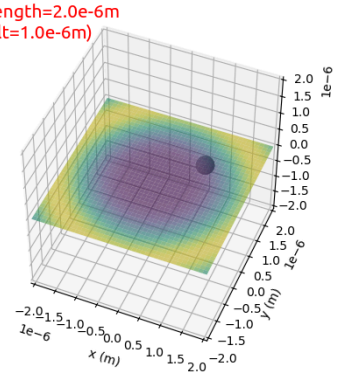
, with the beam having its high intensity at a radius of roughly 1.25 micrometers and so a circumference of beam is about 7.5 micrometers, with 8 particles of radius 2e-7m (diameter 4e-7m) we have 3.2e-6m taken up by particles present, meaning for the 8 gaps between particles seen now they take up [(7.5-3.2)/8]e-6m ~ 0.5 micrometers (slightly more than this). If you now consider the differences in the positions of centres of the particles (not just the gaps) you add on an additional 0.4e-6m (2*radii), to essentially get a result ~1e-6m, which is the wavelength, and so could be the cause if this switch in behaviour. This can be tested by changing the wavelength and seeing if this peak shifts (note that both figure 1&2 show the peak at this 8 particle point)




```
options:
  frames: 1
parameters:
  wavelength: 1.0e-6
  dipole_radius: 40e-9
  time_step: 1e-4
output:
  vmd_output: True
  excel_output: True
  include_force: True
  include_couple: True
display:
  show_output: True
  frame_interval: 2
  max_size: 2e-6
  resolution: 201
  frame_min: 0
  frame_max: 1
  z_offset: 0.0e-6
beams:
  beam_1:
    beamtype: BEAMTYPE_LAGUERRE_GAUSSIAN
    E0: 300
    order: 3
    w0: 0.6
    jones: POLARISATION_LCP
    translation: None
    rotation: None
particles:
```

Sphere vary parameters- For figure 1&2

Wavelength=2.0e-6m (default=1.0e-6m)



It is also worth pointing out that in the above figures the ***'total force'*** shown in red does NOT include the Z force component either, as we are only interested in this circular motion, hence the offset seen in figure 2 is purely from the total force in the XY plane being less tangential at Z=0, and is NOT due to the Z component of force that would also be present here at this point.

By changing the wavelength we see that the beam width also changes (by the same factor), and so we would have to move particles back into the centre of this beam to try and observe trapping again. But note that this beam has x2 the radius (for x2 the wavelength), hence has 2x the circumference, and so will be able to fit twice as many particles with a gap of x2 its prior value, and so we would expect an identical situation. MAybe this can instead be investigated by just changing this radius of particles involved instead, so the wavelength will interfere differently over the full particle volume (how if the particle is too large then Mie scattering will no longer be valid, and equally the particle will not trap, which is also true if the particle is too small).

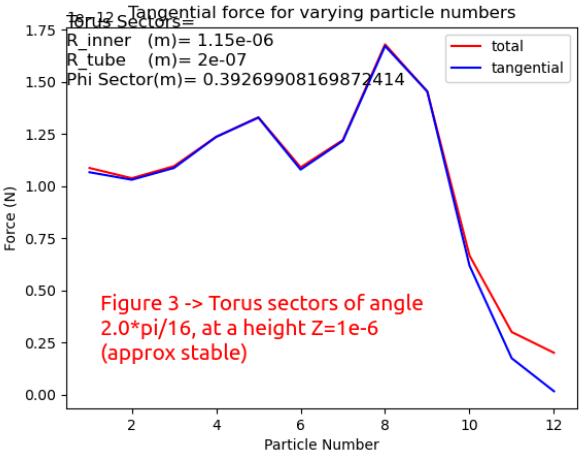
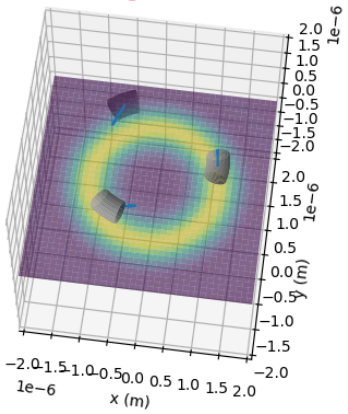


Figure 3 -> Torus sectors of angle 2.0*pi/16, at a height Z=1e-6 (approx stable)

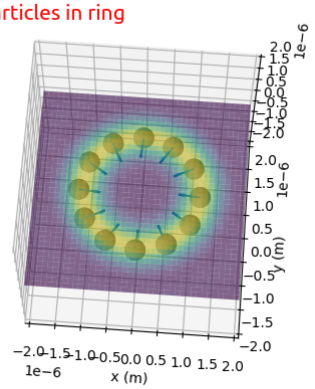
When considering the same test as above with torus sectors, we get the same looking results, with the same 8 particle turning point seen. This is expected as for this small angle and same tube radius as the spheres, the torus sectors are approximately spheres. A more interesting case is now to consider different angles and see if similar results are seen.

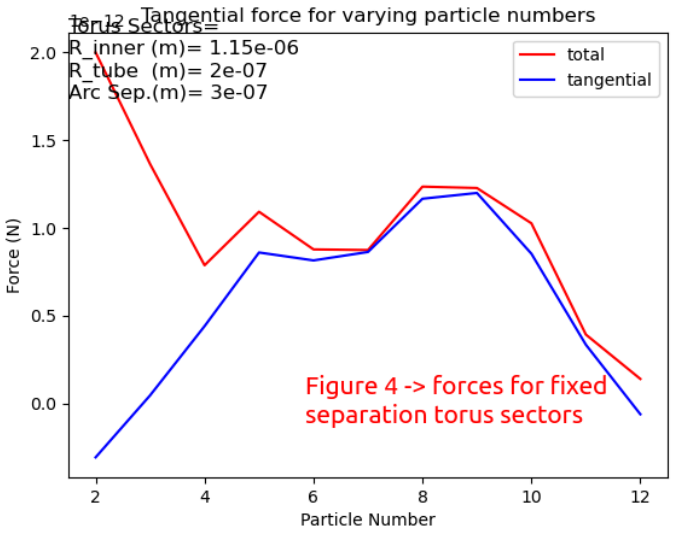
We can also consider cases where the sector angle is not fixed, but rather the separation between sectors, when this is done, we get figure 4, which shows how the forces change as more particles are added, which is equivalent to more of the space becoming empty space from the extra separations introduced. Here there also appears to be a peak somewhere between 8-10 particles where a dipping behaviour is then seen after. This is for a fairly small separation, and so would be good to experiment with some other larger separation values. It would also be good to see how this is affected for varying inner radii,

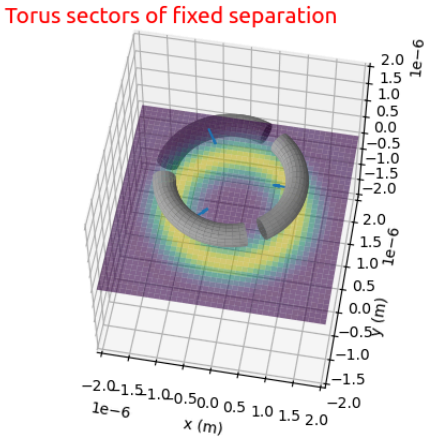
Torus sectors of fixed angle



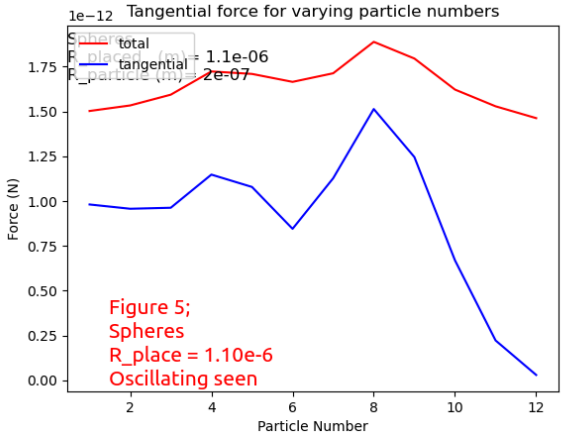
12 particles in ring

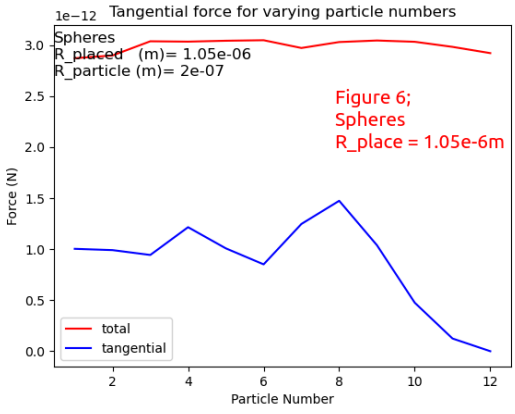


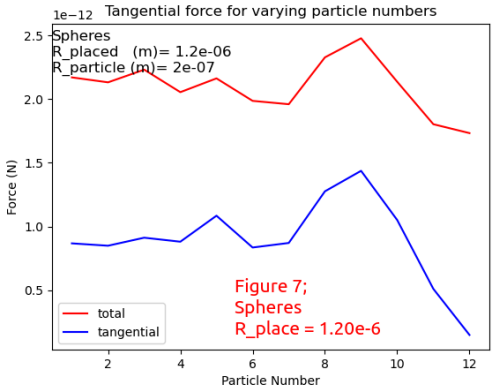




(1) Change wavelength, see if peak shifts → ensure beam does not change dramatically / talk about it
(2) Do same sphere plot but at Z=0m
(3) Do equivalent plot but for torus sectors
(4) Then do unique torus plots for separation changes
(5) Vary inner radii for other torus plots



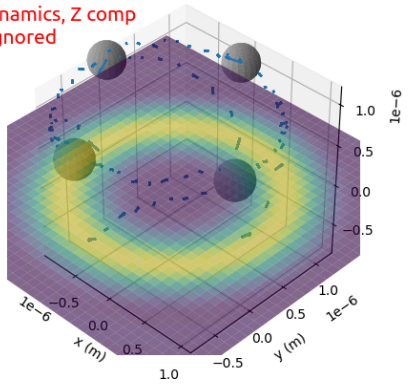


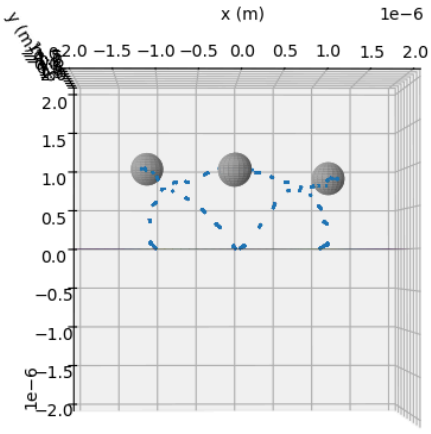


16/12/24

I have revised the recently added plotting function (for quiver plots) to now plot quiver arrows for each frame of an animation as oppose to just a single frame/time in order to see more closely how the force direction changes in the simulation in order to see why the forces observed yesterday were mostly normally pointing (not as tangential as hoped for). I have seen this is in part due to two reasons, first the forces jump quite a bit each frame due to the time step allowing particles to jump noticeable portions of the beam intensity ring in a single frame (not huge jumps, but for the tight, hence high gradient, ring we have small jumps are still enough), leading to each force arrow varying, and so an average should really be taken to get a more realistic value, which I will try implementing through the use of N brownian steps which then have their forces averaged in hopes of simulating so wiggle room about where the particle sits to retrieve a better average. The other key reason for the different force is due to the particles migrating up roughly 1 micrometer before actually settling into their orbital motion, and so I should start the particles from here to begin with in order to better read their actual forces when in the orbiting regime. This may need more careful consideration for the torus sector particles as perhaps they would prefer to settle at a different Z height, so I should try to experiment with some different heights to see where the tangential force dominates for these particles (likely the same, especially in the limit they are small and so close to spherical).

Forces and Tracers shown for sphere dynamics, Z comp of force ignored

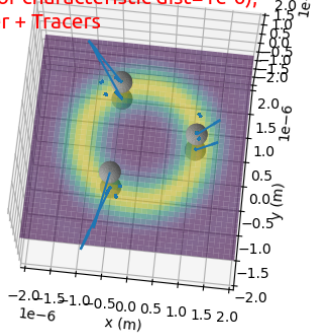




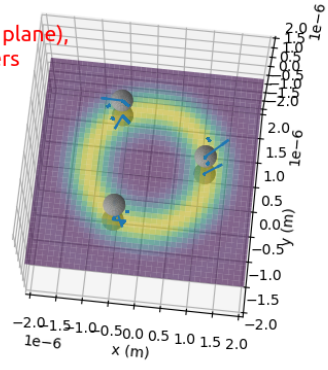
Spheres settle in orbit at ~1micrometer for FusedSilica (n~1.46)

Starting particles initially at their higher stable height seems to give the same mostly radial force when located in the XY plane at a radial distance of exactly 1.0e-6 (note this comparison will add an additional small optical force between the compared particles, however this should not affect the results significantly). When a small

Frame 1, 5% offset (of characteristic dist=1e-6), Force quiver + Tracers

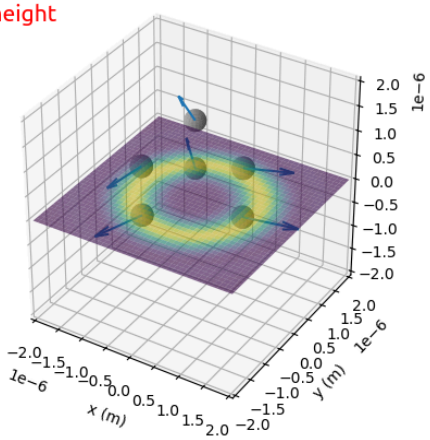


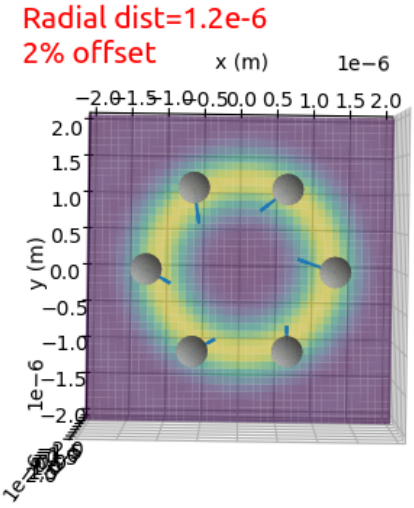
Frame 2, 5% offset (XY plane), Forces + Tracers

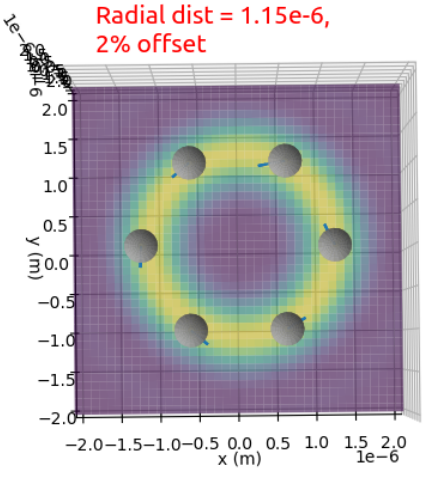


random perturbation is added to the XY position, we see very similar results with perhaps slightly more tangential forces, however when allowed to evolve for even a single frame (shown below) the forces near instantly align tangentially. Considering the forces calculated in the program, it can be seen that the '***optforce'*** used in these quiver plots (returned by the ***simulation()*** method) does indeed only include the optical forces, and does not mix the additional (e.g. Buckingham) forces into this (further made sure of by removing this Buckingham calculation temporarily), hence I would attribute this change of behaviour to the position change from the diffusion matrix which is dotted with the force. This means I should get better (more tangential forces) by more carefully picking the starting position to account for this diffusion that would have occurred in a direction favouring the electric gradient (force direction). When doing this figures below are shown (with a new more gentle random motion added of 2% of the 1e-6 characteristic distance), which shows a significantly improved tangential force, due to this better positioning, and so the random motion should be left for now as it is not required in I believe in this situation would only add possibility for anomalous data.

Force comparison for particles started at the (1) focus and (2) stable height



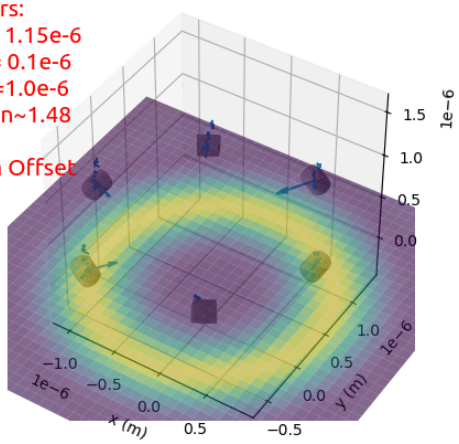




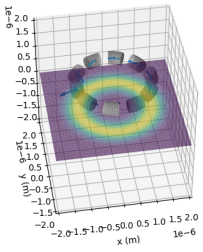
With the tangential forces acting as expected now for spheres, I checked the forces for the torus sectors to see how their behaviour looked, first considering the case on the right for small approximately spherical sectors (hence a radius of roughly the tube radius, which was 100e-9 here) sitting at 1.15e-6 , which oscillated radially slightly, showing no rotational motion in the 100 frames it was observed for (tracers can be seen for this small motion). For a larger sphere we would now expect the orbital motion to be seen as the correct size (as observed from previous

experimenting) has been reached, and here we see rotational motion start, however we also observe the torus sectors begin to leave the beam through the Z axis (as predicted, so we may need to move this further down to make them more stable, or lower their dimensions/refractive index to make them more stable). The greatest tangential forces were seen when placed at approximately 1.15e-6 radius (1.1 collapses and 1.2 diverged quickly), but note that this motion only has these good tangential values for small times (later times result in the torus sectors moving out of the beam in the positive Z direction, where the forces then point inwards towards the centre radially).

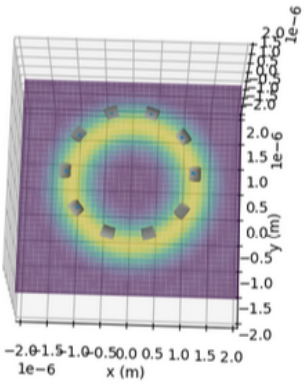
Torus sectors:
Inner radii= 1.15e-6
Tube radii = 0.1e-6
separation=1.0e-6
FusedSilica:n~1.48
Z=1e-6
No Random Offset

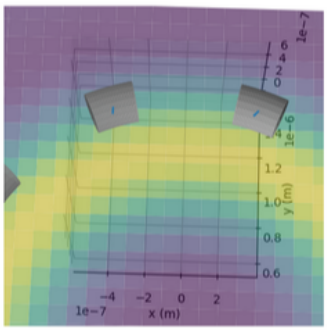


10 sectors, small separation, rotation seen



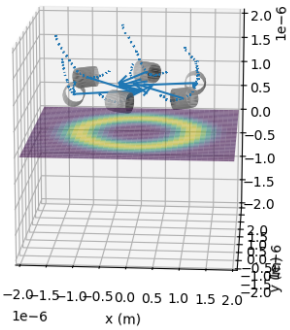
Testing this in smaller limits (see left figure) we also continue to see rotation, and do see what initially appears to be reduced magnitude (which would support the hypothesis of forces tending to zero in limit of many particles).



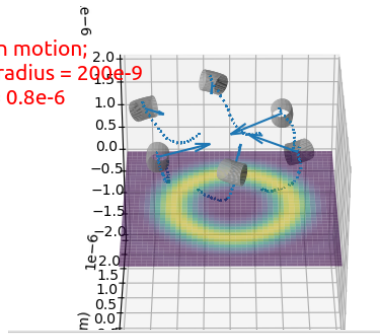


It is important to note here that no hydrodynamic interactions are included here (purely optical forces), hence overlapping will cause singularities with the distances between dipoles and so errors, but equally misleading data from hydrodynamics will not be shown either (which we theorised with our supervisor could be part of the reason why more particles resulted in speeding-up motion, which was opposite to what we predicted).

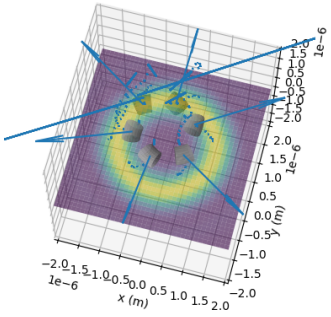
Early in motion;
Tube radius = 200e-9
Separation = 0.8e-6



Late in motion;
Tube radius = 200e-9
Sep. = 0.8e-6



For 1.1e-6 inner radius -> Collapses



With these cases seen to working when simulated and with working force data storage I now need to create some functions to analyse the data in plots.

(1) The first plot (for spheres) I want to observe is to consider the tangential force of the 0th particle as more particles of equal radii are added (the 0th as this is unmoved when more particles are added).
(2) I then want to consider this, but for the tangential forces of all particles averaged (hopefully less noisy than just the 0th consideration).
(3) I then want to consider this same thing with the torus segments.
(4) After this I want to consider (for both) how the tangential force (with a large given number of particles) and how the lattice resolution changes this force.
(5) For the torus sectors only I want to see how the separation changes the forces for a given number of particles.
(6) Then it would also be good to consider a complete torus (no gaps) to observe the (hopefully) lack of rotation (this will require the tracking of torque specifically).

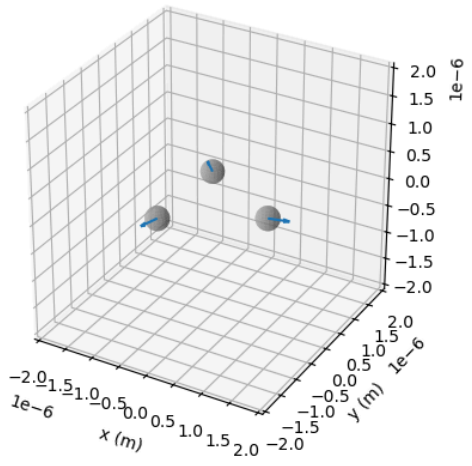
14/12/24 - 15/12/24

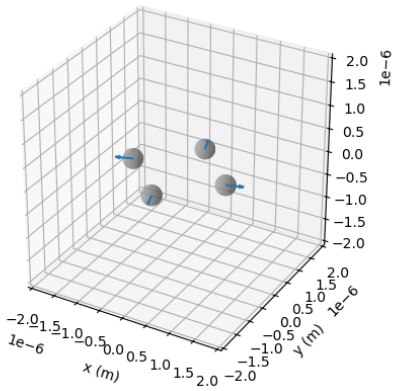
A private Github repository (https://github.com/James-Paget/SH_DDA/tree/main) has been set up for me and my partner to work on the project collaboratively over Christmas. Our current approach to this is to each work on branches of the main program, then perform a pull request with the main branch, merge into the main, then delete this old branch (create a fresh branch when working on another aspect of the program).

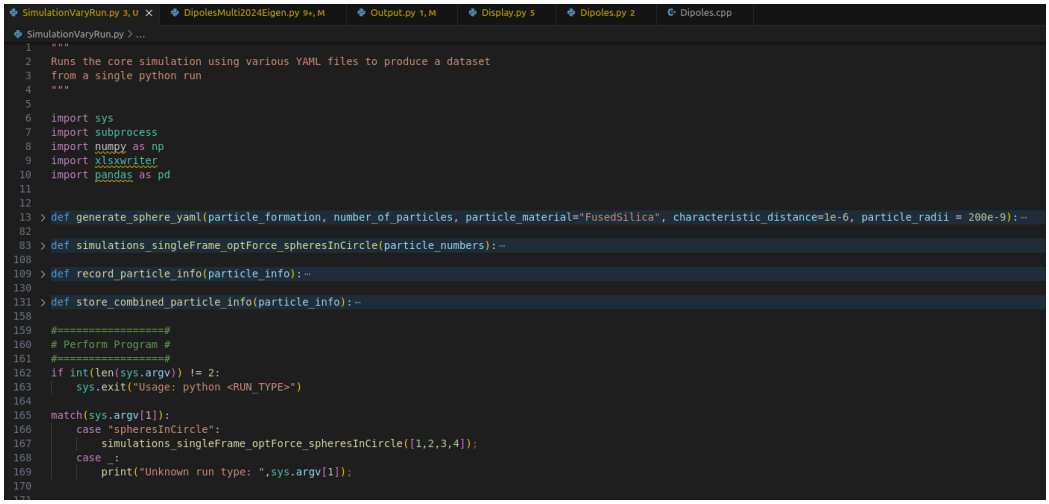
I have now started writing setup functions to get the large datasets needed for comparison of tangential forces here. This is done through a separate python file which generates a series of new YAML files (for spheres of varying parameters), runs the ***DipolesMulti2024Eigen.py*** file with this generated YAML, records the pos and force data from this run then repeats the process for new scenarios. Data recorded like this is then stored in a new XLSX file (1 line for each scenario, holding position and force for each particle) for comparison later. Shown below is an example of the plots generated for varying particle parameters, as well as the structure of the code used to generate this.

Next the forces can be compared for each particle, however I have seen that the forces (arrows shown in plots, implemented by my partner) are pointing more normally, and so could be a problem or caused by placement too close into the beam (currently at 1 micrometer radius, may need to be larger).

Once this is fixed, I can generate a series of plots for this with varying particle sizes, dipole sizes, grid resolution, spherical vs torus particles, particle numbers and separation between particles.






SimulationVaryRun.py 3, U
DipolesMulti2024Eigen.py 9+, M
Output.py 1, M
Display.py 5
Dipoles.py 2
Dipoles.cpp
SimulationVaryRun.py > ...
"""
Runs the core simulation using various YAML files to produce a dataset
from a single python run
"""
import sys
import subprocess
import numpy as np
import xlsxwriter
import pandas as pd
def generate_sphere_yaml(particle_formation, number_of_particles, particle_material="FusedSilica", characteristic_distance=1e-6, particle_radii = 200e-9):
def simulations_singleFrame_optForce_spheresInCircle(particle_numbers):
def record_particle_info(particle_info):
def store_combined_particle_info(particle_info):
#================#
Perform Program
#================#
if int(len(sys.argv)) != 2:
sys.exit("Usage: python <RUN_TYPE>")
match(sys.argv[1]):
case "spheresInCircle":
simulations_singleFrame_optForce_spheresInCircle([1,2,3,4]);
case _:
print("Unknown run type: ",sys.argv[1]);

5/12/24

Today me and my partner had a meeting with our supervisor and discussed the changes made and steps we should take now. After talking about the hydrodynamics not working properly for the torus due to many of the equations assuming spherical geometry we essentially come to the conclusion that dynamics for the torus should not be considered for now due to the highly complicated nature, and the risk of going down another rabbit hole of unrelated research (as we are really just concerned with the optical forces). If we really wanted to observe dynamics, we could try modelling the torus with a series of spheres, as is common in literature (this approach used for many simulations of biological chains of particles) due to this complexity and is the approach taken by Simon's previous paper on nanowires (constrained lengths of dipoles). This point also means we can ignore the rigid body rotation of the torus as we just want to consider the single instant of time and forces observed to look for optically induced torque and how this changes with different separations, particle numbers, etc (these are ideas I was aware of before this meeting, but it was good to clear up the fact that the dynamics is not particularly important here).

Hence the next steps are to remove the hydrodynamic forces from the simulation to just have optical forces, observe these values for several parameter changes, and compare these forces to (1) a set of spheres instead of dipoles tested similarly and (2) a dipole generation on a polar grid (not cubic as is currently performed) to see what differences this makes to force and if a correction has to be made (we are currently unsure if the assumptions about polarisation we have made will break here due to refractive index changes in R, density changes in R or volume changes in R, or perhaps it will still give very similar results to the straightforward cubic lattice approach). This polar lattice would let us place dipoles for a torus in a regime that allows cuts and breakages in the torus with artifact effects at the edge (where the cubic lattice would give a step-like gradient, not a smooth cut, introducing phantom forces which would become less prominent as the dipole size is reduced and dipole number increased) and so would be good to work with if possible. I feel as though the change will come from a non-isotropic volume being used for each dipole (they become less dense for dipoles placed at a greater radii, if stepping in constant angular jumps), and that this could possibly be corrected for by a linearly (or possibly cubic) scaling factor that can be remembered in the shape object **args**. This would also let us possibly try other lattice formations like hexagonal lattices.

Another factor of our talk was about the hydrodynamics between spheres of different radii, which currently uses the average radii of the sphere for its force calculation, which is not strictly correct either, and that this is also actually a slightly gray area of research too, where most researchers will simply stick to spheres of the same radii due to the added complexity of dealing with different spheres. Simon mentioned that there was a paper he found before that handled hydrodynamics of spheres with different radii, but this was apparently quite a rare topic and so may not be worth pursuing any further. However, the system is now set up to allow this change to be made for when this is eventually accounted for in the future, however should be avoided mostly for now unless specifically needed (it would still be good to find this paper however to leave the option open to us).

4/12/24

Lots of progress was made today working directly with my partner on the same program. Our aim was to generalize the aforementioned problems with hard-coded sphere qualities in the main program, and lack of variability in shapes (it could only work with spheres, and all the spheres had to be of the same radius and so same number of dipoles, leaving the only variation in the materials/refractive indices used for each).

After the work performed today, a new ***shape*** and ***args*** parameter in the particle YAML dictionary was introduced to specific different shapes and their own unique list of parameters, which was immediately used to implement a torus shape and convert the previous sphere shape to the new system. With this new system, the particles could each have varied sizes (and so dipole numbers, however the actual volume of each dipole was still kept the same) and systems of different sizes and types of particle can interact to find the forces on each (totalled over each particle's dipole set).

The torus generated here can be generated as a sector over any given phi range. Currently this uses a function to generate a list of dipoles for a shape primitive, which is then offset by the ***torus_centre_radius*** to move the particle's origin from the coordinates origin to the position in the r-hat direction that intersects with the centre of the torus cross section.

This process of generalising the interactions of particles required extensive changes to the python **DipolesMultiEigen2024, Dipoles and Particles files, as well as to the cpp and hpp Dipoles file too**. The main change that will be seen in all of these is now most calculations have a match -> case condition to alter the calculation depending on the shape type being considered, and generally most calculations have another ***dipole_primitives_num*** list that follows the particle set around that keeps track of the number of dipoles belonging to the flattened ***dipole_primitives*** list holding all the dipoles for all particles together.

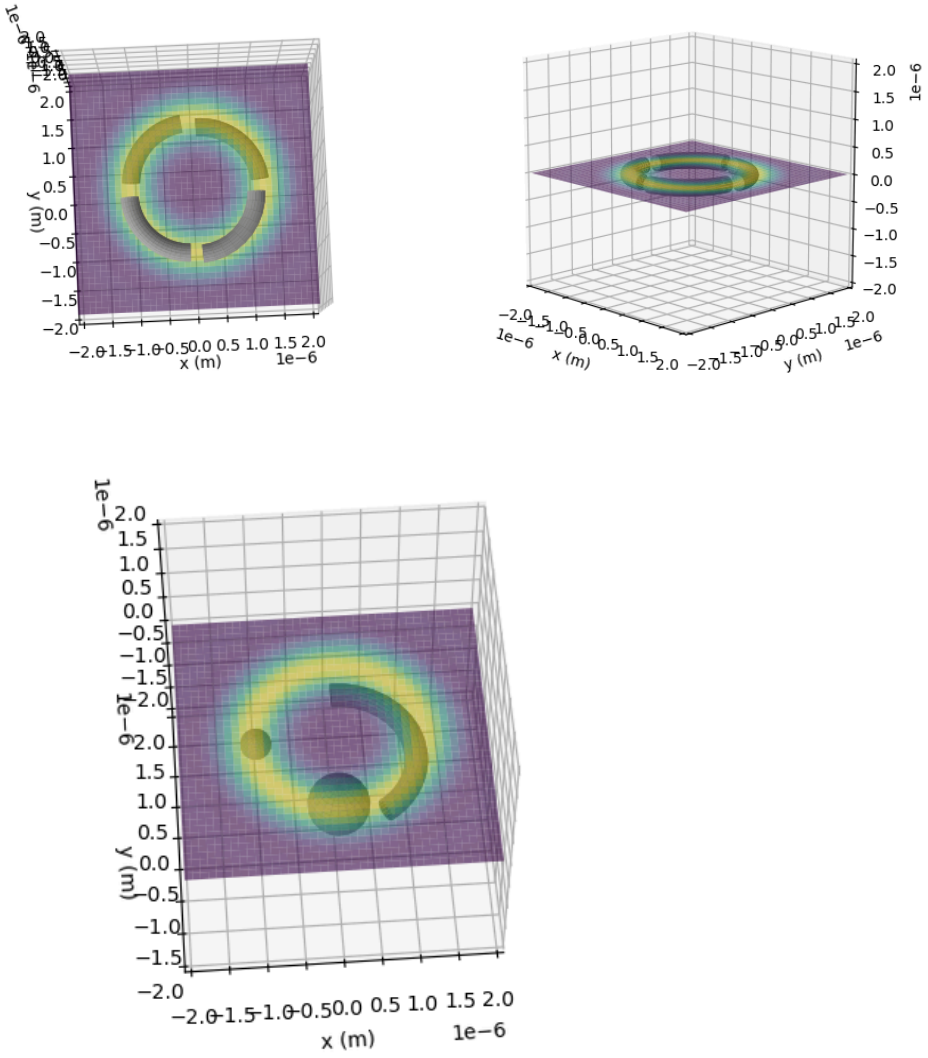
The ***animate_particles3d()*** function also had features of it improved, including the accurate display of torus sectors being considered, which is setup to allow further particle shapes to be added without any issue.

Note as well that changes to the c++ files required additional wrapper arguments to made in the python version that are used, and it is worth noting that issues were encountered using the ***int*** wrapper which had to be changed to the ***np.int64*** wrapper, with ***long**** arguments taken in c++ (as opposed just int arguments) when dealing with lists of integers, in which we observed the length of the integer was not correctly recognised by c++, and so lists had the values with 0 terms interlaced between them, making the list twice as long as required.

Current problems with this implementation so far are as follows; the orientation of the particle (when test with the torus') is fixed, meaning no rotation occurs (this was already known with the spheres but was not a big problem as they were axisymmetric), the Buckingham force and a few other interactions (such as drag) assumed spherical properties which were not obvious for torus shapes (shape cross sections and coefficients for drags, etc) were left as their spherical

counterparts for the time being, which can cause repulsion forces to appear incorrect for the torus as has had to assume a spherical bounding region about its rough centre of mass. As well as this, the centre of mass is not used for the origin but rather another offset from the coordinate origin the the point coinciding with the torus' centre radii and the the r-hat direction (this can be changed quite easily to COM, but for now sufficed to just use this transformation).

The forces are calculated for this object and so now they can be placed near each other and have their forces compared as the gap and number of sectors is varied. The only caveat to this working fully now is essentially the Buckingham force which, due to the close proximity we place particles, needs to more accurately represent the torus shape and not a spherical shape so particles can be brought together. It should also be noted that the torus generation here still places dipoles on the lattice, however these could be spaced differently if required as a FFT is not used here to solve the linear equations ('Eigen' library used instead).



2/12/24

Only a small amount of work was done today in adding the torus generation to the SH_DDA program. I have copied the conditions over into a new ***torus_positions()*** and ***torus_sector_positions()*** functions to do a full or segmented torus (prior can be removed once the latter is confirmed working well, as it is simply a more generalised version). This can then be called instead of the ***sphere_positions()*** function. While adding this, I saw that the simulation can currently only support particles of the same radius, and some-what hard-coded to deal with just spheres (as this is the only particle type considered thus far), and so I can generalise this slightly to better manage variable shapes. This generalisation should also make it easier to pull out other parameters stored in the particle details in the YAML file (such as sector phi ranges, as well as centre and tube radii for the torus [as opposed to just the radius for the sphere]).

```
#
dipole_primitive = sphere_positions(args[1][0], args[0])  #sphere_radius, dipole_radius
#dipole_primitive = torus_positions(args[1][0], args[1][1], args[0])
#print(dipole_primitive)
```

```
def torus_sector_positions(torus_centre_radius, torus_tube_radius, phi_lower, phi_upper, dipole_radius):
    #
    # Only supports torus flat in XY plane
    # torus_centre_radius = distance from origin to centre of tube forming the torus
    # torus_tube_radius = radius of the tube cross section of the torus
    # phi_lower = smaller angle in XY plane, from positive X axis, to start torus sector from (0,2*PI)
    # phi_upper = larger angle in XY plane, from positive X axis, to end torus sector at (0,2*PI)
    #
    # ** Could be extended to be tilted
    # ** Could also move x,y,z calcualtion into if to speed up program -> reduce waste on non-dipole checks
    #
    dipole_diameter = 2*dipole_radius
    dd2 = dipole_diameter**2
    ttr2 = torus_tube_radius**2
    print(torus_centre_radius, torus_tube_radius, dipole_radius)
    num_xy = int( (torus_tube_radius+torus_centre_radius)//dipole_diameter)     #Number of dipoles wide in each direction (XY, wide directions)
    num_z  = int( torus_centre_radius//dipole_diameter)                         #Number of dipoles tall (shorter)
    #
    # Counts number of points in object
    #
    number_of_dipoles = 0
    for i in range(-num_xy,num_xy+1):
        i2 = i*i
        for j in range(-num_xy,num_xy+1):
            j2 = j*j
            phi = arctan2(j/i);
            if( (phi_lower < phi) and (phi < phi_upper) ):
                for k in range(-num_z,num_z+1):
                    k2 = k*k
                    rad_xy_2 = (i2+j2)*dd2
                    #pow( centre_R-sqrt( pow(point.x,2) + pow(point.y,2) ) ,2) +pow(point.z,2) <= pow(tube_R,2)
                    if (torus_centre_radius -np.sqrt(rad_xy_2))**2 +k2*dd2 < ttr2:
                        number_of_dipoles += 1
    #
    # With pts size known now, particles are added to this array
    #
    pts = np.zeros((number_of_dipoles, 3))
    number_of_dipoles = 0
    for i in range(-num_xy,num_xy+1):
        i2 = i*i
        x = i*dipole_diameter
        for j in range(-num_xy,num_xy+1):
            j2 = j*j
            y = j*dipole_diameter
            phi = arctan2(j/i);
            if( (phi_lower < phi) and (phi < phi_upper) ):
```

This code needs to be tested fully, which will be performed once a general approach for reading particles has been added to allow the method to work as intended. Once this works the visualisation can be done and forces considered.

1/12/24

I have started trying to implement a new dipole placement method to generate the arc of dipoles for the torus particle (which be split into sections, and will be placeable in a grid and cubic underlying lattice to compare the breaking of anisotropic polarisability assumptions), but before I can do this I have been attempting to understand the procedure the program follows.

```
particleinfo = ReadYAML.read_section(sys_params,'particles')
#=============================================================== (1)
# Read particle options and create particle collection
#=============================================================================
particle_collection = Particles.ParticleCollection(particleinfo)  (2)
print(particle_collection.num_particles)
```

```
def sphere_positions(sphere_radius, dipole_radius):
    dipole_diameter = 2*dipole_radius
    dd2 = dipole_diameter**2
    sr2 = sphere_radius**2
    print(sphere_radius,dipole_radius)
    num = int(sphere_radius//dipole_diameter)
    number_of_dipoles = 0
    for i in range(-num,num+1):
        i2 = i*i
        for j in range(-num,num+1):
            j2 = j*j
            for k in range(-num,num+1):
                k2 = k*k
                rad2 = (i2+j2+k2)*dd2
                if rad2 < sr2:
                    number_of_dipoles += 1
    pts = np.zeros((number_of_dipoles, 3))
    number_of_dipoles = 0
    for i in range(-num,num+1):
        i2 = i*i
        x = i*dipole_diameter
        for j in range(-num,num+1):
            j2 = j*j
            y = j*dipole_diameter
            for k in range(-num,num+1):
                k2 = k*k
                z = k*dipole_diameter
                rad2 = (i2+j2+k2)*dd2
                if rad2 < sr2:
                    pts[number_of_dipoles][0] = x+1e-20
                    pts[number_of_dipoles][1] = y+1e-20
                    pts[number_of_dipoles][2] = z
                    number_of_dipoles += 1
    print(number_of_dipoles," dipoles generated")
    return pts
```

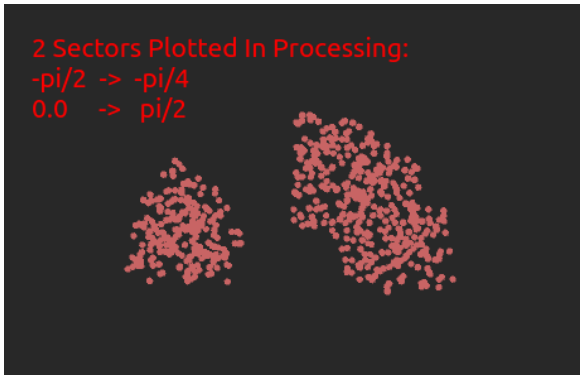
As far as I have been able to follow so far, the set of full particles in the simulation is specified inside the YAML file being considered (SingleLaguerre1 currently being tested, which also contains the other parameters such as beam used, the resolution of the dipole grid, the frame count to simulate up to, etc.).

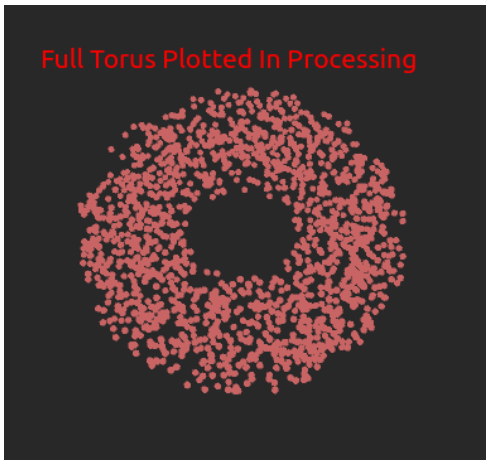
Each particle specifies its total size (and so the number of dipoles in each direction from the general dipole size specified for all particles), its refractive index, colour and centre position.

The set of these particles are then read inside the ***DipolesMulti2024Eigen.py*** main file and stored in a particle collection, which is a python object storing details about each particle. This object sets many of its variables (possibly all?) on initialisation (by considering the ***particle_spec*** supplied).

The ***simulation()*** is then run at the end of this main python file, which creates a list of particles (full particles), and considers the dipoles of each through a ***sphere_positions()*** function call. Therefore this appears to be primary palace in which dipoles are generated, and so a new generator can be made here for the arc version. Alongside this, I will need to implement an easier way for the plotter to read the type of primitive made (may need to be a new argument in particles YAML too), so it can then draw a corresponding shape.

In processing, a sample of the code used to generate torus' and torus sectors was tested, and as shown below worked as expected, and so can be simply transferred to the simulation. Note that this test generated random points in a cube and plotted them only if within the torus bounds, given by a simple closed equation, and so the simulation would not sample random points like this, but just sample on the lattice given, and added to the list if within the bounds.






```
boolean withinShape = pow( centre_R-sqrt( pow(point.x,2) + pow(point.y,2) ) ,2) +pow(point.z,2) <= pow(tube_R,2);
```

29/11/24

More functionality of the SH_DDA software has been tested. New particle materials can be added (Calcite was added with n=1.55), with different radii and dipole radii. As shown yesterday multiple particles can also be added easily. I also tested starting the particle closer to the ~1 micrometer stable height it reaches (started off at 0.75 micrometers), and it is still stably trapped there (perhaps slightly above where before).

Talking to our supervisor today about the project also highlighted an issue with the way we ran his program, where we did not include a ***"-O3"*** flag in the Dipole.o and Beams.o build step, which optimises this process seemed to give our program a big performance boost (~6s run time down to an ~1s run, when performing a calculation on 1 particle, 81 dipoles for 50 time steps), which also appears to have worked well on BC4 (not sure of exact speedup, we estimated somewhere between 2x and 4x, but will have to explore this a bit more).

Our plans now are to extend the particle generator to create dipoles in arc shapes, so we can test creating full and segmented ring segments, to see how they react when separated slightly. We also want to try this will particles and possibly cubes in a plane wave again. This requires either single timesteps to be run to extract the forces, or another small script to be written to quickly get forces for each setup and test other arrangements.

We then hope in the future to study the dynamics more using 2 counter-propagating bessel beams (an approach discussed early on before the Laguerre beam) to trap particles and hopefully encourage them to stay within a plane (as the Laguerre beam allows particles to drift in the Z axis until the reach a stable, or not, position). We could also pin them with a single bessel beam against a plate and observe their dynamics here two, but would have to add other frictional effects to the hydrodynamics force calculation.

28/11/24

The problem about the software not working yesterday was fixed by using the commands (in order)

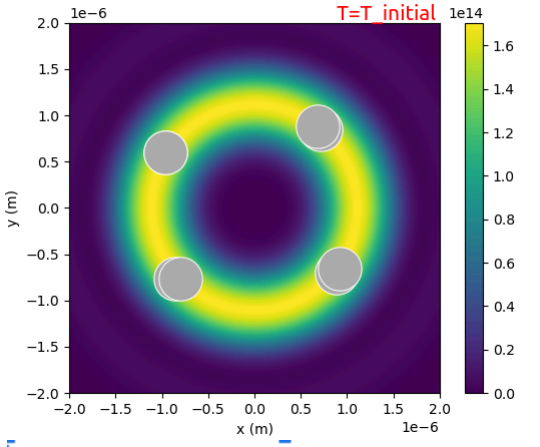
“

// g++-14 -std=c++14 -fopenmp -Wall -Wextra -pedantic -c -fPIC Dipoles.cpp -o Dipoles.o
// g++-14 -fopenmp -fPIC -c Beams.cpp -o Beams.o
// g++-14 -fopenmp -shared Dipoles.o Beams.o -o libDipoles.so
// g++-14 -shared -o libBeams.so -fPIC -fopenmp Beams.cpp ←Order of this shouldn't matter

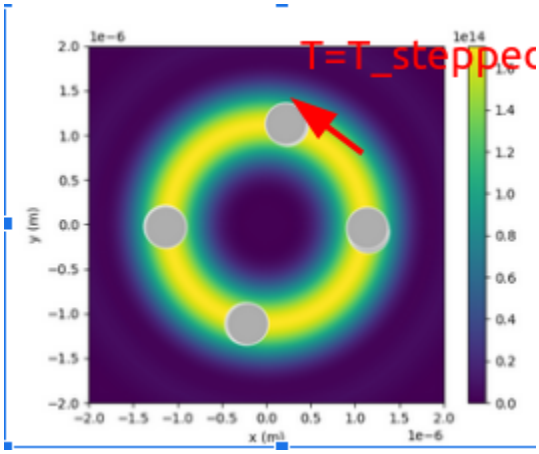
”

Where the main 2 fixes were giving the .so files different names to the origin .cpp, .hpp, .py, etc files (hence lib… added), and that the Beams.o had to be compiled manually too, as originally there was no command for it but it had already been compiled, but this was compiled on Mac NOT Linux.

Now the software is working, and we saw the test scenario of a trapped particle working, I attempted to add more particles which could simply be done through the addition of more particles in the ***“SingleLaguerre1.yml”*** file, which worked without issue, showing 4 particles trapped rotating CCW.



This approach requires the .yml file to be changed each run, and so when testing lots of steps these could also be made in advance and then called after each other in sequence to test as more particles are added to the system. It would be nice to have a method for generating these automatically (perhaps through editing the file in a python function, or less cleanly generating a copy each time to overwrite the previous with a new particle line added, the positions al shifted around the circle as performed in OTT), but this is not a huge priority.



and

We then agreed for my partner to start getting the 3D visualisation for the particles working (using the raw output data as oppose to already made 3D viewer, as this required Pyvista which is supposedly hard to version control), and I would try and get the program running on BC4 (Blue Crystal Phase 4 supercomputer) as I am already using Linux. We running this on BC4, I used a full node as BC4 has had problems before with numpy demanding full nodes as oppose to smaller fractions, and

```
[yy21991@bc4login2(BlueCrystal4) SH_DDA]$ cat slurm-10825551.out
Unable to load the Beams C++ library
[yy21991@bc4login2(BlueCrystal4) SH_DDA]$
```

so I am unsure as to whether Eigen (OpenMP parallelised linear equation solver) would do this too, hence to be on the safe side used the full node. Running this produced the error ***"unable to load the Beams C++ library"***. This is likely due to me only including the python module install in the sbatch file submitted (no C++ inclusion). WIth this now included in SBATCH script (seen on right), which when run produces another error, but essentially has the same problem. It is also worth noting that when ***"module load languages/Intel-OneAPI/2024.0.2"*** is run just on the login node, it states that it has installed ***"gcc/12.3.0"***, which in theory should not work with the version 14 needed for bessel functions in this program, which could also be preventing the file from being run. Another problem could be that I have not built the files on BC4 here (the Dipoles.o, Beams.o, libDipoles.so, libBeams.so) and so perhaps this is causing difficulty for it. Having tried building the files on BC4 first before running then (note that i had to replace g++-14 in the commands to g++, which is using g++ 12.3.0, and so errors are likely to occur but still worth trying anyway) I then find that the program surprisingly runs, but just encounters an error with the writing to an xlsx file. In order to add this module, we would have to ask Simon to have it included in BC4, so for now I will try disabling the output of data and just see if the program runs in full.

runSHDDA.sh

BC4 SBATCH script

```
#!/bin/bash
#SBATCH --job-name=runSHDDA
#SBATCH --partition=teach_cpu
#SBATCH --account=PHYS033184
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=28
#SBATCH --time=0:5:0
#SBATCH --mem=100M

# Load modules
module add languages/python/3.12.3
module load languages/Intel-OneAPI/2024.0.2
cd $SLURM_SUBMIT_DIR

# Run Program
python DipolesMulti2024Eigen.py SingleLaguerre1
```

```
[yy21991@bc4login2(BlueCrystal4) SH_DDA]$ cat slurm-10825622.out

:: WARNING: setvars.sh has already been run. Skipping re-execution.
   To force a re-execution of setvars.sh, use the '--force' option.
   Using '--force' can result in excessive use of your environment variables.

usage: source setvars.sh [--force] [--config=file] [--help] [...]
  --force        Force setvars.sh to re-run, doing so may overload environment.
  --config=file  Customize env vars using a setvars.sh configuration file.
  --help         Display this help message and exit.
  ...            Additional args are passed to individual env/vars.sh scripts
                 and should follow this script's arguments.

  Some POSIX shells do not accept command-line options. In that case, you can pass
  command-line options via the SETVARS_ARGS environment variable. For example:

  $ SETVARS_ARGS="ia32 --config=config.txt" ; export SETVARS_ARGS
  $ . path/to/setvars.sh

  The SETVARS_ARGS environment variable is cleared on exiting setvars.sh.

Unable to load the Beams C++ library
[yy21991@bc4login2(BlueCrystal4) SH_DDA]$
```

```
[yy21991@bc4login3(BlueCrystal4) SH_DDA]$ ls
Beams.cpp     Dipoles.py                ReadYAML.py            __pycache__
Beams.hpp     DipolesMulti2024Eigen.py  Simulation.py          libBeams.so
Beams.o       Display.py                SingleLaguerre1.vtf    libDipoles.so
Beams.py      Eigen                     SingleLaguerre1.xlsx   runSHDDA.sh
Dipoles.cpp   Output.py                 SingleLaguerre1.yml    slurm-10825535.out
Dipoles.hpp   Particles.py              SingleLaguerre2.yml    slurm-10825551.out
Dipoles.o     README-arm-build.txt      'Support Programs'     slurm-10825622.out
[yy21991@bc4login3(BlueCrystal4) SH_DDA]$ g++-14 -std=c++14 -fopenmp -Wall -Wextra -pedantic -c -fPIC Dipoles.cpp -o Dipoles.o
-bash: g++-14: command not found
[yy21991@bc4login3(BlueCrystal4) SH_DDA]$ g++ -std=c++14 -fopenmp -Wall -Wextra -pedantic -c -fPIC Dipoles.cpp -o Dipoles.o
Dipoles.cpp: In function 'Eigen::MatrixXcd dipole_moment_array(Eigen::MatrixXd, int, double, int, Eigen::VectorXcd, BEAM_COLLECTION*)':
Dipoles.cpp:585:104: warning: unused parameter 'dipole_radius' [-Wunused-parameter]
  585 | gen::MatrixXd array_of_positions, int number_of_dipoles, double dipole_radius, int number_of_dipoles_in_primitive, Eigen::VectorXcd inverse_polars, BEAM_COLLECTION* beam_collection){
      |                                                                                                    ~~~~~~~^~~~~~~~~~~~~

[yy21991@bc4login3(BlueCrystal4) SH_DDA]$ g++ -fopenmp -fPIC -c Beams.cpp -o Beams.o

[yy21991@bc4login3(BlueCrystal4) SH_DDA]$ g++ -fopenmp -shared Dipoles.o Beams.o -o libDipoles.so
[yy21991@bc4login3(BlueCrystal4) SH_DDA]$ g++ -shared -o libBeams.so -fPIC -fopenmp Beams.cpp
[yy21991@bc4login3(BlueCrystal4) SH_DDA]$ cat runSHDDA.sh
```

```
Traceback (most recent call last):
  File "/user/home/yy21991/SH_DDA/DipolesMulti2024Eigen.py", line 23, in <module>
    import Output
  File "/user/home/yy21991/SH_DDA/Output.py", line 6, in <module>
    import xlsxwriter
ModuleNotFoundError: No module named 'xlsxwriter'
[yy21991@bc4login3(BlueCrystal4) SH_DDA]$
```

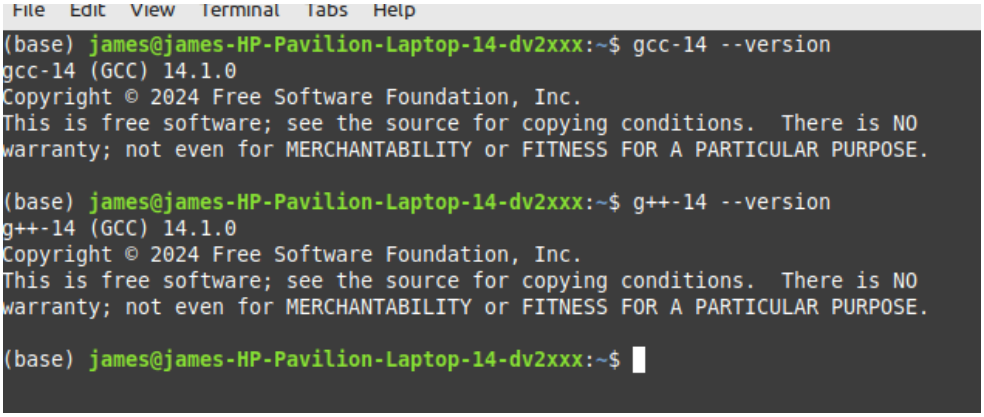
Preventing this output required me to remove the Output.py file and all mentions of it in the DipolesMulti2024Eigen.py file, but once this was performed the program ran on BC4 and took approximately 2.5 minutes to complete a simulation of 50 frames with 3 particles using all 28 cores a single node, which is not particularly quick, but may scale better with larger dipole numbers (only parallel part is the linear equation solver, which is a lot of work for the simulation, but is most dominating when there are many dipoles), and

```
 [-6.88102477e-13 -9.11492953e-13  2.34580873e-13]]
Step  40
40 [[-6.66864522e-13  9.74763759e-13  1.30772056e-13]
 [ 8.22375716e-13 -9.44985375e-13  7.04762684e-15]
 [ 6.78081334e-14 -1.06007372e-12  1.64085476e-13]]
Elapsed time: 152.680861 s
(201, 201)
[yy21991@bc4login3(BlueCrystal4) SH_DDA]$
```

offers another avenue of computation to be explored. We will still need the output in the end if we want to test in BC4, however for now knowing it works gives us options.

27/11/24

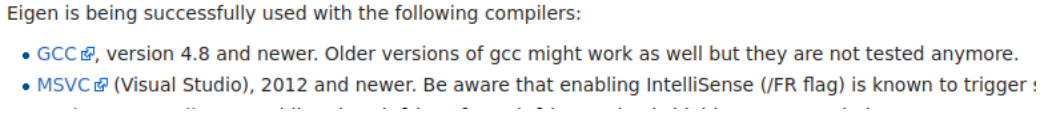
Having spoken to our supervisor today about his implementation of DDA software, we decided to try and run his home-made program for DDA (as talked about in our previous meeting) in order to try and get results for forces on multiple trapped particles. Simon had managed to achieve trapping of 1 particle in this approach, and had tested adding another particle of different refractive index to the system, however this would move upwards too far and eventually leave the beam (helix motion, then eventually leaves). The other particle would move in a helix roughly 1 micrometer up above the focus, where it would then stably travel in a circular motion.



This program records the forces as it calculates dynamics, and so we can compare how these forces change as more particles are added.

I also still intend to work in ADDA to allow another approach to be explored for this motion if the Laguerre beam and forces can work (which I am currently in the process of testing, by first comparing my calculations to official ADDA E field and force values in near and far fields).

Most of today has been spent attempting to get this software to work on my computer (a linux system, which Simon mentioned had worked for another student). My partner (on Mac) has managed to get the simulation calculating. The problem I am encounter is that my OS (Linux Mint XFCE) runs on Ubuntu 22.04 Jammy, which does not support g++-14 (which is required for bessel functions used in Laguerre beam). I have attempted work-arounds found online, however it is seeming likely that I will need to install a newer OS on my computer instead (Ubuntu latest versions uses Ubuntu 24.04, which does appear to support g++-14), but I will like to avoid this if at all possible as lots of reinstallation of programs will be required if this is done.



I have tried manually installing g++-14 through a tar.gz download, but it did not appear to fix the problem. However, after restarting my computer the issue was then fixed partially as running the command ***"g++ –version"*** now gave the correct 14.1.0 version of g++ (similarly for gcc –version), but ***"g++-14 –version"*** was not recognised. Exploring this more, it seemed that this is simply a linking step that is not performed when installing g++ manually, meaning the program

would run if I replaced all instances of ***g++-14*** with ***g++*** instead, but to avoid this the command ***"sudo ln -s /usr/local/bin/gcc /usr/local/bin/gcc-14"*** links the two, so now g++-14 works as expected. I can now try to install 'Eigen' (iterative solver) and build the program. Eigen states that is supported for gcc versions greater than 4.8, of which I believe g++-14 is version 4.9+ and so should work fine.

When running the 1st build command ***"***`g++-14 -std=c++14 -fopenmp -Wall -Wextra -pedantic -c -fPIC Dipoles.cpp -o Dipoles.o`***"*** this runs correctly with a few error messages (expected and not of concern). For the next build command, it used .dylib formats (supported by Mac not Linux) so I had to instead try convert this .so formats (shared object). This command, ***"***`g++-14 -fopenmp -shared Dipoles.o Beams.o -o Dipoles.dylib`***"*** did not work for either .dylib or .so initially, as I had just built the Dipoles.o file, NOT the Beams.o file (this was built originally on Mac therefore caused some issues when being used I believe, hence i used the command ***"***`g++-14 -std=c++14 -fopenmp -Wall -Wextra -pedantic -c -fPIC Beams.cpp -o Beams.o`***",*** which appeared to work). After rebuilding this file, the 2nd command worked and .dylib / .so file was built.

Now to run the python simulation the command ***"python DipolesMulti2024Eigen.py SingleLaguerre1"*** is run in a conda environment. Here I get the error `"Unable to find the specified library"` which occurs in the ***Beams.py*** file when searching for ***"***`ctypes.util.find_library("./Beams")`***"***. I think this requires a .so file to be found, and so maybe I need to do another build step for the beams as well as the dipoles.

25/11/24

For the plane wave, ADDA uses ***imExp()*** function, which may handle complex exponential calculation better than ***cexp()***, however once implemented the error still occurs.

```
    }
}
else for (i=0;i<local_nvoid_Ndip;i++) { // standard (non-surface) plane wave
    j=3*i;
    // can be replaced by complex multiplication (by precomputed phase factor), does not seem beneficial
    vSubtr(DipoleCoord+j,beam_center,r1);
    ctemp=imExp(WaveNum*DotProd(r1,prop)); // ctemp=exp(ik*r.a)
    cvMultScal_RVec(ctemp,ex,b+j); // b[i]=ctemp*ex
}
return;
case B_DIPOLE: {
```

Plane Wave ADDA

I have also tried manually setting ADDA to use other iterative solvers to see if the calculation can at least be completed. I have so far tested (as far as I am aware, via usage of the ADDA-gui program) all of ADDA's iterative solvers, namely;

NAME	***COMMAND***
-> **Bi-CGStab(2)**	-> ***-iter bcgs2***
-> **Bi-CG**	-> ***-iter bicg***
-> **Bi-CGStab**	-> ***-iter bicgstab***
-> **CGNR**	-> ***-iter cgnr***
-> **CSYM**	-> ***-iter csym***
-> **QMR**	-> ***-iter qmr*** *(default)*
-> **QMR2**	-> ***-iter qmr2***

Of these, **Bi-CGStab(2), Bi-CGStab, CGNR** and **CSYM** all gave non-null results, and all with essentially identical cross section values computed. The other solvers all encounter the same errors of **"<some factor> too small"**. Note here, in ADDA-gui it is stated that ***"All iterative solvers except CGNR and CSYM are susceptible to breakdowns. Although such situations are very improbable, they may happen in specific cases."***. This may mean CGNR and CSYM are naturally better choices anyway then, however I will have to investigate each of these methods individually to see why they may prove unsuitable over QMR (when they are not chosen by default).

```
RE_009 = 1.9052829958E-06  +
Cext    = 3.24330845e+22
Qext    = 9.105021475e+20
Cabs    = -548009.0909
Qabs    = -15384.39719
```

Bi-CGStab(2)

```
RE_009 = 6.9801312822E-06  +
Cext    = 3.243289144e+22
Qext    = 9.104967275e+20
Cabs    = -548008.7098
Qabs    = -15384.38649
```

BI-CGStab

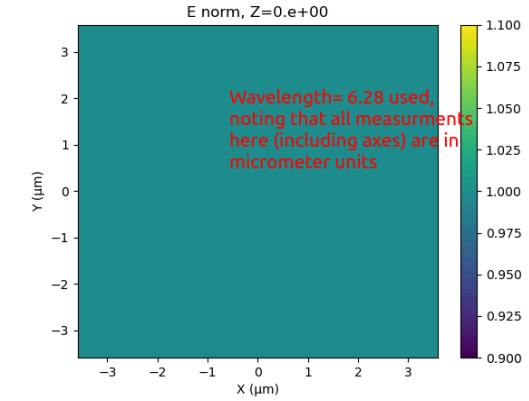
```
RE_024 = 9.0479691199E-06  +
Cext    = 3.24329218e+22
Qext    = 9.104975799e+20
Cabs    = -548008.8935
Qabs    = -15384.39165
```
CGNR

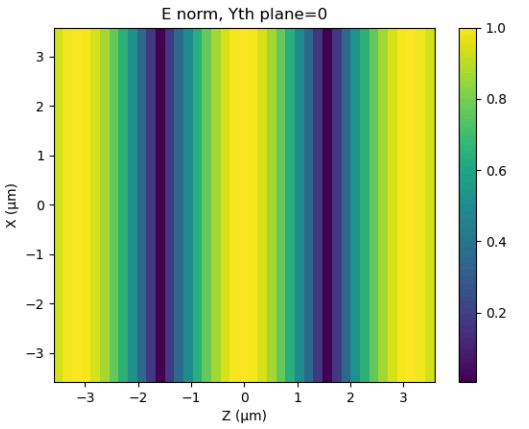
```
RE_046 = 9.0401585412E-06  +
Cext    = 3.243320053e+22
Qext    = 9.10505405e+20
Cabs    = -548009.0381
Qabs    = -15384.39571
(base) james@james-HP-Pavilion-Laptop-14-dv2xx
```
CSYM

I need to test if the dipole values for various operations (e.g. incoming, force, **polarisation (mainly this)**, etc) match to know that these methods give similar reliable outputs.

I have also checked and noted that other fields in ADDA, such as the B_PLANE, B_BES_*, etc, also multiply a real vector by complex by a complex factor, and so do have a complex field being output (stored in the **b[i]** list as expected) as expected, so ADDA should have no issues dealing with complex linear equations to solve.

When trying to implement the plane wave from ADDA into the python version, there was some confusion in the definition of **ex** used by ADDA. Initially I thought this was simply an easy reference for the X axis to quickly get E fields in different components, however **ex** is actually the incident polarisation of the tested wave polarisation (which I still also have some confusion over, as ADDA tests an **ex & ey**, which are perpendicular polarisations for the beam, both perpendicular to the propagation direction, and seemly I suppose therefore different to the polarization wanted to be computed, however I believe the point of this is to calculate the scattering matrix and so allow any polarisation to be found possibly). Ensuring I explicitly specified this polarisation in the python script (which should really have been implemented anyway due to its essential description of the beam) now gives expected plane wave form, and so hopefully a point of comparison to ADDA.





Currently code is being written to compare the ADDA incident fields and python found incident fields at each dipole, which can then be extended to the full E experienced at each, which can then allow forces at each to compared, which will then prove whether the force calculation I

have written is working correctly, which will then allow the Laguerre-Gaussian forces to be found with confidence that the method is not clearly in error.

21/11/24

Here I attempted to replace my custom bessel function with an inbuilt C version, as my supervisor pointed out to me that he had used ***"jn(order, value)"*** in his work for bessel functions. To the right is the list of included headers in this ***GenerateB.c*** file (in case I need to include another library to use this bessel function).

```
 * You should have received a copy of the GNU General Public License along with ADDA. If not, see
 * <http://www.gnu.org/licenses/>.
 */
#include "const.h" // keep this first
// project headers
#include "cmplx.h"
#include "comm.h"
#include "io.h"
#include "interaction.h"
#include "param.h"
#include "vars.h"
// system headers
#include <stdio.h>
#include <stdlib.h> // for abs()
#include <string.h>

// SEMI-GLOBAL VARIABLES

// defined and initialized in param.c
extern const int beam_Npars;
```

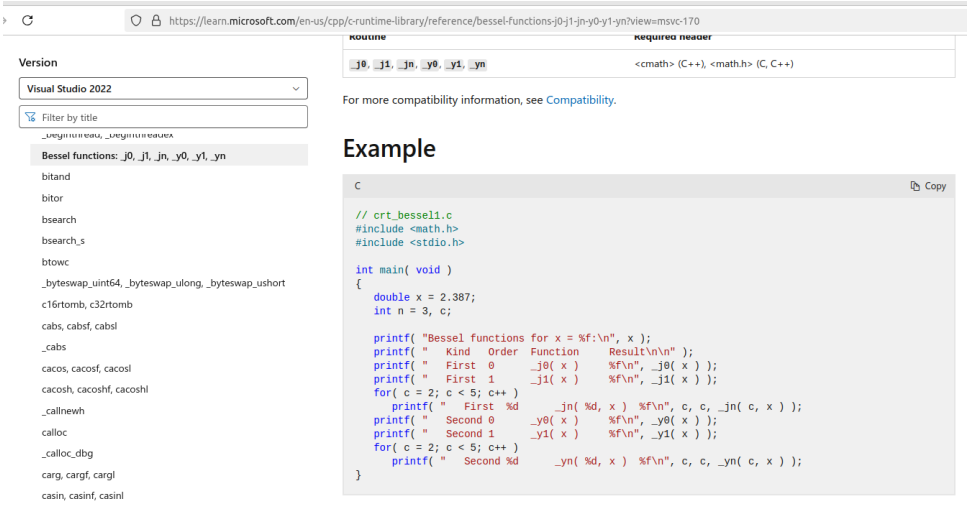
```
printf("bessel func jn(0,0)= %e \n",jn(0, 0));
printf("bessel func jn(1,0)= %e \n",jn(1, 0));
printf("bessel func jn(0,1)= %e \n",jn(0, 1));
printf("bessel func jn(3,10)= %e \n",jn(3, 10));
```

```
bessel func jn(0,1)= 1.197908e-32
bessel func jn(3,10)= 1.197908e-32
bessel func jn(0,0)= 1.197908e-32
bessel func jn(1,0)= 1.197908e-32
bessel func jn(0,1)= 1.197908e-32
```

When i try calling **jn()** with different parameters the program compiles correct and does not crash when running, but I seem always get a value ~0

I tried installing the GSL library using ***"sudo apt-get install libgsl-dev"*** which let me include gls in the project, but it still did not recognise the bessel function required, which I am believe would require me to include some path to gsl in the make step that adda uses (when I compile with ***make seq***) but this is particularly difficult as I do not know where ADDA has its make file.

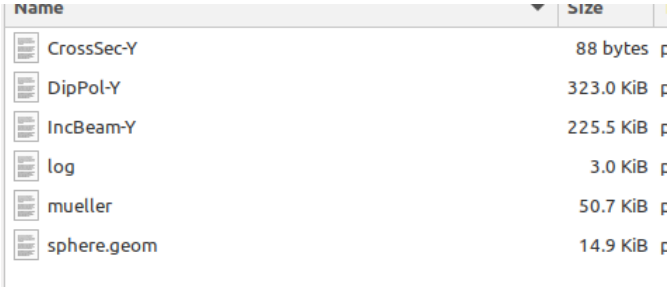
I tried including the ***#include <amth.h>*** header as well but this did not help with value produced by **jn()**. When trying to include ***#include <cstdlib>*** or ***#include <cmath>*** I also get the error "fatal error: cmath: No such file or directory". I also tried ***_j0(), _j1(), _jn()*** which are supposed to be from the math.h header, but these functions were also not recognised (included math.h).



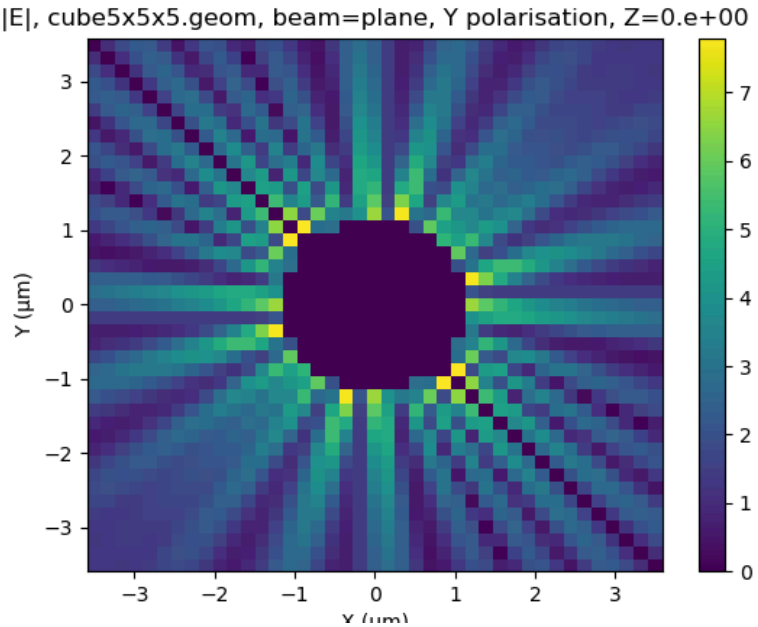
18/11/24

Work on the project temporarily stopped for a few days due to deadlines for assessed work in 2 other modules.

Picking up from where I last was, the complex form of the electric field for the Laguerre-Gaussian beam was not working as the complex iterative solver (QMR_CS) was not working (due to some small value detected). Hence I need to compare the real and imaginary parts separately, as they do work in the real iterative solver. This requires the far-field plotter in python which uses the polarisations found from ADDA. I can also use ***"-store_beam"*** to get the value of the incoming beam at the dipole positions, which could be used alongside the far field plotter for a better visualisation.

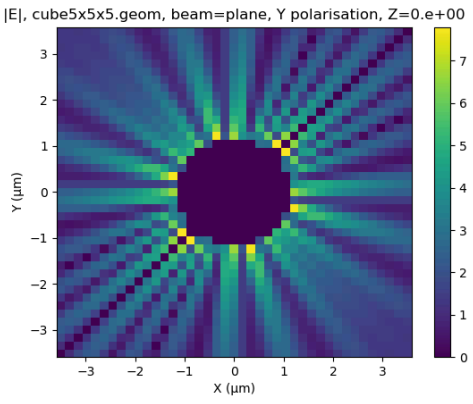


Using the 'Calculate_FarField_Grid()' function made before for this real part, as well as my 'Plot_2D_Array()' function I get the plot shown on the right. This is the far-field plot from using the polarisations. This does show similarities to the full E field plot (its mirroring about the y=x diagonal) but I now appreciate how the incident field really is necessary here in order to accurately judge if the effect is working. Hence my new idea is to plot the incoming field given from ADDA, but by taking a dipole large enough it completely engulfs the high intensity part of the laguerre beam (where that high intensity should be from our predictions with OTT at least). Then for fine enough resolution of the lattice it should show the incident light (directly from the Laguerre beam, no scattering contributions) so it can be seen if its profile is correct (if those high and low intensities are seen). Obviously the scattering from this system would be inaccurate as the particle is several times the wavelength, which would almost certainly be outside mie scattering range, but it would let us see the beam is the correct beam performing physically valid simulations. This approach could also be done with the plane wave to ensure our analysis is correct.



Equally, as expected the imaginary part gave the opposite symmetry for its plot. This is a good sign however the incident plot needs to be done to effectively analyse this data.

Care should be taken that what ADDA refers to as the ‘incoming field’ is the purely the beam used in the system with no scattering, as the term ‘incident’ can mean from the source beam and the scattered dipole beams, hence a similar idea may be true for incoming (likely not, but worth considering).



13/11/24

The ***"QMR_CS"*** error encountered yesterday is a problem with the iterative solver (of the set of linear equations formed to give the polarisations) with complex linear equations, called **Quasi Minimum Residual for Complex Symmetric systems**. A paper is linked inside the code for the method used here, however I would prefer to not slow down the process of this beam implementation by delving too deep into this side of ADDA's procedure. What can be seen here however is the error message encountered, which refers to absolute value after which errors occur, which here is 1^-10, where my program was giving a parameter ~10^-17, which is significantly far off. I changed my alpha and beta values to approximately (10^+-8)*(1+-i), where 10^-8 was used in my python version, but these changes seemed to have little to no effect on this parameter (changed by roughly a actor of *4 from the default (1+-i) error message).

```
998
999  ITER_FUNC(QMR_CS)
1000 /* Quasi Minimum Residual for Complex Symmetric systems, based on:
1001  * Freund R.W. "Conjugate gradient-type methods for linear systems with complex symmetric coefficient matrices",
1002  * SIAM Journal of Scientific Statistics and Computation, 13(1):425-448,1992.
1003  */
1004 {
1005 #define EPS1 1E-10 // for (vT.v)/(v.v)
1006 #define EPS2 1E-40 // for overflow of exponent number
1007     static double c_old,c_new,omega_old,omega_new,zetaabs,dtmp1,dtmp2;
1008     static doublecomplex alpha,beta,theta,eta,zeta,zetatilda,tau,tautilda;
```

```
1061             return;
1062         case PHASE_ITER:
1063             // check for very high omega (very small beta/||v||)
1064             dtmp1=1/(omega_new*omega_new);
1065             Dz("|vT.v|/(v.v)="GFORM_DEBUG,dtmp1);
1066             if (dtmp1<EPS1) LogError(ONE_POS,"QMR_CS fails: |vT.v|/(v.v) is too small ("GFORM_DEBUG").",dtmp1);
1067             // A.v_k; alpha k=v_k(*).(A.v_k)
1068             if (niter==1 && matvec_ready) { // uses that v_1=r_0/beta
1069                 temp1=1/beta;
1070                 nMultSelf_cmplx(Avecbuffer,temp1);
1071             }
```

```
File  Edit  View  Terminal  Tabs  Help
(base) james@james-HP-Pavilion-Laptop-14-dv2xxx:~/Desktop/Custom_Adda/adda/src/seq$ ./adda
-shape read cube5x5x5.geom -beam laguerre 0 8
all data is saved in 'run388_read_g5_m1.5'
box dimensions: 5x5x5
lambda: 6.283185307   Dipoles/lambda: 15
Required relative residual norm: 1e-05
Total number of occupied dipoles: 125
Memory usage for MatVec matrices: 0.1 MB
Calculating Green's function (Dmatrix)
Fourier transform of Dmatrix
Initializing FFTW3
Total memory usage: 0.1 MB

here we go, calc Y

CoupleConstant: 0.005259037198+1.843854149e-05i
local_nvoid_Ndip= 125
wavelength= 6.283185e+00
k = 1.000000e+00
w0= 3.769911e+00
integral_start= 0.000000e+00
integral_end  = 1.000000e+00
t1 = 4.000000e-09, 4.000000e-09
t2 = 4.000000e-09, -4.000000e-09
z_R= 7.106115e+00
t6 = -3.182994e-12, -3.295791e-11
t7 = -3.295791e-11, 3.182994e-12
t8 = -7.914291e-11, -6.610749e-11
x_0 = 0
RE_000 = NAN
Cext    = nan
Qext    = nan
Cabs    = 0
Qabs    = 0

here we go, calc X

CoupleConstant: 0.005259037198+1.843854149e-05i
local_nvoid_Ndip= 125
wavelength= 6.283185e+00
k = 1.000000e+00
w0= 3.769911e+00
integral_start= 0.000000e+00
integral_end  = 1.000000e+00
t1 = 4.000000e-09, 4.000000e-09
t2 = 4.000000e-09, -4.000000e-09
z_R= 7.106115e+00
t6 = -3.182994e-12, -3.295791e-11
t7 = -3.295791e-11, 3.182994e-12
t8 = -6.610749e-11, 7.914291e-11
x_0 = 0
RE_000 = NAN
Cext    = nan
Qext    = nan
Cabs    = 0
(base) james@james-HP-Pavilion-Laptop-14-dv2xxx:~/Desktop/Custom_Adda/adda/src/seq$
```

Reading the ADDA manual I found a section talking about its iterative solvers, and that their accuracy can be scaled with a **-eps <arg>** input when running the simulation (affecting 10^-<arg> minimal value allowed before crashing). Since I was getting errors with values ~10^-17, I set -eps 20 to get a 10^-20 minimum, but this

produced that exact same error message. I also get similar results for -eps 1, -eps 100 and was not allowed to perform -eps 10^-20 (expected since the negative is already included in the arg). ADDA also stated that issues can occur when solving for situations where geometry lines up exactly with parameters like the wavelength (e.g. cube width exactly equal to wavelength), hence I tried using different values for the ***dipoles per wavelength / -dpl***, which still resulted in the error even when trying non-integer values (such as -dpl 4.5, which when dealing with a 15x15x15 sphere should prevent exact line-up of values). After this I tried reading different shapes from .geom files (with default dpl) and found that this allowed the calculator to be performed (as shown in the screenshot here), however the results gave NaN outputs for the cross sections (when tried with a 5x5x5 cube and 5x5x5 sphere). When run with a 10x10x15 capsule (default dpl) this failed again due to a small |vT.v|/(v.v). It is worth noting that cases where this error has occurred have all had real, finite ***"RE_000"*** values in the log, which is the *"relative norm of the residual"* value in each step of the iterative solver, which is seen to only run once here as NAN is returned, indicating that the iterative solver was broken by NAN being found. This residual describes how close the approximation is to the true value, so it makes sense why it tending to infinity immediately would warrant the process being stopped.

is given (both total and for FFT part, §5). "`calc Y`" denotes beginning of calculation for y incident polarization. "`CoupleConstant`" is dipole polarizability, "`x_0`" denotes which initial vector is used for iterative solver (§12.1). After each iteration the relative norm of the residual is shown together with its relative decrease compared to the previous iteration (`progress`). A sign in between is one of `+`, `-` or `-+` indicating respectively that the residual is the smallest of all the previous iterations, larger than the previous one, and smaller than the previous one but not the smallest of all the previous. `log` finishes with timing information (§13.1). This file may contain more information depending on

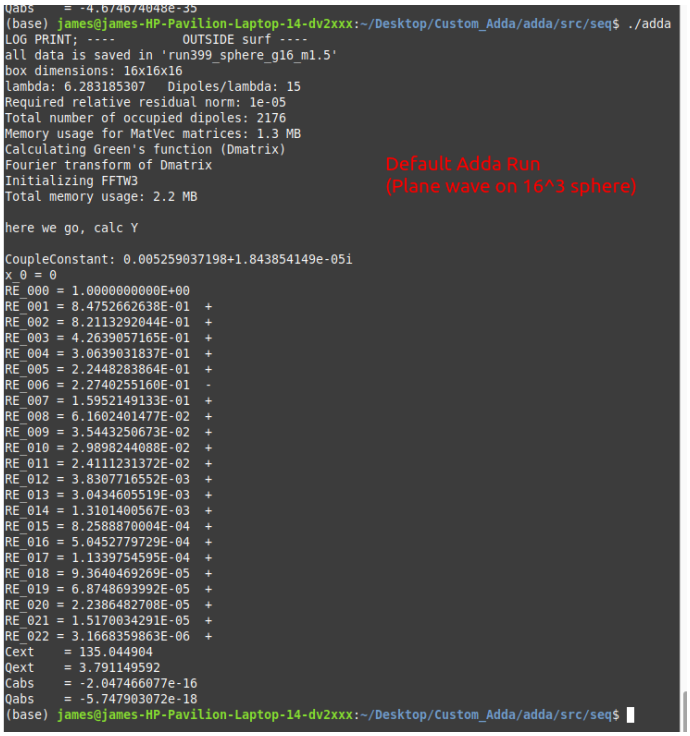
```
RE_12273 = 4.1410341239E-01  -
RE_12274 = 4.1410312431E-01  -+
RE_12275 = 4.1410319916E-01  -
RE_12276 = 4.1410291102E-01  -+
RE_12277 = 4.1410298580E-01  -
RE_12278 = 4.1410269758E-01  -+
RE_12279 = 4.1410277227E-01  -
RE_12280 = 4.1410248398E-01  -+
RE_12281 = 4.1410255858E-01  -
RE_12282 = 4.1410227025E-01  -+
RE_12283 = 4.1410234473E-01  -
RE_12284 = 4.1410205635E-01  -+
RE_12285 = 4.1410213107E-01  -
RE_12286 = 4.1410184263E-01  -+
RE_12287 = 4.1410191779E-01  -
RE_12288 = 4.1410162925E-01  -+
WARNING: (../iterative.c:1618) Iterations haven't converged in 12288 iterations.
Further calculated scattering quantities may be less accurate.
Cext     = 4.64288119e-17
Qext     = 1.303407727e-18
Cabs     = -1.665170135e-33
Qabs     = -4.674674048e-35
(base) james@james-HP-Pavilion-Laptop-14-dv2xxx:~/Desktop/Custom_Adda/adda/src/seq$
```

On Laguerre Run, with error ABS=1.0E-20, (Default is E-10)

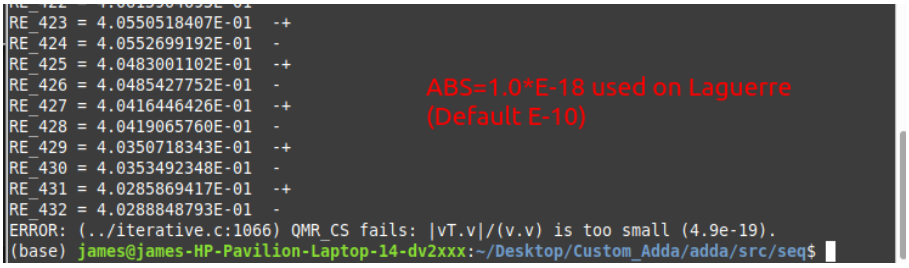
With the previous approach not working, I attempted to change the error value it used to be smaller than the 10^-17 value I was getting. However, this value may be below floating point accuracy (in python, epsilon value ~10^-15), resulting in errors which are important to bear in mind. Once this was done, the simulation did run and produce non-NaN outputs at the end, however the simulation ran for much longer (from far more steps in the iterative solver) and had far flower values for the cross sections. Most importantly it stated that it did not converge in this time and hence is set to cancel after ~12000 iterations, and so the value is not necessarily accurate anyway by doing this (as expected if below floating point accuracy). This does indicate that some value in my program is extremely small, leading to errors with the iterative solve and its outputs / is incorrectly configured. Since this was tested on 3 different shapes (of varying

sizes relative to the wavelength, using dpl), it seems that this is (more likely than not) NOT an error with wavelengths lining up at a specific point, and more likely a problem with the formulation.

Also note that when trying an error value closer to 10^-17 I had seen, the program simply ran for longer until it encountered another small value (10^-19), and so clearly did not help solve the problem either.



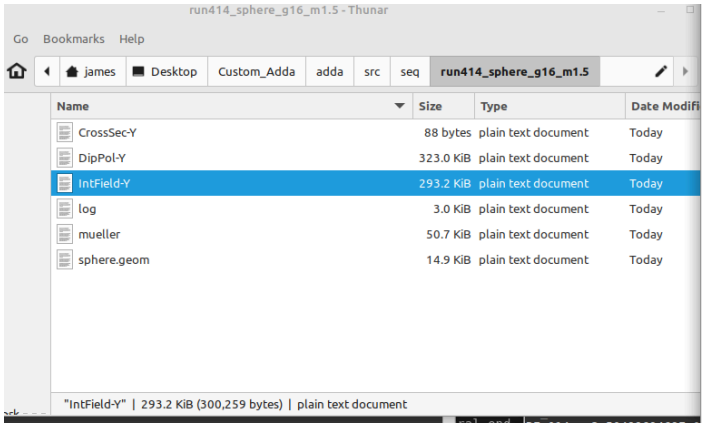
From this I have started considering the equation a bit more carefully to try and find any more mistakes present in it, and while testing I found this; when the real part of t6,t7 and t8 (the E_x, E_y and E_z components at a given dipole) are taken instead and stored in the **b** vector in ADDA (where the field is stored) for the 0th dipole (the one printed in the command line) you get pretty typical E field values (~10^-2 in each component) and the same is true for the imaginary parts too, but when finding the magnitude (sqrt(creal(t.)*cimag(t.))) I suddenly seemed to get a **nan** value for my t7 real part (0 for imaginary, as expected), which continued to occur even after multiple runs (not a null pointer picking random / broken values).

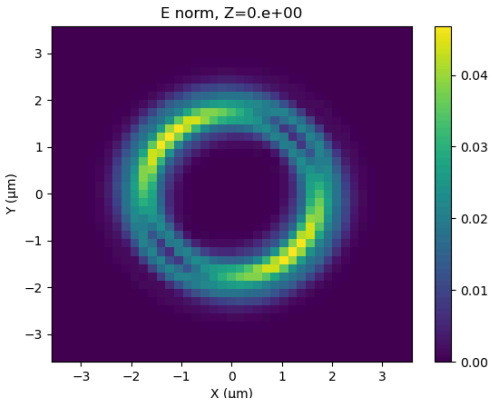


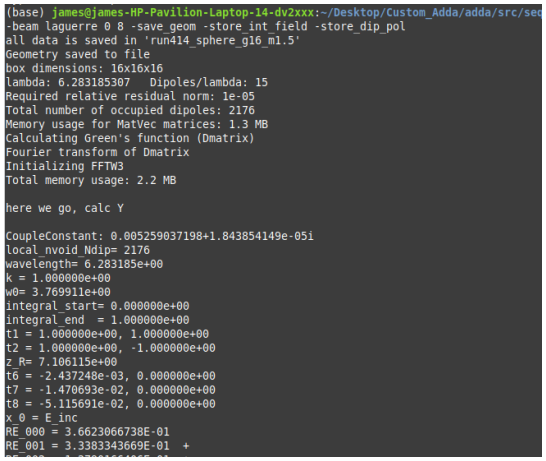
Therefore a similar effect may be happening on other dipoles, which could be causing extremely small all nan values elsewhere.

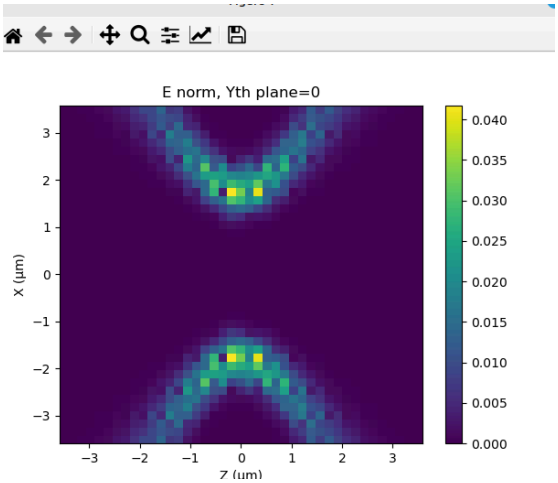
What can be tested now however is how well the real and imaginary parts of this field match python's version, which may help me find this other problem involved (could be due to the complex iterative solver now I think of it, as the real and imaginary parts computed fine when both made real doubles, hence the QMR solver was not used [see below for command line log]). This allows the internal fields and polarisations to be extracted, and so plotted in python. The beam figure shown is that of the real part of python's Laguerre beam implementation, and so I hope to see properties from this in the near field (will have to ensure the particle is large enough to have its internal space in the

high intensity part of the ring perhaps).

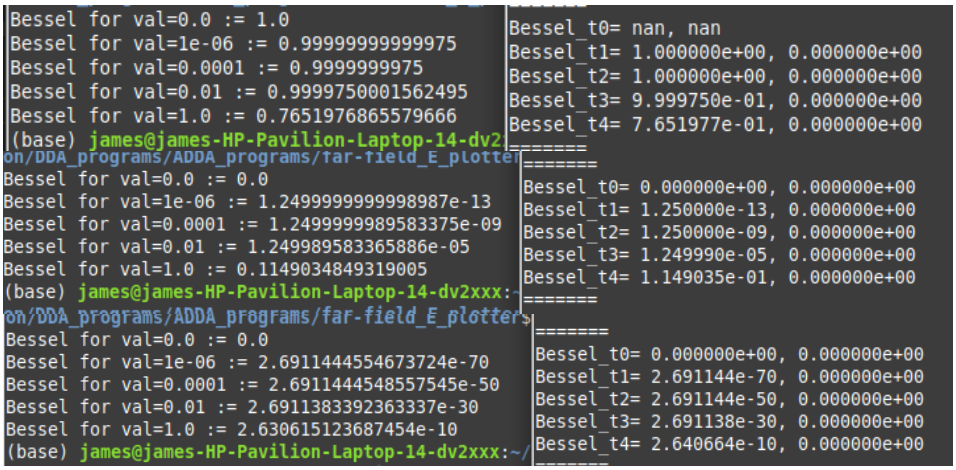








12/11/24



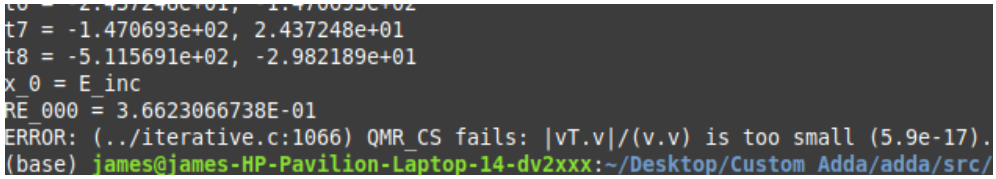
I have tested the results from the cplmn() function in python match the results in the ADDA implementation case where l=0, using just the legendre polynomials. The bessel functions also appear to match for for any order any value given (tests targeted value=0.0, 10^-6, 10^-4, 10^-2, 1.0, for order 0, 2, 10). This is using the first 6 terms of the taylor series for bessel, and as can be seen all results match very well.

After comparing each function output (Legendre, Bessel, normalising C, w parameter, u parameter)and obtaining matches in test scenarios, I then re-ran the program with its default values. Here I saw that ADDA wanted measurements in micrometres (e.g for the wavelength), which has lead to the output giving values of reasonable order and not ***Nan*** outputs (examples of reasonable outputs being electric fields in range of 10^-2 to 10^-12 when measured at different dipoles, rather than orders of 10^95 or extremely low orders simply being rounded to 0.000). I am now going to try and and store the internal fields so I can plot the E fields at each dipole and hopefully see that the Laguerre-Gaussian beam is of the correct form. However, an error is occurring in the calculation ***“QMR_CS fails: |vT.v|/(v.v) is too small (5.2e-17)”*** which prevents this additional data being stored. I have tried tuning the magnitude of the electric field to prevent this (increasing it to see if the error value becomes large enough to work) but this has not helped. To resolve this I am going to try and follow what command is doing, then work backwards from this to see where the laguerre beam fails. This effect could be linked to the implementation of the vorticity, as it is only ever used elsewhere in the ***B_BES*** beams, which appear to have some separation from the rest of the beams in ADDA’s case select function for which beam to generate (this is a guess however, only guided by the fact that most other aspects of beam generation are shared between ALL beams, and so hence singles this specific detail out as having potential to break).

I have also been talking to and helping my lab partner with his work trying to get a particle to trap in OTT. Recent issues that have arised link to the Maxwell-Stress Tensor calculation we manually perform, which has been giving some unusual forces. At first we thought this was mainly linked to the electric field differences incurred from changing Nmax, however increasing Nmax from say 10 to 15 (quite convergent from 15 onwards) only changed the values by a factor of 2/3 roughly (this being said, there are problems with OTTs implementation of this approximation, where the total power of the beam changes significantly as Nmax changes due to a large region of electric field appearing in the XZ plane, hence giving a larger total integral here). It turns out a large source of error is coming from the face integrated over (for cubic

bounding box around the spherical particle) that is close to the origin in the X=a plane (Y-Z integrated over), where a singularity at the origin caused by OTT's particle scattering causes extremely large values in this local vicinity. We partially overcame this problem by implementing a cutoff electric field, scaled based on the magnitude of the rest of the field, acting over some fixed radial distance that can be varied, which essentially sets an upper limit on electric field in that region (replaced with the maximum value if exceeding this limit). This could be better performed with some gradual smoothing function to prevent abrupt changed in E field magnitude, but for now this served in making the integral far more accurate (reduced the force observed at that point and another point further out in the field, which we would expect to have a force in range of roughly 10 times more/less, depending if sat on a peak or minima, from being an order of 20 magnitudes out to 6-8 orders of magnitude out, which is still far from perfect but now much better, and it is an effect that reduces significantly with distance from the origin, where here the face was 0.05 micrometers from the origin, as the particle sat at 0.45 and had 0.5 radius [in micrometers]).

Now that I have compared each part of the Laguerre beam E field in ADDA to the python version (and found matches for most) the field should be in the correct form in ADDA, and just requires this error about encountering small values to be fixed, then the field can be probed through the incident E field data saved by ADDA to ensure a Laguerre beam is seen, and then the force calculations that I have already made in python can be used (however, they first need to be tested using ADDA plane waves to ensure my force calc and ADDA's plane-wave-only force calc match, which should be simple to do as the exact formulation for ADDA's plane wave can be seen readily in the ***GenerateB()*** src file). After this trapping can try to be achieved in ADDA and compared to OTT.



11/11/24

The problem has now been narrowed down to an error within the ***"laguerre_getParam_E_component()"*** function, seemingly to do with having too many returns deeply nested, causing the error: ***"*** stack smashing detected ***: terminated Aborted (core dumped)"***. The previous segmentation error was fixed, and was caused by incorrectly defined pointers using the **bIndd_()** function. Currently, the ADDA code runs and generates outputs correctly when the above mentioned problem causing function is not used (and just tested with some fixed value), hence my focus is on finding the exact reason this is breaking. My first approach is to break up this function into each nested function call and see if any of these functions give invalid outputs, OR if they are all valid then the problem would appear to be from the nested nature of them, hence I can instead calculate them in a more 'inline' way (this seems unlikely however).

While printing each function involved in the laguerre_getParam_E_component() function, I came across 2 values which had mismatching data types (I had defined them as doubles, but could have been double complex values in reality). These changes seemed to fix the process, completing the full calculation with an output log. Hence the integral and E component calculation both are working now, with the exception that the functions for getting the associated legendre polynomial ***"laguerre_getParam_LIp()"*** and ***"laguerre_getParam_C()"*** were using fixed const values (for testing purposes). The value 'C' should be easy to find using the factorial function I wrote prior, but the associated legendre polynomial will require more effect to get working (possibly requiring recursion into its recurrence relation to find the Nth order). There also appears to have been further confusion in data types used, as parts of ADDA use both doublecomplex and double complex, and some of the temporary variables I have been using that were setup in the script originally use doublecomplex whereas I had treated them as double complex. This would clearly cause memory problems hence segmentation and stack problems.

After lots of investigating I have found that all the errors encountered were manifestations of the same problem, stemming from the ***"bjndd_()"*** function which, despite careful effort to have undefined pointers, seemed to have a dereferenced pointer or value within it which caused problems such as printf statements causing segmentation crashes & stack errors, for loops running for semi-random lengths before crashing, the type and length of variable names changing whether errors occurred or not, and many other similar examples. After removing this none of the unusual

n	$P_n(x)$
0	1
1	x
2	$\frac{1}{2}\left(3x^2-1\right)$
3	$\frac{1}{2}\left(5x^3-3x\right)$
4	$\frac{1}{8}\left(35x^4-30x^2+3\right)$
5	$\frac{1}{8}\left(63x^5-70x^3+15x\right)$
6	$\frac{1}{16}\left(231x^6-315x^4+105x^2-5\right)$
7	$\frac{1}{16}\left(429x^7-693x^5+315x^3-35x\right)$
8	$\frac{1}{128}\left(6435x^8-12012x^6+6930x^4-1260x^2+35\right)$

$$J_\alpha(x) = \sum_{m=0}^{\infty} \frac{(-1)^m}{m!\,\Gamma(m+\alpha+1)} \left(\frac{x}{2}\right)^{2m+\alpha}$$

behaviour is occurring anymore which is good, however it means I will have to manually implement a bessel function. For the time being, I will set this to the specific order where p=8 and l=0 (order 8 Laguerre Gaussian beam) as this is the order of the beam we are experimenting with to achieve trapping (if it does not cause problems with the values produced, however, I may generalise this sooner). The approach will be done for the associated legendre polynomial too for now (which has the same value as regular legendre polynomials when l=0, making it easy to write).

$$\Gamma(n) = (n-1)!$$

I have now implemented both of these, taking 6 terms of the Taylor series for the bessel function. However, the associated legendre polynomial is producing extremely large values (order 10^95) which is clearly occurring from the x^8 term, so I will need to check the order of the input values I am parsing into the function to see where the process is breaking down. The whole program is compiling and running correctly, it is just a matter of magnitudes of a few values now. The only other part of the ADDA beam process that I am unsure of now is whether/where to set the vorticity of the beam (described by the azimuthal component, p here, orbital angular momentum). Currently, I have set ***vorticity=p*** in the beamInit() function, however I would need to follow where all calls of this occur in order to see what effect it is actually having on the beam, since the electric field has been specified in full in the GenerateB() function below this, which you may expect to describe the vorticity through complex phase factors involved directly in its equation (several points where exp(i*phi) is used).

8/11/24

I have continued trying to fix the segmentation fault error, which I can now see is occurring inside my main function call to get the E field components, when calling the bessel function ***"bjndd_(n,x,d0,d1,d2)"*** (which makes sense as this is the first function I added that uses pointers, potentially allowing for unsafe behaviours).

Also, as a reminder I will put the equation for ADDA's polarisability of each dipole here, which needs to be added for the force calculation (ignored as the for now as the bigger priority was getting the beam working in ADDA, and the force calculation was left in state where force magnitudes were produced as expected, just with differing magnitude to expected from this lack of implementation). The radiative reaction term is important here (from complex value). The paper ***"Efficient computation of optical forces with the coupled dipole method"*** talks about some specifics of this and the force in DDA a bit more explicitly.

$$\overline{\boldsymbol{\alpha}}_i^{\mathrm{CM}} = \overline{\mathbf{I}} V_{\mathrm{d}} \frac{3}{4\pi} \frac{\varepsilon_i - 1}{\varepsilon_i + 2},$$

)ther polarizability formulatic
)ciated with finite size of the d

$$\overline{\boldsymbol{\alpha}}_i = \overline{\boldsymbol{\alpha}}_i^{\mathrm{CM}} \left(\overline{\mathbf{I}} - \overline{\mathbf{M}}_i \overline{\boldsymbol{\alpha}}_i^{\mathrm{CM}} / V_{\mathrm{d}} \right)^{-1},$$

in particular, $\overline{\mathbf{M}}^{\mathrm{CM}} = 0$. RR is a third-order (in kd) correction to the CM [42]:

$$\overline{\mathbf{M}}^{\mathrm{RR}} = (2/3)\mathrm{i}(kd)^3 \overline{\mathbf{I}}.$$

Whilst I have been implementing this beam into ADDA, my partner has continued to work in OTT and try to find parameters for trapping to occur. Correctional forces are now included and we have observed particle 'trapping' at single points on the axes, which looks like possible simulation errors (perhaps linked to the lower Nmax we are using, ~10). Plots have been constructed here to visualise the motion on planes at various starting points, and lower radii particles have seemed to work better in several of these cases.

7/11/24

```
case B_LAGUERRE:
    """
    . This 'GenerateB()' is called once (looped over every dipole point)
    . Returned at the end ONLY, where b is the E field returned (by reference, not directly)

    i       => each dipole index
    j=3*i   => jth vector start point
    ctemp   => value of E of beam at this point (store here temporarily) ---> doublecomplex type
                --> This is the magnitude of the E field essentially
    r1      => vector distance (double[3]) of dipole from origin (beam_center)
    DorProd(a,b) => dot product between two vectors (k=prop)
    DipoleCoord  => array of dipole coordinates;
    HENCE, DipoleCoord+j => coord of ith dipole -> [doublecomplex, doublecomplex, doublecomplex]
    cvMultScal_RVec()    => Multiplies a !doublecomplex SCALAR! by a !doublecomplex VECTOR!, and stores the result in a !doublecomplex VECTOR!
    b => the output LIST of VECTORS for each dipole's E field
    ex, ey, prop         => The X,Y,Z vectors for the electric field (assuming prop is always in Z direction, true by default and in the cases important here)
    """
    printf("LOG PRINT; ---- Gen beam: LAGUERRE Overall ---- \n");
    else for (i=0;i<local_nvoid_Ndip;i++) { // standard (non-surface) plane wave
        printf("LOG PRINT; ---- Gen beam; LAGUERRE Dipole---- \n");
        j=3*i;  //Pick out next start point (vector E on ith particle)
        vSubtr(DipoleCoord+j,beam_center,r1);   //<--- Give it the X coord index, it will pull out the +1 & +2 coords in this operation, find the diff with the
```

I have continued breaking up the code for a plane wave and identified what each variable is being used for in their example, and can now see how I would implement the Laguerre-Gaussian beam. There is a section at the top of the **GenerateB()** function that lets temporary variables be stored, allowing me to split up the calculation into several variables as done in python (as expected), and in python the beam is already split into components (hence will work nice with ***cvMultScal_RVec()*** multiplying the ex,ey or prop, where the prop does not appear to need normalisation either, as seen in **B_BES_TML**). Each of these components is formed through a series of complex multiplication (which as seen is already implemented in ADDA). That leaves the only difficulties coming from evaluating the (1) integral and getting a form for the (2) associated legendre polynomial of given order and (3) bessel functions of given order. I have already seen a fortran routine ***"subroutine bjndd (n, x, bj, dj, fj)"*** that calculates bessel functions, implying that other well known parameters like legendre polynomials may be included too, however I have not been able to find it. The associated legendre polynomials can be

$$P_l^m(x) = (-1)^m \cdot 2^l \cdot (1-x^2)^{m/2} \cdot \sum_{k=m}^{l} \frac{k!}{(k-m)!} \cdot x^{k-m} \cdot \binom{l}{k} \binom{\frac{l+k-1}{2}}{l}$$

$$P_\lambda^\mu(z) = \frac{1}{\Gamma(1-\mu)} \left[\frac{1+z}{1-z}\right]^{\mu/2} {}_2F_1(-\lambda, \lambda+1; 1-\mu; \frac{1-z}{2})$$

where Γ is the gamma function and ${}_2F_1$ is the hypergeometric function

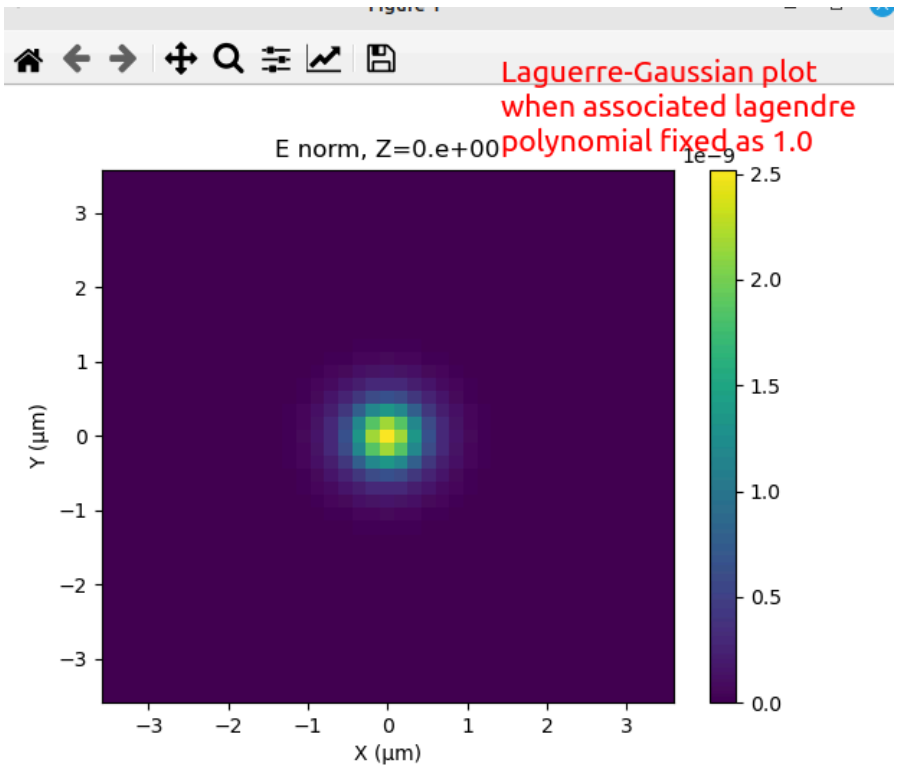
$${}_2F_1(\alpha,\beta;\gamma;z) = \frac{\Gamma(\gamma)}{\Gamma(\alpha)\Gamma(\beta)} \sum_{n=0}^{\infty} \frac{\Gamma(n+\alpha)\Gamma(n+\beta)}{\Gamma(n+\gamma)\,n!} z^n,$$

```
double laguerre_getParam_w(double zeta, double w0) {
    /*
    . Get the 'w' term used in laguerre-gaussian beam
    */
    return w0*sqrt(1.0+pow(zeta,2));
}
double laguerre_getParam_Llp(double calue, int l, int p) {
    /*
    . Get associated legendre polynomial of given order
    */
    return 1.0; //#### IMPLEMENT THIS ####
}
double laguerre_getParam_C(int l, int p) {
    /*
    . Get normalisation constant C, used in u_l_p function
    . From "Gaussian Beams In Optics of Course" paper
    */
    return (factorial(p))*sqrt( (2)/(PI*factorial(p)*( factorial( abs(l) + p ) )) );
}
double laguerre_getParam_Ulp(double C, int l, int p, double rho, double phi, double WaveNum, double z, double z_R, double w0) {
    /*
    . Get u_l_p, which describes behaviour of laguerre-gaussian beam
    */
    return ( (C)/sqrt(1 +(pow(z,2)/pow(z_R,2))) )*pow( (rho*sqrt(2))/(laguerre_getParam_w(z, w0)), l)*(laguerre_getParam_Llp((2.0*pow(rho,2))/(pow(laguerre_getParam_
}
double laguerre_getParam_E(double kappa, double rho, double phi, double z, double WaveNum, int l, int p, double z_R, double w0) {
    /*
    . Get E magnitude term for laguerre-gaussian beam
    */
    double C = laguerre_getParam_C(l, p);
    return laguerre_getParam_Ulp(C, l, p, rho, phi, WaveNum, z, z_R, w0)*cexp( (-WaveNum*pow(kappa,2)*z_R)/(2.0*(pow(WaveNum,2) - pow(kappa,2))) )*pow( (pow(kappa,2)
}
double laguerre_getParam_E_component(double complex kappa, int component, double rho, double phi, double z, double WaveNum, int l, int p, double z_R, double w0, doub
    /*
    . Returns magnitude of the ith component of E (x,y or z)
    */
    if(component == 0) {
        double temp_v0 = kappa*rho;
        double* temp_v1;
        double* temp_v2;
        double* temp_v3;
        //void bjndd (const int *n,const double *x,double *bj,double *dj,double *fj);
        bjndd (&l, &temp_v0, temp_v1, temp_v2, temp_v3 );      //Bessel function, only care about t2, not t3 or 4  //##### CAREFUL OF TYPES FOR THIS RETURN #####
        return laguerre_getParam_E(kappa, rho, phi, z, WaveNum, l, p, z_R, w0)*cexp(I*l*phi)*cexp(I*z*sqrt(pow(WaveNum,2)-pow(kappa,2)))*( alpha*(*temp_v1) );
```

found through a closed form expression (on the right), but this closed form may not extend to the complex case (also shown). I could also calculate this using the recurrence relations and recursively calling a function to get the desired order. For the integral, it can be numerically calculated. I would have to implement my own version of this as scipy will not be available here (perhaps another library could be used, but I am hesitant as to whether it will work nicely when brought into ADDA).

I have now implemented functions to get the core complex values, which uses the fortran routine to fetch bessel functions. The complex numerical integration is also now written (basic simpson's rule, which can be changed once the process is confirmed working and if we feel the process requires more accuracy). The only section not written is the calculator of the associated legendre polynomials (for now just returns 1.0+0.0*I), however we should still get a positive result with a beam out (as pictured to the right). Currently ADDA cancels with the following error on its first dipole iteration; ***"Segmentation fault (core dumped)"***, which seems to suggest a vector used was not initialised properly (could be one of the temp. vectors used).
Briefly testing this shows that the error occurs during the integral, and so each variable will now need to be further tested to see which is causing this core dump problem.

```
File  Edit  View  Terminal  Tabs  Help
(base) james@james-HP-Pavilion-Laptop-14-dv2xxx:~/Desktop/Custom_Adda/adda/src/s
eq$ ./adda -beam laguerre 4 4
LOG PRINT; ---- RUN BEAM INIT ----
LOG PRINT; Laguerre Init-> Start of case
LOG PRINT; Laguerre Init-> Mid of case
LOG PRINT; Laguerre Init-> End of case
all data is saved in 'run023_sphere_g16_m1.5'
box dimensions: 16x16x16
lambda: 6.283185307   Dipoles/lambda: 15
Required relative residual norm: 1e-05
Total number of occupied dipoles: 2176
Memory usage for MatVec matrices: 1.3 MB
Calculating Green's function (Dmatrix)
Fourier transform of Dmatrix
Initializing FFTW3
Total memory usage: 2.2 MB

here we go, calc Y

CoupleConstant: 0.005259037198+1.843854149e-05i
LOG PRINT; ---- Gen beam: LAGUERRE Overall ----
LOG PRINT; ---- Gen beam; LAGUERRE Dipole----
Segmentation fault (core dumped)
(base) james@james-HP-Pavilion-Laptop-14-dv2xxx:~/Desktop/Custom_Adda/adda/src/s
eq$
```



```
printf("LAGUERRE LOG;  t2= %d \n",t2);
printf("LAGUERRE LOG; z_R= %d \n",z_R);

printf("LAGUERRE LOG; Valid To Middle \n");

//laguerre_getParam_E_component(double complex k
t6 = integrate_simpsons_rule(0.0+0.0*I, WaveNum*
t7 = integrate_simpsons_rule(0.0+0.0*I, WaveNum*
t8 = integrate_simpsons_rule(0.0+0.0*I, WaveNum*

//printf("LAGUERRE LOG; t6=E_x= %d, %d \n",creal
//printf("LAGUERRE LOG; t7=E_y= %d, %d \n",creal
//printf("LAGUERRE LOG; t8=E_z= %d, %d \n",creal

printf("LAGUERRE LOG; Valid To Near End \n");

i=3*i;
```

```
Terminal - james@james-HP-Pavil
File  Edit  View  Terminal  Tabs  Help
LAGUERRE LOG; rho= 1
LAGUERRE LOG; phi= 0
LAGUERRE LOG;   z= 0
LAGUERRE LOG;  t1= 0
LAGUERRE LOG;  t2= 0
LAGUERRE LOG; z_R= 0
LAGUERRE LOG; Valid To Middle
Segmentation fault (core dumped)
(base) james@james-HP-Pavilion-Laptop-14
```

```
static void AllocateEverything(void)
// allocates a lot of arrays and performs memory analysis
{
    double tmp;
    size_t temp_int;
    double memmax;

    // redundant initialization to remove warnings
    temp_int=0;

    /* It may be nice to initialize all pointers to NULL here, so that any pointer, which is not initialized below, will
     * surely stay NULL (independent of a particular compiler). But even without this forgetting to allocate a necessary
     * vector, will surely cause segmentation fault afterwards. So we do not implement these extra tests for now.
     */
    // allocate all the memory
    tmp=sizeof(doublecomplex)*(double)local_nRows;
    if (!prognosis) { // main 5 vectors, some of them are used in the iterative solver
```

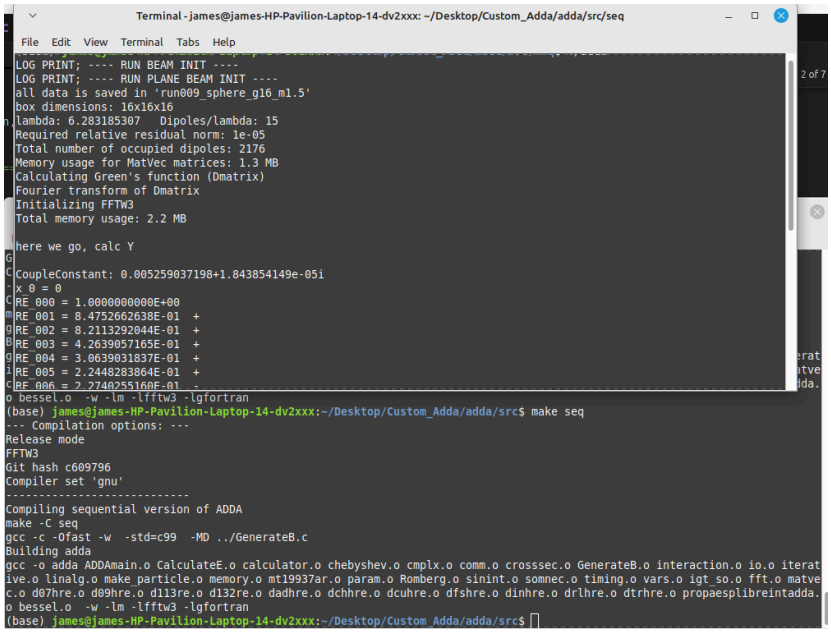
6/11/24

The interaction term from my last work needs to be adjusted to fix its magnitude, however it seems to work in principle so now I will start working back on the more difficult challenge of getting the beam working in ADDA. I am going to approach this by trying to pick apart existing beams in the program with print statements to figure out what each arable is supposed to represent (as many go unexplained, hence making the process confusing). This will require re-compilation of the program each time a change is made. In Linux, this requires the following commands listed to be run in the ***adda/src*** folder of the ADDA software.

```
sudo apt install gcc gfortran libfftw3-dev
make seq
sudo apt install libopenmpi-dev
make mpi
sudo apt install ocl-icd-opencl-dev libclfft-dev
make ocl
```

I have re-downloaded ADDA to reduce the chance of errors from old changes to the previous version I was using when testing ideas out. As I make changes to each file needed in beam generation I am running compiling the code with ***'make seq'*** to ensure nothing is breaking along the way.

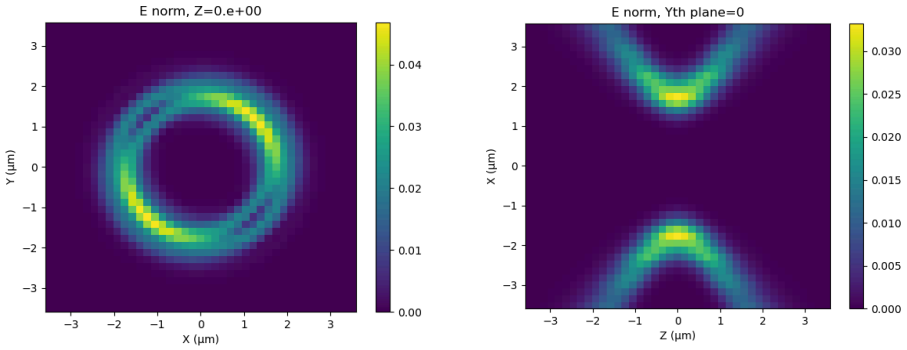
While performing these test runs, I found that polarisability is set by the 'couple constant' in ADDA, which is printed in log. Also it's worth reminding myself that '-store_beam' DOES give the incident beam E field, but only at the dipole positions, hence is not useful for far-field scattering, but could be used for near-field if wanted.

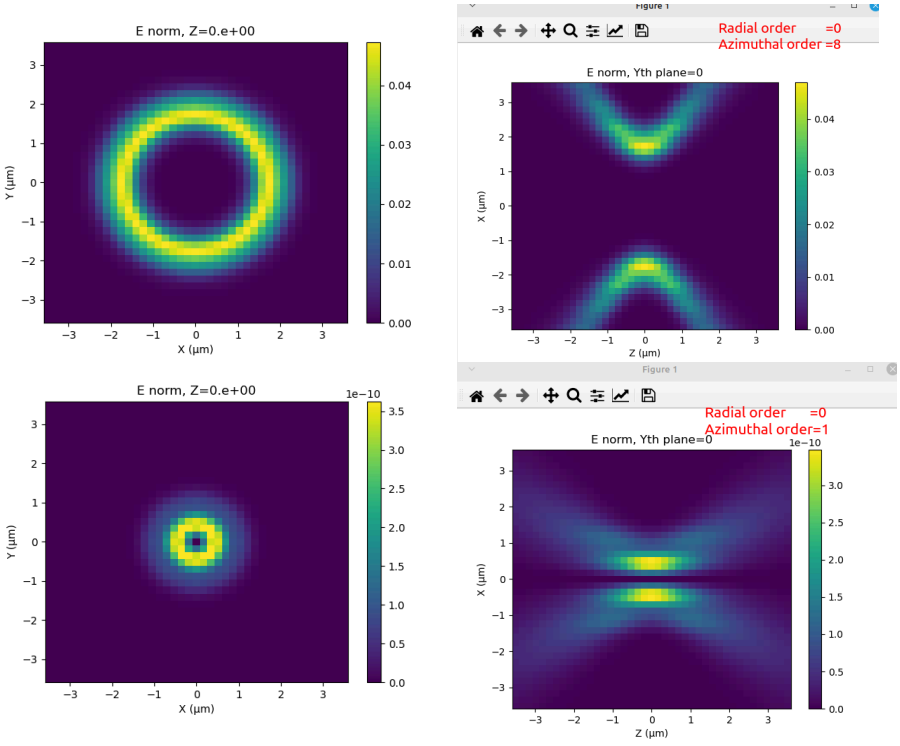


Using regular ***printf()*** commands I could find that default ***./adda*** uses a plane wave, and so I will test this first (however I could also easily specify any other type of beam if I wanted to test its components, which I will later want to do for beams that use non-zero vorticity, like any of the bessel B_BES_… beams).

5/11/24

Thinking about my last statement in the lab book yesterday I didn't really agree with what I said and thought that the integral not being evaluated properly in the complex plane could cause some much bigger problems,and after looking at the sci.integrate.quad function again I saw that it actually does have a the ability to evaluate complex integrals (which otherwise would have required me to manually separate the real and complex part then evaluate two real integrals and combine afterwards, which may also have had issues fundamentally). This immediately lead to a plot in ZX plane that looked better (bright high intensity range, more similar to OTT), but with a separate more unusual pattern in the XY plane. This however still clearly looked like an imbalance in the X Y forces, and so it was quite simple to try the complex values alpha=1+i and beta=1-i to then start seeing circular rings.





The figures show order 8 for alpha=beta=1,
order 8 for alpha=C*(1+i), beta=C*(1-i),
Order 1 alpha=C*(1+i), beta=C*(1-i), where C is scaling constant to the get the E magnitude to match OTT, which is 0.5*10^-8 for all plots here.

Now that these plots appear to be working properly, I will try to implement a force calculator using the field, so when this beam is added to ADDA its forces can be found. I am using the paper ***"Time-averaged total force on a dipolar sphere in an electromagnetic field"*** to implement the forces.

This paper talks about the forces involved, and here the change in momentum can be seen, which is a possible avenue where the Abraham-Minkowski controversy could be tested (both formulations for momentum used, see how simulation changes from change in force found, hence see if one produces closer results to reality + can also try combination suggested by Steven Barnett)

> the dipole approximation (cf. Ref. 1): a gradient force $(\mathbf{p}\cdot\nabla)\mathbf{E}$, which is essentially due to interaction of the particle-induced dipole moment $\mathbf{p}$ with the electric field $\mathbf{E}$ and scattering and absorbing forces $^1/_c\,\dot{\mathbf{p}}\times\mathbf{B}$, where $\mathbf{B}$ is the magnetic vector, $\dot{\mathbf{p}} = \partial\mathbf{p}/\partial t$, and c is

It is also mentioned that for many frequencies the scattering/absorption forces are not observed (only a T averaged force), but the scattering cross section (assume non-absorbing => C_abs=0) is given by ADDA so this could be used.

$$\mathbf{F} = \frac{|E|^2}{(8\pi)}\left(C_{\text{abs}} + C_{\text{scat}}\right)\frac{\mathbf{k}}{k},$$

The time averaged force for the ith component is given by the equation to the right (in terms of just E field). This gives the force acting on **each dipole** in the simulation. This uses einstein notation, so the j is the sum over the components of the **E** vector field present at the dipole in question (not summing over every other dipole or similar, however the **E** experienced at this point will involve a sum over the others, according to E_j=E_j_inc + sum(other dipole contributions)). ADDA gives the internal field at each dipole, which (as described in ADDA here) includes the contribution from the beam, the dipoles near this dipole, and the target dipole itself (whereas the excited field does not include the dipole itself). Therefore this can be used directly with the **E** talked about here.

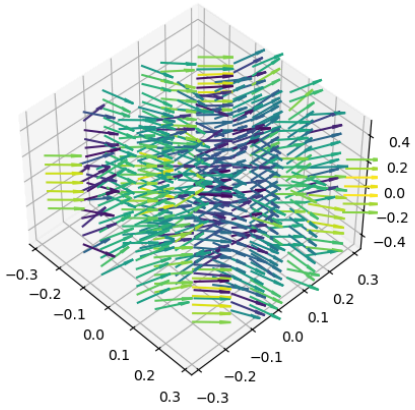
$$\langle F^i\rangle = (1/2)\mathrm{Re}[\alpha E_{0j}\partial^i(E_0^j)^*].$$

$$\mathbf{P}_i = \overline{\boldsymbol{\alpha}}_i\mathbf{E}_i^{\text{exc}} = V_{\text{d}}\chi_i\mathbf{E}_i,$$

$$\overline{\boldsymbol{\alpha}}_i^{-1}\mathbf{P}_i - \sum_{j\neq i}\overline{\mathbf{H}}_{ij}\mathbf{P}_j = \mathbf{E}_i^{\text{inc}}, \qquad (20)$$

where $\mathbf{E}_i^{\text{inc}}$ is the incident electric field (§9.2), $\overline{\boldsymbol{\alpha}}_i$ is the dipole polarizability (self-term), $\overline{\mathbf{H}}_{ij}$ is the total interaction term (Green's tensor of the environment), and indices i and j enumerate the dipoles. In the free-space mode $\overline{\mathbf{H}}_{ij} = \overline{\mathbf{G}}_{ij}$ (direct interaction), while in the surface mode (§7) it additionally contains reflection term $\overline{\mathbf{R}}_{ij}$ ($\overline{\mathbf{H}}_{ij} = \overline{\mathbf{G}}_{ij} + \overline{\mathbf{R}}_{ij}$). The (total) internal electric field $\mathbf{E}_i$ is the one present in a homogenous particle modeled by an array of dipoles, also known as macroscopic field [41]. It should be distinguished from the exciting electric field $\mathbf{E}_i^{\text{exc}}$ that is a sum of $\mathbf{E}_i^{\text{inc}}$ and the field due to all other dipoles, but excluding the field of the

Using this approach the forces can be found at the dipole positions as pictured below. Note however that this graph is NOT showing the force of Laguerre beam on the particle (5x5x15 capsule in this case), as this ADDA calculation, which gave the internal E field, which allowed the force to be found, was performed with a plane wave. This plot is just proving how the forces would be found at each dipole by evaluating



the Laguerre beam at beam. This being said, when using the data just provided by ADDA it will not be possible to evaluate the derivative, hence the force calculation here will not work. The process being used currently is also not strictly correct as this is just evaluating the incident beam at each dipole position, not accounting for the internally scattered terms. Therefore it seems best to externally calculate this scattering too, and only rely on ADDA's polarisation output (in order to find these scattered interaction terms). This will require the Green's tensor to be evaluated.

$$\mathbf{E}(\mathbf{r}_i) = \frac{\mathbf{P}_i}{\varepsilon_0 V_i \chi_i} \quad \text{internal}$$

$$\mathbf{E}(\mathbf{r}) = \mathbf{E}^{\mathrm{inc}}(\mathbf{r}) + \frac{k^2}{\varepsilon_0} \sum_i \bar{\mathbf{G}}(\mathbf{r}, \mathbf{r}_i) \cdot \mathbf{P}_j \quad \text{near}$$

$$\mathbf{E}^{\mathrm{sca}}(\mathbf{r}) = \frac{\exp(\mathrm{i}kr)}{-\mathrm{i}kr} \mathbf{F}(\mathbf{n}) \quad \text{far}$$

$$\bar{\mathbf{G}}(\mathbf{r}, \mathbf{r}') = \bar{\mathbf{G}}(\mathbf{R}) = \frac{\exp(\mathrm{i}kR)}{4\pi R}\left[\left(\bar{\mathbf{I}} - \frac{\mathbf{R} \otimes \mathbf{R}}{R^2}\right) + \frac{\mathrm{i}kR - 1}{k^2 R^2}\left(\bar{\mathbf{I}} - 3\frac{\mathbf{R} \otimes \mathbf{R}}{R^2}\right)\right]$$
$$\boldsymbol{R} = \mathbf{r} - \mathbf{r}'$$

The reason why ADDA can calculate plane wave forces despite this is that plane waves forces can be reduced to an equation in terms of just the scattering cross section (explained in paper link at the start of today), which is a quantity produced by ADDA. Otherwise, derivatives of the E field are required, which are not provided, and so is the reason for its exclusion in ADDA I would presume.

I have started to write this part of the program, but am encountering unusual values for the interaction term, which is giving obviously large E values (of order 10^10, when the incident beam is showing orders of 10^-6). THis could be linked to the k^2/e_0 factor out the front, where perhaps I have included too many k factors (each of micrometre order, therefore 1/ giving orders of 10^6, hence ~10^12 if squared for example).

4/11/24

My current list of jobs to do are (1) vary the 3 undefined constants in the Laguerre beam generated yesterday and try to get the high intensity ring to appear at a similar radial distance as generated in OTT, (2) try to get a continuous uniform ring intensity rather than the ‘dotted’ look currently seen, (3) make a plotter for the XZ plane for varying Y (not just the XY planes for varying Z) so we can see the other profile of the beam for comparison with OTT, (4) add the beam to ADDA through ‘src’ files, (5) calculate the forces experienced by each by each dipole and sum for a total rigid force, then finally (6) compare setups in ADDA to OTT to see get trapping in both and compare if they occur in the same conditions or if the models give differing trapping parameters.

After the meeting with our supervisor, my partner has also implemented a drag and brownian motion force into the OTT dynamics simulation, in order to better capture the motion of the particle (which would have likely been suspended in water or possibly just air). The drag force also allows us to approximate the particle quasi-statically, meaning it is assumed to come to a complete stop every timestep (given it is not too small, currently ~1e-4s), which usually make a simulation more stable (a problem we were encountering where the particle would quickly jump away after reach some z height).

When testing different parameters, C seemed to have more effect than expected (expected no effect, perhaps slight noise from pyplot plotting, as all E fields scaled by the same C). Hence the following example was considered, showing that the error appears to be from the numerical integration accuracy.

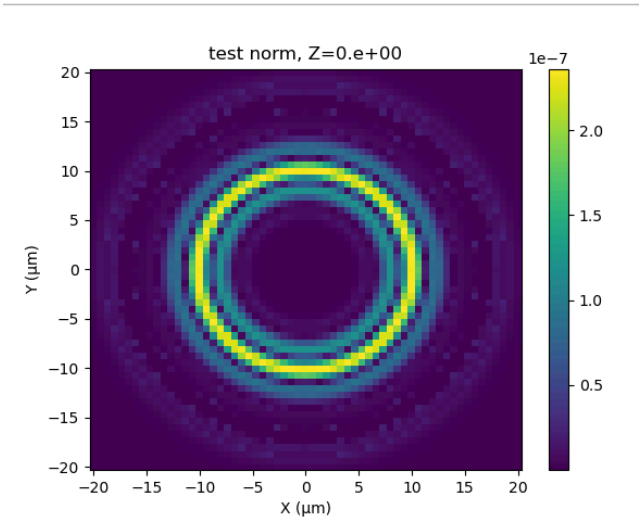
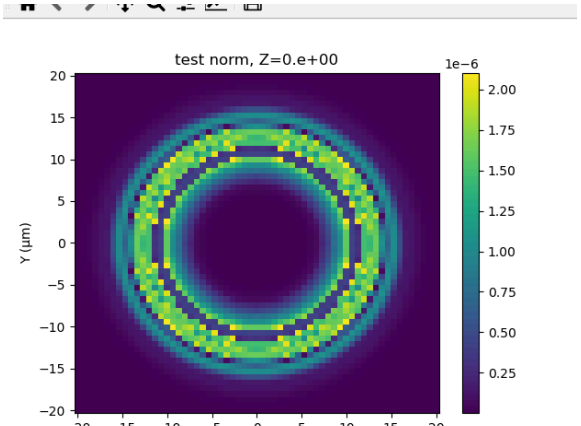


Figure 1; For C=100, OR for C=1 then *100 before integral step

Figure 2; For C=1, then *100 after integral step (integrate.quad)

Hence we have to be careful with this constant factor as the integration can result in some quite different. The paper ***“Gaussian Beams In Optics of Course”*** gave a formulation for the constant factor (which importantly uses the same equation for u_l_p).

As well as this, the radius of the beam can be changed through the ‘beam waist’ parameter w_0, which had been suggested to be ~4.0*wavelength for other Laguerre beams, but in terms of trying to match the situation modelled for OTT this needs to be closer to t values.



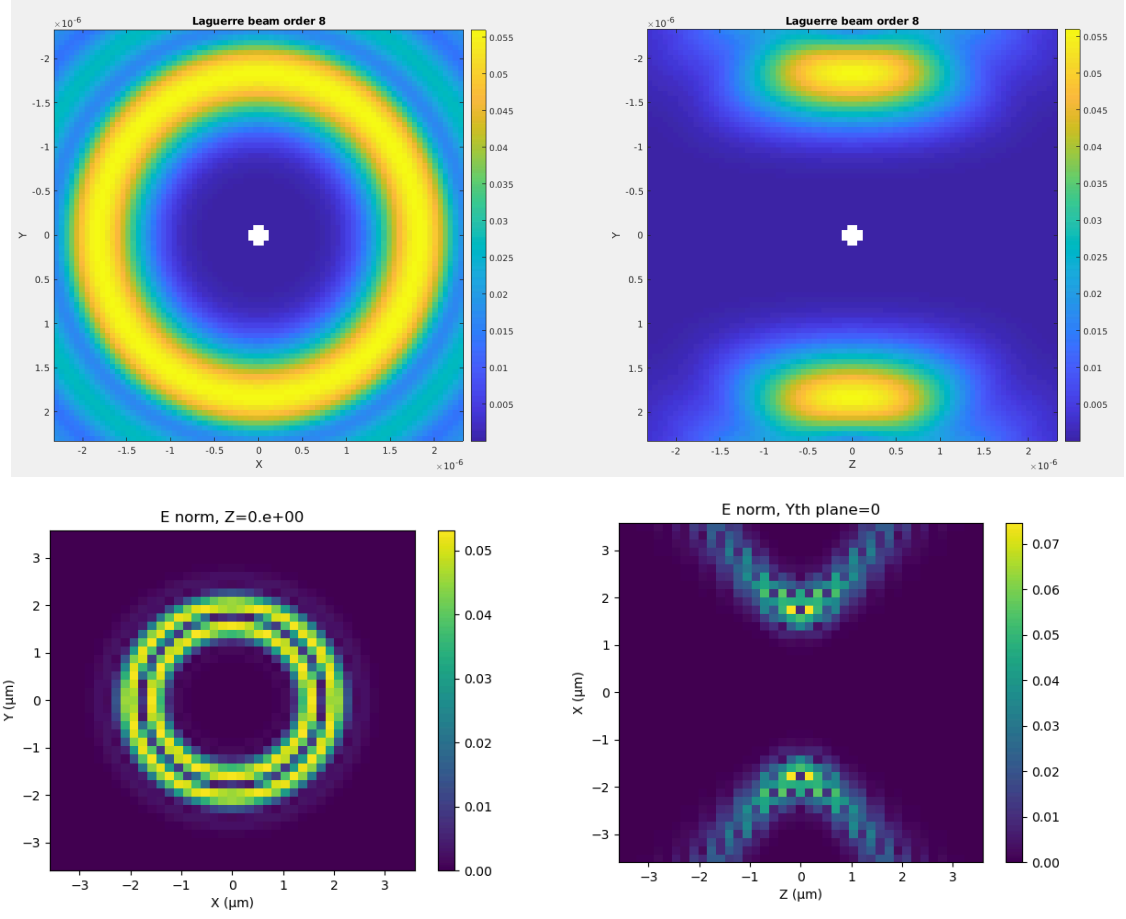
The following graphs compare the Laguerre beam generated in OTT to that made in python from the equations given. This python version was deliberately tuned to have similar beam radius (by adjusting beam waist,w_0, to 0.6*wavelength here, for wavelength = 1.064e-6m), and to have similar magnitude for E (both of the order ~10^-2, achieved by adjusting alpha and beta, both set to 10^-8 here). For this example the normalisation constant ‘C’ was also found in the equation list (called ‘A’ in the paper, a convention not followed by the previous paper used, hence the continued usage of the name ‘C’). The beam also uses radial=0, azimuthal=8 for the orders. For the OTT plots Nmax = 10. Using these parameters the plot for the XY and ZX plane were generated.

by

$$\mathcal{E}_{p,\ell}(\rho,\theta,z) = \frac{A}{w(z)}\left(\frac{\sqrt{2}\rho}{w(z)}\right)^{\ell} L_p^{\ell}\left(\frac{2\rho^2}{w(z)^2}\right) e^{-\rho^2/w(z)^2} e^{ik\rho^2/[2R(z)]} e^{i\ell\theta} e^{i\phi(z)}. \quad (8)$$

The solutions contain the associated Laguerre function L_p^{ℓ}, and therefore called Laguerre-Gauss modes. The subindices p and ℓ label the solutions of order $N = 2p + |\ell|$, which also reduce to the zero-order solution when $p = \ell = 0$. The normalization constant is given by[10]

$A = p!\{2/[\pi p![(|\ell| + p)!)]\}^{1/2}$.



The main notable features are that the python Laguerre beam appears to have 2 more defined rings as opposed to OTT’s more singular high-intensity ring (this could however in part be due to a somewhat misleading colorbar). Also, in the ZX plot, the python version clearly has a more curved Z component, however it is in decent agreement with the low Nmax OTT plot (not true for higher Nmax OTT plots), as both are seen to have a tapering high intensity flat section between Z=+-1 micrometre.

Therefore I would say that this Laguerre beam generated in python matches the OTT version quite well, enough such that trapping should be able to occur or show some trapping-like behaviour, and the function can be further tuned as it is tested. The main focus should now be to have ADDA implement this, in order to allow particle interactions and force calculations. If it is seen that its behaviour deviates from OTT too much, then this equation can be revisited to look for further problems. One such problem that could be explored is to check that the scipy.integrate.quad is correctly performing the complex integral (and not just treating it as real), which seems like it could be a likely source of errors.

3/11/24

I have continued adding to the program to generate the Laguerre-Gaussian beam. I found a description for ***'u_l_p'*** which depends on the cylindrical position being considered, which makes sense as this describes the amplitude of the beam.

polarised light. Here $u(\rho, \phi, z)$ is the complex scalar function, expressed in cylindrical polar coordinates, describing the distribution of the field amplitude of a Laguerre–Gaussian beam and is given by

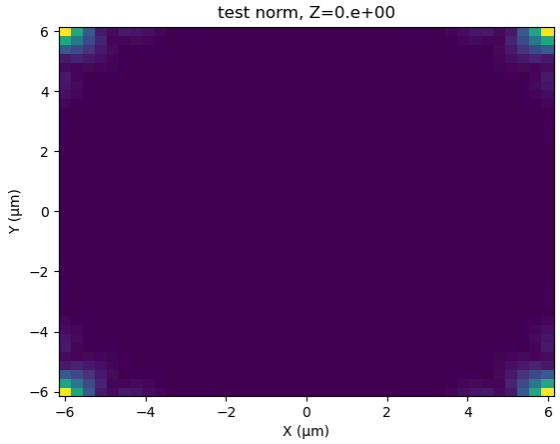
$$u_{pl}(\rho, \phi, z) = \frac{C}{\sqrt{1+z^2/z_R^2}} \left(\frac{\rho\sqrt{2}}{w(z)}\right)^l L_p^l\left(\frac{2\rho^2}{w^2(z)}\right) \exp\left(\frac{-\rho^2}{w^2(z)}\right) \exp\left(\frac{-ik\rho^2 z}{2(z^2+z_R^2)}\right)$$

$$\times \exp(\pm il\phi) \exp[i(2p+l+1)\tan^{-1}(z/z_R)]\,. \qquad (2.8)$$

where z_R is the Rayleigh range, $w(z)$ is the radius of the beam, L_p^l is the associated Laguerre polynomial, C is a constant and the beam waist is situated at $z=0$. As may be seen, the z-component of both the angular momentum/energy ratio and the angular momentum density naturally devolve into orbital and spin components associated with the values l and σ_z, respectively. The quantity l arises from the azimuthal dependence in the term $\exp(il\phi)$ while σ_z, where $\sigma_z = i(\alpha\beta^* - \beta\alpha^*)$, is readily identifiable with spin when the electric field in the x- and y-directions is proportional to $(\alpha\hat{x} + \beta\hat{y})$. It may be noted that the spin term in the equation for M_z depends on

I have had to separate the final vector expression for E into components as the scipy.integrate function seems to be giving arrays when parsing numpy arrays (in the hope the process would be vectorised).

My first run of this function has produced the following result on the right. This same pattern also appears for any scale it is viewed on. Obviously this is not correct however there is some visualisation occurring shows it is producing X,Y,Z outputs. The problem likely is to do with the Legendre polynomial I am using, as the scipy function ***'special.lpmn(m,n,z)'*** was producing errors for having a larger m than n, which I can see why this would cause problems for an associated legendre polynomial (to do with taking to many derivatives for than the polynomial can handle), however this is the case setup for this equation. There is also some confusion where the paper labels ***'L_l_p'*** as the associated legendre polynomial, however I believe this is usually labelled as P_m_n, so it may be that the scipy function is for a different legendre polynomial. The problem may also be that the l and p talked about here are NOT the same as the azimuthal and radial order (respectively) of the beam, hence I shouldn't be using l=8, p=0 (however they were labelled as the orbital and spin angular momentum of the beam, which is what I have always interpreted the radial and azimuthal order to be).



scipy.special.

lpmn

lpmn(*m, n, z*) [source]

Sequence of associated Legendre functions of the first kind.

Computes the associated Legendre function of the first kind of order m and degree n, `Pmn(z)` = $P_n^m(z)$, and its derivative, `Pmn'(z)`. Returns two arrays of size `(m+1, n+1)` containing `Pmn(z)` and `Pmn'(z)` for all orders from `0..m` and degrees from `0..n`.

This function takes a real argument `z`. For complex arguments `z` use clpmn instead.

Parameters:

m : *int*
`|m| <= n`; the order of the Legendre function.

n : *int*
where `n >= 0`; the degree of the Legendre function. Often called `l` (lower case L) in descriptions of the associated Legendre function

z : *array_like*
Input value.

where z_R is the Rayleigh range, $w(z)$ is the radius of the beam, L_p^l is the associated Laguerre polynomial, C is a constant and the beam waist is situated at $z=0$. As may be seen, the z-component of both the angular momen-

Also, I found while researching the beam a site https://teaching.smp.uq.edu.au/scims/optics/lgbeam.html which talks about OTT and mentions a couple of the equations it used to make its Laguerre-Gaussian beam, which seems to have an equation matching the u_l_p I used, and also states that the l and p I am using are reversed. However, when I switch these values I encounter errors that ***special.lpmn*** can't deal with complex numbers, so I need to calculate this value with another module (or manually).

Laguerre-Gaussian Beams

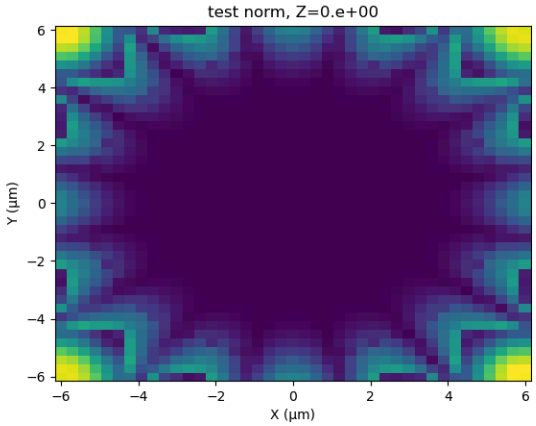
Laguerre-Gaussian (LG) beams, sometimes referred to as doughnut shaped beams, are circularly symmetric solutions to the wave equation. Higher order modes appear as concentric rings. Beams with non-zero radial order carry orbital angular momentum and appear with a dark spot in the centre of the beam due to the phase discontinuity.

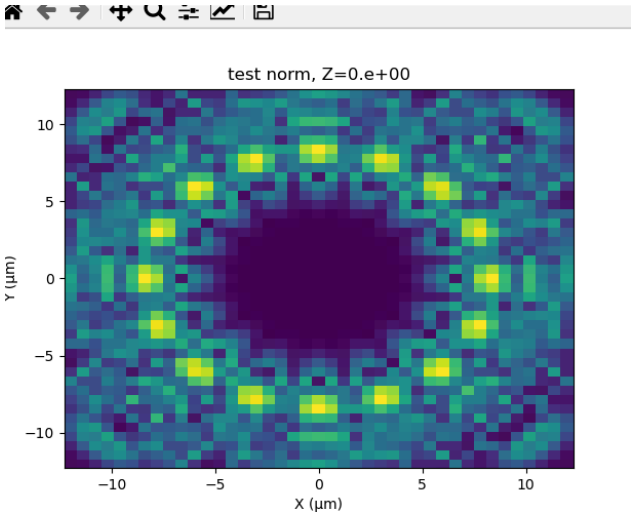
The following visualisation shows linearly polarized LG mode solutions to the paraxial wave equation. The field amplitude at any plane along the beam axis is proportional to \[U(R, \phi) \propto R^{|l|} L_p^{|l|}(R^2) \exp(il\phi)\exp(-i(2p + l + 1)\arctan(z/z_R))\] where \(R(z)\) and \(\phi\) are the normalized radial and angular coordinates at plane \(z\), \(L_p^{|l|}\) are the Laguerre polynomials, \(z_R\) is the Rayleigh range and \(l\) and \(p\) are the radial and azimuthal mode numbers.

The left panel shows the beam in 3-D space. The top panel shows the amplitude/intensity profile of the beam and the bottom panel shows the phase of the electric field.

Having changed to complex associated legendre polynomial calculator (special.clpmn) I am now getting these plots (on the right), which do scale and are resembling a Laguerre beam. The distances appear different to where the patterns occurred in OTT (e-5 rather than e-6) almost certainly due to constants for amplitude, etc being chosen arbitrarily. Also, I may need to normalise my cylindrical coords (mentioned in one of the papers, although this seems counter intuitive, you want to know the distances).

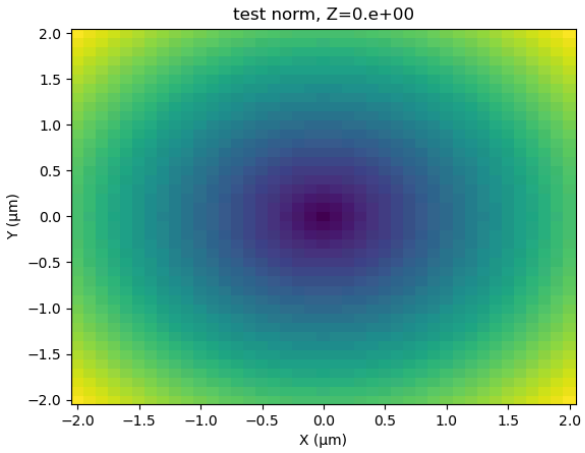
```
z   = pos[2];
phi = math.atan2(pos[1],pos[0]); ### CHECK MAKE SURE IS CORRECT -> principle angle ###
rho = cmath.sqrt( pow(pos[0],2) + pow(pos[1],2) );

l     = radial;#azimuthal;#8.0;     #Orbital angualr momentum number
p     = azimuthal;#radial;   #0.0;     #Spin angular momentum number
k     = 2.0*math.pi/wavelength;
w_0   = 4.0*wavelength; #Beam waist, may not be needed
z_R   = k*pow(w_0, 2)/2.0;     #Rayleigh factor
alpha = 1.0;    ## WHAT ARE THESE
beta  = 1.0;    ## WHAT ARE THESE
```





1/11/24

I have continued to try and add the Laguerre-Gaussian beam to ADDA. I have written a program to visualise an incident E from a beam (tested with E=|**r**|), and so can now try and implement the




```
def Generate_Beam(beam_spec, space_data):
    """
    . Generates the electric on the grid queried
    . Replacates the space data structure as you traverse through it, returns the field data

    . beam_spec = [beam_type, <params>]
    """
    field_data = [];
    space_jump = [
        (space_data[0][1] -space_data[0][0])/(space_data[0][2]-1),
        (space_data[1][1] -space_data[1][0])/(space_data[1][2]-1),
        (space_data[2][1] -space_data[2][0])/(space_data[2][2]-1)
    ];
    for k_ind in range(0, space_data[2][2]):
        field_data.append([]);
        for j_ind in range(0, space_data[1][2]):
            field_data[k_ind].append([]);
            for i_ind in range(0, space_data[0][2]):
                pos = [
                    space_data[0][0] +i_ind*space_jump[0],
                    space_data[1][0] +j_ind*space_jump[1],
                    space_data[2][0] +k_ind*space_jump[2]
                ];
                E_field = [0.0, 0.0, 0.0];
                if(beam_spec[0] == "test"):
                    E_field = [pos[0], pos[1], pos[2]];
                elif(beam_spec[0] == "laguerreGaussian"):
                    E_field = Get_E_LaguerreGaussian_Beam(pos, beam_spec[1], beam_spec[2]);
                else:
                    print("Invalid beam type");
                field_data[k_ind][j_ind].append(E_field);
    return field_data;
```

Laguerre form for E to see if it is correct.

Previous Form

$$E^L_{m,n} = \frac{e^{-in\phi}}{w(\zeta)}\left(\frac{\rho}{w(\zeta)}\right)^n L^n_m\left(\frac{2\rho^2}{w^2(\zeta)}\right)\exp\left[ikz - \frac{\rho^2}{w_0^2(1+i\zeta)} - i\psi^L_{n,m}\right].$$

$$\psi^L_{n,m} = (n+2m+1)\arctan\zeta$$

$$L^n_m(r) = \frac{e^r r^{-n}}{m!}\frac{d^m}{dr^m}\left(e^{-r}r^{n+m}\right),$$

$$w(\zeta) = w_0\left(1+\zeta^2\right)^{1/2}.$$

$$\zeta = z/z_R$$

$$z_R = \frac{kw_0^2}{2} = \frac{\pi w_0^2}{\lambda}$$

beam waist $w_0 = 4\lambda$

Current Form

$$\boldsymbol{E} = \int_0^k \mathrm{d}\kappa\, E(\kappa)\, \mathrm{e}^{\mathrm{i}l\phi}\exp(\mathrm{i}\sqrt{k^2-\kappa^2}z)$$
$$\times\left((\alpha\hat{\boldsymbol{x}}+\beta\hat{\boldsymbol{y}})\, J_l(\kappa\rho) + \hat{\boldsymbol{z}}\frac{\kappa}{2\sqrt{k^2-\kappa^2}}\left[(\mathrm{i}\alpha-\beta)\,\mathrm{e}^{-\mathrm{i}\phi}J_{l-1}(\kappa\rho) - (\mathrm{i}\alpha+\beta)\,\mathrm{e}^{\mathrm{i}\phi}J_{l+1}(\kappa\rho)\right]\right).$$

$$E(\kappa) = u(l,p)\exp\left(-\frac{k\kappa^2 z_R}{2(k^2-\kappa^2)}\right)\left(\frac{\kappa^2}{k^2-\kappa^2}\right)^{(2p+l+1)/2}\left(\frac{k^2}{k^2-\kappa^2}\right)^{1/2},$$

After speaking with my supervisor, I found what seems to be a better form for the electric field of a Laguerre-Gaussian beam (contains an integral, more in line with expectation for his past experience). I have started implementing this new form, using a scipy '***.integrate.quad***' function to evaluate the integral and ***scipy.special.jv*** to evaluate the bessel functions of the first kind, however I am trying to understand what the alpha and beta in the equation mean, as the paper that provided the equation only briefly introduces them as "the x- and y-components of the electric field are proportional to exp(-i*phi)", which suggests that I should be able to choose them arbitrarily, however this cannot be the chase as the X-Y gradients are very important in the Laguerre-Gaussian field when thinking about trapping. In an earlier section talking about the paraxial case, it says that l and sigma_z are the "orbital and spin components" where "sigma_z = alpha*beta_conjugate -beta*alpha_conjugate" with E proportional to alpha*x_hat + beta*y_hat. Once I can conclude how to resolve this the field should be simple to plot and test, where I can then after try and convert this to be generated inside ADDA. This new formulation is from the paper ***"Orbital angular momentum and nonparaxial light beams"*** spread across equations 3.1-3.3 and 4.1 (non-paraxial), although some of these equations are more intermediary and so won't be used directly.

As well as this, I was directed to another paper about how the force calculations in DDA work (force acting on each time oscillating dipole) in order to manually find forces (which will be essential when I have implemented the laguerre beam). The ADDA manual and other similar papers do not describe this process as far as I have seen, which would explain why ADDA tends to avoid force calculation (only implemented for plane waves). ***"Time-averaged total force on a dipolar sphere in an electromagnetic field"*** explains this time averaged force (for ith component) as:

$$\langle F^i \rangle = (1/2)\mathrm{Re}[\alpha E_{0_j} \partial^i (E_0^j)^*].$$

I have not tested or explored this any further yet, but this seems promising as it is purely in terms of electric field (which is all we have for the Laguerre beam, which is very helpful). I assume this would then be applicable by finding the total field at each dipole and applying as expected (hence near/internal field will be needed).

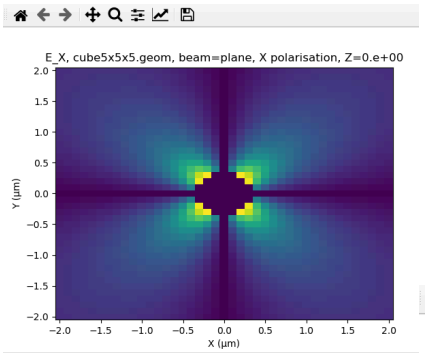
31/10/24

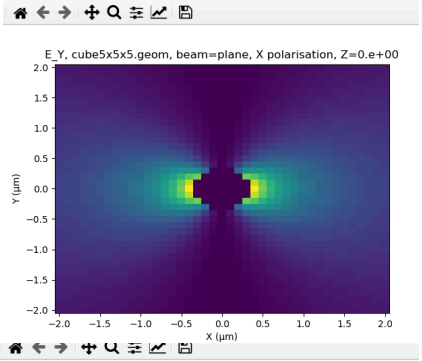
Today I have continued to implement the far-field E plotter using the polarisations from ADDA. This worked quite simply using the formula from before (stated again here for completeness). Now I have the scattered part of the field, I need to somehow retrieve the incident field from ADDA to plot alongside this scattered part for a total field, which appears to be more complicated. An alternative is I could manually enter the form of the electric field for that beam into my python program and then sum this manual incident field with the scattered, however this somewhat opposes the purpose of the program, which is to test that ADDA is giving the expected total field. This being said, I could try and manually input the E field for the Laguerre-Gaussian beam into the python program then plot it to ensure I am giving ADDA the correct form of the electric field (this would still be better if I could pull the value from ADDA instead but it will nonetheless ensure I am using the correct equation, as ADDA's input method for beams is quite confusing, so it would be a good test).

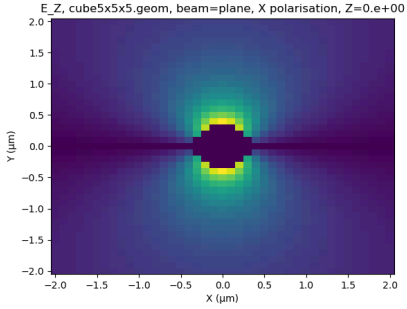
$$\mathbf{E}_{\mathrm{sca}}(\mathbf{r}) = \frac{\exp(\mathrm{i}k_{\mathrm{sca}}r)}{-\mathrm{i}k_{\mathrm{sca}}r}\mathbf{F}(\mathbf{n}),\ \ \mathbf{n} = \mathbf{r}/r\,, \tag{42}$$

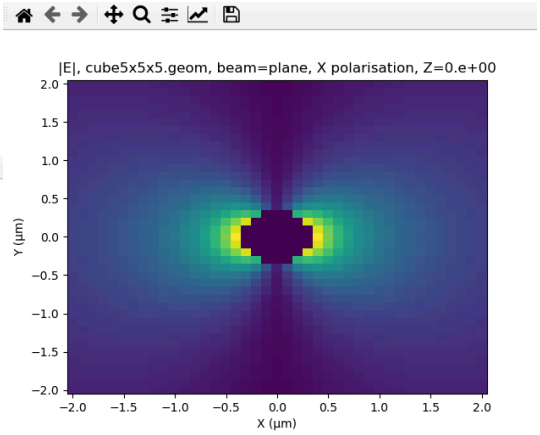
where k_{sca} is the wave vector for the scattering direction. The original definition of Eq. (42) was given in [50] for the free-space scattering, when $k_{\mathrm{sca}} = k$ and the scattering amplitude is

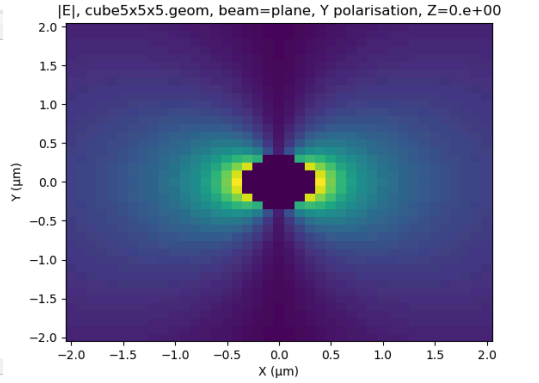
$$\mathbf{F}(\mathbf{n}) = -\mathrm{i}k^3(\bar{\mathbf{I}} - \hat{n}\hat{n})\sum_i \mathbf{P}_i \exp(-\mathrm{i}k\mathbf{r}_i \cdot \mathbf{n})\,. \tag{43}$$

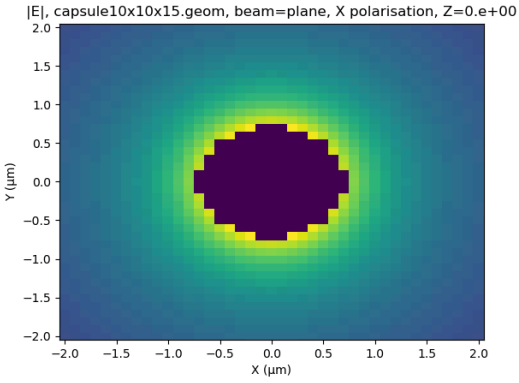






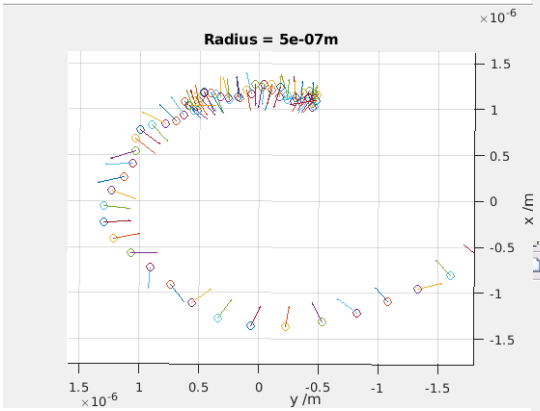


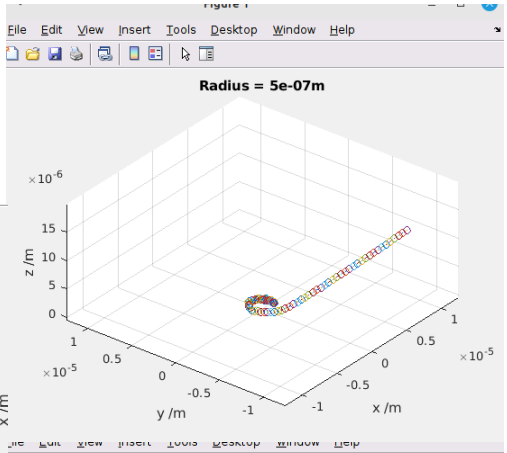


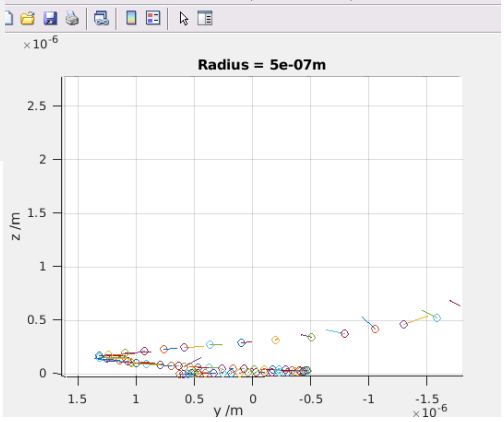


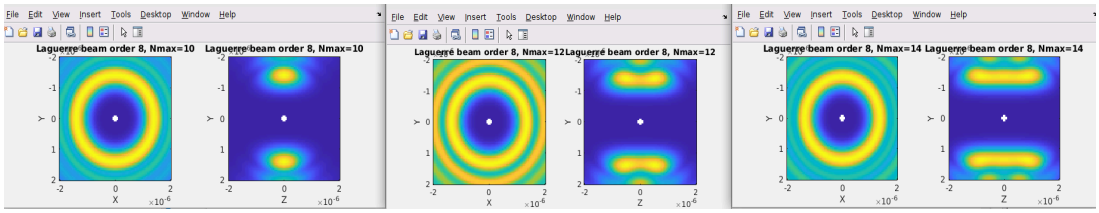
My partner has also been experimenting with some of the values we found (yesterday's plots) that appeared to be close to circular. We were unsure as to how large the Z forces may be so could not

tell if the result be circular, helical or just purely divergent (as perhaps the Z force pushed too far from the circular region, leading back to radial motion out or in with no circular part). Because of this he simulated some small scale dynamic motion with eulerian time steps (which were small, 1e-4, to try to improve accuracy as best as possible for this simple but quick to implement method). The mass of the calcite fragment was also estimated using its density (2.7g/cm^3). The total time here is 150e-4s simulated. Observing this path in the XY and YZ plane, we see that the path was very circular (until the end of its motion) but with some Z motion, meaning it had a helical orbit. As well as this, the Z height can be seen to start diverging at approximately 0.5e-6m, which also happens to be where the Laguerre gaussian beam appears to break (with our truncation of the infinite series of VSWF with Nmax = 10), which is resolved by the Nmax being increased (makes the Z axis of the beam longer and more uniform). Hence, this explosive motion at Z=0.5e-6 that breaks the helical motion appears to be from the Nmax chosen (which was chosen to be low to increase the speed of the simulation). This should be simulated again but for larger Nmax to prove this is true, and if will show trapped helical motion.









I started to setup a function to plot the internal field at the dipoles (from the data ADDA gives), however I am unsure how useful this will be as it is located on a different grid to the one I may specify for the scattered field, so it would be hard to superimpose. It may be possible to use this data however to find the incident field from the beam using this and the polarisability of each dipole, however I am unsure how I would account for the electric field scattered from each other dipole on the dipoles (maybe I would need to write a near-field E plotter too, then evaluate it at each dipole).

29/10/24 - 30/10/24

Note that the Nmax used now has been upped to anywhere between 10 and 30, as these larger values are needed to see order 8 Laguerre beams correctly.

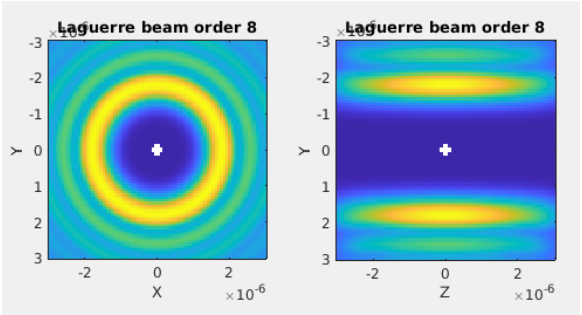
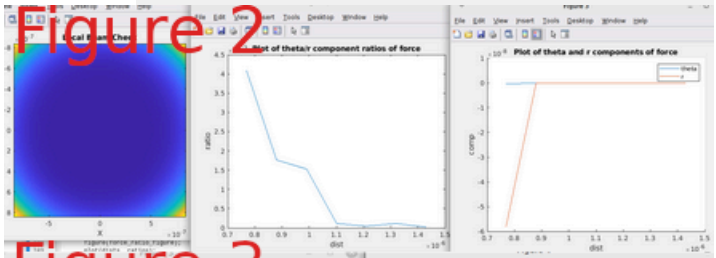


Figure 1; Beam being used (matching experiment previously considered)

Figure 2; For varying distance from the origin of beam (using all the new parameters found). Essentially seeing only a radial force (no theta component), and this is only seen at the lowest distance plotted here.



This may be because of problems with the z position chosen here (currently at z=0).

Figure 3; Shows how the r and theta components of the particle at (1.1e-6, 0, Z) vary as Z changes. We can see the radial force is largely driven by gradient in the Z direction, as it tends to 0 as you |Z| increases. This means that some radial force is to be expected, even in trapped motion, due to this Z gradient.

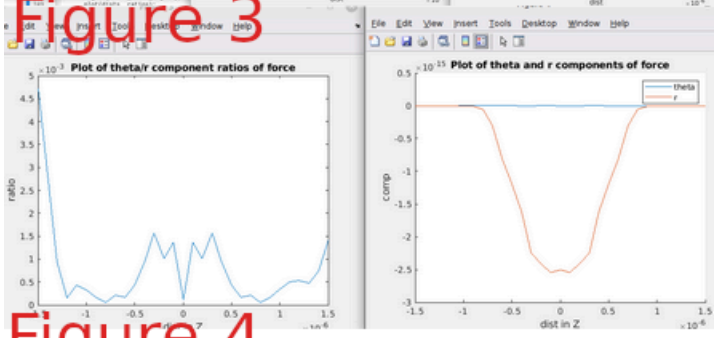


Fig.4 from paper ***"Optical trapping and moving of microparticles by using asymmetrical Laguerre–Gaussian beams"***.

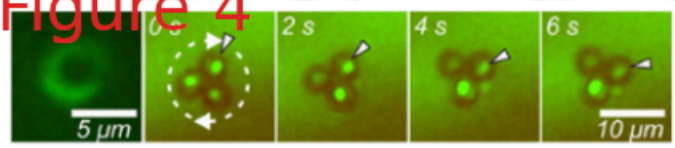
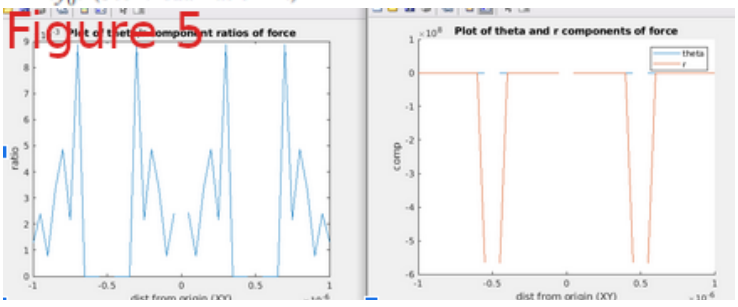


Fig. 4. Intensity distribution in the trapping area (left frame) and stages of motion of three polystyrene microspheres trapped by using the asymmetric Laguerre–Gaussian beam with $n = 3$, $x_0 = 0.1w = -iy_0$. (See Visualization 1.)

Figure 5; Effect of changing JUST distance from origin (no Z) again. See the 0 force unless at the bright gradient ring.



$$\zeta = z/z_R$$

$$w(\zeta) = w_0\left(1+\zeta^2\right)^{1/2}.$$

After trying this, I tried modifying the formulation of the T-matrix to offset the beam instead of

$$E_{m,n}^{L} = \frac{e^{-in\phi}}{w(\zeta)}\left(\frac{\rho}{w(\zeta)}\right)^{n} L_m^n\left(\frac{2\rho^2}{w^2(\zeta)}\right)\exp\left[ikz - \frac{\rho^2}{w_0^2(1+i\zeta)} - i\psi_{n,m}^{L}\right].$$

moving the particle directly, as I realised that for these 1 particle cases the T-matrix would not be centering the system properly (it tries to centre to the bounding box of the union of the particles). A series of calculations were then set to be performed in which the position of the particle in XYZ space was checked at several points (10x10x5 grid).

I evaluated the forces in the XY plane to see if circular/helical motion ever appeared with some combination of these variables. Note that each set of plots represents 1 variable change being measured over the fixed grid points in space, where the other variables are set to their 'base' value; **radius=0.5e-6m, wavelength=1.064e-6m, n_medium=1.0, n_particle=1.55**. Note as well that for figures 6-12 Nmax=9, the 8 being the lowest such value where the order 8 Laguerre beams calculated properly. This value was chosen as the computation takes a very long time (roughly 20-40 seconds per Maxwell Stress-Tensor force calculation call, hence ~2 hours per set of data sampling the space of a 10x10x5 grid) and so 9 offered the best time for computation whilst still holding all the main detailed characteristics of the order 8 beam.

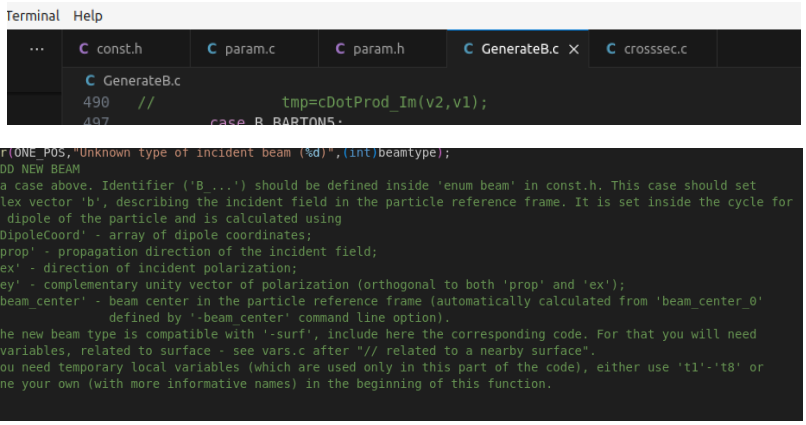
From this I got the figures 6-12 (force in XY plane, at 3 separate Z heights, with the exact magnitudes of the quiver plot shown in the surface plot beside each corresponding quiver, noting that no Z components or magnitudes are considered here), where figures 6,7,8 show how the wavelength (in free space, base=1.064e-6, where ***'lambda=n'=>'wavelength0=base*(1.0 +0.1*n)'*** e.g. the percentage change from the base value) changing affects the forces, which appears to show a critical value (around n~1) where the magnitude of the forces become larger and more circular.

Figures 9,10 showed the n_medium variable changing (same percentage change with **n** as wavelength had), which appeared quite stable for the base value (n=0), but appeared to diverge a bit for larger n_medium (or perhaps alternatively made the distance from the origin at which particles want to be trapped further out, however it is hard to see this with the current grid dimensions observed). It would be interesting to watch this change on a finer resolution grid and at smaller steps, as this variable had a more significant effect on trapping than maybe expected.

Figures 11,12 showed the change with radius, which for the values tested appeared to have very little effect, however these values only ranged between (for n=0 to 2, n=2 not shown below for conciseness but essentially the same as n=0&1) 0.5->0.6e-6 => 1.0->1.2e-6 diameter, which is within the somewhat vague size range given by the **"*Intrinsic and Extrinsic Nature of the Orbital Angular Momentum of a Light Beam*"** paper (which saw no orbital momentum for a particle of 3x1.5 micrometres [major and minor axis], and orbiting for approximately 1 micrometres). This should absolutely be tested for larger radii and much smaller radii too to see if it loses some of its nicer looking circular motion.

Now I can view the Z component of force at the radii where circular motion appears to be present (and perhaps perform some small scale time-stepping for dynamic motion) to see if helical trapping is possible (as was suggested by the previous experimental paper mentioned with the calcite fragments).

While these plots are being processed, I will try to implement the Laguerre Gaussian beam in ADDA. The electric field for a Laguerre Gaussian was found in the paper ***"Gaussian, Hermite-Gaussian, and Laguerre-Gaussian beams: A primer"***. ADDA gives a guide as to how to add a beam here; https://github.com/adda-team/adda/wiki/AddingBeam .This link is given in the manual as well, but not elaborated on any further. The main difficulty here is in the documentation provided, which is limited and so makes understanding and effectively making changes to ADDA difficult conceptually. Essentially the beam fields need to be specified at each dipole location, where ADDA parses in the list of dipoles for this purpose. The beam has currently been set to take 2 parameters (radial and azimuthal orders) and be centred around the origin. Currently, my partner and I are trying to find a way to write the electric field from the beam in the program in the format required (which wants a 'b' vector to be set as the E field xyz components for each dipole in what appears to be a 1D array 3N long, where N is the number of dipoles). However, it is apparent now that I first should get a program working to visualise the far-field before continuing with this, otherwise it will be extremely difficult to tell if the beam is actually working as expected.



An alternative to adding the beam like this is to (1) try and implement an angular spectrum instead, however with DDA this could cause problems from too many terms being scattered and used, which could make the process extremely slow or break. We could also try (2) using the first N terms of a angular spectrum to find plane waves the build up a Laguerre-Gaussian beam, then run each plane wave in ADDA individually and sum the electric field/forces (principle of linear superposition), which should not miss any inter-scattering terms and, with a large enough N, give a good approximation for a Laguerre order 8 beam.

$$\mathbf{E}_{\mathrm{sca}}(\mathbf{r}) = \frac{\exp(\mathrm{i}k_{\mathrm{sca}}r)}{-\mathrm{i}k_{\mathrm{sca}}r}\mathbf{F}(\mathbf{n}),\ \ \mathbf{n} = \mathbf{r}/r\,, \tag{42}$$

where k_{sca} is the wave vector for the scattering direction. The original definition of Eq. (42) was given in [50] for the free-space scattering, when $k_{\mathrm{sca}} = k$ and the scattering amplitude is

$$\mathbf{F}(\mathbf{n}) = -\mathrm{i}k^3(\overline{\mathbf{I}} - \hat{n}\hat{n})\sum_i \mathbf{P}_i \exp(-\mathrm{i}k\mathbf{r}_i \cdot \mathbf{n})\,. \tag{43}$$

In any case, in order to compare this result with OTT we will also want to write a program to display the far-field electric field for this scattering (not supported by ADDA by default, however ADDA's calculation obviously provides us with the polarisations and mueller/scattering matrix for the simulation, and these values should be somewhat simple to calculate).

Once this is done, if trapping can be found on either system it can be replicated on the other, compared, then extended to a continuous limit with N trapped particles. The forces can then be found on each particle and considered to observe where the models either break down in this continuous case or where they tend to one-another. This will then give a particle-and-spring validity range where particles approximate a large object, and hence flexible shapes can be attempted to be simulated.

Figure 6

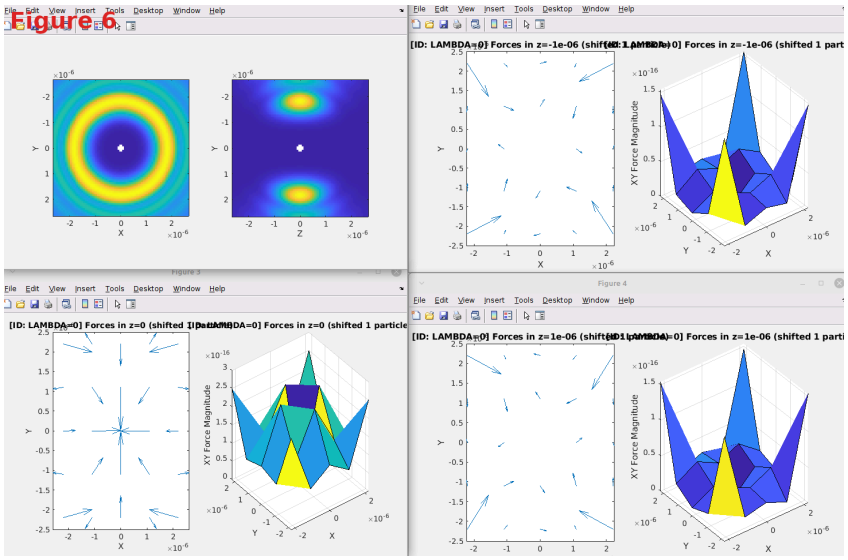


Figure 7

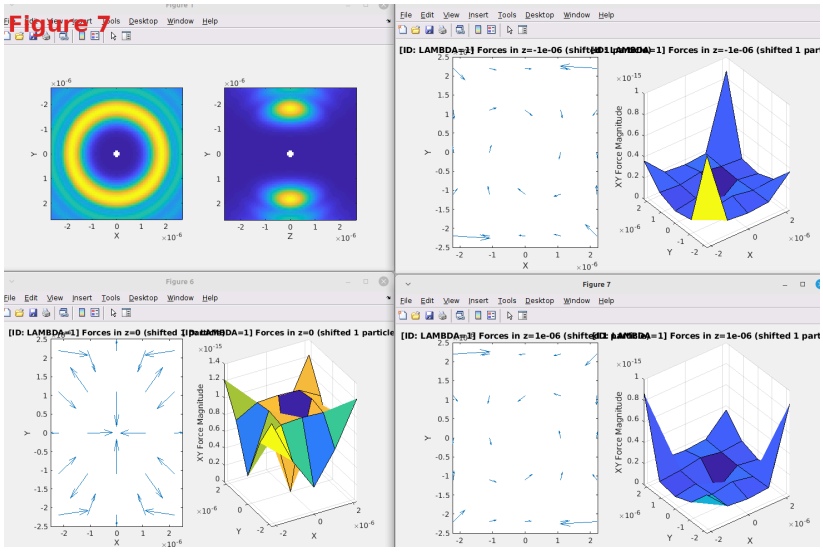


Figure 8

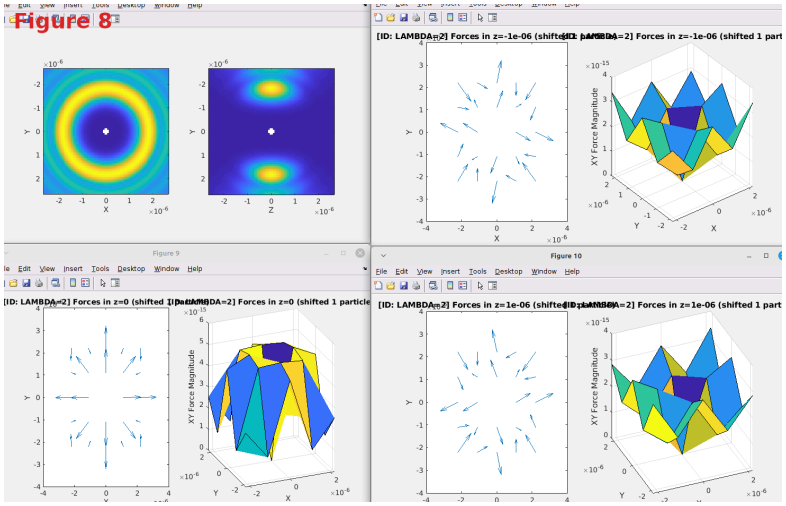


Figure 9

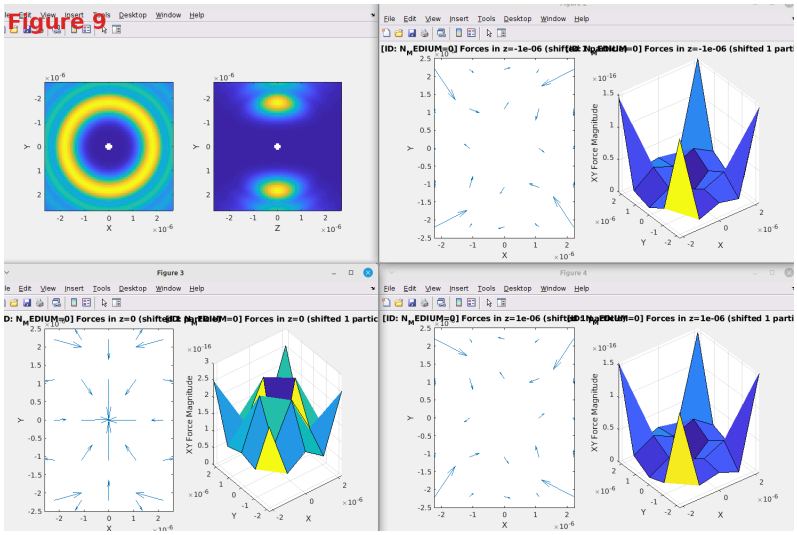


Figure 10

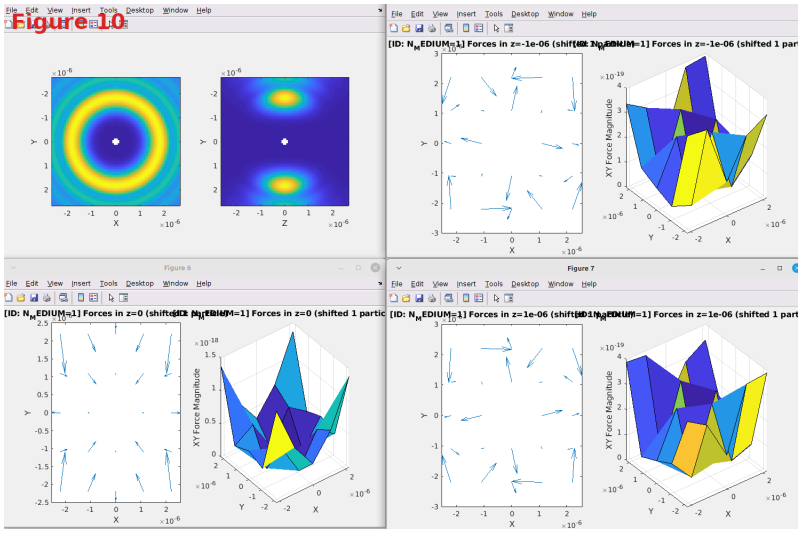


Figure 11

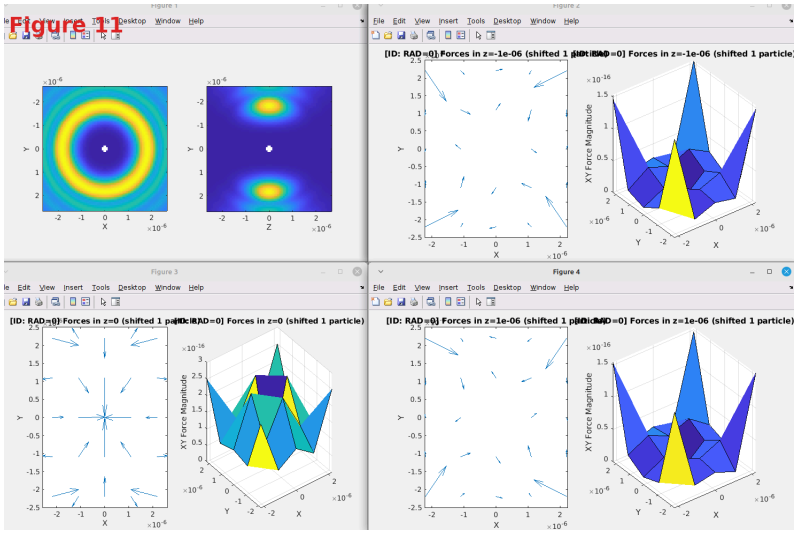
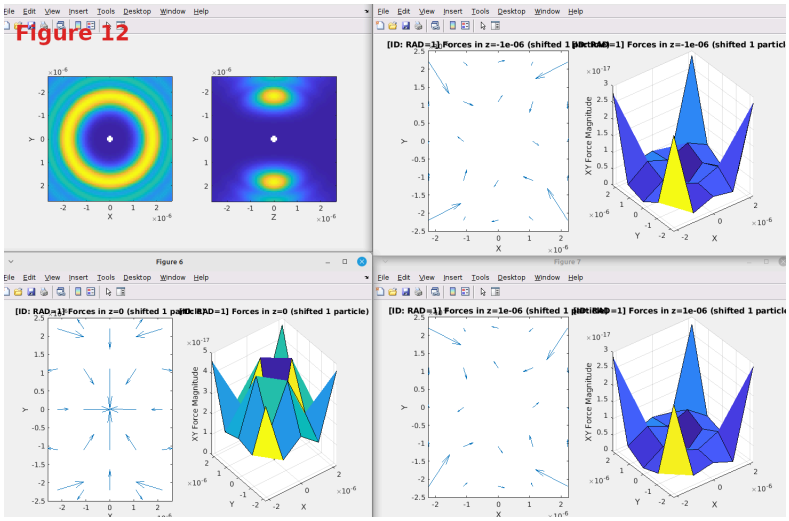


Figure 12



28/10/24

My presentation was performed today and so work can now continue properly on the project. I am still trying to simulate a trapped particle in a Laguerre Gaussian beam and am trying to sweep different parameters in order to find this. I also have work on getting the laguerre beam working ADDA (as well as scattered field and forces in ADDA) working, or I can test if the DDA formulation of the T-Matrix gives more reliable results. These can be tested if I find that trapping the particle is making no progress (as our supervisor had mentioned it was very difficult to getting working in a simulation, especially since a lot of literature does not list all of the parameters they used to achieve their trapping, so it is hard to copy a situation exactly). For this I am trying to more closely follow the parameters in the paper ***"Intrinsic and Extrinsic Nature of the Orbital Angular Momentum of a Light Beam"***, as this shows exactly where the (single) particle sits in the beam and a rough size for the particle, hence all the other wave parameters can then be varied to try achieve this trapping.

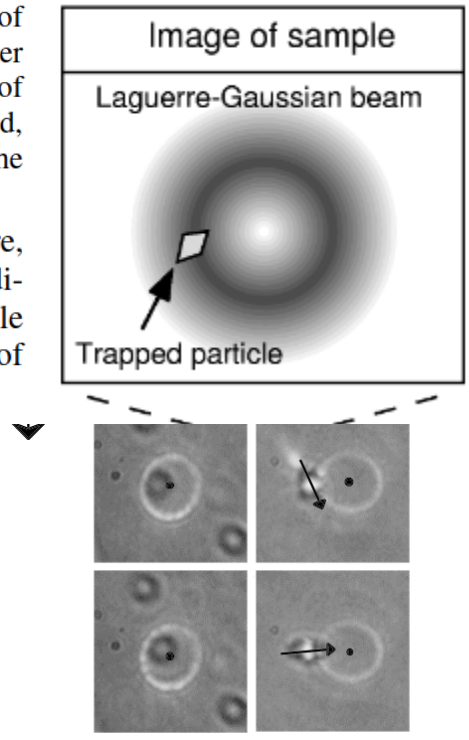
Note that the 'l=8' mentioned in this paper does appear to march the order of the laguerre gaussian beams we are talking

prism, respectively [13]. The radius of maximum intensity, r_{max} of a Laguerre-Gaussian mode is given by [14],

$$r_{max} = \sqrt{\frac{z_R l}{k}}, \quad (9)$$

plane. This beam is generated from the 100 mW output of a commercial Nd:YLF laser transformed, using a computer generated hologram, to give a Laguerre-Gaussian mode of approximately 30 mW. The beam is circularly polarized, $\sigma = \pm 1$ with a high azimuthal mode index, $l = \pm 8$. The

Our optical tweezers are based on a 1.3 numerical aperture, ×100 objective lens, configured with the trapping beam directed upwards, which allows easier access to the sample plane. This beam is generated from the 100 mW output of



For small particles the force arising from the light scattering, the momentum recoil force, becomes important. For a tightly focused Laguerre-Gaussian mode, the dominant component of the scattering force lies in the direction of beam propagation. The gradient force again constrains the particle to the annulus of maximum beam intensity. However, as the intensity distribution is cylindrically symmetric, the particle is not constrained azimuthally. Because the particle is trapped off the beam axis, the inclination of the helical phase fronts and the corresponding momentum result in a tangental force on the particle. We observe that a small particle, while still contained within the annular ring of light, orbits the beam axis in a direction determined by the handedness of the helical phase fronts; see Fig. 3. We conclude that the larger calcite and small

FIG. 3. Successive video frames showing particles trapped near the focus of an $l = 8$ and $\sigma = 1$ Laguerre-Gaussian mode. The left column shows particles of ≈1 μm diam. These particles are sufficiently small to be subject to a well-defined scattering force, allowing them to interact with the orbital angular momentum of the beam. They are set in motion, orbiting the beam axis at a frequency of ≈1 Hz. The right column shows a calcite fragment with a length of ≈3 μm and a width of about ≈1.5 μm, which is large enough not to interact detectably with the beam's orbital angular momentum. However, due to its birefringence it interacts with the spin angular momentum of the beam and is set spinning about its own axis at ≈0.3 Hz.

about, where it represents some orbital angular momentum in terms of the number of times it wraps the axis.

When this data is collated, we get the following parameters for trapping to have occurred for this paper;
-Beam parameters= (
    l=8 /order=8 [orbital angular momentum l*h_bar],
    σ=1 [spin angular momentum σ*h_bar],
    Circularly polarised [handedness=>orbit direction],
    Numerical aperture=1.3

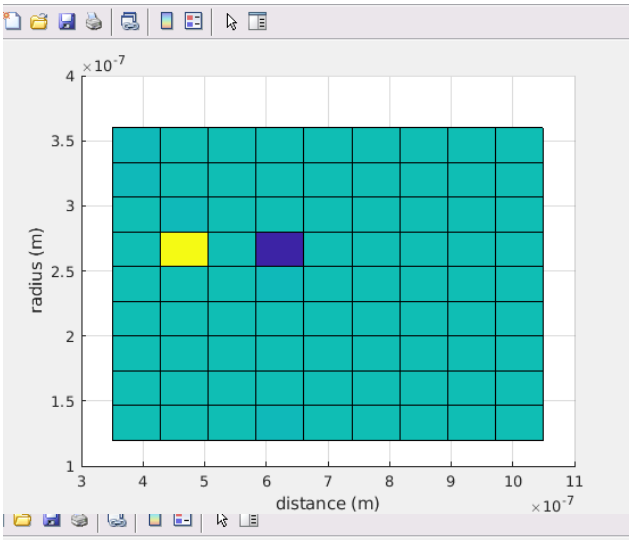
    Also stated that 1st high intensity ring of highly focussed LG of order of several μ diameter
)
-Particle parameters= (
    Diameter = 1μm [other fragment was 3μm by 1.5μm, which showed spinning not orbiting due to too large size],
    Calcite fragment => n =1.66 to 1.49
)

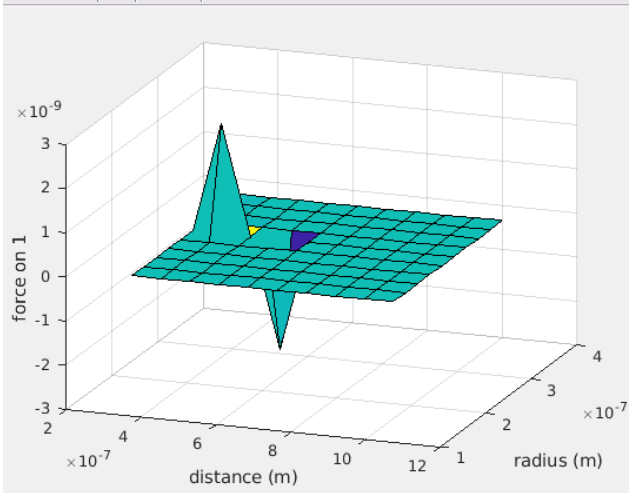
- The refractive index of the surrounding medium is not mentioned

Hence my program needs to sweep values for the (1) wavelength of the beam (2) the refractive index of the outer medium, and then (3/4) move the radius and distance (from the centre of the beam) of the particle (HOWEVER, this should only be shifted by small amounts for fine adjustment, the setup does show these values quite well and so should not be tweaked too much).

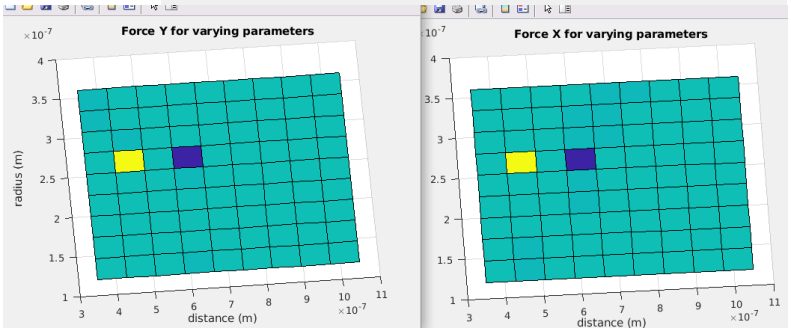
25/10/24

The focus of today has been on trying to determine parameters needed for optical trapping to occur. After the information read yesterday, it seemed that we were close in value to the required values (their particles' radii were of the same order as ours, roughly 5 micrometers) but did require very exact values in order to found accurately (one paper suggested a difference of 0.5 microns lead to either orbital motion or rotation motion only, so very different). Hence today I have tried setting up a program in OTT to search phase space for the required variables. To do this I have setup a system to view the position of the particle for varying distances from the origin, with varying radii, in order to get a ball-park idea of values to test the particle with. After this, these initial values are put into a function which searches space with varying distances and radii from these initial values and finds the resultant force in the theta-hat direction on a given particle in a ring in the laguerre beam (which can have its order varied to see which has a steep enough gradient that trapping may occur), this is plotted (in a 3D surface plot) to visualise the forces in the varying setups. The goal is then to test the dynamics of the system with parameters found in this surface plot where the theta-hat component is as expected, AND where the radi-hat component is also as expected (where they are in agreement that would lead to circular motion). This would then hopefully confirm that trapping has occurred, or if more space needs to be searched for another plausible solution.





The 2 graphs to the right show initial results using this plot, where it has located a positive and negative force in the Y-direction for the particle sitting on the positive X-axis, which indicates there may be circular motion here. These plots used a 10x10 sampling size, with Nmax=4, NA=1.4, wavelength=1.064e-6m, n_medium=1.3, n_particle=1.6, circularly polarised Laguerre Gaussian order 3 beam.



Comparison to the X forces however shows that the x force may also be large here, so a finer resolution grid should be taken to see if there are any solutions here in this range, or if another area should be searched.

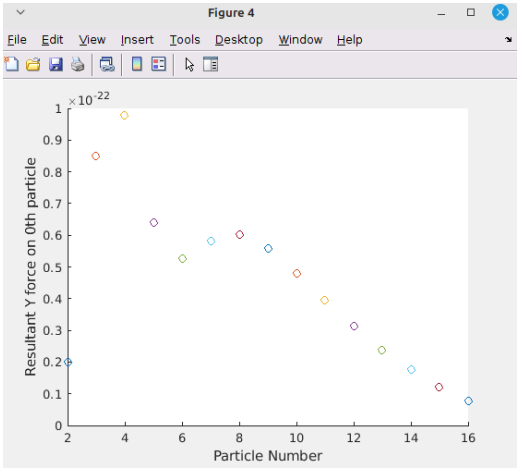
As well as this, my partner implemented a new method for finding the maxwell stress-tensor integral using a gaussian quadrature method provided by MATLAB, which required some re-evaluation of our previous formula as this method could only find components individually for the vector integral. This appears to match the values we had before, but seems to converge faster (which was the aim) at a slight cost of speed.

Some other lines of research I can also perform if this method has trouble is manually input the laguerre gaussian beam into ADDA (using the instructions they provide to edit their source code), and so find the scattering from this beam and so find the forces from the polarisations given by the output. This will then allow similar testing in ADDA, as well as comparison between the two methods.

24/10/24

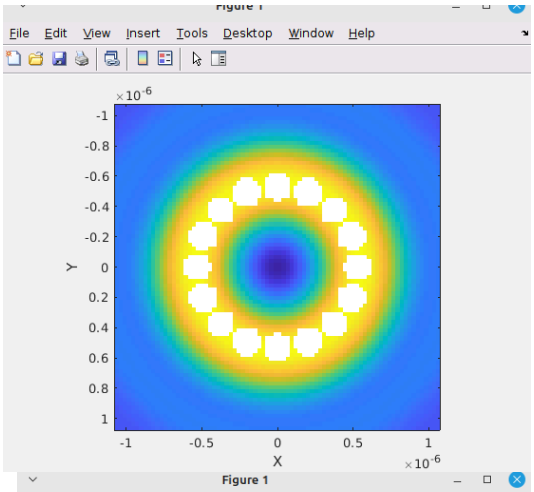
The project presentation has now been submitted now. This presentation has been my main focus the last few days, with no major points to note (hence no information noted in this lab book) , just some changes to layouts, researching some of the equations more in depth and practicing the presenting itself.

Today I have tried looking at the forces of the continuous limit of the ring case of continuous particles in a laguerre order 2 beam. All calculations here use Nmax=4, which appeared sufficient as it converged



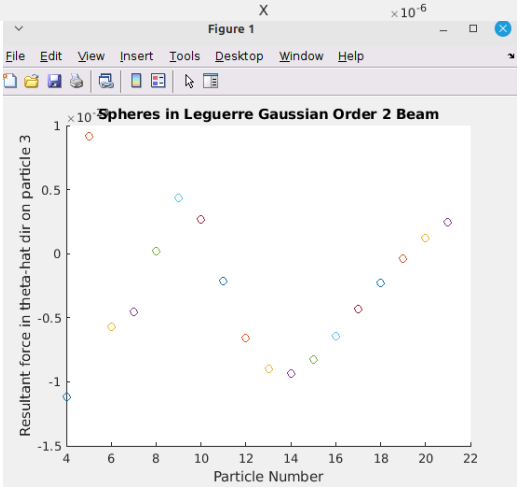
The 1st graph is showing the Y force on the 1st particle as more particles are added to the system (particles created in a ring from 0 to 2pi radians, starting from index 0).

The 2nd graph shows the layout of the full 16 particles used in the 1st graph (for its final iteration). All previous setups of N particles have their particles evenly distributed about the circle like this, and the radius of the particles does not change iteration (it is constant through, fixed at the start to be nearly touching when the full iteration set is complete)

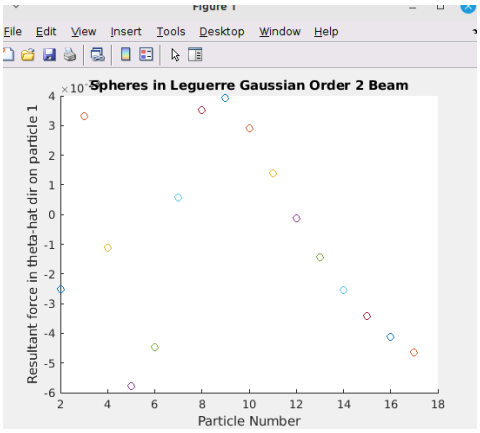


The 3rd graph shows the force in the theta-hat direction (CCW in polar coordinates) when considering the 4th particle (index 3). THis oscillation appears to be from the position of the particle moving each iteration as, due to the way the particles are placed, each iteration the particles will all shift clockwise around towards the 0th particle, and so perhaps this shift is causing a force change.

The 4th graph here shows the 1st particle observed again but using this new force_theta calculation. However, here we can see that this is different to before, meaning the force before was never entirely in the Y direction, despite this being our prediction for particles trapped in a laguerre gaussian beam. Hence this means I need to ensure the particles are in fact trapped and moving as expected (and not with some additional X or Y force) before repeating these calculations, as the standard expected behaviour is not occurring.



The trapping will require lots of changes to parameters. My supervisor suggested we increase the numerical aperture of the laguerre beam, which will focus the light beam to the peak band, resulting in a larger gradient and hopefully better trapping. It was also suggested that higher order beams would work better for similar reasons, as well as looking for values of n_medium and particle in literature that have experimentally worked. It is also possible that our particles are too small and so do not trap well.



The exert is from ***"Orbital motion of optically trapped particles in Laguerre–Gaussian beams"***, which explains some tested conditions that lead to trapping.

The paper ***"Transfer of orbital angular momentum to an optically trapped low-index particle"*** also talks about how different sized particles get trapped in different rings of this laguerre.

trapping are possible. Particles large enough to completely encompass the bright ring are trapped axially while smaller particles reside within the ring itself. Ef-

where z_R is the Rayleigh range of the beam. Even under the tight focusing associated with optical tweezers, the peak intensity ring of a Laguerre-Gaussian mode of high index l may be made several μm in diameter and, consequently, be much larger than the particles it is attempting to trap.

It is not surprising, for such conditions, that we observe the particles to be confined by the gradient force at the radius of maximum light intensity and not on the beam axis. When a birefringent particle such as a calcite fragment is trapped, and circularly polarized light is converted to linear, we observe that the particle spins about its own axis. The sense of rotation is governed by the handedness of the circular polarization.

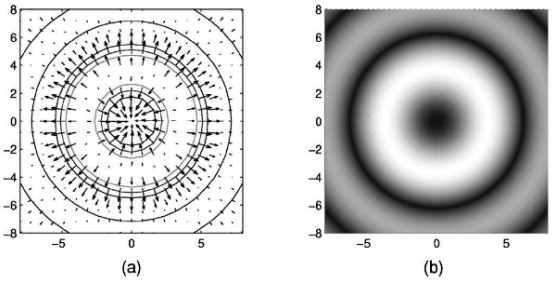


FIG. 4. In (a) we see a vector diagram showing the total transverse forces and contour curves. In (b) we see the corresponding Bessel beam profile.

mentioned above, we found the critical value for the maximum diameter of the particle to be 6 μm, that is, a particle whose diameter is equal or larger than that value can only be held in the central minimum of the beam. In contrast, for a 5.75 μm diameter sphere, which can be trapped in any of the dark rings, we found that its equilibrium positions around the first and second dark rings are shifted about 16.6% and 7.2%, respectively, in relation with the corresponding minima of the intensity distribution. A 2 μm diameter sphere suffers a

the particle is, the more the equilibrium positions are shifted radially outwards, until a critical value for which the sphere can be trapped in the central minimum of the beam, but it cannot be stably trapped in any of the outer dark rings. This is in accordance with our experimental observations discussed later. For instance, with the same beam parameters we

21/10/24

Work has continued on setting up my project presentation. I had written up a first draft and talked through each slide with my supervisor on Friday. I now have close to a final draft, where my focus is now on clearing up small misconceptions I have about specific parts of the project (mainly in relation to how formulas are derived and justified).

Changes suggested by my supervisor in terms of material that was missing or should be expanded on has now been implemented.

I have had thoughts about what should be tested next in the project (partially as a result of the presentation planning, which has served as a good tool to reflect on where we are and what we are trying to achieve), and so I next plan to test how the force plots I have produced so far will be affected by increasing the accuracy of both methods, hopefully with the aim to show they converge to the value I have given. I also plan to try and test the continuous limit of particles in a circular formation in the Laguerre Gaussian beam again, but without time stepping (just placing them at separate points and messing forces), with the aim to hopefully see the angular forces decrease in magnitude as more particles are brought into the beam and the particles are hence brought closer together. This would then give a clearer idea where the particles start approximating a larger object, which would then allow us to start actually testing springs in the system.

17/10/24-18/10/24

Yesterday (17th) and some of today was mostly taken up by creating my presentation for my project to be performed in the coming weeks (28th). This time was taken planning what was the most important information to be conveyed ,what order was best to present it and reminding myself of core details of the methods used (as a lot of this is hidden behind the software being used, and so can be easy to skip over without realising).

I also received an answer to my email about how the union of objects and the forcetorque() function worked, which I think for completeness is best to include in full here for future reference;

“

Hi James,

Sorry for my delayed reply.

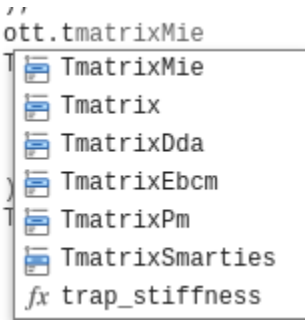
The shape union just stores information about the shapes, so it will depend on which tmatrix method you are using. There are three scenarios I looked at during my PhD:
- calculating the tmatrix of an extended shape (each shape in the union overlaps, and there is just one surface formed). For this you should be able to use point matching on the surface.
- calculating individual tmatrices and then calculating a combined tmatrix with an iterative method (or assuming that interaction is negligible). I can't remember if I included theses codes in the toolbox.
- calculating the total tmatrix of a multiparticle system using a method like DDA (DDA might still be a bit buggy).

For the forcetorque of N individual particles, you might be able to calculate the force on N-1 particles, and N particles and compare the results to get the individual possible contribution. There might be a more efficient way (maybe send an email to Timo Nieminen, he might have suggestions). I wanted to look at this more during my PhD, but I also needed to graduate, so I didn't get a chance.

Hope this helps.

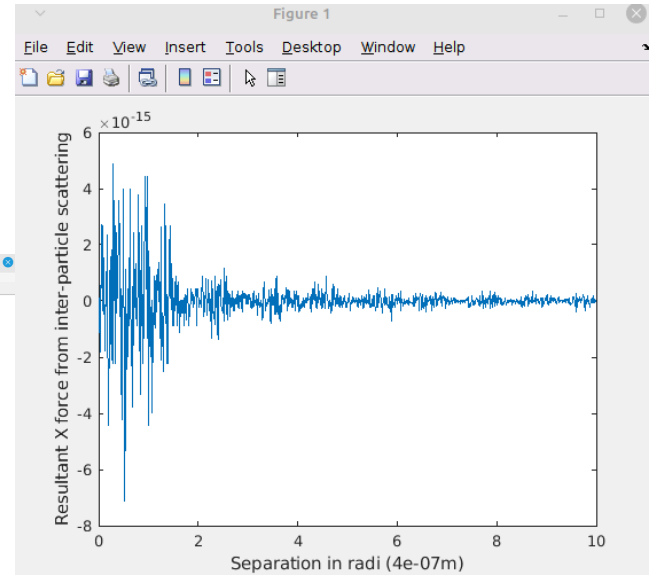
Best,
Isaac

”

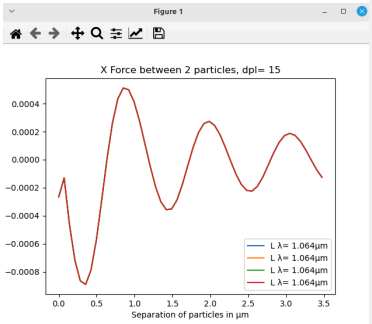


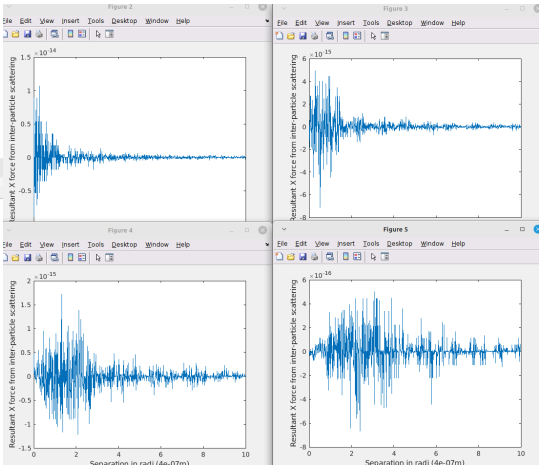
Importantly this clarified that the resulting character of union is different for different T-Matrix calculation methods, which would have been a long process to confirm manually and potentially tricky to spot. The approach to forcetorque() mentioned was also similar to approach we had taken by separating the fields for each particle and comparing it to the unioned result, but we had not tried this with forcetorque() directly. From the electric field perspective, I think this approach would likely work as the forcetorque() found for individual particles would include their scattered beams and an incident beam if wanted, and

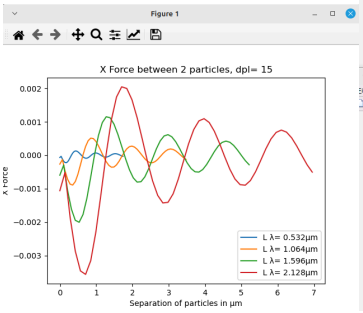
hence would not include the inter-particle scattering (which is what is of interest and ideally the field contributing to the force in our plot), and so the difference should leave exactly this term (union -left particle -right particle) for the resultant force over that system from inter-particle terms. My only worry with this is that the force of the left particle and right particle may have no force individually, and the union has a net of 0 due to equal and opposite sign on both particles, hence we cannot see the granularity of the force between the two particles due to averaging occurring within forcetorque(). This would also likely be further affected by the particle being centred back at the origin when the T-Matrix is found (although for the individual case the beam can be moved).



On the right is my first result trying this approach, which does appear to have a decaying structure like we would hope if it agreed with DDA. Similarly, the oscillating could be for the same reason, where the wavelength is of such a size that as the particles pass through it the net direction of the force. I was initially worried about the magnitude of the force seen here, however with the DDA I was observing the X force from the incident+scattered_1+scattered_2+inter-particle fields, and so would clearly be much larger (just from the incident field alone). This being said, if the union is giving a resultant force of 0, then this would just be plotting 2x the force felt by 1 particle under the beam influence, and so would not be from inter-particle scattering, but just from the scattered beam off 1 of the particles. This would mean inter-particle forces are completely ignored and the dropoff we are seeing is just a 1/r^2 drop of incident beam strength with distance.



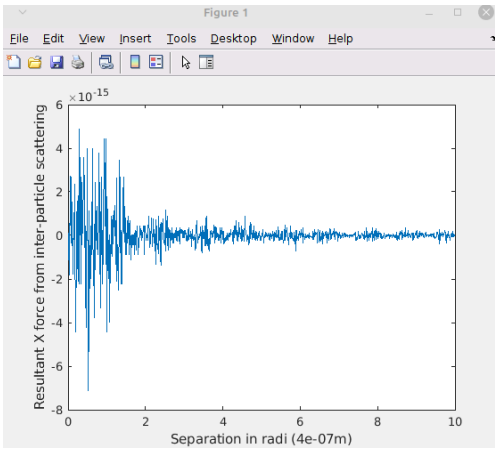


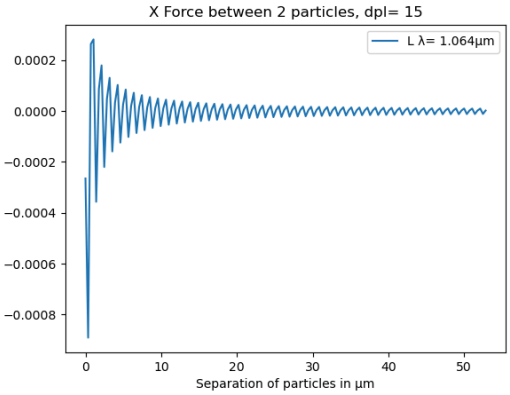


Next figure about changing wavelength for this calculation, starting at [0.5*base_wavelength] (where base wavelength is used in the previous plot), up to [2.0*base_wavelength] (up in ½

intervals). As we can see this changes the phase of the graph produced essentially. This is what was seen for the DDA, where changing the wavelength changed the magnitude of force at 0 separation.

Adda graph (1) = 5x5x5 cube with same wavelength as single original in OTT, Adda graph (2) = 5x5x5 cube with different wavelengths (same as tested in OTT)

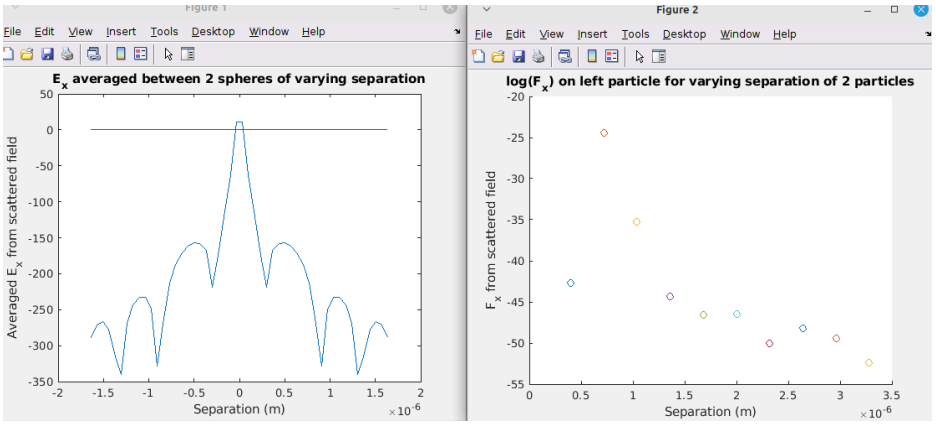




. For DDA; 15x15x15 sphere, 150 iter, 15 dpl, 5 dipole_sep_step, dipole_size =0.0709 => total of 1.063 (1 wavelength) => radius of 0.5*wavelength
For OTT; Same wavelength, 0.5*wavelength radius

. (Both share wavelength, radius, separation)
. Both share similar decay
. Both start high then decay to low magnitude
. Magnitude difference (difference between trying to measure only inter-particle scattering VS full force from everything in X direction).

16/10/24



I have continued working on the Maxwell stress tensor force calculation for the T-matrix and extended the previous comparison to now compare both electric field and force (both in the x direction) as the 2 two spherical particles are separated. The log of both values are plotted as there is a tendency for extremely large or small values to be generated from some anomalous peaks in the scattered wave (that I think are appearing as this is a far-field calculation so inside the spheres these large spikes appear, visible in the full surface plot for the scattered beam, however it is unusual that these peaks can occur at the origin when the particles are separated to +-D in the x-axis). This force plot suggests a similar behaviour to the DDA force calculation with rapid decay (although here the decay may be stronger as this log plot looks like it is somewhat exponential, hence the actual plot would be very exponential), and seems to show the force blowing-up at 0 separation. Note that this plot considers the left particle only, and so x forces are positive (pointing towards the right particle), and that this force calculation was done by computing the integral over the sphere with a cubic bounding box split into N samples per axis, where N=53 for this simulation. The stress tensor is computed for each sample area here on each face of the bounding cube and the integral results are summed. This method did have problems arise for certain values of N, where the value would diverge massively to get total forces of the order of 10^53 newtons, rather than values between 10 to 10^-5 newtons. This I believe was caused by the unusual bright spot at the centre of the grid, which is also supported by the fact that this occurs left often now that the particles are offset to the left and right (when testing the calculation, I had a particle sitting at the origin, hence this issue arised roughly whenever I had an even N value as it sampled the central point). This calculation is performed in the matlab function *'ForceCalc_modified.m'*, as this is a modified version of the work my partner performed in order to make sure the calculation worked.

```
function resultant_force = ForceCalc_modified(shape_position, shape_radius, field, samples)
    mu0 = 4e-7*pi;
    e0 = 8.854e-12;

    sample_width = (2.0*shape_radius)/samples;  %Width of sample segment in real units
    sample_area = (sample_width)^2;             %Small sample surface element area

    resultant_force = [0;0;0];
    normals = { [1;0;0],[-1;0;0], [0;1;0],[0;-1;0], [0;0;1],[0;0;-1] };
    for normal_index = 1:6
        normal = normals{normal_index};
        for j = 1:samples
            for i = 1:samples
                %Get coordinates of this point
                xyz = FetchBoundingCubeXyz(shape_position, shape_radius, sample_width, normal, j, i);
                %Calculate the stress matrix at each point
                [E, H] = field.emFieldXyz(xyz);

                %kron(A, B^T) => dyadic product for column vectors A,B
                TM = 1/2 * real(e0*kron(E, conj(E).') + mu0*kron(H, conj(H).') ...
                    - 1/2 * (e0 * dot(E,conj(E)) + mu0 * dot(H,conj(H))) * eye(3) );

                %Get integrand here (dot with normal)
                integrand = (TM * normal)*sample_area;
                resultant_force = resultant_force + integrand;
            end
        end
    end
```

Ideally now I would like to find the form of the translation matrix for VSWF so I can manually translate the T-matrices generated in OTT, so I can then finalise my test that unions work and also allow the option to more rigorously check other calculations the T-matrix is performing (by

ensuring translations are carried out by comparing them to a manual method). Papers such as Mishchenko's *'T-Matrix computations of light scattering by non-spherical particles: A Review'* do discuss these translations, however it cites that they are simple to perform without actually providing the paper cited, which could be a problem with how the pdf for the paper was stored, as maybe the citations were provided separately. EIther way, I will try to find similar examples in other papers to clear up this idea and allow for some more interesting and important calculations to ensure the current results are valid and make sense.

a scatterer that is contained within a radius r_0, $N_{\max} \approx kr_0$ is usually adequate, but $N_{\max} = kr_0 + 3\sqrt[3]{kr_0}$ is advisable if higher accuracy is needed [7]. Although we assume in this paper (as is usually the

Attached below are some equations from a referenced textbook that are talking about translation of the T-matrix, however I am having difficulty understanding this as it does not explain this section into detail, and it follows directly from an explanation about a rotated not translated coordinate system, which perhaps has lead to this two ideas being combined into a single equation, making it more difficult to grasp. The end of this section also cites another paper as giving a better explanation, but searching for this paper by *"Fuller and Mackowski (2000)"*, seems to lead to the paper titled *"Electromagnetic Scattering by Compounded Spherical Particles"*, which does not appear to have pdf copy anywhere I have looked so far (the publisher's website produces an error when you search for this paper).

$$\begin{matrix}\mathbf{M}_{mn}\\ \mathbf{N}_{mn}\end{matrix}(kr_1,\vartheta_1,\varphi_1)=\sum_{\nu=1}^{\infty}\sum_{\mu=-\nu}^{\nu}\begin{bmatrix}A_{\mu\nu mn}\\ B_{\mu\nu mn}\end{bmatrix}(kr_{12},\vartheta_{12},\varphi_{12})\mathrm{Rg}\mathbf{M}_{\mu\nu}(kr_2,\vartheta_2,\varphi_2)$$
$$+\begin{matrix}B_{\mu\nu mn}\\ A_{\mu\nu mn}\end{matrix}(kr_{12},\vartheta_{12},\varphi_{12})\mathrm{Rg}\mathbf{N}_{\mu\nu}(kr_2,\vartheta_2,\varphi_2)\Bigg],\quad r_2<r_{12},\tag{C.68}$$

where

$$A_{\mu\nu mn}(kr_{12},\vartheta_{12},\varphi_{12})=\frac{\gamma_{mn}}{\gamma_{\mu\nu}}(-1)^{\mu}\sum_{p=|n-\nu|}^{n+\nu}a(m,n|-\mu,\nu|p)a(n,\nu,p)h_p^{(1)}(kr_{12})$$
$$\times P_p^{m-\mu}(\cos\vartheta_{12})\exp[\mathrm{i}(m-\mu)\varphi_{12}],\tag{C.69}$$

$$B_{\mu\nu mn}(kr_{12},\vartheta_{12},\varphi_{12})=\frac{\gamma_{mn}}{\gamma_{\mu\nu}}(-1)^{\mu+1}\sum_{p=|n-\nu|}^{n+\nu}a(m,n|-\mu,\nu|p,p-1)b(n,\nu,p)h_p^{(1)}(kr_{12})$$
$$\times P_p^{m-\mu}(\cos\vartheta_{12})\exp[\mathrm{i}(m-\mu)\varphi_{12}]\tag{C.70}$$

are translation coefficients. Here

$$a(m,n|\mu,\nu|p)=(-1)^{m+\mu}(2p+1)\left[\frac{(n+m)!(\nu+\mu)!(p-m-\mu)!}{(n-m)!(\nu-\mu)!(p+m+\mu)!}\right]^{1/2}$$
$$\times\begin{pmatrix}n & \nu & p\\ m & \mu & -(m+\mu)\end{pmatrix}\begin{pmatrix}n & \nu & p\\ 0 & 0 & 0\end{pmatrix},\tag{C.71}$$

$$a(m,n|\mu,\nu|p,q)=(-1)^{m+\mu}(2p+1)\left[\frac{(n+m)!(\nu+\mu)!(p-m-\mu)!}{(n-m)!(\nu-\mu)!(p+m+\mu)!}\right]^{1/2}$$
$$\times\begin{pmatrix}n & \nu & p\\ m & \mu & -(m+\mu)\end{pmatrix}\begin{pmatrix}n & \nu & q\\ 0 & 0 & 0\end{pmatrix},\tag{C.72}$$

$$a(n,\nu,p)=\frac{\mathrm{i}^{\nu-n+p}(2\nu+1)}{2\nu(\nu+1)}[n(n+1)+\nu(\nu+1)-p(p+1)],\tag{C.73}$$

$$b(n,\nu,p)=-\frac{\mathrm{i}^{\nu-n+p}(2\nu+1)}{2\nu(\nu+1)}[(n+\nu+1+p)(n+\nu+1-p)$$
$$\times(p+n-\nu)(p-n+\nu)]^{1/2},\tag{C.74}$$

and the coefficients

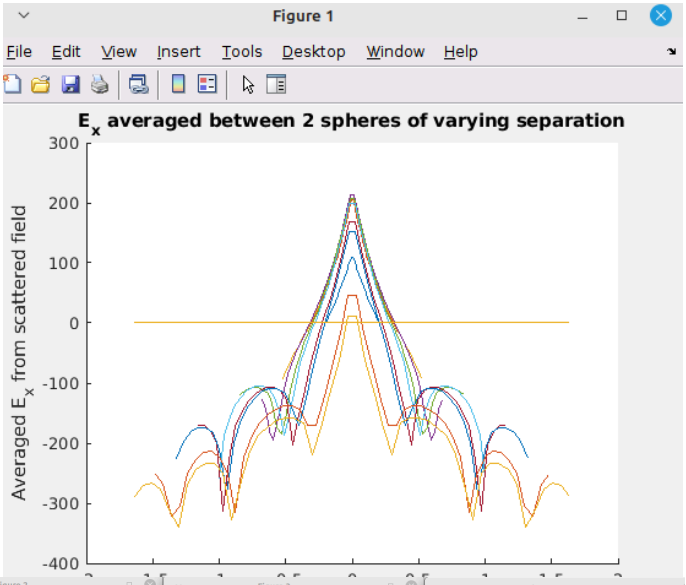
If I can figure out how each of these translation coefficients fit to give a matrix that can then be added or multiplied with the T-matrix (hard to tell which is happening in the equations, as A does not appear to be a matrix from its equation and context in giving RgM/N, and so where does the required matrix come from that involves A/B?) then I will be able to manually test

OTT is working, and in cases where it is not correct it, and it will also allow me to manually calculate the inter-particle scattering (in order to test validity and test forces/electric fields from just these inter-scattering components). Reading these papers has helped improve my understanding of VSWF and so is still worth the time spent even if the translation matrices cannot be found.

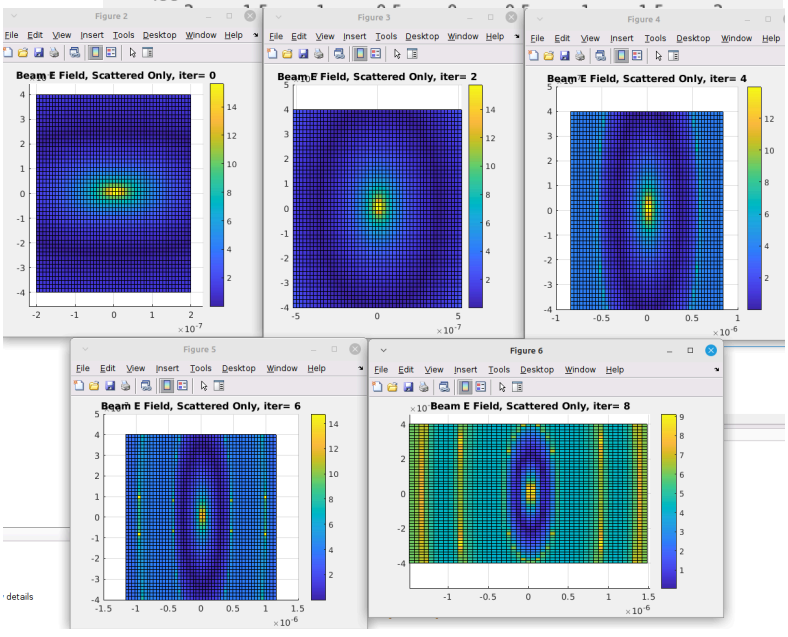
15/10/24

I have organised my files for the project and collected the information me and my lab partner had been uploading to google drive in order to work together (things like papers that were useful to read, code snippets we worked on together, information helping us setup and install some of the different software, etc) and uploaded it to a github repository instead (https://github.com/James-Paget/Light-Driven-Deformation/tree/main) so the code can be backed-up in another form and be easily accessible to assessors or my supervisor should they want to see it.

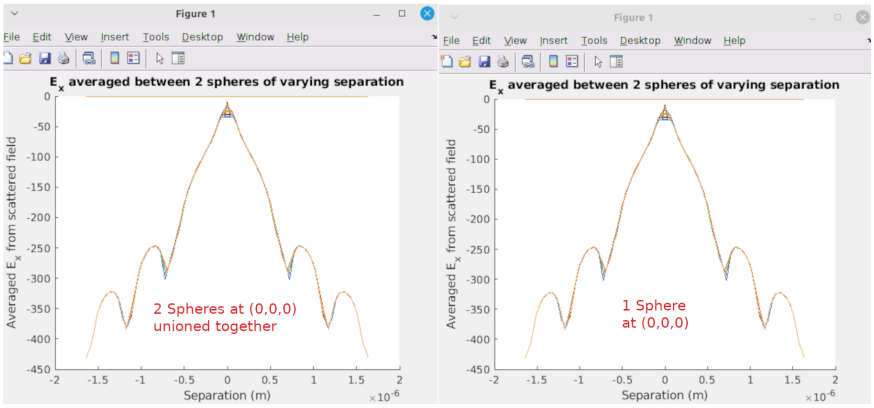
I have also been writing a script in Matlab and python to compare the electric fields for the T-Matrix and DDA approach of separating particles since the '*forcetorque()'* function in OTT appears to not be able to give individual particle forces (this being said, my partner has been working on a manual calculation of the force on particle from E,H field given in OTT using the maxwell stress tensor, however progress on this has stopped temporarily for external reasons). This should give an equally valid metric for comparing if DDA and T-matrix agree on the interaction of separate particles brought very close together. This has been performed in my *'particleSeparationComparison.m'* script file.

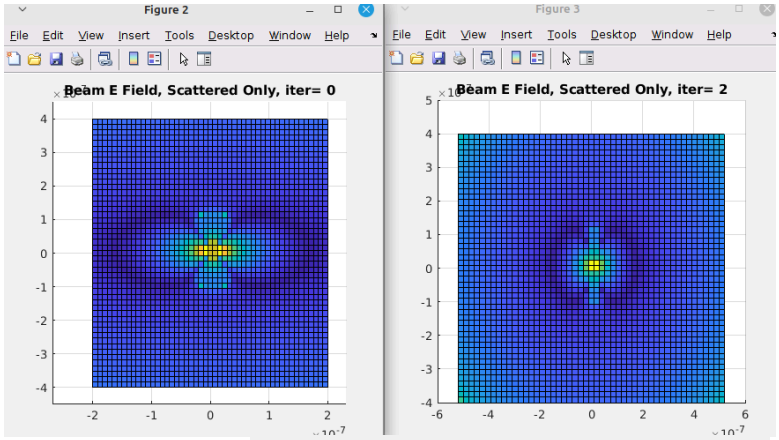


For the T-matrix, the plot on the right was formed which averaged the E_x values along the y-axis for each sample space along the x-axis. This used the union function with multiple particle placed at a position +- d from the origin, with the beam (plane wave, linearly polarised in x direction) at the origin. As I could not perform the translations yesterday fully I am still not entirely sure if union includes all scattering interactions between particles (however it seemed promising that it did), but this plot does show that the magnitude of the field drops off with distance. Also bear in mind that the 3D surface plots use the log(abs(E_x)) and the 3D plots use the log(E_x), as the produced results all had an extremely high intensity spot at the origin, making all other data

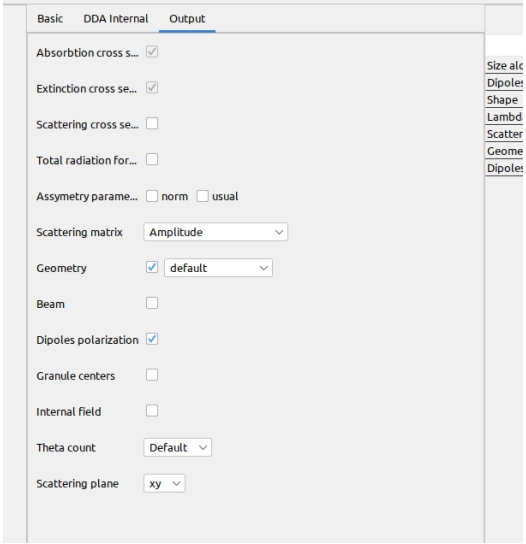


impossible to see. This also means the negative values seen in the 2D plot are just small values. The 2D plot appears to just show fringes formed from the scattering superpositions, but could be inaccurate if the T-matrix of this unioned particle ignored the offset I applied to the spherical particles, which is a worry I have that is hard to test. The figure below shows the same program performing this calculator for (a) 2 spheres unioned together with fixed coordinates at (0,0,0) and (b) a single sphere at (0,0,0). First of all, this shows that the T-matrix OTT finds for a unioned object does NOT translate the 2 objects to origin similar to how it treats a single object, and secondly that it treats this unioned pair over each other the same a single copy (as the name union would imply, meaning that this behaviour is working too). Hence the original data above does hold more validity as we can see there must be some translating behaviour occurring. Also depicted is the surface plot for iterations 0 and 2, which as expected seem to show the same data just with differently sized axes (as expected for particles not changing).



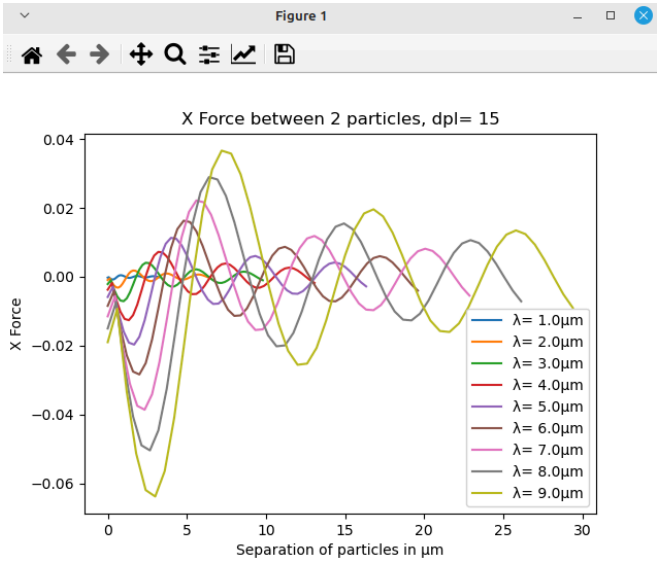


The problem I am encountering now is that ADDA does not automatically give the far-field E for a calculation, so I will have to manually find this. I could either try and do this using the mueller / scattering matrix calculated by ADDA, considered when viewed from above, or I could try to find the E field from the polarisations given by ADDA for each dipole in the particles considered. If I can calculate this then I can compare the results to the T-Matrix results and see if they agree. The figure here shows the possible outputs ADDA can give for a calculation (note that internal fields can be calculated, but not of interest here).

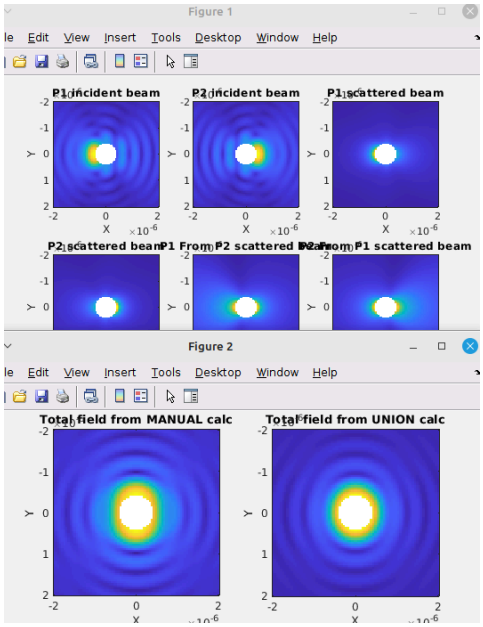


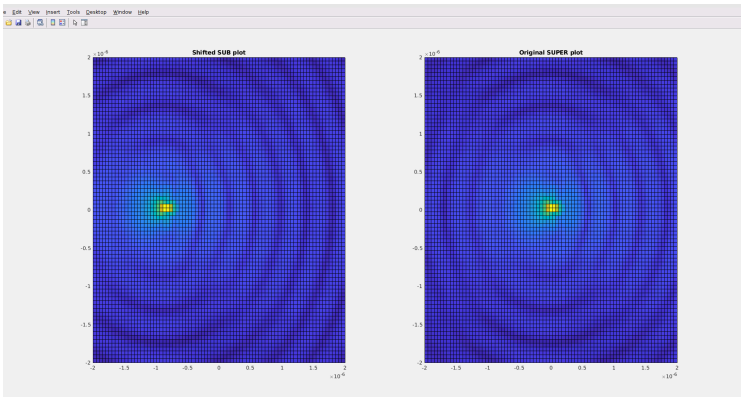
14/10/24

ADDA calculation with 15 dipoles per lambda (dpl), but with varying wavelength. The shape produced here is very different to my previous graph as the old graph had a wavelength luckily chosen such that the particles were of similar length to the wavelength, so we did not see this oscillating effect (caused by the force on each side of the particle being asymmetric due to the phase of incident beam).



After addressing that concern I had with ADDA and its plot, I turn to look back at OTT and testing whether the ott.shapes.Union function does include inter-particle scattering terms (also, it should be noted that today I also contacted **Isaac Lenton** from the University of Queensland, who worked on and was cited as a point of contact in the OTT documentation https://ott.readthedocs.io/en/latest/Introduction.html, to ask about this exact question and also how the forcetorque() function interacted with multiple particle systems in union). To do this, I attempted to manually perform the scattering operation with 2 particles, and so confirm if the union gave the same total field plot in the end. FOr 2 particles this is not particularly complicated and so I felt it was worth trying. While performing this by far the biggest complication is the unusual way in which OTT handles translations. T-matrices generated for particles not positioned at the origin are automatically moved there once this is applied, and there is no built-in method to translate the T-matrix. I made attempts to instead shift the beam to simulate the same scattering that would occur and then translate this image found afterwards, however when performing this subsequent translation OTT appears to recalculate the T-matrix interaction, and so the pattern is removed (as can be observed in the set of figures on the left, 2nd figure from the top). This lead me to create my own method for shifting the pattern observed as seen in the 3rd image. This will let me sum these E field components at the end of the calculation, but means I can no longer interact with the T-matrix. The 2nd figure shows how the manually

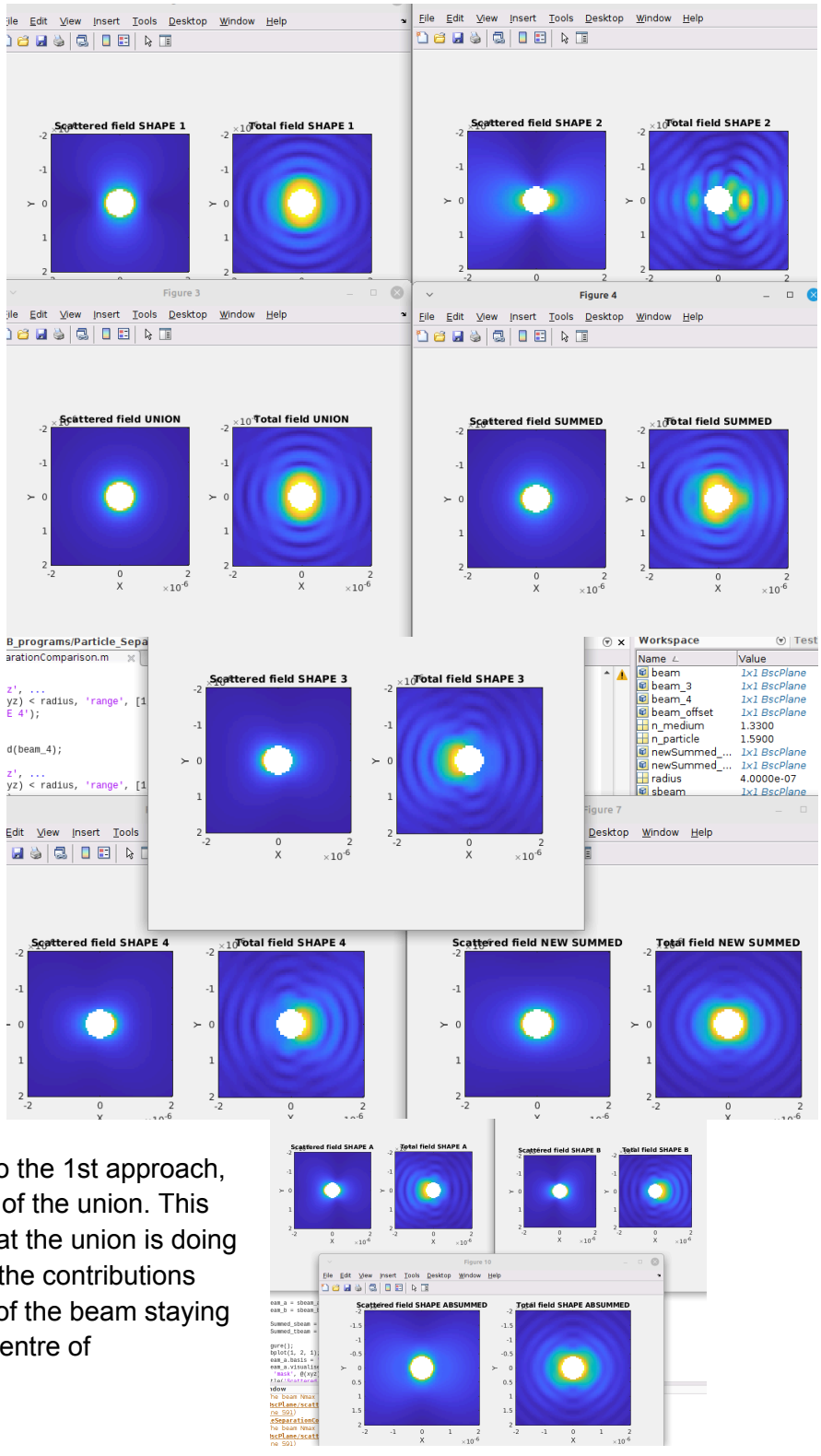




calculated field is almost the same as the union calculated field, which I believe would match if I could properly calculate the shift back needed for these particles, and so it seems likely that this union will work multiple particles, however I have sufficiently performed the calculation yet to prove this for certain.

13/10/24

I have been working on comparing the forces/electric fields experiences by two spheres separated in the X direction in a plane wave (linearly polarised in X dir) using the T-matrix approach, however first I have had to perform some tests to ensure that OTT (optical tweezers toolbox, the software I have been using for T-Matrix simulation) is working correctly. After speaking to my supervisor he brought to my attention that we did not have a good understanding of how this software performed its calculations and if they were in fact what we expected. This prompted me first to test whether the *'ott.shapes.Union()'* function was correctly calculating the interaction between the shapes in the union (inter-particle, not just exacting on each, summed). The plots on the right show me simulating the scattered and total (scattered + incident) field for *shape_1* and *shape_2*, where *shape_1* is located at the origin and *shape_2* is located at (3*radius, 0, 0) so they are separated and not overlapping (both particles also have the same radius). A simulation was done for these shapes individually AND with the union method, as well as including a plot of the two individual shapes summed together. I also did another test after this where I instead tried summing new particles *shape_3* and *shape_4* who were offset by +-1.5*radius in the x direction respectively. As can be seen, this second result appeared closer to the result the union had compared to the 1st approach, but still does not quite get the result of the union. This small difference supports the fact that the union is doing something more than just summing the contributions from each particle (in either regime of the beam staying at the origin or being shifted to the centre of



bounding-box frame), as I performed manually. This being said, the union does appear to nearly exactly resemble the plot of *shape_1*, and so maybe the answer is that it simply ignores offsets of shapes given to it.

Note that the offsets for shapes 2,3,4 where done by moving the beam instead, as offsets did not appear to be applied after the T-matrix acted on the single shapes (shifted them back to the origin, as seen by the plots showing the particles at (X,Y)=(0,0)).

```
shape_a = ott.shapes.Sphere(radius, [-1.5*radius;0;0]);
shape_b = ott.shapes.Sphere(radius, [+1.5*radius;0;0]);

T_a = ott.TmatrixMie.simple(shape_a, 'wavelength0', wavelength0, ...
   'index_medium', n_medium, 'index_particle', n_particle);
T_b = ott.TmatrixMie.simple(shape_b, 'wavelength0', wavelength0, ...
   'index_medium', n_medium, 'index_particle', n_particle);

beam_a = translateXyz(beam, [-1.5*radius;0;0]);
beam_b = translateXyz(beam, [ 1.5*radius;0;0]);

sbeam_a = T_a * ((beam_a) + (T_b * beam_b));    %Incident on [a] is beam_a and scattered part of [b]
sbeam_b = T_b * ((beam_b) + (T_a * beam_a));    % "" "" Vice versa

tbeam_a = sbeam_a.totalField((beam_a) + (T_b * beam_b));
tbeam_b = sbeam_b.totalField((beam_b) + (T_a * beam_a));

abSummed_sbeam = sbeam_a + sbeam_b;
abSummed_tbeam = tbeam_a + tbeam_b;
```

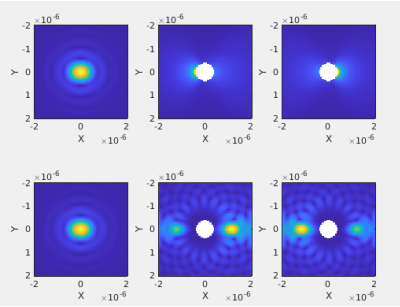
Another more careful look at how the scattering should be summed with 2 particles (by including the scattered contribution from the other particle in the incident field of the other) yielded the same results as before. This pattern does seem to be correct however as can be seen in X asymmetry caused by separating the particles

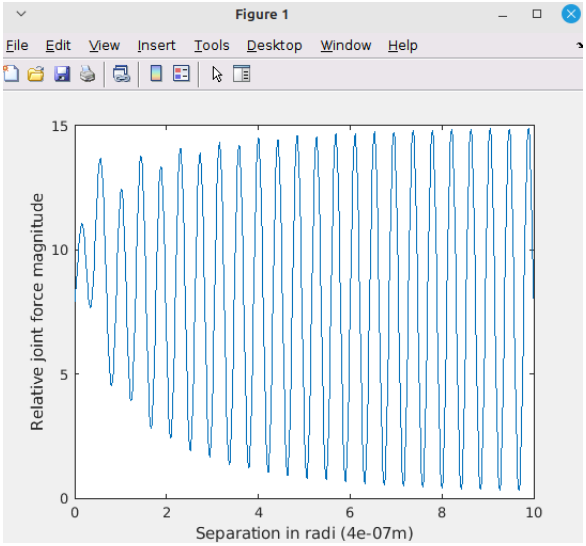
$$\mathbf{E}^{\mathrm{sca}} = \sum_{j=1}^{N} \mathbf{E}_j^{\mathrm{sca}}. \tag{65}$$

Because of electromagnetic interactions between the particles, the individual scattered fields are interdependent and the electric field exciting each particle is the superposition of the external field $\mathbf{E}_0^{\mathrm{inc}}$ and the sum of the individual fields scattered by all other particles:

$$\mathbf{E}_j^{\mathrm{inc}} = \mathbf{E}_0^{\mathrm{inc}} + \sum_{l \neq j} \mathbf{E}_l^{\mathrm{sca}}, \quad j = 1, \ldots, N. \tag{66}$$

Having done this, I think it would be safer to have a T-matrix for both particles and then manually ensure the cross-interaction terms are included (as dealing with 2 particles should be simple in this approach). However, I am now having trouble where it seems *'translateXyz()'* translates the scattered beams I find, but then also re-applies the T-matrix or performs some other calculator, as seen in a graph below where I simply want to shift the pattern I get from evaluating the T-matrix at the origin with a shifted beam at the the negative particle position, but instead of just shifting this pattern it produces an entirely new petal-like pattern.





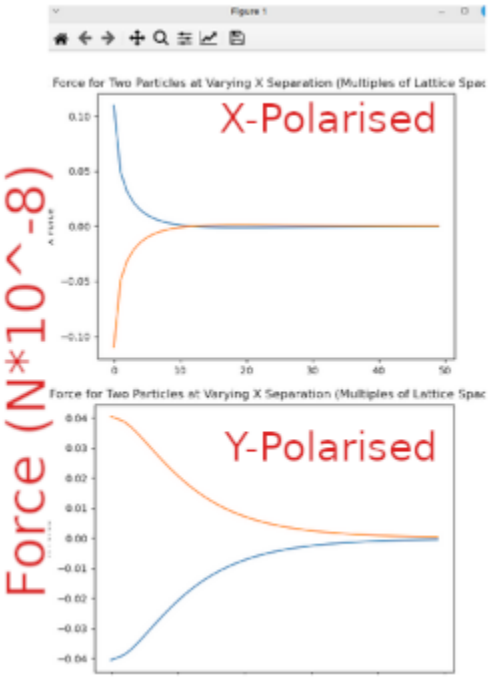
If we assume that the union() and forcetorque() functions are trustworthy and working, then the following plot is produced; which shows a trend of force (although this is the norm of the force, not x component, which does not appear to be working fully with my implementation of

forcetorque) similar to before with rapid decay to 0 force at ~ 10 radi separation, however this also some oscillatory term. I think this is not an issue however as the force found here is the net over both particles between the incident and scattered beam, which would be split equally between the 2 in the x direction (for the x component, ignoring the y and z component, which is likely causing part of this problem here), and so may have cause some doubling-up effect where the force is evaluated for some +- value as it switches between the 2 particles, then the norm is taken, resulting in this purely positive oscillating term. Clearly since the X component of force is not correctly being found there are some other issues with forcetorque() I need to investigate, but as an initial proof of concept this is not terrible in showing some agreement between DDA and T-Matrix.

12/10/24

As a small note I decided to change the title of these notes because as time has progressed studying this subject and talking to my supervisor about the aims of the project I feel now that the Abraham-Minkowski side of the project is far less important than the studying of the light deformation side. I did not want to change the title like this before as I thought it might lead to some confusion, however I now believe that it is far less confusing to just make this transition now to be more consistent with the desires of this project from here on.

I have now started working on a simpler angle to the ideas we are interested in (about the continuous limit of particles) where I will just compare how DDA and T-matrix models particles being brought closer together (by comparing a single component force normal to the surface being 'merged' with the other particle) with a simple beam (like a plane wave with Z propagation direction, hence forces and movement in the X or Y direction considered).



I have adapted the program written before for ADDA (written in python) that time-stepped particles to now instead just deal with 2 particles, and just test them at fixed lattice point differences. I vary the number of lattice separations (0 being directly touching) between the two, calculate the forces they experience on each dipole (calculated by ADDA) and sum them, then plot the resultant X force that each particle experiences. Note that due to the generation of ADDA's coordinates, when the particles are separated by a distance D, each particle will sit at +-D/2 respectively and the beam will be centred about the origin (irrelevant here for plane waves, but could be important later on). Because a plane wave moving in the Z direction was used here, the forces in the X/Y direction should only have occurred from the scattering experienced between the two particles (which are both spheres, hence symmetrical about Z, hence will not experience net X/Y forces from within themselves from the scattered incident light, just from the scattered light of the other particle creating an imbalance). This plot shows that value seems to blow up at a separation of 0 lattice separation, and decays for larger spacings (essentially 10 by 10 lattice separations). There is a clear difference between the 2 results ADDA simulated with the X and Y linearly polarised beam, however the fact that it blows up 0 separation and decays with distance appears the same, but I am not sure why these results appear so different (maybe this could be from some difference in phase, causing the Y polarisation to have forces starting from a lower value and then decay as usual after, as seen by its lower initial value). This system used two spherical particles each in a bounding box of

17x17x17, with *'-size 6 -wavelength 6'*, so a particle of width 6 micrometres and beam wavelength of 6 micrometres.

I want to experiment more with different size particles relative to the wavelength, as well as the effect of particles of fixed dipole number in a grid as the dipole width changes (e.g. particles scaling with the grid size) as well as the opposite (particles of fixed size in a changing lattice, a more interesting case for what is being considered but also considerable more expensive as I try to create a finer resolution lattice).

C.8 RadForce

This file stores the results for radiation force on each dipole (§11.6). A separate file is used for each simulated incident polarization: `RadForce-X` and `RadForce-Y`. It looks like

```
    x y z |F|^2 Fx Fy Fz
    -0.2094395102 -1.047197551 -3.141592654 2.348525769e-005 \
-0.001450116615 0.004378387866 -0.001487326185
    …
    0.2094395102 1.047197551 3.141592654 0.001840340346 \
0.01038653728 -0.04118221174 -0.006040333213
```

where "\" denotes continuation of the line. This file describes the dependence of the force vector on the coordinates inside the particle (in µm). All values are specified in the particle reference frame (§6.1). The squared norm of the force vector is presented for convenience. The unit of force is 10^{-8} dyne when all lengths are measured in µm and unity amplitude (1 statV/cm) of incident field is assumed (see §11.6 for details). The above file can be produced by the command

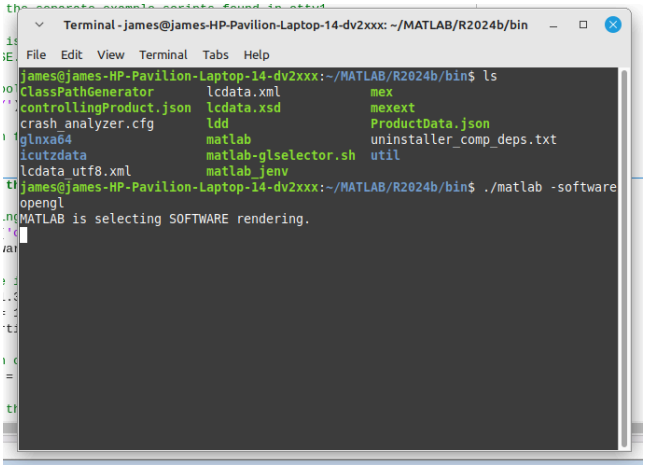
```
adda -store_force
```

I will create a similar plot from data taken from the T-matrix approach and see how that compares to the data found here. The T-matrix is generally a more trustworthy approach than DDA due to its more analytical nature (however both methods are numerically exact as more dipoles/terms are considered), and so agreement here would give reason to believe that a continuous limit of particles can accurately model a larger combined particle, or at the very least that both simulation methods have the same problem when trying to model this. Comparison to an actual combined case would allow us to compare forces and see if the continuous version tends to other or not.

I can also compare the electric fields found for both methods at each point and consider the difference of these values to see how each method compares to (as opposed to only measuring the forces in a component to compare them). Can also ensure Tmatrix is working properly (with cross scattering terms) by finding the field for each and summing them.

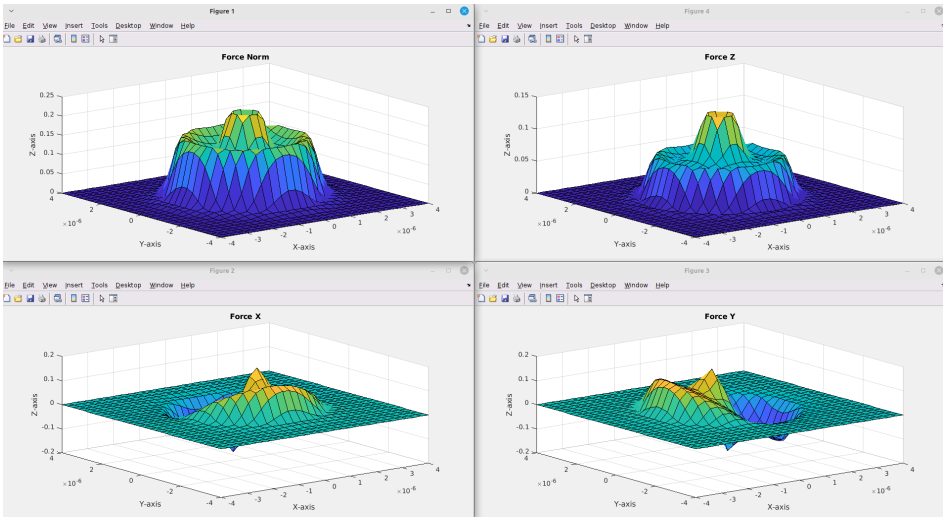
8/10/24 - 9/10/24

I have been learning how to use the optical tweezers toolbox (OTT), a tool that runs in MATLAB, as the previous piece of software TERMS did not have built-in method to generate new beam types (without modifying the source code of the software, which would be incredibly difficult given the lack of documentation about how the software works, as well as my own lack of experience performing that type of work).
Hence, in order to future proof the project, OTT seems to be a far better tool.



So far me and my partner have been working together to understand how to (1) use MATLAB (as neither of us have used this language) and (2) learn the capabilities of OTT. From this we found how to generate a T-Matrix for the some of the primitive objects in the program (e.g spheres, cubes) with an arbitrary translation applied to it, how to generate a beam that propagates in any direction, how to find the electric field from the scattered and incident beam and how to plot this data with the usage of masks to show where particles are located. This was tested for 2 particles manually added and treated with an *'ott.Shape.Union'* function.

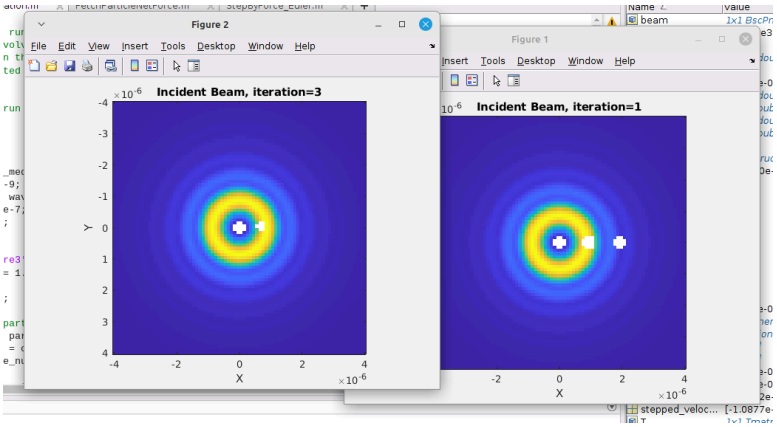
I also tested the Laguerre-Gaussian beam (which OTT natively supports) in order to test if the plots made sense, where I plotted the X,Y and Z components of force across the Z=0 plane where the beam sat, in which I then saw that the components of force pointed around the beam as expected.



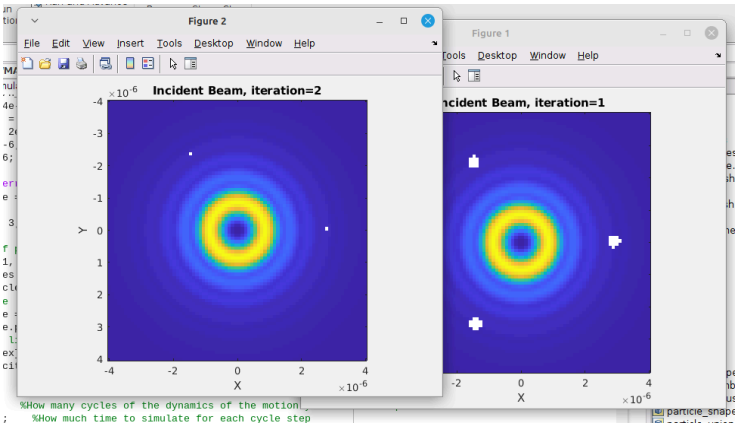
Something to note as well is that in order for MATLAB to run its graphical elements I have to launch the .exe with *'./matlab -softwareopengl'* (because the computer I run MATLAB on does not have a graphics card and may also lack some graphics drivers).

—

I have further extended the previous example to work with N particles in arbitrary locations, moved under the influence of the force from the beam, then created an animated plot from these plots. I am having trouble with the force calculation however as it does not want to take a 3D grid of points as an argument to find forces, only a 2D grid, and so the program is currently only summing forces in the Z=0 plane between the between the bounding box X and Y values around the particle (that accounts for scattering from other particles).



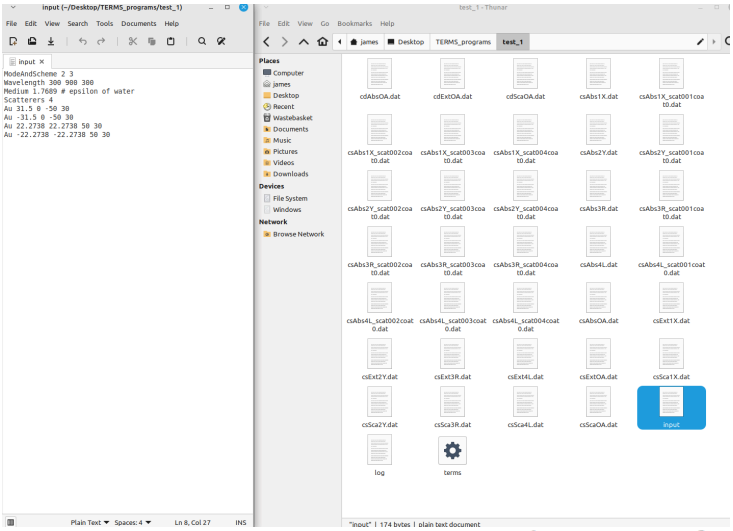
As a side note about the plot shown above, I believe the particles look like they are disappearing as the Z component of force is quite large (see the force plots further above) and so the masking calculation that checks if the particles within a spherical region about the position on the Z=0 plane becomes less true as the simulation progresses (particle is moved in Z direction).



Once these problems with forces and some display issues with the plots are fixed then we will be able to analyse the forces in the continuous limit in the Laguerre-Gaussian beam to see how well it is modelled (in terms of rotational forces) and so if flexible models can described by this limit of small particles closely packed and linked with spring forces.

7/10/24

I have continued reading the paper *"Multiple scattering of light in nanoparticle assemblies_ user guide for the TERMS program"* which explains how TERMS can be used. An example program is given in here which calculates scattering with 4 particles (specified as 'Au', gold particles), and as can be seen produced *'cs' and 'cd'* files for the orientation-averaged cross-sections and 'optical activity' respectively.



The T-Matrix method is made more accurate by increasing a maximum multiple expansion parameter, to generate more terms of the multiple expansion used (where you will see convergence eventually). This is also required when bringing particles closer together (more intricate calculation as each term holds more value due to low separation of components).

Varies wavelength to produce results through multiple calculations. The mode at the top of the input file specifies the parameters to calculate and output. Mode 1 => near-field calculations, 2 => far-field, 3 => calculate some properties about polarisation of the system (like certain cross sections).

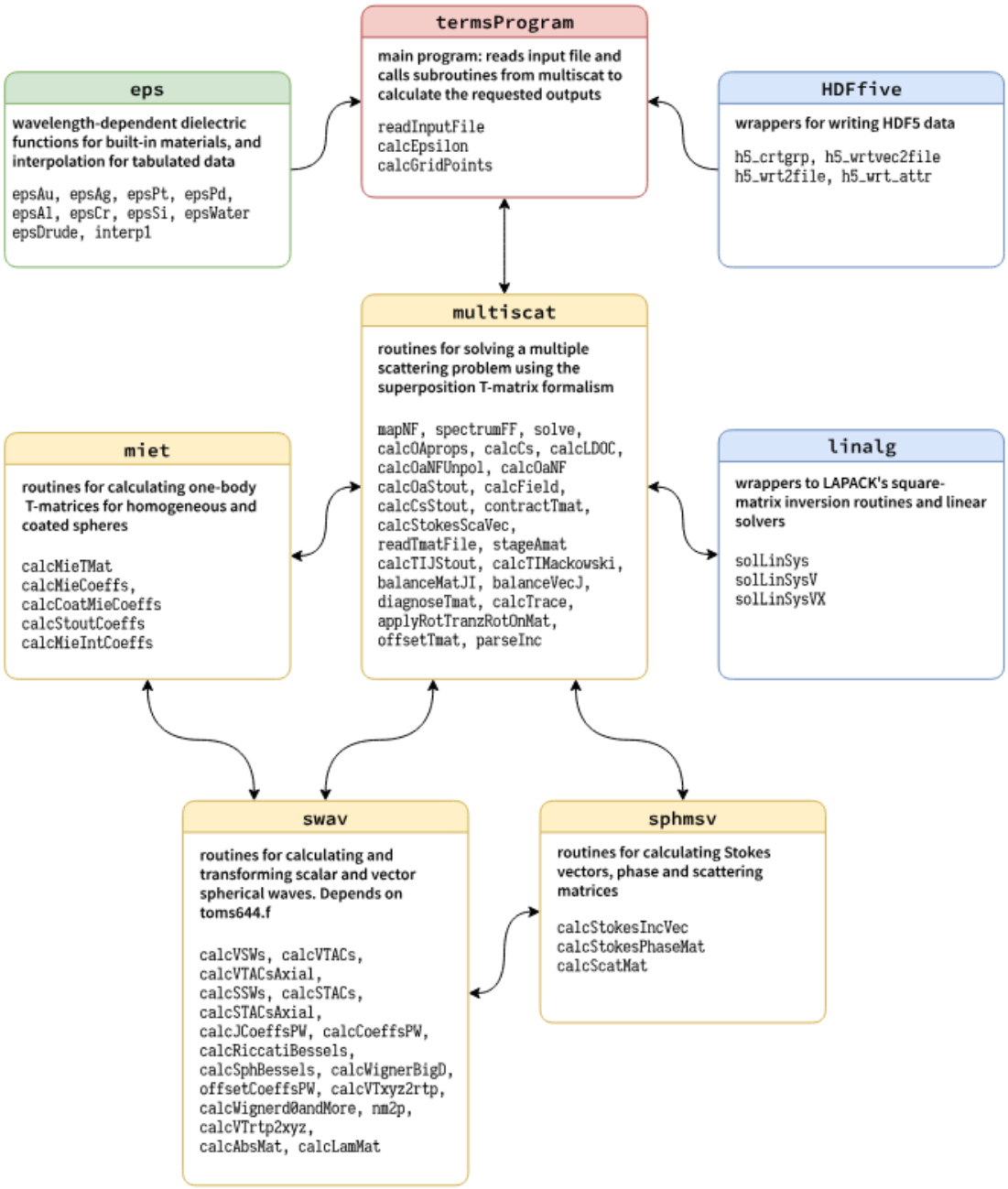
> Multiple-scattering generally introduces a loss of precision compared to single-particle calculations, and requires larger values of $n_{\max}$. The user is advised to consider the different solution schemes implemented in TERMS, as they can offer substantial benefits in specific situations. For instance, Stout and co-workers introduced a balancing scheme [15] that stabilises the numerical calculations and proves very effective for closely-spaced resonant particles. TERMS has extended this improvement to other schemes by default (controlled with the keyword `StoutBalancing`). A dramatic difference between Scheme 2 and 3 is observed when particles are widely-separated: our implementation of Mackowski & Mishchenko's scheme fails where separations are above a few hundred nanometres even at large $n_{\max}$, while Stout's scheme maintains good accuracy without requiring a $n_{\max}$ value much larger than dictated by the single-particle response. The key difference between the two schemes is that Mackowski & Mishchenko's translates all VSWFs to a common origin, while Stout's maintains particle-centred expansions throughout [14–18,61].

For non-spherical shapes, TERMS takes the T-Matrix for that shape in the input file given. Hence it can work with any given shape, so long as you have the T-Matrix for it calculated (*'TmatrixFiles' keyword*). The paper also states that TERMS can calculate for larger composite bodies made of smaller particles too.

http://nano-optics.ac.nz/terms/articles/Keywords.html

Near-field specific keywords

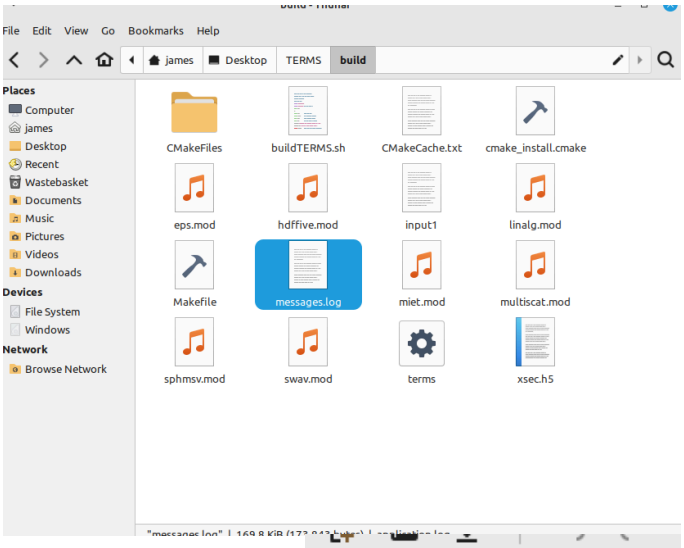
`SpacePoints filename` or
`SpacePoints xlo xhi nx ylo yhi ny zlo zhi nz`
Read (from a file) or calculate (on a regular grid) the cartesian coordinates of points in space, where the local field quantities are to be evaluated. The file's first line should contain the total number of space-points, and the subsequent lines must contain the *x*, *y*, and *z* coordinates of each point. A regular grid is specified by a closed interval, e.g. *[xlo, xhi]*, and the number of bins (*nx*) the interval is to be divided into (thus producing *nx+1* grid points along that dimension).



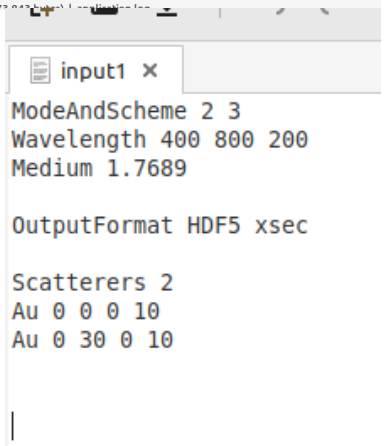
6/10/24

After speaking with my supervisor to discuss the problems me and my lab partner had with dealing with a DDA calculation that jumped to lattice points, and how this was supposed to work with us trying to test the continuous limit of particles being pushed together, we came to the conclusion that the T-Matrix approach to solving these problems may give better results instead. Hence work today has been focussed around learning about the T-Matrix and which software can be used to run this simulation.

A candidate I found for running this simulation was TERMS (T-Matrix Electromagnetic Radiation with Multiple Scatterers), which claims to be able to produce results for multiple particles of multiple shapes being scattered in a single system, and due to the nature of the T-Matrix these can be translated and rotated about the space completely freely. Getting this working proved somewhat difficult on linux, as I had to install a newer version of 'HDF5' and 'pthread.os' through the terminal using commands for *"libhdf5-dev" and "libpthread-stub0-dev",* as well as a module cmake required using *"sudo apt install libblas-dev liblapack-dev",* the module being called *"BLAS",* which is used for linear algebra-based calculations (clearly important here). After discovering these dependencies it was simple to use cmake to build the software.



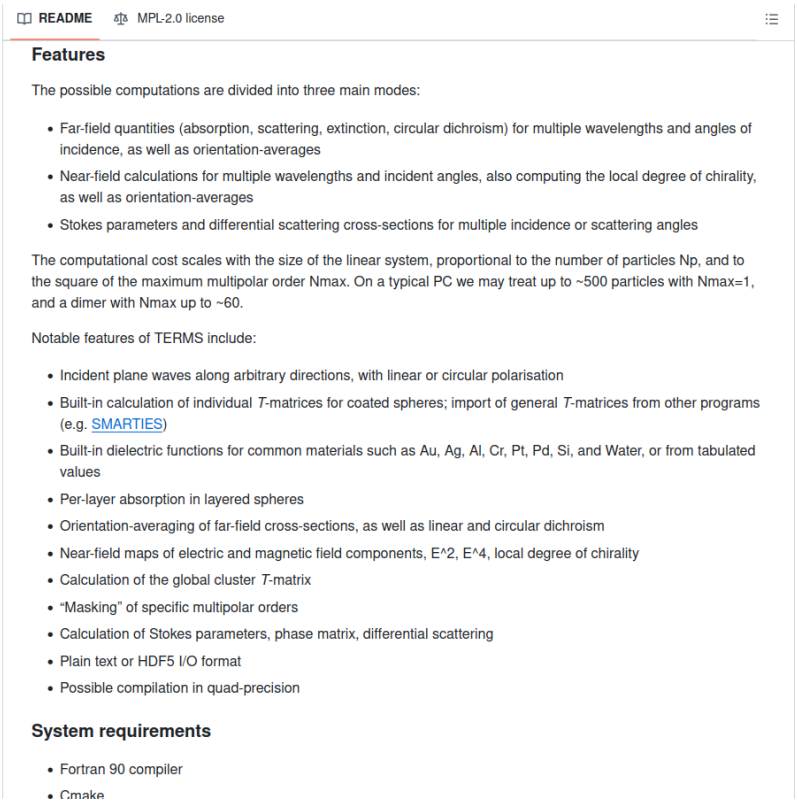
TERMS requires an input file to be included as an argument when running './terms' (the built exe) in the command line (e.g as ./terms [INPUTFILE NAME]), and an output file of the terminal log can also be made by including "> [OUTPUT NAME].log" at the end of the prior command too. Also, it appears the exe produces an output cross-section result too if specified in the input (using the test input, 'input1', a file called 'xsec.h5' is produced, which appears to be a complex structure to view). This input file specifies the output wanted, the refractive indices involved and parameters about the particles involved. This software has a user guide and paper to explain its usage further (as well as what appear to be thorough instructions on Github too).



The website 'ScattPort' (https://scattport.org/index.php/programs-menu/t-matrix-codes-menu) also lists several other programs that can perform similar calculations (although not all appear to support features like multiple particles, which should definitely be considered when deciding which to use), where i found *'scadyn'* and *'mstm-spectrum'* to be of particular interest too as they are recent programs which support multiple particles. Optical Tweezers Toolbox (OTT,

https://www.mathworks.com/matlabcentral/fileexchange/73541-ott-optical-tweezers-toolbox) was also suggested to us by out supervisor (hesitantly however as it is written in MATLAB which neither me nor my partner have used, and there appears to be less documentation surrounding this than other software too).

My next steps will be further learn how to use TERMS and explore some other options too, as well as research the fundamentals of the T-Matrix method more (this should be done carefully however as installing just this one program has taken a significant amount of time, and so I should be careful not to waste too much time by over-installing new software alternates).



README MPL-2.0 license

Features

The possible computations are divided into three main modes:

- Far-field quantities (absorption, scattering, extinction, circular dichroism) for multiple wavelengths and angles of incidence, as well as orientation-averages
- Near-field calculations for multiple wavelengths and incident angles, also computing the local degree of chirality, as well as orientation-averages
- Stokes parameters and differential scattering cross-sections for multiple incidence or scattering angles

The computational cost scales with the size of the linear system, proportional to the number of particles Np, and to the square of the maximum multipolar order Nmax. On a typical PC we may treat up to ~500 particles with Nmax=1, and a dimer with Nmax up to ~60.

Notable features of TERMS include:

- Incident plane waves along arbitrary directions, with linear or circular polarisation
- Built-in calculation of individual *T*-matrices for coated spheres; import of general *T*-matrices from other programs (e.g. SMARTIES)
- Built-in dielectric functions for common materials such as Au, Ag, Al, Cr, Pt, Pd, Si, and Water, or from tabulated values
- Per-layer absorption in layered spheres
- Orientation-averaging of far-field cross-sections, as well as linear and circular dichroism
- Near-field maps of electric and magnetic field components, E^2, E^4, local degree of chirality
- Calculation of the global cluster *T*-matrix
- "Masking" of specific multipolar orders
- Calculation of Stokes parameters, phase matrix, differential scattering
- Plain text or HDF5 I/O format
- Possible compilation in quad-precision

System requirements

- Fortran 90 compiler
- Cmake

4/10/24

I started reading *"The discrete-dipole-approximation code ADDA: Capabilities and known limitations"* to get a better understanding of how the calculations involved in ADDA / DDA in general work, specifically how the force calculations work (as it is likely these will have to be manually calculated for non-plane wave beams due to ADDA not implementing this).

Interaction term to find contribution from other dipoles, separate to incident field term.

For the 3rd screenshot, F vector is the *"scattering amplitude"*, NOT the force. This is used to find the scattering and mueller matrices (by also considering 2 other entirely separate incident polarised beams).

as macroscopic field [43]. It should be distinguished from the exciting electric field $\mathbf{E}_i^{\text{exc}}$ that is a sum of $\mathbf{E}_i^{\text{inc}}$ and the field due to all other dipoles, but excluding the field of the dipole i itself. Both total and exciting electric field can be determined once the polarizations are known:

$$\mathbf{P}_i = \overline{\boldsymbol{\alpha}}_i \mathbf{E}_i^{\text{exc}} = V\chi_i \mathbf{E}_i, \tag{11}$$

where $V=d^3$ is the volume of a dipole and $\chi_i=(\varepsilon_i-1)/4\pi$ is the susceptibility of the medium at the location of the dipole (ε_i is the relative permittivity). In the following we

5.2. Interaction term

A few formulations for the interaction term are known [1]. Currently, ADDA can use the simplest one (interaction of point dipoles), FCD (in other words, filtered Green's tensor [21]), the quasistatic version of FCD, and the Integrated Green's Tensor (IGT, [16]). The interaction of point dipoles is described by the Green's tensor

$$\overline{\mathbf{G}}_{ij} = \overline{\mathbf{G}}(\mathbf{r}_i,\mathbf{r}_j) = \frac{\exp(\mathrm{i}kR)}{R}\left[k^2\left(\bar{\mathbf{I}}-\frac{\hat{R}\hat{R}}{R^2}\right)-\frac{1-\mathrm{i}kR}{R^2}\left(\bar{\mathbf{I}}-3\frac{\hat{R}\hat{R}}{R^2}\right)\right], \tag{21}$$

where $\mathbf{r}_i$ is the radius-vector of the dipole center, $\mathbf{R}=\mathbf{r}_j-\mathbf{r}_i$, $R=|\mathbf{R}|$, $\bar{\mathbf{I}}$ is the identity tensor, and $\hat{R}\hat{R}$ is a tensor defined as $\hat{R}\hat{R}_{\mu\nu}=R_\mu R_\nu$. The filtered Green's tensor is

amplitude $\mathbf{F}$ for any scattering direction $\mathbf{n}$ is given as

$$\mathbf{F}(\mathbf{n}) = -\mathrm{i}k^3(\bar{\mathbf{I}}-\hat{n}\hat{n})\sum_i \mathbf{P}_i \exp(-\mathrm{i}k\mathbf{r}_i\cdot\mathbf{n}). \tag{27}$$

The amplitude and Mueller scattering matrices for direction $\mathbf{n}$ are determined from $\mathbf{F}(\mathbf{n})$ calculated for two incident polarizations [39]. Scattering cross section C_{sca} and asymmetry vector $\mathbf{g}$ is determined by integration of $\mathbf{F}(\mathbf{n})$ over the whole solid angle:

$$C_{\text{sca}} = \frac{1}{k^2}\oint \mathrm{d}\Omega |\mathbf{F}(\mathbf{n})|^2, \tag{28}$$

$$\mathbf{g} = \frac{1}{k^2 C_{\text{sca}}}\oint \mathrm{d}\Omega \mathbf{n} |\mathbf{F}(\mathbf{n})|^2. \tag{29}$$

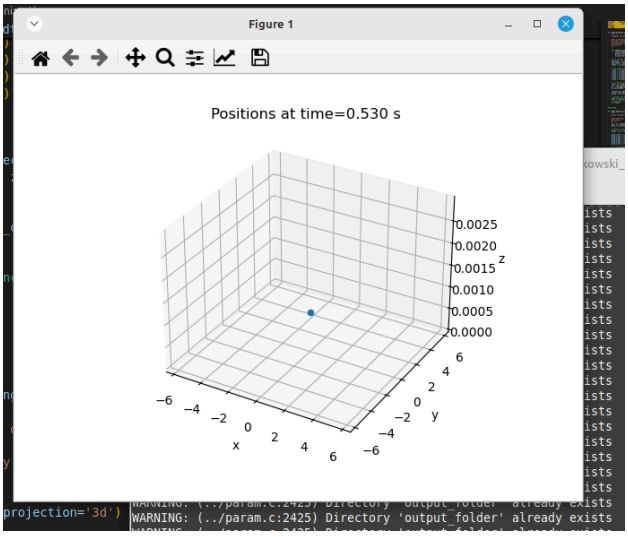
However, the latter features are still under development. In particular, the FFT-acceleration of radiation-force calculation [6] has not yet been implemented, limiting its applicability to relatively small number of dipoles.

6.3. Internal and near fields

1/10/24 - 2/10/24

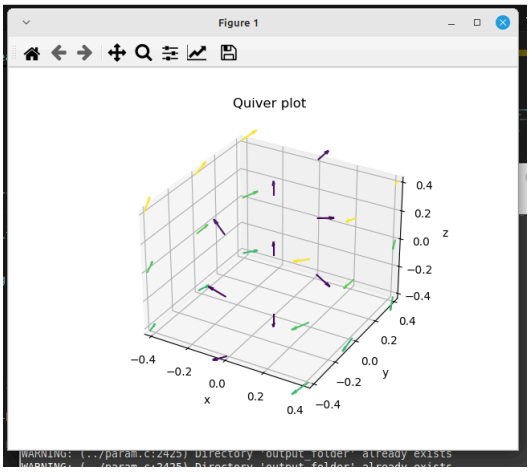
Work over these 2 days focused on producing an animation in which DDA calculations simulate the motion of a small particle within a beam. This was collaboratively programmed with my partner, in which we delegated half of the commands to each person to write. I wrote the functions to scrape values from the log, .geom and stored force files whereas David wrote the the plotter for the data to form an animation and to perform a quiver plot of forces at a given time frame. We both then shared other smaller functions as-and-when we finished out tasks to calculate forces, manage left over files and the passing of information between to different sections of the program in the required format and to string the process of DDA calculations together.

We tried to write the program as generally as possible so as many parameters as possible could be exchanged, however we did have trouble when considering multiple particles in one system as although ADDA supports multi-domain systems (wherein the .geom file can have “Nmat= N” specified to state that N geometries are stored in that file, and that each should be bounded individually to avoid excessive DDA calculation over empty space, which would be pointless) due to the nature of dipoles having to sit on lattice points for more efficient solving of the linear equations generated during the ADDA calculation we saw the problem of dipoles wanting to sit offset from lattice points, and so this would have to be resolved by snapping the dipoles to the nearest grid point. This was not a problem in the single particle case as ADDA allows you to offset the beam as well, and so we just offset the beam by negative-particle offset instead, which allowed non-integer offsets with no snapping, which was ideal. This appears to not be possible with multiple particles as each would want the beam to be offset by different amounts, which is not possible to do in a single simulation unless you calculated the interaction with the given particle and incident beam, THEN calculated the effect from each other dipole with their offset beam acting on the given particle, which is not possible in ADDA as the core functions for performing this calculation cannot simply be called by us without essentially rewriting ADDA independently. If we tried to handle this differently, we could move the geometries of the particle each DDA step (in the .geom file) and so not move the beam, then account for the change in the centre of bounds frame (acting on the particles as well as beam), then perform the next DDA calculation. The problem with this however is that because we are changing the geometries, we will end up with dipoles at fractions of a dipole-width offset, which will have to be resolved by snapping them to the near lattice points, which would then cause accuracy problems. The only way I can see this approach working is if each time you did this shift you dynamically calculated a new lattice width to best account for all the particles in order to minimise the snapping as much as possible (however I do not think this



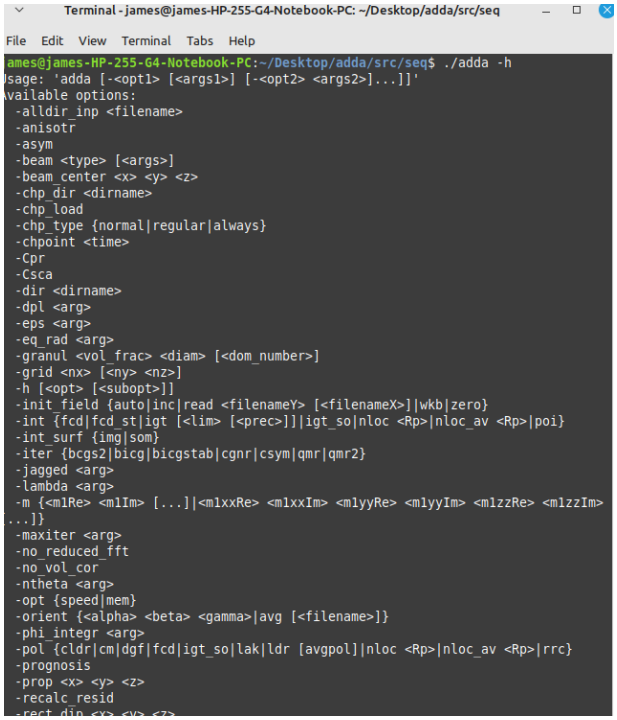
work particularly well as I think it would struggle to find the correct balance, and if it did you may have vastly more/less dipoles in order to make the spacing work and retain the particle width, and so the accuracy and timings of your simulation would constantly vary).

Also when experimenting with ADDA, we found that the dipoles specified in the .geom file are not fixed according a coordinate axis, but rather just specify relative positions to one-another, which are all then centred on the ‘centre of bounds’ (of the bounding box) of the particle given (as opposed to some centre of mass frame it could have moved to). The beam also defaults to being centred at the zero position here.



30/09/24

Work began on preparing ADDA to compute moving simulations, which started off with resolving the prior problem of using customs shapes inside the DDA which was broken for the ADDA-GUI version.

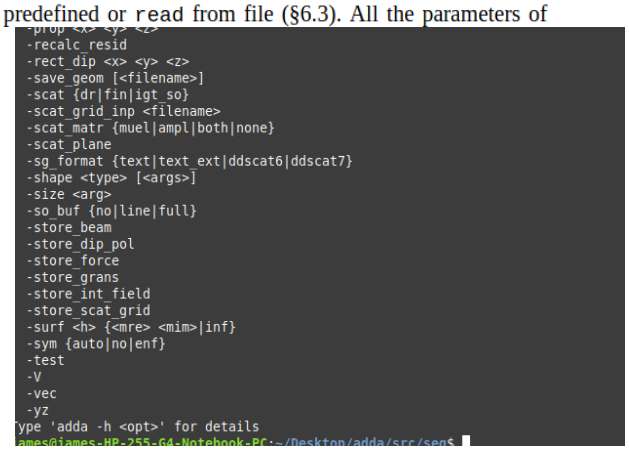


My plan was to run ADDA directly through the command terminal and avoid the GUI, but have struggled to understand the manual's explanation of how this was performed. After talking to my partner about this we noticed the GUI gave command line instructions as well for any given action, and so this presented the correct structure to follow. From this I could trivially test the different commands available (as seen when using the "./adda -h" help command) and found it was simple to perform the DDA calculation on my new shape I had defined.

Whilst testing these functions, I realised there was an even easier way to define shapes through a file specifying the positions of each dipole (in coordinates relative to a fixed dipole spacing). This was in the same format as dipole data pulled from the simulation too. As well as this forces, polarisabilities, scattering matrices, etc could also be easily pulled from the program into separate files, which therefore allows another program to interact with the data and queue up another calculation of the simulation.

`-sg_format {text|text_ext|ddscat6|ddscat7}`
Specifies format for saving geometry files (§6.3). The first two are ADDA default formats for single- and multi-domain particles respectively. DDSCAT 6 and 7 formats are described in §C.11. Default: `text`

`-shape <type> [<args>]`
Sets shape of the particle, either predefined or `read` from file (§6.3). All the parameters of



From this I can now without much trouble setup a python program to perform a series of DDA calculations on a particle, applying rigid body motion each time from the forces found, and applying an offset to the particle from an inverted offset on the beam (only the beam can be offset, particles seem to only be allowed to sit at the origin). Then the position, electric field, etc data can be saved each iteration and plotted as an animation to visualise the motion of the particles, which will allow us to test how the particles behave when considering a 'continuous limit' of smaller spheres forming a larger torus shape, which will let us assess the reliability of smaller particles modelling a larger particle, which can then be extended to linking these particles with springs to simulate some binding, which will let us simulate flexible motion in a more discrete case, which can then be tested with differing force

densities and application of Abraham-Minkowski differences, which can then be compared to reality to (1) asses the accuracy of the flexible simulation and (2) ascertain the validity of the combined solution to the Abraham-Minkowski problem. Note as well that ADDA also lets us specify specific beams similarly to the specifying dipoles for a shape in a .geom file.

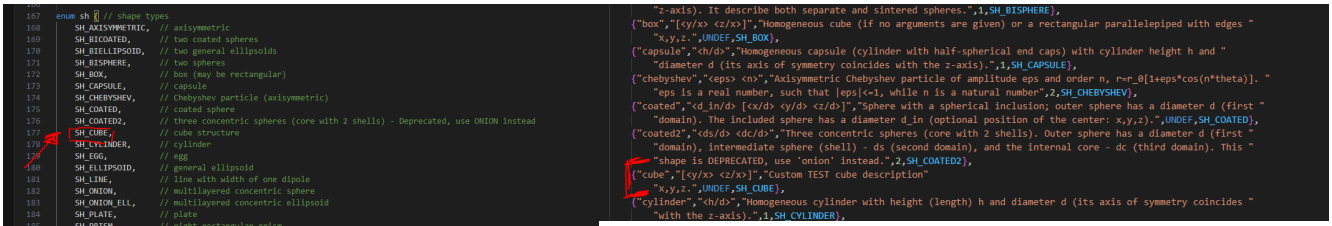
The only major problem with this approach I have seen so far is that ADDA currently does not have an algorithm to calculate the total forces produced on particles under the influence of a non-plane wave beam. I don't think this will be a fundamental problem however as there are approaches to calculating this manually, especially since we have arrays of the specific polarisabilities and electric fields present at all lattice points, and so should in theory to be calculate this without too much trouble (however making our approach efficient and fast could be difficult). Also, there appears to be a function called "-Cpr" that can be called that returns the 'force cross section', which appears to be some sort of measure of force on individual dipoles. I do not fully understand how this works and so this could also be a key value to help in calculating forces.

I also want to research the different iterative solver methods ADDA uses to solve the linear equations generated to calculate the polarisabilities and the DDA formulations it uses for different polarisability forms. It seems like it would be a very good idea to compare how well each combination of these perform. I also saw that ADDA seems to have functionality for multi-domains, which I think lets you split up the domain of DDA splitting over each particle in a multi-particle system, rather than having it applied over a larger single bounding area, which would have lots of wasted space.

As well as this, I was thinking about another approach for flexible/deformable DDA calculations where instead of lots of smaller particles you just work with a single mass of dipoles. In other approaches I have seen attempts to apply forces to just individual dipoles, however this can lead to problems of dipoles overlapping or having to be snapped to grid points (which can break too with multiple snapping to the same point). My idea is that the DDA calculation can be performed, forces calculated for each lattice point, but then a mesh of the outside of the particle can be generated (just over the surface). This mesh can then have forces applied to its vertices (off lattice points vertices could have their forces interpolated) to deform it, then it can have an algorithm either squash/pull it towards/away from its centre of mass until its volume is the same as before, then the mesh can be converted back to a grid of dipoles (or granules, which may be easier to deal with) and a new DDA simulation can be run. This process can then be repeated. Note that this means that sometimes due to discrete splitting of space to create dipoles the number of dipoles may change very slightly between calculations (or not at all if granules were used), but should overall average to the correct number (due to the fixed volume maintained each cycle).

29/09/24

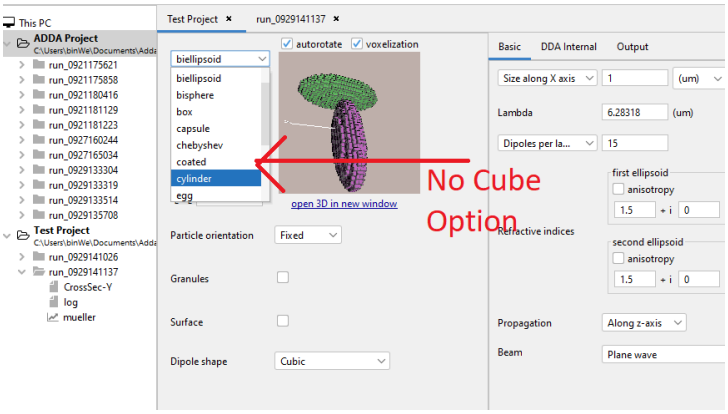
I have been attempting to create a custom shape for DDA analysis in ADDA. The github page for ADDA listed instructions for how to approach adding a new shape into the default list of options to choose from, which involved adding essentially a few lines of code in *const.h, param.c* and *make_particle.c*. These are all files that are compiled to create the 'adda.exe' and were very straight-forward to change. For simplicity I created a cube shape which was basically identical to the box shape already in ADDA, which I could then change and adjust once I saw it was working and being recognised. However, when I loaded ADDA-GUI (a separate program that has the interface for ADDA, and runs the 'adda.exe' file created by the other program) it did not recognise the changes I had made. This made perfect sense as I had forgotten to recompile the program, as in windows the compilation process was never described in detail by the authors, and instead a pre-compiled program was supplied for windows instead which was now going to cause some difficulty. Because of this I then attempted the same process on my linux version of ADDA and ADDA-GUI which I knew how to manually compile, however the same problem occurred where ADDA-GUI had not recognised the addition of this new information in ADDA. The ADDA manual is also not particularly helpful here as it only notes in section 6.4



about '*predefined shapes*' that ***"Adding a new shape is straightforward"***. Since ADDA-GUI

```
case SH_CUBE:
    if (fabs(yr)<=haspY && fabs(zr)<=haspZ) mat=0;
    break;
case SH_CYLINDER:
```

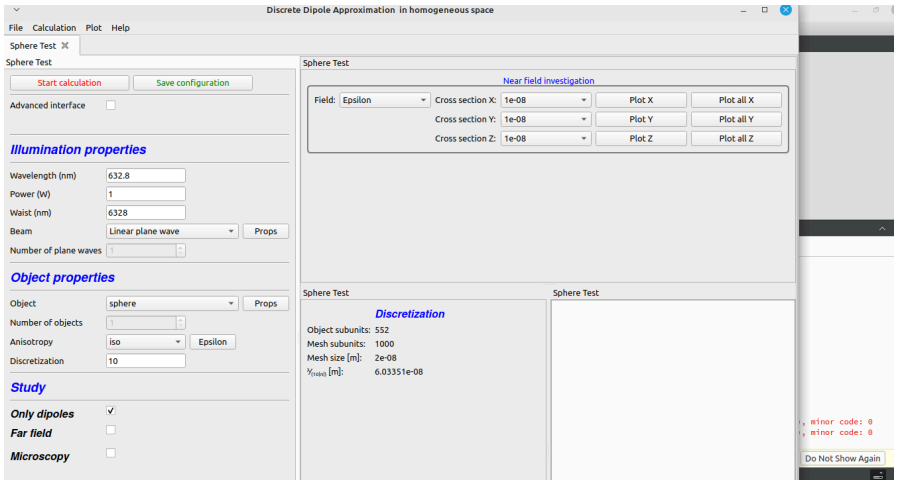
can be run stand-alone with predefined shape selection still available (however it will not be able to perform calculations like this, that requires its sequential or multithreaded .exe to be added) this suggests to me that ADDA and ADDA-GUI each hold their own lists of predefined objects, and so I need to make a change in both not just one or the other. This appears tricky however as ADDA-GUI has a very different file structure to ADDA, for example it does not contain *const.h, param.c or*

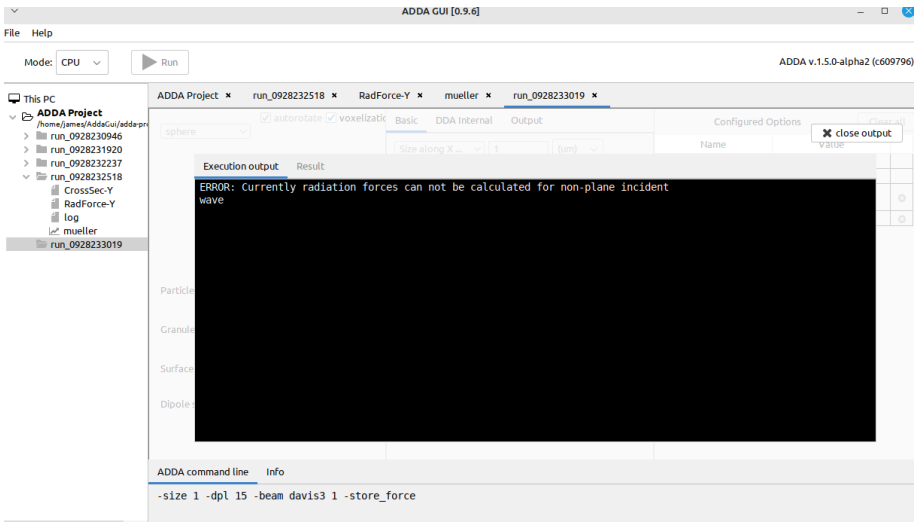


make_particle.c and so the changes I made cannot be swapped out here 1-for-1. What seems most likely is ADDA-GUI has some functionality to fetch changes made in ADDA which I simply have not called yet. If I cannot find a way to synchronise these changes I might have to try just running the program in ADDA with no GUI and see if the results are what I would expect.

28/09/24

Most of my work today has been on getting IF-DDA (another piece of DDA simulation software) to run on both my Windows desktop and Linux laptop with mixed success on both. This software (as oppose to ADDA) has given me significant difficulty, mainly in that it uses another piece of software called “Qt” to facilitate its visualisation, and the fact that it seems to be mostly written in fortran (which has historically given me problems with requirements of older packages in order to work). Patrick C Chaumet (one of the authors of this software) had a github page for this supporting version 4 and (at a push) 5 of Qt. Current installations of Qt default to version 6 and install an IDE called “Qt Creator” that works with this version. When attempting to run this project with an archived version 5 of Qt and Qt Creator other aspects of the program began to break when trying to build the cdm_install.pro file. For Windows, I have not been able to fix this problem (the closest I have gotten is up to the programming missing 1 library for “HDF5”, however this is very awkward to install and does not appear to have worked when I tried, despite suggestions that Windows seems to have a version of HDF by default), but for the Linux computer I have managed to build the program using Qt 6. The problem I am encountering for Linux here is that the test files provided (0-4) give a clean crash error whenever I run them (they do build correctly however, which is very unusual), and so it seems when I run the main GUI for the software (called “cdm”) I get some interface but no results, plots and very few buttons. THis may be due to low performance of my Linux laptop, as Qt appeared to struggle to load some functionality due to auto-detected computer specs, so this could be the cause of some of these crashes (however I think that is unlikely and it is probably related one of the newer libraries I had to install).

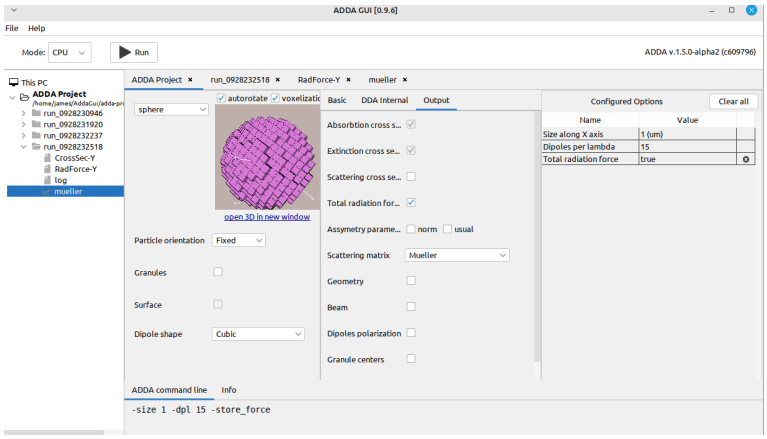




Note, the Linux version also works with Qt 6.2.0 (was originally on Qt 6.7.3), but with the same problem crashing on each test file.

User Guide; https://www.fresnel.fr/perso/chaumet/ifdda/userguide-EN.pdf

When considering ADDA again, I found here that for plane waves a scattering matrix and force value can be calculated, but for non-plane waves this is not allowed in ADDA. This is supposed to be the benefit of IF-DDA, however IF-DDA seems to have some other complexity making it hard to setup.



Next I need to investigate how to (1) put custom shapes into ADDA, (2) timestep particles then run another ADDA simulation and finally (3) assess whether ADDA or IF-DDA will be a more useful and flexible tool to prevent so much shifting/attention splitting between the two.

24/09/24 - 26/09/24:

Over these past few days I have primarily been reading more in-depth about specifics of how the DDA simulation is supposed to run, especially since the latest meeting with my supervisor confirmed a key detail that the DDA must be run N times to produce rigid motion (as most papers talked about the forces involved when particles/dipoles were in a given configuration, but did not address the fact that this wasn't handled within a single DDA calculation).

SImilarly I have been researching current ideas about dealing with more flexible / complex systems of particles, e.g. with dipoles that sit off grid axes or with multiple particles touching, as this is closely related to what me and my partner plan to explore first.

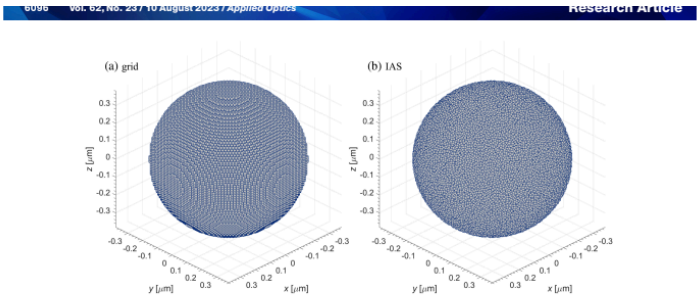


Fig. 1. Scatterer with an overall spherical shape with diameter 0.75 μm and comprised of 221,119 dipoles. (a) Dipoles are located on a regular cubical grid with a lattice constant of 10 nm. (b) Dipoles are located in an IAS configuration [39,40] with an average distance of 10.5 nm between nearest neighbors. Note that, for the purposes of illustration, the dipole locations are indicated as small spheres, but the polarizability formulation is that of cubical dipoles with the dipole volume according to Eq. (10).

We hope to start getting valid polarisabilities and forces from existing DDA software (either ADDA or IF-DDA seem likely currently) by importing a custom particle shape with a custom beam (just a simple sphere for now as it is well-known and analytically solved using maxwell's equations I believe, meaning results we find can be compared this solution). If this does not cause too many problems (which is entirely possible due to the nature of this software having a relatively low amount of documentation/advice when elements break) then I hope to also have the forces calculated/used from these results to generate a new particle position (rigid body motion, not on dipoles) and rotation to be input back into the DDA and so retrieve some motion (this also could in theory be quite difficult as we were warned that ADDA only gives force outputs for very simple beams, but IF-DDA should give forces in more cases however neither me nor my partner have tested this software yet, and so could be troublesome). If this works, then the situation can be expanded to have N spheres moving in the space, which can then be brought closer together to observe whether their angular momentums change as their boundaries are brought closer (surface integral area reduced => change in resultant force and torque from force densities). The result of this will indicate as to whether a flexible body can therefore be modelled with many smaller particles (possibly bound by elastic forces or maybe just packed tightly), or whether another approach (such as applying forces to dipoles individually and to deform the body, or by squashing and stretching cubic dipoles into rectangular dipoles may work, as

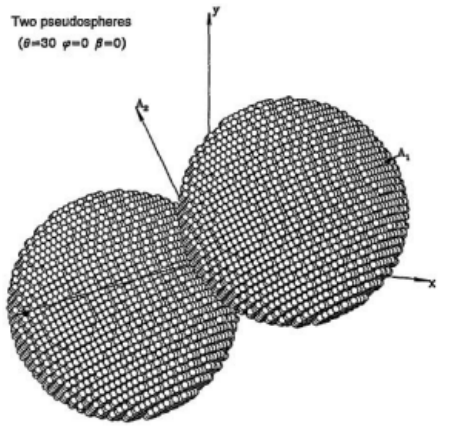


Fig. 1. Two pseudospheres composed of $2 \times 32 \times 32 \times 32$ point dipoles. Light is travelling along the x axis. Spheres are rotated ($\alpha = 30$).

demonstrated in papers I have seen mentioning the efficiency of rectangular dipoles in certain asymmetric situations)

I have used ADDA to run a simulation for a simple sphere model with a preset beam, and it did produce results, however I have not examined this to make sure the results seem physically accurate, and so this should be my aim next time I use ADDA.

16/09/24 - 22/09/24

Began reading papers on;
. What are optical tweezers and how do they work;
- ***"Optical tweezers and their applications", Paolo Polimeno, 2018***
- ***"Motion of atoms in a radiation trap", J. P. Gordon & Ashkin, 1979***

. What is the Abraham-Minkowski controversy;
- ***"A heuristic resolution of the Abraham-Minkowski controversy", Guoxo Feng, 2021***
- ***"Resolution of the Abraham-Minkowski Dilemma", Stephen Barnett, 2009***

. What is the 'Discrete Dipole Approximation' (DDA) calculating and what does it require in order to be calculated;
- ***"Discrete-dipole approximation for scattering calculations", Bruce Draine, 1993***
- ***"Coupled dipole method to compute optical torque: Application to a micropropeller", Patrick Chaumet, 2007***

These have helped me understand some of the core principles that this project requires, in that the controversy stems from the existence of canonical and mechanical momentum which may each have a larger contribution in different circumstances, and so certain experiments appear to only show one effect dominating. Some also argued that is a relatively common problem seen in other frames of reference problems. DDA calculations are now becoming a bit clearer in terms of why periodic lattices are used and how the electric field may be retrieved from the incident and scattered light. With DDA, however, I am still unsure as to how motion is fully simulated from these calculations (as the forces can be found, but do they need to be applied within the process of 1 DDA calculation, or do multiple DDA calculations need to be performed and time-stepped to achieve this?).

Researching optical tweezers has also given me a better appreciation of how light interacts in these types of systems (for example when using DDA, t-matrix, ray-tracing, etc) as well as motivating the idea of using light to manipulate these structures, which is the core message of this project. This being said, it is clear from what I have read so far that flexible systems are very much actively avoided due their unfriendliness when dealing with dipoles at fixed grid positions (small time-steps may under-shoot lattice points or try to occupy lattice points of other dipoles, obviously leading to unphysically large forces).

Our initial discussion with Simon Hanna looked at some ideas about optical tweezers how electric field gradients seen by different polarisations of light will influence the motion of particles inside the trap, cases where this may break (for instance when you are particles larger than the 'trapped area') and a thought experiment about the continuous limit of particles being brought together when being modeled with dipoles. Some software (ADDA, DDSCAT, etc) and authors (Masud Mansuripur, Steven Barnet, etc) were also talked about for further examples and explanation of material involved.

Personally, the biggest hurdle to overcome so far has been collating the wide range of views and different formulations for this topic, from different but also equivalent expressions for force and momentum densities in materials, whether their application is purely across surface or volume elements (which I would consider to be very closely related and so not too different due to ideas like the divergence and stokes' theorems, but appear to manifest quite differently here). Also the Abraham-Minkowski controversy is itself very divisive (as to be expected) which has lended to this confusion.

My plan is to try setup and use some of the available DDA software to see how it works in order to make some judgment as to what we should try and pursue further if we were to have a closer look at the continuous limit thought experiment proposed.