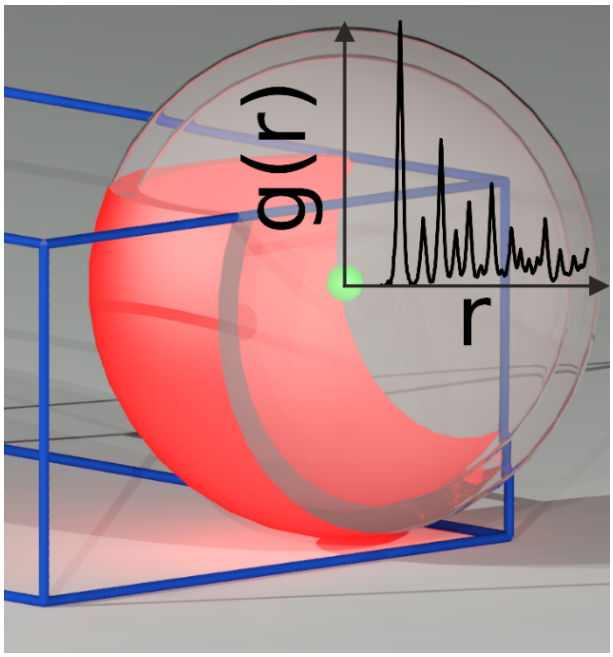
Supporting Information

Computing the 3D Radial Distribution Function from Particle Positions: An Advanced Analytic Approach

Bernd A. F. Kopera and Markus Retsch

Physikalische Chemie I
Universität Bayreuth
Universitätsstraße 30
95440 Bayreuth
Germany

Corresponding author: markus.retsch@uni-bayreuth.de



Contents

1 Theory

1.1 How to Compute $g(r)$ from Particle Coordinates

This section describes how the radial distribution function, $g(r)$, is computed from particle coordinates. First, we look at the relation between $g(r)$ and the local particle density

$$\rho(r) = \rho_{avg} \cdot g(r) \quad \Leftrightarrow \quad g(r) = \frac{1}{\rho_{avg}} \cdot \rho(r). \tag{1.1}$$

The average density, ρ_{avg}, is given by the number of all particles, N, divided by the known sample volume, V_{Sample}, like

$$\rho_{avg} = \frac{N}{V_{\text{Sample}}}. \tag{1.2}$$

The actual local particle density, $\rho_{\text{real}}(r)$, at one specific radius, r, is hard to obtain. Technically, $\rho_{\text{real}}(r)$, is defined by

$$\rho_{\text{real}}(r) = \lim_{\Delta r \to 0} \frac{\#\text{P}(\text{r} - \frac{\Delta \text{r}}{2}, \text{r} + \frac{\Delta \text{r}}{2})}{V_{\text{Shell}}(r - \frac{\Delta r}{2}, r + \frac{\Delta r}{2})}. \tag{1.3}$$

Here, $\#P$ denotes the number of particles in a spherical shell with the given inner and outer radii. This definition has some issues when it comes to actually computing $\rho(r)$ and $g(r)$. Both the volume of a spherical shell and the number of particles in the shell approach zero when Δr approaches zero. In practice, this issue is solved by using a finite Δr. The equation for the estimated local particle density is therefore

$$\rho_{\text{calc}}(r, \Delta r) = \frac{\#\text{P}(\text{r} - \frac{\Delta \text{r}}{2}, \text{r} + \frac{\Delta \text{r}}{2})}{V_{\text{Shell}}(r - \frac{\Delta r}{2}, r + \frac{\Delta r}{2})}. \tag{1.4}$$

Before we analyze the implications of using a finite, Δr, we take a look at the shell volume. The volume of a spherical shell is given by the difference of the inner and outer bounding sphere. Therefore, we have

$$V_{\text{Shell}}(r, \Delta r) = V_{sphere}(r + \frac{\Delta r}{2}) - V_{sphere}(r - \frac{\Delta r}{2}) = \frac{4}{3}\pi \left[\left(r + \frac{\Delta r}{2} \right)^3 - \left(r - \frac{\Delta r}{2} \right)^3 \right] \tag{1.5}$$

We simplify this equation by expanding the cube roots into

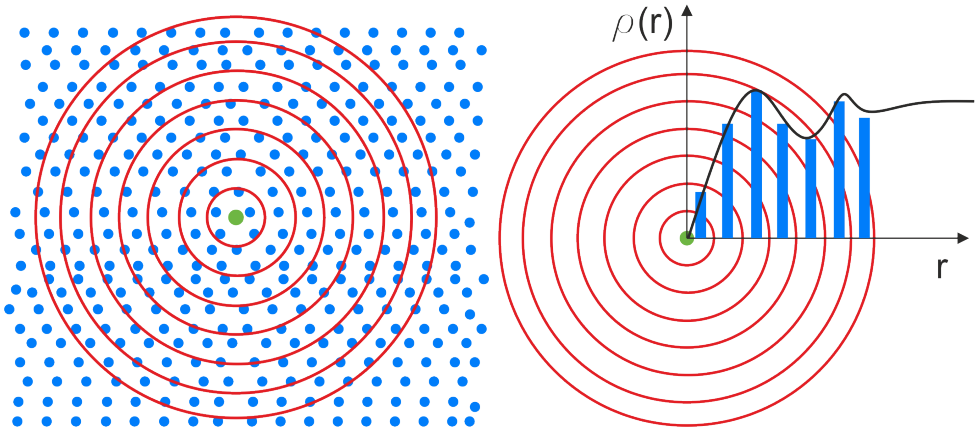
$$\frac{4}{3}\pi \left[r^3 + 3r^2 \cdot \frac{\Delta r}{2} + 3r \cdot \frac{\Delta r^2}{4} + \frac{\Delta r^3}{8} - \left[r^3 - 3r^2 \cdot \frac{\Delta r}{2} + 3r \cdot \frac{\Delta r^2}{4} - \frac{\Delta r^3}{8} \right] \right]. \tag{1.6}$$

Next, we add all terms with equal power, which leaves us with

$$\boxed{V_{\text{Shell}}(r, \Delta r) = \frac{4}{3} \cdot \pi \cdot \left[3 \cdot r^2 \cdot \Delta r + \frac{\Delta r^3}{4} \right] = 4\pi \cdot r^2 \cdot \Delta r + \pi \cdot \frac{\Delta r^3}{3}}. \tag{1.7}$$

This equation gives us a simple way to calculate the volume of a spherical shell. Equation (1.4) becomes

$$\rho_{\text{calc}}(r, \Delta r) = \frac{\#\text{P}(\text{r} - \frac{\Delta \text{r}}{2}, \text{r} + \frac{\Delta \text{r}}{2})}{4\pi r^2 \Delta r + \pi \frac{\Delta r^3}{3}}. \tag{1.8}$$



(a) (b)

Figure S1: (a) Slightly distorted cubic grid of particles. The red circles create spherical shells around the central green particle. Each shell has a certain number of blue particles in it. (b) Histogram showing the particle density, $\rho(r)$, as a function of radial distance.

Figure S1 illustrates the procedure defined in equation (1.4). We move the center of our radial coordinate system onto a single particle. The red shells have a thickness of Δr. The average density in each shell is the number of particles divided by the shell volume (see equation (1.2)). The radial distribution function around a single particle, i, is therefore given by

$$g_i(r, \Delta r) = \frac{1}{\rho_{avg}} \cdot \frac{\#\mathrm{P}(\mathrm{r} - \frac{\Delta \mathrm{r}}{2}, \mathrm{r} + \frac{\Delta \mathrm{r}}{2})}{4\pi r^2 \Delta r + \pi \frac{\Delta r^3}{3}}. \tag{1.9}$$

We now average over all possible particles

$$g_{\mathrm{calc}}(r, \Delta r) = \frac{1}{N} \cdot \sum_{i=1}^{N} g_i(r, \Delta r), \tag{1.10}$$

to get better statistics. Note, that we assumed a homogeneous particle distribution.

1.2 The Effects of Artificial Periodic Boundaries

In this section, we look at the effecs of applying artificial periodic boundaries to our sample. Figure S2 shows a small sample volume with periodic boundaries. The spherical shells for the g(r) computation are now always inside the sample volume.

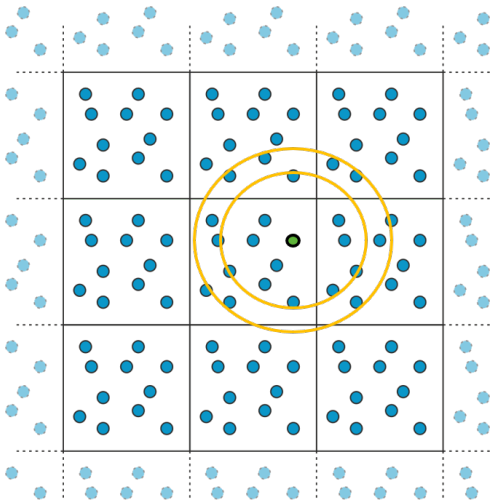
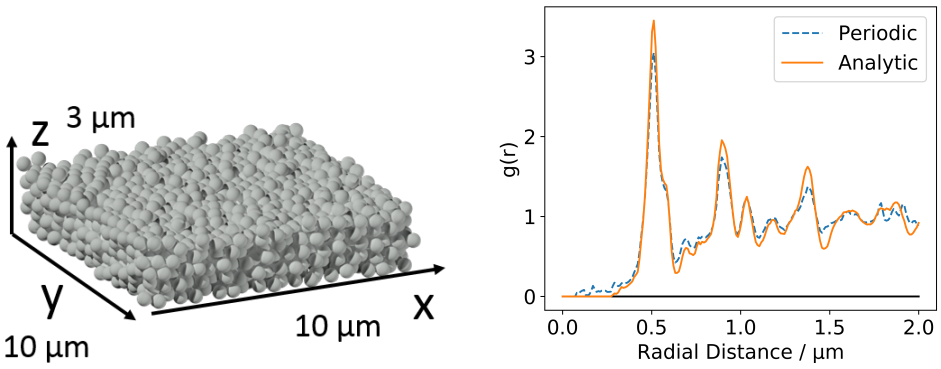


Figure S2: Small sample volume extended to infinity by periodic boundary conditions. Note the creation of artificial periodicities.



(a) (b)

Figure S3: (a) Colloidal assemble measured with confocal scanning microscopy and analyzed with TrackPy. (b) Radial distributiong function for the particle assembly in a. Periodic boundaries distort the peak shape and position.

The effects of periodic boundaries are most pronounced for small samples. Larger finite sample volumes are less affected by periodic boundaries. The main effects are:

- Stitching errors from unknown sample boundary positions
- Artificial periodicities, especially in small samples
- Rough boundaries from missing particles distort g(r)

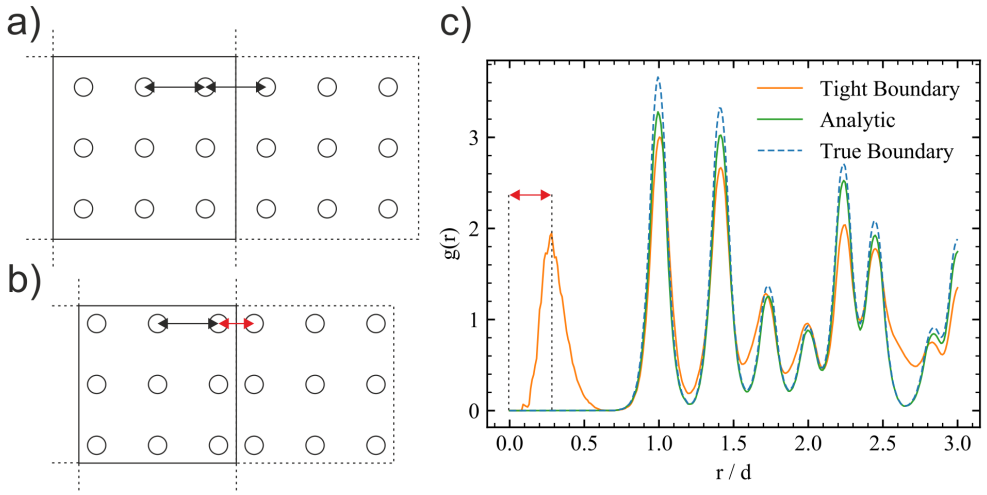


Figure S4: a) Simple cubic lattice with periodic boundaries a half next neighbor distance away from the outer most particles. No stitching errors (artificial periodicities) are created. b) Same lattice as in a) but with a tighter boundary. The red distance is artificial. c) g(r) of a simple cubic lattice with 2197 particles arranged in a cubic sample volume. The unit cell has size 1 and the particle positions are distorted with a 3D Gaussian blur ($\sigma = 0.07$). The graph labeled true boundary corresponds to a) and the tight boundary to b). Our analytic solution uses a tight boundary but reproduces the true boundary case well. Note the large peak at $r < 1$ created by the stitching error when a tight periodic boundary is used. The size of this peak strongly depends on the particle number in the sample and the sample shape. The position of the peak is at $r > 1$ when the boundary is more then a half next neighbor distance away from the outer most particle.

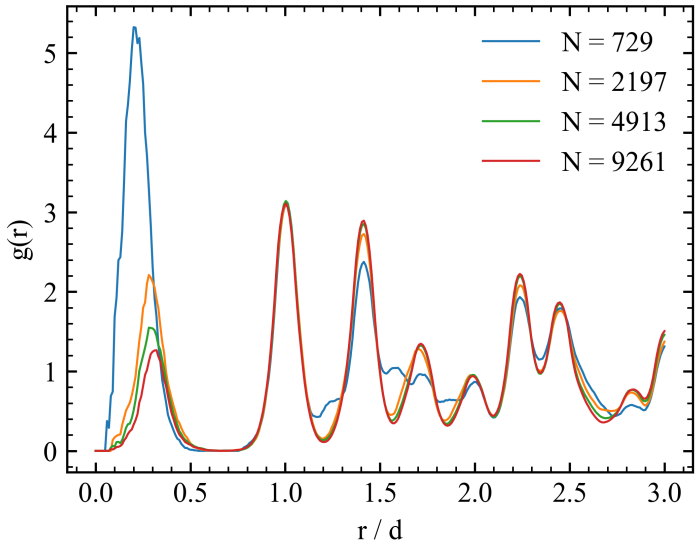


Figure S5: g(r) of a simple cubic lattice arranged in a cubic sample volume similar to Figure S4 c). We increase the number of particles in the sample volume from 729 to 9261. The unit cell is 1. The artificial peak at $r < 1$ is reduced when the particle number is increased. We attribute this decrease to the decreasing surface to volume ratio.

1.3 The Effects of Averaging Over a Small Subvolume

In this section we look at a way of avoiding the normalization problem. We avoid it by confining the averaging over the local g(r) to a small sub-volume. At the same time, we limit the radial distance as shown in Figure S6. The spherical bins now never reach beyond the finite sample volume.

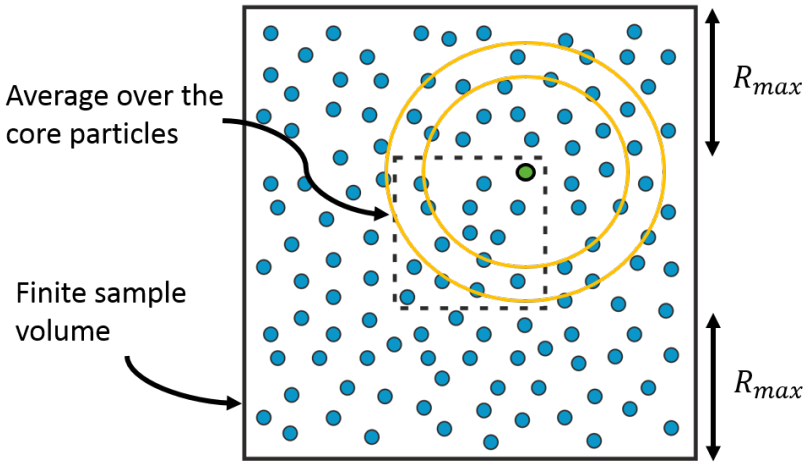


Figure S6: Finite sample volume with a smaller subvolume (dashed lines). The radial distance is limited to R_{max}. Averaging is only perfomed over the particles in the subvolume. The spherical bins (yellow) never reach beyond the sample boundaries.

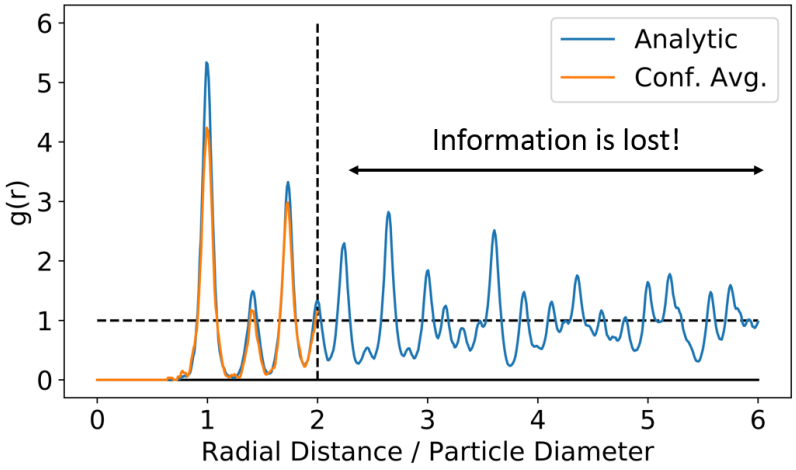


Figure S7: Radial distribution function for our FCC test data set. The confined average algorithm produces the same g(r) as the analytic algorithm. However, radial distances beyond R_{max} are not accessible and information is lost.

The main effect of this method is a limited radial distance. There is also a trade-off between radial distance and noise reduction. Less particles are available for averaging, increasing the noise.

1.4 The Intersection Volume Between a Shell and a Box

In this section, we derive the intersection volume between a spherical shell and a rectangular box. During the calculation of $g(r)$, the volume of a spherical shell, occupied by particles, is needed. In practice, the box where we have information about the particle positions is finite. Therefore, we need the intersection volume between the spherical shell and the box to properly normalize $g(r)$. We assume that the center of the spherical shell is always inside the box. This assumption is valid since the center of our shell is a known particle. Furthermore, we assume that the walls of the box are perpendicular to each other.

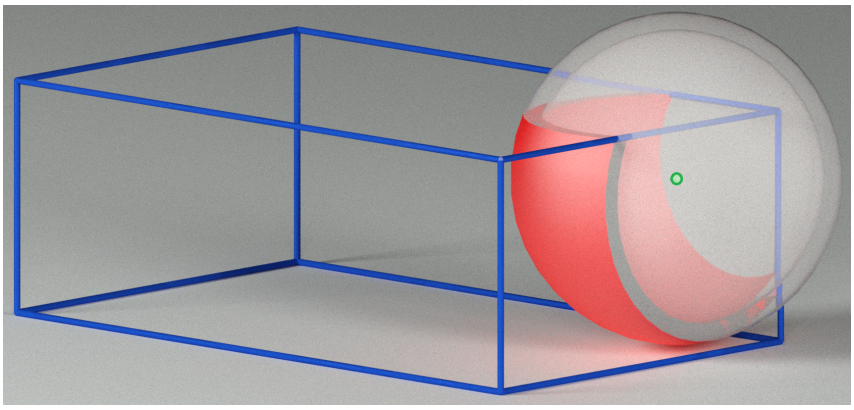


Figure S8: Intersection volume (red) between a spherical shell and a rectangular box (blue). The green particle is at the center of the spherical shell. The red volume is needed to normalize $g(r)$.

An example rendering of a spherical shell, protruding outside of a box, is shown in Figure S8. The red volume is the intersection volume between the spherical shell and the blue box.

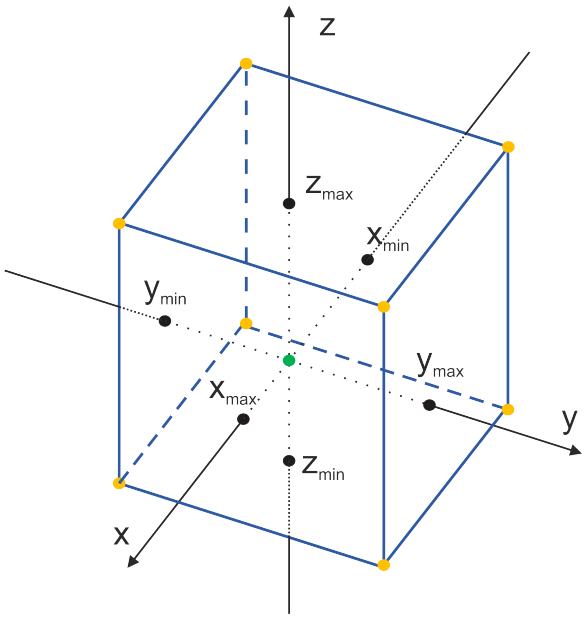


Figure S9: Definition of the box boundaries. Without loss of generality, we assume that the center of this coordinate system is at the center of the current particle. We can always shift the center of the COSY because the coordinates of the central particle are known.

The boundaries of the box are defined in Figure S9. The center of our Cartesian coordinate system is at the center of the special shell. To calculate the red intersection volume we proceed as follows. First, we dived the problem into smaller subproblems that are easier to solve. We then solve the subproblems and reassemble their solutions into the final solution.

We note that the volume of a spherical shell is given by the volume of the inner sphere minus the outer sphere like

$$V_{\text{shell}} = V_{\text{outer sphere}} - V_{\text{inner sphere}}. \tag{1.11}$$

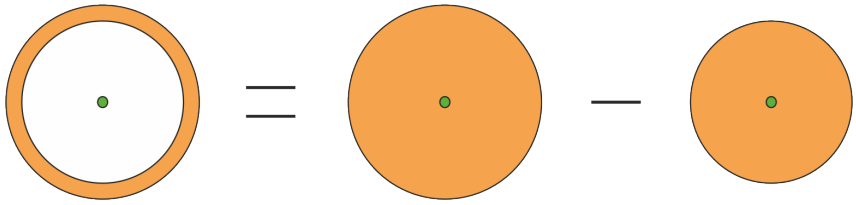
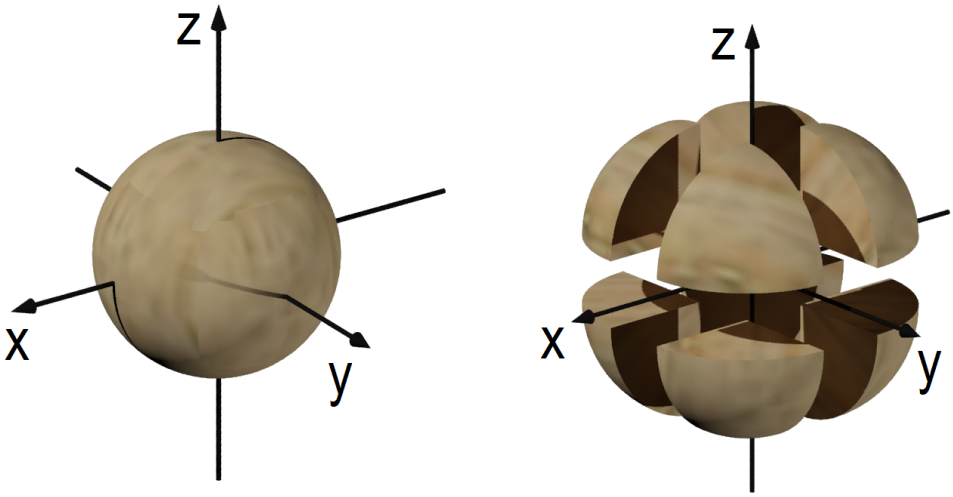


Figure S10: The volume of a spherical shell is given by the difference of the inner and outer sphere. This is true as long as the inner sphere is fully inside the outer sphere.

A graphical proof of equation (1.11) is given in Figure S10. The intersection volume between the smaller sphere and the box is always completely inside the intersection volume of the larger sphere and the box. Therefore, equation (1.11) also holds when both spheres are intersected by a box. We reduced our original problem to the problem of finding the intersection volume of a sphere and a box.



(a) (b)

Figure S11: Separation of s sphere (a) into eight octants (b) along the planes of the Cartesian coordinate system. The octants are moved away from the center for clarity.

Next, we use the intrinsic symmetries present in a sphere and a box. To do so, we split the sphere into eight octants as shown in Figure S11. Note, that the walls of our rectangular box are perpendicular to the axes of the Cartesian coordinate system. Three walls of our box cut into each octant at most, since the center of the sphere is always inside

the box. Therefore, we reduced the original problem to finding the remaining volume of an octant intersected by three perpendicular boundaries. Overall, the sphere volume is then given by the sum of the remaining octant - box intersection volumes like

$$V_{\text{sphere}} = \sum_{i=1}^{8} V_{\text{Octant}}(i). \tag{1.12}$$

For simplicity, we mirror all octants into the first octant bounded by three positive axes. This mirroring reduces the number of different cases significantly. Mirroring preserves the volume. The labels for the new boundaries are shown in Figure S12.

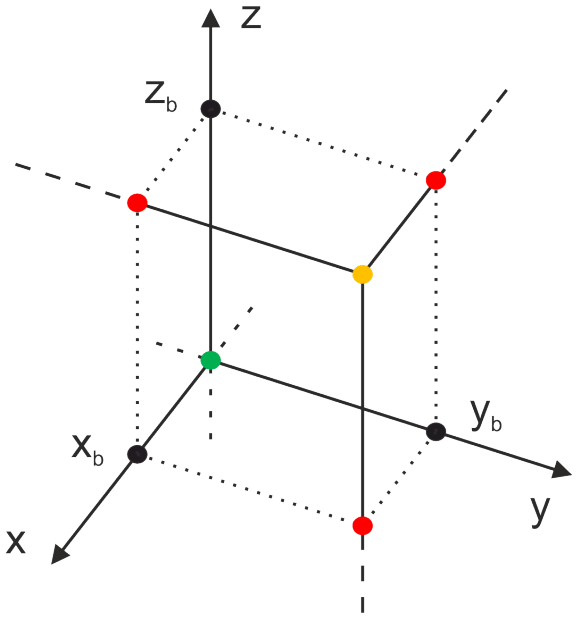


Figure S12: The three boundaries: x_b, y_b, and z_b, defining the dimensions of the corner. Note that we can always mirror a corner into the all positive octant of the COSY. This does not affect the intersection volume between the corner and the sphere octant.

We are looking for a simple algorithm to calculate the intersection volume between a sphere and a box. Using the above definitions and simplifications we arrive at the following function:

```
1  def SphereVolume(Rs, Xmin, Xmax, Ymin, Ymax, Zmin, Zmax):
2
3      VSphere = 0
4
5      # abs() mirrors the boundaries into the first octant
6      for xb in [abs(Xmin), abs(Xmax)]:
7          for yb in [abs(Ymin), abs(Ymax)]:
8              for zb in [abs(Zmin), abs(Zmax)]:
9
10                 VSphere += OctVolume(Rs, xb, yb, zb)
11
12     return VSphere
```

This leaves us with the challenge of finding a simple solution for the intersection volume of a box corner with a sphere octant (OctVolume(R_s, x_b, y_b, z_b)). Again we look for patterns that might simplify this task. First we look at the case that the intersection volume is only bounded by the box and not by the sphere. This is the case when

$$x_b^2 + y_b^2 + z_b^2 \leq R_s^2. \tag{1.13}$$

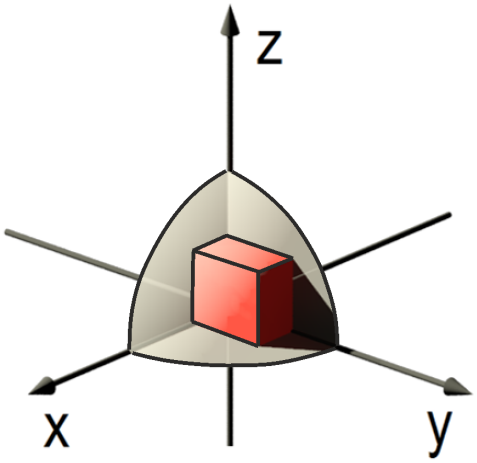


Figure S13: Intersection volume (red) between a box corner and an octant. The box corner is fully inside the octant.

Figure S13 shows the case of a box corner inside the octant. In this case, the orange box corner in Figure S12 is inside or on the sphere. Therefore, the intersection volume is given by:

$$\text{OctVolume}(R_s, x_b, y_b, z_b) = x_b \cdot y_b \cdot z_b. \tag{1.14}$$

This is a special case and we have to treat it as such. Next, we look at a single plane intersecting the octant. This situation is shown in Figure S14.

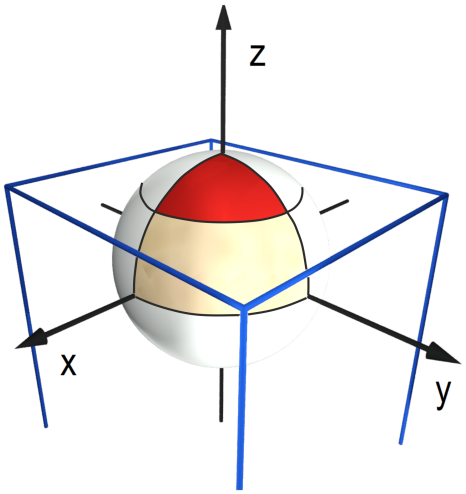


Figure S14: Single boundary intersecting the sphere octant. A quarter of a spherical cap (red) is removed from the octant by the plane (blue).

A single plane removes a quarter of a spherical cap (red) from the octant. For now, we assume that the three planes do not intersect themselves. A boundary plane, B, intersects the octant only when it is closer to the origin than the sphere radius:

$$B < R_s. \tag{1.15}$$

In this case, we subtract a quarter of a spherical cap, derived in appendix 2.2,

$$\frac{1}{4} \cdot V_{\text{cap}}(R_s, B) = \frac{\pi}{4} \cdot \left[\frac{2}{3} \cdot R_s^3 - R_s^2 \cdot B + \frac{1}{3} \cdot B^3\right] \tag{1.16}$$

from the octant. The last thing left to consider is the overlap between two spherical caps. Two boundary planes, a, and b, intersect each other inside the octant, when

$$a^2 + b^2 < R_s^2. \tag{1.17}$$

This case is shown in Figure S15.

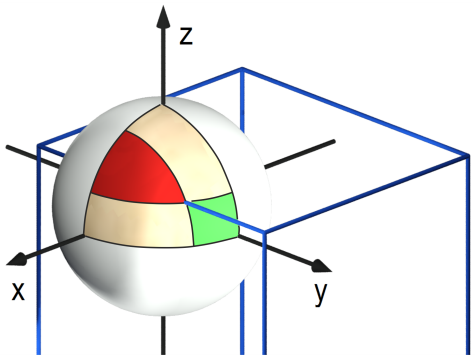


Figure S15: Two boundaries intersecting the octant. The boundaries also intersect themselves inside the octant. During the first step, we subtracted the red volume twice from the octant. Therefore, we have to add the intersection volume between the two spherical caps (red).

For simplicity, we name the intersection volume of two spherical caps a spherical cut. The volume of a spherical cut is derived in section 1.5. For every pair of boundaries, we have to check if they intersect. If they do, we need to add the spherical cut volume, since we removed it twice when we removed the spherical caps. Overall, we obtain the following algorithm for the volume of a sphere octant intersected by up to three planes:

```
def OctVolume(Rs, xb, yb, zb):

    # if all boundaries intersect inside the octant
    if xb**2 + yb**2 + zb**2 < Rs**2:

        return xb * yb * zb

    # if they dont intersect we start with a full octant
    VOctant = 1/8 * 4/3 * pi * Rs**3

    # then, we remove the spherical caps
    for B in [xb, yb, zb]:

        if B < Rs:
            VOctant -= pi/4*(2/3*Rs**3-B*Rs**2+1/3*B**3)

    # finally, we add the intersections of the caps
    for (a, b) in [(xb, yb), (xb, zb), (yb, zb)]:

        if a**2 + b**2 < Rs**2:

            VOctant += VSphereCut(Rs, a, b)

    return VOctant
```

With these two algorithms and equation (1.11) we are able to efficiently calculate the intersection volume between a spherical shell and a rectangular box.

1.5 The Volume of a Spherical Cut

In this section, we derive the volume of a spherical cut. Without loss of generality, we limit ourselves to the X- and Z-boundary planes. Figure S16 shows a 3D rendering of the spherical cut in red. The red volume is created by slicing a sphere twice, once perpendicular to the X-, and once perpendicular to the Z-axis.

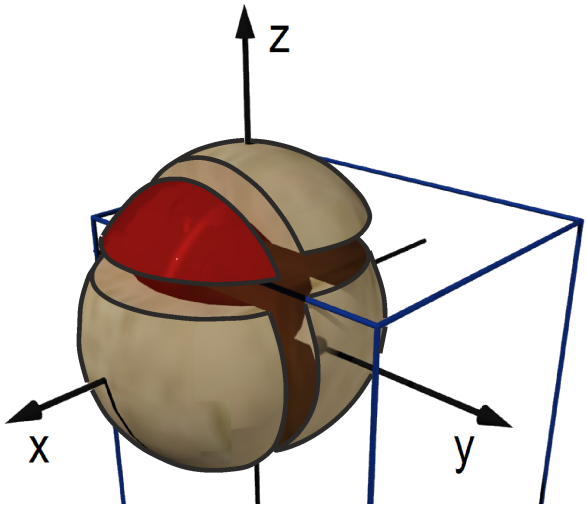


Figure S16: Spherical cut (red) created by slicing a sphere with two planes perpendicular to the X and Z axis. The parts created by the cuts are exploded for better visualization. Note that we defined the cut volume to be half of the red volume.

The integral to obtain the red volume directly is hard to solve. Therefore, we look for a decomposition into easier integrals. We choose the decomposition shown in Figure S17.

Figure S17: Decomposition used to calculate the volume of a spherical cut (red) by adding and subtracting different volumes from the spherical part. Note that the volumes are rendered from two octants for easier understanding. Only the internal volume (green) is hard to obtain.

Using this decomposition, the volume of a spherical cut is given by

$$V_{\text{cut}}(R_s, x_b, z_b) = \frac{1}{8} \cdot V_{\text{sphere}}(R_s) - \frac{1}{4} \cdot V_{\text{seg}}(R_s, x_b) - \frac{1}{4} \cdot V_{\text{seg}}(R_s, z_b) + V_{\text{int}}(R_s, x_b, z_b). \quad (1.18)$$

All volumes, except for the green, internal volume, V_{int} are known. We insert the volume of the spherical segment derived in section 2.1 and get

$$V_{\text{cut}}(R_s, x_b, z_b) = \frac{1}{8} \cdot \frac{4}{3}\pi R_s^3 - \frac{\pi}{4} \cdot \left[R_s^2 \cdot x_b - \frac{x_b^3}{3}\right] - \frac{\pi}{4} \cdot \left[R_s^2 \cdot z_b - \frac{z_b^3}{3}\right] + V_{\text{int}}(R_s, x_b, z_b). \quad (1.19)$$

The analytic solution for the internal volume, V_{int}, is given in appendix 2.3. We insert the solution for V_{int} into equation (1.19) and obtain

$$V_{\text{cut}}(R_s, x_b, z_b) = \frac{1}{8} \cdot \frac{4}{3}\pi R_s^3 - \frac{\pi}{4} \cdot \left[R_s^2 \cdot x_b - \frac{x_b^3}{3}\right] - \frac{\pi}{4} \cdot \left[R_s^2 \cdot z_b - \frac{z_b^3}{3}\right] + \tag{1.20}$$
$$\frac{z_b \cdot x_b \cdot \sqrt{R_s^2 - x_b^2 - z_b^2}}{3} + \frac{1}{2} \cdot \arctan\left(\frac{x_b}{\sqrt{R_s^2 - x_b^2 - z_b^2}}\right) \cdot \left[R_s^2 \cdot z_b - \frac{z_b^3}{3}\right] +$$
$$\frac{1}{2} \cdot \arctan\left(\frac{z_b}{\sqrt{R_s^2 - x_b^2 - z_b^2}}\right) \cdot \left[R_s^2 \cdot x_b - \frac{x_b^3}{3}\right] - \frac{R_s^3}{3} \cdot \arctan\left(\frac{z_b \cdot x_b}{R_s \cdot \sqrt{R_s^2 - x_b^2 - z_b^2}}\right).$$

Next, we factor out the $R_s^3/6$ terms in the square brackets to simplify the equation into

$$V_{\text{cut}}(R_s, x_b, z_b) = \frac{R_s^3}{6} \cdot \left[\pi - 2 \cdot \arctan\left(\frac{z_b \cdot x_b}{R_s \cdot \sqrt{R_s^2 - x_b^2 - z_b^2}}\right)\right] \tag{1.21}$$
$$+ \frac{1}{2} \cdot \left[\arctan\left(\frac{z_b}{\sqrt{R_s^2 - x_b^2 - z_b^2}}\right) - \frac{\pi}{2}\right] \cdot \left[R_s^2 \cdot x_b - \frac{x_b^3}{3}\right]$$
$$+ \frac{1}{2} \cdot \left[\arctan\left(\frac{x_b}{\sqrt{R_s^2 - x_b^2 - z_b^2}}\right) - \frac{\pi}{2}\right] \cdot \left[R_s^2 \cdot z_b - \frac{z_b^3}{3}\right]$$
$$+ \frac{z_b \cdot x_b \cdot \sqrt{R_s^2 - x_b^2 - z_b^2}}{3}.$$

This is the final analytic solution for the volume of the spherical cut, V_{cut}. In general, a boundary pair, (a, b), for which

$$a^2 + b^2 < R_s^2 \tag{1.22}$$

holds, creates the following spherical cut volume:

$$V_{\text{cut}}(R_s, a, b) = \frac{R_s^3}{6} \cdot \left[\pi - 2 \cdot \arctan\left(\frac{a \cdot b}{R_s \cdot \sqrt{R_s^2 - a^2 - b^2}}\right)\right] \tag{1.23}$$
$$+ \frac{1}{2} \cdot \left[\arctan\left(\frac{a}{\sqrt{R_s^2 - a^2 - b^2}}\right) - \frac{\pi}{2}\right] \cdot \left[R_s^2 \cdot b - \frac{b^3}{3}\right]$$
$$+ \frac{1}{2} \cdot \left[\arctan\left(\frac{b}{\sqrt{R_s^2 - a^2 - b^2}}\right) - \frac{\pi}{2}\right] \cdot \left[R_s^2 \cdot a - \frac{a^3}{3}\right]$$
$$+ \frac{a \cdot b \cdot \sqrt{R_s^2 - a^2 - b^2}}{3}.$$

2 Mathematical Appendix

2.1 The Volume of a Spherical Segment

In this section, we derive the volume of a spherical segment. A spherical segment of height, B, is shown in Figure S18.

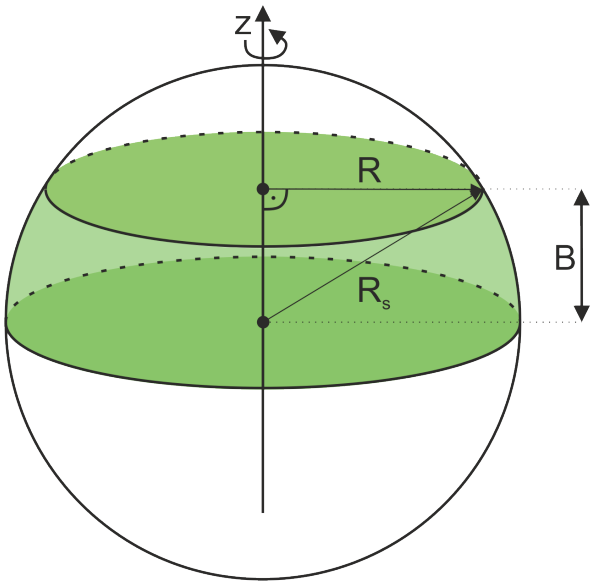


Figure S18: Spherical segment (green) of height, B, inside a sphere of radius, R_s.

A spherical segment can be viewed as a rotation body. The rotation axis is vertical (z) and the radius, R, depends on the height. Using Pythagoreans theorem, we see that

$$R^2 + z^2 = R_s^2 \quad \Leftrightarrow \quad R^2 = R_s^2 - z^2. \tag{2.1}$$

To get the volume of the spherical segment, V_{seg}, we integrate over tiny discs along the z-axis like

$$V_{\text{seg}}(R_s, B) = \int_0^B \pi \cdot R^2 \; dz = \int_0^B \pi \cdot \left[R_s^2 - z^2\right] \; dz. \tag{2.2}$$

Then, we split the integral at the minus sign and factor out the sphere radius, R_s, to get

$$\pi \cdot \left[\int_0^B R_s^2 \; dz - \int_0^B z^2 \; dz\right] = \pi \cdot \left[R_s^2 \cdot \int_0^B 1 \; dz - \int_0^B z^2 \; dz\right]. \tag{2.3}$$

Finally, we solve the integrals by

$$\pi \cdot \left[R_s^2 \cdot [z]_0^B - \left[\frac{1}{3} \cdot z^3\right]_0^B\right] = \pi \cdot \left[R_s^2 \cdot [B - 0] - \left[\frac{1}{3} \cdot B^3 - \frac{1}{3} \cdot 0^3\right]\right], \tag{2.4}$$

which leaves us with

$$\boxed{V_{\text{seg}}(R_s, B) = \pi \cdot \left[R_s^2 \cdot B - \frac{1}{3} \cdot B^3\right]}. \tag{2.5}$$

This is the final equation for the volume of a spherical segment as a function of the boundary position, B, and the sphere radius, R_s.

2.2 The Volume of a Spherical Cap

In this section, we derive the volume of a spherical cap. A spherical cap is shown in Figure S19. This volume is very similar to the volume of a spherical segment.

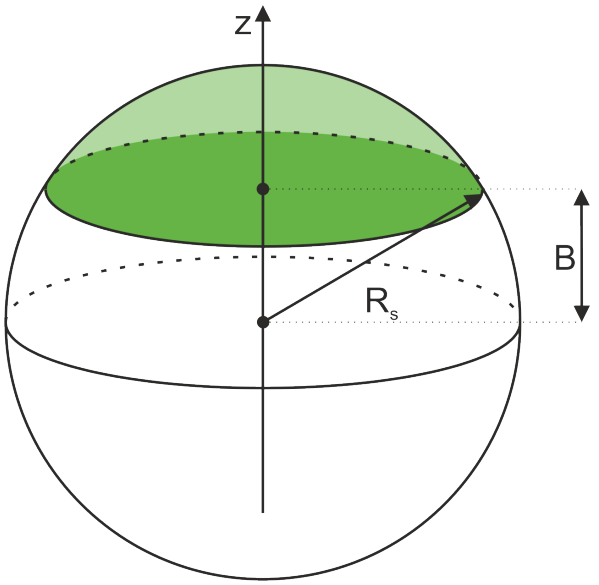


Figure S19: Spherical cap (green) with radius, R_s, and height, $R_s - B$.

First, we note that a spherical cap is just a hemi-sphere minus a spherical segment:

$$V_{\text{cap}}(R_s, B) = \frac{1}{2} \cdot \frac{4}{3}\pi R_s^3 - V_{\text{seg}}(R_s, B) \tag{2.6}$$

We insert the equation for a spherical segment, derived in appendix 2.1, and simplify the first term to get

$$V_{\text{cap}}(R_s, B) = \frac{2}{3}\pi R_s^3 - \pi \cdot \left[R_s^2 \cdot B - \frac{1}{3} \cdot B^3\right]. \tag{2.7}$$

Finally, we factor out π and obtain

$$\boxed{V_{\text{cap}}(R_s, B) = \pi \cdot \left[\frac{2}{3} \cdot R_s^3 - R_s^2 \cdot B + \frac{1}{3} \cdot B^3\right]}. \tag{2.8}$$

This is the final equation for the volume of a spherical cap as a function of the boundary position, B, from the sphere center.

2.3 The Internal Volume V_{int}

In this section, we derive the volume of the internal element, V_{int}. Figure S20 shows a rotated drawing of V_{int} inside a Cartesian coordinate system.

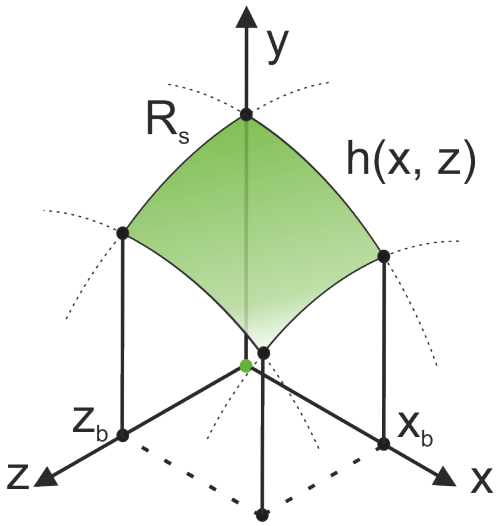


Figure S20: Integration in Cartesian coordinates yields the volume under a 3D surface. In our case, the 3D surface (green) is part of the surface of a sphere.

To obtain V_{int}, we integrate a section of the surface of a sphere. The section is bounded by perpendicular planes through x_b and z_b. These planes are also orthogonal to their respective coordinate axis. A sphere, with its center at the origin, is defined by

$$x^2 + y^2 + z^2 = R_s^2. \tag{2.9}$$

The height, $h(x, z) = y$, is therefore

$$h(x, z) = \sqrt{R_s^2 - z^2 - x^2}. \tag{2.10}$$

To obtain the volume under the green surface, we integrate along the x and z coordinate. The integration boundaries are from 0 to the values x_b and z_b respectively, like

$$V_{\text{int}}(R_s, x_b, z_b) = \int_0^{x_b}\int_0^{z_b} h(x, z)\; dz\; dx = \int_0^{x_b}\int_0^{z_b} \sqrt{R_s^2 - z^2 - x^2}\; dz\; dx. \tag{2.11}$$

We solve the two nested integrals one after another. First we integrate over the z-coordinate. Applying the anti-derivative, given in appendix 2.4, we get

$$V_{\text{int}} = \int_0^{x_b} \left[\frac{1}{2} \cdot \left[z \cdot \sqrt{R_s^2 - z^2 - x^2} + \left(R_s^2 - x^2\right) \cdot \arctan\left(\frac{z}{\sqrt{R_s^2 - z^2 - x^2}}\right)\right]\right]_0^{z_b} dx. \tag{2.12}$$

We insert the upper and lower bound, yielding

$$\begin{aligned} V_{\text{int}} = \int_0^{x_b} &\frac{1}{2} \cdot \left[z_b \cdot \sqrt{R_s^2 - z_b^2 - x^2} + \left(R_s^2 - x^2\right) \cdot \arctan\left(\frac{z_b}{\sqrt{R_s^2 - z_b^2 - x^2}}\right)\right] \\ &- \frac{1}{2} \cdot \left[0 \cdot \sqrt{R_s^2 - 0^2 - x^2} + \left(R_s^2 - x^2\right) \cdot \arctan\left(\frac{0}{\sqrt{R_s^2 - 0^2 - x^2}}\right)\right] dx. \end{aligned} \tag{2.13}$$

Since $\arctan(0) = 0$, we are left with

$$V_{\text{int}} = \int_0^{x_b} \frac{1}{2} \cdot \left[z_b \cdot \sqrt{R_s^2 - z_b^2 - x^2} + \left(R_s^2 - x^2 \right) \cdot \arctan\left(\frac{z_b}{\sqrt{R_s^2 - z_b^2 - x^2}} \right) \right] \, dx. \tag{2.14}$$

Next, we split the remaining integral into three parts. We also bring the factor of 2 onto the side of V_{int}. This gives us

$$\begin{aligned} 2 \cdot V_{\text{int}} = z_b \cdot & \int_0^{x_b} \sqrt{R_s^2 - z_b^2 - x^2} \, dx \\ + R_s^2 \cdot & \int_0^{x_b} \arctan\left(\frac{z_b}{\sqrt{R_s^2 - z_b^2 - x^2}} \right) \, dx \\ - & \int_0^{x_b} x^2 \cdot \arctan\left(\frac{z_b}{\sqrt{R_s^2 - z_b^2 - x^2}} \right) \, dx. \end{aligned} \tag{2.15}$$

We now have three individual integrals to solve. We label them according to

$$2 \cdot V_{\text{int}} = z_b \cdot I_1 + R_s^2 \cdot I_2 - I_3. \tag{2.16}$$

For simplicity, we define the terms

$$A = R_s^2 - z_b^2 \quad \text{and} \quad S = \sqrt{R_s^2 - z_b^2 - x_b^2} = \sqrt{A - x_b^2}, \tag{2.17}$$

that are always greater than 0. The anti-derivative of the first integral,

$$I_1 = \int_0^{x_b} \sqrt{R_s^2 - z_b^2 - x^2} \, dx = \int_0^{x_b} \sqrt{A - x^2} \, dx \tag{2.18}$$

is solved in appendix 2.4 and yields

$$I_1 = \left[\frac{1}{2} \cdot \left[x \cdot \sqrt{A - x^2} + A \cdot \arctan\left(\frac{x}{\sqrt{A - x^2}} \right) \right] \right]_0^{x_b}. \tag{2.19}$$

Inserting the upper and lower boundaries gives us

$$\begin{aligned} I_1 = & \frac{1}{2} \cdot \left[x_b \cdot \sqrt{A - x_b^2} + A \cdot \arctan\left(\frac{x_b}{\sqrt{A - x_b^2}} \right) \right] \\ & - \frac{1}{2} \cdot \left[0 \cdot \sqrt{A - 0^2} + A \cdot \arctan\left(\frac{0}{\sqrt{A - 0^2}} \right) \right]. \end{aligned} \tag{2.20}$$

Since $\arctan(0) = 0$, we are left with

$$\boxed{I_1 = \frac{1}{2} \cdot \left[x_b \cdot S + A \cdot \arctan\left(\frac{x_b}{S} \right) \right]}. \tag{2.21}$$

The second integral,

$$I_2 = \int_0^{x_b} \arctan\left(\frac{z_b}{\sqrt{R_s^2 - z_b^2 - x^2}} \right) \, dx = \int_0^{x_b} \arctan\left(\frac{z_b}{\sqrt{A - x^2}} \right) \, dx, \tag{2.22}$$

is solved in appendix 2.5 and yields

$$I_2 = \Bigg[z_b \cdot \arctan\left(\frac{x}{\sqrt{A-x^2}}\right) + x \cdot \arctan\left(\frac{z_b}{\sqrt{A-x^2}}\right) - R_s \cdot \arctan\left(\frac{x \cdot z_b}{R_s \cdot \sqrt{A-x^2}}\right) \Bigg]_0^{x_b}. \tag{2.23}$$

Inserting the upper and lower boundaries gives us

$$\begin{aligned} I_2 = {} & z_b \cdot \arctan\left(\frac{x_b}{\sqrt{A-x_b^2}}\right) \\ & + x_b \cdot \arctan\left(\frac{z_b}{\sqrt{A-x_b^2}}\right) \\ & - R_s \cdot \arctan\left(\frac{x_b \cdot z_b}{R_s \cdot \sqrt{A-x_b^2}}\right) \\ & - \Bigg[z_b \cdot \arctan\left(\frac{0}{\sqrt{A-0^2}}\right) \\ & + 0 \cdot \arctan\left(\frac{z_b}{\sqrt{A-0^2}}\right) \\ & - R_s \cdot \arctan\left(\frac{0 \cdot z_b}{R_s \cdot \sqrt{A-0^2}}\right) \Bigg] \end{aligned} \tag{2.24}$$

Since $\arctan(0) = 0$, we are left with

$$\boxed{I_2 = z_b \cdot \arctan\left(\frac{x_b}{S}\right) + x_b \cdot \arctan\left(\frac{z_b}{S}\right) - R_s \cdot \arctan\left(\frac{x_b \cdot z_b}{R_s \cdot S}\right)}. \tag{2.25}$$

The third integral,

$$I_3 = \int_0^{x_b} x^2 \cdot \arctan\left(\frac{z_b}{\sqrt{R_s^2 - z_b^2 - x^2}}\right)\, dx = \int_0^{x_b} x^2 \cdot \arctan\left(\frac{z_b}{\sqrt{A-x^2}}\right)\, dx, \tag{2.26}$$

is solved in the appendix 2.6, like

$$\begin{aligned} & \Bigg[-\frac{z_b}{6} \cdot x \cdot \sqrt{A-x^2} - \frac{z_b}{6} \cdot \left(z_b^2 - 3 \cdot R_s^2\right) \cdot \arctan\left(\frac{x}{\sqrt{A-x^2}}\right) \\ & + \frac{x^3}{3} \cdot \arctan\left(\frac{z_b}{\sqrt{A-x^2}}\right) - \frac{R_s^3}{3} \cdot \arctan\left(\frac{z_b \cdot x}{R_s \cdot \sqrt{A-x^2}}\right) \Bigg]_0^{x_b}. \end{aligned} \tag{2.27}$$

Inserting the upper and lower boundaries gives us

$$
\begin{aligned}
&-\frac{z_b}{6}\cdot x_b\cdot\sqrt{A-x_b^2}-\frac{z_b}{6}\cdot\left(z_b^2-3\cdot R_s^2\right)\cdot\arctan\left(\frac{x_b}{\sqrt{A-x_b^2}}\right)\\
&\quad+\frac{x_b^3}{3}\cdot\arctan\left(\frac{z_b}{\sqrt{A-x_b^2}}\right)-\frac{R_s^3}{3}\cdot\arctan\left(\frac{z_b\cdot x_b}{R_s\cdot\sqrt{A-x_b^2}}\right)\\
&-\Bigg[-\frac{z_b}{6}\cdot 0\cdot\sqrt{A-0^2}-\frac{z_b}{6}\cdot\left(z_b^2-3\cdot R_s^2\right)\cdot\arctan\left(\frac{0}{\sqrt{A-0^2}}\right)\\
&\quad+\frac{0^3}{3}\cdot\arctan\left(\frac{z_b}{\sqrt{A-0^2}}\right)-\frac{R_s^3}{3}\cdot\arctan\left(\frac{z_b\cdot 0}{R_s\cdot\sqrt{A-0^2}}\right)\Bigg].
\end{aligned}
\tag{2.28}
$$

Since $\arctan(0)=0$, we are left with

$$
\boxed{I_3=\frac{-x_bz_bS}{6}-\frac{z_b}{6}\left(z_b^2-3R_s^2\right)\arctan\left(\frac{x_b}{S}\right)+\frac{x_b^3}{3}\arctan\left(\frac{z_b}{S}\right)-\frac{R_s^3}{3}\arctan\left(\frac{z_bx_b}{R_sS}\right)}.
\tag{2.29}
$$

Now, we combine the solutions for the three integrals and get:

$$
\begin{aligned}
2\cdot V_{\text{int}}&=z_b\cdot\left[\frac{1}{2}\cdot\left[x_b\cdot S+\left(R_s^2-z_b^2\right)\cdot\arctan\left(\frac{x_b}{S}\right)\right]\right]\\
&+R_s^2\cdot\left[z_b\cdot\arctan\left(\frac{x_b}{S}\right)+x_b\cdot\arctan\left(\frac{z_b}{S}\right)-R_s\cdot\arctan\left(\frac{z_b\cdot x_b}{R_s\cdot S}\right)\right]\\
&-\left[\frac{-x_b\cdot z_b\cdot S}{6}-\frac{z_b}{6}\cdot\left(z_b^2-3R_s^2\right)\cdot\arctan\left(\frac{x_b}{S}\right)+\frac{x_b^3}{3}\cdot\arctan\left(\frac{z_b}{S}\right)-\frac{R_s^3}{3}\cdot\arctan\left(\frac{z_b\cdot x_b}{R_s\cdot S}\right)\right].
\end{aligned}
\tag{2.30}
$$

Next, we expand all products to remove the brackets

$$
\begin{aligned}
2\cdot V_{\text{int}}&=\frac{z_b\cdot x_b\cdot S}{2}+\frac{z_b\cdot\left(R_s^2-z_b^2\right)}{2}\cdot\arctan\left(\frac{x_b}{S}\right)\\
&+R_s^2\cdot z_b\cdot\arctan\left(\frac{x_b}{S}\right)+R_s^2\cdot x_b\cdot\arctan\left(\frac{z_b}{S}\right)-R_s^3\cdot\arctan\left(\frac{z_b\cdot x_b}{R_s\cdot S}\right)\\
&+\frac{x_b\cdot z_b\cdot S}{6}+\frac{z_b}{6}\cdot\left(z_b^2-3R_s^2\right)\cdot\arctan\left(\frac{x_b}{S}\right)-\frac{x_b^3}{3}\cdot\arctan\left(\frac{z_b}{S}\right)\\
&+\frac{R_s^3}{3}\cdot\arctan\left(\frac{z_b\cdot x_b}{R_s\cdot S}\right).
\end{aligned}
\tag{2.31}
$$

We then factor out the arcus tangens functions

$$
\begin{aligned}
2\cdot V_{\text{int}}&=\frac{z_b\cdot x_b\cdot S}{2}+\frac{x_b\cdot z_b\cdot S}{6}\\
&+\arctan\left(\frac{x_b}{S}\right)\cdot\left[\frac{z_b\cdot\left(R_s^2-z_b^2\right)}{2}+R_s^2\cdot z_b+\frac{z_b}{6}\cdot\left(z_b^2-3R_s^2\right)\right]\\
&+\arctan\left(\frac{z_b}{S}\right)\cdot\left[R_s^2\cdot x_b-\frac{x_b^3}{3}\right]+\arctan\left(\frac{z_b\cdot x_b}{R_s\cdot S}\right)\cdot\left[\frac{R_s^3}{3}-R_s^3\right].
\end{aligned}
\tag{2.32}
$$

Next, we take a closer look at the term in the square brackets containing z_b and see that

$$\begin{aligned} \frac{z_b \cdot (R_s^2 - z_b^2)}{2} + R_s^2 \cdot z_b + \frac{z_b}{6} \cdot \left(z_b^2 - 3R_s^2\right) = \\ \frac{z_b \cdot R_s^2}{2} - \frac{z_b^3}{2} + R_s^2 \cdot z_b + \frac{z_b^3}{6} - \frac{z_b \cdot R_s^2}{2} = \\ R_s^2 \cdot z_b - \frac{z_b^3}{3} \end{aligned} \tag{2.33}$$

Using equation (2.33), we simplify the terms into

$$\begin{aligned} 2 \cdot V_{\text{int}} = \frac{2 \cdot z_b \cdot x_b \cdot S}{3} + \arctan\left(\frac{x_b}{S}\right) \cdot \left[R_s^2 \cdot z_b - \frac{z_b^3}{3}\right] \\ + \arctan\left(\frac{z_b}{S}\right) \cdot \left[R_s^2 \cdot x_b - \frac{x_b^3}{3}\right] + \arctan\left(\frac{z_b \cdot x_b}{R_s \cdot S}\right) \cdot R_s^3 \cdot \left[-\frac{2}{3}\right]. \end{aligned} \tag{2.34}$$

Finally, we bring the factor of 2 on the other side

$$\begin{aligned} V_{\text{int}} = \frac{z_b \cdot x_b \cdot S}{3} + \frac{1}{2} \cdot \arctan\left(\frac{x_b}{S}\right) \cdot \left[R_s^2 \cdot z_b - \frac{z_b^3}{3}\right] \\ + \frac{1}{2} \cdot \arctan\left(\frac{z_b}{S}\right) \cdot \left[R_s^2 \cdot x_b - \frac{x_b^3}{3}\right] - \frac{R_s^3}{3} \cdot \arctan\left(\frac{z_b \cdot x_b}{R_s \cdot S}\right), \end{aligned} \tag{2.35}$$

and substitute $S = \sqrt{R_s^2 - x_b^2 - z_b^2}$ back into the equation

$$\begin{aligned} V_{\text{int}} = \frac{z_b \cdot x_b \cdot \sqrt{R_s^2 - x_b^2 - z_b^2}}{3} + \frac{1}{2} \cdot \arctan\left(\frac{x_b}{\sqrt{R_s^2 - x_b^2 - z_b^2}}\right) \cdot \left[R_s^2 \cdot z_b - \frac{z_b^3}{3}\right] \\ + \frac{1}{2} \cdot \arctan\left(\frac{z_b}{\sqrt{R_s^2 - x_b^2 - z_b^2}}\right) \cdot \left[R_s^2 \cdot x_b - \frac{x_b^3}{3}\right] - \frac{R_s^3}{3} \cdot \arctan\left(\frac{z_b \cdot x_b}{R_s \cdot \sqrt{R_s^2 - x_b^2 - z_b^2}}\right). \end{aligned} \tag{2.36}$$

This is the final equation for the internal volume, V_{int}, as a function of the octant radius, R_s, and the boundary positions, x_b and z_b.

2.4 Proof for the Integral I_1

In this section, we proof that the integral

$$I_1 = \int \sqrt{A - x^2}\, dx, \quad (A - x^2) > 0 \tag{2.37}$$

is solved by

$$\frac{1}{2} \cdot \left[x \cdot \sqrt{A - x^2} + A \cdot \arctan\left(\frac{x}{\sqrt{A - x^2}} \right) \right] + \mathcal{C}. \tag{2.38}$$

To do so, we take the derivative of equation (2.38) with respect to x. Note that

$$\frac{d}{dx} \arctan(x) = \frac{1}{1 + x^2}. \tag{2.39}$$

The derivative is then given by

$$\begin{aligned} &\frac{d}{dx} \frac{1}{2} \cdot \left[x \cdot \sqrt{A - x^2} + A \cdot \arctan\left(\frac{x}{\sqrt{A - x^2}} \right) \right] + \mathcal{C} = \\ &\quad \frac{1}{2} \cdot \left[1 \cdot \sqrt{A - x^2} + x \cdot \frac{1}{2\sqrt{A - x^2}} \cdot (-2x) + A \cdot \frac{1}{1 + \frac{x^2}{A - x^2}} \cdot \frac{\sqrt{A - x^2} - x \cdot \frac{1}{2\sqrt{A - x^2}} \cdot (-2x)}{A - x^2} \right]. \end{aligned} \tag{2.40}$$

We combine all fractions to get

$$\frac{1}{2} \cdot \left[\sqrt{A - x^2} - \frac{x^2}{\sqrt{A - x^2}} + \frac{A}{A - x^2 + x^2} \cdot \left[\sqrt{A - x^2} + \frac{x^2}{\sqrt{A - x^2}} \right] \right]. \tag{2.41}$$

Finally, we simplify the fractions and see that

$$\frac{1}{2} \cdot \left[\sqrt{A - x^2} - \frac{x^2}{\sqrt{A - x^2}} + \sqrt{A - x^2} + \frac{x^2}{\sqrt{A - x^2}} \right] = \sqrt{A - x^2}, \tag{2.42}$$

which is the integrand we were looking for. Therefore, the integral, I_1, is solved by the anti-derivative given in equation (2.38).

2.5 Proof for the Integral I_2

In this section, we prove that the integral

$$I_2 = \int \arctan\left(\frac{z_b}{\sqrt{A-x^2}}\right)\, dx, \quad (A-x^2) > 0 \tag{2.43}$$

is solved by the following anti-derivative

$$z_b \cdot \arctan\left(\frac{x}{\sqrt{A-x^2}}\right) + x \cdot \arctan\left(\frac{z_b}{\sqrt{A-x^2}}\right) - R_s \cdot \arctan\left(\frac{x \cdot z_b}{R_s \cdot \sqrt{A-x^2}}\right) + \mathcal{C}. \tag{2.44}$$

To prove that, we calculate the derivative with respect to x like

$$\begin{aligned} &\frac{d}{dx}\ z_b \cdot \arctan\left(\frac{x}{\sqrt{A-x^2}}\right) \\ &+\frac{d}{dx}\ x \cdot \arctan\left(\frac{z_b}{\sqrt{A-x^2}}\right) \\ &-\frac{d}{dx}\ R_s \cdot \arctan\left(\frac{x \cdot z_b}{R_s \cdot \sqrt{A-x^2}}\right) + \mathcal{C} \end{aligned} \tag{2.45}$$

We calculate the three derivatives individually. The first derivative is given by

$$\frac{d}{dx}\ z_b \cdot \arctan\left(\frac{x}{\sqrt{A-x^2}}\right) = z_b \cdot \frac{1}{1+\frac{x^2}{A-x^2}} \cdot \frac{1 \cdot \sqrt{A-x^2} - x \cdot \frac{1}{2 \cdot \sqrt{A-x^2}} \cdot (-2x)}{A-x^2} \tag{2.46}$$

we simplify the fractions and get

$$\frac{z_b}{A-x^2+x^2} \cdot \left[\sqrt{A-x^2} + \frac{x^2}{\sqrt{A-x^2}}\right]. \tag{2.47}$$

The x^2 cancel each other out, leaving us with

$$\frac{z_b}{A} \cdot \left[\sqrt{A-x^2} + \frac{x^2}{\sqrt{A-x^2}}\right]. \tag{2.48}$$

Next, we create equal denominators in the square brackets

$$\frac{z_b}{A} \cdot \left[\frac{A-x^2}{\sqrt{A-x^2}} + \frac{x^2}{\sqrt{A-x^2}}\right], \tag{2.49}$$

which leaves us with

$$\boxed{\frac{d}{dx}\ z_b \cdot \arctan\left(\frac{x}{\sqrt{A-x^2}}\right) = \frac{z_b}{\sqrt{A-x^2}}}. \tag{2.50}$$

The second derivative is

$$\begin{aligned} &\frac{d}{dx}\ x \cdot \arctan\left(\frac{z_b}{\sqrt{A-x^2}}\right) = \\ &1 \cdot \arctan\left(\frac{z_b}{\sqrt{A-x^2}}\right) + x \cdot \frac{1}{1+\frac{z_b^2}{A-x^2}} \cdot \frac{\sqrt{A-x^2} \cdot 0 - z_b \cdot \frac{1}{2\sqrt{A-x^2}} \cdot (-2x)}{A-x^2}. \end{aligned} \tag{2.51}$$

We combine the main fractions and get

$$\arctan\left(\frac{z_b}{\sqrt{A-x^2}}\right)+\frac{x}{A-x^2+z_b^2}\cdot\frac{z_b\cdot x}{\sqrt{A-x^2}}. \tag{2.52}$$

Using our definition $A=R_s^2-z_b^2$ we have

$$\boxed{\frac{d}{dx}\ x\cdot\arctan\left(\frac{z_b}{\sqrt{A-x^2}}\right)=\arctan\left(\frac{z_b}{\sqrt{A-x^2}}\right)+\frac{z_b\cdot x^2}{(R_s^2-x^2)\cdot\sqrt{A-x^2}}}. \tag{2.53}$$

The third derivative is given by

$$\begin{aligned}&\frac{d}{dx}\ R_s\cdot\arctan\left(\frac{z_b\cdot x}{R_s\cdot\sqrt{A-x^2}}\right)= \\ &R_s\cdot\frac{1}{1+\frac{z_b^2\cdot x^2}{R_s^2\cdot(A-x^2)}}\cdot\frac{R_s\cdot\sqrt{A-x^2}\cdot z_b-z_b\cdot x\cdot R_s\cdot\frac{1}{2\sqrt{A-x^2}}\cdot(-2x)}{R_s^2\cdot(A-x^2)}.\end{aligned} \tag{2.54}$$

We combine the main fraction while factoring out $z_b\cdot R_s$ from the numerator, to get

$$\frac{R_s^2\cdot z_b}{R_s^2\cdot(A-x^2)+z_b^2\cdot x^2}\cdot\left[\sqrt{A-x^2}+\frac{x^2}{\sqrt{A-x^2}}\right] \tag{2.55}$$

Next, we create identical denominators in the square brackets

$$\frac{R_s^2\cdot z_b}{R_s^2\cdot(A-x^2)+z_b^2\cdot x^2}\cdot\left[\frac{A-x^2}{\sqrt{A-x^2}}+\frac{x^2}{\sqrt{A-x^2}}\right], \tag{2.56}$$

which leaves us with

$$\frac{R_s^2\cdot z_b}{R_s^2\cdot(A-x^2)+z_b^2\cdot x^2}\cdot\frac{A}{\sqrt{A-x^2}}, \tag{2.57}$$

Before we proceed, we take a closer look at the largest denominator and see that

$$\begin{aligned}R_s^2\cdot\left(A-x^2\right)+z_b^2\cdot x^2= \\ R_s^2\cdot A-R_s^2\cdot x^2+z_b^2\cdot x^2= \\ R_s^2\cdot A+x^2\cdot\left(z_b^2-R_s^2\right).\end{aligned} \tag{2.58}$$

Since

$$A=R_s^2-z_b^2\Leftrightarrow -A=(z_b^2-R_s^2), \tag{2.59}$$

we have

$$R_s^2\cdot A-A\cdot x^2=A\cdot\left(R_s^2-x^2\right). \tag{2.60}$$

Using this identity in equation (2.57), we get

$$\frac{R_s^2\cdot z_b}{A\cdot(R_s^2-x^2)}\cdot\frac{A}{\sqrt{A-x^2}}, \tag{2.61}$$

which leaves us with

$$\boxed{\frac{d}{dx}\ R_s\cdot\arctan\left(\frac{z_b\cdot x}{R_s\cdot\sqrt{A-x^2}}\right)=\frac{R_s^2\cdot z_b}{(R_s^2-x^2)\cdot\sqrt{A-x^2}}}. \tag{2.62}$$

Finally, we combine all the separate derivatives like

$$\frac{z_b}{\sqrt{A-x^2}} + \arctan\left(\frac{z_b}{\sqrt{A-x^2}}\right) + \frac{z_b \cdot x^2}{(R_s^2-x^2)\cdot\sqrt{A-x^2}} - \frac{R_s^2 \cdot z_b}{(R_s^2-x^2)\cdot\sqrt{A-x^2}}. \quad (2.63)$$

We expand the first fraction to create equal denominators like

$$\frac{(R_s^2-x^2)\cdot z_b}{(R_s^2-x^2)\cdot\sqrt{A-x^2}} + \arctan\left(\frac{z_b}{\sqrt{A-x^2}}\right) + \frac{z_b \cdot x^2}{(R_s^2-x^2)\cdot\sqrt{A-x^2}} - \frac{R_s^2 \cdot z_b}{(R_s^2-x^2)\cdot\sqrt{A-x^2}}. \quad (2.64)$$

Then, we add all fractions

$$\frac{z_b\cdot(R_s^2-x^2) + z_b\cdot x^2 - R_s^2\cdot z_b}{(R_s^2-x^2)\cdot\sqrt{A-x^2}} + \arctan\left(\frac{z_b}{\sqrt{A-x^2}}\right), \quad (2.65)$$

and expand the round brackets into

$$\frac{z_b\cdot R_s^2 - z_b\cdot x^2 + z_b\cdot x^2 - R_s^2\cdot z_b}{(R_s^2-x^2)\cdot\sqrt{A-x^2}} + \arctan\left(\frac{z_b}{\sqrt{A-x^2}}\right). \quad (2.66)$$

We see, that all terms in the numerator cancel each other out, leaving us with

$$\arctan\left(\frac{z_b}{\sqrt{A-x^2}}\right), \quad (2.67)$$

which is the integrand we were looking for. Therefore, the integral, I_2, is solved by the anti-derivative given in equation (2.44).

2.6 Proof for the Integral I_3

In this section, we proof that the integral

$$I_3 = \int x^2 \cdot \arctan\left(\frac{z_b}{\sqrt{A-x^2}}\right)\, dx, \quad (A-x^2) > 0 \quad (2.68)$$

is solved by

$$\begin{aligned} &-\frac{z_b}{6}\cdot x\cdot\sqrt{A-x^2} - \frac{z_b}{6}\cdot\left(z_b^2 - 3\cdot R_s^2\right)\cdot\arctan\left(\frac{x}{\sqrt{A-x^2}}\right) \\ &+\frac{x^3}{3}\cdot\arctan\left(\frac{z_b}{\sqrt{A-x^2}}\right) - \frac{R_s^3}{3}\cdot\arctan\left(\frac{z_b\cdot x}{R_s\cdot\sqrt{A-x^2}}\right) + \mathcal{C}. \end{aligned} \quad (2.69)$$

To do so, we take the derivative of this solution with respect to x like

$$\begin{aligned} &-\frac{z_b}{6}\cdot\frac{d}{dx}x\cdot\sqrt{A-x^2} \\ &-\frac{z_b}{6}\cdot\left(z_b^2 - 3\cdot R_s^2\right)\cdot\frac{d}{dx}\arctan\left(\frac{x}{\sqrt{A-x^2}}\right) \\ &+\frac{d}{dx}\frac{x^3}{3}\cdot\arctan\left(\frac{z_b}{\sqrt{A-x^2}}\right) \\ &-\frac{R_s^3}{3}\cdot\frac{d}{dx}\arctan\left(\frac{z_b\cdot x}{R_s\cdot\sqrt{A-x^2}}\right). \end{aligned} \quad (2.70)$$

We calculate the derivatives separately. The first derivative is given by

$$-\frac{z_b}{6}\cdot\frac{d}{dx}x\cdot\sqrt{A-x^2}=-\frac{z_b}{6}\cdot\left[1\cdot\sqrt{A-x^2}+x\cdot\frac{1}{2\sqrt{A-x^2}}\cdot(-2x)\right]. \tag{2.71}$$

We simplify the terms in the square brackets and get

$$\frac{z_b}{6}\cdot\left[\frac{x^2}{\sqrt{A-x^2}}-\sqrt{A-x^2}\right]. \tag{2.72}$$

Then, we expand the second term in the square bracket

$$\frac{z_b}{6}\cdot\left[\frac{x^2}{\sqrt{A-x^2}}-\frac{A-x^2}{\sqrt{A-x^2}}\right], \tag{2.73}$$

and simplify, which leaves us with

$$\boxed{-\frac{z_b}{6}\cdot\frac{d}{dx}x\cdot\sqrt{A-x^2}=\frac{z_b}{6}\cdot\frac{2\cdot x^2-A}{\sqrt{A-x^2}}}. \tag{2.74}$$

The second derivative is given by

$$-\frac{z_b}{6}\cdot\left(z_b^2-3\cdot R_s^2\right)\cdot\frac{d}{dx}\arctan\left(\frac{x}{\sqrt{A-x^2}}\right)= \tag{2.75}$$
$$-\frac{z_b}{6}\cdot\left(z_b^2-3\cdot R_s^2\right)\cdot\frac{1}{1+\frac{x^2}{A-x^2}}\cdot\frac{1\cdot\sqrt{A-x^2}-x\cdot\frac{1}{2\sqrt{A-x^2}}\cdot(-2x)}{A-x^2}.$$

We multiply the fractions

$$-\frac{z_b}{6}\cdot\left(z_b^2-3\cdot R_s^2\right)\cdot\frac{1}{A-x^2+x^2}\left[\sqrt{A-x^2}+\frac{x^2}{\sqrt{A-x^2}}\right], \tag{2.76}$$

and simplify to get

$$-\frac{z_b\cdot(z_b^2-3\cdot R_s^2)}{6\cdot A}\cdot\left[\sqrt{A-x^2}+\frac{x^2}{\sqrt{A-x^2}}\right]. \tag{2.77}$$

Then, we expand the terms in the square brackets

$$-\frac{z_b\cdot(z_b^2-3\cdot R_s^2)}{6\cdot A}\cdot\left[\frac{A-x^2}{\sqrt{A-x^2}}+\frac{x^2}{\sqrt{A-x^2}}\right], \tag{2.78}$$

which leaves us with

$$\boxed{-\frac{z_b}{6}\cdot\left(z_b^2-3\cdot R_s^2\right)\cdot\frac{d}{dx}\arctan\left(\frac{x}{\sqrt{A-x^2}}\right)=-\frac{z_b\cdot(z_b^2-3\cdot R_s^2)}{6\cdot\sqrt{A-x^2}}}. \tag{2.79}$$

The third derivative is given by

$$\frac{d}{dx}\frac{x^3}{3}\cdot\arctan\left(\frac{z_b}{\sqrt{A-x^2}}\right)= \tag{2.80}$$
$$x^2\cdot\arctan\left(\frac{z_b}{\sqrt{A-x^2}}\right)+\frac{x^3}{3}\cdot\frac{1}{1+\frac{z_b^2}{A-x^2}}\cdot\frac{\sqrt{A-x^2}\cdot 0-z_b\cdot\frac{1}{2\sqrt{A-x^2}}\cdot(-2x)}{A-x^2}.$$

We combine the fractions

$$x^2 \cdot \arctan\left(\frac{z_b}{\sqrt{A-x^2}}\right) + \frac{x^3}{3} \cdot \frac{1}{A-x^2+z_b^2} \cdot \frac{x \cdot z_b}{\sqrt{A-x^2}}, \tag{2.81}$$

and simplify the denominator, using $A = R_s^2 - z_b^2$, yielding

$$\boxed{\frac{d}{dx}\frac{x^3}{3} \cdot \arctan\left(\frac{z_b}{\sqrt{A-x^2}}\right) = x^2 \cdot \arctan\left(\frac{z_b}{\sqrt{A-x^2}}\right) + \frac{x^4 \cdot z_b}{3 \cdot (R_s^2 - x^2) \cdot \sqrt{A-x^2}}}. \tag{2.82}$$

The last derivative is given by

$$-\frac{R_s^3}{3} \cdot \frac{d}{dx} \arctan\left(\frac{z_b \cdot x}{R_s \cdot \sqrt{A-x^2}}\right) = \tag{2.83}$$
$$-\frac{R_s^3}{3} \cdot \frac{1}{1+\frac{z_b^2 \cdot x^2}{R_s^2 \cdot (A-x^2)}} \cdot \frac{R_s \cdot \sqrt{A-x^2} \cdot z_b - z_b \cdot x \cdot R_s \cdot \frac{1}{2\sqrt{A-x^2}} \cdot (-2x)}{R_s^2 \cdot (A-x^2)}.$$

We combine the fractions, leaving us with

$$-\frac{z_b \cdot R_s^4}{3} \cdot \frac{1}{R_s^2 \cdot (A-x^2) + z_b^2 \cdot x^2} \cdot \left[\sqrt{A-x^2} + \frac{x^2}{\sqrt{A-x^2}}\right]. \tag{2.84}$$

Before we proceed, we take a closer look at the largest denominator and see that

$$\begin{aligned} R_s^2 \cdot \left(A - x^2\right) + z_b^2 \cdot x^2 = \\ R_s^2 \cdot A - R_s^2 \cdot x^2 + z_b^2 \cdot x^2 = \\ R_s^2 \cdot A + x^2 \cdot \left(z_b^2 - R_s^2\right). \end{aligned} \tag{2.85}$$

Since $A = R_s^2 - z_b^2 \Leftrightarrow -A = z_b^2 - R_s^2$ we get

$$R_s^2 \cdot A - A \cdot x^2 = A \cdot \left(R_s^2 - x^2\right) \tag{2.86}$$

Using this identity, we combine the fractions to get

$$-\frac{R_s^4 \cdot z_b}{3 \cdot A \cdot (R_s^2 - x^2)} \cdot \left[\sqrt{A-x^2} + \frac{x^2}{\sqrt{A-x^2}}\right]. \tag{2.87}$$

Next, we create equal denominator in the square brackets

$$-\frac{R_s^4 \cdot z_b}{3 \cdot A \cdot (R_s^2 - x^2)} \cdot \left[\frac{A-x^2}{\sqrt{A-x^2}} + \frac{x^2}{\sqrt{A-x^2}}\right], \tag{2.88}$$

which leaves us with

$$\boxed{-\frac{R_s^3}{3} \cdot \frac{d}{dx} \arctan\left(\frac{z_b \cdot x}{R_s \cdot \sqrt{A-x^2}}\right) = -\frac{R_s^4 \cdot z_b}{3 \cdot (R_s^2 - x^2) \cdot \sqrt{A-x^2}}}. \tag{2.89}$$

Finally, we sum up all the individual derivatives and get

$$\begin{aligned} \frac{z_b}{6} \cdot \frac{2 \cdot x^2 - A}{\sqrt{A-x^2}} - \frac{z_b \cdot (z_b^2 - 3 \cdot R_s^2)}{6 \cdot \sqrt{A-x^2}} + x^2 \cdot \arctan\left(\frac{z_b}{\sqrt{A-x^2}}\right) \\ + \frac{x^4 \cdot z_b}{3 \cdot (R_s^2 - x^2) \cdot \sqrt{A-x^2}} - \frac{R_s^4 \cdot z_b}{3 \cdot (R_s^2 - x^2) \cdot \sqrt{A-x^2}}. \end{aligned} \tag{2.90}$$

We add the fractions like

$$\frac{2 \cdot x^2 \cdot z_b - A \cdot z_b - z_b^3 + 3 \cdot R_s^2 \cdot z_b}{6 \cdot \sqrt{A - x^2}} + x^2 \cdot \arctan\left(\frac{z_b}{\sqrt{A - x^2}}\right) + \frac{z_b \cdot (x^4 - R_s^4)}{3 \cdot (R_s^2 - x^2) \cdot \sqrt{A - x^2}}. \tag{2.91}$$

We simplify the second fraction with the third binomic equation:

$$(a + b) \cdot (a - b) = a^2 - b^2. \tag{2.92}$$

This leaves us with

$$\frac{2 \cdot x^2 \cdot z_b - A \cdot z_b - z_b^3 + 3 \cdot R_s^2 \cdot z_b}{6 \cdot \sqrt{A - x^2}} + x^2 \cdot \arctan\left(\frac{z_b}{\sqrt{A - x^2}}\right) + \frac{z_b \cdot (x^2 - R_s^2) \cdot (x^2 + R_s^2)}{3 \cdot (R_s^2 - x^2) \cdot \sqrt{A - x^2}} \tag{2.93}$$

We now cancel the $R_s^2 - x^2 = -(x^2 - R_s^2)$ terms and are left with

$$\frac{2 \cdot x^2 \cdot z_b - A \cdot z_b - z_b^3 + 3 \cdot R_s^2 \cdot z_b}{6 \cdot \sqrt{A - x^2}} + x^2 \cdot \arctan\left(\frac{z_b}{\sqrt{A - x^2}}\right) - \frac{2 \cdot z_b \cdot (x^2 + R_s^2)}{6 \cdot \sqrt{A - x^2}}. \tag{2.94}$$

Again, we add the fractions with identical denominators. We also use $A = R_s^2 - z_b^2$ in the numerator, giving us

$$\frac{2 \cdot x^2 \cdot z_b - (R_s^2 - z_b^2) \cdot z_b - z_b^3 + 3 \cdot R_s^2 \cdot z_b - 2 \cdot z_b \cdot (x^2 + R_s^2)}{6 \cdot \sqrt{A - x^2}} + x^2 \cdot \arctan\left(\frac{z_b}{\sqrt{A - x^2}}\right) \tag{2.95}$$

We expand the terms in the round brackets

$$\frac{2 \cdot x^2 \cdot z_b - R_s^2 \cdot z_b + z_b^3 - z_b^3 + 3 \cdot R_s^2 \cdot z_b - 2 \cdot z_b \cdot x^2 - 2 \cdot z_b \cdot R_s^2}{6 \cdot \sqrt{A - x^2}} + x^2 \cdot \arctan\left(\frac{z_b}{\sqrt{A - x^2}}\right). \tag{2.96}$$

All terms in the numerator cancel each other out, which leaves us with

$$\boxed{x^2 \cdot \arctan\left(\frac{z_b}{\sqrt{A - x^2}}\right)}. \tag{2.97}$$

Therefore, the integral, I_3, is solved by the anti-derivative given in equation (2.69).

3 Pseudo code

Algorithm 1 Overlap volume between an octant and a corner

Ensure: $R, x_b, y_b, z_b \geq 0$ ▷ Mirror into the all positive octant

procedure OctantVolume(R, x_b, y_b, z_b)
  if $x_b^2 + y_b^2 + z_b^2 \leq R^2$ **then**
    return $x_b \cdot y_b \cdot z_b$ ▷ The corner is inside the octant
  end if
  $V_{Octant} \leftarrow \frac{1}{8} \cdot \frac{4}{3} \cdot \pi \cdot R^3$ ▷ Start with a complete octant
  for $b \in [x_b, y_b, z_b]$ **do**
    if $b < R$ **then** ▷ Remove the spherical caps
      $V_{Octant} \leftarrow V_{Octant} - \frac{\pi}{12} \cdot (2 \cdot R^3 - 3 \cdot b \cdot R^2 + b^3)$
    end if
  end for
  for $(a, b) \in [(x_b, y_b), (x_b, z_b), (y_b, z_b)]$ **do**
    if $a^2 + b^2 < R^2$ **then** ▷ Add the cap intersections
      $V_{Octant} \leftarrow V_{Octant} + \text{SphereCutVol}(R, a, b)$
    end if
  end for
  return V_{Octant}
end procedure

Algorithm 2 Compute the volume of a spherical cut.

Ensure: $R, A, B \geq 0,\ A^2 + B^2 < R^2$

procedure SphereCutVol(R, A, B)
  $Root \leftarrow \sqrt{R^2 - A^2 - B^2}$
  $V_{cut} \leftarrow \frac{R^3}{6} \cdot \left(\pi - 2 \cdot \arctan\left(\frac{A \cdot B}{R \cdot Root}\right)\right)$
  $V_{cut} \leftarrow V_{cut} + \frac{1}{2} \cdot \left(\arctan\left(\frac{A}{Root}\right) - \frac{\pi}{2}\right) \cdot \left(R^2 \cdot B - \frac{B^3}{3}\right)$
  $V_{cut} \leftarrow V_{cut} + \frac{1}{2} \cdot \left(\arctan\left(\frac{B}{Root}\right) - \frac{\pi}{2}\right) \cdot \left(R^2 \cdot A - \frac{A^3}{3}\right)$
  $V_{cut} \leftarrow V_{cut} + \frac{1}{3} \cdot A \cdot B \cdot Root$
  return V_{cut}
end procedure

Algorithm 3 Compute the shell-box intersection volume.

Ensure: $R_{min} \geq 0,\ R_{max} > 0,\ R_{min} < R_{max},\ x_{min} < x_{max},\ y_{min} < y_{max},\ z_{min} < z_{max}$

procedure ShellVolume($R_{min}, R_{max}, x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max}$)
  $InnerShell \leftarrow \text{SphereVolume}(R_{min}, x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max})$
  $OuterShell \leftarrow \text{SphereVolume}(R_{max}, x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max})$
  return $OuterShell - InnerShell$
end procedure

Algorithm 4 Compute the sphere-box intersection volume.

Ensure: $R \geq 0,\ x_{min} < x_{max},\ y_{min} < y_{max},\ z_{min} < z_{max}$

procedure SphereVolume($R, Boundaries$)
  $V_{Sphere} \leftarrow 0$ ▷ Start with an empty volume
  for $x_b \in [|x_{min}|, |x_{max}|]$ **do**
    for $y_b \in [|y_{min}|, |y_{max}|]$ **do**
      for $z_b \in [|z_{min}|, |z_{max}|]$ **do**
        $V_{Sphere} \leftarrow V_{Sphere} + \text{OctantVolume}(R, x_b, y_b, z_b)$
      end for
    end for
  end for
  return V_{Sphere}
end procedure

Algorithm 5 Compute the shell-sphere intersection volume.

Ensure: $R_{min} \geq 0,\ R_{max} > 0,\ R_{min} < R_{max}$

procedure ShellSphereVolume($R_{min}, R_{max}, R_{boundary}, x_c, y_c, z_c, CentralP$)
  $dx \leftarrow x_c - CentralP.XPos$
  $dy \leftarrow y_c - CentralP.YPos$
  $dz \leftarrow z_c - CentralP.ZPos$
  $PartD \leftarrow \sqrt{dx^2 + dy^2 + dz^2}$ ▷ Distance between the sphere and the shell centers
  $InnerSphere \leftarrow \text{SphereSphereVolume}(R_{min}, R_{boundary}, PartD)$
  $OuterSphere \leftarrow \text{SphereSphereVolume}(R_{max}, R_{boundary}, PartD)$
  return $OuterSphere - InnerSphere$
end procedure

Algorithm 6 Compute the sphere-sphere intersection volume.

Ensure: $R_{sphere} \geq 0,\ R_{boundary} > 0, Dist > 0$

procedure SphereSphereVolume($R_{sphere}, R_{boundary}, Dist$)
  if $PartD > R_{sphere} + R_{boundary}$ **then** ▷ Check if the spheres intersect
    return 0
  end if
  $A \leftarrow R_{boundary} - \frac{R_{boundary}^2 - R_{sphere}^2 + Dist^2}{2 \cdot Dist}$
  $B \leftarrow \frac{R_{boundary}^2 - R_{sphere}^2 + PartD^2}{2 \cdot Dist} - (Dist - R_{sphere})$
  $InnerCap \leftarrow A \cdot (3 \cdot R_{boundary} - A)$
  $OuterCap \leftarrow B \cdot (3 \cdot R_{sphere} - B)$
  $IntersectVol \leftarrow \frac{\pi}{3} \cdot (OuterCap + InnerCap)$
  return IntersectVol
end procedure

Algorithm 7 Compute g(r) from a particle set bounded by a rectangular box.

Ensure: $\Delta r > 0, \textit{Rectangular box boundary.}$ ▷ Positive radial bin size

procedure RDF($Particles, \Delta r$)
  $Bounds \leftarrow$ FINDBOUNDARY($Particles$) ▷ Determine the sample boundaries
  $V_{Sample} \leftarrow Bounds.L_x \cdot Bounds.L_y \cdot Bounds.L_z$
  $RhoMean \leftarrow \frac{Particles.size()}{V_{Sample}}$ ▷ Mean particle density
  $Rmax \leftarrow \frac{1}{2} \cdot \sqrt{Bounds.L_x^2 + Bounds.L_y^2 + Bounds.L_z^2}$ ▷ Maximal radial distance
  $BinNum \leftarrow \lfloor \frac{Rmax}{\Delta r} \rfloor$ ▷ Number of radial bins
  $r \leftarrow [\,]$ ▷ Empty list for the radial bins
  for $k = 0, k < BinNum$ **do**
    $r.append((k + 0.5) \cdot \Delta r)$ ▷ List with radial bin centers
    $k \leftarrow k + 1$
  end for
  $Gr \leftarrow [0...]$ ▷ Empty bins for g(r)
  $NonEmptyShells \leftarrow [0...]$ ▷ Keeps track of non empty shells
  for $CentralP \in Particles$ **do** ▷ Use every particle as the center
    $LocalGr \leftarrow [0...]$ ▷ Empty bins for the local g(r)
    for $Neighbor \in Particles \wedge Neighbor \neq CentralP$ **do**
      $D \leftarrow$ DISTANCE($Neighbor, CentralP$) ▷ Inter particle distance
      $Idx \leftarrow \lfloor \frac{1}{\Delta r} \cdot D \rfloor$ ▷ Index into $LocalGr$
      $LocalGr[Idx] \leftarrow LocalGr[Idx] + 1$ ▷ Bin the particle
    end for
    $ShiftBounds \leftarrow$ SHIFTCENTER($Bounds, CentralP$) ▷ Shift the COSY center to $CentralP$
    for $k = 0, k < BinNum$ **do** ▷ Compute the local particle density
      $V_{shell} \leftarrow$ SHELLVOLUME($r[k] - \frac{\Delta r}{2}, r[k] + \frac{\Delta r}{2}, ShiftBounds$)
      if $V_{shell} > 0$ **then**
        $LocalGr[k] \leftarrow LocalGr[k]/V_{shell}$ ▷ Use the shell-box intersection volume
        $NonEmptyShells[k] \leftarrow NonEmptyShells[k] + 1$
      end if ▷ Empty intersection volumes are not used
    end for
    $Gr \leftarrow Gr + LocalGr$ ▷ Add up all local g(r) bins for averaging
  end for
  for $k = 0, k < BinNum$ **do**
    $Gr[k] \leftarrow Gr[k]/NonEmptyShells[k]$ ▷ Average over all non emtpy shells
  end for
  $Gr \leftarrow \frac{Gr}{RhoMean}$ ▷ Final normalization by the average particle density
  return r, Gr
end procedure

Algorithm 8 Compute g(r) from a particle set bounded by a sphere.

Ensure: $\Delta r > 0, \textit{Rectangular box boundary.}$ ▷ Positive radial bin size

procedure RDF($Particles, \Delta r$)
  $x_c \leftarrow \textsc{Mean}(Particles.XPos)$ ▷ Determine the boundary sphere center
  $y_c \leftarrow \textsc{Mean}(Particles.YPos)$
  $z_c \leftarrow \textsc{Mean}(Particles.ZPos)$
  $Radii \leftarrow \textsc{Distance}(Particles, (x_{center}, y_{center}, z_{center}))$ ▷ Radial particle distances from the center
  $R_{boundary} \leftarrow \textsc{Max}(Radii)$ ▷ Sphere surrounding all particles
  $V_{Sample} \leftarrow \frac{4}{3} \cdot \pi \cdot R^3_{boundary}$
  $RhoMean \leftarrow \frac{Particles.size()}{V_{Sample}}$ ▷ Mean particle density
  $BinNum \leftarrow \lfloor \frac{R_{boundary}}{\Delta r} \rfloor$ ▷ Number of radial bins
  $r \leftarrow [\,]$ ▷ Empty list for the radial bins
  for $k = 0, k < BinNum$ **do**
    $r.append((k + 0.5) \cdot \Delta r)$ ▷ List with radial bin centers
    $k \leftarrow k + 1$
  end for
  $Gr \leftarrow [0...]$ ▷ Empty bins for g(r)
  for $CentralP \in Particles$ **do** ▷ Use every particle as the center
    $LocalGr \leftarrow [0...]$ ▷ Empty bins for the local g(r)
    for $Neighbor \in Particles \wedge Neighbor \neq CentralP$ **do**
      $dx \leftarrow Neighbor.XPos - CentralP.XPos$ ▷ Inter particle distance
      $dy \leftarrow Neighbor.YPos - CentralP.YPos$
      $dz \leftarrow Neighbor.ZPos - CentralP.ZPos$
      $Idx \leftarrow \lfloor \frac{1}{\Delta r} \cdot \sqrt{dx^2 + dy^2 + dz^2} \rfloor$ ▷ Index into $LocalGr$
      $LocalGr[Idx] \leftarrow LocalGr[Idx] + 1$ ▷ Bin the particle
    end for
    for $k = 0, k < BinNum$ **do** ▷ Compute the local particle density
      $V_{shell} \leftarrow \textsc{ShellSphereVolume}(r[k] - \frac{\Delta r}{2}, r[k] + \frac{\Delta r}{2}, R_{boundary}, x_c, y_c, z_c, CentralP)$
      if $V_{shell} > 0$ **then**
        $LocalGr[k] \leftarrow LocalGr[k]/V_{shell}$ ▷ Use the shell-box intersection volume
      end if
    end for
    $Gr \leftarrow Gr + LocalGr$ ▷ Add up all local g(r) bins for averaging
  end for
  $Gr \leftarrow \frac{Gr}{Particles.size()}$ ▷ Average over all particles
  $Gr \leftarrow \frac{Gr}{RhoMean}$ ▷ Final normalization by the average particle density
  return r, Gr
end procedure

4 Source code

4.1 FCC Coordinate Generation

```
from numpy import sqrt
from random import random
from numpy.random import normal
from matplotlib.pyplot import *
from mpl_toolkits.mplot3d import Axes3D

"""
This program calculates FCC coordinates.
A 3D Gaussian disturbance can be added.
Random Sphere Point Picking Algorithm from:
G. Marsaglia, The Annals of Mathematical Statistics,
1972, Vol. 43, No. 2, 845-646
"""

### DASHBOARD

SimuName = "FCC_Blurr_Cube"  # Name of the Experiment and the Files
GitterKonst = sqrt(2)        # Cell constant for the FCC unit cell
PartDiam = 1                 # Diameter of the Particles
Blurr = True                 # add a 3D gaussian blurr?
sigma = 0.07                 # standard deviation of the blurr
Lx = 4                       # Distance in x-direction from -Lx to Lx
Ly = 4                       # Distance in y-direction from -Ly to Ly
Lz = 4                       # Distance in z-direction from -Lz to Lz

"""************************************************************"""

def calcFCCCoor(GitterKonst, Lx, Ly, Lz):

    CoorList = []

    XLim = int(Lx/GitterKonst*5)
    YLim = int(Ly/GitterKonst*5)
    ZLim = int(Lz/GitterKonst*5)

    L = GitterKonst/2

    A = [0, L, L]
    B = [L, 0, L]
    C = [L, L, 0]

    for h in range(-ZLim*2, ZLim*2, 1):
        for k in range(-YLim*2, YLim*2, 1):
            for l in range(-XLim*2, XLim*2, 1):

                NewX = h * A[0] + k * B[0] + l * C[0]
                NewY = h * A[1] + k * B[1] + l * C[1]
                NewZ = h * A[2] + k * B[2] + l * C[2]

                if abs(NewX) < Lx and\
                   abs(NewY) < Ly and\
                   abs(NewZ) < Lz:

                    CoorList.append([NewX, NewY, NewZ])

    return CoorList

"""************************************************************"""

def getRandOffset(sigma):

    x1 = 2
    x2 = 2

    while x1**2 + x2**2  >= 1:

        x1 = random()*2-1
        x2 = random()*2-1
```

```
69
70      R = normal(0, sigma)
71
72      Wurzel = sqrt(1 - x1**2 - x2**2)
73      x = 2 * x1 * Wurzel
74      y = 2 * x2 * Wurzel
75      z = 1 - 2 * (x1**2 + x2**2)
76
77      return [x * R, y * R, z * R]
78
79  """************************************************************"""
80
81  def addGaussianBlurr(CoorList, sigma):
82
83      BlurrList = []
84
85      for P in CoorList:
86
87          Offset = getRandOffset(sigma)
88
89          NewX = P[0] + Offset[0]
90          NewY = P[1] + Offset[1]
91          NewZ = P[2] + Offset[2]
92
93          BlurrList.append([NewX, NewY, NewZ])
94
95      return BlurrList
96
97  """************************************************************"""
98
99  def plotCorr(CoorList, SimuName):
100
101     X = [0] * len(CoorList)
102     Y = [0] * len(CoorList)
103     Z = [0] * len(CoorList)
104
105     for P in range(0, len(CoorList), 1):
106
107         X[P] = CoorList[P][0]
108         Y[P] = CoorList[P][1]
109         Z[P] = CoorList[P][2]
110
111     fig = figure()
112     ax = fig.add_subplot(111, projection = "3d", aspect = "equal")
113
114     ax.scatter(X, Y, Z)
115
116     ax.set_xlabel("X")
117     ax.set_ylabel("Y")
118     ax.set_zlabel("Z")
119
120     savefig(SimuName + "_3D.png", dpi = 600)
121     show()
122
123     return True
124
125 """************************************************************"""
126
127 def saveData(CoorList, SimuName):
128
129     DataFile = open(SimuName + "_rawCoor.txt", "w")
130
131     for Particle in CoorList:
132
133         DataFile.write(str(Particle[0]) + "\t" +\
134                        str(Particle[1]) + "\t" +\
135                        str(Particle[2]) + "\n")
136
137     DataFile.close()
138
139     return True
140
141 """************************************************************"""
```

```
142
143 def createAutoCADFile(CoorList, PartDiam, SimuName):
144
145     CADFile = open(SimuName + "_CAD.scr", "w")
146
147     for Particle in CoorList:
148
149         CADFile.write("SPHERE\n")  # SPHERE command
150         CADFile.write(str(round(Particle[0], 5)) + ",") # X-Position
151         CADFile.write(str(round(Particle[1], 5)) + ",") # Y-Position
152         CADFile.write(str(round(Particle[2], 5)) + "\n")# Z-Position
153         CADFile.write(str(PartDiam/2) + "\n") # Radius
154
155     CADFile.write("\n")  # Blank line at the end
156     CADFile.close()
157
158     return True
159
160 """************************************************************"""
161
162 ###MAIN
163
164 print("Calculating FCC Coordinates.")
165 FCC = calcFCCCoor(GitterKonst, Lx, Ly, Lz)
166
167 print("Particle Number: " + str(len(FCC)))
168
169 if Blurr == True:
170
171     print("Adding a Gaussian blurr with Sigma = " + str(sigma))
172     FCC = addGaussianBlurr(FCC, sigma)
173
174 print("Saveing the Coordinats to a .txt file.")
175 saveData(FCC, SimuName)
176
177 print("Creating a AutoCAD file.")
178 createAutoCADFile(FCC, PartDiam, SimuName)
179
180 print("Plotting the Coordinates.")
181 plotCorr(FCC, SimuName)
```

4.2 Simple g(r) Implementation in Python 3

```
1  from scipy import spatial
2  from numpy import sqrt, pi, zeros
3
4  """
5  This function calculates the pair correlation function g(2)(r)
6  for a dense set of particles in a rectengular box. The particle
7  coordinates are supplied as [[x, y, z],...] lists.
8  All particles are used as central particles.
9  The spherical shell might extend beyond the known region.
10 A KDTree data structure is used for efficiency.
11 """
12
13 """************************************************************"""
14
15 def RDF_Simple(Particles, r, dr):
16
17     # preallocate list for Gr
18     Gr = zeros(len(r))
19
20     # maximal radial distance
21     MaxDist = r[-1] + dr/2
22
23     # sort all Particles in a KDTree
24     ParticleTree = spatial.KDTree(Particles)
25
26     # use every particle as the center once
27     k = 0
28     for CentralP in Particles:
```

```
29
30          # these are the indices for the particles at most MaxDist away:
31          NNIndices = ParticleTree.query_ball_point(CentralP,\
32                                                    MaxDist, p = 2, eps = 0)
33
34          # look at every other particle at most MaxDist away:
35          for Neighbour in NNIndices:
36
37              if CentralP != Particles[Neighbour]:
38
39                  # calc the distance to the neighbour
40                  dx = CentralP[0] - Particles[Neighbour][0]
41                  dy = CentralP[1] - Particles[Neighbour][1]
42                  dz = CentralP[2] - Particles[Neighbour][2]
43
44                  d = sqrt(dx**2 + dy**2 + dz**2)
45
46                  # what bins is the particle in?
47                  IdxList = [k for k in range(0, len(r), 1)\
48                             if abs(r[k] - d) <= dr]
49
50                  # add one to every bin the particle is in
51                  for Pos in IdxList:
52
53                      # count the particle
54                      Gr[Pos] += 1
55          k += 1
56          print("Finished Particle " + str(k) + " of " + str(len(Particles)))
57
58      # final normalization
59      Gr[0] = 0
60
61      BoxVol = (max(Particles[0]) - min(Particles[0]))*\
62               (max(Particles[1]) - min(Particles[1]))*\
63               (max(Particles[2]) - min(Particles[2]))
64
65      for i in range(1, len(r), 1):
66
67          Gr[i] /= len(Particles) # average over all central particles
68
69          # the most inner shells are spheres!
70          if r[i]- dr/2 > 0:
71
72              # by the shell volume
73              Gr[i] /= 4/3 *pi * ((r[i] + dr/2)**3 - (r[i]- dr/2)**3)
74
75          else:
76              Gr[i] /= 4/3 *pi * (r[i] + dr/2)**3 # by the shell volume
77
78          Gr[i] *= BoxVol / len(Particles) # by the number density
79
80      return Gr
81
82  """*********************************************************"""
```

4.3 Analytic g(r) Implementation in Python 3

```
1  from numpy import sqrt, pi, zeros, ones, arctan
2  from random import random, seed
3  from time import time
4  seed(time())
5
6  """
7  This function calculates the pair correlation function g(2)(r)
8  for a dense set of particles in a rectengular box. The particle
9  coordinates are supplied as [[x, y, z],...] lists.
10 All particles are used as central particles.
11 The spherical shell might extend beyond the known region.
12 An empty sourrounding is assumed.
13 """
14
```

```
15  """*********************************************************"""
16
17  def SphereCutVol(Rs, A, B):
18
19      Root = sqrt(Rs**2-A**2-B**2)
20
21      Vcut = 1/6*Rs**3*(pi - 2*arctan(A*B/(Rs*Root)))
22      Vcut += 1/2 * (arctan(A/Root) - pi/2)*(Rs**2*B-1/3*B**3)
23      Vcut += 1/2 * (arctan(B/Root) - pi/2)*(Rs**2*A-1/3*A**3)
24      Vcut += 1/3 * A * B * Root
25
26      return Vcut
27
28  """*********************************************************"""
29
30  def OctVolume(Rs, xb, yb, zb):
31
32      # if all boundaries are fully in the octant
33      if xb**2 + yb**2 + zb**2 < Rs**2:
34
35          return xb * yb * zb
36
37      # if no boundary intersects we start with
38      VOctant = 1/8 * 4/3 * pi * Rs**3
39
40      # remove the spherical caps
41      for B in [xb, yb, zb]:
42
43          if B < Rs:
44
45              VOctant -= pi/4*(2/3*Rs**3-B*Rs**2+1/3*B**3)
46
47      # add the intersections of the caps
48      for (a, b) in [(xb, yb), (xb, zb), (yb, zb)]:
49
50          if a**2 + b**2 < Rs**2:
51
52              VOctant += SphereCutVol(Rs, a, b)
53
54      return VOctant
55
56  """*********************************************************"""
57
58  def SphereVolume(Rs, BoxBounds):
59
60      [Xmin, Xmax, Ymin, Ymax, Zmin, Zmax] = BoxBounds
61
62      VSphere = 0
63
64      # abs() mirrors the boundaries into the first octant
65      for xb in [Xmin, Xmax]:
66          for yb in [Ymin, Ymax]:
67              for zb in [Zmin, Zmax]:
68
69                  VSphere += OctVolume(Rs, abs(xb), abs(yb), abs(zb))
70
71      return VSphere
72
73  """*********************************************************"""
74
75  def ShellVolume(Rmin, Rmax, BoxBounds):
76
77      # check for negative Rmin values
78      Rmin = max([Rmin, 0])
79
80      InnerShell = SphereVolume(Rmin, BoxBounds)
81      OuterShell = SphereVolume(Rmax, BoxBounds)
82
83      Volume = OuterShell - InnerShell
84
85  ##      if Volume <= 0:
86  ##          print("Volume = " + str(Volume))
87  ##          print("Rmin = " + str(Rmin))
```

```
88  ##          print("Rmax = " + str(Rmax))
89  ##          print("BoxBounds: " + str(BoxBounds))
90
91      return Volume
92
93  """*********************************************************"""
94
95  def RDF_AnalyticNorm(Particles, r, dr):
96
97      # Gr averaged over all particles
98      Global_Gr = zeros(len(r))
99
100     # Keep track of the usefull shell volumes
101     NonEmptyShells = zeros(len(r))
102
103     # maximal radial distance
104     MaxDist = r[-1] + dr/2
105
106     # Box boundaries
107     XList = [Particles[k][0] for k in range(0, len(Particles), 1)]
108     YList = [Particles[k][1] for k in range(0, len(Particles), 1)]
109     ZList = [Particles[k][2] for k in range(0, len(Particles), 1)]
110
111     BoxBounds = [min(XList), max(XList),\
112                  min(YList), max(YList),\
113                  min(ZList), max(ZList)]
114
115     # box size
116     Lx = BoxBounds[1] - BoxBounds[0]
117     Ly = BoxBounds[3] - BoxBounds[2]
118     Lz = BoxBounds[5] - BoxBounds[4]
119
120     print("Lx = " + str(Lx))
121     print("Ly = " + str(Ly))
122     print("Lz = " + str(Lz))
123
124     MeanDensity = len(Particles) / (Lx * Ly * Lz)
125
126     # use every particle as the center once
127     for CentralP in range(0, len(Particles), 1):
128
129         # local Gr around the current particle
130         Local_Gr = zeros(len(r))
131
132         # look at every other particle at most MaxDist away:
133         for Neighbour in range(0, len(Particles), 1):
134
135             if CentralP != Neighbour:
136
137                 # calc the distance to the neighbour
138                 dx = Particles[CentralP][0] - Particles[Neighbour][0]
139                 dy = Particles[CentralP][1] - Particles[Neighbour][1]
140                 dz = Particles[CentralP][2] - Particles[Neighbour][2]
141
142                 d = sqrt(dx**2 + dy**2 + dz**2)
143
144                 # what bins is the particle in?
145                 IdxList = [k for k in range(0, len(r), 1) if abs(r[k] - d) <= dr/2]
146
147                 # add one to every bin the particle is in
148                 for Pos in IdxList:
149
150                     # count the particle
151                     Local_Gr[Pos] += 1
152
153         # shift the center of box cosy
154         LocalBox = [BoxBounds[0] - Particles[CentralP][0],\
155                     BoxBounds[1] - Particles[CentralP][0],\
156                     BoxBounds[2] - Particles[CentralP][1],\
157                     BoxBounds[3] - Particles[CentralP][1],\
158                     BoxBounds[4] - Particles[CentralP][2],\
159                     BoxBounds[5] - Particles[CentralP][2]]
160
```

```
161         # normalize with the shell volume
162         for RIdx in range(0, len(r), 1):
163 
164             SVolume = ShellVolume(r[RIdx]-dr/2, r[RIdx]+dr/2, LocalBox)
165 
166             if SVolume > 0.0:
167 
168                 Local_Gr[RIdx] /= SVolume
169                 NonEmptyShells[RIdx] += 1
170 
171         # normalize by the mean particle density
172         Local_Gr = Local_Gr / MeanDensity
173 
174         # save in the global gr for the average over particles
175         Global_Gr = Global_Gr + Local_Gr
176 
177         print("Finished Particle " + str(CentralP) + " of " + str(len(Particles)))
178 
179     # final normalization considering the non empty shell volumes
180 
181     for k in range(0, len(Global_Gr), 1):
182 
183         if NonEmptyShells[k] != 0:
184 
185             Global_Gr[k] /= NonEmptyShells[k]
186 
187         else:
188             print("All Shells at R =  " + str(r[k]) + " are Empty!")
189 
190     return Global_Gr
191 
192 """************************************************************"""
```

4.4 Analytic - Internal Volume

```
1  def VInt_Analytic(Rs, xb, zb):
2  
3      S = sqrt(Rs**2-xb**2-zb**2)
4  
5      Vint = 1/3 * (xb * zb * S)
6      Vint += 1/2 * arctan(xb/S)*(Rs**2*zb-(1/3)*zb**3)
7      Vint += 1/2 * arctan(zb/S)*(Rs**2*xb-(1/3)*xb**3)
8      Vint -= 1/3 * Rs**3 * arctan(xb*zb/(Rs * S))
9  
10     return Vint
```

4.5 Monte Carlo - Internal Volume

```
1  def VInt_MC(Rs, xb, zb, NPoints):
2  
3      InsidePoints = 0
4  
5      for k in range(0, NPoints, 1):
6  
7          # random point in the first (Rs,Rs,Rs) octant
8          XRand = random() * Rs
9          YRand = random() * Rs
10         ZRand = random() * Rs
11 
12         if XRand**2 + YRand**2 + ZRand**2 < Rs**2 and\
13            XRand < xb and ZRand < zb:
14 
15             InsidePoints += 1
16 
17     return InsidePoints / NPoints * Rs**3
```

4.6 Numeric Integration - Internal Volume

```
from numpy import sqrt, pi, arctan, linspace, array
from scipy.integrate import quad

def VInt_Num(Rs, xb, zb):

    def height(x, z, Rs):

        return sqrt(Rs**2 - x**2 - z**2)

    def VInt(zb, Rs):

        return quad(height, 0, xb, args = (zb, Rs))[0]

    Volume = quad(VInt, 0, zb, args = (Rs))[0]

    return Volume
```

4.7 Analytic - Spherical Cut Volume

```
def SphereCutVol(Rs, A, B):

    Root = sqrt(Rs**2-A**2-B**2)

    Vcut = 1/6*Rs**3*(pi - 2*arctan(A*B/(Rs*Root)))
    Vcut += 1/2 * (arctan(A/Root) - pi/2)*(Rs**2*B-1/3*B**3)
    Vcut += 1/2 * (arctan(B/Root) - pi/2)*(Rs**2*A-1/3*A**3)
    Vcut += 1/3 * A * B * Root

    return Vcut
```

4.8 The Octant Sum

```
def SphereVolume(Rs, Xmin, Xmax, Ymin, Ymax, Zmin, Zmax):

    VSphere = 0

    # abs() mirrors the boundaries into the first octant
    for xb in [abs(Xmin), abs(Xmax)]:
        for yb in [abs(Ymin), abs(Ymax)]:
            for zb in [abs(Zmin), abs(Zmax)]:

                VSphere += OctVolume(Rs, xb, yb, zb)

    return VSphere
```

4.9 The Octant Volume

```
def OctVolume(Rs, xb, yb, zb):

    # if all boundaries intersect inside the octant
    if xb**2 + yb**2 + zb**2 < Rs**2:

        return xb * yb * zb

    # if they dont intersect we start with a full octant
    VOctant = 1/8 * 4/3 * pi * Rs**3

    # then, we remove the spherical caps
    for B in [xb, yb, zb]:

        if B < Rs:
            VOctant -= pi/4*(2/3*Rs**3-B*Rs**2+1/3*B**3)

```

```
17      # finally, we add the intersections of the caps
18      for (a, b) in [(xb, yb), (xb, zb), (yb, zb)]:
19
20          if a**2 + b**2 < Rs**2:
21
22              VOctant += VSphereCut(Rs, a, b)
23
24      return VOctant
```

4.10 Monte Carlo - Spherical Cut Volume

```
def SphereCutVol_MC(Rs, A, B, NPoints):

    InsidePoints = 0

    for k in range(0, NPoints, 1):

        # random point in the first (Rs,Rs,Rs) octant
        XRand = random() * Rs
        YRand = random() * Rs
        ZRand = random() * Rs

        if XRand**2 + YRand**2 + ZRand**2 < Rs**2 and\
           XRand > A and YRand > B:

            InsidePoints += 1

    return InsidePoints / NPoints * Rs**3
```

4.11 Monte Carlo - Sphere Volume

```
def SphereVolume_MC(Rs, Xmin, Xmax, Ymin, Ymax, Zmin, Zmax, NPoints):

    InsidePoints = 0

    for k in range(0, NPoints, 1):

        # random point in the box
        XRand = random() * (Xmax-Xmin) + Xmin
        YRand = random() * (Ymax-Ymin) + Ymin
        ZRand = random() * (Zmax-Zmin) + Zmin

        # if the point is inside the sphere
        if XRand**2 + YRand**2 + ZRand**2 < Rs**2:

            InsidePoints += 1

    return InsidePoints / NPoints*(Xmax-Xmin)*(Ymax-Ymin)*(Zmax-Zmin)
```

4.12 Simple g(r) in C++

```
#include <iostream>
#include <sstream>
#include <chrono>
#include <cmath>
#include <cstdlib>
#include <string>
#include <fstream>
#include <istream>
#include <vector>
#include <streambuf>

long double pi = 3.14159265358979323846264338;

/*****************************************************************/
struct Box
{
  long double Xmin = 0;
  long double Xmax = 0;
  long double Ymin = 0;
  long double Ymax = 0;
  long double Zmin = 0;
  long double Zmax = 0;
  long int PartNum = 0;
};

```

```
26  /*************************************************************************/
27
28  struct Particle
29  {
30    long double XPos = 0;
31    long double YPos = 0;
32    long double ZPos = 0;
33  };
34
35  /*************************************************************************/
36
37  std::vector<std::string> split(const std::string& s, char delimiter)
38  {
39     std::vector<std::string> tokens;
40     std::string token;
41     std::istringstream tokenStream(s);
42
43     while (std::getline(tokenStream, token, delimiter))
44     {
45        tokens.push_back(token);
46     }
47     return tokens;
48  };
49
50  /*************************************************************************/
51
52  inline std::vector<Particle> readPartCoor(const std::string FileName)
53  {
54    // open the raw data file
55    std::ifstream DataFile(FileName + ".txt");
56    std::string TxtData;
57
58    // get length of file:
59    DataFile.seekg(0, std::ios::end);   // go to the end of the file
60    TxtData.reserve(DataFile.tellg());  // allocate enough space
61    DataFile.seekg(0, std::ios::beg);   // go to the beginning of the file
62
63    TxtData.assign((std::istreambuf_iterator<char>(DataFile)),
64             std::istreambuf_iterator<char>());
65
66    DataFile.close();
67
68    // split the data into lines
69    std::vector<std::string> datalines = split(TxtData, '\n');
70
71    Particle P;  // single particle struct
72    std::vector<Particle> Particles; // list of particles
73
74    std::cout << "The␣first␣lines␣in␣the␣raw␣data␣file␣are:" << std::endl;
75    std::cout << datalines.at(0) << std::endl;
76    std::cout << datalines.at(1) << std::endl;
77    std::cout << datalines.at(2) << std::endl;
78    std::cout << datalines.at(3) << std::endl;
79
80    std::cout << "The␣RawDataFile␣has␣" << datalines.size() << "␣lines." << std::endl;
81
82    for(long int k = 0; k < datalines.size() ; k++)
83    {
84      std::vector<std::string> results;
85
86      // split the individual lines (X, Y, Z)
87      results = split(datalines.at(k), '\t');
88
89      // save the single particle positions
90      P.XPos = std::stold(results.at(0));
91      P.YPos = std::stold(results.at(1));
92      P.ZPos = std::stold(results.at(2));
93
94      Particles.push_back(P);
95    }
96
97    return Particles;
98  };
```

```
99
100 /*****************************************************************/
101
102 inline Box FindBoxBounds(const std::vector<Particle> Particles)
103 {
104   // Box boundaries
105   Box BoxBounds;
106
107   // initialize the boundaries with the first particle
108   BoxBounds.Xmin = Particles.at(0).XPos;
109   BoxBounds.Xmax = Particles.at(0).XPos;
110   BoxBounds.Ymin = Particles.at(0).YPos;
111   BoxBounds.Ymax = Particles.at(0).YPos;
112   BoxBounds.Zmin = Particles.at(0).ZPos;
113   BoxBounds.Zmax = Particles.at(0).ZPos;
114   BoxBounds.PartNum = Particles.size();
115
116   // loop over all other particles and find minima and maxima
117   for(long int Pos = 1; Pos < Particles.size(); Pos++)
118   {
119     if(Particles.at(Pos).XPos < BoxBounds.Xmin)
120       BoxBounds.Xmin = Particles.at(Pos).XPos;
121
122     if(Particles.at(Pos).XPos > BoxBounds.Xmax)
123       BoxBounds.Xmax = Particles.at(Pos).XPos;
124
125     if(Particles.at(Pos).YPos < BoxBounds.Ymin)
126       BoxBounds.Ymin = Particles.at(Pos).YPos;
127
128     if(Particles.at(Pos).YPos > BoxBounds.Ymax)
129       BoxBounds.Ymax = Particles.at(Pos).YPos;
130
131     if(Particles.at(Pos).ZPos < BoxBounds.Zmin)
132       BoxBounds.Zmin = Particles.at(Pos).ZPos;
133
134     if(Particles.at(Pos).ZPos > BoxBounds.Zmax)
135       BoxBounds.Zmax = Particles.at(Pos).ZPos;
136   };
137
138   /* print the boundaries to the console
139   std::cout << "I found the Box Boundaries: " << std::endl;
140   std::cout << "Xmin = " << BoxBounds.Xmin << std::endl;
141   std::cout << "Xmax = " << BoxBounds.Xmax << std::endl;
142   std::cout << "Ymin = " << BoxBounds.Ymin << std::endl;
143   std::cout << "Ymax = " << BoxBounds.Ymax << std::endl;
144   std::cout << "Zmin = " << BoxBounds.Zmin << std::endl;
145   std::cout << "Zmax = " << BoxBounds.Zmax << std::endl;
146   */
147
148   return BoxBounds;
149 };
150
151 /*****************************************************************/
152
153 inline std::vector<long double> RDF_AnalyticNorm(const std::vector<Particle> Particles,
154                           const std::vector<long double> r,
155                          const long double dr, const Box BoxBounds)
156 {
157     // Gr averaged over all particles
158     std::vector<long double> Global_Gr(r.size(), 0);
159
160     // maximal radial distance
161     long double MaxDist = r.at(r.size()-1) + dr/2.0;
162
163     // box size
164     long double Lx = BoxBounds.Xmax - BoxBounds.Xmin;
165     long double Ly = BoxBounds.Ymax - BoxBounds.Ymin;
166     long double Lz = BoxBounds.Zmax - BoxBounds.Zmin;
167
168   // Show this to the user
169   std::cout << "The Sample Volume has the following size:" << std::endl;
170     std::cout << "Lx = " << Lx << std::endl;
171     std::cout << "Ly = " << Ly << std::endl;
```

```
172     std::cout << "Lz = " << Lz << std::endl;
173
174     long double MeanDensity = Particles.size() / (Lx * Ly * Lz);
175
176     // use every particle as the center once
177     for(long int CentralP = 0; CentralP < Particles.size(); CentralP++)
178   {
179         // local Gr around the current particle
180         std::vector<long double> Local_Gr(r.size(), 0);
181
182         // look at every other particle at most MaxDist away:
183         for(long int Neighbour = 0; Neighbour < Particles.size(); Neighbour++)
184     {
185             if(CentralP != Neighbour)
186       {
187                 // calc the distance to the neighbour
188                 long double dx = Particles.at(CentralP).XPos - Particles.at(Neighbour).XPos;
189                 long double dy = Particles.at(CentralP).YPos - Particles.at(Neighbour).YPos;
190                 long double dz = Particles.at(CentralP).ZPos - Particles.at(Neighbour).ZPos;
191
192                 long double d = std::sqrt(dx*dx + dy*dy + dz*dz);
193
194         // what bins is the particle in?
195         for(long int RPos = 0; RPos < r.size(); RPos++)
196         {
197           if(std::abs(r.at(RPos) - d) <= dr/2.0)
198           {
199             Local_Gr.at(RPos) += 1;
200           }
201         }
202       }
203     }
204
205     // local box for the shell box intersection volume
206     Box LocalBox = BoxBounds;
207
208     // shift the center of the box cosy
209     LocalBox.Xmin = BoxBounds.Xmin - Particles.at(CentralP).XPos;
210     LocalBox.Xmax = BoxBounds.Xmax - Particles.at(CentralP).XPos;
211     LocalBox.Ymin = BoxBounds.Ymin - Particles.at(CentralP).YPos;
212     LocalBox.Ymax = BoxBounds.Ymax - Particles.at(CentralP).YPos;
213     LocalBox.Zmin = BoxBounds.Zmin - Particles.at(CentralP).ZPos;
214     LocalBox.Zmax = BoxBounds.Zmax - Particles.at(CentralP).ZPos;
215
216     /*print the boundaries to the console
217     std::cout << "The local Box Boundaries are: " << std::endl;
218     std::cout << "Xmin = " << LocalBox.Xmin << std::endl;
219     std::cout << "Xmax = " << LocalBox.Xmax << std::endl;
220     std::cout << "Ymin = " << LocalBox.Ymin << std::endl;
221     std::cout << "Ymax = " << LocalBox.Ymax << std::endl;
222     std::cout << "Zmin = " << LocalBox.Zmin << std::endl;
223     std::cout << "Zmax = " << LocalBox.Zmax << std::endl;
224     */
225
226         // normalize with the shell volume
227         for(long int RIdx = 0; RIdx < r.size(); RIdx++)
228     {
229       // calculate the shell box intersection volume
230             long double SVolume = 4.0/3.0 * pi * (std::pow(r.at(RIdx)+dr/2.0, 3) - std::pow(r.at(RIdx)-
231             Local_Gr.at(RIdx) /= SVolume;
232
233       // normalize by the mean particle density
234       Local_Gr.at(RIdx) /= MeanDensity;
235
236       // save in the global gr for the average over particles
237       Global_Gr.at(RIdx) += Local_Gr.at(RIdx);
238     }
239
240     if(CentralP % 1000 == 0)
241       std::cout << "Finished Particle " << CentralP << " of " << Particles.size() << std::endl;
242   }
243
244     // final average over all particles
```

```
245       for(long int k = 0; k < Global_Gr.size(); k++)
246     {
247           Global_Gr.at(k) /= Particles.size();
248     }
249 
250     return Global_Gr;
251 };
252 
253 /*****************************************************************/
254 
255 void SaveData(const std::string SimuName, const std::string FileName, const std::vector<long double> Gr
256           const std::vector<long double> r, const long double dr, const long double ComputeTime,
257           Box BoxBounds)
258 {
259   // box size
260     long double Lx = BoxBounds.Xmax - BoxBounds.Xmin;
261     long double Ly = BoxBounds.Ymax - BoxBounds.Ymin;
262     long double Lz = BoxBounds.Zmax - BoxBounds.Zmin;
263 
264   long double RmaxNatural = std::sqrt(Lx*Lx + Ly*Ly + Lz*Lz)/2.0;
265 
266   std::ofstream DataFile;
267   DataFile.open(SimuName + ".txt", std::ios::out);
268   DataFile.precision(25);
269 
270   DataFile << "Results for the radial distribution function computation." << std::endl;
271   DataFile << "Particle File Name: " << FileName << std::endl;
272   DataFile << "Computation time for g(r): " << ComputeTime << " nanoseconds." << std::endl;
273   DataFile << "Radial Bin Width dr: " << dr << std::endl;
274   DataFile << "Number of radial Bins: " << r.size() << std::endl;
275   DataFile << "Number of Particles in the sample: " << BoxBounds.PartNum << std::endl;
276   DataFile << "Average Particle Number Density: " << BoxBounds.PartNum / (Lx * Ly * Lz) << std::endl;
277   DataFile << "Maximal useful radial distance: " << RmaxNatural << std::endl;
278   DataFile << "Box Boundary Xmin: " << BoxBounds.Xmin << std::endl;
279   DataFile << "Box Boundary Xmax: " << BoxBounds.Xmax << std::endl;
280   DataFile << "Box Boundary Ymin: " << BoxBounds.Ymin << std::endl;
281   DataFile << "Box Boundary Ymax: " << BoxBounds.Ymax << std::endl;
282   DataFile << "Box Boundary Zmin: " << BoxBounds.Zmin << std::endl;
283   DataFile << "Box Boundary Zmax: " << BoxBounds.Zmax << std::endl;
284 
285   DataFile << "Box Boundary X-length: " << Lx << std::endl;
286   DataFile << "Box Boundary Y-length: " << Ly << std::endl;
287   DataFile << "Box Boundary Z-length: " << Lz << std::endl;
288 
289   DataFile << "Box Volume: " << Lx * Ly * Lz << std::endl;
290 
291   DataFile << "# r \t g(r)" << std::endl;
292 
293   for(long int l = 0; l < r.size(); l++)
294   {
295     DataFile << r.at(l) << "\t";
296     DataFile << Gr.at(l) << std::endl;
297   }
298 
299   DataFile.close();
300 };
301 
302 /*****************************************************************/
303 
304 int main(void)
305 {
306   // Dashboard
307   std::string SimuName = "";     // Name of the simulation
308   std::string FileName = "";     // Name of the simulation
309   long int Bins = 1;                      // number of sample calculations
310   long double dr = 1;                    // bin size
311 
312   // User Input
313   std::cout << "Please enter the following information:" << std::endl;
314   std::cout << "Name for the simulation: ";
315   std::cin >> SimuName;
316   std::cout << "Particle File Name: ";
317   std::cin >> FileName;
```

```
318     std::cout << "Bin size (dr): ";
319     std::cin >> dr;
320     std::cout << "Starting g(r) computation." << std::endl;
321 
322     // output precision
323     std::cout.precision(20);
324 
325     // read in the raw particle coordinates
326     std::vector<Particle> Particles = readPartCoor(FileName);
327 
328     // benchmark the g(r) calculation
329     auto start = std::chrono::steady_clock::now();
330 
331     // find the Box Boundaries
332     Box BoxBounds = FindBoxBounds(Particles);
333 
334     // box size
335       long double Lx = BoxBounds.Xmax - BoxBounds.Xmin;
336       long double Ly = BoxBounds.Ymax - BoxBounds.Ymin;
337       long double Lz = BoxBounds.Zmax - BoxBounds.Zmin;
338 
339     // maximal useful radial distance
340     long double RMAX = std::sqrt(Lx*Lx + Ly*Ly + Lz*Lz)/2.0;
341 
342     // show the maximal usefull radius to the user
343     std::cout << "Maximal usefull radius: " << RMAX << std::endl;
344 
345     // Number of Bins
346     Bins = static_cast<long int>(RMAX / dr);
347 
348     // the radial position and g(r) vector
349     std::vector<long double> r(Bins, 0);
350     std::vector<long double> Gr(Bins, 0);
351 
352     // create the radial distance vector
353     for(long int k = 0; k < Bins; k++)
354     {
355       r.at(k) = k * RMAX/Bins;
356     }
357 
358     // compute the radial distribution function
359     Gr = RDF_AnalyticNorm(Particles, r, dr, BoxBounds);
360 
361     auto end = std::chrono::steady_clock::now();
362     auto diff = end - start;
363     double ComputeTime = std::chrono::duration <double, std::nano> (diff).count();
364 
365     // information for the user
366     std::cout << "Time for the g(r) computation: ";
367       std::cout << ComputeTime;
368       std::cout << " nanoseconds" << std::endl;
369 
370     // save all the data into a file
371     SaveData(SimuName, FileName, Gr, r, dr, ComputeTime, BoxBounds);
372 
373     std::cout << "Program Finished!" << std::endl;
374 };
```

4.13 Analytic g(r) in C++

```
1  #include <iostream>
2  #include <sstream>
3  #include <chrono>
4  #include <cmath>
5  #include <cstdlib>
6  #include <string>
7  #include <fstream>
8  #include <istream>
9  #include <vector>
10 #include <streambuf>
11 
```

```
12  long double pi = 3.14159265358979323846264433;
13
14  /******************************************************************/
15  struct Box
16  {
17    long double Xmin = 0;
18    long double Xmax = 0;
19    long double Ymin = 0;
20    long double Ymax = 0;
21    long double Zmin = 0;
22    long double Zmax = 0;
23    long int PartNum = 0;
24  };
25
26  /******************************************************************/
27
28  struct Particle
29  {
30    long double XPos = 0;
31    long double YPos = 0;
32    long double ZPos = 0;
33  };
34
35  /******************************************************************/
36
37  std::vector<std::string> split(const std::string& s, char delimiter)
38  {
39     std::vector<std::string> tokens;
40     std::string token;
41     std::istringstream tokenStream(s);
42
43     while (std::getline(tokenStream, token, delimiter))
44     {
45        tokens.push_back(token);
46     }
47     return tokens;
48  };
49
50  /******************************************************************/
51
52  inline std::vector<Particle> readPartCoor(const std::string FileName)
53  {
54    // open the raw data file
55    std::ifstream DataFile(FileName + ".txt");
56    std::string TxtData;
57
58    // get length of file:
59    DataFile.seekg(0, std::ios::end);    // go to the end of the file
60    TxtData.reserve(DataFile.tellg());  // allocate enough space
61    DataFile.seekg(0, std::ios::beg);    // go to the beginning of the file
62
63    TxtData.assign((std::istreambuf_iterator<char>(DataFile)),
64             std::istreambuf_iterator<char>());
65
66    DataFile.close();
67
68    // split the data into lines
69    std::vector<std::string> datalines = split(TxtData, '\n');
70
71    Particle P;  // single particle struct
72    std::vector<Particle> Particles; // list of particles
73
74    std::cout << "The␣first␣lines␣in␣the␣raw␣data␣file␣are:" << std::endl;
75    std::cout << datalines.at(0) << std::endl;
76    std::cout << datalines.at(1) << std::endl;
77    std::cout << datalines.at(2) << std::endl;
78    std::cout << datalines.at(3) << std::endl;
79
80    std::cout << "The␣RawDataFile␣has␣" << datalines.size() << "␣lines." << std::endl;
81
82    for(long int k = 0; k < datalines.size() ; k++)
83    {
84      std::vector<std::string> results;
```

```
85
86      // split the individual lines (X, Y, Z)
87      results = split(datalines.at(k), '\t');
88
89      // save the single particle positions
90      P.XPos = std::stold(results.at(0));
91      P.YPos = std::stold(results.at(1));
92      P.ZPos = std::stold(results.at(2));
93
94      Particles.push_back(P);
95    }
96
97    return Particles;
98  };
99
100 /*****************************************************************/
101
102 inline long double SphereCutVol(const long double Rs,
103                 const long double A,
104                 const long double B)
105 {
106     long double Root = std::sqrt(Rs*Rs-A*A-B*B);
107
108     long double Vcut = Rs*Rs*Rs*(pi - 2.0*std::atan(A*B/(Rs*Root))) / 6.0;
109     Vcut +=  (std::atan(A/Root) - pi/2.0)*(Rs*Rs*B-1.0/3.0*B*B*B) / 2.0;
110   Vcut += (std::atan(B/Root) - pi/2.0)*(Rs*Rs*A-1.0/3.0*A*A*A) / 2.0;
111   Vcut +=  A * B * Root / 3.0;
112
113   /*
114   if(Vcut <= 0)
115   {
116     std::cout << "Vcut is less than 0!  = " << Vcut << std::endl;
117     std::cout << "Rs is = " << Rs << std::endl;
118     std::cout << "A = " << A << std::endl;
119     std::cout << "B = " << B << std::endl;
120   }*/
121
122     return Vcut;
123 };
124
125 /*****************************************************************/
126
127 inline long double OctVolume(const long double Rs,
128                  const long double xb,
129                const long double yb,
130                const long double zb)
131 {
132
133   /*
134   std::cout << "Box Corner Bounds:" << std::endl;
135   std::cout << xb << std::endl;
136   std::cout << yb << std::endl;
137   std::cout << zb << std::endl;
138   */
139
140     // if all boundaries are fully in the octant
141     if(xb*xb + yb*yb + zb*zb < Rs*Rs)
142   {
143         return xb * yb * zb;
144   }
145
146   // if no boundary intersects we start with a full octant
147     long double VOctant = pi * Rs*Rs*Rs / 6.0;
148
149     // remove the spherical caps
150
151   if(xb < Rs)
152   {
153         VOctant -= pi/4.0*(2.0/3.0*Rs*Rs*Rs-xb*Rs*Rs+1.0/3.0*xb*xb*xb);
154   }
155
156   if(yb < Rs)
157   {
```

```
158           VOctant -= pi/4.0*(2.0/3.0*Rs*Rs*Rs-yb*Rs*Rs+1.0/3.0*yb*yb*yb);
159     }
160
161     if(zb < Rs)
162     {
163           VOctant -= pi/4.0*(2.0/3.0*Rs*Rs*Rs-zb*Rs*Rs+1.0/3.0*zb*zb*zb);
164     }
165
166       // add the intersections of the caps
167
168       if((xb*xb + yb*yb) < (Rs*Rs))
169     {
170           VOctant += SphereCutVol(Rs, xb, yb);
171     }
172
173     if((xb*xb + zb*zb) < (Rs*Rs))
174     {
175           VOctant += SphereCutVol(Rs, xb, zb);
176     }
177
178     if((yb*yb + zb*zb) < (Rs*Rs))
179     {
180           VOctant += SphereCutVol(Rs, yb, zb);
181     }
182
183       return VOctant;
184   };
185
186   /*********************************************************************/
187
188   inline long double SphereVolume(const long double Rs, const Box BoxSize)
189   {
190       long double VSphere = 0.0;
191
192       // std::abs() mirrors the boundaries into the first octant
193     VSphere += OctVolume(Rs, std::abs(BoxSize.Xmin),
194     std::abs(BoxSize.Ymin), std::abs(BoxSize.Zmin));
195     VSphere += OctVolume(Rs, std::abs(BoxSize.Xmin),
196     std::abs(BoxSize.Ymin), std::abs(BoxSize.Zmax));
197     VSphere += OctVolume(Rs, std::abs(BoxSize.Xmin),
198     std::abs(BoxSize.Ymax), std::abs(BoxSize.Zmin));
199     VSphere += OctVolume(Rs, std::abs(BoxSize.Xmin),
200     std::abs(BoxSize.Ymax), std::abs(BoxSize.Zmax));
201     VSphere += OctVolume(Rs, std::abs(BoxSize.Xmax),
202     std::abs(BoxSize.Ymin), std::abs(BoxSize.Zmin));
203     VSphere += OctVolume(Rs, std::abs(BoxSize.Xmax),
204     std::abs(BoxSize.Ymin), std::abs(BoxSize.Zmax));
205     VSphere += OctVolume(Rs, std::abs(BoxSize.Xmax),
206     std::abs(BoxSize.Ymax), std::abs(BoxSize.Zmin));
207     VSphere += OctVolume(Rs, std::abs(BoxSize.Xmax),
208     std::abs(BoxSize.Ymax), std::abs(BoxSize.Zmax));
209
210       return VSphere;
211   };
212
213   /*********************************************************************/
214
215   inline long double ShellVolume(const long double Rmin, const long double Rmax,
216                       const Box BoxSize)
217   {
218     long double InnerSphere = SphereVolume(Rmin, BoxSize);
219     long double OuterSphere = SphereVolume(Rmax, BoxSize);
220
221     long double ShellVol = OuterSphere - InnerSphere;
222
223     /*
224     if(ShellVol <= 0.0)
225     {
226       std::cout << ShellVol << std::endl;
227     }*/
228
229     return ShellVol;
230   };
```

```
231
232 /*****************************************************************/
233
234 inline Box FindBoxBounds(const std::vector<Particle> Particles)
235 {
236   // Box boundaries
237   Box BoxBounds;
238
239   // initialize the boundaries with the first particle
240   BoxBounds.Xmin = Particles.at(0).XPos;
241   BoxBounds.Xmax = Particles.at(0).XPos;
242   BoxBounds.Ymin = Particles.at(0).YPos;
243   BoxBounds.Ymax = Particles.at(0).YPos;
244   BoxBounds.Zmin = Particles.at(0).ZPos;
245   BoxBounds.Zmax = Particles.at(0).ZPos;
246   BoxBounds.PartNum = Particles.size();
247
248   // loop over all other particles and find minima and maxima
249   for(long int Pos = 1; Pos < Particles.size(); Pos++)
250   {
251     if(Particles.at(Pos).XPos < BoxBounds.Xmin)
252       BoxBounds.Xmin = Particles.at(Pos).XPos;
253
254     if(Particles.at(Pos).XPos > BoxBounds.Xmax)
255       BoxBounds.Xmax = Particles.at(Pos).XPos;
256
257     if(Particles.at(Pos).YPos < BoxBounds.Ymin)
258       BoxBounds.Ymin = Particles.at(Pos).YPos;
259
260     if(Particles.at(Pos).YPos > BoxBounds.Ymax)
261       BoxBounds.Ymax = Particles.at(Pos).YPos;
262
263     if(Particles.at(Pos).ZPos < BoxBounds.Zmin)
264       BoxBounds.Zmin = Particles.at(Pos).ZPos;
265
266     if(Particles.at(Pos).ZPos > BoxBounds.Zmax)
267       BoxBounds.Zmax = Particles.at(Pos).ZPos;
268   };
269
270   /* print the boundaries to the console
271   std::cout << "I found the Box Boundaries: " << std::endl;
272   std::cout << "Xmin = " << BoxBounds.Xmin << std::endl;
273   std::cout << "Xmax = " << BoxBounds.Xmax << std::endl;
274   std::cout << "Ymin = " << BoxBounds.Ymin << std::endl;
275   std::cout << "Ymax = " << BoxBounds.Ymax << std::endl;
276   std::cout << "Zmin = " << BoxBounds.Zmin << std::endl;
277   std::cout << "Zmax = " << BoxBounds.Zmax << std::endl;
278   */
279
280   return BoxBounds;
281 };
282
283 /*****************************************************************/
284
285 inline std::vector<long double> RDF_AnalyticNorm(const std::vector<Particle> Particles,
286 const std::vector<long double> r,const long double dr, const Box BoxBounds)
287 {
288     // Gr averaged over all particles
289     std::vector<long double> Global_Gr(r.size(), 0);
290
291     // Keep track of the usefull (non-empty) shell volumes
292     std::vector<long double> NonEmptyShells(r.size(), 0);
293
294     // maximal radial distance
295     long double MaxDist = r.at(r.size()-1) + dr/2.0;
296
297     // box size
298     long double Lx = BoxBounds.Xmax - BoxBounds.Xmin;
299     long double Ly = BoxBounds.Ymax - BoxBounds.Ymin;
300     long double Lz = BoxBounds.Zmax - BoxBounds.Zmin;
301
302   // Show this to the user
303   std::cout << "The Sample Volume has the following size:" << std::endl;
```

```
304     std::cout << "Lx = " << Lx << std::endl;
305     std::cout << "Ly = " << Ly << std::endl;
306     std::cout << "Lz = " << Lz << std::endl;
307
308     long double MeanDensity = Particles.size() / (Lx * Ly * Lz);
309
310     // use every particle as the center once
311     for(long int CentralP = 0; CentralP < Particles.size(); CentralP++)
312   {
313         // local Gr around the current particle
314         std::vector<long double> Local_Gr(r.size(), 0);
315
316         // look at every other particle at most MaxDist away:
317         for(long int Neighbour = 0; Neighbour < Particles.size(); Neighbour++)
318     {
319             if(CentralP != Neighbour)
320       {
321                 // calc the distance to the neighbour
322                 long double dx = Particles.at(CentralP).XPos - Particles.at(Neighbour).XPos;
323                 long double dy = Particles.at(CentralP).YPos - Particles.at(Neighbour).YPos;
324                 long double dz = Particles.at(CentralP).ZPos - Particles.at(Neighbour).ZPos;
325
326                 long double d = std::sqrt(dx*dx + dy*dy + dz*dz);
327
328         // what bins is the particle in?
329         for(long int RPos = 0; RPos < r.size(); RPos++)
330         {
331           if(std::abs(r.at(RPos) - d) <= dr/2.0)
332           {
333             Local_Gr.at(RPos) += 1;
334           }
335         }
336       }
337     }
338
339     // local box for the shell box intersection volume
340     Box LocalBox = BoxBounds;
341
342     // shift the center of the box cosy
343     LocalBox.Xmin = BoxBounds.Xmin - Particles.at(CentralP).XPos;
344     LocalBox.Xmax = BoxBounds.Xmax - Particles.at(CentralP).XPos;
345     LocalBox.Ymin = BoxBounds.Ymin - Particles.at(CentralP).YPos;
346     LocalBox.Ymax = BoxBounds.Ymax - Particles.at(CentralP).YPos;
347     LocalBox.Zmin = BoxBounds.Zmin - Particles.at(CentralP).ZPos;
348     LocalBox.Zmax = BoxBounds.Zmax - Particles.at(CentralP).ZPos;
349
350     /*print the boundaries to the console
351     std::cout << "The local Box Boundaries are: " << std::endl;
352     std::cout << "Xmin = " << LocalBox.Xmin << std::endl;
353     std::cout << "Xmax = " << LocalBox.Xmax << std::endl;
354     std::cout << "Ymin = " << LocalBox.Ymin << std::endl;
355     std::cout << "Ymax = " << LocalBox.Ymax << std::endl;
356     std::cout << "Zmin = " << LocalBox.Zmin << std::endl;
357     std::cout << "Zmax = " << LocalBox.Zmax << std::endl;
358     */
359
360         // normalize with the shell volume
361         for(long int RIdx = 0; RIdx < r.size(); RIdx++)
362     {
363       // calculate the shell box intersection volume
364             long double SVolume = ShellVolume(r.at(RIdx)-dr/2.0, r.at(RIdx)+dr/2.0, LocalBox);
365
366             if(SVolume > 0.0)
367       {
368                 Local_Gr.at(RIdx) /= SVolume;
369                 NonEmptyShells.at(RIdx) += 1;
370       }
371
372       // normalize by the mean particle density
373       Local_Gr.at(RIdx) /= MeanDensity;
374
375       // save in the global gr for the average over particles
376       Global_Gr.at(RIdx) += Local_Gr.at(RIdx);
```

```
377       }
378
379       if(CentralP % 1000 == 0)
380         std::cout << "Finished Particle " << CentralP << " of " << Particles.size() << std::endl;
381     }
382
383       // final normalization considering the non empty shell volumes
384       for(long int k = 0; k < Global_Gr.size(); k++)
385     {
386           if(NonEmptyShells.at(k) != 0)
387       {
388               Global_Gr.at(k) /= NonEmptyShells.at(k);
389       }
390
391           else
392       {
393         std::cout << "All Shells at R = " << r.at(k) << " are Empty!" << std::endl;
394       }
395     }
396
397     return Global_Gr;
398   };
399
400   /*********************************************************************/
401
402   void SaveData(const std::string SimuName, const std::string FileName, const std::vector<long double> Gr
403           const std::vector<long double> r, const long double dr, const long double ComputeTime,
404           Box BoxBounds)
405   {
406     // box size
407       long double Lx = BoxBounds.Xmax - BoxBounds.Xmin;
408       long double Ly = BoxBounds.Ymax - BoxBounds.Ymin;
409       long double Lz = BoxBounds.Zmax - BoxBounds.Zmin;
410
411     long double RmaxNatural = std::sqrt(Lx*Lx + Ly*Ly + Lz*Lz)/2.0;
412
413     std::ofstream DataFile;
414     DataFile.open(SimuName + ".txt", std::ios::out);
415     DataFile.precision(25);
416
417     DataFile << "Results for the radial distribution function computation." << std::endl;
418     DataFile << "Particle File Name: " << FileName << std::endl;
419     DataFile << "Computation time for g(r): " << ComputeTime << " nanoseconds." << std::endl;
420     DataFile << "Radial Bin Width dr: " << dr << std::endl;
421     DataFile << "Number of radial Bins: " << r.size() << std::endl;
422     DataFile << "Number of Particles in the sample: " << BoxBounds.PartNum << std::endl;
423     DataFile << "Average Particle Number Density: " << BoxBounds.PartNum / (Lx * Ly * Lz) << std::endl;
424     DataFile << "Maximal useful radial distance: " << RmaxNatural << std::endl;
425     DataFile << "Box Boundary Xmin: " << BoxBounds.Xmin << std::endl;
426     DataFile << "Box Boundary Xmax: " << BoxBounds.Xmax << std::endl;
427     DataFile << "Box Boundary Ymin: " << BoxBounds.Ymin << std::endl;
428     DataFile << "Box Boundary Ymax: " << BoxBounds.Ymax << std::endl;
429     DataFile << "Box Boundary Zmin: " << BoxBounds.Zmin << std::endl;
430     DataFile << "Box Boundary Zmax: " << BoxBounds.Zmax << std::endl;
431
432     DataFile << "Box Boundary X-length: " << Lx << std::endl;
433     DataFile << "Box Boundary Y-length: " << Ly << std::endl;
434     DataFile << "Box Boundary Z-length: " << Lz << std::endl;
435
436     DataFile << "Box Volume: " << Lx * Ly * Lz << std::endl;
437
438     DataFile << "# r \t g(r)" << std::endl;
439
440     for(long int l = 0; l < r.size(); l++)
441     {
442       DataFile << r.at(l) << "\t";
443       DataFile << Gr.at(l) << std::endl;
444     }
445
446     DataFile.close();
447   };
448
449   /*********************************************************************/
```

```
450
451 int main(void)
452 {
453   // Dashboard
454   std::string SimuName = "";     // Name of the simulation
455   std::string FileName = "";     // Name of the simulation
456   long int Bins = 1;                      // number of sample calculations
457   long double dr = 1;                    // bin size
458
459   // User Input
460   std::cout << "Please enter the following information:" << std::endl;
461   std::cout << "Name for the simulation: ";
462   std::cin >> SimuName;
463   std::cout << "Particle File Name: ";
464   std::cin >> FileName;
465   std::cout << "Bin size (dr): ";
466   std::cin >> dr;
467   std::cout << "Starting g(r) computation." << std::endl;
468
469   // output precision
470   std::cout.precision(20);
471
472   // read in the raw particle coordinates
473   std::vector<Particle> Particles = readPartCoor(FileName);
474
475   // benchmark the g(r) calculation
476   auto start = std::chrono::steady_clock::now();
477
478   // find the Box Boundaries
479   Box BoxBounds = FindBoxBounds(Particles);
480
481   // box size
482     long double Lx = BoxBounds.Xmax - BoxBounds.Xmin;
483     long double Ly = BoxBounds.Ymax - BoxBounds.Ymin;
484     long double Lz = BoxBounds.Zmax - BoxBounds.Zmin;
485
486   // maximal useful radial distance
487   long double RMAX = std::sqrt(Lx*Lx + Ly*Ly + Lz*Lz)/2.0;
488
489   // show the maximal usefull radius to the user
490   std::cout << "Maximal usefull radius: " << RMAX << std::endl;
491
492   // Number of Bins
493   Bins = static_cast<long int>(RMAX / dr);
494
495   // the radial position and g(r) vector
496   std::vector<long double> r(Bins, 0);
497   std::vector<long double> Gr(Bins, 0);
498
499   // create the radial distance vector
500   for(long int k = 0; k < Bins; k++)
501   {
502     r.at(k) = k * RMAX/Bins;
503   }
504
505   // compute the radial distribution function
506   Gr = RDF_AnalyticNorm(Particles, r, dr, BoxBounds);
507
508   auto end = std::chrono::steady_clock::now();
509   auto diff = end - start;
510   double ComputeTime = std::chrono::duration <double, std::nano> (diff).count();
511
512   // information for the user
513   std::cout << "Time for the g(r) computation: ";
514     std::cout << ComputeTime;
515     std::cout << " nanoseconds" << std::endl;
516
517   // save all the data into a file
518   SaveData(SimuName, FileName, Gr, r, dr, ComputeTime, BoxBounds);
519
520   std::cout << "Program Finished!" << std::endl;
521 };
```