Projecting Spherical Objects onto a 2-Dimensional plane within 3-Dimensional Space

Contents

1 Scenario

1.1 Introduction

Consider 3-Dimensional space containing objects we will define as spheres of any size. Within this space, we will have a point of observation, called the camera. The camera itself is a singular point, however at a fixed distance from the camera, in a direction given by a unit vector $\hat{\mathbf{r}}$, is a 2-Dimensional plane representing the screen of the computer running the simulation. The location of the camera and each object will be given as a vector:

$$P = \begin{pmatrix} x_P \\ y_P \\ z_P \end{pmatrix}, C = \begin{pmatrix} x_C \\ y_C \\ z_C \end{pmatrix}, \hat{\mathbf{r}} = \begin{pmatrix} x_r \\ y_r \\ z_r \end{pmatrix} : |\hat{\mathbf{r}}| = 1$$

In Figure 1, the line perpendicular to the plane through C is parallel to $\hat{\mathbf{r}}$, and the distance from the camera to the screen will be defined as S_d.

1.2 Projecting spheres onto a plane

Projecting a spherical object onto a 2-Dimensional plane, if the plane is not perpendicular to the vector between the camera and the object, will produce a conic section, ie, an ellipse. This is essentially the definition of a conic section, the intersection of a 2-Dimensional plane and a cone extending from the camera at C to the object at P, made up of the tangents that join them. See figure 1 for a visualisation. Such an ellipse will have certain properties, such always being aligned so that the line through the major axis passes through the centre of the screen, meaning there are three unknowns:

- The length of the minor axis
- The length of the major axis
- The x and y position of the object *on the screen*

In Figure 1, the plane shown by the grid represents the screen, the red sphere and the blue dot at C correspond to the object in space and the camera, also in space, respectively. It is clear how the projection of the sphere onto the plane would produce a conic section, and it is worth noting that if an observer were to observe this ellipse fom C, the ellipse would appear to be a circle. In the example shown in the figure, the object is so far from the centre of the screen (marked by S_C) relative to the distance of the camera from the screen that the projection would place the ellipse well outside the physical bounds of the screen, and in reality the eccentricity of the projections would be very low.

Figure 2 shows what the projection would look like on the screen, as it would appear looking perpendicular towards the plane from underneath it. The perpendicular lines are shown for reference.

2 Finding the Shape of the Ellipse Using Geometry

Calculating the lengths of the major and minor axis of the projection is fairly simple, and involves a bit of trigonometry.

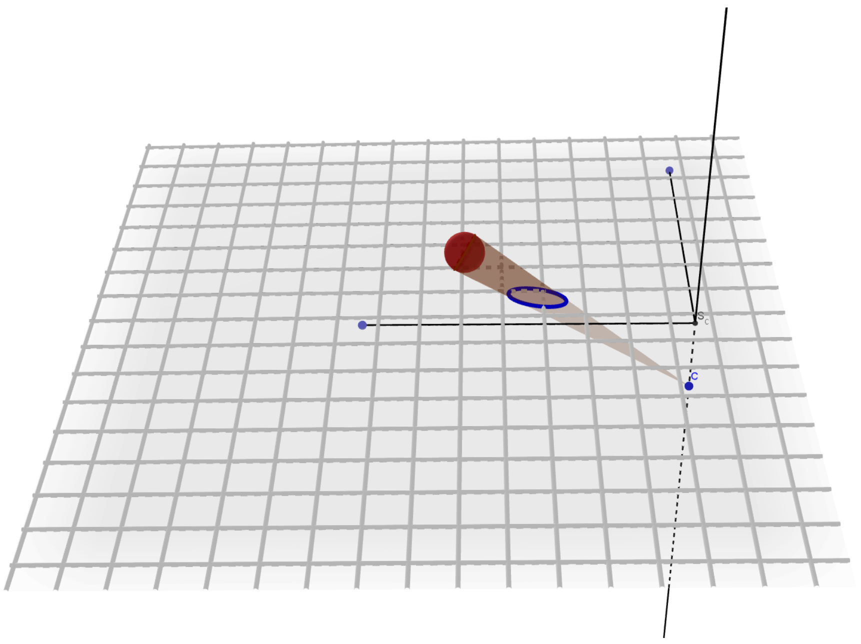


Figure 1: Projection of a sphere onto a 2-Dimensional plane.

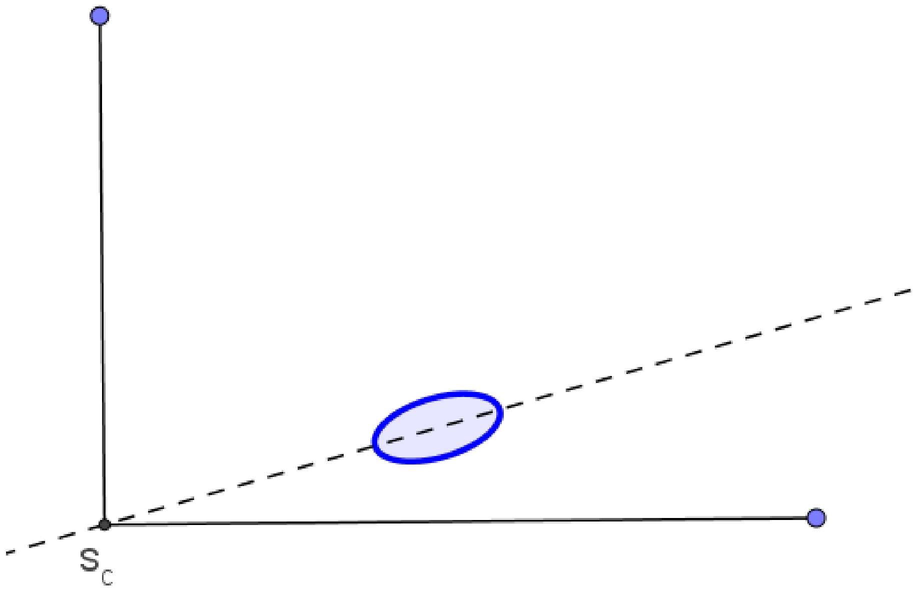


Figure 2: View of the ellipse looking onto the plane (from underneath)

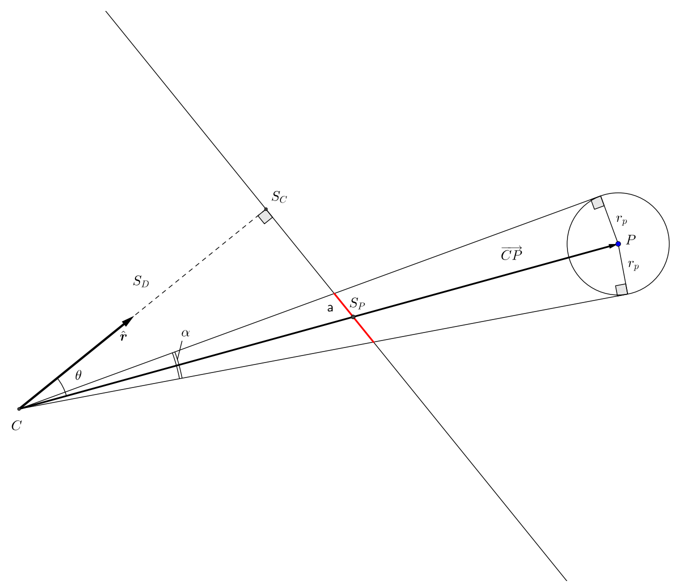


Figure 3: Diagram of the projection of a circle onto a line, in this case, the red line a is equivalent to the major axis in the 3-Dimensional case.

2.1 Finding the Length of the Major Axis

As the major axis is aligned with the origin - the centre of the screen - there is no use in breaking the problem into x and y components, it is better to look at a cross sectional diagram parrallel with the major axis, as shown in Figure 3. In the diagram, the displacement vector $\overrightarrow{CP}$, the vector $\hat{\mathbf{r}}$, the length of CS_C and the radius of the particle r_p are the only known variables. The angle θ is going to be the angle between $\hat{\mathbf{r}}$ and $\overrightarrow{CP}$. Thus, to find theta, we use the dot product of the two vectors:

$$\hat{\mathbf{r}} \cdot \overrightarrow{CP} = |\hat{\mathbf{r}}||\overrightarrow{CP}| \cos\theta$$

$$\cos\theta = \frac{\hat{\mathbf{r}} \cdot \overrightarrow{CP}}{|\hat{\mathbf{r}}||\overrightarrow{CP}|}$$

$$\cos\theta = \frac{\hat{\mathbf{r}} \cdot \overrightarrow{CP}}{|\overrightarrow{CP}|} \tag{1}$$

Since $\hat{\mathbf{r}}$ is defined to have a length of 1. Then,

$$\cos\theta = \frac{S_D}{|CS_P|}$$

$$|CS_P| = \frac{S_D}{\cos\theta} \tag{2}$$

Thus we now know both the angle of the particle from the centre of the screen and the distance from the origin of the camera to the projection. This will allow us to also calculate $|S_C S_P|$, the distance from the centre of screen to the centre of the projection, although isn't needed for this part of the problem, but will be used later on.

$$\left|\overrightarrow{S_C S_P}\right| = \sqrt{|CS_P|^2 - S_D^2}$$

Now, to find the length of a, we will also need the angle α, which is the angle between the two tangents joining the circle at P to the camera at C. This is found using the right angled triangles on either side of $\overrightarrow{CP}$.

$$\sin\left(\frac{\alpha}{2}\right) = \frac{r_p}{\left|\overrightarrow{CP}\right|}, \ \cos\left(\frac{\alpha}{2}\right) = \frac{\sqrt{\left|\overrightarrow{CP}\right|^2 - r_p^2}}{\left|\overrightarrow{CP}\right|}$$

$$\begin{aligned}\cos\alpha &= \cos^2\frac{\alpha}{2} - \sin^2\frac{\alpha}{2} \\ &= \frac{\left|\overrightarrow{CP}\right|^2 - r_p^2}{\left|\overrightarrow{CP}\right|^2} - \frac{r_p^2}{\left|\overrightarrow{CP}\right|^2} \\ &= \frac{\left|\overrightarrow{CP}\right|^2 - 2r_p^2}{\left|\overrightarrow{CP}\right|^2}\end{aligned}$$

$$\begin{aligned}\sin\alpha &= 2\sin\frac{\alpha}{2}\cos\frac{\alpha}{2} \\ &= 2r_p\frac{\sqrt{\left|\overrightarrow{CP}\right|^2 - r_p^2}}{\left|\overrightarrow{CP}\right|^2}\end{aligned}$$

$$\therefore \cos\alpha = \frac{\left|\overrightarrow{CP}\right|^2 - 2r_p^2}{\left|\overrightarrow{CP}\right|^2} \tag{3}$$

$$\therefore \sin\alpha = 2r_p\frac{\sqrt{\left|\overrightarrow{CP}\right|^2 - r_p^2}}{\left|\overrightarrow{CP}\right|^2} \tag{4}$$

Now the length of a can be calculated from the difference in the distance from S_C to the closest tangent line and the furthest tangent line. In the following working, we will call those distances d_1 and d_2 for the shortest distance and longest distance respectively, such that $d_2 - d_1 = a$.

$$d_2 = \tan\left(\theta + \frac{\alpha}{2}\right)S_D$$

$$d_1 = \tan\left(\theta - \frac{\alpha}{2}\right)S_D$$

$$\begin{aligned}a = d_2 - d_1 &= S_D\left(\tan\left(\theta + \frac{\alpha}{2}\right) - \tan\left(\theta - \frac{\alpha}{2}\right)\right) \\ &= S_D\left(\frac{\sin\left(\theta + \frac{\alpha}{2}\right)}{\cos\left(\theta + \frac{\alpha}{2}\right)} - \frac{\sin\left(\theta - \frac{\alpha}{2}\right)}{\cos\left(\theta - \frac{\alpha}{2}\right)}\right) \\ &= S_D\frac{\sin\alpha}{\cos\left(\theta + \frac{\alpha}{2}\right)\cos\left(\theta - \frac{\alpha}{2}\right)} \\ &= S_D\frac{\sin\alpha}{\cos^2\theta\cos^2\frac{\alpha}{2} - \sin^2\theta\sin^2\frac{\alpha}{2}}\end{aligned}$$

All terms of which are known. And so we have,

$$a = S_D\frac{\sin\alpha}{\cos^2\theta\cos^2\frac{\alpha}{2} - \sin^2\theta\sin^2\frac{\alpha}{2}} \tag{5}$$

There are other ways to find this length, some of them are probably easier. Expanding this to be in terms of the known variables gives:

$$a = S_D \left(\frac{2r_p \left(\left|\overrightarrow{CP}\right|^2 - r_p^2 \right)^{(3/2)}}{\left|\overrightarrow{CP}\right|^4 \cos^2\theta} - \frac{r^2 \sin^2\theta}{\left|\overrightarrow{CP}\right|^2} \right)$$

Or as code:

```
a = SD * (2 * rp * (CP**2 - rp**2)**(3/2) /
    (CP**4 * cos(theta)**2) - sin(theta)**2 * rp**2 / CP**2)
```

2.2 Finding the Length of the Minor Axis

Finding the length of the minor axis, b, is much simple than the major axis. The minor axis is orthogonal to the line joining the camera and the object, so methods like the ones used in the previous section won't be necessary.

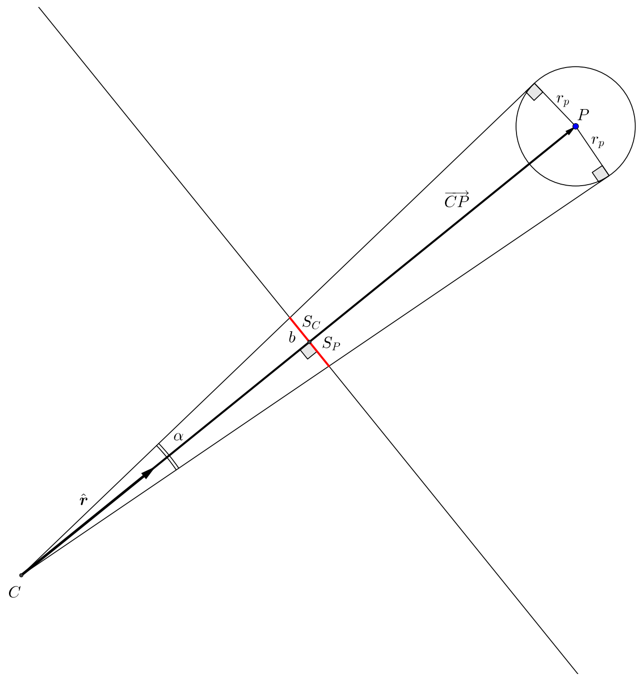


Figure 4: Diagram of the projection of a circle onto a plane where the red line represents the minor axis of the ellipse.

From Figure 4:

$$\tan\frac{\alpha}{2} = \frac{r_p}{\sqrt{\left|\overrightarrow{CP}\right|^2 - r_p^2}}$$

$$b = 2S_D \tan\frac{\alpha}{2}$$

And so we have:

$$b = S_D \frac{2r_p}{\sqrt{\left|\overrightarrow{CP}\right|^2 - r_p^2}} \tag{6}$$

As code:

```
b = SD * 2 * rp / (CP**2 - rp**2)**(1/2)
```

Thus we have the major and minor axis of the ellipse. Now two of the three unknowns have been calculated, all that remains is to find the position of the ellipse on the screen.

3 Finding the Position of the Ellipse on the Screen

3.1 Vector Geometry

To find the x and y component of the position on the screen, vector geometry is needed. The most important function we will use is the projection of one vector onto another. Defined as:

$$\text{Proj}_{\mathbf{b}}\mathbf{a} = \frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{b}|^2}\mathbf{b}$$

The result gives you the projection of **a** onto **b**. When just distances are needed, we divide by the unit vector of **b**, ie $\frac{\mathbf{b}}{|\mathbf{b}|}$ to get

$$|\text{Proj}_{\mathbf{b}}\mathbf{a}| = \frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{b}|}$$

Vector projection can infact be thought of to be defined as multiplying the above scalar value by the unit vector of the vector being projected onto.

To get the x and y components, consider two vectors $y\prime$ and $x\prime$, both parallel to the plane of the screen but:

- $x\prime$ is parallel to the x-axis of the screen, meaning it is orthogonal to both the vector $\overrightarrow{CS_C}$ and the global y-axis.
- $y\prime$ is parallel to the y-axis of the screen, meaning it is orthogonal to both the vector $\overrightarrow{CS_C}$ and $x\prime$.

The lengths of $x\prime$ and $y\prime$ are irrelevant, only their directions are important. Thus, from their definitions,

$$\begin{aligned}
x\prime &= \overrightarrow{CS_C} \times \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \\
&= S_D \begin{pmatrix} x_r \\ y_r \\ z_r \end{pmatrix} \times \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \\
&= \begin{pmatrix} -S_D z_r \\ 0 \\ S_D x_r \end{pmatrix}
\end{aligned}$$

$$\therefore x\prime = \begin{pmatrix} -z_r \\ 0 \\ x_r \end{pmatrix} \tag{7}$$

And,

$$\begin{aligned}
y\prime &= \overrightarrow{CS_C} \times x\prime \\
&= S_D \begin{pmatrix} x_r \\ y_r \\ z_r \end{pmatrix} \times \begin{pmatrix} -z_r \\ 0 \\ x_r \end{pmatrix} \\
&= S_D \begin{pmatrix} x_r y_r \\ -\left({x_r}^2 + {z_r}^2\right) \\ z_r y_r \end{pmatrix}
\end{aligned}$$

$$\therefore y\prime = \begin{pmatrix} x_r y_r \\ -\left({x_r}^2 + {z_r}^2\right) \\ z_r y_r \end{pmatrix} \tag{8}$$

Now that we have the x and y axes of the screen, the x and y components of the position of particle are the respective lengths of the projection of $\overrightarrow{S_C S_P}$ onto $x\prime$ and $y\prime$. We recall $\overrightarrow{S_C S_P}$ to be:

$$\begin{aligned}
\overrightarrow{S_C S_P} &= S_P - S_C \\
&= CS_P - CS_C \\
&= \frac{\left|\overrightarrow{CP}\right|}{\left|\overrightarrow{CS_P}\right|}\overrightarrow{CP} - S_D \hat{\mathbf{r}} \qquad \left|\overrightarrow{CS_P}\right| \text{ is defined in equation (2).}
\end{aligned}$$

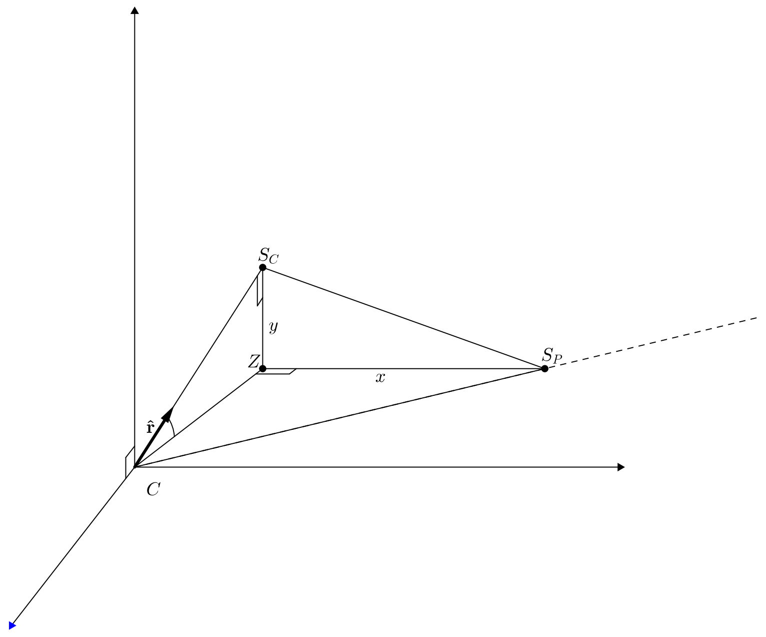


Figure 5: The triangle ZS_CS_P lies on the plane of the screen, perpendicular to $\hat{\mathbf{r}}$. The point P will lie on any point on the ray through $\overrightarrow{CS_P}$ represented by the dotted line.

Therefore,

$$x = \frac{\overrightarrow{S_C S_P} \cdot x\prime}{|x\prime|}$$

$$y = \frac{\overrightarrow{S_C S_P} \cdot y\prime}{|y\prime|}$$

4 Rotation of the x and y axes about each other

In order to rotate the screen, two methods can be used. Either the vector $\hat{\mathbf{r}}$ be rotated and the x, y axes $x\prime, y\prime$ be recalculated for each draw, or the x and y axes be rotated and $\hat{\mathbf{r}}$ be recalculated. The latter is preferable as the first method restricts the orientation of the screen such that $y\prime$ always passes through the y-axis. If the axes can be rotated independently of each other then the user is able to orientate the screen in any direction, and rotation of the screen about it's z-axis becomes possible and feasible. However, to rotate a vector about another vector is no trivial task.

4.1 Rotation of a vector about another vector

Consider the vectors;

- a: The axis of rotation
- v: The initial vector to be rotated
- $v\prime$: The rotation of v about a

For our situation, a and v would be perpendicular, making the problem slightly easier. Although, it is useful to see the general solution.

The vector v can be broken down into $v_{\|a} + v_{\perp a}$, the component along a and the component orthogonal to a. $v_{\|a}$ is simply the projection of v onto a,

$$v_{\|a} = \frac{a \cdot v}{|v|^2} v$$

And the orthogonal part is then just $v - v_{\|a}$

$$v_{\perp a} = v - \frac{a \cdot v}{|v|^2} v$$

Now, consider the plane orthogonal to a, a vector on this plane can hence be made by the linear combination of $x_1 v_{\perp a} + x_2 a \times v$, both vector components of which are orthogonal to a. Thus, to get a rotation by a specific angle θ, let,

$$x_1 = \frac{\cos\theta}{|v_{\perp a}|} \text{ and } x_2 = \frac{\sin\theta}{|a \times v|}$$

Such that the component of the resulting vector on this plane is

$$v\prime_{\perp a} = |v_{\perp a}| \left(\frac{\cos\theta v_{\perp a}}{|v_{\perp a}|} + \frac{\sin\theta(a \times v)}{|a \times v|} \right)$$

The final result can be given by $v\prime = v\prime_{\perp a} + v_{\parallel a}$, therefore;

$$v\prime = |v_{\perp a}| \left(\frac{\cos\theta v_{\perp a}}{|v_{\perp a}|} + \frac{\sin\theta(a \times v)}{|a \times v|} \right) + \frac{a \cdot v}{|v|^2} v \tag{9}$$

As stated, in our case $v_{\parallel a} = 0$, and so the last term in the equation drops out. Thus to rotate $x\prime$ about $y\prime$ we substitute $a = y\prime$ and $v = x\prime$, and vice versa.

4.2 Re-orientating $\hat{\mathbf{r}}$ to the new screen

Now, we are able to rotate the vectors about each other, although we haven't changed $\hat{\mathbf{r}}$ yet. By definition, $|\hat{\mathbf{r}}| = 1$, but is parallel to the normal through $x\prime$ and $y\prime$ (although in the code the length isn't always assumed to be 1 as a safety net). Additionally, S_D is fixed and does not change, so we are easily able to find the new direction of $\hat{\mathbf{r}}$ and thus the new position of the screen.

Expanding on their definitions,

$$\hat{\mathbf{r}} \parallel x\prime \times y\prime$$

$$\hat{\mathbf{r}} = \frac{x\prime \times y\prime}{|x\prime \times y\prime|}$$

Note that this requires $x\prime$ and $y\prime$ to be in the direction of the right and the top of the screen respectively.

5 Simulation Using the Leapfrog Method

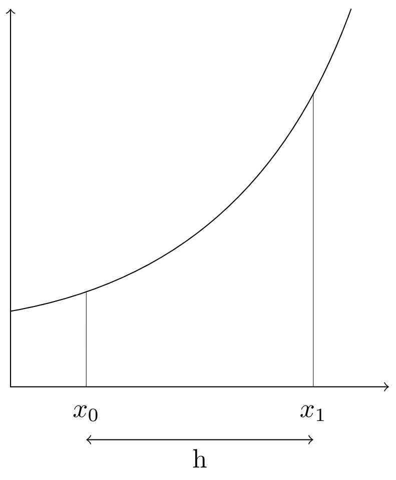
The most obvious way to simulate an n-body universe is to calculate the acceleration of each particle at a certain time, apply that to the velocity of the particle using a specified time step, then apply the new velocity to the position of the particle.

$$\begin{aligned} \mathbf{a}_i &= \mathbf{F}_i/m \\ \mathbf{v}_{i+1} &= \mathbf{a}_i \Delta t \\ \mathbf{p}_{i+1} &= \mathbf{v}_{i+1} \Delta t \\ \text{or } \mathbf{p}_{i+1} &= \mathbf{v}_i \Delta t \end{aligned}$$

However this method has a significant error, which gets much larger as the time step increases. The leapfrog approach is a more accurate method, and differs from the previous method in that it uses the velocity at one half step ahead of the current time.

$$\begin{aligned}\mathbf{v}_{i+1/2} &= \mathbf{v}_{i-1/2} + \mathbf{a}_i \Delta t \\ \mathbf{p}_{i+1} &= \mathbf{p}_i + \mathbf{v}_{i+1/2} \Delta t\end{aligned}$$

In a parabolic position time scenario, the approximation provides an exact result, with 100% accuracy. In an n-body simulation this is essentially never the case but it is still a good approximation, as shown in the figure.



5.1 Starting the simulation

The main difficulty with this method is starting the process, where normally the position and velocities are given at a single instant, not a half step difference. To start the simulation, we must offset the velocity using the normal methods shown at the start of the section. ie, if v_0 and x_0 are given, then;

$$v_{1/2} = v_0 + a_0 \frac{\Delta t}{2}$$

This makes it difficult to change the time step in the simulation, as the offset at the start is based on the initial time step. To do so, we have to first remove the offset between the position and velocity, one way to do this is to first subtract $a_i \frac{\Delta t_0}{2}$ from, then adding $a_i \frac{\Delta t_1}{2}$ to the current v, where Δt_0 and Δt_1 are the previous and new time steps respectively.

5.2 Using matrix mathematics to perform faster calculations

Generally using a static n-dimensional array as a matrix will be faster than using a dynamic n-dimensional list. This is especially the case with Numpy's arrays compared to the native python lists.

To make calculations we require the following arrays to be known at all times:

- $\mathbf{P}_N \in \mathbb{R}^{N\times 3} \rightarrow$ Array of position vectors.
- $\mathbf{R} \in \mathbb{R}^{\mathbb{N}} \rightarrow$ Array of radii of particles.
- $\mathbf{M} \in \mathbb{R}^{\mathbb{N}} \rightarrow$ Array of masses of particles.

Where N is the number of particles. For a calculation such as standard gravitational attraction, we would perform the following particle for each particle. Here the individual particle we are simulating for is as position $\mathbf{P}$, with mass M and radius R.

Find the distance to each other particle:

$$\mathbf{D}_N = \mathbf{P}_N - \{\mathbf{P}\}$$

(The braces indicate the subtraction is for each element in the matrix $\mathbf{P}_N$)

The absolute distances are given by:

$$\mathbf{D} = |\mathbf{D}_N| = \{|\mathbf{x}| \forall \mathbf{x} \in \mathbf{D}_N\}$$

For the force:

$$\mathbf{F}_N = GM \odot \mathbf{M} \odot (\mathbf{D}_N) \oslash (|\mathbf{D}|^3)$$

Where $\odot, \oslash$ are element wise multiply and divide respectively. Also note that cubic power at the right is element wise aswell.

Now we sum the forces to get the net force:

$$\mathbf{F} = \sum_{n=0}^{N} \mathbf{F}_{N_n}$$

The acceleration is simply derived from $\mathbf{F} = m\mathbf{a}$, however it would be fine to simply remove the mass of the particle at the start in this case to directly calculate acceleration instead of force.

We also want to check for collisions. Lets consider a vector **A** containing the 'altitude' of each particle, ie the surface-surface distance.

$$\begin{aligned} \mathbf{A} &= \mathbf{D} - \mathbf{R} - \{R\} \\ \mathbf{B}_{\text{collision}} &= \{\mathbf{A} < 0\} \end{aligned}$$

Where **B** is a boolean vector where a value of True indicates that our current particle is colliding with the particle at that index.

6 Drawing Crescents on Bodies

To draw a crescent on a body, we first imagine drawing a disc that represents the boundary between the surface of the body illuminated and the surface of the body in darkness. The radius of this disk is given by:

$$r = R_2\sqrt{2\left(1 - \cos 2\theta\right)}$$

where θ is given by;

$$\cos\theta = \frac{R_1 - R_2}{d}$$

where R_1 and R_2 are the radii of the parent body (eg. the sun) and the child body (eg. the earth) respectively. d is the distance between the centre of each body, as shown in Figure 6.

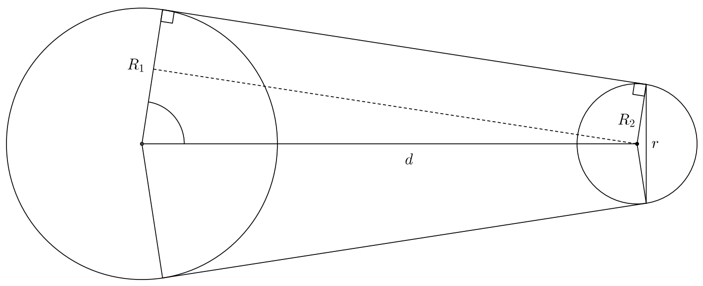


Figure 6: A cross section representing the radius of the boundary disc between illuminated and non illuminated surface, the disc is the line of length $2r$, such that r is the radius of that disc.