

# Creating Lightcones in HACC

Joe Hollowed, Cosmological Physics and Advanced Computing Group

*Last edited May 23, 2018*

This document outlines our current approach for generating lightcones from HACC simulation products. In § 1, we derive the condition which defines all spacetime events that can be seen by an observer at any given time (we parameterize the surface of the observer’s past lightcone). In § 2, we describe how we apply this condition to find approximate lightcone-crossing times of objects whose positions and velocities are known only at discrete time snapshots. Higher order effects and corrections are discussed in § 3. § 4 describes the details of our implementation of this problem in HACC, and finally, a summary is given in § 6. The code which implements the methods described here can be found in the HACC repository at <https://svn.alcf.anl.gov/repos/DarkUniverse/hacc/> (see § 4 for details). § 1 - § 2.1 of this document’s content was originally based on in-line documentation in the above mentioned code, written by Hal Finkel.

## 1 Parameterizing the Lightcone Surface

At any given time  $t$ , an observer will see only what is on the surface of her past-lightcone. Any event located inside the lightcone surface has been seen in the past, and any event outside will be seen in the future, in general. Hence, our goal is to parameterize this surface in terms of quantities that are available to us as simulation outputs. Applying this parameterization to simulation datasets will allow us to locate all particles on an observer’s past-lightcone at any given time, yielding the observable universe.

Let us first consider the problem in a static Euclidean space. Recall that the space-time separation between an observation, occurring at  $(t, r, \theta, \varphi)$ , and some event at  $(t + dt, r + dr, \theta + d\theta, \varphi + d\varphi)$ , by special relativity, is given by the Minkowski metric:

$$ds^2 = -c^2 dt^2 + dr^2 + r^2 d\Omega^2, \quad (1)$$

where  $d\Omega^2$  includes the change of variables from Cartesian to spherical coordinates as

$$d\Omega^2 \equiv d\theta^2 + \sin^2 \theta d\varphi^2. \quad (2)$$

An event being located on the lightcone surface is defined by the fact that the line joining it to the observer is a null geodesic, where  $ds^2 = 0$ . If we impose the constraint that the observer is located at the spatial origin (which is always done in practice), then this null geodesic is constant in  $\theta$  and  $\varphi$ , and thus  $d\Omega^2 = 0$ . Objects located on the lightcone therefore satisfy the condition

$$0 = -c^2 dt^2 + dr^2 \quad (3)$$

which can be rearranged to express the distance to the event in terms of the light travel time as

$$c^2 dt^2 = dr^2 \quad \rightarrow \quad \int_{t_{\text{event}}}^{t_{\text{obs}}} c dt = c(t_{\text{obs}} - t_{\text{event}}) = \|\mathbf{r}_{\text{event}}\|, \quad (4)$$

where  $t_{\text{obs}}$  and  $t_{\text{event}}$  are the time coordinates of the observer and any observed event, respectively.  $\|\mathbf{r}_{\text{event}}\|$  represents the magnitude of the spatial distance to the observed event.

So, we have parameterized the lightcone surface by the rightmost equality in Eq.(4) for a static universe. However, the universe *is not* static, and so light travel time between two events is *not*  $\|\mathbf{r}_{\text{event}}\|/c$ , but should effectively be increased by the scale factor  $a(t)$ . Obtaining the correct condition for finding an event on the lightcone, then, must require repeating the simple procedure in Eq.(1-4) using the more general Robertson-Walker metric:

$$ds^2 = -c^2 dt^2 + a(t)^2 [dr^2 + S_z(r)^2 d\Omega^2]. \quad (5)$$

Here,  $S_z(r)$  defines the curvature of space, which can be ignored since  $d\Omega^2 = 0$ , as per our assumptions stated above (the curvature information remains encoded in the form of  $a(t)$ ). In that case, a null geodesic spacetime separation under this metric (again set  $ds^2 = 0$  and solve) gives

$$\int_{t_{\text{event}}}^{t_{\text{obs}}} c \frac{dt}{a(t)} = \|\mathbf{r}_{\text{event}}\|. \quad (6)$$

It will also be handy to have this condition in another form; we can use the Hubble parameter  $H$  to cast the variable of integration to the scale factor, as a proxy for time, if we note the following substitutions:

$$dt = \frac{dt}{da} da = \frac{da}{\dot{a}} \quad \text{and} \quad H = \frac{\dot{a}}{a}. \quad (7)$$

Combining Eq.(6-7) results in

$$\int_{a_{\text{event}}}^1 c \frac{da}{a^2 H} = \|\mathbf{r}_{\text{event}}\|. \quad (8)$$

Eq.(6, 8) is our sought after parameterization<sup>a</sup>; all events whose spacetime coordinates satisfy this condition are found on the lightcone surface and are visible to the observer, at the spatial origin, at time  $t_{\text{obs}}$ .

## 2 Crossing the Lightcone

Given pristine knowledge of particle positions resultant from cosmological simulations, Eq.(8) appears straightforward to apply: for any time  $t$ , find the associated scale factor  $a(t)$ , and evaluate the integral on the left hand side of Eq.(8) to obtain some value  $R$ . Then, check all particles in the simulation for the condition that  $\|\mathbf{r}_i\| = R$ , where the subscript  $i$  denotes the  $i$ th particle. If the condition is satisfied for any particle  $i$ , then

---

<sup>a</sup>Eq.(8) is actually the parameterization of a sphere in three-dimensional space, whose radius is dependent on the parameter  $a$  (A more readable form of a general expanding/contracting sphere is  $\|\mathbf{r}\| = a(t)\sqrt{x^2 + y^2 + z^2}$ ). Actually visualizing such a sphere on a static figure requires projecting the object into two spatial dimensions and allowing time to occupy the third axis— this results visually in a cone, hence the name "lightcone". Indeed the lightcone surface is truly a spherical object.

that particle crossed the lightcone and is seen by the observer at time  $t$ . Repeat this for all times  $t$  to fill the lightcone.

Unfortunately, simulation outputs do not include “all times  $t$ ”. Since we only have of order 100 time snapshot outputs from  $a \approx 0$  to  $a = 1$ , the naive approach described in the previous paragraph is not applicable. Consider a particle  $i$  that crosses the lightcone surface (is seen) at time  $t_{\text{event}}$ <sup>b</sup>. This event is only captured in the simulation outputs by the fact that particle  $i$  is inside the observer’s past lightcone at some snapshot time  $t_j < t_{\text{event}}$ , and outside of the lightcone at the next snapshot time  $t_{j+1} > t_{\text{event}}$ . Ultimately, the problem that we need to solve in order to construct the lightcone is finding  $t_{\text{event}}$  for all particles. This has traditionally been done in the HACC codebase using particle extrapolation from time  $t_j$ , which is discussed in § 2.1. More recently implemented is an improved approach which rather interpolates between snapshot times  $t_j$  and  $t_{j+1}$ , described in § 2.3.

## 2.1 Particle Extrapolation

Since we would ultimately like to solve Eq.(8) for  $a_{\text{event}}$  (equivalently Eq.(6) for  $t_{\text{event}}$ ), we must first write the relevant quantities in terms of simulation data products. Let us consider a single simulation timestep for the remainder of this section, which begins at snapshot  $j$  and ends at snapshot  $j + 1$ <sup>c</sup>. The time it spans is  $t_{j+1} - t_j$ , and the scale factor evolution is  $a_{j+1} - a_j$ .

Each particle in snapshot  $j$  has six quantities that are relevant to our purposes; three comoving Cartesian coordinates and corresponding velocities

$$\mathbf{r}_j = \begin{bmatrix} x_j \\ y_j \\ z_j \end{bmatrix}, \quad \mathbf{v}_j = \dot{\mathbf{r}}_j. \quad (9)$$

We now define  $\|\mathbf{r}_{\text{event}}\|$  by linear extrapolation from the particle’s position and velocity at  $t_j$ :

$$\begin{aligned} \|\mathbf{r}\| &= \|\mathbf{r}_j + \mathbf{v}_j \delta t\| \\ &= [(x_j + \dot{x}_j \delta t)^2 + (y_j + \dot{y}_j \delta t)^2 + (z_j + \dot{z}_j \delta t)^2]^{1/2} \end{aligned} \quad (10)$$

where we have introduced the quantity  $\delta t$  as  $\delta t = t_{\text{event}} - t_j$ . Let us also obtain an expression for  $a_{\text{event}}$  in Eq.(8), the scale factor at the time  $t_j + \delta t$  to first order:

$$a_{\text{event}} = a(t_j + \delta t) \approx a_j + \dot{a} \delta t. \quad (11)$$

Finally, let’s describe  $a_{j+1}$  in terms of  $a_j$  and the timestep width  $\tau = t_{j+1} - t_j$ , as

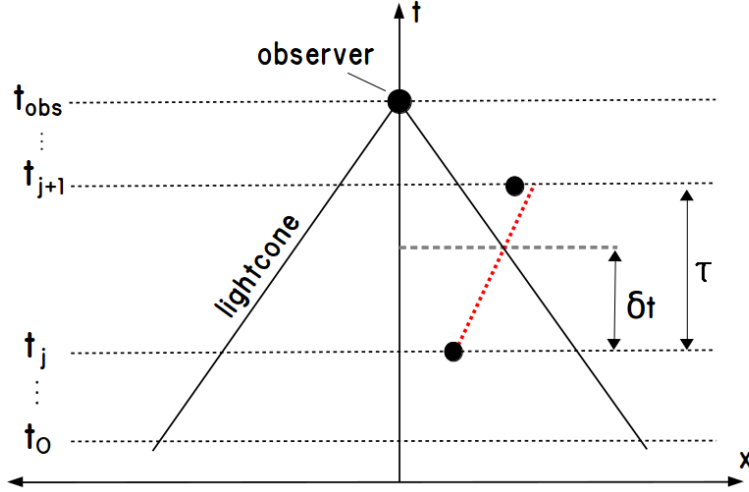
$$a_{j+1} \approx a_j + \dot{a} \tau. \quad (12)$$

---

<sup>b</sup>To say that the particle “is seen” at time  $t_{\text{event}}$  is shorthand for the more accurate statement: “photons emitted from the particle’s position at time  $t_{\text{event}}$  are seen by the observer at time  $t_{\text{obs}}$ ”. The observer of course observes the entire sky at a single time  $t_{\text{obs}}$ .

<sup>c</sup>This notation is not meant to imply that lightcone construction is performed on *all* simulation steps.  $j$  and  $j + 1$  are only meant to refer to adjacent simulation snapshots in whatever set of outputs is in use.

Refer to Figure 1 for a visual summary of the important quantities discussed above. Again, our ultimate goal is to find  $t_{\text{event}}$  for all particles. The remainder of this section will do this by solving for the quantity  $\delta t$ .



**Figure 1:** A schematic of linear particle position extrapolation. This figure focuses on an arbitrary timestep bounded by snapshots  $j$  and  $j + 1$ , where the vertical axis is time and the horizontal axis is space. Time increases upward. The particle (black disc) is found inside the lightcone at time  $t_j$ , and outside the lightcone at the time  $t_{j+1}$  of the following snapshot. It is estimated to cross the lightcone, by extrapolation (red dashed line), at time  $t_j + \delta t$ . Note that the extrapolation does not in general estimate the correct position of the particle at time  $t_{j+1}$ .

At this point, we will take the assumption that  $0 \leq \delta t \leq \tau \ll 1$ , retaining up to second order terms in  $\delta t$ , and assuming that  $\ddot{\mathbf{r}}_j = 0$ .<sup>de</sup> This allows us to approximate  $\|\mathbf{r}_{\text{event}}\|$  as

$$\begin{aligned} \|\mathbf{r}_{\text{event}}\| &= \|\mathbf{r}_j + \mathbf{v}_j \delta t\| \\ &\approx \|\mathbf{r}_j\| + \delta t \frac{(\mathbf{r}_j \cdot \mathbf{v}_j)}{\|\mathbf{r}_j\|} - \frac{\delta t^2 (\mathbf{r}_j \cdot \mathbf{v}_j)^2}{2 \|\mathbf{r}_j\|^3} + \frac{\delta t^2 (\mathbf{v}_j \cdot \mathbf{v}_j)}{2 \|\mathbf{r}_j\|} \end{aligned} \quad (13)$$

in order to isolate  $\delta t$ .<sup>f</sup> Next, we can break the left-hand-side of Eq.(8) into two integrals; one that spans the time from the particle's lightcone intersection (at  $a_{\text{event}}$ ) to the end of the snapshot  $t_{j+1}$  (at  $a_{j+1}$ ), and one that spans the remaining history of the universe:

$$\int_{a_{\text{event}}}^1 c \frac{da}{a^2 H} = \int_{a_{\text{event}}}^{a_{j+1}} c \frac{da}{a^2 H} + \int_{a_{j+1}}^1 c \frac{da}{a^2 H}. \quad (14)$$

The first thing to note, here, is that the the bounds and integrand of the latter piece over  $[a_{j+1}, 1]$  are entirely known in the simulation parameters and snapshot information, when we express  $H$  as

$$H(t) = \frac{H_0}{a(t)^{3/2}} \sqrt{\Omega_m + \Omega_\Lambda a(t)^3}. \quad (15)$$

<sup>d</sup>With certain assumptions made about our time units, the assertion that  $\tau \ll 1$  can seem absurd; know that our temporal units will ultimately be *program time*, which is essentially the scale factor. See § 4.4 for details.

<sup>e</sup>In the case that the lightcone is being constructed in-situ, this assumption is likely appropriate. Constructing the lightcone as a post-processing task on the output of the simulation, however, operates on much more coarsely resolved time steps (the output snapshots are only a downsampling of the total number of time steps taken in the simulation), and this approximation may be bad. For simplicity, we will retain it throughout this section, but see § 3 for a detailed discussion of nonlinear effects.

<sup>f</sup>A subtle approximation made here; write it as  $\|\mathbf{r}\|^2 = (\mathbf{r}_j + \mathbf{v}_j \delta t) \cdot (\mathbf{r}_j + \mathbf{v}_j \delta t)$ , expand, and apply the binomial series to second order, dropping terms higher than  $\delta t^2$ .

So, we can calculate this integral numerically via Simpson's rule quadrature, and will henceforth refer to that result as

$$\Theta_{j+1} \equiv \text{Simpson} \left( \int_{a_{j+1}}^1 c \frac{da}{a^2 H} \right). \quad (16)$$

Turning our attention to the  $[a_{\text{event}}, a_{j+1}]$  piece of Eq.(14), we see that we cannot trivially numerically evaluate this integral, since we do not know  $a_{\text{event}}$ . In our interest in finding  $\delta t$ , let's instead solve the integral explicitly, and then approximate the result by dropping terms higher than  $\delta t^2$ . We do this by writing this piece in the form of Eq.(6) and changing the variable of integration from  $t$  to  $t' = (t - t_j)$ :

$$\begin{aligned} \int_{a_{\text{event}}}^{a_{j+1}} c \frac{da}{a^2 H} &= \int_{t_{\text{event}}}^{t_{j+1}} c \frac{dt}{a(t)} \\ &\approx \int_{\delta t}^{\tau} c \frac{dt'}{a_j + \dot{a}_j(t')} \\ &\approx \frac{c}{a_j} \left[ (\tau - \delta t) - \frac{\dot{a}_j (\tau^2 - \delta t^2)}{2} \right] \end{aligned} \quad (17)$$

With that, we have all of the pieces required to cast Eq.(8) in terms of quantities available in our simulation output. The left hand side of Eq.(8) is replaced with Eq.(16-17), while the right hand side is replaced with Eq.(13):

$$\begin{aligned} \int_{a_{\text{event}}}^1 c \frac{da}{a^2 H} &= \|\mathbf{r}_{\text{event}}\| \quad (8) \\ &\downarrow \\ \Theta_{j+1} + \frac{c}{a_j} \left[ (\tau - \delta t) - \frac{\dot{a}_j (\tau^2 - \delta t^2)}{2} \right] &= \|\mathbf{r}_j\| + \delta t \frac{(\mathbf{r}_j \cdot \mathbf{v}_j)}{\|\mathbf{r}_j\|} - \frac{\delta t^2 (\mathbf{r}_j \cdot \mathbf{v}_j)^2}{2 \|\mathbf{r}_j\|^3} + \frac{\delta t^2 (\mathbf{v}_j \cdot \mathbf{v}_j)}{2 \|\mathbf{r}_j\|} \end{aligned} \quad (18)$$

which we can then finally solve for  $\delta t$ . The most apparent way to do this is via a quadratic formula

$$\alpha \cdot \delta t^2 + \beta \cdot \delta t + \gamma = 0 \quad (19)$$

$$\implies \delta t = \frac{-\beta \pm \sqrt{\beta^2 - 4\alpha\gamma}}{2\alpha} \quad (20)$$

where we have

$$\alpha = \frac{c\dot{a}_j}{2a_j^2} + \frac{1}{2} \left( \frac{(\mathbf{r}_j \cdot \mathbf{v}_j)^2}{\|\mathbf{r}_j\|^3} - \frac{(\mathbf{v}_j \cdot \mathbf{v}_j)}{\|\mathbf{r}_j\|} \right) \quad (21)$$

$$\beta = -\frac{c}{a_j} - \frac{(\mathbf{r}_j \cdot \mathbf{v}_j)}{\|\mathbf{r}_j\|} \quad (22)$$

$$\gamma = \Theta_{j+1} + \frac{c\tau}{a_j} - \frac{c\dot{a}_j\tau^2}{2a_j^2} - \|\mathbf{r}_j\| \quad (23)$$

We can now solve for  $t_{\text{event}} = t_j + \delta t$  for all particles in any snapshot of the simulation using extrapolation with second order approximations.

If we compute  $\delta t$  in this way for a given particle at snapshot  $j$ , and the result is that  $t_{\text{event}} > t_{j+1}$  (or equivalently  $\delta t > \tau$ ), then we discard the recovered  $\delta t$ , since we will almost certainly obtain a better answer if we extrapolate from the next snapshot  $j + 1$  instead. If  $t_j < t_{\text{event}} \leq t_{j+1}$ , then we save it (an entry for this particle is created in the lightcone output). After doing this for every particle in snapshot  $j$ , we then advance to snapshot  $j + 1$  and repeat the process again. After advancing through all snapshots, the lightcone will be filled.

## 2.2 Particle Duplication

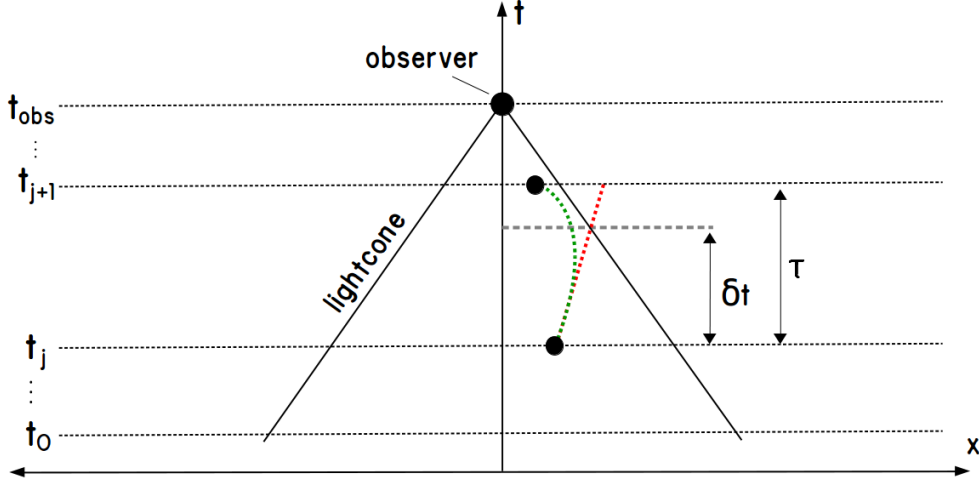
Extrapolating particle positions from their velocities at snapshot  $j$  to find their lightcone-crossing time will work in general, albeit potentially inaccurate because of our approximations. Upon more careful consideration, however, particle extrapolation suffers from the particularly serious ailment of outputting the same particle at two or more times in the final lightcone.

To understand why this "particle duplication" happens, consider the diagram shown in Figure 2. Here, we have a particle whose velocity  $\mathbf{v}_j$  at snapshot  $j$  suggests it to be moving in the  $+x$  direction (in the diagram,  $x$  is representing all of space rather than a Cartesian component). Extrapolating its position places it outside of the lightcone at time  $t_{j+1}$  of the next snapshot, and estimates it to have crossed the lightcone at time  $t_j + \delta t$ . This result is saved and seen in the lightcone output. What *actually* occurred is that the particle underwent some kind of interaction, which is temporally unresolved in the downsampled snapshot outputs, that altered its trajectory. Therefore, its real path diverges significantly from the extrapolated estimate. In the extreme case as shown in Figure 2, the particle won't actually cross the lightcone at all until some later time  $t_{\text{event}} > t_{j+1}$ , which will also be saved in the output of some later snapshot. The net effect is that the extrapolation method described in § 2.1 places this particle on the lightcone *twice*<sup>§</sup>, meaning the observer sees the same object at two different redshifts. This of course should be impossible without the object exceeding the speed of light. The converse situation is also true, and equally as serious, in which certain particles *never* appear in the lightcone (the true path crosses the lightcone, while the extrapolated estimate does not).

This duplication error is significant, and effects a few percent of particles, with an even worse effect on halo lightcone catalogs near 5-10%. The obvious fix for this is to calculate lightcone-crossing times by particle interpolation rather than extrapolation (in Figure 2, a new estimated path connecting the two particle positions would be a much better representation of the true trajectory in green). This approach was not immediately implemented due to the simplicity of the extrapolation method, and lack of foresight into the presently discussed bugs of which we are now well acquainted. Building the lightcone in-situ makes the extrapolation method especially appealing since no knowledge of the upcoming timestep is needed. In any case, particle position interpolation is a necessary requirement for clean and reliable lightcone catalogs, which is described next in § 2.3.

---

<sup>§</sup>Or even more than twice, if the duplication artifact appears more than once for a particular particle.



**Figure 2:** A schematic showing the origin of particle duplication issues with the extrapolation approach. The vertical axis is time, and the horizontal is space. Time increases upward. The particle is found at snapshot  $j$  with an initial velocity pointing in the  $+x$  direction. Extrapolating the position based on this velocity yields the estimated trajectory in red, which crosses the lightcone at time  $t_j + \delta t$ . The particle's true path is shown in green, which experienced some kind of interaction shortly after the time  $t_j$ . Since the particle is actually found *inside* the lightcone at time  $t_{j+1}$ , it must end up crossing the lightcone again<sup>h</sup>, thus being seen by the observer at two different times. The path shapes are greatly exaggerated—no simulation particles reach such relativistic speeds.

### 2.3 Particle Interpolation

We have taken a very simple approach to the position-interpolation improvement, so that we would avoid having to perform significant code refactoring (with respect to the original implementation described in § 2.1). Consider again a particle at position  $\mathbf{r}_j$  (velocity  $\mathbf{v}_j$ ) at snapshot  $j$ , and position  $\mathbf{r}_{j+1}$  (velocity  $\mathbf{v}_{j+1}$ ) at snapshot  $j+1$ . We first find an "equivalent linear velocity"—a constant velocity such that would move the particle from  $\mathbf{r}_j$  to  $\mathbf{r}_{j+1}$  in time  $\tau = t_{j+1} - t_j$ :

$$\mathbf{v}_{\text{linear}} = \frac{\mathbf{r}_{j+1} - \mathbf{r}_j}{\tau}. \quad (24)$$

This of course requires that, when working on a particular snapshot  $j$  in the process of lightcone construction, we also read in snapshot  $j+1$  (not necessary in the original implementation). Once we have done this, the result of Eq.(20) can simply be fed into the framework of § 2.1, where  $\mathbf{v}_{\text{linear}}$  would replace any occurrence of  $\mathbf{v}_j$ . In that case, Eq.(10), which becomes

$$\begin{aligned} \|\mathbf{r}\| &= \|\mathbf{r}_j + \mathbf{v}_{\text{linear}}\delta t\| \\ &= [(x_j + \dot{x}_{\text{linear}}\delta t)^2 + (y_j + \dot{y}_{\text{linear}}\delta t)^2 + (z_j + \dot{z}_{\text{linear}}\delta t)^2]^{1/2}, \end{aligned} \quad (25)$$

is exactly an interpolation. Eq.(13) is then the approximated interpolated position which allows us to solve for  $\delta t$ , and the rest of the procedure is unchanged. Obviously, by using a linear interpolation, we are enforcing a discontinuous derivative in our particle positions. Really, we could push that discontinuity back to second order (we *could* use the velocities  $\mathbf{v}_j$  and  $\mathbf{v}_{j+1}$  to perform a nonlinear interpolation, rather than throwing them away in favor of  $\mathbf{v}_{\text{linear}}$ ). See § 3 for further discussion.

### 3 Nonlinear Effects

Through the implementation and testing of the problem solution laid out in § 1-§ 2.3, we have discovered several shortcomings, in which our approximations manifest as discreteness effects and other unwanted artifacts in the final lightcone outputs. We should of course like to minimize these effects, and remove them entirely where possible, so it will be useful to go over a road map of all approximations made up until this point, and what their behavior may be in cases of interest.

Our condition defining the lightcone surface, Eq.(8), is exactly correct. Downstream, however, we dropped higher order derivatives and terms on each side of this equation, which we should consider with care. Here is the expanded form of the Eq.(8) that we derived using discrete simulation outputs, Eq.(18), again:

$$\Theta_{j+1} + \frac{c}{a_j} \left[ (\tau - \delta t) - \frac{\dot{a}_j (\tau^2 - \delta t^2)}{a_j} \right] = \|\mathbf{r}_j\| + \delta t \frac{(\mathbf{r}_j \cdot \mathbf{v}_j)}{\|\mathbf{r}_j\|} - \frac{\delta t^2 (\mathbf{r}_j \cdot \mathbf{v}_j)^2}{2 \|\mathbf{r}_j\|^3} + \frac{\delta t^2 (\mathbf{v}_j \cdot \mathbf{v}_j)}{2 \|\mathbf{r}_j\|} \quad (18)$$

First, focusing on the left-hand side; the second summand, here, was evaluated as the integral in Eq.(17). During the second line of that equation, we make the approximation that  $a(t_j + \delta t) \approx a_j + \dot{a}_j \delta t$ , in other words,  $\{d^n a / dt^n = 0 : n \geq 2\}$ . In principle, we could analytically obtain higher order derivatives of  $a(t)$  from the Friedmann equation, though the scale factor evolution is quite smooth, and any noticeable impact is doubtful. More importantly, to get the final line of Eq.(17), we perform a first-order Taylor expansion about  $a_j$  to evaluate the integral. In doing this, we have again invoked the assumption that  $\delta t \ll 1$ , which seems acceptable on paper, though the direct effects of this particular approximation are probably not fully understood.

Turning our attention to the right-hand side of Eq.(18); as done with the scale factor, we have assumed that  $\{d^n \mathbf{r} / dt^n = 0 : n \geq 2\}$ . For the extrapolation approach, this is the best that we can do, as no acceleration data is exposed to us. For the interpolation routine, however, we could make an estimate of  $\ddot{\mathbf{r}}$  as  $(\mathbf{v}_{j+1} - \mathbf{v}_j) / \tau$ , which would relax the restriction that all particle trajectories be represented as linear functions. This may be a desired change to make, though ultimately, the second line of Eq.(13) will still require choosing a center for the Taylor expansion. We choose to use the point  $a_j$  to center our expansions, which may be as good as any.

Finally, we must note that, in the code, we do not actually evaluate the traditional quadratic form as is written in Eq.(20). Rather, we perform an approximated quadratic solution, expanded to second order, which is

$$\begin{aligned} \alpha \cdot \delta t^2 + \beta \cdot \delta t + \gamma &= 0 \\ \implies \delta t &= \frac{-\beta \pm \sqrt{\beta^2 - 4\alpha\gamma}}{2\alpha} \\ &\approx \frac{-\gamma}{\beta} - \frac{\alpha\gamma^2}{\beta^3} \end{aligned} \quad (26)$$

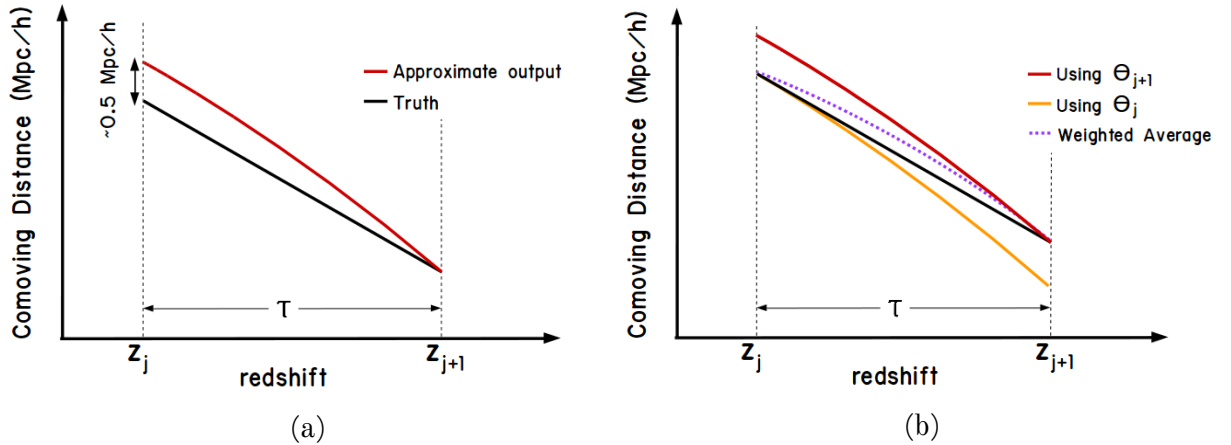
We do this to minimize potential numerical instabilities associated with evaluating the full quadratic solution. There are many such possible instabilities, e.g. the case when



$4\alpha\gamma$  is small compared to  $\beta^2$ , and the sign  $\beta$  and that of the radical term are the same, will give us catastrophic cancellation. One can try to avoid this problem by also using the *Citardauq Formula*, which inverts the sign of the radical term for any particular solution, and choosing to use the more stable expression for the root  $\delta t$ . We have, for nothing more than simplicity, instead made the more casual decision to use the approximated solution, Eq.(26), which still does eliminate numerical artifacts which otherwise cause the volume of our lightcone output to explode at high redshifts.

While we have not dug deep enough to discover all of the specific influences that each of the above discussed approximations have on our answer, we are aware that compounding all of them has adverse effects. The statistic in which this is most obviously seen is the comoving-distance-redshift relation. We consistently find that our solution over-estimates the distance to the lightcone surface at the high-redshift end of each timestep bounded by snapshots  $j$  and  $j + 1$ , as illustrated in the toy plot given in Figure 3a. Specifically, we see that particles whose lightcone crossing time was found to be  $\delta t \rightarrow 0$  are around  $\sim 0.5 \text{ Mpc}$  too far from the true lightcone surface (the black curve in Figure 3a). Alternatively, we can think of this effect as simply being an apparent increase in the speed of light, in some way due to our approximations.

In response to this issue, we offer what may be seen as a workaround if not a remedy. Notice that the distance-redshift relation recovered from our solution (red curve in



**Figure 3:** A toy-figure showing all the qualitative features of the comoving distance-redshift relation, as recovered from the output of the lightcone problem solution described in § 1-§ 2.3. **Fig(3a):** The output relation if we apply the particle interpolation routine to Eq.(18-23). Due to our approximations, the relation departs from the truth by a maximum of  $\sim 0.5 \text{ Mpc}$  at the high- $z$  end. This error repeats for each step  $j$ , introducing a sawtooth pattern in the relation. **Fig(3b):** The same as Fig(3a), with the same relation overplotted, in the case that we replace  $\Theta_{j+1}$ , in Eq.(18) and Eq.(23), with  $\Theta_j$ . The error in these two answers appears symmetric, and we can take a weighted average of the two answers to mitigate our approximation errors. Doing this ultimately gives a maximum departure from the true relation of  $\sim 100 \text{ kpc}$ .

Figure 3a) is exactly correct for particles that cross the lightcone at  $\delta t = \tau$ . To understand this, let's look more closely at the LHS of Eq.(18):

$$\begin{aligned} \int_{a_{\text{event}}}^1 c \frac{da}{a^2 H} &= \int_{a_{\text{event}}}^{a_{j+1}} c \frac{da}{a^2 H} + \int_{a_{j+1}}^1 c \frac{da}{a^2 H} \\ &= \frac{c}{a_j} \left[ (\tau - \delta t) - \frac{\dot{a}_j (\tau^2 - \delta t^2)}{a_j} \right] + \Theta_{j+1} \end{aligned}$$

From here, it should be clear that we expect a good answer at  $\delta t = \tau$ , because in that case,  $a_{\text{event}} = a_{j+1}$ , and the integral over  $[a_{\text{event}}, a_{j+1}]$  vanishes. That only leaves the integral represented as  $\Theta_{j+1}$  to worry about, which is just a numerical Simpson's method calculation. Simpson's method can be as accurate as we'd like it to, with such a smooth function, and all of our approximations in  $a(t)$  have dropped out.

Given this, we see that there is a symmetry to be taken advantage of in this problem. We'd like to calculate the entire integral over  $[a_{\text{event}}, 1]$  numerically, though we cannot trivially do that, since our lower integration bound is unknown. We therefore made the arbitrary choice to split this integral into  $[a_{\text{event}}, a_{j+1}]$  and  $[a_{j+1}, 1]$  pieces, though we could have just as well come up with the following arrangement:

$$\begin{aligned} \int_{a_{\text{event}}}^1 c \frac{da}{a^2 H} &= \int_{a_j}^1 c \frac{da}{a^2 H} - \int_{a_j}^{a_{\text{event}}} c \frac{da}{a^2 H} \\ &= \Theta_j - \frac{c}{a_j} \left[ \delta t - \frac{\dot{a}_j \delta t^2}{2} \right], \end{aligned} \quad (27)$$

where we have introduced  $\Theta_j$  as

$$\Theta_j \equiv \text{Simpson} \left( \int_{a_j}^1 c \frac{da}{a^2 H} \right). \quad (28)$$

Eq.(26) leads to the exact solution for  $\delta t$  as given in Eq.(19-23), except that the constant term in the quadratic has its summands involving  $\tau$  removed, and  $\Theta_{j+1}$  is replaced with  $\Theta_j$ :

$$\gamma_{\text{symmetric}} = \Theta_j - \|\mathbf{r}_j\|. \quad (29)$$

If we re-solve for  $\delta t$  (Eq.(26)) using this new constant quadratic term, then we should now have perfect agreement with the true distance-reshift relation at *high-z*, rather than low, with our approximations now manifesting as error at *low-z*. This is shown in Figure 3b, and we will refer to the result of solving for  $\delta t$  in this revised calculation as  $\delta t_{\text{symmetric}}$ .

Now that we have established a way to move the approximation-induced error to opposite ends of the time domain of a simulation step, we can leverage this symmetry by using a weighted average of our two measures of the lightcone crossing time per particle,  $\delta t$  and  $\delta t_{\text{symmetric}}$ . As should be clear from the curves in Figure 3b, the weights for this average should be something like  $\delta t_{\text{true}}/\tau$ . Since we do not have  $\delta t_{\text{true}}$ , and have no real way to assert the accuracy of  $\delta t$  against  $\delta t_{\text{symmetric}}$ , or vice-versa, we will use the mean value of these two values in the weighting:

$$w = \frac{\delta t + \delta t_{\text{symmetric}}}{2} / \tau \quad (30)$$

with our improved lightcone crossing time given as

$$\delta t w + \delta t_{\text{symmetric}}(1 - w) \quad (31)$$

This procedure results in the  $\sim 0.5\text{Mpc/h}$  error in the distance-redshift relation at the earlier timestep endpoint, mentioned above, being reduced to a maximum of  $\sim 100\text{kpc}$  at the center of the timestep (the new error is the dashed purple curve in Figure 3b, which is still not quite zero, since the concavity of each solution has the same sign).

Now, let's emphasize an important point; this workaround, while clever, is cumbersome and inelegant. The ideal way to deal with nonlinear effects is to take a fully numerical approach. That is, *both* integrals of Eq.(14) should be solved iteratively, rather than one of them being given an approximate analytical evaluation. In other words, the second summand on the LHS of Eq.(18) should also be a numerical evaluation. To reiterate once more, this seemingly obvious task is nontrivial due to the fact that one of the integration limits itself is bound up in the RHS of Eq.(18). Finding a solution, however, should be doable for a physicist with free research time (i.e. not me or any other contributors to this document). But we do promise, it would be fun!

## 4 Implementation

This section will describe where and how, in HACC, each of the components of the lightcone construction (§ 1 - § 3) are performed. What follows is written with frequent reference to the structure of the HACC framework and its important classes, though familiarity is not necessarily assumed, on the part of the reader. Special attention will be given to discussing the units of various quantities involved, which proves to be a nuanced and sub-optimally documented ingredient of the codebase. If one finds difficulty in understanding, or developing, any lightcone-relevant codes such as those described here, time should be afforded for careful dimensional analysis.

§ 4.1 introduces the lightcone driver and gives an overview of the input parameters and options. § 4.2 gives a detailed look at the problem geometrically, explaining the details of simulation box duplication, and § 4.3 describes the location of all components of the solver. § 4.4 makes sense of the units of the important quantities in the problem, and § 4.5 finally describes all output products.

### 4.1 Parameters & Options

The source files for this project are located in `/hacc/nbody`. The driver for the interpolation scheme (§ 2.3) is at `/nbody/simulation/driver_lc_interpolation.cxx`, and its `main()` function takes six arguments:

1. <code>&lt;paramName&gt;</code>	path to a HACC param file
2. <code>&lt;outName&gt;</code>	output file path
3. <code>&lt;inName&gt;</code>	input file path for snapshot $j$ particle data
4. <code>&lt;stepNumber&gt;</code>	step number $j$
5. <code>&lt;nextInName&gt;</code>	input file path for snapshot $j + 1$ particle data
6. <code>&lt;nextStepNumber&gt;</code>	step number $j + 1$

Table 1: Input parameters for the lightcone driver's `main()` function

Likewise, the driver for the original extrapolation code (§ 2.1) is found at `/nbody/simulation/driver_lc.cxx`, and takes the same arguments as listed above,

Table 2: Parameter file options most relevant to lightcone construction

Option	Type	Description
1. LCX, LCY, LCZ	float	The cartesian x, y, or z position, respectively, of the lightcone origin (observer) in physical Mpc/h
2. LC_ALL_OCTANTS_X, LC_ALL_OCTANTS_Y, LC_ALL_OCTANTS_Z	bool	Whether or not to generate lightcone output in both sky octants along the cartesian x, y, or z directions, respectively <sup>i</sup>
3. LC_REPLICANTS	int	The number of simulation boxes to replicate along one axis, in the positive direction, not including the initial volume <sup>j</sup>
4. LC_REDSHIFT	float	The extent of the lightcone specified as a redshift. This value is used to determine LC_REPLICANTS automatically if LC_USE_REDSHIFT is TRUE
5. LC_USE_REDSHIFT	bool	Whether or not to allow the driver to calculate LC_REPLICANTS itself, given the value of LC_REDSHIFT <sup>k</sup>
6. LC_AUTO_REDSHIFT	bool	Whether or not to allow the driver to automatically set LC_REDSHIFT to be the redshift of the input step, with a 10% padding, given that LC_USE_REDSHIFT is also TRUE <sup>l</sup>
7. LC_ROTATE	bool	Whether or not to apply random rotations to simulation box replications
8. LC_ROTATE_SEED	int	Seed for random box rotations
8. OUTPUT_FRACTION	float	Fraction of input particles to output, in the range [0, 1]
9. LC_FULL	bool	Whether or not to use the full input dataset (as given by parameter 3. in Table 1). If TRUE, then the value for OUTPUT_FRACTION is ignored.
10. LC_SOLVER_NONLIN	bool	Whether or not to use the full numerical lightcone crossing-time solver (not yet implemented; motivated in § 3)
11. LC_REVERSE	float	Whether or not to reverse the solver, projecting objects at time $t_{j+1}$ backward, rather than objects at time $t_j$ forward.
12. LC_MIN_MASS	bool	The minimum mass that a simulation object must be to be included in the lightcone output <sup>m</sup>
13. LC_WRITE_ORIG_POS	bool	Whether or not to write out the original object positions (from the base snapshot), rather than only the extrapolated or interpolated positions found at the lightcone crossing time.

<sup>i</sup>If the x, y, and z versions of this option are all set to FALSE, then the lightcone generated is one octant of the sky. Having only one of these options set to TRUE will produce a quarter sky, etc., and all of them being set to TRUE will output a full-sky lightcone.

<sup>j</sup>Though this option specifies the number of replications along one axis, the box replication actually occurs along all three cartesian axes. It also occurs in both the negative *and* positive direction for each axis  $k$  where LC\_ALL\_OCTANTS\_K = TRUE. Thus, finding the total number of replications requires a calculation, which is described in § 4.2.

<sup>k</sup>If this option is TRUE, then the param file entry for LC\_REPLICANTS is ignored. Otherwise, LC\_REDSHIFT is ignored

<sup>l</sup>Therefore, if both this option and LC\_USE\_REDSHIFT are set to TRUE, the decision of the number of simulation box duplicates is fully automated, and the param file entries for both LC\_REPLICANTS and LC\_REDSHIFT are ignored

<sup>m</sup>This option is relevant in the case that the lightcone is being built from halo datasets, or some other object that varies in mass, rather than gravity-only mass particles.

excluding `<nextInName>` (finding lightcone crossing times using extrapolation does not require knowledge of a particle’s position at step  $j + 1$ ). The remainder of this section will apply equally to either the approaches of § 2.1 or § 2.3 unless otherwise noted.

Running the driver code begins by initializing MPI, and then creating an instance of the `MC3Options` class. This object is known to the driver as `options`, and is the interface through which run parameters are stored, set, and retrieved. The state of the `options` object is modified, with respect to the default constructor, by the content of a `Basedata` object in charge of reading the parameter file (argument *1* in Table 1), which is expected to have been passed by the caller of the lightcone driver. Valid parameter file options most relevant to the building of lightcones using this code are given in Table 2 below.

At this point, any reader that finds the content of Table 2 to be clear may decide to skip to § 4.3. Otherwise, § 4.2 gives a very detailed description of the geometry of the problem we are trying to solve, which many of the options in Table 2 concern themselves with.

## 4.2 Geometrical Considerations

Before touring the rest of the code, we should discuss how best to visualize the geometry of the problem in the context of cubic cosmological simulation volumes. This discussion will highlight the purpose and usage of most of the options in Table 2 that may appear somewhat abstruse.

An important characteristic of numerical simulations that have guided our problem-solving thus far is that information is only known at discrete time snapshots. Our approach, conclusions, and notation all acknowledge this limitation. However, we have, thus far, implicitly assumed that we have infinite spatial information, or at least that we have spatial information out to the furthest possible extent of our lightcone. Of course, this is not true; the largest simulations volumes that have been run are expanding cubes whose comoving side-lengths are in the neighborhood of  $4\text{Gpc}/h$ . If we place an observer at the corner of one such simulation volume, to what extent can our lightcone reach? The lightcone surface is a spherical object centered on the observer, and it’s radius could not exceed, in this case, a comoving distance  $4\text{Gpc}/h$ — a distance which light will cross in a reasonable amount of time, and, assuming a WMAP7 cosmology, only allows our observer to see to a maximum redshift of  $\sim 2.3$ .

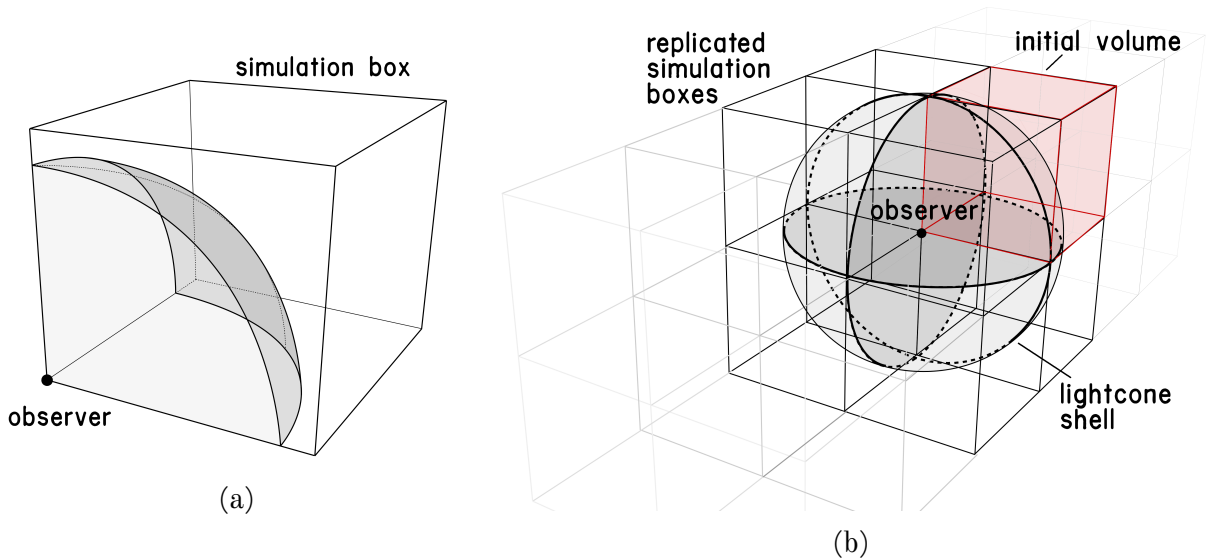
On top of this distance limitation, we have also introduced a direction limitation; situating our observer in the corner of the box maximizes the radius of the lightcone, but only allows one octant of the sky to be seen. Placing an observer, instead, in the center of the box allows for a full-sky lightcone to be generated, but in that case the lightcone extent would be reduced to redshift  $\sim 0.8$ . These issues are of course only more constraining with smaller simulation volumes.

Surely we would like to fill lightcones with simulation particles/objects to any arbitrary redshift we choose, and surely we would also like the option to do so over the entire angular domain of any observer. Our solution is to grow the spatial extent of our simulation output, effectively, by replicating the box many times and “tiling” these replications in space. This idea is shown visually in Figure 4. For example, if we’d like to fill an all-sky lightcone out to a redshift of 2 (a comoving distance of  $\sim 3.7\text{Mpc}/h$  given WMAP7) with particle output from a  $2\text{Gpc}/h$  simulation, we would replicate the

box once in each cartesian direction for a limiting lightcone radius of  $4\text{Gpc}/h$ .

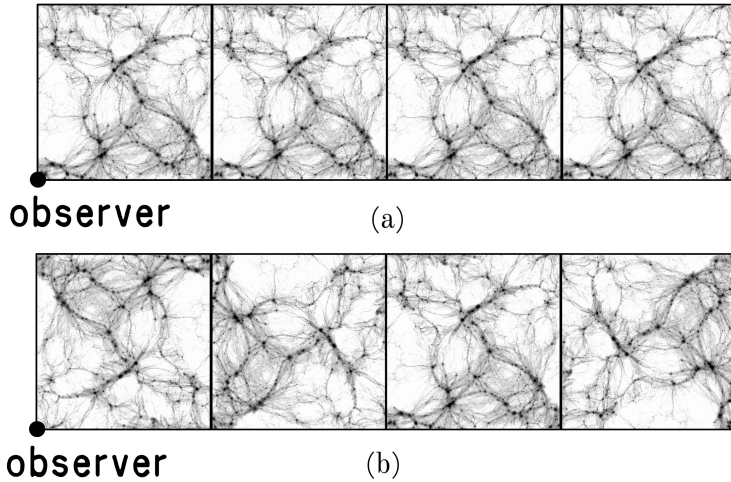
While implementing this replication technique does solve the issues described above, and allows us to generate a lightcone to any arbitrary redshift, it introduces some further complications of its own. If we carelessly replicate the simulation volume many times, the presence of repeating large-scale structure will start to become visually obvious, and inject artifacts into various statistics, e.g. power spectra, redshift distributions, etc. For this reason, the code has been given the capability to randomly rotate each replication by swapping the  $x$ ,  $y$ , and/or  $z$  positions and velocities of its particles. This solution, in turn, has yet another side-effect: before rotating the replicated boxes, they are all seamlessly joined to each other due to the periodic boundary condition of each box edge. This property will be lost after randomly-rotating each volume, and discontinuities can be created at box edges (filaments and halos may be clipped).

In short, we are forced to decide between introducing either repeating large-scale structure, or large-scale edge effects, into the lightcone output. The latter has tended to be the less significant problem. These issues are further detailed in Figure 5, where the box replications along a single axis are visualized.



**Figure 4:** A comparison of the problem geometry in the case that one simulation volume is used, with the case that the volume is replicated and tiled. **Fig(4a):** A view of a simulation volume containing a lightcone, filling an octant of the sky, being generated with respect to an observer placed at the origin (bottom left corner). The spherical shell is the extent of the lightcone (a surface upon which lie the furthest, and youngest, objects that the observer can see) at some time  $t$ . Since the extent of the lightcone has not reached the other side of the box yet (light from the other side of the box has yet to reach the observer), it is true that  $t < L/c$ , where  $L$  is the box length. Notably, we *cannot* generate a lightcone, using this simulation volume, to allow the observer to see events occurring beyond a time  $t_{\text{max}} > L/c$  into the past<sup>n</sup>. **Fig(4b):** A view of a full-sky lightcone now being generated across a tiling of duplicated simulation boxes. The red volume indicates the initial volume shown in Fig(4a). Notice that now we have eight times the sky-coverage, without having had to reduce the radius of the lightcone. We can replicate the simulation boxes as many times as we'd like to produce an arbitrarily large lightcone.

<sup>n</sup>More accurately, if we consider cosmological dynamics as discussed in § 1, it is true that setting  $t_{\text{obs}} = t$  in Eq.(6) will yield a distance of  $\|\mathbf{r}\| < L$ , and that we cannot enable the observer, using this simulation volume, to see events occurring more than a time  $t_{\text{max}}$  into the past, such that setting  $t_{\text{obs}} = t_{\text{max}}$  in Eq.(6) gives  $\|\mathbf{r}\| = L$ .



**Figure 5:** A comparison of the consequences of simulation box replications, with (Fig(5b)) and without (Fig(5a)) enabling random rotations. The problem of repeating LSS is clear in Fig(5a), as the density along the lower box edge is noticeably higher than the top edge. Repeating structures along this lower edge have very little angular separation in the observer’s line of sight, making the effect particularly strong. These issues are cured in Fig(5b), though some density discontinuities are obvious (notably along the vertical edge separating the initial and first replicated volumes).

With all of this in place, we are equip to parse the meaning and usage of the options listed in Table 2. Option 7 in Table 2, `LCRotate`, is simply a boolean value which turns the box replication rotation, as described above, on or off. Option 8, `LCRotateSeed` is self-explanatory, though it is worth taking a moment to note that the random number generator at work here, given a constant seed, gives reproducible results in general when running in parallel; this means that the MPI topology is free to change without effecting the spatial configuration of the output (for instance, one may desire to run a full particle lightcone to obtain shear maps, and also a corresponding galaxy lightcone, where the galaxy lightcone requires many fewer nodes to compute in a reasonable amount of time. The rotations of each simulation box replication would be preserved between these two runs).

Options 2-6 in Table 2 ought to be treated more carefully. Option 3 gives the number of box replications desired, *along one axis, in the positive direction*. Options 2 deliver three boolean values to the code which control the sky-coverage of the lightcone (see footnote j). How these options are actually operating on the spatial configuration of the problem are as follows: setting `LCAllOctantsX` to `TRUE` will replicate boxes in *both* the positive and negative  $x$  directions, and likewise for the Y and Z variants of this option. Therefore, in order to actually calculate the number of replications, something like following operation is performed:

```
int NumRepsX = LCReplicants+1 + (LCAllOctantsX ? 1 : 0)*(LCReplicants+1);
int NumRepsY = LCReplicants+1 + (LCAllOctantsY ? 1 : 0)*(LCReplicants+1);
int NumRepsZ = LCReplicants+1 + (LCAllOctantsZ ? 1 : 0)*(LCReplicants+1);
int numBoxesTotal = NumRepsX*NumRepsY*NumRepsZ;
```

Let’s dissect the declaration of `NumRepsX`; why is it that each time `LCReplicants` is used (the value of Option 3 defined by the user), it is incremented by one? The leftmost +1 is adding the initial simulation volume to the total box count, and the rightmost +1 is to keep the total replicated volume symmetric across the  $y$ - $z$  plane. This means that

<sup>o</sup>Credit for the cosmic web image used in the creation of this figure to Benedikt Diemer: <http://www.benediktdiemer.com/visualization/images/>

if `LCA110ctantsX` is true, then the number of replications in the  $x$  direction is doubled, which makes sense. This all of course holds for the Y and Z variants of the quantity as well.

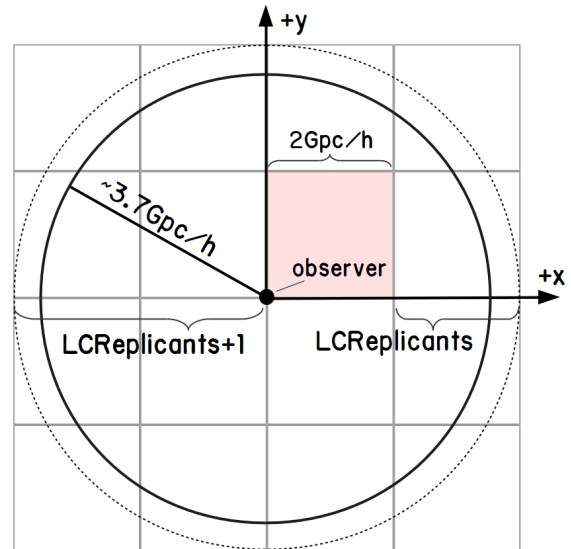
The potential clumsiness of the last few paragraphs suggest that a figure would be most helpful at this point, so let's return to our example as stated on page 13. There, we imagined constructing a full-sky lightcone out to a redshift of 2 (comoving distance of  $\sim 3.7\text{Mpc/h}$ ) using a  $2\text{Gpc/h}$  simulation. We said that this would require one box replication, in each cartesian direction, yielding a volume capable of building, at most, a lightcone with an extent of  $4\text{Gpc/h}$ . Figure 5 shows the  $x$ - $y$  plane of the presently discussed situation. Does this plan make sense, given the behavior of Options 2 and 3 described above? It does, if we define these options properly. The problem outline above suggests that we use the option values

```
LCReplicants = 1
LCA110ctantsX = LCA110ctantsY = LCA110ctantsZ = TRUE .
```

Were we only looking to generate an octant of the sky, we would change the latter three of these options to

```
LCA110ctantsX = LCA110ctantsY = LCA110ctantsZ = FALSE .
```

**Figure 6:** A visualization of the spatial configuration of the simulation boxes being duplicated in the example problem discussed on page 16. The initial  $2\text{Gpc/h}$  volume is tinted red. Recall that `LCReplicants` was set to 1 in this example, meaning that one box will be replicated in the *positive* direction of some single cartesian axis. With the added consideration of the initial volume, that makes two box lengths of data from the observer at the origin to the lightcone extent. Since we have specified that we would like to generate a full sky, we replicate two more boxes in the *negative* directions. The extent of the lightcone that we desire to build is shown by the dark circle, at  $\sim 3.7\text{Mpc/h}$ , and the largest possible lightcone that this box configuration could accommodate is shown as the dashed circle, at  $\sim 4\text{Mpc/h}$ . In this example, we see 16 total simulation volumes, or 64 total volumes if we also consider those replicated in the  $z$  direction (omitted from this figure). Solving the expression for `numBoxesTotal` on pg. 15, given our parameter choices, agrees with this figure.



Finally, as a last consistency check, it should now be easy to understand that the options necessary to generate the configuration shown by the bold-faced boxes in Figure 3b would be

```
LCReplicants = 0
LCA110ctantsX = LCA110ctantsY = LCA110ctantsZ = TRUE .
```



In summary, the behavior of `LCReplicants` isn't necessarily intuitive— it acts this way so that one need only find the number of box-lengths required to reach the desired lightcone radius, in one dimension, to specify the number of replications (remembering that the initial volume does not count as a “replication”).

Now, we can briefly describe Options 4-6. If Option 5, `LCUseRedshift`, is set to `TRUE`, then the code will look toward Option 4, `LCRedshift` to calculate `LCReplicants` internally. This was implemented as a convenience to the user— since lightcones, as a part of science projects, are usually planned to extend to a particular redshift, rather than a distance, the user would have to manually convert this redshift to a distance under the cosmology of the simulation run, and then find the necessary value of `LCReplicants`. Specifying the redshift of the lightcone instead, via Options 4&5, prompts the code to solve the integral given in Eq.(16) to compute the necessary `LCReplicants`, by setting the lower integration bound  $a_{j+1}$  to  $1/(1+LCRedshift)$  (recall the discussion about Eq.(8) to see why this works).

This *still* leaves work for the user, however, since the lightcone code is run on a snapshot-by-snapshot basis. If a user is running the snapshot output of step 487 ( $z \approx 0.02$ ) through the lightcone solver, as part of an effort to build a lightcone to  $z = 3$ , then they will certainly be doing far more calculations than is necessary if `LCRedshift=3` (which will replicate the volume many times). To combat this last complication, Option 6 is offered. If both Option 5 and Option 6, `LCAutoRedshift`, are set to `TRUE`, then the `LCRedshift`, and hence `LCReplicants`, will be computed internally. This works by simply calculating the redshift of the input snapshot number, plus a 10% redshift padding, to set the value of `LCRedshift`.

In summary, generating a lightcone using this code offers three distinct parameter usages, as they relate to box replication:

1. `LCUseRedshift=LCAutoRedshift=FALSE`  
`LCReplicants = some integer`
2. `LCAutoRedshift = FALSE`  
`LCUseRedshift = TRUE`  
`LCRedshift = some value`
3. `LCReplicantsLCUseRedshift = LCAutoRedshift = TRUE`

Use case 1 in the list above will offer full control to the user in specifying the details of the problem configuration. Use case 2 is slightly more convenient, while use case 3 will minimize the calculations demanded from the user, lowering the potential for error, and possibly allow the user to forget most of the present section of this document.

### 4.3 Code Layout

Let us recall the discussion as left at the end of § 4.1. After the first few tasks performed by the lightcone driver, we now have MPI initialized, with `MC3Options` and `Basedata` objects storing our run parameters (including those so explicitly treated in § 4.2). In a moment, two more important objects will be introduced, though I will not subsequently continue to detail every operation and all HACC objects created by the driver, as that is a job better left to the in-line documentation, of which there is plenty.

This section will, instead, offer a high level road-map of where the various steps of the problem solution are carried out.

To borrow the notation introduced in § 2.1, consider running the driver on a timestep bounded by snapshots  $j$  and  $j + 1$ . The original extrapolation version of the driver (`nbody/simulation/driver_lc.cxx`) continues by reading the snapshot  $j$  particle data from a specified `gio` file (parameter 3 in Table 1). Now, for the two important objects mentioned above; a `ParticleActions` object (henceforth `pa`) is then created, as well as two `TimeStepper` instances for keeping track of  $z$ -dependent cosmological parameters at times  $t_j$  and  $t_{j+1}$ . The member functions of `pa` are what perform most of the calculations that we care about in solving for lightcone crossing times, and they are found at `nbody/cpu/ParticleActions.cxx`.

Moving on; if `LCUseRedshift` and/or `LCAutoRedshift` (see Table 2) are given as `TRUE` by the user, then the function `pa.calcLCReplicants()` is called, which performs some of the duties laid out in § 4.2. Ultimately, the particle data is passed to `pa.map_lc()`, which in turn calls the solver code at `pa.update_lc()`.

`pa.update_lc()` is where most of the magic happens. The box replications are assigned random rotations, positions and velocities are scaled to the proper units (see § 4.4), and some prerequisite computations are done to prepare for running the solver code. Eq.(15), the time-dependent Hubble parameter, is calculated in `pa.Hat()`. Eq.(16,28), Simpson’s rule numerical integrations giving the distance from the observer to the lightcone extent at  $t_{j+1}$  and  $t_j$  (which the code denotes as  $A$  rather than  $\Theta$ ), is found in `pa.calcA()`.

With all of this in place, the particle data can be passed to the solver function, either `pa.calc_dt()` or `pa.calc_dt_nonlinear()`, depending on the value of the option `LCSolverNonlin`. The former of these solver functions ultimately implements Eq.(31) in § 3, and the latter is yet to be implemented. After this routine has been visited by every particle, in every box replication, the results are written out in `gio` format via `pa.writeLC()`. The specific form and content of this output is described in § 4.5.

The particle interpolation version of the driver (`driver_lc_interpolation.cxx`) does all of the things discussed thus far, with some additions. In that case, we need particle position information at *both* snapshots  $j$  and  $j + 1$ , so two `gio` files are now read in (the second given by parameter 5 in Table 1). For each particle belonging to snapshot  $j$ , we use a binary search algorithm to find its location also in snapshot  $j + 1$ . With this mapping completed, we may interpolate the particle’s position, which is done exactly as described in § 2.3. If any particle from snapshot  $j$  is found to have no match in  $j + 1$ , which can happen, then we default the calculation of  $\delta t$  for that particle to the extrapolation method (we do not update its velocity).

## 4.4 Units

This section, as promised, gives a careful look at the units of the important quantities in this problem, and their transformations. Particle coordinate values are converted between different systems in HACC via the `ParticleCoords` class, whose constructor is called as

```
ParticleCoords(units, scope, periodic, symplectic)
```

where the latter two arguments are expected to be boolean values. The particle data is read in from the input snapshot files by the driver in the same form that they are written out by the simulation, where `units=PHYSICAL`, `scope=GLOBAL`, `periodic=true`, `symplectic=false`. `PHYSICAL` specifies that particle coordinates are given in physical units (Mpc/h for positions, and km/s for velocities) and `GLOBAL` says that those coordinates are given with respect to the global simulation origin (as opposed to the local origin of the MPI rank to which the particle belongs). `periodic` refers to the boundary conditions of the coordinates, while `symplectic` specifies whether or not velocities are multiplied by a factor of  $a^2$ .

Now, are these the proper units for our quantities to be in with respect to our solution strategy derived in § 1-§ 2.3? The answer is no, and some transformations are going to be required. It should at least be immediately clear that the unit mismatch between our particle positions and velocities is of no help, since we seek to use those velocities to approximate new positions (either by extrapolation or interpolation). To understand what is about to happen, let's print the first term of the approximated quadratic form which we are using to compute  $\delta t$ , Eq.(26):

$$\delta t \approx \frac{-\gamma}{\beta} = \left[ -\Theta_{j+1} - \frac{c\tau}{a_j} + \frac{c\dot{a}\tau^2}{2a_j^2} + \|\mathbf{r}_j\| \right] / \left[ -\frac{c}{a_j} - \frac{(\mathbf{r}_j \cdot \mathbf{v}_j)}{\|\mathbf{r}_j\|} \right] \quad (32)$$

All of the summands in the numerator on the right hand side of Eq.(32) clearly need to have dimensions of length, which is asserted by the  $\|\mathbf{r}_j\|$ . The quantity on the left hand side,  $\delta t$ , is a time, implying that the units of the denominator on the RHS must have dimensions of length/time. So, we can fix our velocity units to be km/s, and apply conversions to our times and lengths, or vice-verse. The latter is the obvious choice, since we should certainly like to keep our lengths as a comoving unit, and our time units defined with respect to the scale factor  $a$ .

$\tau$  is the temporal width of the timestep bounded for snapshots  $j$  and  $j + 1$ , which is supplied in HACC via the `TimeStepper` class as a dimensionless *program time* (henceforth `pt`), which is some power of the scale factor; `pt`  $\equiv a^\alpha$ , where  $\alpha$  is typically chosen to be  $1^p$ . Given that our distances are in units of Mpc/h, our velocities must be scaled to units of (Mpc/h)/`pt`, including that of  $c$ . Note that occurrences of  $H$  and  $\dot{a}$  must also undergo this conversion, since their unit definitions contain km/s, as given by the `TimeStepper` and Eq.(15).

Carrying out the dimensional analysis of Eq.(32) with these changes does indeed give the units of  $\delta t$  as dimensionless program time, which is desired. For further convincing, unit analyses can be performed on the entirety of Eq.(26), and also Eq.(29-31), to prove that these dimension choices are correct for the implemented quadratic solution, as well as the symmetric calculation and weighted average  $\delta t$ .

Now, the conversion described above turns out to be slightly involved. Assuming that we have velocities in km/s, then we need a conversion factor with units of s/(h·pt). Multiplying our velocities by this value should produce the dimensions we are after.

---

<sup>p</sup> $\alpha$ , here, is an input parameter given by the user running the simulation. To understand the choice to use a power of the scale factor as the program time variable, see the RRU Simulation and Code Overview.

We can start by trying to obtain an expression relating physical time to program time; we know that

$$H = \frac{\dot{a}}{a} = \frac{da}{dt} a^{-1}, \quad (33)$$

and we can use this to find the operator  $d/dt$ :

$$\begin{aligned} aH &= \frac{da}{dt} \\ dt(aH) &= da \\ \frac{d}{dt} &= aH \frac{d}{da} \end{aligned} \quad (34)$$

which is what we stated in Eq.(7). We can use the presence of the scale factor  $a$ , here, to involve the program time  $pt \equiv a^\alpha$  via the chain rule:

$$\frac{d}{da} = \frac{d}{dpt} \alpha a^{\alpha-1}. \quad (35)$$

Combining Eq.(34) and Eq.(35) leaves us with

$$\frac{d}{dt} = aH\alpha a^{\alpha-1} \frac{d}{dpt}. \quad (36)$$

Eq.(28) has a temporal differential on each side, and acting those operators on a generic position  $r$  gives us

$$\frac{dr}{dt} = v_{\text{phys}} = aH\alpha a^{\alpha-1} \frac{dr}{dpt} = aH\alpha a^{\alpha-1} v_{\text{pt}} \quad (37)$$

which makes it clear that our conversion factor must be

$$\frac{1}{aH\alpha a^{\alpha-1}} \quad (38)$$

and must have units of s/pt (since the units of  $v_{\text{phys}}$  and  $v_{\text{pt}}$  are km/s and km/pt, respectively). We must not forget that we'd also like our distances in comoving coordinates, so finally we multiply the conversion factor Eq.(38) by  $h/h$ , where  $h$  in the numerator becomes part of the value, and the  $h$  in the denominator becomes part of the units (reminded by the italics, or lack thereof). In summary, we learn that we can convert our physical velocities (including  $c$ ) to dimensions of [comoving distance]/[program time] by multiplying them by

$$\frac{h}{aH\alpha a^{\alpha-1}} \frac{\text{s}}{h \cdot \text{pt}} \quad (39)$$

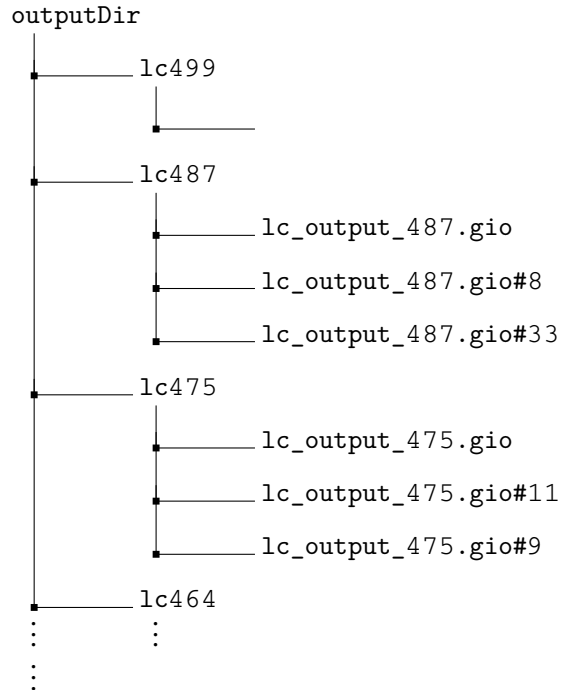
and also that, we can't forget, we can also convert our values of  $H$  and  $\dot{a}$  to the proper units by multiplying by Eq.(38). This is precisely what the lightcone solver does; in `pa.update_lc()`, the variable `s_pt` holds the conversion factor obtained from evaluating Eq.(38), and likewise the variable `scale_vel` for Eq.(39). Any outstanding opaqueness should be able to be cleared via a careful reading of the in-line documentation, and a sufficient understanding of HACC's coordinate conversion utilities (namely, the member functions of `ParticleCoords`, which make use of the various `Domain::grid2phys_()` offerings).

## 4.5 Output

The output of the lightcone code looks very much like simulation snapshot output; GIO files are written out for each snapshot that the code is run on, with a data row per particle/object. Rows have at most 17 columns, each corresponding to some property of the object, which are enumerated in Table 4. The structure of the output directory is fully dependent on the nature of the job submission script written by the user (where parameter 2 of Table 1 is defined), though the convention is to end up with something that looks like Figure 6.

Most of the output quantities described in Table 4 are sufficiently clear, though two of them, `rotation` and `replication` require further explanation. The former gives

**Figure 7:** The conventional output directory structure resultant from running the lightcone solver on at least the first four snapshots of some simulation which ran through a total of 500 full timesteps. The hashed (#) filenames indicate portions of the output that were written by different MPI ranks; only the unhashed filename (the “header”) need be passed to GIO in order to read the output. Notice that step 499 has no lightcone output—this makes sense; step 499 corresponds to  $z = 0$ , which is the time of observation. The lightcone volume, then, at step 499 is of course zero (setting  $a_{\text{event}} = 1$  in Eq.(8) gives  $\|\mathbf{r}\| = 0$ , telling us that the only events that the observer can see at  $z = 0$  are those that have zero spatial separation from them self).



an identifier, for each object, denoting the coordinate transformation that was applied to the simulation replication which hosts the object. This identifier is an integer  $i$  where  $i \in [0, 5]$ . The meaning of those values are as follows:

Table 3

value of rotations	0	1	2	3	4	5
transformation applied	none	$x \leftrightarrow y$	$y \leftrightarrow z$	$x \leftrightarrow y,$ $y \leftrightarrow z$	$z \leftrightarrow x$	$x \leftrightarrow y,$ $z \leftrightarrow x$

where the “swap operator”  $\leftrightarrow$  acting on two dimensions  $n$  and  $m$  is  $n \leftrightarrow m = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} n \\ m \end{pmatrix}$ , and a comma between two of these transformations means “followed by”.

The latter output product, `replication`, is again some integer identifier for each object, this time specifying the box replication to which the object belongs. This is useful information to have because, for example, one may find multiple particles, let’s

call them  $p$  and  $q$ , with the same id in some lightcone output. If  $p$  resides in the initial simulation volume, while  $q$  resides in some distant replication of the volume, then all is well ( $p$  and  $q$  are effectively different objects)<sup>9</sup>. If  $p$  and  $q$  reside in the *same* volume, however, then that would be indicative of a bug of the § 2.2 variety. In general, someone doing analysis with lightcone catalogs may need this information.

So, we give each replication a unique 32-bit integer identifier, within which is packed it's spatial coordinates in units of box-lengths. Each coordinate ( $x$ ,  $y$ ,  $z$ ) is allotted ten of these bits (2 bits leftover), with the specific anatomy of the value shown in Figure 7. Note that, under this scheme, the initial volume containing the observer is *not* at  $(0,0,0)$ . Rather, the origin of this setup is at the positive corner replication (for example, the top-right box in Figure 5), which puts the initial volume at  $x = y = z = \text{LCReplicants}$ . This is done to avoid having to bit-shift negative signed integers.

**Figure 8:** The anatomy of the 32 bit integer, `replication`, as found in the lightcone output. The first three sets of ten bits, starting from the left, are the  $x$ ,  $y$ , and  $z$  positions of the replication, respectively, in units of box lengths. In this example, the considered replication is found at  $(2, 1, 1)$ . There are two unused trailing bits.

$$\text{replication} = 00 \underbrace{0000000010}_x \underbrace{0000000001}_y \underbrace{0000000001}_z$$

One can therefore easily extract the coordinates of each box replication by bit shifting and masking:

```
x = replication >> 20
y = (replication >> 10) & 0x3ff
z = replication & 0x3ff
```

It is perhaps worth noting that, since each box coordinate is allowed a maximum of ten bits, we must have  $\text{LCReplicants} \leq 1024$  for an octant sky, and  $\text{LCReplicants} < 512$  for a full-sky lightcone. This is no problem, as our number of replications will never be nearly this large; even with a small simulation volume of 256Mpc/h, a lightcone built to utilize all 512 box replications along it's radial axis would reach far beyond the redshift limits of typical simulation output (where  $z \leq 200$ ).

Table 4: The output properties for each particle/object that passed through the lightcone solver

	Output Column	Type <sup>r</sup>	Description
1.	x, y, z	float	The cartesian x, y, or z position, respectively, of the object at the time $t_j + \delta t$ (given by either interpolation, § 2.3, or extrapolation, § 2.1) in comoving Mpc/h.

*Continued on next page*

<sup>9</sup>in fact, one could expect up to as many duplications of particle  $p$  as there are simulation boxes, that is, `numBoxesTotal` as computed on pg. 16. The values of `replication` for each of these particles should be unique.

Table 4 – Continued from previous page

	Output Column	Type <sup>r</sup>	Description
2.	vx, vy, vz	float	The x, y, or z velocity components, respectively, of the object at the time $t_j + \delta t$ (given as $v_{\text{linear}}$ if interpolation was used (§ 2.3) or as $v_j$ if extrapolation was used (§ 2.1)) in physical km/s.
3.	id	int	The id of the object as found in the input object catalog (perhaps particle ids coming from snapshot files, or halo tags coming from merger trees, etc.).
4.	a	float	the scale factor $a(t)$ at the time the object crossed the lightcone, $t = t_{\text{event}} = t_j + \delta t$ .
5.	step	int	The simulation snapshot from which this object originated (where it's position at time $t_{\text{event}}$ was approximated from).
6.	rotation	int	An identifier giving the coordinate rotation that was applied to the simulation box replication in which this particle resides. Possible values are 0-5, which are discussed in the text.
7.	replication	int	A unique identifier specifying which simulation box replication this object resides in. A method for extracting information about the spatial position of this box replication identifier is given in the text.
8.	mask	int	A 16 bit mask, where each bit represents the state of some flag relevant to the simulation run. In the lightcone output, this value is simply set to 1 for all particles, though it could conceivably be used to specify flags that may be useful.
9.	phi	float	The scalar potential used by the gravity solver during the simulation run. This value is not useful in lightcone outputs, and is set to 0 for all entries. The column is retained to maintain consistency with the form of particle snapshot outputs.
10.	mass	float	The mass of the object, in the case that the code was run on a halo catalog or merger tree.
11.	origX, origY, origZ	float	The cartesian x, y, or z position, respectively, of the object at the time $t_j$ (the objects original position in the input catalog snapshot before being permuted by the lightcone solver) in comoving Mpc/h.

<sup>r</sup>The types listed here are also only conventional, and can be changed by the user. Positions and velocities, for example, can alternatively be given as `doubles`. Look toward `common/HaccTypes.h` for some relevant type declarations.

## 5 Too Long; Didn't Read

Assume an observer located at the spatial origin of a comoving coordinate system. Any event located on this observer's past-lightcone will satisfy the condition

$$\int_{a_{\text{event}}}^1 c \frac{da}{a^2 H} = \|\mathbf{r}\|. \quad (8)$$

where  $a$  is the scale factor,  $H = \dot{a}/a$  (the Hubble parameter), and  $\mathbf{r}$  is the displacement vector from the observer to the event. Refer to § 1 for a full derivation. Now, consider such an event in the context of cosmological simulation output; a particle crosses the lightcone somewhere between snapshot  $j$  and snapshot  $j+1$  (constituting a timestep of width  $\tau = t_{j+1} - t_j$ ) at an unknown time  $t_j \leq t_{\text{event}} < t_{j+1}$ . We define a quantity  $\delta t$  such that  $\delta t = t_{\text{event}} - t_j$ . If we assume that  $\delta t \ll 1^s$ , then we can approximate, to second order, Eq.(8) using quantities available to us as simulation outputs, as

$$\Theta_{j+1} + \frac{c}{a_j} \left[ (\tau - \delta t) - \frac{\dot{a}_j (\tau^2 - \delta t^2)}{2} \right] = \|\mathbf{r}_j\| + \delta t \frac{(\mathbf{r}_j \cdot \mathbf{v}_j)}{\|\mathbf{r}_j\|} - \frac{\delta t^2 (\mathbf{r}_j \cdot \mathbf{v}_j)^2}{2 \|\mathbf{r}_j\|^3} + \frac{\delta t^2 (\mathbf{v}_j \cdot \mathbf{v}_j)}{2 \|\mathbf{r}_j\|} \quad (18)$$

where  $\Theta_{j+1}$  is the Simpson's-rule numerical evaluation of a definite integral similar to that given above in Eq.(8):

$$\Theta_{j+1} \equiv \text{Simpson} \left( \int_{a_{j+1}}^1 c \frac{da}{a^2 H} \right). \quad (16)$$

We can then solve for  $\delta t$  via the quadratic formula:

$$\alpha \cdot \delta t^2 + \beta \cdot \delta t + \gamma = 0 \quad (19)$$

$$\implies \delta t = \frac{-\beta \pm \sqrt{\beta^2 - 4\alpha\gamma}}{2\alpha} \quad (20)$$

where we have

$$\alpha = \frac{c\dot{a}_j}{2a_j^2} + \frac{1}{2} \left( \frac{(\mathbf{r}_j \cdot \mathbf{v}_j)^2}{\|\mathbf{r}_j\|^3} - \frac{(\mathbf{v}_j \cdot \mathbf{v}_j)}{\|\mathbf{r}_j\|} \right) \quad (21)$$

$$\beta = -\frac{c}{a_j} - \frac{(\mathbf{r}_j \cdot \mathbf{v}_j)}{\|\mathbf{r}_j\|} \quad (22)$$

$$\gamma = \Theta_{j+1} + \frac{c\tau}{a_j} - \frac{c\dot{a}_j\tau^2}{2a_j^2} - \|\mathbf{r}_j\| \quad (23)$$

The ultimate utility of the lightcone code is to compute  $\delta t$  for every particle at every step of the simulation (find each particle's lightcone-crossing time). The solution presented in Eq.(20-23) is derived by extrapolating the position of a given particle (as

<sup>s</sup>This approximation is allowed in the case that we use a normalized time unit, such as the scale factor  $a(t)$ , or more specifically a power of  $a$  called the program time;  $a^\alpha$



seen on the right hand side of Eq.(18)), using it's velocity, from time  $t_j$  (known) to time  $t_j + \delta t$  (unknown). If it is found, after solving for  $\delta t$ , that  $0 < \delta t \leq \tau$ , then the result is saved. If rather  $\delta t > \tau$  or  $\delta t \leq 0$ , the result is discarded (the first case means we will almost certainly obtain a better estimate of  $\delta t$  if we instead extrapolate from time  $t_{j+1}$  during the next step of the lightcone construction, and the second case, that the particle should have already crossed the lightcone at some earlier time). This procedure is discussed at length in § 2.1.

For a cleaner result, we can interpolate the position of the particle between snapshot  $j$  ( $t_j$ ) and snapshot  $j+1$  ( $t_{j+1}$ ), rather than extrapolating. This simply requires that we replace the velocity  $\mathbf{v}_j$  in Eq.(18) with the equivalent linear velocity

$$\mathbf{v}_{\text{linear}} = \frac{\mathbf{r}_{j+1} - \mathbf{r}_j}{\tau}, \quad (24)$$

as more thoroughly described in § 2.3. We also discuss the possibility of relaxing the assumption that  $\delta t \ll 1$ , and solving the integral in Eq.(8) entirely numerically. There are a few shortcomings of the approximated solution described above which inspire such an idea, including various discreteness artifacts and inaccuracies in the lightcone output. We have come up with a few workarounds to these problems, which are presented along with remarks about potential directions for improvement, in § 3.

Lightcone construction is implemented in HACC in `nbody/cpu/ParticleActions.cxx`, and driven by `nbody/simulation/driver_lc.cxx`, or `driver_lc_interpolation.cxx`, for the extrapolation (§ 2.1) and interpolation (§ 2.3) approaches, respectively. The driver's `main()` function expects at most six arguments, as given below in § 4.1, Table 1, which is shown below (where only the interpolation driver requires argument 5). There are also many other run parameters that can be set by the user in a `.param` file, several of which are ultimately crucial to the structure of the lightcone output. Not all of these parameters will be visited in this summary section, but are given in Table 2, and discussed in high detail in § 4.1. Briefly, those parameters which are the most influential to the spatial configuration of the problem are `LCA110ctants`, `LCReplicants`, and `LCRotate`. The purpose of these three options can be understood under the follo-

1. <code>&lt;paramName&gt;</code>	path to a HACC param file
2. <code>&lt;outName&gt;</code>	output file path
3. <code>&lt;inName&gt;</code>	input file path for snapshot $i$ particle data
4. <code>&lt;stepNumber&gt;</code>	step number $i$
5. <code>&lt;nextInName&gt;</code>	input file path for snapshot $f$ particle data
6. <code>&lt;nextStepNumber&gt;</code>	step number $f$

Table 1: Input parameters for the lightcone driver's `main()` function

wing considerations: we are limited in the extent to which any observer's lightcone can reach (how far back in time they can see) by the dimensions of our simulations volume. For example, if we are interested generating the observable universe to a redshift of  $z = 3$ , then our observer's past-lightcone will be a spherical object whose radius is  $\sim 4.6\text{Mpc}/h$ , assuming a WMAP7 cosmology. To get around the fact that we almost never deal with simulations larger than  $4\text{Mpc}/h$  (and often much smaller) we decide to replicate and tile the box in space to cover the volume that we require (as thoroughly detailed in § 4.2, and seen in Figure 3).

The option `LCReplicants` (Table 2, option 3), then, sets the number of simulation boxes required to reach the extent of the lightcone in one dimension, not including the initial volume. `LCRotate` (Table 2, option 7) is a boolean value turning on or off the functionality of applying random rotations to each of these replications, which intends to avoid regular repetition of large-scale structures. Finally, `LCAllOctants` (Table 2, options 2) allows the user to specify the angular domain of the observer, from octant to full-sky. Again, more details are offered in § 4.1-§ 4.3. Those seeking to run the code and build lightcones should at least read through Table 2.

The output of the presently discussed code is given in Table 4, immediately preceding this summary section, with finer details being discussed in the text of § 4.5. In summary, the output includes the following: the scale factor corresponding to the calculated lightcone crossing time,  $t_{\text{event}} = t_j + \delta t$ ; the approximated positions of each object at the time  $t_{\text{event}}$ ; velocities (also approximated in the case that interpolation was used); and ids. Also written out is information on the location and rotation of the simulation box replication in which each object is found, and other optional and/or incidental data products.

## 6 Acknowledgements

The work described over the previous pages was conducted at Argonne National Laboratory, Cosmological Physics and Advanced Computing group. The original code design and implementation was done by Hal Finkel, upon which our work has been built. Significant extensions and improvements were completed by Joe Hollowed beginning in December 2017, with indispensable collaboration with Patricia Larsen and Steve Rangel. Guidance throughout the project was given by Katrin Heitmann and Salman Habib, as well as valuable input from other members of CPAC; Lindsey Bleem, JD Emberson, Danila Korytov, and Nesar Ramachandra. Please send any questions/comments to [jphollowed@anl.gov](mailto:jphollowed@anl.gov), or post to the group's general Basecamp at `Docs & Files > HACCLightcones_vX.X.pdf`.