Nancy Grace Roman Space Telescope (Roman) Technical Report

Title: Simulating Cosmic Rays for the Roman Wide Field Instrument	Doc #: Roman-STScI-000502, SC-01 Date: April 22, 2023 Rev: -
Authors: **John Wu**, Javier Sanchez, Stefano Casertano, Tyler Desjardins Phone: (410) 338-4575	Release Date: May 9, 2023

1 Abstract

This technical report describes an algorithm that simulates cosmic ray events for the Roman Wide Field Instrument. We present a framework for sampling cosmic ray properties based on simulated and empirical data, and depositing cosmic ray hits onto the detector. We compute the typical jump in counts per cosmic ray event, and compare against preliminary JWST data. We also benchmark the algorithm's computation time scaling. Finally, we discuss current limitations of our method, and consider future updates once JWST cosmic ray data at L2 are fully analyzed. We envision that this cosmic ray algorithm will be useful as part of a suite of simulation tools for scientific studies and for testing the calibration software ahead of the Roman mission.

2 Introduction

The Roman Space Telescope Wide Field Instrument (WFI) will transform observational astronomy, exoplanet science, time domain astrophysics, and cosmology by providing astronomical survey imaging at exquisite angular resolution. The WFI comprises 18 near-infrared H4RG-10 detectors and delivers 288 megapixels over a 0.281 deg^2 field of view at 0.11 arcsec pixel^{-1} resolution. Additional information about the H4RG-10 detectors can be found in the detailed report by Mosby et al. (2020). Older generation (H2RG) detectors reside aboard JWST as part of the NIRCam, NIRSpec, and NIRISS instruments, and their on-flight performance is presently being characterized through the analysis of commissioning and early science data.

In order to accomplish its core science objectives, Roman must be able to measure faint astronomical sources in the presence of distortions or contaminating signals. These contaminants may originate from the instrument electronics, such as $1/f$ noise, dark current, or detector persistence, or from unrelated astronomical phenomena, such as scattered light or high-energy particles. We will exclusively focus on these high-energy particles, or cosmic rays (CRs), for the rest of this technical report.

CRs are relativistic charged particles produced by the Sun or by energetic events within and even outside our Galaxy. As they pass through the detector, CRs lose energy by ionizing the detector medium and depositing energy along their path at different angles of incidence. CR hits produce trails ranging from a single pixel to hundreds of pixels as they pass through the detector. In the energy range relevant for near-infrared detectors, the CR flux can be empirically measured although, as we will discuss later, this CR flux changes in response to the solar weather.

There are various algorithms for removing CR hits from detectors, e.g. `LACosmic` (van Dokkum 2001), modules from the JWST and Roman Space Telescope Calibration Pipeline (Bushouse et al. 2023; Romancal Developers 2022), and other data-driven and machine learning methods (e.g., `deepCR`; Zhang & Bloom 2020). CR simulations are critical for comprehensively testing count rate estimation algorithms (i.e., up-the-ramp fitting; Casertano 2022), jump detection algorithms (Anderson & Gordon 2011, Casertano et al., *in prep.*), and other parts of the calibration pipeline (e.g., Giardino et al. 2019). In order to assess CR removal and calibration workflows on the Roman WFI detectors, we need an algorithm to generate realistic CR trails. We present a fast, empirically motivated approach for generating CR parameters and adding their signatures to a single WFI detector.

In this technical report, we focus on injecting CR signals as part of a larger set of Roman WFI simulation tools. In Section 3, we provide an overview of the physics and observed properties of CRs. In Section 4, we present an algorithm for simulating cosmic rays onto a single detector. In Section 5, we provide example Python code and discuss some results. In Section 6, we estimate typical jumps in the count rate due to cosmic rays, and compare our simulations with early JWST data. Finally, in Section 7, we provide a summary of the algorithm, discuss its current limitations, and present future directions.

3 A Primer on Cosmic Rays

There are at least two major sources of CRs: solar and galactic. Somewhat confusingly, the latter category includes CRs from not just the Milky Way but also extragalactic sources. Solar CRs are typically found at lower energies, while galactic CRs—which may have been accelerated through magnetic fields around shocks and supernovae millions of years ago—can extend to far higher energies. It is expected that most of the lowest-energy CRs ($\lesssim 10$ MeV) will be adequately shielded by the Roman Solar Array Sun Shield (e.g., Robberto 2009), and therefore the WFI will predominantly encounter galactic CRs.

The solar and galactic CR fluxes anticorrelate with each other (e.g., Barth et al. 2000). In other words, the solar CR flux is high when the galactic CR flux is low, and vice versa. During peak sunspot activity in the 11-year solar cycle, the strong solar wind deflects a larger fraction of galactic CRs and decreases the total CR flux seen by the observatory (for a review, see Potgieter 2013). At solar minimum, simulations suggest that the CR flux passing through 2.5 mm aluminum shielding is of order $1-10$ events cm^{-2} s^{-1} (Robberto 2009). The Sun will occasionally experience solar flares and coronal mass ejections, accompanied by a blast of high-energy CRs; these events can increase the CR flux by three orders of magnitude. It is assumed that Roman WFI will not observe under such conditions, so we are not concerned about validating our simulations during extremely high CR fluxes.

Our understanding of CRs is primarily based on historical measurements within hundreds of kilometers from the surface of Earth, often using charge-coupled device (CCD) detectors (e.g., Fisher-Levine & Nomerotski 2015). However, the upper atmosphere and the van Allen belts significantly modulate the CR properties, and differences between CR populations near Earth and at L2 are still not well-constrained. We anticipate that JWST analyses with up-to-date CR measurements will supersede HST observations and simulated CR data (Robberto 2009; Miles et al. 2021), which form the basis of CR properties assumed in this work. Therefore we have designed our algorithm in Section 4 to be modular and free of hard-coded values, so that it can be updated to include the latest CR measurements at L2 (e.g., supplied as part of a future reference file).

There are many programs available for simulating CRs and their passage through various media. For example, `CREME96` provides 1-D numerical models for solar and galactic CR populations (Tylka et al. 1997). `SRIM` (Stopping and Range of Ions in Matter, Ziegler et al. 2010) can simulate simple 1-D models for Coulomb interactions between high-energy ions and other media. Robberto (2009) present a library of CR events simulated using `SRIM` for various solar and galactic CR fluxes. We note that more sophisticated models are also now available, such as `Geant-4`, which can simulate in 3-D the secondary particles produced by high-energy events as well as their impacts on science instruments (Agostinelli et al. 2003; Allison et al. 2016); however, this level of realism is beyond the scope of our current work.

4 Simulating Cosmic Rays

We present an overview of our CR algorithm in Section 4.1. We discuss some of the assumptions and rationale behind the empirical CR path length distribution and fiducial CR energy loss distribution in Sections 4.2 and 4.3, respectively. In Section 4.4, we describe the algorithm for drawing CR trails onto the detector.

4.1 Sampling algorithm overview

I. We determine the average number of CR events μ_N for some exposure time, given a constant CR flux and detector area:

$$\mu_N = \left(\frac{\text{flux}}{\text{cm}^{-2}\ \text{s}^{-1}}\right) \times \left(\frac{\text{area}}{\text{cm}^2}\right) \times \left(\frac{\text{time}}{\text{s}}\right), \tag{1}$$

and then sample a number of events from a Poisson distribution centered on this value $N \sim \text{Poisson}(\mu_N)$.

II. We draw random samples for CR parameters, with default hyperparameter values provided below. Detector positions are 0-indexed. For each CR event, we sample:

(a) a random location on the detector $i, j \sim \text{Uniform}(-0.5, 4095.5)$[1],

(b) the CR path angle $\alpha \sim \text{Uniform}(0, 2\pi)$,

(c) the CR path length x in units of μm, sampled from a power-law distribution with slope of -4.33 (Miles et al. 2021; see Section 4.2),

(d) the CR energy loss dE/dx in units of eV μm^{-1}, sampled from a Moyal (1955) distribution centered at 120 eV μm^{-1} and a width parameter of 50 eV μm^{-1} (see Section 4.3).

III. For each event, we distribute electrons in detector pixels across the full CR trail, assuming a zero-width trail. The number of electrons per pixel is drawn from a Poisson distribution whose mean value scales with the traversed path length.

In Figure 1, we display sampled CR parameters for 10^6 events. Panel (a) shows CR positions on a single 4096×4096 WFI detector. Panel (b) shows the distribution of orientation angles of the CR trails, and panel (c) shows the distribution of CR trail lengths. Panel (d) shows the distribution of energy losses for the full CR trail. We use inverse transform sampling (with linear interpolation) in order to draw samples from the CR length and energy loss distributions. Although it is likely that the CR lengths and energy losses are correlated, we independently sample from the two distributions.

4.2 Distribution of path lengths

Miles et al. (2021) analyzed 1.2 billion cosmic rays captured over 25 years of HST observations spanning two solar cycles. They fit the empirical power-law distribution of CR path lengths using calibration dark frames for the STIS, ACS/HRC, ACS/WFC, WFPC2, WFC3/UVIS instruments. We note that these are all CCDs rather than near-infrared detectors, and that the HST orbital environment gives rise to a different CR population than what is expected

[1]We use i, j to represent real numbers, not only integers.

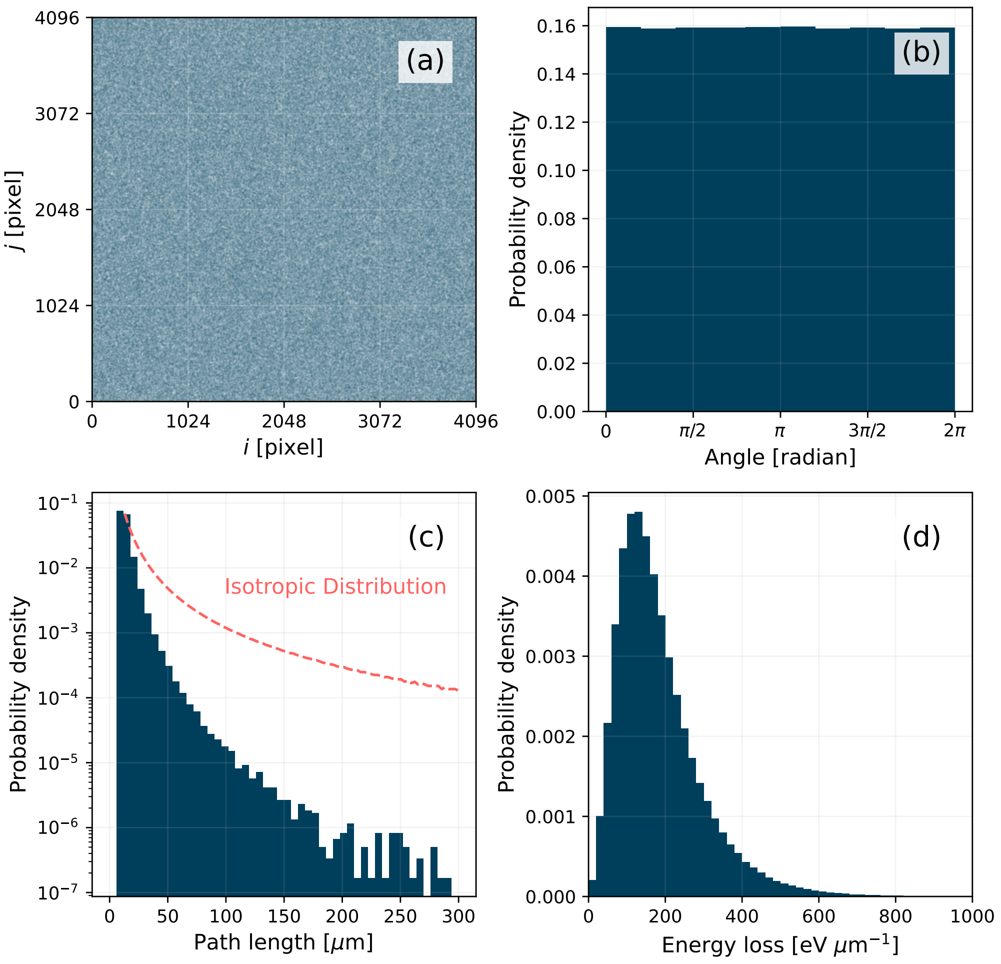

Figure 1: Distributions of 10^6 simulated CR parameters: (a) positions, (b) orientation angles, (c) trail lengths, and (d) energy losses. In panel (c), we show the distribution of path lengths under the assumption of isotropic CRs in the red dashed line. Note that the y-axis in this panel is displayed in logarithmic scale.

for Roman at L2. The best-fit power-law model has a log-log slope of -4.33, and slight variations are seen across different detectors. This slope can be interpreted as the effective shielding strength, and it is known to vary for different instruments depending on their location in the focal plane. We adopt their power-law fit for sampling CRs (shown as the blue histogram in panel (c) of Figure 1), even though the L2 population of CRs will likely differ.

The path length distribution can also be analytically computed under the assumption of an isotropic distribution of CRs. The path lengths can then be computed using the angle of incidence θ and the pixel depth ($z = 5\ \mu$m). If we sample an isotropic distribution of angles over a unit hemisphere: $\theta = \arccos(1-u)$, where $u \sim \text{Uniform}(0, 1)$, then θ will more frequently take on values near $\pi/2$ than 0. We can determine the geometric path length

$x = z \tan \theta$, or

$$x = (5\ \mu\text{m}) \tan \left[\arccos(1 - u)\right]. \tag{2}$$

This distribution is shown in the red dashed line in panel (c) of Figure 1, and we find that a strong tail of large CR path lengths emerges from the isotropic CR assumption. In practice, strong shielding at large angles of incidence can suppress these CRs with large path lengths. Moreover, the orientation of the observatory relative to solar and galactic CR sources will be anisotropic, which will further modulate the CR path length distribution. Therefore, we continue using the empirical path length distribution sampled from $x^{-4.33}$ determined by Miles et al. (2021).

We cut off the power-law distribution at a fiducial minimum length because the cumulative number counts will diverge as the minimum length approaches zero. We note that altering this minimum cut-off can drastically change the relative fraction of CR-affected pixels. By default, this minimum value is set to $10\,\mu$m, the size of a WFI pixel, but in reality it will depend on the shielding characteristics and empirical distribution of CR trajectories. We also impose a maximum path length with a default value of $10^4\,\mu$m, although such events are rare enough that it does not impact the final result by much.

4.3 Distribution of energy losses

We assume that the probability distribution of CR energy losses follows a Moyal (1955) distribution :

$$M(\Delta) \sim \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}\Big(\Delta + \exp(-\Delta)\Big)\right), \tag{3}$$

where $\Delta = (dE/dx - \mu)/w$ is the energy loss dE/dx re-centered on a mean value μ and re-scaled by a width parameter w. The Moyal function is a skewed (long-tailed) probability distribution, as can be seen in panel (d) of Figure 1. By default, we adopt $\mu = 120$ eV μm^{-1} and $w = 50$ eV μm^{-1} based on simple estimates from simulated (Robberto 2009) and observed CR distributions (Miles et al. 2021; Rauscher 2022). For comparison, `SRIM` simulations for typical CRs in the 1000 MeV per nucleon range lose energy at rates of ~ 100 eV μm^{-1} (these rates are approximately constant with depth; see Figure 8 of Robberto 2009).

Let us briefly consider Equation 3 and the rationale for selecting such a functional form. The probability of ionization energy losses for a relativistic charged particle traversing a detector can be described by a Landau-Vavilov distribution to within a few percent error (see Section 27 of Amsler et al. 2008, and see also Section 4.8.2 of Miles et al. 2021). This approximation breaks down for very low-energy particles or for very thin detector media. However, the Landau-Vavilov distribution cannot be written in an analytical form and must be numerically integrated. In order to avoid extra computation (and because our goal is not to generate a fully realistic population of CRs), we further approximate the CR energy loss distribution using the Moyal (1955) function. Qualitatively, these distributions are all very similar: they have strong peaks, resulting in a fairly constant range of energy losses, with the exception of a tail toward higher energy loss values. Other simulators, such as WFISim (Bellini et al. 2022), use a flat CR energy distribution, or sample from the Robberto (2009) simulated library of CR events (as MIRAGE does; Hilbert et al. 2019).

In order to convert from energy to electron counts (e^-), we use the H4RG-10 bandgap energy, $E_g = 0.5$ eV counts^{-1} (based on a Cadmium mole fraction of 0.445 and operating temperature of 95 K, Hansen et al. 1982). Using this assumption, a typical energy loss of 120 eV μm^{-1} corresponds to a deposited value of 2400 e^- for a Roman pixel pitch (size) of 10 μm. In comparison, preliminary JWST NIRSpec CR data exhibits a mode at 4250 e^- over its 18 μm pixels with bandgap energy $E_g = 0.225$ eV counts^{-1} (Rauscher 2022; see also discussion in Section 6.2).

4.4 Drawing straight-line trails

We provide an algorithm for drawing straight-line trails through the detector. If we assume that the CR hit maintains a constant energy loss throughout its trail, then the value of any pixel intersected by the CR trail should be proportional to its traversal length. We briefly outline our traversal algorithm, which is similar to the Fast Traversal Algorithm (Amanatides & Woo 1987):

1. Receive as input the starting coordinates (i_0, j_0) and ending coordinates (i_1, j_1), in fractional units of pixels. Note that these are floating-point numbers, and we assume that the detector grid is 0-indexed. The start and end are re-arranged such that $i_1 > i_0$.

2. Compute $\Delta j = i_1 - i_0$ and $\Delta j = j_1 - j_0$, such that $\Delta j/\Delta i$ is the slope of the trail, and $s = \text{sign}(\Delta j)$ tracks whether the slope is positive or negative.

3. Compute the coordinates where the trail intersects horizontal (half-integer j) and vertical (half-integer i) pixel border crossings. For example, to determine vertical border crossings, we generate the sequence of integer values $\lfloor i \rceil$ between the starting and ending coordinates and add a half; we can then compute the j values using $\Delta j/\Delta i$:

$$i_{\times,\text{vert}} = 1/2 + \left(\lfloor i_0 \rceil, \quad \lfloor i_0 \rceil + 1, \quad \cdots, \quad \lfloor i_1 \rceil\right), \tag{4a}$$
$$j_{\times,\text{vert}} = j_0 + \Delta j/\Delta i \cdot (i_{\times,\text{vert}} - i_0)\,, \tag{4b}$$

   where $\lfloor\rceil$ is an operator that rounds to the nearest integer, and standard array broadcasting rules apply here.[2] The $_{\times,\text{vert}}$ subscript simply denotes that these are the pixel locations for vertical border crossings. If we were to compute the horizontal border crossings, then we need to increment or decrement the half-integer $j_{\times,\text{horiz}}$ depending on s:

$$j_{\times,\text{horiz}} = s/2 + (\lfloor j_0 \rceil, \quad \lfloor j_0 \rceil + s \cdot 1, \quad \cdots, \quad \lfloor j_1 \rceil) \tag{5a}$$
$$i_{\times,\text{horiz}} = i_0 + \Delta i/\Delta j \cdot (j_{\times,\text{horiz}} - j_0) \tag{5b}$$

4. Record the integer center of each traversed pixel using the most recently crossed boundary. Since Δi is positive by construction in step 1, we can write the center of all traversed pixels, except the ending pixel, as:

$$\begin{pmatrix} i_{\text{t,vert}} \\ j_{\text{t,vert}} \end{pmatrix} = \begin{pmatrix} \lfloor i_{\times,\text{vert}} - 1/2 \rceil \\ \lfloor j_{\times,\text{vert}} - s \cdot \epsilon \rceil \end{pmatrix}, \tag{6}$$

[2]See Numpy broadcasting rules at https://numpy.org/doc/stable/user/basics.broadcasting.html.

where ϵ is a small constant introduced for stable numerical rounding. Similarly, we can compute the pixel centers prior to horizontal boundary crossings:

$$\begin{pmatrix} i_{\rm t,horiz} \\ j_{\rm t,horiz} \end{pmatrix} = \begin{pmatrix} \lfloor i_{\times,\rm horiz} - \epsilon \rceil \\ \lfloor j_{\times,\rm horiz} - s/2 \rceil \end{pmatrix}. \tag{7}$$

5. Merge the horizontal and vertical crossing coordinates into a single set of coordinates. Then, sort by i-axis, add the ending pixel centers, and remove pixels that fall outside the detector border.

6. Calculate traversal distances using the hypotenuse of the triangle formed between border crossing locations (and the starting and ending coordinates). If no borders are crossed, then the traversal distance is just the Euclidean distance between the starting and ending coordinates.

We note that drawing a line onto a grid (i.e. rasterization) is a common computer graphics problem. However, rasterization is primarily used to create visually pleasing graphics, rather than for simulations, and it relies on different techniques than what we have employed (e.g., anti-aliasing; Wu 1991; Zingl 2012). For our algorithm used to deposit CR signal onto a detector grid, we assume a zero-width trail and need to keep track of each pixel segment's traversal distance, which is typically not done in computer graphics.

5 The cosmic ray algorithm

5.1 An implementation in Python

We demonstrate a Python implementation of the CR algorithm that uses the Numpy/Scipy software stack. For readability purposes, we place the code listing in Appendix A. The assumed CR properties are not hard-coded into the program, with the exception of the uniform distributions of CR locations and orientations. All other values are encoded as default (but modifiable) hyperparameters as part of the functions defined in the program.

5.2 Simulating cosmic rays in practice

In the listing below, we show some code demonstrating the algorithm in action. The WFI area, pixel size, and pixel depth are not expected to change, and are implicitly set to their default values of 16.8 cm^{-2}, 10 μm, and 5 μm, respectively. In the imaging mode, WFI reads have an exposure time of 3.04 s, and we simulate 500 separate read frames. Because each simulated read requires a Poisson draw of (integer-valued) CR events, the total number of CR hits will be slightly different than if we instead simulate a single long resultant frame with $500 \times 3.04 = 1520$ s exposure time.

```python
wfi_image = np.zeros((4096, 4096), dtype=int)

cr_flux = 8       # events/cm^2/s
wfi_area = 16.8   # cm^2
t_exp = 3.04      # s
```

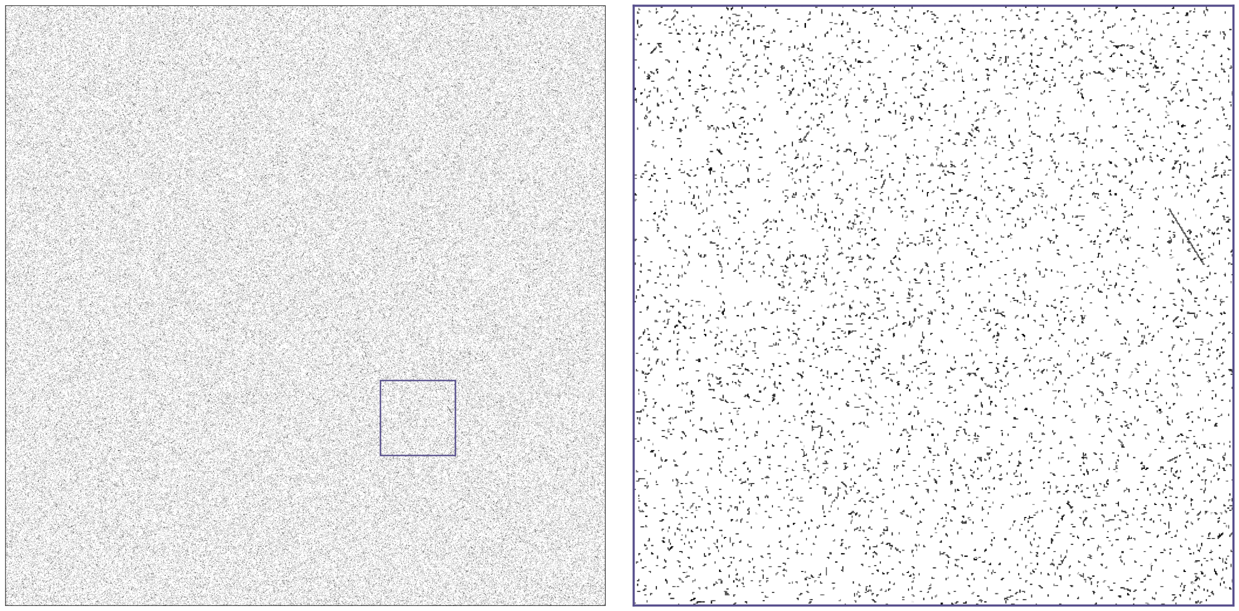

Figure 2: Simulated detector image with 10^6 CR hits (left), zoomed in to a 512×512-pixel region (right), where the purple outline in the right panel indicates the extent of the right panel. Detector counts are shown in grayscale over a logarithmic stretch, spanning from 10^2 (white) to 10^4 electrons (black). This figure is best viewed electronically.

```
# simulate 500 resultant frames
for _ in range(500):
    wfi_image = simulate_crs(wfi_image, cr_flux, wfi_area, t_exp)
```

In Figure 2, we show a simulated image with 100,000 CR hits. It is difficult to visualize the entire 4096×4096-pixel detector image, so we also show a 512×512-pixel zoom in the left panel.

5.3 Time complexity

Our algorithm is implemented in two steps: first, we draw random CR parameters for a Poisson-distributed number of CR events, and then we deposit counts onto the detector. To test the first step, we sample 10^6 cosmic ray parameters, which takes 139 ± 1 ms based on 7 runs with 10 loops each (measured using the `%%timeit` module in IPython).[3] We also test the time complexity for our full algorithm over a variety of simulated exposures times, ranging from 3 to 10^5 s, for a CR flux of 8 event cm^{-2} s^{-1} and a 16.8 cm^2 single WFI detector area. We show the results in Figure 4. The algorithm appears to incur a minimum time of 7 ms, and then it scales linearly with simulated exposure time. However, the maximum exposure time should not exceed ~ 2000 s, so we do not realistically expect a need for simulating such long exposures.

[3]All tests have been run on a 2020 MacBook Pro laptop with an M1 processor and 16 GB of RAM.

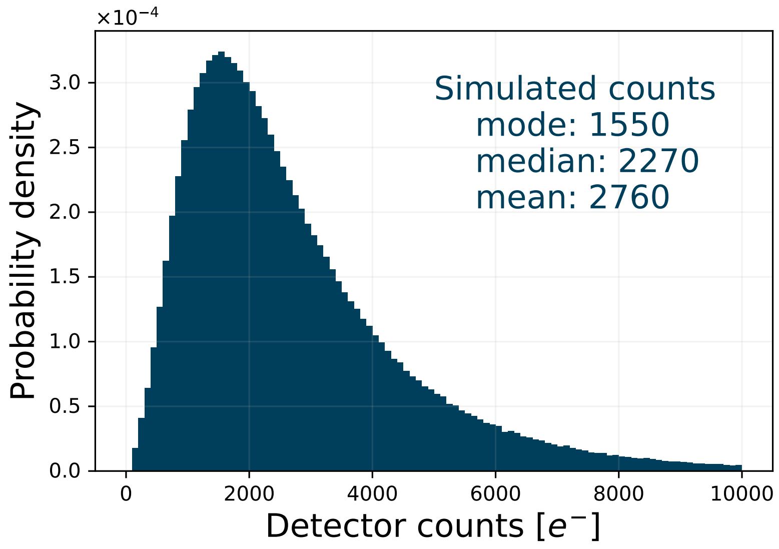

Figure 3: Distributions of CR counts from the simulated detector images. We have also listed the mode, median, and mean for the distribution.

Percentile of distribution	0.1	0.5	1	2.5	5	16	50
Simulated counts [e$^-$]	170	280	370	530	700	1150	2270

Table 1: Percentiles corresponding to simulated electron counts due to CRs, per pixel, per 3.04 s read.

6 Discussion

6.1 Typical jumps in detector counts

We are interested in estimating the typical increase in detector counts for a CR-affected WFI pixel, which can be used to tune the jump detection algorithm. We simulate CRs for 10^3 independent single reads using the default CR flux and a read time of 3.04 s, each. We select affected pixels (i.e., those with non-zero counts) and plot their distribution in Figure 3; here we have selected over one million affected pixels. The mode of the simulated distribution occurs at 1550 e$^-$, whereas the median and mean values are 2270 and 2760 e$^-$, respectively.[4] The simulated distribution qualitatively resembles the Moyal function in terms of its skewed long tail. However, the simulated distribution exhibits higher variance than the input Moyal distribution; this broadening is due to the range of traversal lengths per pixel, and Poisson sampling of CR events. There is a small chance that an affected pixel can be impacted by multiple CRs in a single read, which should slightly increase the mode of the simulated distribution relative to the original Moyal distribution.

Based on these simulations, we estimate that the typical CR will cause the Roman WFI to jump by 1550 e$^-$. In Table 1, we list various percentiles in the simulated distribution of counts. These values may be useful for tuning the Roman jump detection algorithm.

[4]Simulated counts are rounded to the nearest 10 e$^-$.

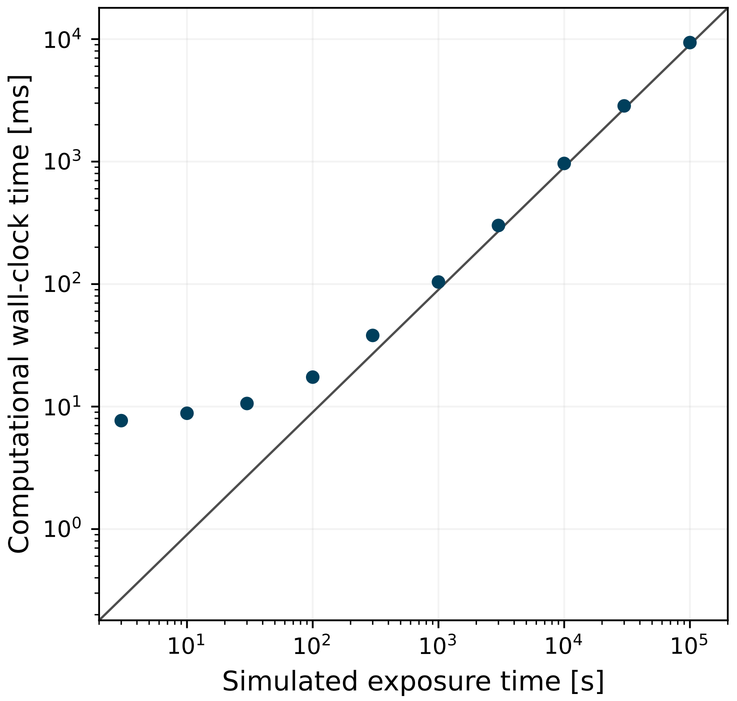

Figure 4: The blue markers show how computation time (in milliseconds) scales with CR simulation exposure times (in seconds). Uncertainties are very small and effectively invisible on this log-log plot. A linear scaling is shown in the gray line.

6.2 Very preliminary comparison to JWST data

Our algorithm is extremely modular and can also be used to simulate CRs for a JWST H2RG-like detector. This flexibility allows us to compare simulated CR properties to measurements from early JWST NIRSpec dark images (Rauscher 2022). First, we replace all of the default values with those for a H2RG detector: 2048 × 2048 pixels over a $13.6\,\mathrm{cm}^2$ area, $18\,\mu$m pixel pitch, $6\,\mu$m pixel depth, and $E_g = 0.225$ eV counts^{-1} conversion factor. Because we are comparing against JWST hour-long darks, we sets the exposure time to 3600 s. We use the same Moyal energy loss distribution as before, with peak and width values of 120 and 50 eV μm, respectively. Using these assumptions, we are able to simulate CRs for JWST H2RG detectors.

If we assume CR fluxes in the range between 5 and 8 events $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$, we find a mode of $4050 - 4350$ e$^-$ in the JWST-like simulated distribution. The mode of our simulated distribution is consistent with the measured peak value of 4250 e$^-$ (Rauscher 2022; note that they estimate a CR flux of ~ 5.5 events $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$). However, their preliminary analysis combines all JWST pixels affected by a CR before reporting the distribution of detector counts, whereas we report the distribution for individual affected pixels. Rauscher (2022) find a 10-pixel area per event using their custom jump detection algorithm, whereas Rigby et al. (2022) find that most events are only one or two pixels, in agreement with our simulated results. We do not simulate inter-pixel capacitance, which impact neighboring pixels and can increase the number of affected pixels by a factor of ~ 9, and may explain some of the discrepancy. We also do not incorporate snowball events, which are expected to significantly increase the tail towards higher counts (Rigby et al. 2022). Despite these different assumptions, our comparison suggests that our fiducial distributions of CR parameters are reasonable.

7 Summary, caveats, and future directions

We have presented an algorithm for simulating CR events onto a WFI-like detector. Along the way, we made several simplifying assumptions on the statistics of CR trajectories, shapes, and deposited energies, based on simulated and empirical data; thus, it is important to mention some limitations and caveats of the CR algorithm. Some limitations are due to the practical trade-off between realism and computational efficiency. Others are simply because we are still awaiting full analyses of empirical (on-flight) data for better understanding of the CR properties and of their impacts on near-infrared detectors. Already, preliminary analyses are suggesting that the JWST CR flux in the NRS1 and NRS2 NIRSpec detectors may be somewhat lower than expected (5.5 rather than 8 $\mathrm{cm^{-2}\,s^{-1}}$; Rauscher 2022). The true conversion factor from CR energy to detector counts may also differ from our assumed value based on the bandgap energy $E_g = 0.5$ eV $\mathrm{counts^{-1}}$; for example, multiple electrons can be produced depending on the quantum yield (e.g., Givans et al. 2022, see also McCullough et al. 2008 and Rauscher et al. 2014).

The true distributions of CR spatial locations and orientation angles will depend on the Roman Space Telescope shielding and pointings, but we have adopted uniform distributions for CR locations and orientations (see II.a and II.b in Section 4.1). We use a power-law distribution to characterize the relative frequency of CR path lengths, calibrated to empirical HST measurements (Miles et al. 2021). All CR trails are treated as zero-width linear paths, even though empirically some CRs produce thick paths (possibly due to inter-pixel capacitance or charge diffusion) or curved trails through the detector, occasionally resulting in “worm”-like shapes. We do not allow for variations in the CR energy loss, and also do not allow for breaks in the signal, despite these effects being observed in real data.[5] We also do not include snowballs[6] in the CR algorithm since they are not yet well-understood. Finally, we assume a single constant distribution of CR energy losses, when in reality there are many species of CR particles, as well as a strong dependence of CR energy spectra on the solar cycle (Section 3) and perhaps the telescope pointing. This assumed energy loss distribution is further simplified by our use of a Moyal function.

We also do not implement effects due to detector physics or instrumental electronics. For example, our algorithm does not include inter-pixel capacitance, the brighter-fatter effect, charge diffusion, or other instrumental processes. It is assumed that these effects will be captured in a later step of the full simulation package. Since we are only simulating for a single detector, we have implicitly assumed that all detectors have the same CR properties.

[5]There is early evidence that high-amplitude, long-trail JWST NIRSpec CR hits are accompanied by compact “sparkles” continuing to radiate in the direction of the CR path (Rauscher 2022).

[6]https://jwst-docs.stsci.edu/data-artifacts-and-features/snowballs-and-shower-artifacts

We also have not included any kind of depth effects for the CR trails, and we assume that the CR passes through the entire 5 μm pixel depth (although this assumption more frequently breaks down for very low-energy or very heavy particles). CR hits deposit energy instantaneously, and we do not divide their trails across multiple read frames.

The presented algorithm is able to produce CR signatures that roughly comport with preliminary JWST data (see Section 6.2). However, our CR implementation is expected to evolve as JWST collects more science and calibration data at L2. For example, we currently sample CR path lengths from a power-law distribution, but this is based on HST data taken in a different environment. The frequency of CRs with long paths is highly sensitive to our choice of minimum path length (currently set to 10 μm), and we expect that this parameter will need to be updated. We also anticipate that snowballs will be characterized in greater detail at some point in the future (Rigby et al. 2022), and that these will be injected into simulated images in order to test the Roman calibration software pipeline. We may also consider sampling from a bivariate distribution of the CR path lengths and energy losses, since these two parameters are likely to be physically correlated.

Acknowledgments — It is our pleasure to thank Andrea Bellini, Sanjib Sharma, and Eddie Schlafly for reading over our tech report and providing valuable feedback. We are also grateful to Gisella De Rosa and Jonathan Hargis for their encouragement and support. We thank Greg Mosby for providing guidance on H4RG-10 detector properties, and we thank Mario Gennaro for help with obtaining JWST NIRCam darks data.

A An implementation in Python

Below, we show the Python code listing for the full cosmic ray algorithm.

```python
import numpy as np
import scipy.interpolate as interpolate


def create_sampler(pdf, x):
    """A function for performing inverse transform sampling.

    Parameters
    ----------
    pdf : callable
        A function or empirical set of tabulated values which can
        be used to call or evaluate `x`.
    x : 1-d array of floats
        A grid of values where the pdf should be evaluated.

    Returns
    -------
    inverse_cdf : 1-d array of floats
        The cumulative distribution function which allows sampling
        from the `pdf` distribution within the bounds described
        by the grid `x`.
    """
    y = pdf(x)
    cdf_y = np.cumsum(y) - y[0]
    cdf_y /= cdf_y.max()
    inverse_cdf = interpolate.interp1d(cdf_y, x)
    return inverse_cdf


def moyal_distribution(x, location=120, scale=50):
    """Return unnormalized Moyal distribution, which approximates a
    Landau distribution and is used to describe the energy loss
    probability distribution of a charged particle through a detector.

    Parameters
    ----------
    x : 1-d array
        An array of dE/dx values (units: eV/micron) that forms the
        grid on which the Moyal distribution will be evaluated.
    location : float
        The peak location of the distribution, units of eV / micron.
    scale : float
        A width parameter for the distribution, units of eV / micron.
    Returns
    -------
    moyal : 1-d array of floats
        Moyal distribution (pdf) evaluated on `x` grid of points.
    """
    xs = (x - location) / scale
    moyal = np.exp(-(xs + np.exp(-xs)) / 2)
    return moyal


def power_law_distribution(x, slope=-4.33):
    """Return unnormalized power-law distribution parameterized by
    a log-log slope, used to describe the cosmic ray path lengths.

    Parameters
    ----------
    x : 1-d array of floats
        An array of cosmic ray path lengths (units: micron).
    slope : float
        The log-log slope of the distribution, default based on
        Miles et al. (2021).

    Returns
    -------
    power_law : 1-d array of floats
        Power-law distribution (pdf) evaluated on `x` grid of points.
    """
    power_law = np.power(x, slope)
    return power_law


def sample_cr_params(
    N_samples,
    N_i=4096,
    N_j=4096,
    min_dEdx=10,
    max_dEdx=10000,
    min_cr_len=10,
    max_cr_len=2000,
    grid_size=10000,
    rng=None,
    seed=48,
):
    """Generates cosmic ray parameters randomly sampled from distribution.
    One might re-implement this by reading in parameters from a reference
    file, or something similar.

    Parameters
    ----------
    N_samples : int
        Number of CRs to generate.
    N_i : int
        Number of pixels along i-axis of detector
    N_j : int
        Number of pixels along j-axis of detector
    min_dEdx : float
        Minimum value of CR energy loss (dE/dx), units of eV / micron.
    max_dEdx : float
        Maximum value of CR energy loss (dE/dx), units of eV / micron.
    min_cr_len : float
        Minimum length of cosmic ray trail, units of micron.
    max_cr_len : float
        Maximum length of cosmic ray trail, units of micron.
    grid_size : int
        Number of points on the cosmic ray length and energy loss grids.
        Increasing this parameter increases the level of sampling for
        the distributions.
    rng : np.random.Generator
        Random number generator to use
    seed : int
        seed to use for random number generator

    Returns
    -------
    cr_x : float, between 0 and N_x-1
        x pixel coordinate of cosmic ray, units of pixels.
    cr_y : float between 0 and N_y-1
        y pixel coordinate of cosmic ray, units of pixels.
    cr_phi : float between 0 and 2pi
        Direction of cosmic ray, units of radians.
    cr_length : float
        Cosmic ray length, units of micron.
    cr_dEdx : float
        Cosmic ray energy loss, units of eV / micron.
    """

    if rng is None:
        rng = np.random.default_rng(seed)

    # sample CR positions [pix]
    cr_i, cr_j = (
        rng.random(size=(N_samples, 2)) * (N_i, N_j) - 0.5
    ).transpose()

    # sample CR direction [radian]
    cr_phi = rng.random(N_samples) * 2 * np.pi

    # sample path lengths [micron]
    len_grid = np.linspace(min_cr_len, max_cr_len, grid_size)
    inv_cdf_len = create_sampler(power_law_distribution, len_grid)
    cr_length = inv_cdf_len(rng.random(N_samples))

    # sample energy losses [eV/micron]
    dEdx_grid = np.linspace(min_dEdx, max_dEdx, grid_size)
    inv_cdf_dEdx = create_sampler(moyal_distribution, dEdx_grid)
    cr_dEdx = inv_cdf_dEdx(rng.random(N_samples))

    return cr_i, cr_j, cr_phi, cr_length, cr_dEdx


def traverse(trail_start, trail_end, N_i=4096, N_j=4096, eps=1e-10):
    """Given a starting and ending pixel, returns a list of pixel
    coordinates (ii, jj) and their traversed path lengths. Note that
    the centers of pixels are treated as integers, while the borders
    are treated as half-integers.

    Parameters
    ----------
    trail_start : (float, float)
        The starting coordinates in (i, j) of the cosmic ray trail,
        in units of pix.
    trail_end : (float, float)
        The ending coordinates in (i, j) of the cosmic ray trail, in
        units of pix.
    N_i : int
        Number of pixels along i-axis of detector
    N_j : int
        Number of pixels along j-axis of detector
    eps : float
        Tiny value used for stable numerical rounding.

    Returns
    -------
    ii : np.ndarray[int]
        i-axis positions of traversed trail, in units of pix.
    jj : np.ndarray[int]
        j-axis positions of traversed trail, in units of pix.
    lengths : np.ndarray[float]
        Chord lengths for each traversed pixel, in units of pix.
    """
    # increase in i-direction
    if trail_start[0] < trail_end[0]:
        i0, j0 = trail_start
        i1, j1 = trail_end
    else:
        i1, j1 = trail_start
        i0, j0 = trail_end

    di = i1 - i0
    dj = j1 - j0
    slope = dj / di
    sign = np.sign(slope)

    # horizontal border crossings at j = integer + 1/2
    if dj != 0:
        j_horiz = np.arange(np.round(j0), np.round(j1), sign) + 0.5 * sign
        i_horiz = i0 + (di / dj) * (j_horiz - j0)
        cross_horiz = np.transpose([i_horiz, j_horiz])
    else:
        cross_horiz = np.array([[]])

    # vertical border crossings at i = integer + 1/2
    if di != 0:
        i_vert = np.arange(np.round(i0), np.round(i1), 1) + 0.5
        j_vert = j0 + (dj / di) * (i_vert - i0)
        cross_vert = np.transpose([i_vert, j_vert])
    else:
        cross_vert = np.array([[]])

    # compute center of traversed pixel before each crossing
    # note `eps` here covers rounnding issues when the corner is
   intersected
    ii_horiz, jj_horiz = np.round(
        cross_horiz - np.array([eps, np.sign(dj)*0.5])
    ).astype(int).T
    ii_vert, jj_vert = np.round(
        cross_vert - np.array([0.5, np.sign(dj) * eps])
    ).astype(int).T

    # combine crossings and pixel centers
    crossings = np.vstack((cross_horiz, cross_vert))
    ii = np.concatenate((ii_horiz, ii_vert))
    jj = np.concatenate((jj_horiz, jj_vert))

    # sort by i axis
    sorted_by_i = np.argsort(crossings[:, 0])
    crossings = crossings[sorted_by_i]
    ii = ii[sorted_by_i]
    jj = jj[sorted_by_i]

    # add final pixel center
    ii = np.concatenate((ii, [np.round(i1).astype(int)]))
    jj = np.concatenate((jj, [np.round(j1).astype(int)]))
    # if no crossings, then it's just the total Euclidean distance
    if len(crossings) == 0:
        lengths = np.linalg.norm([di, dj], keepdims=1)
    # otherwise, compute starting, crossing, and ending distances
    else:
        first_length = np.linalg.norm(crossings[0] - np.array([i0, j0]),
   keepdims=1)
        middle_lengths = np.linalg.norm(np.diff(crossings, axis=0), axis
   =1)
        last_length = np.linalg.norm(np.array([i1, j1]) - crossings[-1],
   keepdims=1)
        lengths = np.concatenate([first_length, middle_lengths,
   last_length])

        # remove 0-length trails
        positive_lengths = lengths > 0
        lengths = lengths[positive_lengths]
        ii = ii[positive_lengths]
        jj = jj[positive_lengths]

    # remove any pixels that go off the detector
    inside_detector = (ii > -0.5) & (ii < (N_i - 0.5)) & (jj > -0.5) & (jj
    < (N_j - 0.5))
    ii = ii[inside_detector]
    jj = jj[inside_detector]
    lengths = lengths[inside_detector]

    return ii, jj, lengths


def simulate_crs(
    image,
    time,
    flux=8,
    area=16.8,
    conversion_factor=0.5,
    pixel_size=10,
    pixel_depth=5,
    rng=None,
    seed=47
):
    """Adds CRs to an existing image.

    Parameters
    ----------
    image : 2-d array of floats
        The detector image with values in units of electrons.
    time : float
        The exposure time, units of s.
    flux : float
        Cosmic ray flux, units of cm^-2 s^-1. Default value of 8
        is equal to the value assumed by the JWST ETC.
    area : float
        The area of the WFI detector, units of cm^2.
    conversion_factor : float
        The convert from eV to electrons, assumed to be the bandgap energy
   ,
        in units of eV / electrons.
    pixel_size : float
        The size of an individual pixel in the detector, units of micron.
    pixel_depth : float
        The depth of an individual pixel in the detector, units of micron.
    rng : np.random.Generator
        Random number generator to use
    seed : int
        seed to use for random number generator

    Returns
    -------
    image : 2-d array of floats
        The detector image, in units of electrons, updated to include
        all of the generated cosmic ray hits.
    """

    if rng is None:
        rng = np.random.default_rng(seed)

    N_i, N_j = image.shape
    N_samples = rng.poisson(flux * area * time)
    cr_i0, cr_j0, cr_angle, cr_length, cr_dEdx = sample_cr_params(
        N_samples, N_i=N_i, N_j=N_j, rng=rng)

    cr_length = cr_length / pixel_size
    cr_i1 = (cr_i0 + cr_length * np.cos(cr_angle)).clip(-0.5, N_i + 0.5)
    cr_j1 = (cr_j0 + cr_length * np.sin(cr_angle)).clip(-0.5, N_j + 0.5)

    # go from eV/micron -> electrons/pixel
    cr_counts_per_pix = cr_dEdx * pixel_size / conversion_factor

    for i0, j0, i1, j1, counts_per_pix in zip(cr_i0, cr_j0, cr_i1, cr_j1,
   cr_counts_per_pix):
        ii, jj, length_2d = traverse([i0, j0], [i1, j1], N_i=N_i, N_j=N_j)
        length_3d = ((pixel_depth / pixel_size) ** 2 + length_2d ** 2) **
   0.5
        image[ii, jj] += rng.poisson(counts_per_pix * length_3d)

    return image
```

References

Agostinelli, S., Allison, J., Amako, K., et al. 2003, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 506, 250, doi: https://doi.org/10.1016/S0168-9002(03)01368-8

Allison, J., Amako, K., Apostolakis, J., et al. 2016, Nuclear Instruments and Methods in

Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 835, 186, doi: https://doi.org/10.1016/j.nima.2016.06.125

Amanatides, J., & Woo, A. 1987, Proceedings of EuroGraphics, 87

Amsler, C., Doser, M., Antonelli, M., et al. 2008, Physics Letters B, 667, 1, doi: 10.1016/j.physletb.2008.07.018

Anderson, R. E., & Gordon, K. D. 2011, PASP, 123, 1237, doi: 10.1086/662593

Barth, J. L., Isaacs, J. C., & Poivey, C. 2000, in The Radiation Environment for the Next Generation Space Telescope

Bellini, A., Desjardins, T. D., Casertano, S., & Ryan, R. E. J. 2022, Roman-STScI-000433, SC-01

Bushouse, H., Eisenhamer, J., Dencheva, N., et al. 2023, JWST Calibration Pipeline, 1.10.2, Zenodo, doi: 10.5281/zenodo.7829329

Casertano, S. 2022, Roman-STScI-000394, SC-01

Fisher-Levine, M., & Nomerotski, A. 2015, Journal of Instrumentation, 10, C08006, doi: 10.1088/1748-0221/10/08/C08006

Giardino, G., Birkmann, S., Robberto, M., et al. 2019, PASP, 131, 094503, doi: 10.1088/1538-3873/ab2fd6

Givans, J. J., Choi, A., Porredon, A., et al. 2022, PASP, 134, 014001, doi: 10.1088/1538-3873/ac46ba

Hansen, G. L., Schmit, J. L., & Casselman, T. N. 1982, Journal of Applied Physics, 53, 7099, doi: 10.1063/1.330018

Hilbert, B., Sahlmann, J., Volk, K., et al. 2019, spacetelescope/mirage: First github release, v1.1.1, Zenodo, doi: 10.5281/zenodo.3519262

McCullough, P. R., Regan, M., Bergeron, L., & Lindsay, K. 2008, PASP, 120, 759, doi: 10.1086/590161

Miles, N. D., Deustua, S. E., Tancredi, G., et al. 2021, ApJ, 918, 86, doi: 10.3847/1538-4357/abfa9b

Mosby, G., Rauscher, B. J., Bennett, C., et al. 2020, Journal of Astronomical Telescopes, Instruments, and Systems, 6, 046001, doi: 10.1117/1.JATIS.6.4.046001

Moyal, J. 1955, The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 46, 263, doi: 10.1080/14786440308521076

Potgieter, M. S. 2013, Living Reviews in Solar Physics, 10, 3, doi: 10.12942/lrsp-2013-3

Rauscher, B. J. 2022, JWST NIRSpec Cosmic Rays Update for Roman WFI Calibration Working Group

Rauscher, B. J., Boehm, N., Cagiano, S., et al. 2014, PASP, 126, 739, doi: 10.1086/677681

Rigby, J., Perrin, M., McElwain, M., et al. 2022, arXiv e-prints, arXiv:2207.05632, doi: 10.48550/arXiv.2207.05632

Robberto, M. 2009, JWST-STScI-001928, SM-12

Romancal Developers. 2022, romancal: The Roman Space Telescope Calibration Pipeline, 0.9.0. https://github.com/spacetelescope/romancal

Tylka, A., Adams, J., Boberg, P., et al. 1997, IEEE Transactions on Nuclear Science, 44, 2150, doi: 10.1109/23.659030

van Dokkum, P. G. 2001, PASP, 113, 1420, doi: 10.1086/323894

Wu, X. 1991, SIGGRAPH Comput. Graph., 25, 143–152, doi: 10.1145/127719.122734

Zhang, K., & Bloom, J. S. 2020, ApJ, 889, 24, doi: 10.3847/1538-4357/ab3fa6

Ziegler, J. F., Ziegler, M., & Biersack, J. 2010, Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms, 268, 1818, doi: https://doi.org/10.1016/j.nimb.2010.02.091

Zingl, A. 2012. http://members.chello.at/easyfilter/Bresenham.pdf