Image Simulations for Testing the Fidelity of SKYSURF Background Measurement Algorithms

Delondrae Carter, Haley Abate, Timothy Carleton

April 7, 2021

Contents

1 Introduction

The goal of Hubble Space Telescope Cycle 27–29 Archival Legacy project "SKYSURF" is to measure the panchromatic sky surface brightness and source catalogs from all archival HST ACS and WFC3 images since the launch of these instruments by the Space Shuttle—more than 57,000 images in total since 2002. All SKYSURF images together will measure the panchromatic Zodiacal brightness, the Diffuse Galactic Light, and the Extragalactic Background Light. SKYSURF will significantly constrain the various amounts of diffuse light in the universe with major ramifications for cosmic star formation and planet formation.

Several sky background measurement algorithms are capable of measuring the background levels of images in the SKYSURF database. To test the fidelity of these sky background measurement algorithms, images with known sky background and noise levels were necessary to determine quantitatively how far a sky measurement algorithm strays from the true value. For this purpose, I developed an algorithm that could create simulated images for filter F125W of the WFC3/IR instrument on the *Hubble Space Telescope* (HST). Filter F125W was selected because the Extragalactic Background Light is brightest in this wavelength band; moreover, the COBE Zodiacal light measurement is also at 1.25 microns. The simulated images created contain stars, galaxies, cosmic rays, and light gradients. We discuss here how these simulated images were made and the different kinds of simulated images that were produced.

2 Image Simulation Details

2.1 Simulated Image Properties

All simulated images were produced to match the flat-fielded images produced by the WFC3/IR camera on Hubble: 1014×1014 pixels, with a 0.13"/pixel pixel scale. When appropriate (e.g. for galaxy/star counts and generating a PSF), parameters associated with the F125W filter were used. We used GALSIM version 2.2.4 [2] to generate simulated images.

2.2 Star Simulation

The number of stars in each simulated image as a function of magnitude was taken from Windhorst et al. 2011 [5]:

$$N_{stars}/(1.0\ mag\ bin) = 10^{[0.04(\mathrm{AB}-18)]},$$

where $N_{stars}/(1.0\ mag\ bin)$ is the number of stars in a 1.0 magnitude bin in the WFC3/IR field of view, and AB is the AB magnitude of the star. The 0.04 in the equation represents the slope of the star counts as a function of magnitude. Stars are restricted to $18 \leq \mathrm{AB} \leq 26$ to avoid unusually bright stars and stars below the F125W detection limit. This resulted in a total of 13 stars generated in each simulated image. Every star was generated as a Gaussian with a full width at half maximum (FWHM) of 0.136". The position of each star in the simulated images was randomly selected with the condition that a star's center be within the 1014×1014 grid.

2.3 Galaxy Simulation

The number of galaxies in each simulated image as a function of magnitude was taken from Windhorst et al. 2011 [5]:

$$N_{galaxies}/(0.5\ mag\ bin) = 10^{[0.26(AB-18)]},$$

where $N_{galaxies}$ is the number of galaxies in a 0.5 magnitude bin in the WFC3 IR field of view. The 0.26 in the equation represents the slope of the galaxy counts as a function of magnitude. Galaxies are restricted to $18 \leq$ AB ≤ 26.5 to avoid unusually bright galaxies and galaxies below the F125W detection limit. This resulted in a total of 624 galaxies generated in each simulated image. Every galaxy was generated using a single-component inclined sersic profile. There are four different properties of these simulated galaxies that were sampled in two different ways for different image sets: AB magnitude, half light radius, sersic index (n), and axis ratio. The two sampling methods used to determine these galactic parameters are discussed in greater detail in section 3. Similarly to the simulated stars, the position of each of the galaxies in the simulated images was randomly selected with the condition that the galaxy's center be within the 1014×1014 grid.

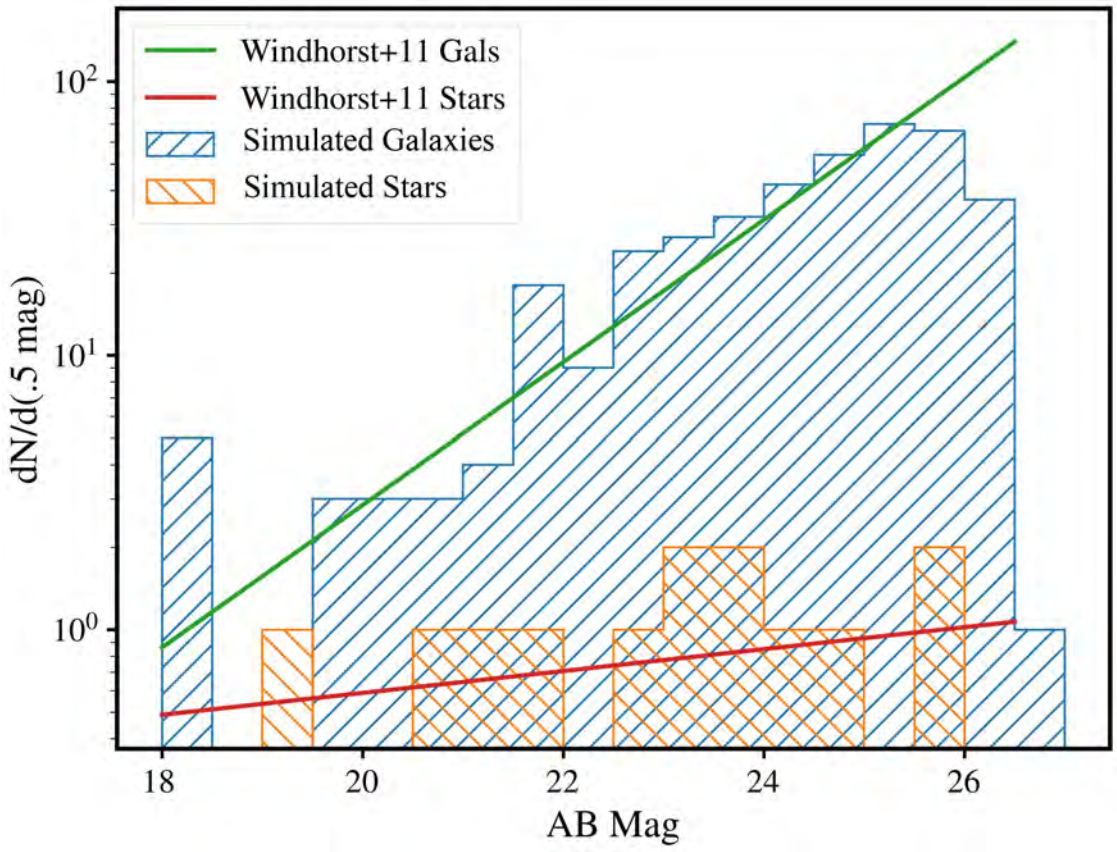

Figure 1: Histogram of star and galaxy magnitudes for a simulated image as measured with Source Extractor [6]. The distribution is consistent with the star and galaxy count equations from Windhorst et al. 2011 [5].

2.4 Noise Generation

After all stars and galaxies were added to a simulated image, noise was generated for the simulated image with an RMS from the combination of Poisson noise and Gaussian read noise:

$$RMS = \frac{\sqrt{S_{sky} \times t + RN^2}}{t},$$

where S_{sky} is the sky background value, t is the exposure time, and RN is read noise. A read noise of 12 electrons was used for all simulated images, and various different sky background and exposure time values were used (detailed in section 4). Poisson noise was added to the images first, followed by the Gaussian read noise.

2.5 Cosmic Ray Simulation

Cosmic rays in the simulated images were generated by selecting cosmic rays randomly from a WFC3 IR cosmic ray template and inserting them directly into the simulated images. The number of cosmic rays in a simulated image is given by:

$$N_{CRs} = R_{CR} \times t$$

where N_{CRs} is the number of cosmic rays in the simulated image, R_{CR} is the cosmic ray rate of the cosmic ray template, and t is the exposure time of the simulated image. The cosmic ray template was generated by identifying spikes in the individual reads of image ibp309bxq_ima.fits. This resulted in a rate of 14.6 cosmic rays per second over the course of the 1302 second exposure.

2.6 Gradient Simulation

To generate more realistic simulated images, some were modeled with linear gradients. These gradients were generated according to:

$$N_{row} = O_{row} + \frac{P}{100}\frac{R_{num}}{R_{tot}}O_{row},$$

where N_{row} are the gradient adjusted pixel values for a particular row of pixels in the image, O_{row} are the non-gradient adjusted pixel values, P is the percent change between the bottom and top row of the image, R_{num} is the row number being adjusted, and R_{tot} is the total number of rows in the image.

3 Galaxy Simulation Inputs

For both methods described below, the ratio of the scale height to the scale radius was set to 0.1 if the galaxy's sersic index was < 2, and set to 0.9 if the galaxy's sersic index was ≥ 2. PSF convolution was performed on each galaxy using a gaussian of the same FWHM as the star FWHM (0.136").

3.1 Galactic Parameter Sampling Method 1: Custom Distributions

For method 1, we generate galaxy parameters from continuous distributions motivated by the Windhorst et al. 2011 catalog of WFC3 ERS data. Images simulated with this method have the half light radius of each galaxy sampled from a distribution of the form:

$$P(r_e) = r_e^{k-1}\frac{e^{-r_e/\theta}}{\theta^k \Gamma(k)}$$

where $P(r_e)$ is the probability of a galaxy having half light radius r_e, $k = 2$, and $\theta = 0.2$. Due to GALSIM memory limitations, only half light radii values of ≤ 2.72" were used to simulate galaxies. The percentage of half light radii values > 2.72 produced by the above distribution is $\ll 1\%$.

Additionally, sersic indices for the galaxies of method 1 follow:

$$P(n) = e^{0.38n}$$

where $P(n)$ is the probability of a galaxy having sersic index n. Because the allowed range of sersic indices for GALSIM is $0.3 \leq n \leq 6.2$, this is the range of sersic index values present for the galaxies in the simulated images.

The magnitudes for each of the 624 galaxies produced from method 1 were randomly selected from a distribution of the form:

$$P(\text{AB}) = 26.5 - \frac{1}{\beta}\exp(-\frac{\text{AB}}{\beta}),$$

where $P(\text{AB})$ is the probability of a galaxy having magnitude AB and $\beta = \frac{1}{0.26\times\ln(10)}$. Since we wanted to generate galaxies with magnitudes between 18 and 26.5 AB magnitudes, only magnitudes ≥ 18 were selected from the above distribution.

Lastly, the inclination of each galaxy produced from method 1 was randomly selected from the range 0 to $\frac{\pi}{2}$ radians.

3.2 Galactic Parameter Sampling Method 2: COSMOS F125W Catalog

Images simulated with this method used parameters sampled from the COSMOS F125W Catalog [1] [3], with the same half light radius, magnitude, and sersic index limitations as method 1.

Each simulated galaxy sampled a random galaxy from this catalog, using its respective half light radius, sersic index, flux, and axis ratio. Inclinations were estimated based on axis ratio using the following:

$$cos^2 i = \frac{(b/a)^2 - \alpha^2}{1 - \alpha^2},$$

where b/a is axis ratio and $\alpha = 0.22$ [4] is the intrinsic ratio of galaxy height to size.

Again, due to GALSIM memory limitations, only half light radii values of ≤ 2.72" were used to simulate galaxies. The percentage of half light radii values > 2.72 produced by sampling method 2 is $< 1\%$.

3.3 Simulated Versus Real Image Comparison Plots

I used Source Extractor to measure the FWHM, axis ratio (b/a), and inclination values of the galaxies in a single simulated image containing no cosmic rays with an exposure time of 500 seconds, then histogrammed the values. I then retrieved a random real WFC3/IR F125W image from the SKYSURF database that had an exposure time of 503 seconds, visually inspected it to ensure there were no abnormalities in the exposure, and then performed the same Source Extractor measurements and histogramming. The resulting plots (displayed below) demonstrate that the galactic parameters of the simulated F125W images are representative of the galaxies in a typical real F125W image. There were a significant number of flagged objects and objects with subpixel FWHM values detected in the real image; these were assumed to be cosmic rays or bad pixels and were thus omitted from the analysis.

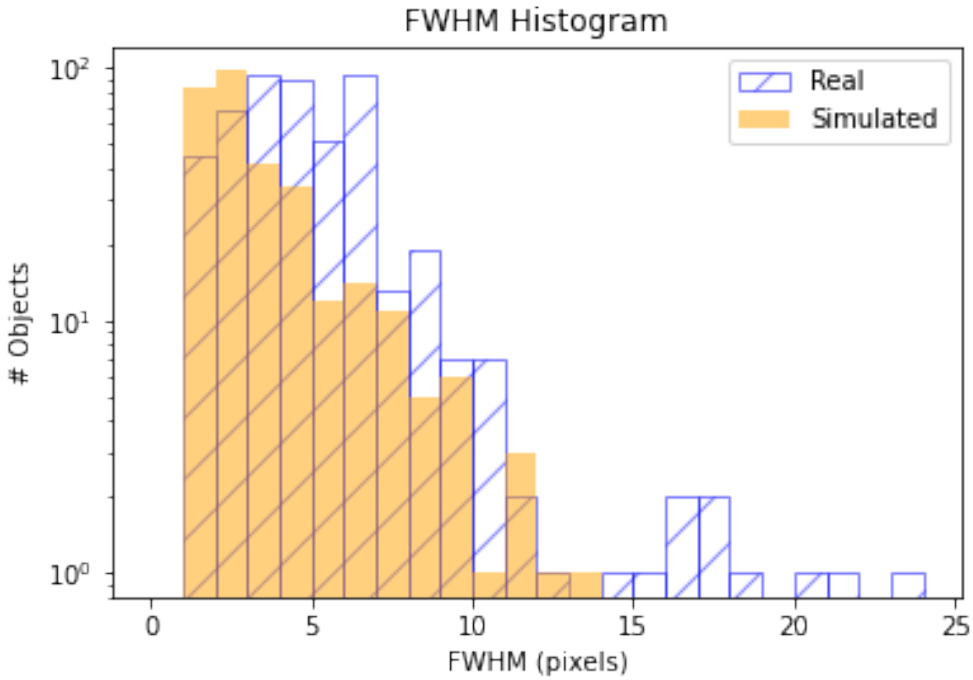

Figure 2: Histograms of galaxy FWHM values for a simulated image and real image.

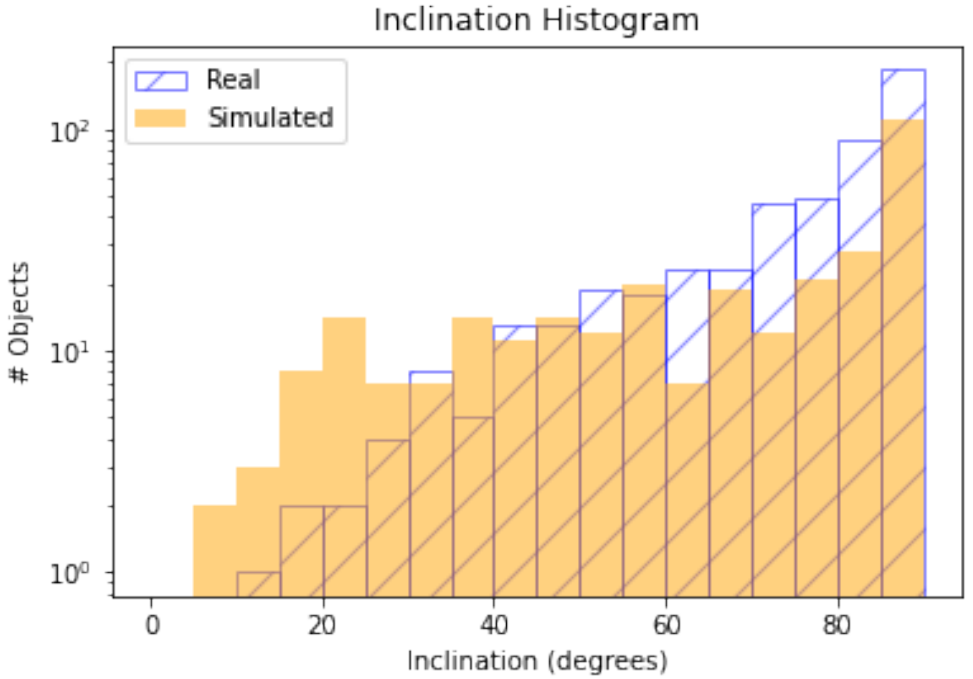

Figure 3: Histograms of galaxy inclination values for a simulated image and real image.

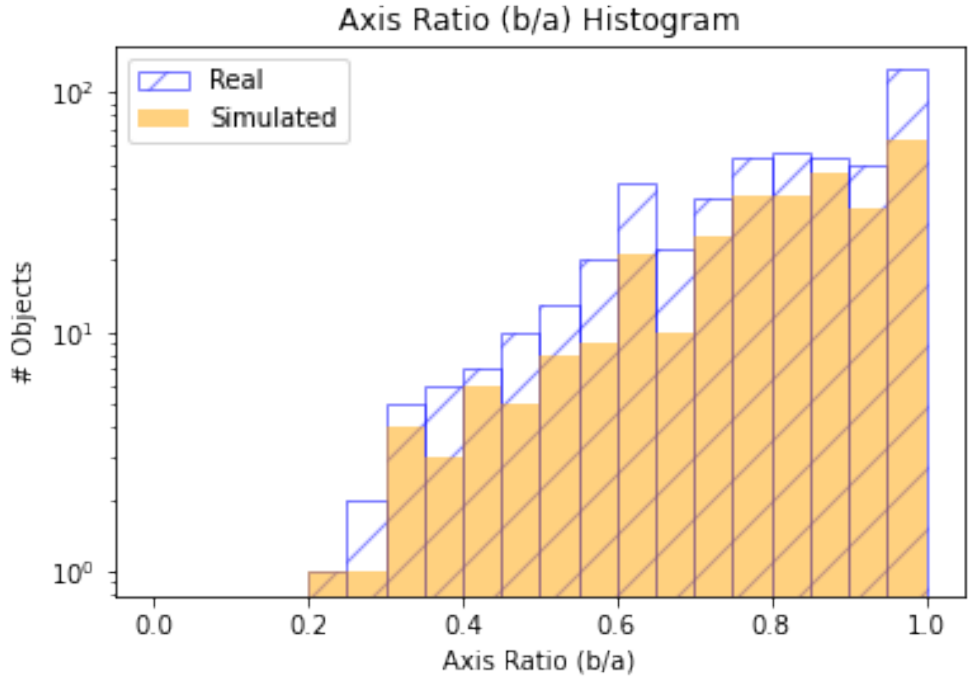

Figure 4: Histograms of galaxy axis ratio (b/a) values for a simulated image and real image.

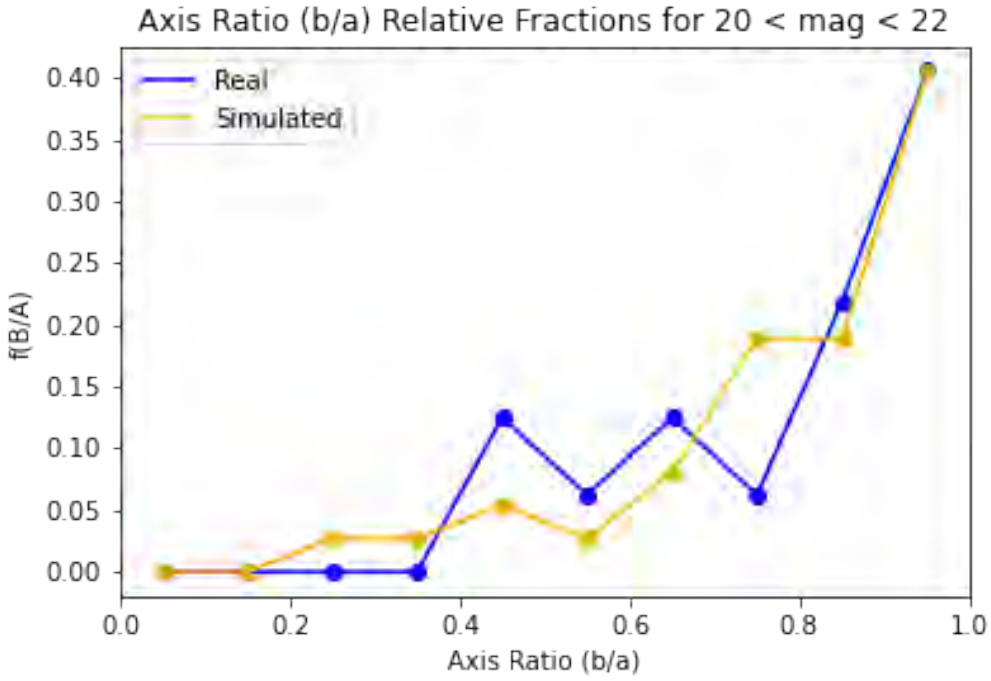

Figure 5: Relative fraction of axis ratios (b/a) for a simulated image and real image. The results differ from the results of figure 3 in Odewahn et al. 1997 [8], but the real and simulated data are consistent with each other.

4 Simulated Image Parameters

4.1 Standard

We simulated a set of images for filter F125W of Hubble's WFC3/IR instrument. These contain realistic stars, galaxies, and cosmic rays. 144 of these images were created using galactic parameter sampling method 1 from a grid of parameters containing all permutations of the following parameters:

$$t = [50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1100, 1302],$$
$$S_{sky} = [\pi/5, \pi/4, \pi/3.5, \pi/3, \pi/2.5, \pi/2.25, \pi/2, \pi/1.75, \pi/1.5, \pi/1.25, \pi/1.125, \pi],$$

where t is exposure time (in seconds) and S_{sky} is the sky background level (in electrons per second). The sky contains both astronomical and instrumental background signal, and the levels are motivated by the WFC3/IR handbook.

200 of these images were created using galactic parameter sampling method 2, each containing parameters randomly selected from the following values:

$$t = [50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1100, 1302],$$
$$S_{sky} = [e/12, e/11, e/10, e/9, e/8, e/7, e/6, e/5, e/4, e/3, e/2, e].$$

4.2 With Gradient

In addition, we simulated images with linear gradients. A total of 240 images with a gradient were created using galactic parameter sampling method 1 from a grid of parameters containing all permutations of the following parameters:

$$t = [50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1100, 1302],$$
$$S_{sky} = [\pi/4],$$
$$P = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20],$$

where P is the percent change between the bottom and top rows of the image ($P = (top - bottom)/bottom$). The S_{sky} values here are the lowest levels in the simulated images (the levels at the bottom rows of the simulated images).

A total of 200 images with a gradient were created using galactic parameter sampling method 2, each containing parameters randomly selected from the following values:

$$t = [50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1100, 1302],$$
$$S_{sky} = [e/12, e/11, e/10, e/9, e/8, e/7, e/6, e/5, e/4, e/3, e/2, e],$$
$$P = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20].$$

4.3 Star Cluster Images

Additionally, we test the fidelity of our sky background measurement algorithms when considering a crowded field, such as a star cluster. These images contain realistic stars and cosmic rays. Stars are placed in a cluster with a distribution of the form:

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

where $\mu = \frac{x-r}{2}$, x is the width of the image, r is the radius of the star, and $\sigma = [5, 6, 7, 8]$, σ being selected at random for each simulated image.

The number of stars in each image as a function of magnitude is described by:

$$N_{stars}/(1.0\ mag\ bin) = 1.0 \times 10^{[2.1+0.394(\mathrm{AB}-19.71)]},$$

where $N_{stars}/(1.0\ mag\ bin)$ is the number of stars in a 1.0 magnitude bin in the WFC3 IR field of view. Stars in the star cluster images are restricted to $18.71 \leq \mathrm{AB} \leq 25.71$. A total of 48774 stars were generated in each star cluster image.

Two sets of 100 images each, one with and one without a gradient, were created using parameters randomly selected from the following values:

$$t = [50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1100, 1302],$$
$$S_{sky} = [e/12, e/11, e/10, e/9, e/8, e/7, e/6, e/5, e/4, e/3, e/2, e],$$
$$P = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20].$$

4.4 With and Without Cosmic Rays

There are two versions of each of the types of simulated image sets discussed above: one containing cosmic rays, and a corresponding set without cosmic rays.

5 Background Measurement Algorithm Fidelity Testing Results

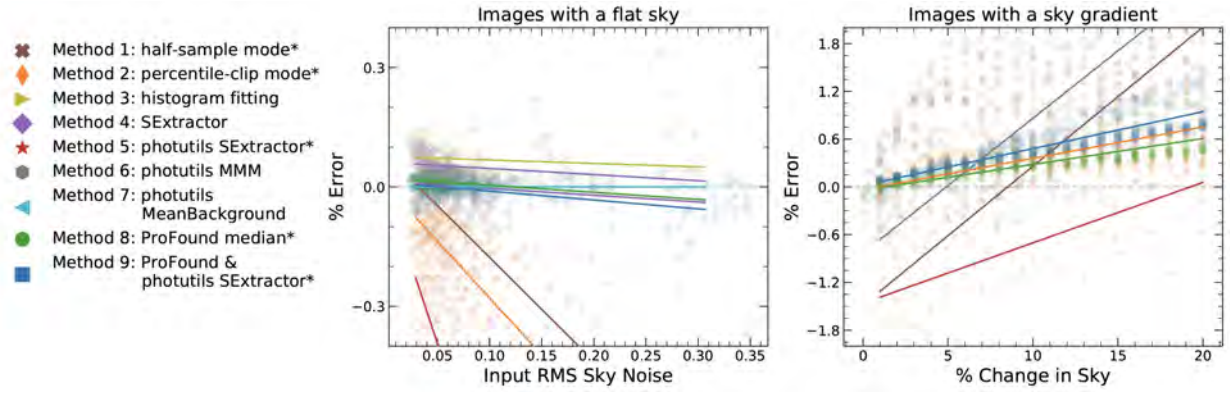

Figure 6: Fidelity testing results for images with flat sky (left) and gradients (right) [10].

Nine different sky measurement algorithms were tested on the simulated images. The left plot in figure 2 shows the percent error of the sky measurements performed by each of these algorithms on images without a gradient as a function of RMS sky noise, and the right plot shows

the percent error of the sky measurements performed on images with a gradient as a function of gradient strength. For images with a flat sky, six of the nine algorithms (histogram fitting, SExtractor, `photutils MMM`, `photutils MeanBackground`, ProFound median, and Profound & `photutils SExtractor`) consistently measured the sky to within 0.3% of the true value, with the `photutils MeanBackground` method performing best. For images with a gradient, three of the nine algorithms (percentile-clip mode, ProFound median, and ProFound & `photutils SExtractor`) consistently measured the sky to within 1.2% of the true value, with the ProFound median method performing best.

6 Conclusions and Future Work

The sky measurement algorithms developed for project SKYSURF are incredibly precise, with many algorithms always measuring within 0.3% of the true value for images with flat sky and within 1.2% for images with a gradient. Of the sky measurement algorithms, the `photutils MeanBackground` method is best for measuring images with flat sky, and the Profound median method is best for measuring images with gradients. The `photutils MeanBackground` method works so well for flat images because it calculates the sky as the sigma-clipped mean. Because there are a significantly greater number of sky background pixels compared to pixels with stars, galaxies, and cosmic rays (most of which will get clipped), the sigma-clipped mean produces an extremely accurate answer for images with little to no light variation. For gradient images, this method fails because the mean pixel value increases from the bottom row to the top row. Algorithms that can break images up into sub-regions, such as the ProFound median method, do a far better job at detecting and accounting for such light variations when measuring the sky. Because we will not know beforehand which of the 60,000 real images in the SKYSURF database will have a light gradient (or how strong the gradient is), it is best to proceed with sky measurements using the ProFound median method. The ProFound median method maintains extraordinary accuracy for flat images while being able to easily map gradients, so it is the most flexible option. Once the ProFound median method is used to measure the sky levels of the images in the database, the results will be used as input to the next step of project SKYSURF: image drizzling.

7 Appendix

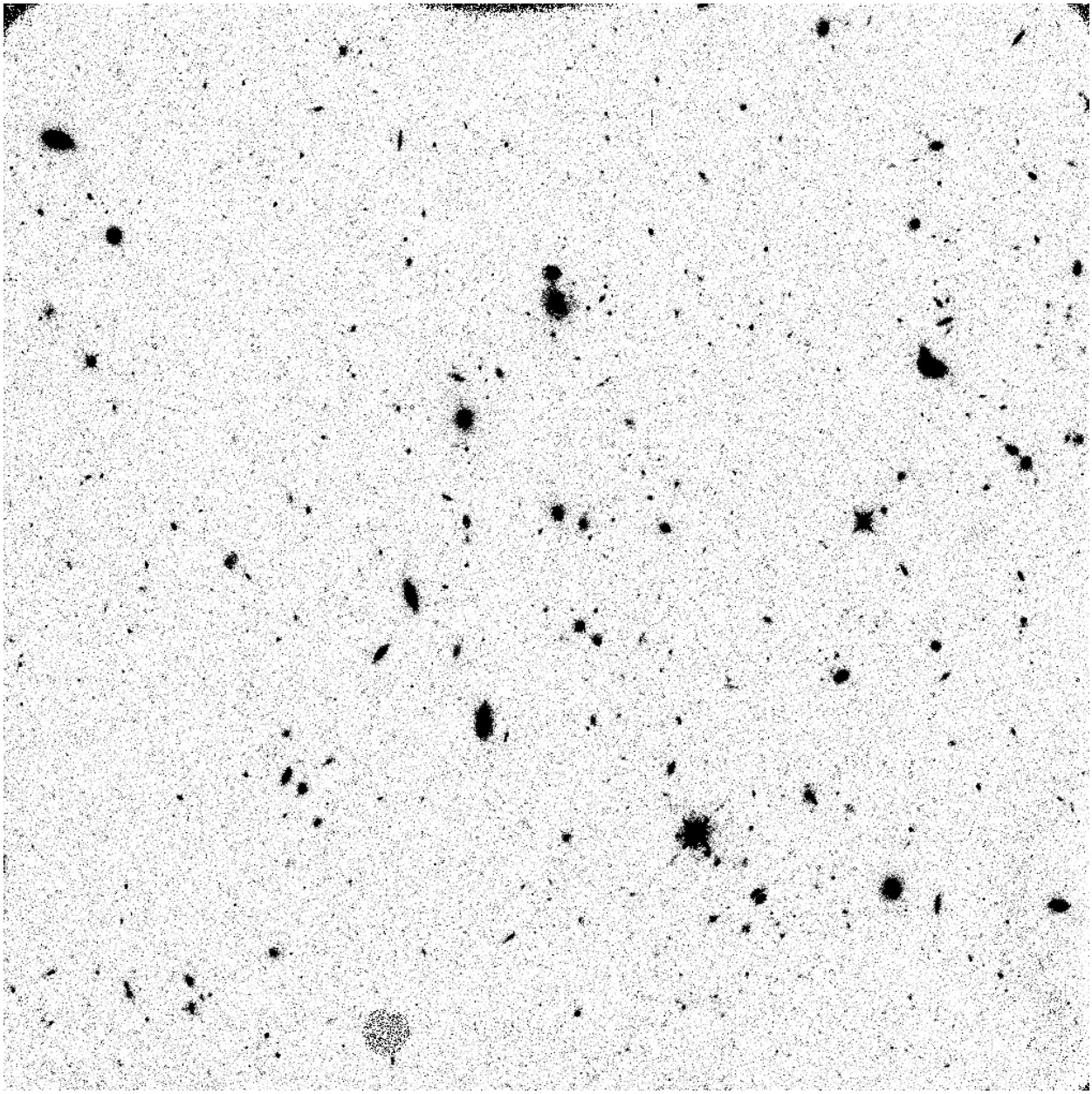

Figure 7: The real WFC3/IR F125W image used for the analysis in section 3.3

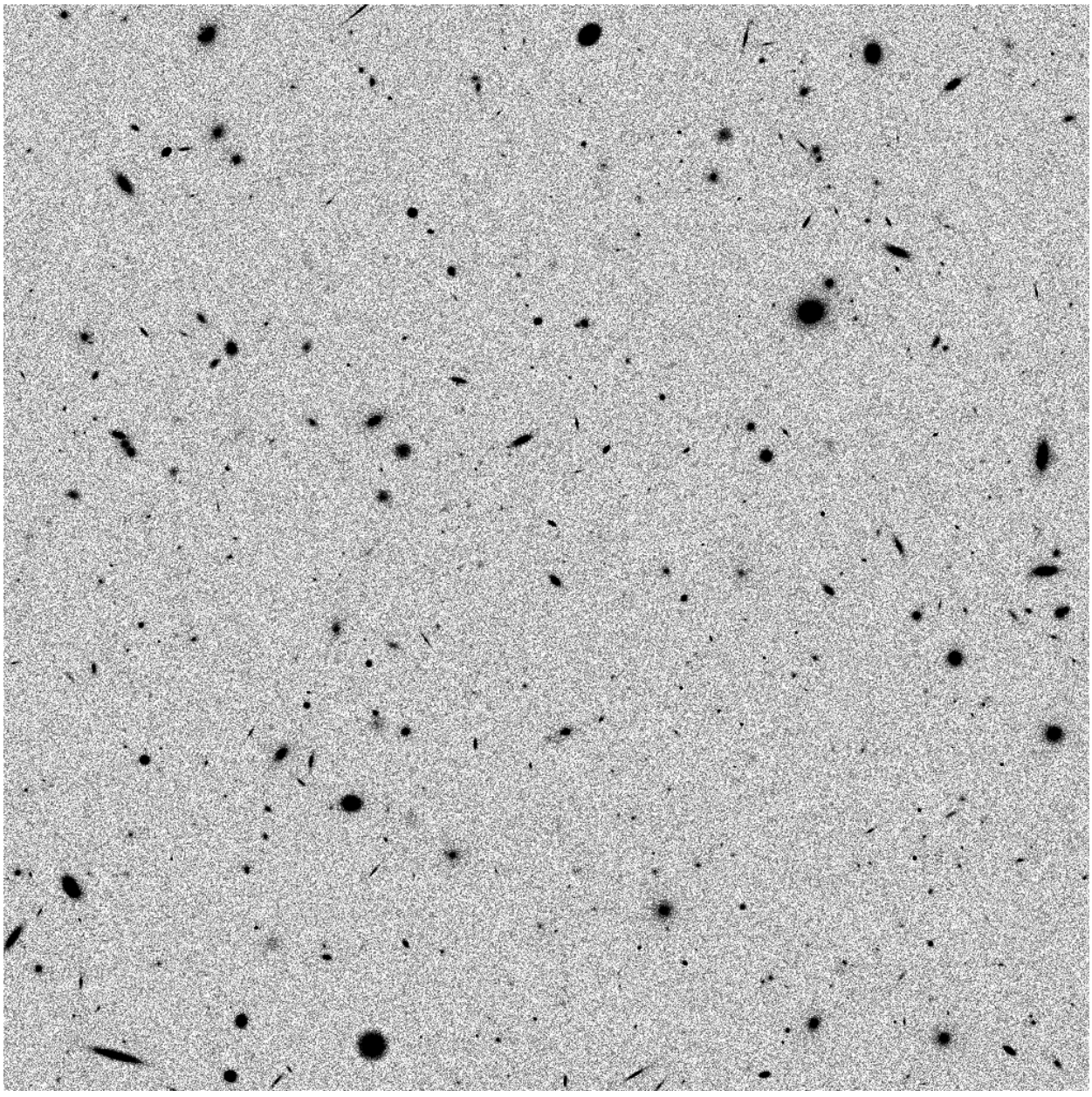

Figure 8: Standard Simulated Image (Method 1 Galaxies)

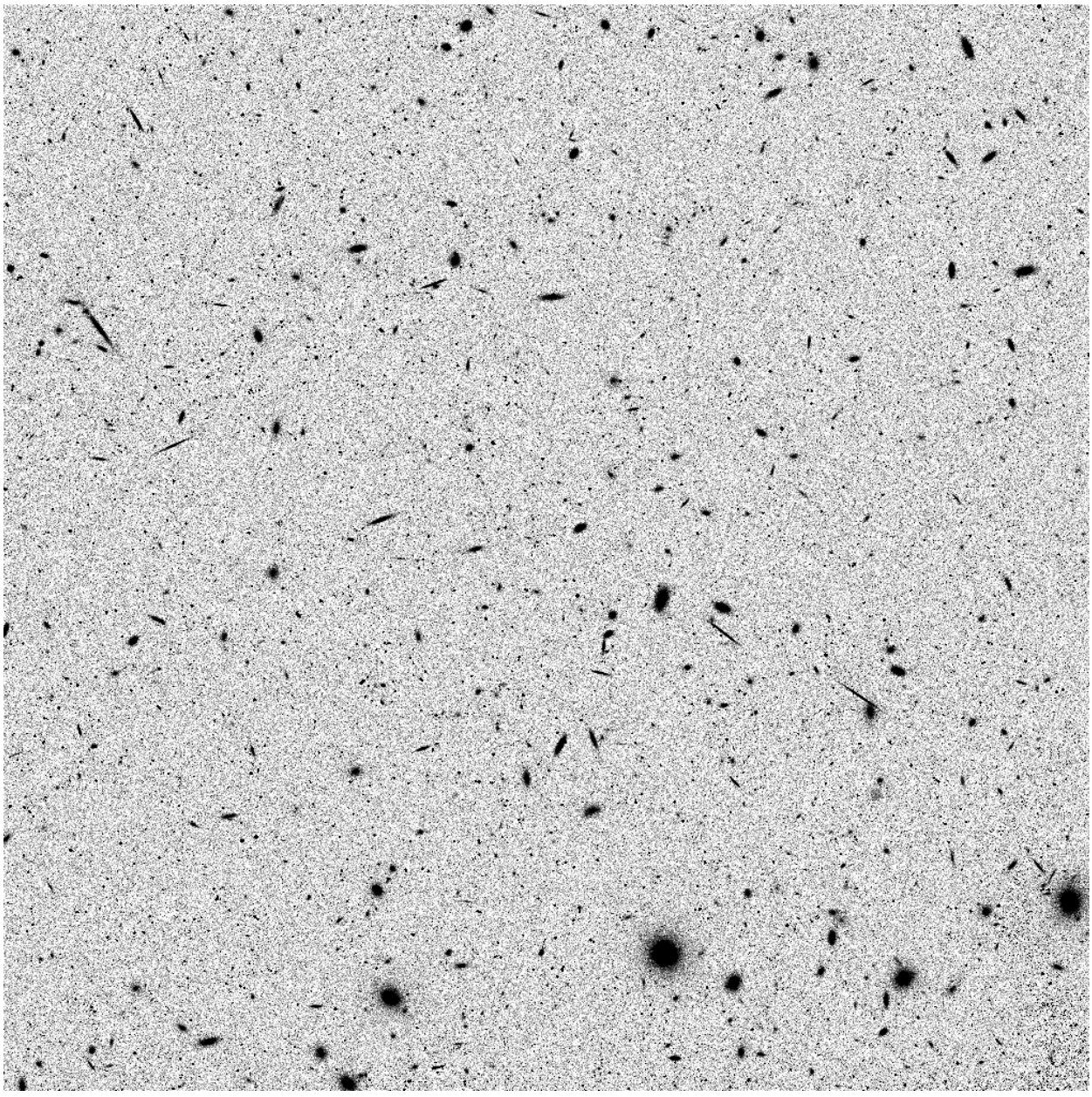

Figure 9: Standard Simulated Image (Method 2 Galaxies)

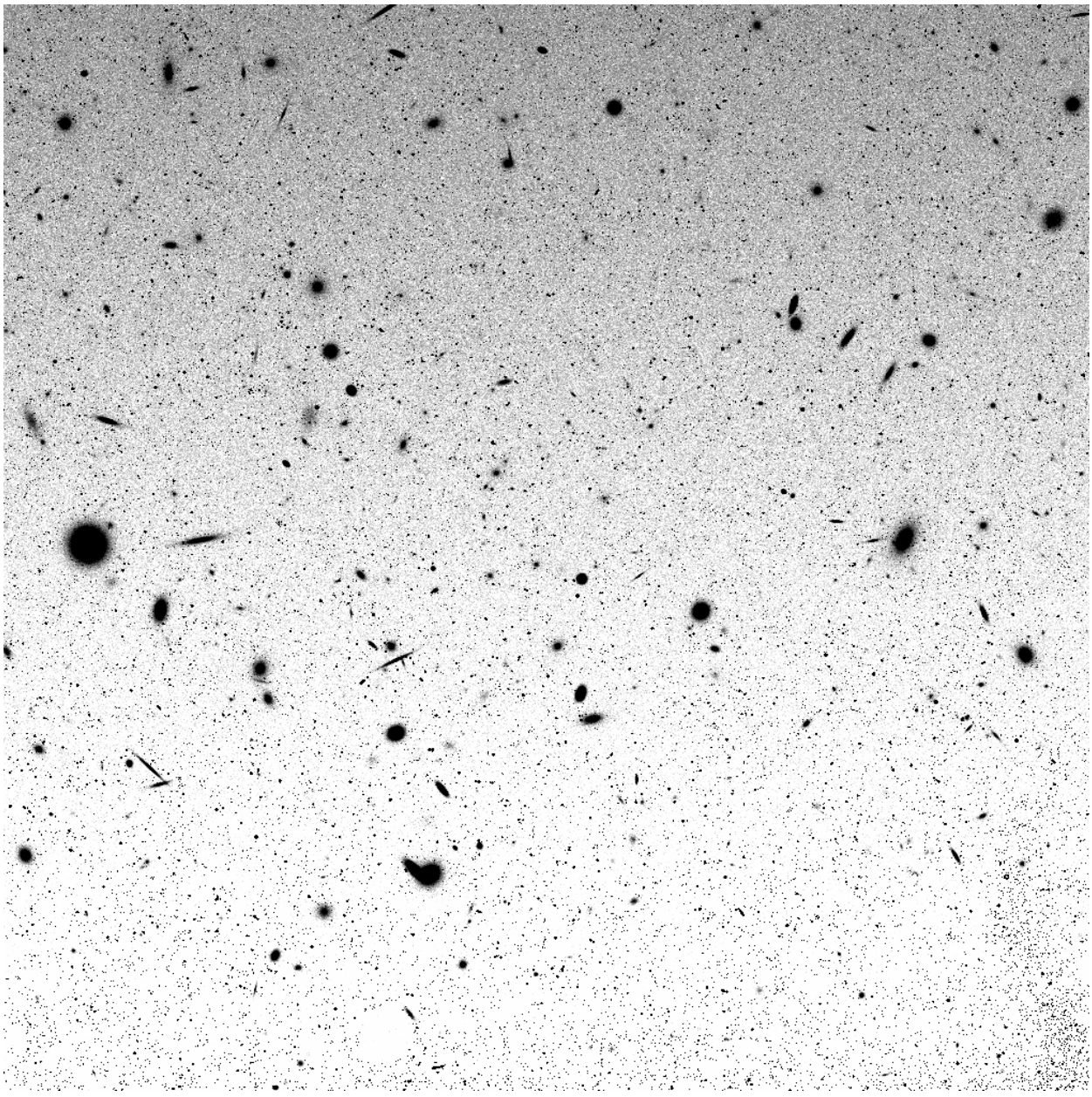

Figure 10: Simulated Image With Gradient (Method 1 Galaxies)

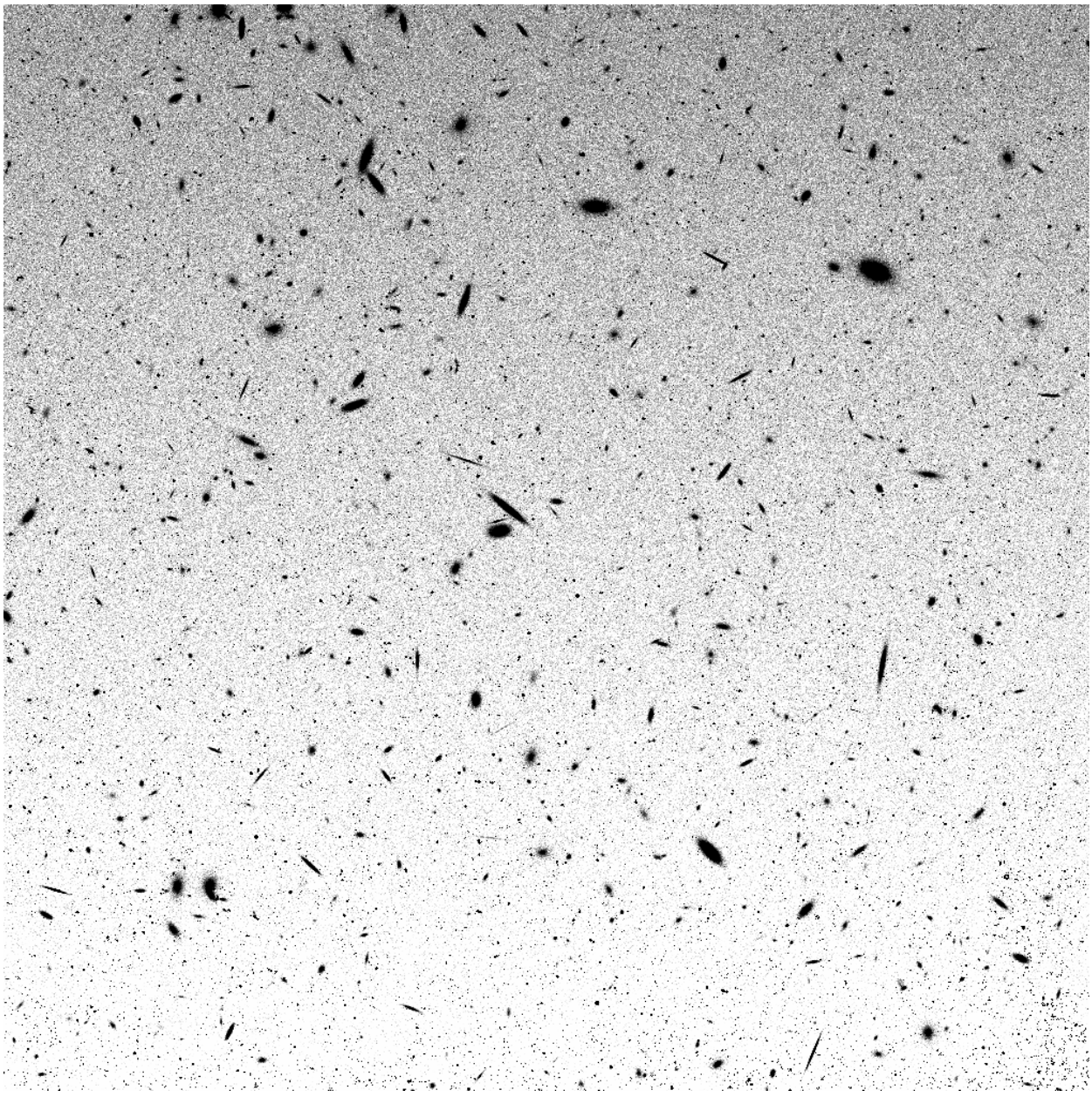

Figure 11: Simulated Image With Gradient (Method 2 Galaxies)

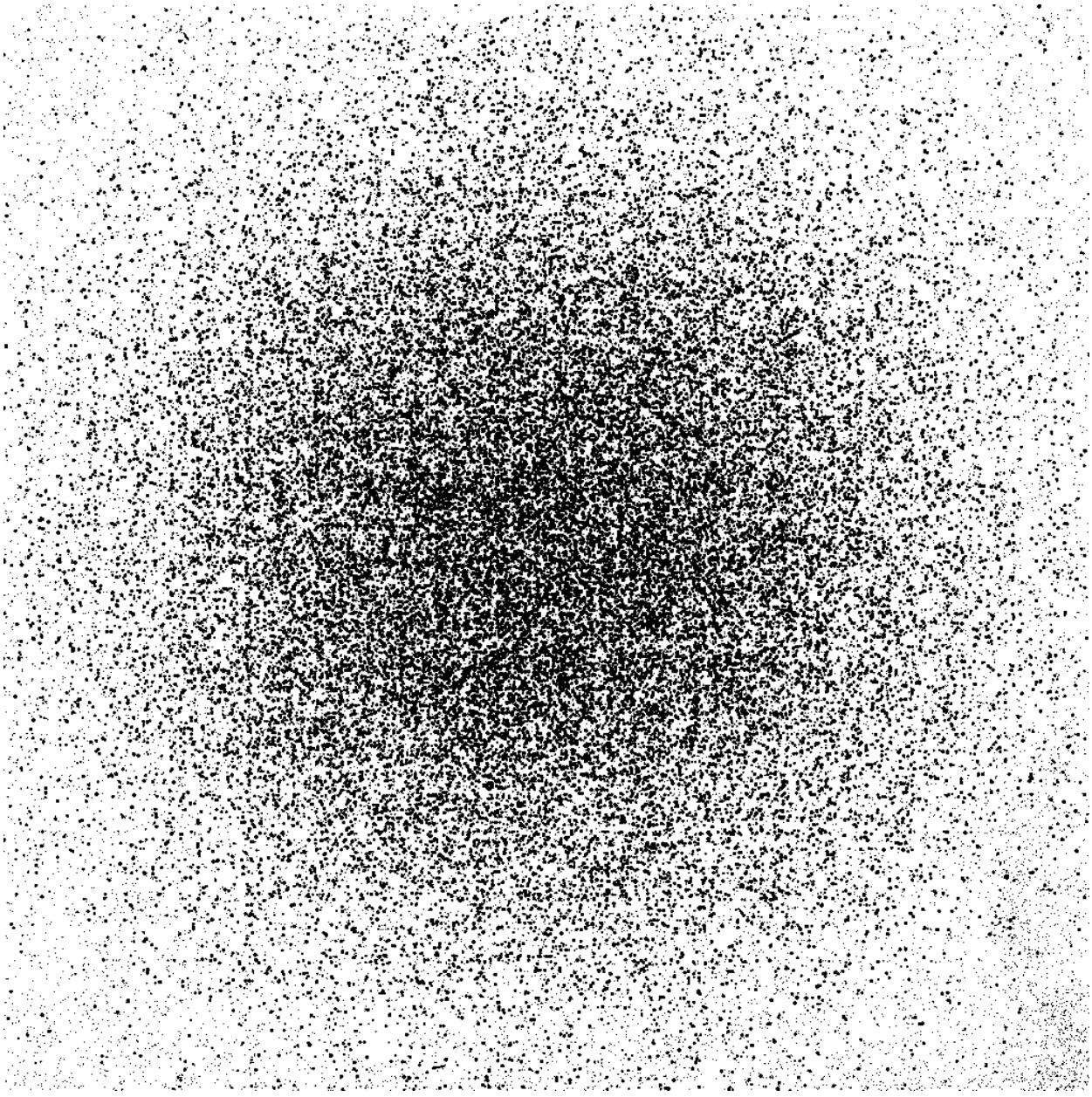

Figure 12: Star Cluster Simulated Image

Figure 13: Hubble WFC3 IR Cosmic Ray Template

The Image Simulation Algorithm

In [1]:
```
# Imports
import time
import galsim
import numpy as np
import random
import glob
import multiprocessing as mp
from astropy.io import fits
from scipy.ndimage import label
import astropy.units as units
import astropy.constants as constants
import os
import scipy
from zipfile import ZipFile

# Start timer
start_time = time.time()
big_fft_params = galsim.GSParams(maximum_fft_size=12186)
```

In [2]:
```
# Create a function that will set variables based on the lines contained in simulate_images_parameters.txt
def define_variables(line):

    # Define the variables in this function as global variables.
    global numimages, sky, all_data, source_catalog_data, percent_change_lower, percent_change_upper, exptime_lower, exptime_upper, star_fwhm, cr_template_name, source_catalogs, cr_

    # Search for the parameters in the parameters file and define them here.
    parameter = line.split(' ')[0]

    if parameter == 'numimages':
        numimages = int(line.split(' ')[2])
        all_data = [None] * numimages
        source_catalog_data = [None] * numimages

    if parameter == 'sky':
        sky = float(line.split(' ')[2])

    if parameter == 'percent_change_lower':
        percent_change_lower = float(line.split(' ')[2])

    if parameter == 'percent_change_upper':
        percent_change_upper = float(line.split(' ')[2])

    if parameter == 'exptime_lower':
        exptime_lower = int(line.split(' ')[2])

    if parameter == 'exptime_upper':
        exptime_upper = int(line.split(' ')[2])

    if parameter == 'star_fwhm':
        star_fwhm = float(line.split(' ')[2])

    if parameter == 'cr_template_name':
        cr_template_name = str(line.split(' ')[2]).strip('\n')

    if parameter == 'source_catalogs':
        source_catalogs = str(line.split(' ')[2]).lower().strip('\n')

    if parameter == 'cr_generation':
        cr_generation = str(line.split(' ')[2]).lower().strip('\n')

    if parameter == 'zero_point':
        zero_point = float(line.split(' ')[2])

    if parameter == 'bpx_generation':
        bpx_generation = str(line.split(' ')[2]).lower().strip('\n')

    if parameter == 'real_image':
        real_image = str(line.split(' ')[2]).strip('\n')

    if parameter == 'min_bpx_error':
        min_bpx_error = float(line.split(' ')[2])

    if parameter == 'star_generation':
        star_generation = str(line.split(' ')[2]).lower().strip('\n')

    if parameter == 'galaxy_generation':
        galaxy_generation = str(line.split(' ')[2]).lower().strip('\n')
```

In [3]:
```
# Create a function to simulate images
def generate_image(imgnum, exptime, sky, percent_change):

    global real_image

    # Define exposure time.
    #exptime = random.randint(exptime_lower, exptime_upper)

    # Make sure exposure time is not greater than 1302.
    if exptime > 1302:
        exptime = 1302

    # Initialize empty lists to hold galaxy and star parameters (only used if the user requested source catalogs).
    object_types = []
    magnitudes = []
    positions = []
    sersic_indices = []
    sizes = []
    inclinations = []
    rotation_angles = []

    # Calcualte the sigma value from the sky value and exposure time.
    value = sky * exptime
```

```
    readnoise = 12.0
    sigma = np.sqrt(value + readnoise**2) / exptime

    # Make a blank image wth ACS size (2048 x 4096) and an initial sky value.
    r = 100
    x = 1014 + r
    y = 1014 + r
    image = galsim.Image(x, y, scale = 0.13, init_value = sky)

    # Only generate stars if the user wants them generated.
    if star_generation == 'true':

        # Create a list of different AB magnitude values that follow the proper distribution, and their respective fluxes (Galsim takes total flux as input).
        star_mags = 26 - np.random.exponential(1 / (0.04 * np.log(10)), size=1000)
        star_mags = np.array([star_mag for star_mag in star_mags if star_mag >= 18])

        # Calculate the total number of stars to create for the magnitude range from 18 to 26.
        star_mag_bins = np.array([18, 19, 20, 21, 22, 23, 24, 25, 26])
        star_count = round(sum(1.0 * 10**(0.04 * (star_mag_bins - 18)))) # * 1.530368245
        #star_count = 10000

        # Randomly select the proper number of star fluxes from the list of star fluxes.
        star_mags = np.random.choice(star_mags, size=int(star_count))
        star_fluxes = (star_mags*units.ABmag).to(units.erg/units.s/units.cm**2/units.Hz) * 8.552784952028751e+29 / (units.erg/units.s/units.cm**2/units.Hz)

        # If the user requested source catalogs, add the star magnitudes to the list of magnitudes.
        if source_catalogs == 'true':
            magnitudes.extend(star_mags)

        '''# Gaussian distributions for star clusters.
        xmin_dist = np.random.normal(size = 10000, loc = (x - r) / 2, scale = (x - r) / 5)
        ymin_dist = np.random.normal(size = 10000, loc = (y - r) / 2, scale = (y - r) / 5)
        # Remove all values from the distribution that fall outside the image bounds.
        xmin_dist = [round(xmin_value) for xmin_value in xmin_dist if 1 <= round(xmin_value) <= (x - r)]
        ymin_dist = [round(ymin_value) for ymin_value in ymin_dist if 1 <= round(ymin_value) <= (y - r)]'''

        # Create a loop that will add all of the randomly generated stars to each image.
        for star_flux in star_fluxes:

            # Make a star as a gaussian with the selected star flux and proper fwhm.
            obj = galsim.Gaussian(flux = star_flux, fwhm = star_fwhm)

            # Randomly determine the x0 and y0 portions of the star's location in the image.
            xmin = random.randint(1, (x - r))
            ymin = random.randint(1, (y - r))
            '''xmin = random.choice(xmin_dist)
            ymin = random.choice(ymin_dist)'''
            xmax = xmin + r
            ymax = ymin + r

            # To put the star in the image, pick out a subsection of the image where the star is going to go. (xmin, xmax, ymin, ymax) is where the star will go.
            bounds = galsim.BoundsI(xmin = xmin, xmax = xmax, ymin = ymin, ymax = ymax)
            sub_psf_image = image[bounds]

            # Define offsets for the stars.
            dx = random.uniform(-0.5, 0.5)
            dy = random.uniform(-0.5, 0.5)

            # On this subsection, draw the object.
            obj.drawImage(sub_psf_image,add_to_image = True, offset=(dx, dy))

            # If the user requested source catalogs, append the object position and type to the proper list.
            if source_catalogs == 'true':
                positions.append(((xmax + xmin)/2 - r/2 + dx, (ymax + ymin)/2 - r/2 + dy))
                object_types.append('STAR')
                sersic_indices.append('N/A')
                sizes.append(star_fwhm)
                inclinations.append('N/A')
                rotation_angles.append('N/A')

    # Set the star count to 0 if no stars were generated.
    else:
        star_count = 0

    # Only generate galaxies if the user wants them generated.
    if galaxy_generation == 'true':

        # Create a list to hold the different AB magnitude values for galaxies, and their respective fluxes.
        galaxy_mags = 26.5 - np.random.exponential(1 / (0.26 * np.log(10)), size=1000)
        galaxy_mags = np.array([galaxy_mag for galaxy_mag in galaxy_mags if galaxy_mag >= 18])

        # Calculate the number of galaxies to create for each galaxy magnitude.
        galaxy_mag_bins = np.array([18, 18.5, 19, 19.5, 20, 20.5, 21, 21.5, 22, 22.5, 23, 23.5, 24, 24.5, 25, 25.5, 26, 26.5])
        galaxy_count = round(sum(1.0 * 10**(0.26 * (galaxy_mag_bins - 18)))) # * 1.530368245

        # Randomly select the proper number of galaxy fluxes from the list of galaxy fluxes.
        galaxy_mags = np.random.choice(galaxy_mags, size=int(galaxy_count))
        galaxy_fluxes = (galaxy_mags*units.ABmag).to(units.erg/units.s/units.cm**2/units.Hz) * 8.552784952028751e+29 / (units.erg/units.s/units.cm**2/units.Hz)

        # Sample sersic indices.
        rng = np.random.uniform(np.exp(-0.38 * 0.3), np.exp(-0.38 * 6.2), size = int(galaxy_count))
        sersic_samples = -np.log(rng) / 0.38

        # If the user requested source catalogs, add the galaxy magnitudes to the list of magnitudes.
        if source_catalogs == 'true':
            magnitudes.extend(galaxy_mags)

        # Create a loop that will add all of the randomly generated galaxies to each image.
        for galaxy_flux, sersic_index in zip(galaxy_fluxes, sersic_samples):

            # Generate a random value for the half light radius of the current galaxy.
            half_light_radius = 5
            while half_light_radius > 2.72:
                half_light_radius = np.random.gamma(shape=2, scale=0.4/2)

            # Generate a random value for the Sersic index and inclination of the current galaxy.
            inclination_angle = random.uniform(0.0, np.pi / 2)
            inclination = galsim._Angle(theta = inclination_angle)
```

```
            # If the Sersic index
            if sersic_index < 2:
                scale_h_over_r = 0.1
            elif sersic_index >= 2:
                scale_h_over_r = 0.9

            # Make a galaxy as a gaussian with the given flux and half light radius.
            obj = galsim.InclinedSersic(n = sersic_index, inclination = inclination, flux = galaxy_flux, half_light_radius = half_light_radius, scale_h_over_r = scale_h_over_r, gsp

            # Perform PSF convolution on the galaxy.
            psf = galsim.Gaussian(fwhm = star_fwhm, gsparams=big_fft_params)
            obj = galsim.Convolve([obj, psf], gsparams=big_fft_params)

            # Randomly determine the x0 and y0 portions of the galaxy's location in the image.
            xmin = random.randint(1, (x - r))
            ymin = random.randint(1, (y - r))
            xmax = xmin + r
            ymax = ymin + r

            # Rotate galaxy by a random angle.
            rotation_angle = random.uniform(0.0, 2 * np.pi)
            theta = rotation_angle * galsim.radians
            obj = obj.rotate(theta)

            # To put the star in the image, pick out a subsection of the image where the star is going to go. (xmin, xmax, ymin, ymax) is where the star will go.
            bounds = galsim.BoundsI(xmin = xmin, xmax = xmax, ymin = ymin, ymax = ymax)
            sub_psf_image = image[bounds]

            # Define offsets for the galaxies.
            dx = random.uniform(-0.5, 0.5)
            dy = random.uniform(-0.5, 0.5)

            # On this subsection, draw the object.
            obj.drawImage(sub_psf_image,add_to_image = True, offset=(dx, dy))

            # If the user requested source catalogs, append the object position and type to the proper list.
            if source_catalogs == 'true':
                positions.append(((xmax + xmin)/2 - r/2 + dx, (ymax + ymin)/2 - r/2 + dy))
                object_types.append('GALAXY')
                sersic_indices.append(sersic_index)
                sizes.append(half_light_radius)
                inclinations.append(inclination_angle * 180 / np.pi)
                rotation_angles.append(rotation_angle * 180 / np.pi)

    # Set the galaxy count to 0 if no galaxies were generated.
    else:
        galaxy_count = 0

    # Save the image.
    image.write('image_'+str(imgnum)+'.fits')

    # Open the fits image that was just generated.
    with fits.open('image_'+str(imgnum)+'.fits', mode='update') as image:

        # Generate random value for the gradient of this image.
        #percent_change = round(random.uniform(percent_change_lower, percent_change_upper), 3)

        # Get the image header and data of the fits file.
        image_header = image[0].header
        image_data = image[0].data

        # Erode the image to the proper dimensions.
        mask = scipy.ndimage.morphology.binary_erosion(image_data, iterations = int(r/2))
        image_data = image_data * mask
        image_data = image_data[~np.all(image_data == 0, axis=1)]
        idx = np.argwhere(np.all(image_data[..., :] == 0, axis=0))
        image_data = np.delete(image_data, idx, axis=1)

        # Add the Poisson noise to the image.
        image_data = np.random.poisson(image_data * exptime)
        image_data = (image_data + np.random.randn(y - r, x - r) * readnoise) / (exptime)

        # Only generate cosmic rays if the user specified that they want cosmic rays generated.
        if cr_generation == 'false':
            cosmic_rays = 0

        else:

            # Open the cosmic ray template.
            with fits.open(cr_template_name) as template:

                # Get the cosmic ray template data.
                template_data = template[0].data

                # Get the cosmic ray header info.
                hdr = template[0].header

                # Create a mask of the cosmic ray data.
                cr_mask = np.where(template_data > 0, 1, 0)

                # Label the cosmic rays and count the total number of cosmic rays in the template.
                labeled_crs, num_crs = label(cr_mask, structure=np.ones((3, 3)))

                # Get the cosmic ray rate of the template.
                cr_rate = num_crs / hdr['EXPTIME']

                # Calculate the expected number of cosmic rays.
                expected_num_crs = np.ceil(exptime * cr_rate)

                # Get the cosmic ray labels.
                cr_labels = list(np.unique(labeled_crs)[1:])

                # Randomly select the cosmic rays to be generated for this image.
                random_crs = random.sample(cr_labels, int(expected_num_crs))

                # Initialize cosmic rays at 0.
                cosmic_rays = 0

                # Add cosmic rays to the simulated image.
```

```
                for cosmic_ray_label in random_crs:

                    # Remove the cosmic ray label from the list of cosmic ray labels.
                    cr_labels.remove(cosmic_ray_label)

                    # Get the indices of the cosmic ray.
                    cr_indices = np.where(labeled_crs == cosmic_ray_label)
                    cr_indices = list(zip(cr_indices[0], cr_indices[1]))

                    # Determine if this cosmic ray gets added.
                    success = False

                    # Loop over the cosmic ray's indices.
                    for cr_index in cr_indices:

                        # Get the value of the cosmic ray at the index and the value of the simulated image pixel at the index.
                        cr_value = template_data[cr_index]
                        img_value = image_data[cr_index]

                        # If the cosmic ray value at this index is greater than the image pixel value at this index, replace the pixel value with the cosmic ray value.
                        if cr_value > img_value:
                            image_data[cr_index] = cr_value
                            success = True

                    # Iterate cosmic ray count.
                    if success == True:
                        cosmic_rays = cosmic_rays + 1

        # Only generate bad pixels if the user specified that they want bad pixels generated.
        if bpx_generation == 'false':
            bpx_count = 0
            real_image_root = 'None'

        else:

            # If the user wants to use a random simulated image for the bad pixel template, select one.
            if real_image.lower() == 'random':

                # Glob all the images in /real_images and select one randomly.
                real_image = random.choice(glob.glob('real_images/*_flt.fits'))

            # Open the real FITS image.
            with fits.open(real_image) as real:

                # Get the root name of the real image.
                real_image_root = os.path.basename(real_image)

                # Get the real image data array.
                real_data = real[1].data
                dummy_variable = 0.0000000000000001
                real_data[real_data == 0] = dummy_variable

                # Get the real image error data array.
                error_data = real[2].data

                # Create a mask of the bad pixels.
                error_mask = np.where(error_data > min_bpx_error, 1, 0)

                # Multiply the error mask by the real data to get an array of bad pixels.
                bpx_array = real_data * error_mask
                temp_mask = image_data * error_mask

                # Count the number of bad pixels in the bad pixel array.
                bpx_count = np.count_nonzero(bpx_array)

                # Set all zeroes to 1 so that non bad pixels can be preserved when multiplying the arrays.
                bpx_array[bpx_array == 0] = 1.0
                temp_mask[temp_mask == 0] = 1.0

                # Multiply the image data by the bad pixel array to add the bad pixels, then divide by the test mask to get the bad pixel values correct.
                image_data = (image_data * bpx_array) / temp_mask
                image_data[image_data == dummy_variable] = 0

                # Save the error mask as a FITS image.
                hdu = fits.PrimaryHDU(error_mask)
                hdu.writeto(f'{bpx_directory}/image_{imgnum}_bad_pixels.fits', overwrite=True)

        # Loop through the rows of the image data.
        y = y - r
        for row, row_number in zip(image_data, range(0, len(image_data))):

            # Apply the gradient to each row.
            image_data[row_number] = row + (percent_change / 100) * (row_number / (y - 1)) * row

        # Remove any negative values that are in the image.
        image_data = np.clip(image_data, a_min=0, a_max=None)

        # Update the image data.
        image[0].data = image_data

        # Update the FITS header with EXPTIME and MAG_ZERO.
        image_header['EXPTIME'] = exptime
        image_header['MAG_ZERO'] = zero_point
        image_header['BPX_IMG'] = real_image_root

        # Save the changes to the fits file.
        image.flush()

    # Move the simulated image to the simulated images directory.
    os.system(f'mv image_{imgnum}.fits {simulated_images_directory}')

    # Notify the user that this image has been created.
    print(f'image_{imgnum}.fits done.')

    return [imgnum, star_count, galaxy_count, cosmic_rays, bpx_count, real_image_root, percent_change, exptime, sky, sigma, object_types, positions, magnitudes, sersic_indices, siz
```

In [4]:

```
# Create a callback function to store data from the asynchronous image simulation process.
def write_to_file(data):
```

```
    global all_data, source_catalog_data

    # Define the image data and add it to the list of all data, which will be written to simulated_images_data.txt.
    image_data = data[0:10]
    all_data[image_data[0] - 1] = image_data

    # Get the source catalog data.
    source_data = data[-7:]
    source_catalog_data[image_data[0] - 1] = source_data

    # Make a source catalog for this image if the user has requested source catalogs.
    if source_catalogs == 'true':

        # Open up a file to write the source catalog data to.
        with open(f'{catalog_directory}/image_{data[0]}_sources.txt', 'w') as catalog:

            # Write the source data to the file.
            format_string = '{:<30} {:<30} {:<30} {:<30} {:<30} {:<30} {:<30} {:<30}'
            catalog.write(format_string.format('OBJECT_TYPE',
                                               'X',
                                               'Y',
                                               'MAGNITUDE(ABS)',
                                               'SERSIC_INDEX',
                                               'STAR_FWHM;GALAXY_HLR(ARCSEC)',
                                               'INCLINATION(DEG)',
                                               'ROTATION(DEG)')+'\n')

            for objtype, xy, mag, n, size, inclination, rotation in zip(source_data[0], source_data[1], source_data[2], source_data[3], source_data[4], source_data[5], source_data[
                catalog.write(format_string.format(objtype,
                                                   xy[0],
                                                   xy[1],
                                                   mag,
                                                   n,
                                                   size,
                                                   inclination,
                                                   rotation)+'\n')
```

In [5]:

```
# Check if simulate_images_parameters.txt is present in the current directory. If it is, get the parameters from it; if it's not, ask the user to put it in the current directory.
if glob.glob('simulate_images_parameters.txt') != []:

    # Open the parameters file.
    with open('simulate_images_parameters.txt', 'r') as parameters_file:

        # Get the lines of the parameters file.
        lines = parameters_file.readlines()

        # Go through the parameter file's lines and define all variables.
        [define_variables(line) for line in lines]

else:

    print('simulate_images_parameters.txt not found. Please put simulate_images_parameters.txt in this directory.')
```

In [6]:

```
# If a directory doesn't exist to hold simulated images, create it.
simulated_images_directory = 'simulated_images'
if os.path.exists(simulated_images_directory) == False:
    os.system(f'mkdir {simulated_images_directory}')

# If a directory does not exist yet for source catalogs, create it.
catalog_directory = 'source_catalogs'
if os.path.exists(catalog_directory) == False:
    os.system(f'mkdir {catalog_directory}')

# If a directory does not exist yet for bad pixel maps, create it.
bpx_directory = 'bad_pixel_maps'
if os.path.exists(bpx_directory) == False:
    os.system(f'mkdir {bpx_directory}')
```

In [7]:

```
# Notify the user that the images are being generated.
print('\nSimulating images (This can take a while for high exposure times)...\n')

# Open a file to record image data in.
with open(f'{simulated_images_directory}/simulated_images_data.txt', 'w') as data:

    # Create list of exposure times, sky values, and gradients.
    exptimes = [50,100,200,300,400,500,600,700,800,900,1100,1302]
    skys = np.array([1/4]) * np.pi
    gradients = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]

    # Create a grid of all unique exposure time/sky value permutations.
    imgnums_exptimes_skys = []
    imgnum = 0
    for exptime in exptimes:
        for sky in skys:
            for percent_change in gradients:
                imgnum = imgnum + 1
                imgnums_exptimes_skys.append([imgnum, exptime, sky, percent_change])

    # Empty arrays for data file and source catalog data.
    all_data = [None] * imgnum
    source_catalog_data = [None] * imgnum

    # Initialize a multiprocessing pool that will use all cores of the user's computer.
    pool = mp.Pool(mp.cpu_count())

    # Generate images using multiprocessing.
    [pool.apply_async(generate_image, args = ([imgnum_exptime_sky[0], imgnum_exptime_sky[1], imgnum_exptime_sky[2], imgnum_exptime_sky[3]]), callback = write_to_file) for imgnum_ex

    # Close the pool and wait for all asynchronous processes to complete.
    pool.close()
    pool.join()

    # Write all the data to a file.
    format_string = '{:<11} {:<11} {:<11} {:<11} {:<11} {:<20} {:<11} {:<11} {:<20} {:<20}'
    data.write(format_string.format('IMAGE',
```

```
                                    'STARS',
                                    'GALAXIES',
                                    'COSMIC_RAYS',
                                    'BAD_PIXELS',
                                    'BPX_IMAGE',
                                    '%_CHANGE',
                                    'EXPTIME',
                                    'SKY',
                                    'RMS')+'\n')

    for paragraph in all_data:
        data.write(format_string.format(paragraph[0],
                                        paragraph[1],
                                        paragraph[2],
                                        paragraph[3],
                                        paragraph[4],
                                        paragraph[5],
                                        paragraph[6],
                                        paragraph[7],
                                        paragraph[8],
                                        paragraph[9])+'\n')
```

```
Simulating images (This can take a while for high exposure times)...

image_4.fits done.
image_7.fits done.
image_2.fits done.
image_20.fits done.
image_8.fits done.
image_5.fits done.image_1.fits done.image_19.fits done.


image_6.fits done.image_17.fits done.

image_10.fits done.
image_3.fits done.image_13.fits done.

image_9.fits done.
image_18.fits done.
image_14.fits done.
image_16.fits done.
image_15.fits done.image_12.fits done.

image_11.fits done.
image_28.fits done.
image_30.fits done.
image_22.fits done.
image_23.fits done.
image_31.fits done.
image_26.fits done.
image_27.fits done.
image_29.fits done.
image_25.fits done.
image_32.fits done.
image_21.fits done.
image_24.fits done.
image_35.fits done.
image_38.fits done.
image_52.fits done.
image_55.fits done.
image_59.fits done.
image_57.fits done.image_54.fits done.

image_60.fits done.
image_56.fits done.
image_53.fits done.
image_37.fits done.
image_63.fits done.
image_62.fits done.
image_36.fits done.
image_64.fits done.
image_40.fits done.
image_61.fits done.
image_58.fits done.
image_48.fits done.
image_51.fits done.
image_50.fits done.
image_47.fits done.
image_42.fits done.
image_44.fits done.
image_34.fits done.
image_49.fits done.
image_41.fits done.
image_43.fits done.
image_33.fits done.
image_46.fits done.
image_39.fits done.
image_45.fits done.
image_67.fits done.
image_83.fits done.
image_74.fits done.
image_68.fits done.
image_85.fits done.
image_69.fits done.
image_77.fits done.
image_80.fits done.
image_78.fits done.
image_73.fits done.
image_72.fits done.
image_84.fits done.
image_71.fits done.
image_86.fits done.
image_66.fits done.
image_70.fits done.
image_82.fits done.
image_65.fits done.
image_79.fits done.
image_87.fits done.
image_75.fits done.
image_88.fits done.
image_96.fits done.
image_94.fits done.
image_90.fits done.
image_81.fits done.
image_95.fits done.
```

image_92.fits done.
image_93.fits done.
image_91.fits done.
image_76.fits done.
image_89.fits done.
image_100.fits done.
image_99.fits done.
image_102.fits done.
image_107.fits done.
image_97.fits done.
image_98.fits done.
image_104.fits done.
image_109.fits done.
image_101.fits done.
image_106.fits done.
image_111.fits done.
image_120.fits done.
image_115.fits done.
image_103.fits done.
image_110.fits done.
image_119.fits done.
image_112.fits done.
image_108.fits done.
image_105.fits done.
image_117.fits done.
image_121.fits done.
image_124.fits done.
image_113.fits done.
image_118.fits done.
image_122.fits done.
image_125.fits done.
image_123.fits done.
image_116.fits done.
image_126.fits done.
image_127.fits done.
image_114.fits done.
image_128.fits done.
image_129.fits done.
image_142.fits done.
image_137.fits done.
image_131.fits done.
image_130.fits done.
image_132.fits done.
image_136.fits done.
image_139.fits done.
image_141.fits done.
image_138.fits done.
image_148.fits done.
image_147.fits done.
image_134.fits done.
image_140.fits done.
image_143.fits done.
image_146.fits done.
image_133.fits done.
image_144.fits done.
image_135.fits done.
image_151.fits done.
image_153.fits done.
image_152.fits done.
image_157.fits done.
image_150.fits done.
image_149.fits done.
image_145.fits done.
image_156.fits done.
image_155.fits done.
image_158.fits done.
image_154.fits done.
image_159.fits done.
image_160.fits done.
image_165.fits done.
image_164.fits done.
image_166.fits done.
image_168.fits done.
image_176.fits done.
image_167.fits done.
image_161.fits done.
image_170.fits done.
image_173.fits done.
image_162.fits done.
image_163.fits done.
image_179.fits done.
image_169.fits done.
image_174.fits done.
image_171.fits done.
image_172.fits done.
image_184.fits done.
image_178.fits done.
image_175.fits done.
image_181.fits done.
image_180.fits done.
image_177.fits done.
image_183.fits done.
image_182.fits done.
image_185.fits done.
image_190.fits done.
image_187.fits done.
image_186.fits done.
image_189.fits done.
image_188.fits done.
image_191.fits done.
image_192.fits done.
image_193.fits done.
image_195.fits done.
image_194.fits done.
image_198.fits done.
image_200.fits done.
image_201.fits done.
image_197.fits done.
image_196.fits done.
image_208.fits done.
image_205.fits done.
image_206.fits done.
image_199.fits done.
image_202.fits done.
image_212.fits done.
image_204.fits done.
image_203.fits done.

```
image_207.fits done.
image_209.fits done.
image_210.fits done.
image_213.fits done.
image_216.fits done.
image_217.fits done.
image_214.fits done.
image_215.fits done.
image_211.fits done.
image_218.fits done.
image_220.fits done.
image_219.fits done.
image_222.fits done.
image_223.fits done.
image_221.fits done.
image_224.fits done.
image_225.fits done.
image_232.fits done.
image_226.fits done.
image_227.fits done.
image_228.fits done.
image_229.fits done.
image_231.fits done.
image_237.fits done.
image_230.fits done.
image_238.fits done.
image_233.fits done.
image_236.fits done.
image_239.fits done.
image_235.fits done.
image_234.fits done.
image_240.fits done.
```

In [8]:

```
# Create zip folders of files.
imgsets = []
imgset = []

# Divide the total number of images into sets of 15.
for num in range(1, imgnum + 1):
    if ((num) / 122).is_integer() != True and num != imgnum:
        imgset.append(num)
    elif num == imgnum:
        imgset.append(num)
        imgsets.append(imgset)
        imgset = []
    else:
        imgset.append(num)
        imgsets.append(imgset)
        imgset = []

# Loop through the image sets.
for imgset in imgsets:

    # Define max and min values of the image set.
    start = min(imgset)
    end = max(imgset)

    # Open a new zip folder.
    with ZipFile(f'{simulated_images_directory}/images{start}-{end}.zip', 'w') as zipObj:

        # Loop through the image numbers in the image set.
        for num in imgset:

            # Add the files of the image set to the zip folder.
            zipObj.write(f'{simulated_images_directory}/image_{num}.fits')

    if bpx_generation == 'true':

        # Open a new zip folder.
        with ZipFile(f'{bpx_directory}/images{start}-{end}_bad_pixels.zip', 'w') as zipObj:

            # Loop through the image numbers in the image set.
            for num in imgset:

                # Add the files of the image set to the zip folder.
                zipObj.write(f'{bpx_directory}/image_{num}_bad_pixels.fits')

if source_catalogs == 'true':

    # Zip all of the source catalogs.
    with ZipFile(f'{catalog_directory}/images1-{imgnum}_sources.zip', 'w') as zipObj:
        for num in range(1, imgnum + 1):
            zipObj.write(f'{catalog_directory}/image_{num}_sources.txt')

print('\nImage data written to simulated_images/simulated_images_data.txt.')

if source_catalogs == 'true':
    print('\nSource catalogs have been written to source_catalogs/.')

if bpx_generation == 'true':
    print('\nBad pixel maps have been written to bad_pixel_maps/.')

#generate_image(1, 1302, np.pi / 3)

# End timer.
print('\nTotal Runtime: '+str((time.time() - start_time))+' seconds\n')
```

```
Image data written to simulated_images/simulated_images_data.txt.

Source catalogs have been written to source_catalogs/.

Total Runtime: 1638.169667005539 seconds
```

Inputs for Flat Images (Method 1 Galaxies With Cosmic Rays)

In [ ]:

IMAGE	STARS	GALAXIES	COSMIC_RAYS	BAD_PIXELS	BPX_IMAGE	%_CHANGE	EXPTIME	SKY	RMS
1	13	624	729	0	None	0.0	50	0.6283185307179586	0.26488935541912434
2	13	624	729	0	None	0.0	50	0.7853981633974483	0.270754433514853
3	13	624	729	0	None	0.0	50	0.8975979010256552	0.27486716431853603
4	13	624	729	0	None	0.0	50	1.0471975511965976	0.2802569375125832
5	13	624	729	0	None	0.0	50	1.2566370614359172	0.287632997461554
6	13	624	729	0	None	0.0	50	1.3962634015954636	0.29244703457533855
7	13	624	729	0	None	0.0	50	1.5707963267948966	0.29835536954427
8	13	624	729	0	None	0.0	50	1.7951958020513104	0.3057841003731656
9	13	624	729	0	None	0.0	50	2.0943951023931953	0.3154170287854857
10	13	624	729	0	None	0.0	50	2.5132741228718345	0.328428808811646
11	13	624	729	0	None	0.0	50	2.792526803190927	0.33682419162497595
12	13	624	729	0	None	0.0	50	3.141592653589793	0.34703292793594653
13	13	624	1458	0	None	0.0	100	0.6283185307179586	0.143816498730777
14	13	624	1458	0	None	0.0	100	0.7853981633974483	0.149177684772135
15	13	624	1458	0	None	0.0	100	0.8975979010256552	0.1528920501865828
16	13	624	1458	0	None	0.0	100	1.0471975511965976	0.15770851439274286
17	13	624	1458	0	None	0.0	100	1.2566370614359172	0.164214404405823
18	13	624	1458	0	None	0.0	100	1.3962634015954636	0.16841209581248798
19	13	624	1458	0	None	0.0	100	1.5707963267948966	0.17351646396797327
20	13	624	1458	0	None	0.0	100	1.7951958020513104	0.1798665005511396
21	13	624	1458	0	None	0.0	100	2.0943951023931953	0.18799986974445473
22	13	624	1458	0	None	0.0	100	2.5132741228718345	0.198828421581821
23	13	624	1458	0	None	0.0	100	2.792526803190927	0.20573105752877777
24	13	624	1458	0	None	0.0	100	3.141592653589793	0.21404655226351565
25	13	624	2915	0	None	0.0	200	0.6283185307179586	0.0821072022029115
26	13	624	2915	0	None	0.0	200	0.7853981633974483	0.08675823198398663
27	13	624	2915	0	None	0.0	200	0.8975979010256552	0.0899332502755698
28	13	624	2915	0	None	0.0	200	1.0471975511965976	0.09399993487222737
29	13	624	2915	0	None	0.0	200	1.2566370614359172	0.0994142107909105
30	13	624	2915	0	None	0.0	200	1.3962634015954636	0.10286552876438888
31	13	624	2915	0	None	0.0	200	1.5707963267948966	0.10702327613175783
32	13	624	2915	0	None	0.0	200	1.7951958020513104	0.1121426725660511
33	13	624	2915	0	None	0.0	200	2.0943951023931953	0.11862535779489129
34	13	624	2915	0	None	0.0	200	2.5132741228718345	0.12714704327808482
35	13	624	2915	0	None	0.0	200	2.792526803190927	0.1325240884366108
36	13	624	2915	0	None	0.0	200	3.141592653589793	0.13895309736723743
37	13	624	4373	0	None	0.0	300	0.6283185307179586	0.06078153586734376
38	13	624	4373	0	None	0.0	300	0.7853981633974483	0.06494608439306787
39	13	624	4373	0	None	0.0	300	0.8975979010256552	0.06776424575997914
40	13	624	4373	0	None	0.0	300	1.0471975511965976	0.0713488507545052
41	13	624	4373	0	None	0.0	300	1.2566370614359172	0.07608409955297092
42	13	624	4373	0	None	0.0	300	1.3962634015954636	0.07908357186326086
43	13	624	4373	0	None	0.0	300	1.5707963267948966	0.08268003238982788
44	13	624	4373	0	None	0.0	300	1.7951958020513104	0.08708608388736802
45	13	624	4373	0	None	0.0	300	2.0943951023931953	0.09263539824482495
46	13	624	4373	0	None	0.0	300	2.5132741228718345	0.09988783914757983
47	13	624	4373	0	None	0.0	300	2.792526803190927	0.10444339460829052
48	13	624	4372	0	None	0.0	300	3.141592653589793	0.10987254212024938
49	13	624	5830	0	None	0.0	400	0.6283185307179586	0.04970710539545525
50	13	624	5830	0	None	0.0	400	0.7853981633974483	0.053511638065878914
51	13	624	5830	0	None	0.0	400	0.8975979010256552	0.05607133628302555
52	13	624	5830	0	None	0.0	400	1.0471975511965976	0.059312678897445646
53	13	624	5830	0	None	0.0	400	1.2566370614359172	0.06357352163904241
54	13	624	5830	0	None	0.0	400	1.3962634015954636	0.0662620442183054
55	13	624	5830	0	None	0.0	400	1.5707963267948966	0.06947654868361872
56	13	624	5830	0	None	0.0	400	1.7951958020513104	0.0734029257259428
57	13	624	5830	0	None	0.0	400	2.0943951023931953	0.0783325459562179
58	13	624	5830	0	None	0.0	400	2.5132741228718345	0.08475367429899182
59	13	624	5830	0	None	0.0	400	2.792526803190927	0.08877678191947103
60	13	624	5830	0	None	0.0	400	3.141592653589793	0.09356271497757257
61	13	624	7288	0	None	0.0	500	0.6283185307179586	0.04280931045270313
62	13	624	7288	0	None	0.0	500	0.7853981633974483	0.04633353350215043
63	13	624	7288	0	None	0.0	500	0.8975979010256552	0.04869492583474494
64	13	624	7288	0	None	0.0	500	1.0471975511965976	0.051675865763363804
65	13	624	7288	0	None	0.0	500	1.2566370614359172	0.05558123894689497
66	13	624	7288	0	None	0.0	500	1.3962634015954636	0.058039011045941565
67	13	624	7288	0	None	0.0	500	1.5707963267948966	0.06097206453442259
68	13	624	7288	0	None	0.0	500	1.7951958020513104	0.06454759177616637
69	13	624	7288	0	None	0.0	500	2.0943951023931953	0.06902745978801762
70	13	624	7288	0	None	0.0	500	2.5132741228718345	0.07485017198205805
71	13	624	7288	0	None	0.0	500	2.792526803190927	0.07849237928857714
72	13	624	7288	0	None	0.0	500	3.141592653589793	0.08282019866662713
73	13	624	8745	0	None	0.0	600	0.6283185307179586	0.038042049776485 46
74	13	624	8745	0	None	0.0	600	0.7853981633974483	0.04134001619491394
75	13	624	8745	0	None	0.0	600	0.8975979010256552	0.04354304194368401
76	13	624	8745	0	None	0.0	600	1.0471975511965976	0.04631769912241247
77	13	624	8745	0	None	0.0	600	1.2566370614359172	0.049943919573789915
78	13	624	8745	0	None	0.0	600	1.3962634015954636	0.05222169730414526
79	13	624	8745	0	None	0.0	600	1.5707963267948966	0.05493627106012469
80	13	624	8745	0	None	0.0	600	1.7951958020513104	0.0582408190483174
81	13	624	8745	0	None	0.0	600	2.0943951023931953	0.06237514331838172
82	13	624	8745	0	None	0.0	600	2.5132741228718345	0.06774060971667137
83	13	624	8745	0	None	0.0	600	2.792526803190927	0.07109297671817902
84	13	624	8745	0	None	0.0	600	3.141592653589793	0.07507321596936546
85	13	624	10203	0	None	0.0	700	0.6283185307179586	0.03451775560557296
86	13	624	10203	0	None	0.0	700	0.7853981633974483	0.0376281135230359
87	13	624	10203	0	None	0.0	700	0.8975979010256552	0.039700884962067834
88	13	624	10203	0	None	0.0	700	1.0471975511965976	0.04230690313329296
89	13	624	10203	0	None	0.0	700	1.2566370614359172	0.04570638197310872
90	13	624	10203	0	None	0.0	700	1.3962634015954636	0.04783868260413994
91	13	624	10203	0	None	0.0	700	1.5707963267948966	0.050377299486817934
92	13	624	10203	0	None	0.0	700	1.7951958020513104	0.0534644085586129
93	13	624	10202	0	None	0.0	700	2.0943951023931953	0.05732251350420058
94	13	624	10203	0	None	0.0	700	2.5132741228718345	0.06232390516585934

95	13	624	10203	0	None	0.0	700	2.792526803190927	0.06544617296358035
96	13	624	10203	0	None	0.0	700	3.141592653589793	0.06915104522817196
97	13	624	11660	0	None	0.0	800	0.6283185307179586	0.031786760819521205
98	13	624	11660	0	None	0.0	800	0.7853981633974483	0.03473827434180936
99	13	624	11659	0	None	0.0	800	0.8975979010256552	0.0367014628629714
100	13	624	11660	0	None	0.0	800	1.0471975511965976	0.03916627297810895
101	13	624	11659	0	None	0.0	800	1.2566370614359172	0.04237683714949591
102	13	624	11660	0	None	0.0	800	1.3962634015954636	0.04438839095973551
103	13	624	11660	0	None	0.0	800	1.5707963267948966	0.046781357488786285
104	13	624	11660	0	None	0.0	800	1.7951958020513104	0.0496889801924344
105	13	624	11660	0	None	0.0	800	2.0943951023931953	0.05331973253863427
106	13	624	11659	0	None	0.0	800	2.5132741228718345	0.05802234615723319
107	13	624	11660	0	None	0.0	800	2.792526803190927	0.06095620152198346
108	13	624	11660	0	None	0.0	800	3.141592653589793	0.0644359435174751
109	13	624	13118	0	None	0.0	900	0.6283185307179586	0.029595767916638177
110	13	624	13118	0	None	0.0	900	0.7853981633974483	0.03241052921158404
111	13	624	13118	0	None	0.0	900	0.8975979010256552	0.03427985966886963
112	13	624	13114	0	None	0.0	900	1.0471975511965976	0.0366241807067498
113	13	624	13118	0	None	0.0	900	1.2566370614359172	0.039674187822477745
114	13	624	13117	0	None	0.0	900	1.3962634015954636	0.04158342887892114
115	13	624	13118	0	None	0.0	900	1.5707963267948966	0.04385324423314776
116	13	624	13118	0	None	0.0	900	1.7951958020513104	0.04660943874428238
117	13	624	13118	0	None	0.0	900	2.0943951023931953	0.05004881064624364
118	13	624	13118	0	None	0.0	900	2.5132741228718345	0.05450050074053178
119	13	624	13118	0	None	0.0	900	2.792526803190927	0.05727639423775565
120	13	624	13118	0	None	0.0	900	3.141592653589793	0.060567617435114914
121	13	624	16032	0	None	0.0	1100	0.6283185307179586	0.0262717895993376
122	13	624	16033	0	None	0.0	1100	0.7853981633974483	0.02886185362765404
123	13	624	16033	0	None	0.0	1100	0.8975979010256552	0.030577873639354238
124	13	624	16032	0	None	0.0	1100	1.0471975511965976	0.03272622859844835
125	13	624	16033	0	None	0.0	1100	1.2566370614359172	0.03551627222895044
126	13	624	16033	0	None	0.0	1100	1.3962634015954636	0.03726041638991163
127	13	624	16033	0	None	0.0	1100	1.5707963267948966	0.03933198348908231
128	13	624	16033	0	None	0.0	1100	1.7951958020513104	0.04184500505610938
129	13	624	16033	0	None	0.0	1100	2.0943951023931953	0.044977814665599164
130	13	624	16033	0	None	0.0	1100	2.5132741228718345	0.0490285928986242
131	13	624	16033	0	None	0.0	1100	2.792526803190927	0.05155258475223507
132	13	624	16033	0	None	0.0	1100	3.141592653589793	0.05454357511117213
133	13	624	18977	0	None	0.0	1302	0.6283185307179586	0.023822784164748544
134	13	624	18976	0	None	0.0	1302	0.7853981633974483	0.026232993053220383
135	13	624	18976	0	None	0.0	1302	0.8975979010256552	0.027827052273340917
136	13	624	18977	0	None	0.0	1302	1.0471975511965976	0.029820206624823493
137	13	624	18976	0	None	0.0	1302	1.2566370614359172	0.03240531687918904
138	13	624	18977	0	None	0.0	1302	1.3962634015954636	0.0340197656464049
139	13	624	18977	0	None	0.0	1302	1.5707963267948966	0.03593597529469436
140	13	624	18977	0	None	0.0	1302	1.7951958020513104	0.038258909390712664
141	13	624	18977	0	None	0.0	1302	2.0943951023931953	0.04115269028404583
142	13	624	18977	0	None	0.0	1302	2.5132741228718345	0.04489168737529893
143	13	624	18976	0	None	0.0	1302	2.792526803190927	0.04722015862524065
144	13	624	18977	0	None	0.0	1302	3.141592653589793	0.04997842646664691

Inputs for Gradient Images (Method 1 Galaxies With Cosmic Rays)

In [ ]:

IMAGE	STARS	GALAXIES	COSMIC_RAYS	BAD_PIXELS	BPX_IMAGE	%_CHANGE	EXPTIME	SKY	RMS
1	13	624	729	0	None	1	50	0.7853981633974483	0.270754433514853
2	13	624	729	0	None	2	50	0.7853981633974483	0.270754433514853
3	13	624	729	0	None	3	50	0.7853981633974483	0.270754433514853
4	13	624	729	0	None	4	50	0.7853981633974483	0.270754433514853
5	13	624	729	0	None	5	50	0.7853981633974483	0.270754433514853
6	13	624	729	0	None	6	50	0.7853981633974483	0.270754433514853
7	13	624	729	0	None	7	50	0.7853981633974483	0.270754433514853
8	13	624	729	0	None	8	50	0.7853981633974483	0.270754433514853
9	13	624	729	0	None	9	50	0.7853981633974483	0.270754433514853
10	13	624	729	0	None	10	50	0.7853981633974483	0.270754433514853
11	13	624	729	0	None	11	50	0.7853981633974483	0.270754433514853
12	13	624	729	0	None	12	50	0.7853981633974483	0.270754433514853
13	13	624	729	0	None	13	50	0.7853981633974483	0.270754433514853
14	13	624	729	0	None	14	50	0.7853981633974483	0.270754433514853
15	13	624	729	0	None	15	50	0.7853981633974483	0.270754433514853
16	13	624	729	0	None	16	50	0.7853981633974483	0.270754433514853
17	13	624	729	0	None	17	50	0.7853981633974483	0.270754433514853
18	13	624	729	0	None	18	50	0.7853981633974483	0.270754433514853
19	13	624	729	0	None	19	50	0.7853981633974483	0.270754433514853
20	13	624	729	0	None	20	50	0.7853981633974483	0.270754433514853
21	13	624	1458	0	None	1	100	0.7853981633974483	0.149177684772135
22	13	624	1458	0	None	2	100	0.7853981633974483	0.149177684772135
23	13	624	1458	0	None	3	100	0.7853981633974483	0.149177684772135
24	13	624	1458	0	None	4	100	0.7853981633974483	0.149177684772135
25	13	624	1458	0	None	5	100	0.7853981633974483	0.149177684772135
26	13	624	1458	0	None	6	100	0.7853981633974483	0.149177684772135
27	13	624	1458	0	None	7	100	0.7853981633974483	0.149177684772135
28	13	624	1458	0	None	8	100	0.7853981633974483	0.149177684772135
29	13	624	1458	0	None	9	100	0.7853981633974483	0.149177684772135
30	13	624	1458	0	None	10	100	0.7853981633974483	0.149177684772135
31	13	624	1458	0	None	11	100	0.7853981633974483	0.149177684772135
32	13	624	1457	0	None	12	100	0.7853981633974483	0.149177684772135
33	13	624	1458	0	None	13	100	0.7853981633974483	0.149177684772135
34	13	624	1458	0	None	14	100	0.7853981633974483	0.149177684772135
35	13	624	1458	0	None	15	100	0.7853981633974483	0.149177684772135
36	13	624	1458	0	None	16	100	0.7853981633974483	0.149177684772135
37	13	624	1458	0	None	17	100	0.7853981633974483	0.149177684772135
38	13	624	1458	0	None	18	100	0.7853981633974483	0.149177684772135
39	13	624	1458	0	None	19	100	0.7853981633974483	0.149177684772135
40	13	624	1458	0	None	20	100	0.7853981633974483	0.149177684772135
41	13	624	2915	0	None	1	200	0.7853981633974483	0.08675823198398663
42	13	624	2915	0	None	2	200	0.7853981633974483	0.08675823198398663
43	13	624	2915	0	None	3	200	0.7853981633974483	0.08675823198398663

44	13	624	2915	0	None	4	200	0.7853981633974483	0.08675823198398663
45	13	624	2915	0	None	5	200	0.7853981633974483	0.08675823198398663
46	13	624	2915	0	None	6	200	0.7853981633974483	0.08675823198398663
47	13	624	2915	0	None	7	200	0.7853981633974483	0.08675823198398663
48	13	624	2915	0	None	8	200	0.7853981633974483	0.08675823198398663
49	13	624	2915	0	None	9	200	0.7853981633974483	0.08675823198398663
50	13	624	2915	0	None	10	200	0.7853981633974483	0.08675823198398663
51	13	624	2915	0	None	11	200	0.7853981633974483	0.08675823198398663
52	13	624	2915	0	None	12	200	0.7853981633974483	0.08675823198398663
53	13	624	2915	0	None	13	200	0.7853981633974483	0.08675823198398663
54	13	624	2915	0	None	14	200	0.7853981633974483	0.08675823198398663
55	13	624	2915	0	None	15	200	0.7853981633974483	0.08675823198398663
56	13	624	2915	0	None	16	200	0.7853981633974483	0.08675823198398663
57	13	624	2915	0	None	17	200	0.7853981633974483	0.08675823198398663
58	13	624	2915	0	None	18	200	0.7853981633974483	0.08675823198398663
59	13	624	2915	0	None	19	200	0.7853981633974483	0.08675823198398663
60	13	624	2915	0	None	20	200	0.7853981633974483	0.08675823198398663
61	13	624	4373	0	None	1	300	0.7853981633974483	0.06494608439306787
62	13	624	4373	0	None	2	300	0.7853981633974483	0.06494608439306787
63	13	624	4373	0	None	3	300	0.7853981633974483	0.06494608439306787
64	13	624	4373	0	None	4	300	0.7853981633974483	0.06494608439306787
65	13	624	4373	0	None	5	300	0.7853981633974483	0.06494608439306787
66	13	624	4373	0	None	6	300	0.7853981633974483	0.06494608439306787
67	13	624	4373	0	None	7	300	0.7853981633974483	0.06494608439306787
68	13	624	4373	0	None	8	300	0.7853981633974483	0.06494608439306787
69	13	624	4373	0	None	9	300	0.7853981633974483	0.06494608439306787
70	13	624	4373	0	None	10	300	0.7853981633974483	0.06494608439306787
71	13	624	4373	0	None	11	300	0.7853981633974483	0.06494608439306787
72	13	624	4373	0	None	12	300	0.7853981633974483	0.06494608439306787
73	13	624	4373	0	None	13	300	0.7853981633974483	0.06494608439306787
74	13	624	4373	0	None	14	300	0.7853981633974483	0.06494608439306787
75	13	624	4373	0	None	15	300	0.7853981633974483	0.06494608439306787
76	13	624	4373	0	None	16	300	0.7853981633974483	0.06494608439306787
77	13	624	4373	0	None	17	300	0.7853981633974483	0.06494608439306787
78	13	624	4373	0	None	18	300	0.7853981633974483	0.06494608439306787
79	13	624	4373	0	None	19	300	0.7853981633974483	0.06494608439306787
80	13	624	4373	0	None	20	300	0.7853981633974483	0.06494608439306787
81	13	624	5830	0	None	1	400	0.7853981633974483	0.053511638065878914
82	13	624	5830	0	None	2	400	0.7853981633974483	0.053511638065878914
83	13	624	5830	0	None	3	400	0.7853981633974483	0.053511638065878914
84	13	624	5830	0	None	4	400	0.7853981633974483	0.053511638065878914
85	13	624	5830	0	None	5	400	0.7853981633974483	0.053511638065878914
86	13	624	5830	0	None	6	400	0.7853981633974483	0.053511638065878914
87	13	624	5830	0	None	7	400	0.7853981633974483	0.053511638065878914
88	13	624	5830	0	None	8	400	0.7853981633974483	0.053511638065878914
89	13	624	5830	0	None	9	400	0.7853981633974483	0.053511638065878914
90	13	624	5830	0	None	10	400	0.7853981633974483	0.053511638065878914
91	13	624	5830	0	None	11	400	0.7853981633974483	0.053511638065878914
92	13	624	5830	0	None	12	400	0.7853981633974483	0.053511638065878914
93	13	624	5830	0	None	13	400	0.7853981633974483	0.053511638065878914
94	13	624	5830	0	None	14	400	0.7853981633974483	0.053511638065878914
95	13	624	5830	0	None	15	400	0.7853981633974483	0.053511638065878914
96	13	624	5830	0	None	16	400	0.7853981633974483	0.053511638065878914
97	13	624	5830	0	None	17	400	0.7853981633974483	0.053511638065878914
98	13	624	5829	0	None	18	400	0.7853981633974483	0.053511638065878914
99	13	624	5830	0	None	19	400	0.7853981633974483	0.053511638065878914
100	13	624	5830	0	None	20	400	0.7853981633974483	0.053511638065878914
101	13	624	7288	0	None	1	500	0.7853981633974483	0.04633353350215043
102	13	624	7288	0	None	2	500	0.7853981633974483	0.04633353350215043
103	13	624	7288	0	None	3	500	0.7853981633974483	0.04633353350215043
104	13	624	7288	0	None	4	500	0.7853981633974483	0.04633353350215043
105	13	624	7288	0	None	5	500	0.7853981633974483	0.04633353350215043
106	13	624	7288	0	None	6	500	0.7853981633974483	0.04633353350215043
107	13	624	7288	0	None	7	500	0.7853981633974483	0.04633353350215043
108	13	624	7288	0	None	8	500	0.7853981633974483	0.04633353350215043
109	13	624	7288	0	None	9	500	0.7853981633974483	0.04633353350215043
110	13	624	7288	0	None	10	500	0.7853981633974483	0.04633353350215043
111	13	624	7288	0	None	11	500	0.7853981633974483	0.04633353350215043
112	13	624	7288	0	None	12	500	0.7853981633974483	0.04633353350215043
113	13	624	7288	0	None	13	500	0.7853981633974483	0.04633353350215043
114	13	624	7288	0	None	14	500	0.7853981633974483	0.04633353350215043
115	13	624	7288	0	None	15	500	0.7853981633974483	0.04633353350215043
116	13	624	7288	0	None	16	500	0.7853981633974483	0.04633353350215043
117	13	624	7288	0	None	17	500	0.7853981633974483	0.04633353350215043
118	13	624	7288	0	None	18	500	0.7853981633974483	0.04633353350215043
119	13	624	7288	0	None	19	500	0.7853981633974483	0.04633353350215043
120	13	624	7288	0	None	20	500	0.7853981633974483	0.04633353350215043
121	13	624	8745	0	None	1	600	0.7853981633974483	0.04134001619491394
122	13	624	8745	0	None	2	600	0.7853981633974483	0.04134001619491394
123	13	624	8745	0	None	3	600	0.7853981633974483	0.04134001619491394
124	13	624	8745	0	None	4	600	0.7853981633974483	0.04134001619491394
125	13	624	8744	0	None	5	600	0.7853981633974483	0.04134001619491394
126	13	624	8745	0	None	6	600	0.7853981633974483	0.04134001619491394
127	13	624	8745	0	None	7	600	0.7853981633974483	0.04134001619491394
128	13	624	8745	0	None	8	600	0.7853981633974483	0.04134001619491394
129	13	624	8745	0	None	9	600	0.7853981633974483	0.04134001619491394
130	13	624	8745	0	None	10	600	0.7853981633974483	0.04134001619491394
131	13	624	8745	0	None	11	600	0.7853981633974483	0.04134001619491394
132	13	624	8745	0	None	12	600	0.7853981633974483	0.04134001619491394
133	13	624	8745	0	None	13	600	0.7853981633974483	0.04134001619491394
134	13	624	8745	0	None	14	600	0.7853981633974483	0.04134001619491394
135	13	624	8745	0	None	15	600	0.7853981633974483	0.04134001619491394
136	13	624	8745	0	None	16	600	0.7853981633974483	0.04134001619491394
137	13	624	8745	0	None	17	600	0.7853981633974483	0.04134001619491394
138	13	624	8745	0	None	18	600	0.7853981633974483	0.04134001619491394
139	13	624	8744	0	None	19	600	0.7853981633974483	0.04134001619491394
140	13	624	8745	0	None	20	600	0.7853981633974483	0.04134001619491394
141	13	624	10203	0	None	1	700	0.7853981633974483	0.0376281135230359
142	13	624	10203	0	None	2	700	0.7853981633974483	0.0376281135230359
143	13	624	10203	0	None	3	700	0.7853981633974483	0.0376281135230359

144	13	624	10203	0	None	4	700	0.7853981633974483	0.0376281135230359
145	13	624	10203	0	None	5	700	0.7853981633974483	0.0376281135230359
146	13	624	10203	0	None	6	700	0.7853981633974483	0.0376281135230359
147	13	624	10203	0	None	7	700	0.7853981633974483	0.0376281135230359
148	13	624	10203	0	None	8	700	0.7853981633974483	0.0376281135230359
149	13	624	10203	0	None	9	700	0.7853981633974483	0.0376281135230359
150	13	624	10203	0	None	10	700	0.7853981633974483	0.0376281135230359
151	13	624	10203	0	None	11	700	0.7853981633974483	0.0376281135230359
152	13	624	10203	0	None	12	700	0.7853981633974483	0.0376281135230359
153	13	624	10203	0	None	13	700	0.7853981633974483	0.0376281135230359
154	13	624	10203	0	None	14	700	0.7853981633974483	0.0376281135230359
155	13	624	10203	0	None	15	700	0.7853981633974483	0.0376281135230359
156	13	624	10203	0	None	16	700	0.7853981633974483	0.0376281135230359
157	13	624	10203	0	None	17	700	0.7853981633974483	0.0376281135230359
158	13	624	10203	0	None	18	700	0.7853981633974483	0.0376281135230359
159	13	624	10203	0	None	19	700	0.7853981633974483	0.0376281135230359
160	13	624	10203	0	None	20	700	0.7853981633974483	0.0376281135230359
161	13	624	11660	0	None	1	800	0.7853981633974483	0.03473827434180936
162	13	624	11660	0	None	2	800	0.7853981633974483	0.03473827434180936
163	13	624	11660	0	None	3	800	0.7853981633974483	0.03473827434180936
164	13	624	11659	0	None	4	800	0.7853981633974483	0.03473827434180936
165	13	624	11660	0	None	5	800	0.7853981633974483	0.03473827434180936
166	13	624	11660	0	None	6	800	0.7853981633974483	0.03473827434180936
167	13	624	11660	0	None	7	800	0.7853981633974483	0.03473827434180936
168	13	624	11660	0	None	8	800	0.7853981633974483	0.03473827434180936
169	13	624	11660	0	None	9	800	0.7853981633974483	0.03473827434180936
170	13	624	11660	0	None	10	800	0.7853981633974483	0.03473827434180936
171	13	624	11660	0	None	11	800	0.7853981633974483	0.03473827434180936
172	13	624	11660	0	None	12	800	0.7853981633974483	0.03473827434180936
173	13	624	11660	0	None	13	800	0.7853981633974483	0.03473827434180936
174	13	624	11660	0	None	14	800	0.7853981633974483	0.03473827434180936
175	13	624	11660	0	None	15	800	0.7853981633974483	0.03473827434180936
176	13	624	11659	0	None	16	800	0.7853981633974483	0.03473827434180936
177	13	624	11660	0	None	17	800	0.7853981633974483	0.03473827434180936
178	13	624	11660	0	None	18	800	0.7853981633974483	0.03473827434180936
179	13	624	11660	0	None	19	800	0.7853981633974483	0.03473827434180936
180	13	624	11660	0	None	20	800	0.7853981633974483	0.03473827434180936
181	13	624	13118	0	None	1	900	0.7853981633974483	0.03241052921158404
182	13	624	13118	0	None	2	900	0.7853981633974483	0.03241052921158404
183	13	624	13117	0	None	3	900	0.7853981633974483	0.03241052921158404
184	13	624	13118	0	None	4	900	0.7853981633974483	0.03241052921158404
185	13	624	13118	0	None	5	900	0.7853981633974483	0.03241052921158404
186	13	624	13118	0	None	6	900	0.7853981633974483	0.03241052921158404
187	13	624	13118	0	None	7	900	0.7853981633974483	0.03241052921158404
188	13	624	13118	0	None	8	900	0.7853981633974483	0.03241052921158404
189	13	624	13118	0	None	9	900	0.7853981633974483	0.03241052921158404
190	13	624	13118	0	None	10	900	0.7853981633974483	0.03241052921158404
191	13	624	13118	0	None	11	900	0.7853981633974483	0.03241052921158404
192	13	624	13118	0	None	12	900	0.7853981633974483	0.03241052921158404
193	13	624	13118	0	None	13	900	0.7853981633974483	0.03241052921158404
194	13	624	13118	0	None	14	900	0.7853981633974483	0.03241052921158404
195	13	624	13118	0	None	15	900	0.7853981633974483	0.03241052921158404
196	13	624	13118	0	None	16	900	0.7853981633974483	0.03241052921158404
197	13	624	13118	0	None	17	900	0.7853981633974483	0.03241052921158404
198	13	624	13118	0	None	18	900	0.7853981633974483	0.03241052921158404
199	13	624	13118	0	None	19	900	0.7853981633974483	0.03241052921158404
200	13	624	13118	0	None	20	900	0.7853981633974483	0.03241052921158404
201	13	624	16033	0	None	1	1100	0.7853981633974483	0.02886185362765404
202	13	624	16033	0	None	2	1100	0.7853981633974483	0.02886185362765404
203	13	624	16033	0	None	3	1100	0.7853981633974483	0.02886185362765404
204	13	624	16033	0	None	4	1100	0.7853981633974483	0.02886185362765404
205	13	624	16033	0	None	5	1100	0.7853981633974483	0.02886185362765404
206	13	624	16032	0	None	6	1100	0.7853981633974483	0.02886185362765404
207	13	624	16033	0	None	7	1100	0.7853981633974483	0.02886185362765404
208	13	624	16033	0	None	8	1100	0.7853981633974483	0.02886185362765404
209	13	624	16032	0	None	9	1100	0.7853981633974483	0.02886185362765404
210	13	624	16032	0	None	10	1100	0.7853981633974483	0.02886185362765404
211	13	624	16031	0	None	11	1100	0.7853981633974483	0.02886185362765404
212	13	624	16033	0	None	12	1100	0.7853981633974483	0.02886185362765404
213	13	624	16033	0	None	13	1100	0.7853981633974483	0.02886185362765404
214	13	624	16032	0	None	14	1100	0.7853981633974483	0.02886185362765404
215	13	624	16033	0	None	15	1100	0.7853981633974483	0.02886185362765404
216	13	624	16033	0	None	16	1100	0.7853981633974483	0.02886185362765404
217	13	624	16033	0	None	17	1100	0.7853981633974483	0.02886185362765404
218	13	624	16033	0	None	18	1100	0.7853981633974483	0.02886185362765404
219	13	624	16033	0	None	19	1100	0.7853981633974483	0.02886185362765404
220	13	624	16032	0	None	20	1100	0.7853981633974483	0.02886185362765404
221	13	624	18977	0	None	1	1302	0.7853981633974483	0.026232993053220383
222	13	624	18977	0	None	2	1302	0.7853981633974483	0.026232993053220383
223	13	624	18977	0	None	3	1302	0.7853981633974483	0.026232993053220383
224	13	624	18977	0	None	4	1302	0.7853981633974483	0.026232993053220383
225	13	624	18976	0	None	5	1302	0.7853981633974483	0.026232993053220383
226	13	624	18977	0	None	6	1302	0.7853981633974483	0.026232993053220383
227	13	624	18977	0	None	7	1302	0.7853981633974483	0.026232993053220383
228	13	624	18977	0	None	8	1302	0.7853981633974483	0.026232993053220383
229	13	624	18976	0	None	9	1302	0.7853981633974483	0.026232993053220383
230	13	624	18977	0	None	10	1302	0.7853981633974483	0.026232993053220383
231	13	624	18977	0	None	11	1302	0.7853981633974483	0.026232993053220383
232	13	624	18977	0	None	12	1302	0.7853981633974483	0.026232993053220383
233	13	624	18977	0	None	13	1302	0.7853981633974483	0.026232993053220383
234	13	624	18977	0	None	14	1302	0.7853981633974483	0.026232993053220383
235	13	624	18977	0	None	15	1302	0.7853981633974483	0.026232993053220383
236	13	624	18977	0	None	16	1302	0.7853981633974483	0.026232993053220383
237	13	624	18977	0	None	17	1302	0.7853981633974483	0.026232993053220383
238	13	624	18977	0	None	18	1302	0.7853981633974483	0.026232993053220383
239	13	624	18977	0	None	19	1302	0.7853981633974483	0.026232993053220383
240	13	624	18976	0	None	20	1302	0.7853981633974483	0.026232993053220383

References

[1] Brammer, G. B., van Dokkum, P. G., Franx, M., et al. 2012, ApJS, 200, 19

[2] Rowe, B.T.P., Jarvis, M., Mandelbaum, R., et al. 2015, Astron. Comput., 10, 121

[3] Skelton, R. E., Whitaker, K. E., Momcheva, I. G., et al. 2014, ApJS, 214, 49

[4] Unterborn, C.T., & Ryden, B. 2008, ApJ, 687, 976

[5] Windhorst, R. A., Cohen, S. H., Hathi, N. P., et al. 2011, ApJS, 193, 33

[6] Bertin, E. & Arnouts, S. 1996, A&AS, 317, 393

[7] Robotham, A. S. G., Davies, L. G. M., Driver, S. P., et al. 2018, MNRAS, 476, 3137

[8] Odewahn, S. C., Burnstein, D., & Windhorst, R. A. 1997, AJ, 114, 2219

[9] This research made use of Photutils, an Astropy package for detection and photometry of astronomical sources (Bradley et al. 2020).

[10] Plot provided by Rosalia O'brien.