Create weather animation with GEOS-R data

In [2]:
```
# Importing necessary libraries for data manipulation, visualization, and animation

# numpy is used for handling numerical operations on arrays. It provides support fo
# multi-dimensional arrays and matrices, along with a large collection of high-leve
# functions to operate on these arrays. Essential for processing numerical data eff
import numpy as np

# matplotlib.pyplot is a plotting library used for creating static, interactive, an
# It provides an object-oriented API for embedding plots into applications.
import matplotlib.pyplot as plt

# matplotlib.animation is used to create animations using matplotlib. This module p
# animated plots and updating them over time, which is crucial for creating dynamic
import matplotlib.animation as animation

# IPython.display.HTML is used to display HTML documents in the Jupyter notebook. I
# inline within the Jupyter notebook, making it easily accessible and interactive.
from IPython.display import HTML

# glob is used for file handling. It provides a function for making file lists from
# which simplifies the process of finding and managing paths to multiple data files
from glob import glob

# xarray is specifically designed to work with labeled data and multidimensional ar
# netCDF files which are commonly used for storing multi-dimensional meteorological
# accessing and manipulating satellite data like that from GEOS-R.
import xarray as xr
```

In [6]:
```
# Import necessary libraries for file and path operations
import os
from glob import glob

# Set up the directory path dynamically. This method allows the script to be run on
# regardless of the operating system, by adapting to the user's current working dir

# Obtain the current working directory where this script or notebook is being execu
current_directory = os.getcwd()

# Append the specific subdirectory 'ABI-L1b-RadM1' to the current working directory
# This is the directory where the GEOS-R satellite data files, specifically ABI L1b
full_directory_path = os.path.join(current_directory, 'ABI-L1b-RadM1')

# Use the glob function to create a list of file paths that match a specific patter
# The pattern '**/OR_ABI-L1b-RadM1-M6C03*.nc' is designed to match any files that s
# and end with '.nc', indicating NetCDF files of a particular channel (Channel 3 in
# The '**' allows for matching this pattern at any level of depth within the specif
# subdirectories in the search.
glst = glob(f'{full_directory_path}/**/OR_ABI-L1b-RadM1-M6C03*.nc', recursive=True)

# Display the list of file paths to verify that the correct files are being found.
```

```
# This step is crucial for debugging and ensuring that the data setup is correct be
glst
```

Out[6]:
```
['c:\\Users\\moham\\OneDrive\\Desktop\\Stevens\\Projects\\FACT\\GitHub\\Module_4
\\ABI-L1b-RadM1\\s20223061734250\\OR_ABI-L1b-RadM1-M6C03_G16_s20223061734250_e2022
3061734308_c20223061734345.nc',
 'c:\\Users\\moham\\OneDrive\\Desktop\\Stevens\\Projects\\FACT\\GitHub\\Module_4
\\ABI-L1b-RadM1\\s20223061735250\\OR_ABI-L1b-RadM1-M6C03_G16_s20223061735250_e2022
3061735308_c20223061735348.nc',
 'c:\\Users\\moham\\OneDrive\\Desktop\\Stevens\\Projects\\FACT\\GitHub\\Module_4
\\ABI-L1b-RadM1\\s20223061737250\\OR_ABI-L1b-RadM1-M6C03_G16_s20223061737250_e2022
3061737308_c20223061737344.nc',
 'c:\\Users\\moham\\OneDrive\\Desktop\\Stevens\\Projects\\FACT\\GitHub\\Module_4
\\ABI-L1b-RadM1\\s20223061738250\\OR_ABI-L1b-RadM1-M6C03_G16_s20223061738250_e2022
3061738308_c20223061738344.nc',
 'c:\\Users\\moham\\OneDrive\\Desktop\\Stevens\\Projects\\FACT\\GitHub\\Module_4
\\ABI-L1b-RadM1\\s20223061739250\\OR_ABI-L1b-RadM1-M6C03_G16_s20223061739250_e2022
3061739308_c20223061739344.nc']
```

In [7]:
```
# Import necessary libraries for handling arrays, plotting, and animations.
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from IPython.display import HTML
import xarray as xr

# Create a new figure for plotting. This figure will serve as the canvas for our an
fig = plt.figure()

# Initialize a list to store frames for the animation. Each frame will consist of a
ims = []

# Loop through each file path in the list 'glst', which contains paths to NetCDF fi
for fl in glst:
    # Open the dataset from the current file using xarray. Xarray enables easy hand
    # scientific data and seamlessly integrates with plotting libraries for visuali
    ncx = xr.open_dataset(fl)

    # Create an image from the 'Rad' (Radiance) data in the dataset. This involves:
    # - Accessing the 'Rad' data array from the dataset.
    # - Plotting this data array using matplotlib's imshow function, which is ideal
    # - Setting the colormap to 'Greys_r' to display data in grayscale, which helps
    # - Marking the image as 'animated' so it can be managed by matplotlib's animat
    im = plt.imshow(ncx['Rad'].data, origin='upper', cmap="Greys_r", animated=True)

    # Append the created image plot to the list of frames. Each frame is a list its
    ims.append([im])

# Close the plt.show() window to prevent display of a static plot, ensuring that on
plt.close()

# Create an animation object from the list of image frames. This uses matplotlib's
# which compiles the frames into a sequence. Parameters include:
# - 'fig', the figure object to animate.
# - 'ims', the list of frames to include in the animation.
# - 'interval=50', setting the display duration of each frame to 50 milliseconds.
# - 'blit=True', enabling blitting, which improves animation performance by only re
```

```
# - 'repeat_delay=1000', setting a 1000 millisecond delay before the animation repe
ani = animation.ArtistAnimation(fig, ims, interval=50, blit=True, repeat_delay=1000

# Convert the animation to a JavaScript HTML representation using IPython.display.H
# the animation to be embedded and interactively played within a Jupyter Notebook.
# visualize dynamic processes like weather patterns in satellite data.
HTML(ani.to_jshtml())
```

Out[7]: