Meghana Sannapareddy (9TE25022K)

Final Report

30-5-2025

Estimating the attitude of artificial satellites from ground-based optical measurements

Problem Statement-

The aim of this project is to simulate how the ACS3 solar sail satellite would appear when observed through a 406.4 mm aperture, f/10 telescope from Earth under realistic conditions. To replicate these observational effects, this project employs Blender's rendering engine to generate synthetic yet realistic images of the ACS3 satellite. The simulation process involves modeling the satellite's position and orientation using observational parameters such as azimuth, elevation, range, and phase angle.

Methodology -

We begin by defining key telescope parameters, such as aperture, focal length, pixel size, and observing wavelength, along with observational parameters including the satellite's range, azimuth, elevation, phase angle, and velocity. Using these parameters, the satellite's position is computed in 3D space relative to the observer by converting azimuth and elevation angles into Cartesian coordinates, which are then scaled appropriately to fit the Blender scene. The ACS3 satellite model is retrieved, scaled based on the quality factor derived from optical diffraction, and oriented according to the given phase angle.

A virtual telescope camera is configured in Blender by setting its lens focal length, sensor size, and clipping parameters. Solar illumination is simulated by adding a directional "Sun" light source aligned with the satellite's phase angle. The satellite's materials are carefully defined to replicate realistic surface properties: the sail uses a reflective metallic shader with low roughness, while the booms use a dark

carbon material with higher roughness. The world background is set to a black sky to represent space. To simulate the real imaging effects, we incorporate a series of optical and atmospheric effects. Finally, all these effects are composited together and rendered using Blender's Cycles engine with physically accurate lighting and exposure control based on the telescope's gain setting.

Optical and Atmospheric effects -

- **Atmospheric Refraction:** The apparent elevation is

  Range R' = $\frac{1.02}{tan\,(\,E + \frac{10.3}{E+5.11})}$ and Elevation E' = E+$\frac{R'}{60}$

  The 3D positions are x=R · cosE · sinA, y=R · cosE · cosA, z=R · sinE

- **Diffraction Blur:** The telescope's aperture introduces diffraction, limiting its angular resolution and causing the appearance of an Airy disk. This effect is approximated using a Gaussian blur.

  $\theta = 1.22\,\frac{\lambda}{D}$

  Blur on the sensor = θ.f, where f is the focal length

- **Atmospheric Seeing:** The Earth's atmosphere introduces random phase distortions (astronomical seeing), broadening the point spread function. We used a seeing blur of 8 px added to the total blur.
- **Motion Blur:** The satellite moves rapidly across the sky. If the telescope tracking is imperfect or the exposure time is finite, this causes motion blur. We calculate the blur due to motion as -

  $\omega = \frac{v}{R}$ Blur = ω.t.$\frac{f}{p}$, where t is the exposure time and p is the pixel scale

- **Chromatic Aberration:** Optical systems can cause different wavelengths to focus at slightly different points. We simulate this by separating RGB channels and shifting the red channel by a small offset. This results in subtle color fringing around bright edges.

  $\Delta x_R = \Delta y_R = 1$ pixel

- **Glare and Bloom:** Highly reflective surfaces, such as the solar sail, scatter light through the telescope optics, creating halos or “bloom.” This is implemented using Blender’s Glare node with the "Fog Glow" setting, simulating the scattering of light around bright regions.
- **Sensor Noise:** Real sensors introduce thermal, shot, and read noise. This is modeled by overlaying a procedural noise texture.

  I’= I+αN, where N is the noise pattern, and α is the noise intensity factor (~0.05).

Observational Parameters-

- **Range:** The distance between the telescope and the satellite, affecting the satellite’s apparent size in the image. Scaled using a scene_scale factor to match Blender's coordinate system.
- **Right Ascension** and **Declination:** Define the satellite’s absolute celestial position. Although Blender uses local azimuth and elevation for positioning, RA/DEC values help in validation.
- **Azimuth:** This angle defines the compass direction from the observer to the satellite. It determines the satellite’s horizontal position in the simulated sky.
- **Elevation:** This angle defines how high above the horizon the satellite appears. It affects the vertical placement of the satellite in the 3D scene.
- **Phase Angle:** The angle between the observer, satellite, and the Sun. It controls how the satellite is illuminated and affects the sunlamp's rotation.

- **Gain:** Gain refers to the sensitivity of the telescope's imaging system. It was used to compute an exposure compensation in Blender using logarithmic scaling: " exposure = log(gain / 1000)". A higher gain simulates a brighter image.
- **Exposure Time:** Although Blender does not simulate time-dependent exposure like a real camera, this value was recorded as a reference to the real-world imaging conditions.

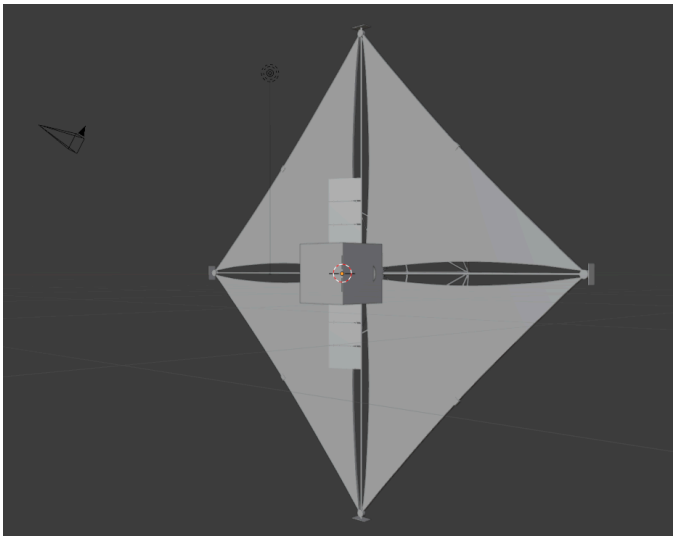
Telescope and Camera-

To realistically simulate telescope-based imaging, the virtual camera was configured with:

- ➔ **Aperture:** 406.4 mm (standard 16-inch telescope).
- ➔ **Focal Length:** 4064 mm (f/10), computed using focal = aperture × f-number.
- ➔ **Sensor Width:** 36 mm (full-frame camera equivalent).
- ➔ **Clipping Range:** 0.1 to 1e6 units to avoid near/far object cutoff.
- ➔ **Render Engine:** Cycles (used for physically accurate light and material rendering).
- ➔ **Resolution:** 1920×1080 (HD image suitable for visual analysis).
- ➔ **Samples:** 128 (balanced for quality and performance).
- ➔ **Gain Adjustment**: Implemented via Blender's exposure setting to replicate detector sensitivity.

ACS3 Model-

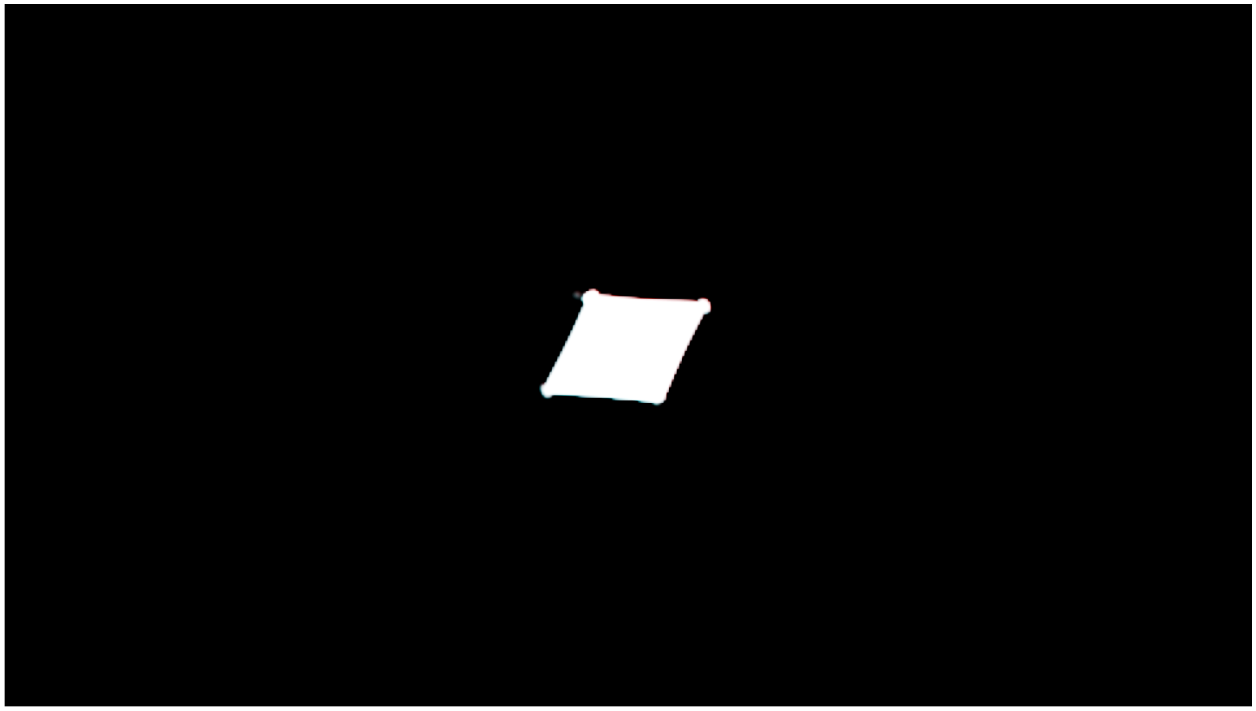
The model used for the simulation is NASA's publicly available ACS3 solar sail model, which is imported into Blender in the .lwo (LightWave Object) format.



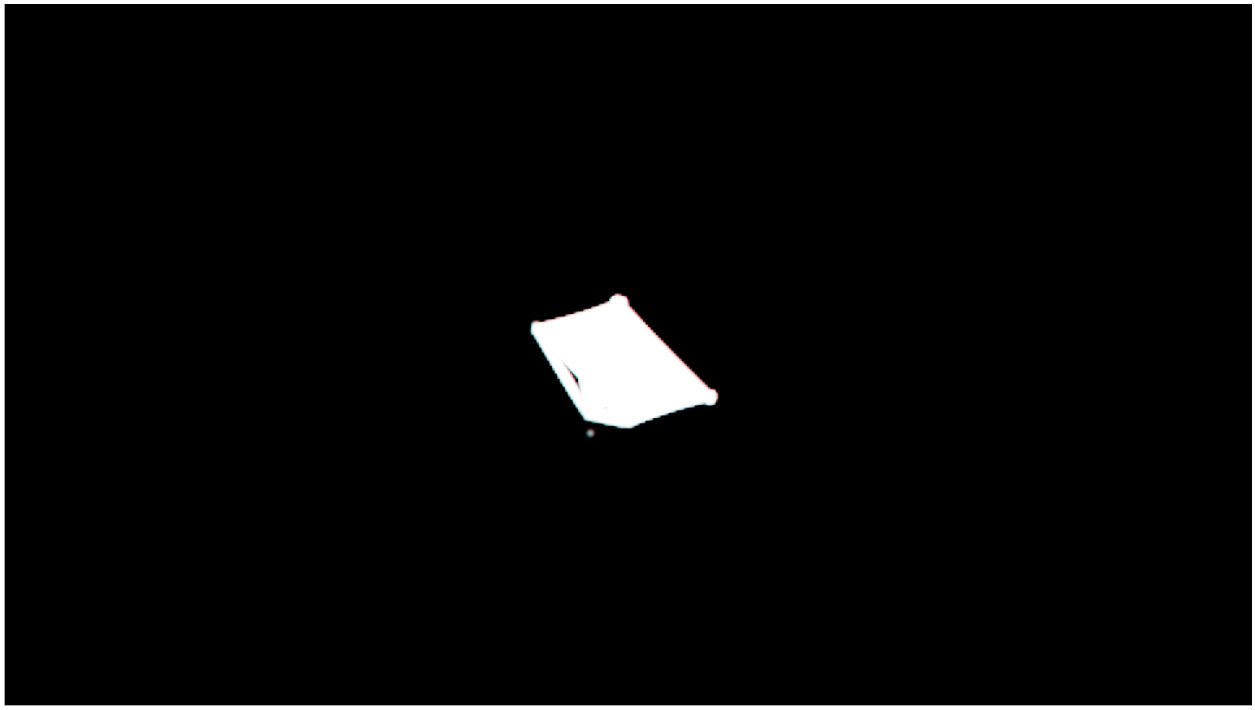
Results-

I was provided with four observation timestamps of ACS3 taken at an observatory. Below are the corresponding simulation results.

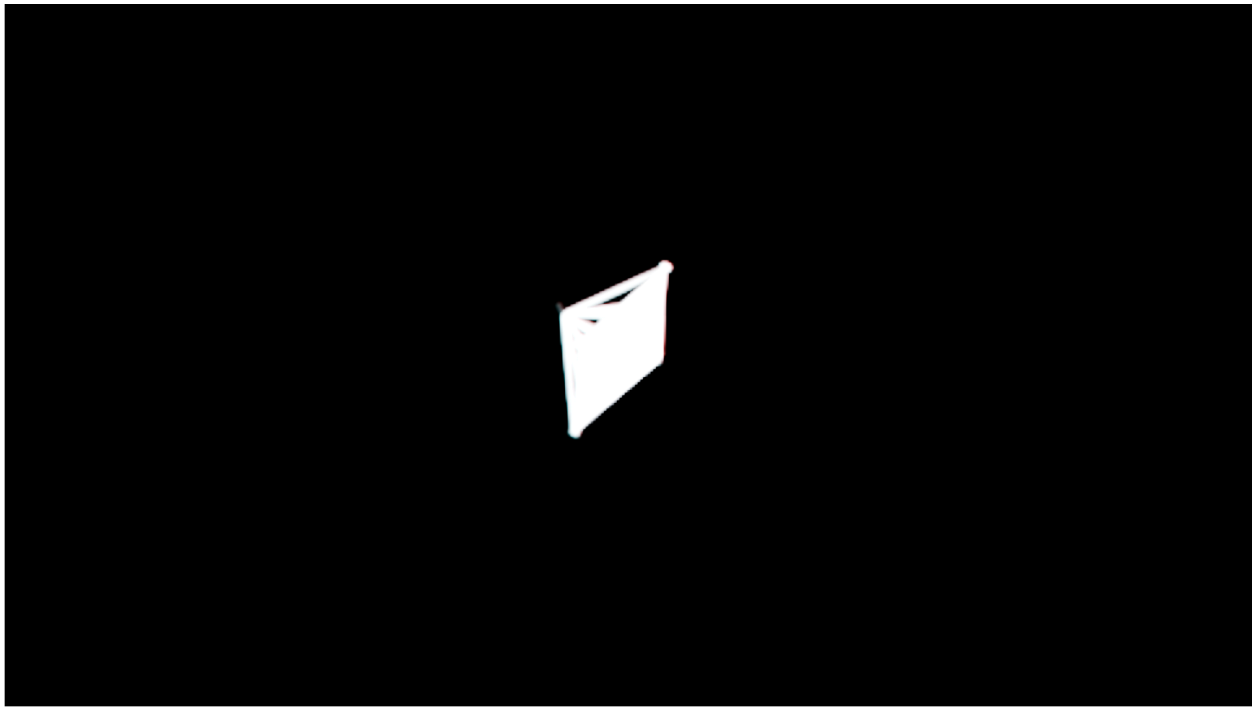
- 2024-12-08T10:01:35 Range-1339 km



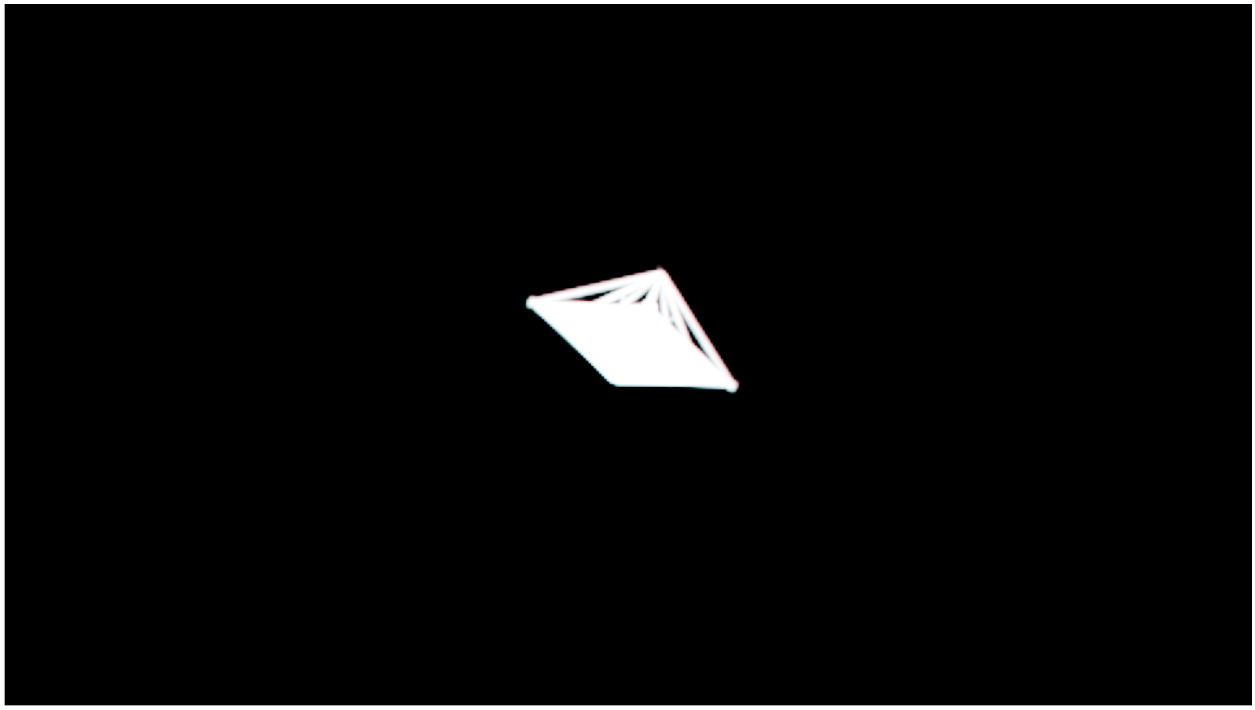
- 2024-12-10T10:58:00 Range-1217 km



- 2024-12-23T10:01:42 Range-1262 km



- 2025-02-11T09:45:51 Range-1101 km



Future Scope -

This simulation framework can be easily adapted for other satellites or spacecraft models. By replacing the ACS3 solar sail model with any other satellite 3D model (e.g., in `.obj`, `.fbx`, or `.lwo` formats) and adjusting the observational parameters (aperture, focal length, range, phase angle, etc.), realistic telescope observations can be replicated for different missions.

Code for Reference -

The following is placed in the text editor of Blender, and the model is saved as an "acs3" collection.

```
import bpy
import math
from mathutils import Vector, Euler

# TELESCOPE + OBSERVATION PARAMETERS
aperture_mm = 406.4
f_number = 10
focal_length_mm = aperture_mm * f_number
wavelength_nm = 550
pixel_size_um = 4.63
range_km = 1262
phase_angle_deg = 40.8598
azimuth_deg = 39.2924
elevation_deg = 51.0299
gain = 2680
satellite_velocity = 7350
feature_size_m = 1.5
output_path = "Enter the desired file location.png"
```

```
# CONVERSIONS
range_m = range_km * 1000
focal_length_m = focal_length_mm / 1000
aperture_m = aperture_mm / 1000
wavelength_m = wavelength_nm * 1e-9
pixel_size_m = pixel_size_um * 1e-6

# ATMOSPHERIC REFRACTION CORRECTION
R_arcmin = 1.02 / math.tan(math.radians(elevation_deg) + 10.3 / (math.radians(elevation_deg) +
5.11))
R_deg = R_arcmin / 60
corrected_elevation_deg = elevation_deg + R_deg

# POSITION
az_rad = math.radians(azimuth_deg)
el_rad = math.radians(corrected_elevation_deg)
scene_scale = 0.0001
x = range_m * scene_scale * math.cos(el_rad) * math.sin(az_rad)
y = range_m * scene_scale * math.cos(el_rad) * math.cos(az_rad)
z = range_m * scene_scale * math.sin(el_rad)
acs3_position = Vector((x, y, z))

# FIND SATELLITE
```

```
acs3_collection = bpy.data.collections.get("acs3")  #You can change the name of your satellite
here
if not acs3_collection:
    raise ValueError("Collection 'acs3' not found in the scene.")

# SCALE BASED ON QUALITY FACTOR
blur_spot = (wavelength_m * focal_length_m) / aperture_m
quality_factor = blur_spot / feature_size_m
satellite_scale = max(quality_factor, 0.01)

for obj in acs3_collection.objects:
    obj.scale = (satellite_scale, satellite_scale, satellite_scale)
    obj.location = acs3_position
    obj.rotation_euler = Euler((math.radians(-phase_angle_deg), 0, 0), 'XYZ')
    obj.hide_render = False

# CAMERA SETUP
cam_data = bpy.data.cameras.new("TelescopeCamera")
cam_data.lens = focal_length_mm
cam_data.sensor_width = 36
cam_data.clip_start = 0.1
cam_data.clip_end = 1e6
```

```
camera = bpy.data.objects.new("TelescopeCamera", cam_data)
bpy.context.scene.collection.objects.link(camera)
bpy.context.scene.camera = camera
camera.location = Vector((0, 0, 0))

# ENABLE DEPTH OF FIELD
#camera.data.dof.use_dof = True
#camera.data.dof.focus_distance = range_m
#camera.data.dof.aperture_fstop = f_number

direction = (acs3_position - camera.location).normalized()
camera.rotation_euler = direction.to_track_quat('-Z', 'Y').to_euler()

# SUN LIGHT
sun_data = bpy.data.lights.new(name="Sun", type='SUN')
sun = bpy.data.objects.new(name="Sun", object_data=sun_data)
bpy.context.collection.objects.link(sun)
sun.rotation_euler = Euler((math.radians(90 - phase_angle_deg), 0, 0), 'XYZ')
sun.location = acs3_position + Vector((0, 0, 1000))

# MATERIALS
sail_mat = bpy.data.materials.new(name="SilverSail")
sail_mat.use_nodes = True
```

```
nodes = sail_mat.node_tree.nodes
principled = nodes.get("Principled BSDF")
principled.inputs["Base Color"].default_value = (0.8, 0.8, 0.9, 1)
principled.inputs["Roughness"].default_value = 0.5
principled.inputs["Metallic"].default_value = 0.6

carbon_mat = bpy.data.materials.new(name="CarbonBoom")
carbon_mat.use_nodes = True
nodes2 = carbon_mat.node_tree.nodes
principled2 = nodes2.get("Principled BSDF")
principled2.inputs["Base Color"].default_value = (0.05, 0.05, 0.05, 1)
principled2.inputs["Roughness"].default_value = 0.8
principled2.inputs["Metallic"].default_value = 0.0

#  Assign materials based on object name
for obj in acs3_collection.objects:
    if obj.type == 'MESH':
        obj.data.materials.clear()
        if "boom" in obj.name.lower():
            obj.data.materials.append(carbon_mat)
        else:
            obj.data.materials.append(sail_mat)
```

```
# WORLD
world = bpy.context.scene.world
world.use_nodes = True
bg = world.node_tree.nodes
bg["Background"].inputs[0].default_value = (0, 0, 0, 1)
bg["Background"].inputs[1].default_value = 0.0
bpy.context.scene.render.film_transparent = False

# RENDER SETTINGS
scene = bpy.context.scene
scene.render.engine = 'CYCLES'
scene.cycles.samples = 128
scene.render.resolution_x = 1920
scene.render.resolution_y = 1080
scene.render.image_settings.file_format = 'PNG'
scene.render.filepath = output_path
scene.view_settings.exposure = math.log(gain / 4000)

# COMPOSITOR: DIFFRACTION + ATMOSPHERIC + MOTION BLUR
scene.use_nodes = True
tree = scene.node_tree
tree.nodes.clear()
```

```
# Compositing Nodes
rlayers = tree.nodes.new("CompositorNodeRLayers")
blur = tree.nodes.new("CompositorNodeBlur")
blur.filter_type = 'GAUSS'
blur.use_relative = False

# Blur computation
theta_rad = 1.22 * wavelength_m / aperture_m
exposure_time = 1 / 1000
angular_speed = satellite_velocity / range_m
motion_blur_angle = angular_speed * exposure_time
pixel_blur = (motion_blur_angle * theta_rad * focal_length_m) / (cam_data.sensor_width /
scene.render.resolution_x)
seeing_blur_px = 8
total_blur = pixel_blur + seeing_blur_px
blur.size_x = blur.size_y = int(total_blur)

# Chromatic Aberration
sep_rgb = tree.nodes.new("CompositorNodeSepRGBA")
trans_r = tree.nodes.new("CompositorNodeTranslate")
trans_r.inputs[1].default_value = 1.0  # X shift
trans_r.inputs[2].default_value = 1.0  # Y shift
```

```
comb_rgb = tree.nodes.new("CompositorNodeCombRGBA")
tree.links.new(rlayers.outputs["Image"], sep_rgb.inputs["Image"])
tree.links.new(sep_rgb.outputs["R"], trans_r.inputs["Image"])
tree.links.new(trans_r.outputs["Image"], comb_rgb.inputs["R"])
tree.links.new(sep_rgb.outputs["G"], comb_rgb.inputs["G"])
tree.links.new(sep_rgb.outputs["B"], comb_rgb.inputs["B"])

# Glare (Bloom)
glare = tree.nodes.new("CompositorNodeGlare")
glare.glare_type = 'FOG_GLOW'
glare.quality = 'LOW'
glare.size = 2

# Sensor Noise
# Step 1: Create a noise texture in data
noise_texture = bpy.data.textures.new("NoiseTex", type='CLOUDS')
noise_texture.noise_scale = 0.2  # Control scale of noise

# Step 2: Use CompositorNodeTexture to bring it in
noise_node = tree.nodes.new("CompositorNodeTexture")
noise_node.texture = noise_texture

# Step 3: Mix with image using overlay
```

```
mix_noise = tree.nodes.new("CompositorNodeMixRGB")
mix_noise.blend_type = 'OVERLAY'
mix_noise.inputs[0].default_value = 0.05

mix_noise = tree.nodes.new("CompositorNodeMixRGB")
mix_noise.blend_type = 'OVERLAY'
mix_noise.inputs[0].default_value = 0.05

# Final Composite Output
comp = tree.nodes.new("CompositorNodeComposite")

# Connections
tree.links.new(comb_rgb.outputs["Image"], blur.inputs["Image"])
tree.links.new(blur.outputs["Image"], glare.inputs["Image"])
tree.links.new(glare.outputs["Image"], mix_noise.inputs[1])
tree.links.new(noise_node.outputs["Color"], mix_noise.inputs[2])
tree.links.new(mix_noise.outputs["Image"], comp.inputs["Image"])

# FINAL RENDER
bpy.ops.render.render(write_still=True)

# PRINT RESULTS
print("Render complete.")
```

```
print(f"Corrected elevation: {corrected_elevation_deg:.2f}°")

print(f"Satellite scale: {satellite_scale:.4f}")

print(f"Diffraction blur (Airy disk): {pixel_blur:.2f} px")
```