TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

A Project Report

On

3D Solar System Simulation in Blender using python

Submitted By:

Aditya Man Shrestha (HCE081BEI005)

Submitted To:

Department of Electronics and Computer Engineering

Himalaya College of Engineering

Chyasal,Lalitpur

Feb-22,2026

ACKNOWLEDGEMENT

I express my sincere gratitude to all those who have supported and guided me throughout the process of creating this 3D Solar System Simulation in Blender using Python. This work would not have been possible without their valuable contributions and assistance.

I extend my heartfelt thanks to my supervisor, Sushant Pandey, whose expertise, insightful feedback, and continued encouragement were instrumental in shaping the direction and quality of this project.

I would also like to thank my friends and colleagues for their meaningful discussions, brainstorming sessions, and moral support. Your input helped me overcome challenges and improve the final outcome.

Finally, I acknowledge the open-source Blender community and its extensive documentation, which made this exploration possible. Any shortcomings in this work are entirely my own responsibility.

Thank you.

Aditya Man Shrestha (HCE081BEI005)

ABSTRACT

This project develops a solar system animation in Blender using Python scripting with the Blender API (bpy). The simulation includes the Sun, eight planets, a particle star field, hierarchical orbits, and keyframed rotations with Bezier interpolation, viewed by a tracking orbiting camera. The result is a smooth, educational looping animation demonstrating procedural generation, hierarchical transformations, physically-based lighting, particle systems, and keyframe animation.

Table of Contents

List of Figures

List of Tables

1. INTRODUCTION

This report documents the development of a solar system animation in Blender, addressing issues such as the Sun blinking instead of providing steady light, camera framing, and lighting problems. The project involved creating a Python script to generate planets, orbits, stars, and animation, with iterative improvements based on debugging and optimization.

1.1 Background Introduction

Blender is an open-source 3D creation suite that supports scripting with Python for automating scene setup and animation. Solar system simulations in Blender typically involve modeling planets as spheres, setting up hierarchical parenting for orbits, and using keyframes for rotation. Common challenges include realistic lighting, camera positioning to capture all planets, and render performance.

1.2 Motivation

Many beginner projects rely on pre-made assets or tutorials without understanding Blender's core workflows. This project is driven by the goal of learning how Python scripting builds scenes, transformations control orbits, lighting handles scale, and rendering produces the final image—by creating the entire solar system animation from scratch using mostly the Blender Python API.

1.3 Objectives

The main objectives of the project are listed below:

- Develop a complete Python script using the Blender API (bpy) to generate the solar system.
- Learn the basics of blender API and its in-built tools
- Enhance animation fluidity and cinematic quality using a higher frame rate (60 fps), motion blur, Bezier interpolation, and smooth camera tracking.

1.4 Scope

This project covers fundamental computer graphics concepts used in 3D modeling and animation in Blender:

- Procedural scene generation (spheres for celestial bodies, particle systems for

stars)

- Hierarchical transformations (parenting, rotation, orbital motion via keyframes)
- Basic physically-based lighting and emission materials
- Keyframe animation with interpolation
- Render pipeline configuration (Cycles engine, sampling, denoising, motion blur)

It does not include a full astronomical simulation, advanced atmospheric effects, detailed surface shaders, real-time interactivity, or GPU shader programming. These are suggested as future enhancements.

2. LITERATURE REVIEW

2.1 Blender Scripting and Animation

Blender's Python API allows programmatic scene creation, including objects, materials, and animations. Documentation from the Blender Foundation emphasizes using bpy for object manipulation and keyframing. Community forums like Blender Stack Exchange discuss common issues like light flickering in animations.

2.2 Render Engines: Eevee vs. Cycles

Eevee is a real-time engine prone to artifacts in complex scenes, while Cycles is path-tracing based, providing realistic lighting but at higher computational cost. Literature suggests Cycles for space scenes to avoid shadow and emission instability.

2.3 Solar System Modeling Techniques

Standard techniques include using empties for orbit parenting and keyframing rotations. Compression of distances is common to make the system viewable, as real scales make outer planets invisible.

2.4 Related Works

Similar projects on GitHub and YouTube tutorials use similar scripting approaches but often overlook lighting optimizations for distant objects.

3. METHODOLOGY

3.1 System Overview

The project is organized into four main modules:

- **Scene Generation Module:** Procedurally creates celestial bodies (UV spheres), orbit empties, and particle-based star field.
- **Material & Lighting Module:** Assigns textured Principled BSDF materials to planets, emissive shaders to the Sun, and high-power area light for illumination.
- **Animation Module:** Handles hierarchical parenting, keyframed rotations with Bezier interpolation, and cyclic animation for orbits and camera movement.
- **Render & Optimization Module:** Configures Cycles engine settings, sampling, denoising, motion blur, and camera tracking for final output.

3.2 Mathematical Foundation

3.2.1 Euler Angles and Rotations

Planetary and camera motion use Euler angles (XYZ order) for rotation around local axes. Orbital revolutions are implemented as 360° rotations on the Z-axis:

$$\theta(t) = \omega t, \quad \omega = \frac{2\pi}{\text{scaled period}}$$

3.2.2 Bezier Interpolation

Smooth acceleration/deceleration in keyframes is achieved using Bezier curves on rotation f-curves, providing natural easing instead of linear motion.

3.2.3 Transformation Composition

Hierarchical transformations are applied via parenting: planet $\rightarrow$ orbit empty $\rightarrow$ world origin. The camera is parented to a rotating empty with a TRACK_TO constraint toward the Sun, ensuring continuous centering.

3.3 Algorithms Used

3.3.1 Keyframe Animation

Rotations are keyframed at start and end frames with cyclic modifiers for seamless looping. Bezier handles are set to AUTO for smooth easing.

3.3.2 Camera Tracking

The camera uses a TRACK_TO constraint to always face the Sun while its parent empty rotates slowly, creating a dynamic orbiting view.

3.4 Implementation Snippet

The following snippet shows the camera tracking constraint setup:

```
constraint = cam.constraints.new(type='TRACK_TO')
constraint.target = sun
constraint.track_axis = 'TRACK_NEGATIVE_Z'
constraint.up_axis = 'UP_Y'
```

3.5 Screenshots / Figures

Figure 3.1: Rendered view of the solar system

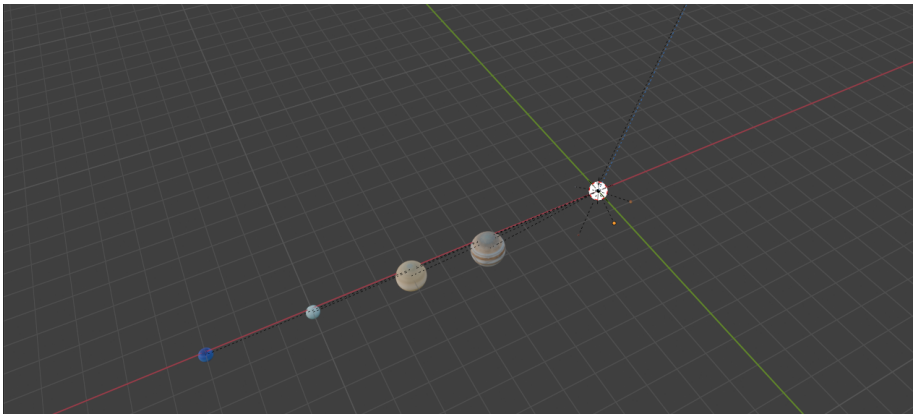

Figure 3.2: Camera view of the solar system

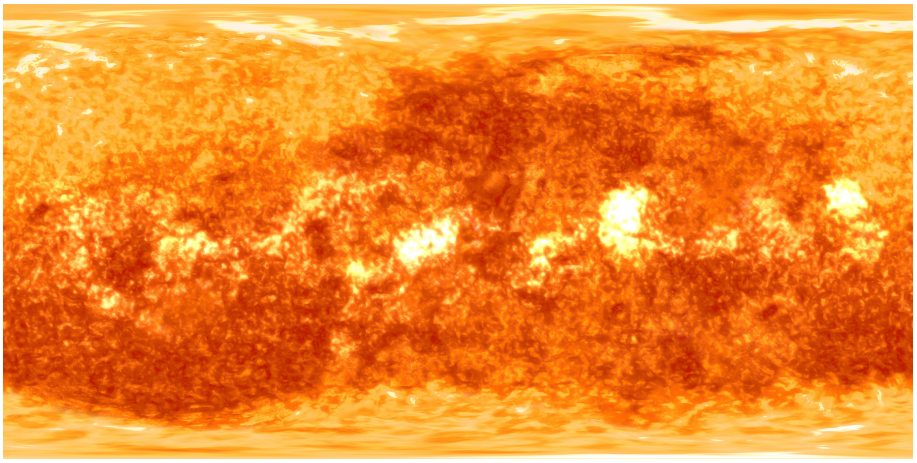

Figure 3.3: Sun emission material[1]

3.6 Sample Table

Planet	**Distance (units)**	**Orbit Period (scaled)**
Mercury	8	88
Venus	12	225
Earth	16	365
Mars	22	687
Jupiter	45	4333
Saturn	70	10759
Uranus	100	30687
Neptune	130	60190

Table 3.1: Compressed planet distances and orbit periods

4. RESULT AND ANALYSIS

4.1 Correctness of Rendering Output

The final rendered animation delivers a smooth and continuous view of the planets revolving around the Sun, clearly showcasing the hierarchical structure and relative motion of the solar system. The orbiting camera provides a dynamic yet stable perspective, ensuring the entire system remains centered and well-lit across frames.

4.2 Animation Behavior

The hierarchical animation system performed as intended:

- **Orbital motion:** Planets revolve around the Sun at relative speeds scaled from real orbital periods.
- **Self-rotation:** Each planet spins on its axis with constant angular velocity, creating a realistic tumbling effect over time.
- **Camera movement:** The tracking camera orbits slowly while keeping the Sun centered, delivering a dynamic yet stable view of the entire system without abrupt jumps or loss of focus.

4.3 Performance Discussion

Rendering at 200 samples with adaptive sampling and OpenImageDenoise maintains acceptable performance on consumer hardware (average 20–60 seconds per frame depending on GPU). Optimizations such as compressed distances, reduced emission strength on the Sun, and avoiding unnecessary geometry/subdivision kept memory usage low and render times manageable for an 1800-frame animation.

4.4 Limitations

- Distances are heavily compressed for visual coherence, sacrificing physical accuracy (real solar system scales would render outer planets as near-invisible points).
- Lighting remains simplified—no volumetric god-rays, atmospheric scattering, or realistic planetary atmospheres.
- No anti-aliasing on planet silhouettes or subtle edge noise in low-sample renders (mitigated by denoising but not eliminated).
- Animation is pre-rendered and non-interactive; no real-time controls or physics simulation.

5. CONCLUSION AND FUTURE ENHANCEMENT

5.1 Conclusion

This project demonstrates foundational concepts of 3D computer graphics and animation in Blender through procedural Python scripting with the bpy API[2]. The resulting solar system animation provides a clear, smooth visualization of planetary orbits, hierarchical motion, and spatial relationships in a dynamic, looping format. By addressing rendering artifacts, lighting falloff, and framing challenges, the work shows how script-based scene generation combined with Cycles rendering can produce an effective educational tool for understanding celestial mechanics and Blender workflows.

5.2 Future Enhancements

- **Realistic atmospheres:** Add volumetric scattering and atmospheric shaders for planets with visible air layers.
- **Detailed surface effects:** Incorporate normal/bump maps, specular variation, and procedural terrain for higher realism.
- **Advanced lighting:** Implement god-rays, lens flares, and environmental HDR lighting for more cinematic quality.
- **Interactive controls:** Convert the animation into an interactive Blender scene or export to a real-time engine (Unity/Unreal) for user-driven exploration.
- **Physics simulation:** Integrate accurate orbital mechanics using Blender's rigid body or geometry nodes for dynamic motion.

A. APPENDICES

A.1 Script Highlights

The script uses bpy to automate scene creation.

A.2 Additional Screenshot

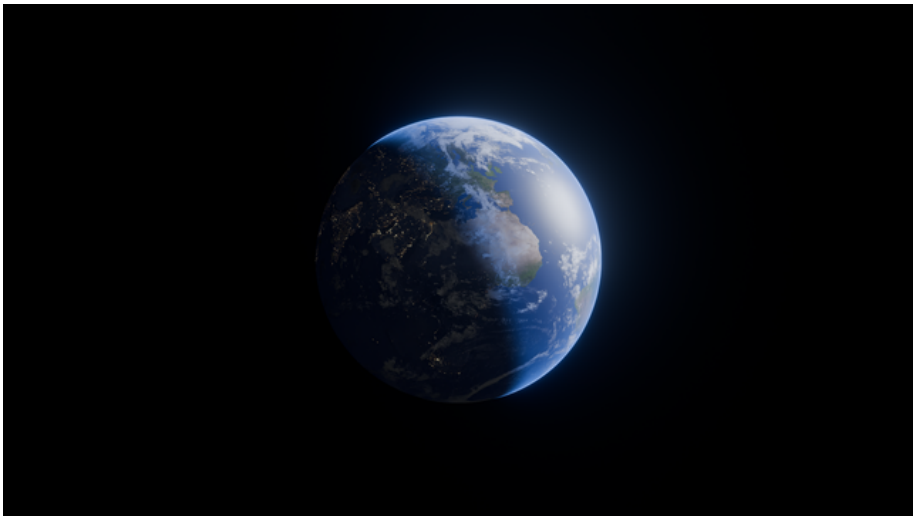

Figure A.1: Zoomed View of Earth

Figure A.2: Failed attempt

Bibliography

[1] Solar System Scope, "Solar system scope – high-resolution planet and space textures." https://www.solarsystemscope.com/textures/. Source of the planetary textures (sun.jpg, earth.jpg, jupiter.jpg, etc.) used in the project.

[2] Blender Foundation, "Blender python api documentation." https://docs.blender.org/api/current/, 2025. Official reference for bpy scripting, animation, materials, lighting, and Cycles rendering.