**Orbit Earth Unity Simulation/DataGen Java Program Guide**

**By Rohith Perla**

**Table of Contents**

[1 Introduction 3](#_Toc7803209)

[2 System Overview 3](#_Toc7803210)

[3 Background 3](#_Toc7803211)

[3.1 Unity 3](#_Toc7803212)

[3.2 Orekit 3](#_Toc7803213)

[3.3 Earth Centered Earth Fixed Coordinate System 3](#_Toc7803214)

[3.4 Machine Learning 4](#_Toc7803215)

[4 Unity Guide 5](#_Toc7803216)

[4.1 SatelliteOrbit.cs 5](#_Toc7803217)

[4.2 SunRotation.cs 5](#_Toc7803218)

[4.3 MoonRotation.cs 6](#_Toc7803219)

[4.4 StagitMainEarth.cs 6](#_Toc7803220)

[4.5 Config Files 6](#_Toc7803221)

[5 Orekit Guide 7](#_Toc7803222)

[5.1 Setup 7](#_Toc7803223)

[5.2 Program Overview 7](#_Toc7803224)

# Introduction

The purpose of this document is to assist users in generating training data for the accompanying neural network in the project, through the use of the Orbit Earth Unity simulation and the DataGen Java program.

# System Overview

The Orbit Earth simulation is a Unity simulation of a camera (representing a satellite) orbiting around the Earth and spinning around while it is orbiting. It also has the functionality of being able to take photos with every frame that the camera is orbiting around the Earth and output them to different directories depending on what continent or ocean that the camera is directly over.

The simulation also contains a directional light source, representing the Sun, as well as a gray sphere object, representing the Moon. They revolve around the Earth, moving between xyz coordinates in a table of coordinates generated by the DataGen Java program that uses Orekit. These coordinates are in the Earth Centered Earth Fixed coordinate system.

# Background

## Unity

Unity is a cross platform game engine used in developing 2D and 3D games as well as simulations. For the purposes of this project, it is used to simulate the orbital environment in space.

## Orekit

Orekit is a free, open source, low-level space dynamics Java library that aims at providing accurate and efficient low level components for the development of flight dynamics applications. It is designed to be easily used in very different contexts, from quick studies up to critical applications. It provides basic elements (orbits, dates, attitude, frames, etc.) and various algorithms to handle them. It is used to generate data tables that are usable in the Orbit Earth Unity simulation.

## Earth Centered Earth Fixed Coordinate System

The Earth Centered Earth Fixed (ECEF) frame is a coordinate system that represents positions as X, Y , and Z coordinates and denotes the point (0, 0, 0) as the center of the Earth. All celestial bodies and other objects in the ECEF frame orbit around the Earth. The Earth does not rotate in this coordinate frame.

## Machine Learning

The method of attitude detection in this project is a neural network that is capable of taking pictures and deducing the continent or ocean that the satellite is directly above. An artificial neural network is an information processing paradigm that functions somewhat like the biological neural networks that make up biological brains. It is a framework for many different machine learning algorithms (neurons in this metaphor) to work together, supply inputs to each other, and process complex data inputs. A neural network is configured for a specific task, and it learns to perform a task by considering examples of it. With each example, the accuracy of the network ideally improves.

An artificial neural network has a layer of neurons that take inputs, a “hidden” layer(s) of neurons that process those inputs, and a layer of neurons that output a solution. The implementation of a convolutional neural network (CNN) is a particular class of neural network commonly applied to the analysis of visual imagery. From an high-level overview, a CNN usually consists of convolutional, pooling, dense, and normalization layers. Through these layers, an image (with relatively little pre-processing) can be analyzed segment by segment while the network learns of different features, or useable data in the image. Through this process, the network gains the ability to perform complex classification of images.

The network in this project takes pictures generated by the Unity simulation and “trains” on them to form generalizations of what to find in pictures to determine what continent or ocean the satellite is above, and then puts those to use when the network is “tested”, in order to output accurate answers.

# Unity Guide

As mentioned before, the Unity environment has 4 main objects in it. There is a high quality model of the Earth, a camera representing a satellite, a directional light source representing the Sun, and a gray sphere representing the Moon. There are 3 functional C# scripts in the assets folder, each one corresponding to an object in the simulation.

## SatelliteOrbit.cs

The SatelliteOrbit.cs script corresponds to the camera object. It is responsible for making the satellite orbit around the Earth and spin while it is orbiting, and it does this through the transform.RotateAround() and transform.Rotate() methods respectively. The other main thing it is responsible for is computing the satellite’s approximate latitude and longitude and using those to determine what continent or ocean it is above.

For context, the Prime Meridian on the Earth is aligned approximately with the X axis in the Unity environment, so if the satellite is above it, then its longitude is 0 degrees. The simulation calculates latitude and longitude through the Vector3.Angle() and Vector3.SignedAngle() methods respectively. The satellite calculates the angles either between the satellites position and the positive x axis or between the satellite’s position and the positive y axis to find longitude and latitude. Then, the satellite checks if its latitude and longitude fall into any of a set of coordinate ranges. There are several coordinate ranges that represent rough rectangles on a world map each that pertain to the major parts of each region (the seven continents or four oceans). For example, for North America, there are coordinate ranges that cover Alaska, Canada, the USA, Mexico, Central America, and so on. If the satellite’s latitude and longitude fall into one of those coordinate ranges, then depending on the range, the satellite believes itself to be above a certain continent or ocean.

A full list of all coordinate ranges can be found in the Latitude Longitude Approximates Microsoft Excel file. A visual representation of all the coordinate ranges as rectangles for the seven continents can be found in the LatLonMap picture file. Rectangles are color coded by region as follows:

North America – Light blue

South America – Green

Europe – Purple

Asia – Red

Africa – Yellow

Australia/New Zealand – Orange

Antarctica - White

## SunRotation.cs

The SunRotation.cs script corresponds to the directional light object. It is responsible for moving the Sun. It does this through reading from a data table of coordinates generated through the DataGen program, the SunData.csv file, and interpolating the directional light’s position between two of these coordinates at a time. The table contains coordinates for the Sun at every hour over the course of two years, starting on January 1st 2019 at 12:00 AM and ending on December 31st 2020 at 11:00 PM.

## MoonRotation.cs

The MoonRotation.cs script corresponds to the gray sphere that represents the Moon. It serves largely the same function, but with a different data table: the MoonData.csv file.

## Config Files

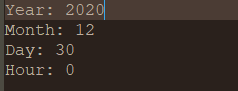
There are 2 .txt files with certain parameters that can configure the Unity simulation according to the user’s needs.

The DateConfig.txt file configures the date that the simulation will start at. There are four lines, corresponding to the year, month, day, and hour respectively. The year can be either 2019 or 2020 currently, but that can be changed (albeit manually), if the user wishes to use another csv file of coordinates. The month parameter is a number from 1 to 12 based on the month being selected. The day can only be in the valid date range for that month in order to work as expected. The hour can only be between 0 and 23. These numbers have to be in the specified ranges in order to prevent undefined behavior from occurring.

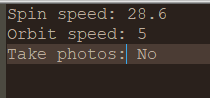
The SimConfig.txt file configures the speed of the orbit of the satellite around Earth along with the rotation speed of the satellite. The user can also specify whether they want to take pictures or not by putting “Yes” in the “Take Pictures?” line. If anything but the exact string “Yes” is put in that line, the simulation will not take photos

### *Config File Usage Examples*

DateConfig.txt file



SimConfig.txt file



# Orekit Guide

The DataGen Java program allows the user to generate csv files that contain coordinates for a given celestial body at various times through the usage of Orekit.

## Setup

In order for the program to be used, some setup and configuration is required. The Orekit library needs to be downloaded and integrated into the user’s work environment. After the library is integrated, JPL Ephemeride data (data on celestial bodies generated by NASA) in binary form needs to be downloaded separately, and a path needs to be set up to that binary data. The SunData.csv and MoonData.csv files were generated by using the JPL DE 430 ephemerides, which contain celestial body data from 1990 to 2069.

Full details on setup and configuration can be found in these links respectively:

<https://www.orekit.org/site-orekit-9.2/building.html>

<https://www.orekit.org/site-orekit-9.2/configuration.html> (The “Quick Setup with Default Data” section can assist users in getting the program in working order quickly)

## Program Overview

The DataGen program starts at a specific date specified by the datetime variable, which is an instance of Java’s LocalDateTime class. Orekit’s version of the Date class, AbsoluteDate, works off of the datetime variable. Two celestial bodies, the Sun and Moon, are instantiated in the program through the CelestialBody class. The main method used in getting coordinates is the CelestialBody class’s getPVCoordinates() method, which takes an AbsoluteDate and a Frame as parameters and outputs position and velocity vectors. The frame used in the program is the International Terrestrial Reference Frame, which is an ECEF frame.

After position and velocity vectors are obtained, only the position vector is saved in a Vector3D variable. The rest of the program is dedicated to looping that same process for 17,543 more times, making for a total of 17,544 sets of coordinates (the number of total hours in the years 2019 and 2020), and then entering those coordinates in a csv file. The csv file’s data in each line is arranged like so:

“id”, “Year”, “Month”, “Day”, “Hour”, “Minute”, “Second”, “X-coordinate”, “Y-coordinate”, and “Z-coordinate”