Low-Earth Orbit (LEO) Satellite Networks: Routing and Data Optimization

Presented by: Ahmad Harakeh
Ahmad Atwi
Jamil Abou Ltaif

Presented To: Dr. Sahar Hoteit
Dr. Alexis Aravanis

1. INTRODUCTION

This report focuses on the fusion of Low Earth Orbit (LEO) satellite networks and Software-Defined Networking (SDN) to enhance communication in the 6G era, making communication quicker and more reliable.

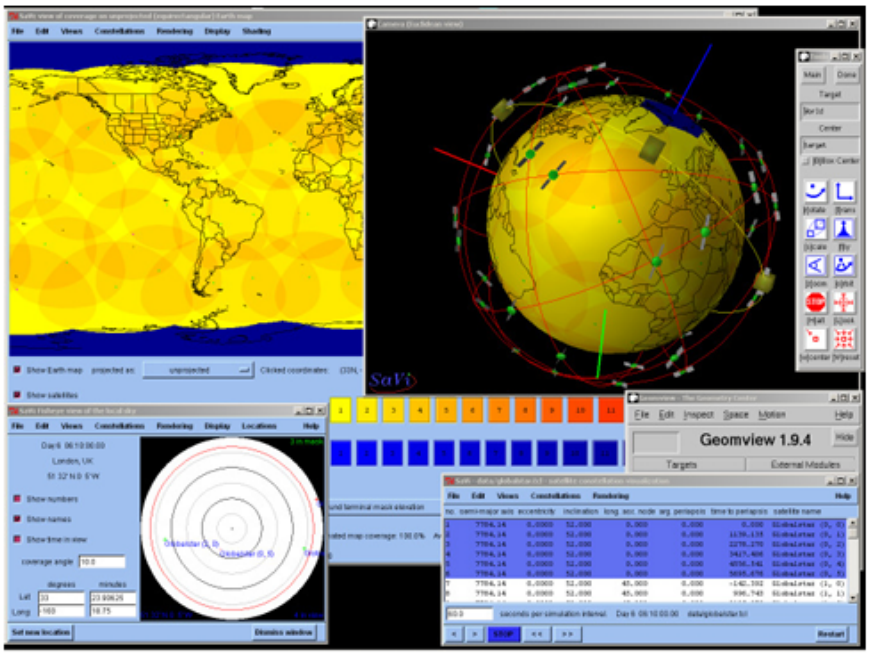
To make things practical, we'll be using the Savi Simulator. It's like a testing ground that helps us see how well our idea works in different situations. By doing this, Savi provides a dynamic and interactive platform for users to track and understand the orbital paths of satellites in real-time.

Utilizing SaVi, we can gain insights into the intricate movements and coverage areas of satellites, contributing to a comprehensive understanding of satellite behavior and communication reach.

2. TECHNICAL APPROACH

SaVi is a computer program that can work on its own or team up with another program called Geomview such that SaVi uses it mainly for making 3D pictures and maps using OpenGL. Even though Geomview can do more for mathematicians, SaVi keeps it simple by focusing on 3D visuals**.**

SaVi enables quick and easy explanation of the basic features of orbital motion and satellite geometry.



SaVi user interface showing Globalstar simulation, with coverage, fisheye and 3D view from Geomview.

2.1- SAVI INSTALLATION

SaVi, a tool for visualizing satellite constellations, can be installed and used on various

platforms that support Unix or have Unix capabilities.

Running Savi and Geomview under Windows:

a) Step 1: Install the necessarily tools

1. Search for ***'WSL'*** in the Microsoft Store and install it.
2. Install ***Ubuntu*** from the Microsoft Store.
3. With earlier versions of WSL, Add graphical ***X server*** functionality for Geomview. You could do this by installing a free X server like ***Xming*** and running it.

b) Step 2: Running Savi and Geomview

- Once WSL and Ubuntu are installed, open Ubuntu from the Search box to get a Unix bash command prompt, where we can type:

  sudo apt-get install geomview

  sudo apt-get install savi

- Then, check Display Settings: Verify the display settings by typing:

  echo $DISPLAY

- If not set, define the display by entering:

  export DISPLAY=:0

- Run Geomview with SaVi: Execute the command:

  geomview -run savi &

Running Savi and Geomview under Windows:

To have the updated version of SaVi you must follow these steps:

Note: Geomview installation is already stated.

1. Requirements for SaVi:

   - **tcl(tool command language) and tk(toolkit) libraries should be downloaded.**
     - **sudo apt-get install tcl**
     - **sudo apt-get install tcl-dev**
     - **sudo apt-get install tk**
     - **sudo apt-get install tk-dev**

- **zlib compression library(for compressing texturemap images sent to geomview)**
  - **sudo apt-get install zlib1g-dev**

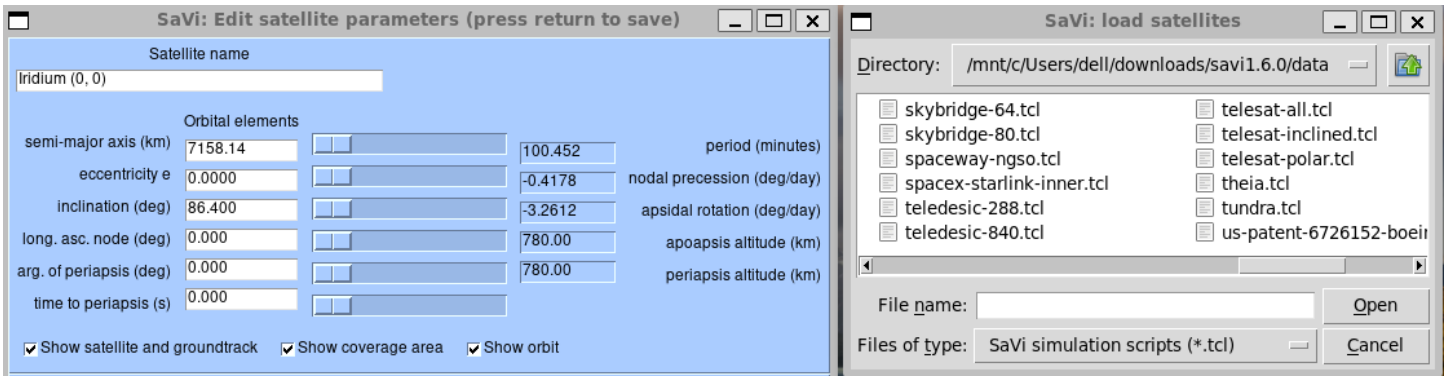
- Savi can optionally use:
  - jpegtopnm(for geomview to uncompress and display the detailed earth map)
    - **sudo apt-get install netpbm**
  - lynx(for handling https or other downloads of TLEs)
    - **sudo apt-get install lynx**
  - gifsicle: for saving animations of satellite coverage
    - **sudo apt-get install gifsicle**

2. Then after downloading the libraries you should access the desired directory:

- If You are using ubuntu on Windows:
  - cd /mnt/path_of_the_directory
  - Make ARCH=ubuntu

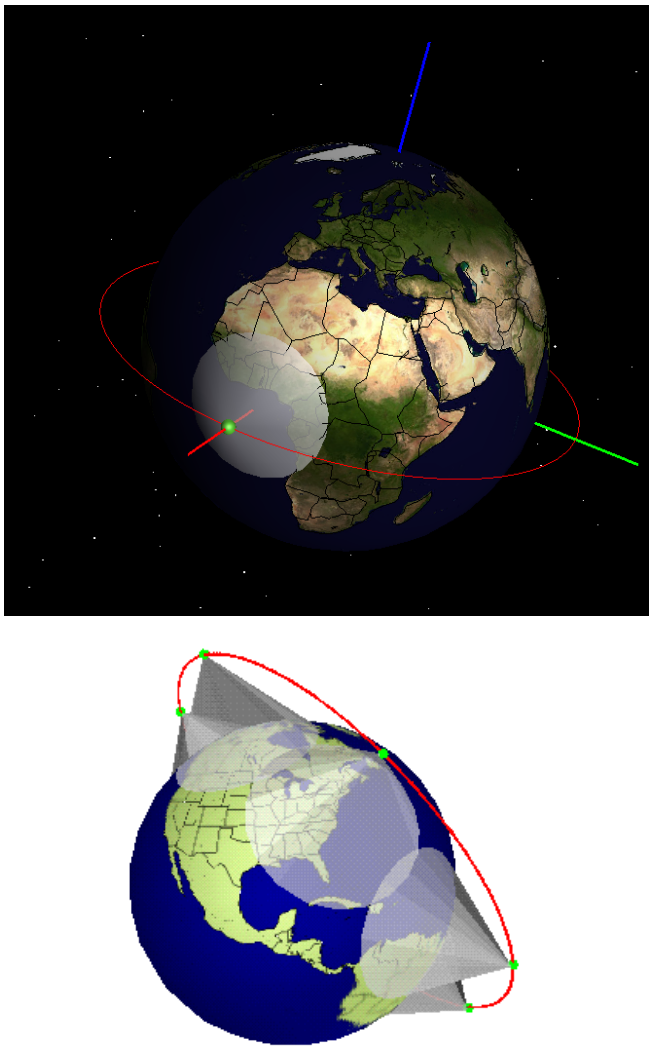
2.2- SAVI FEATURES

- It allows for the editing of satellite and constellation properties.



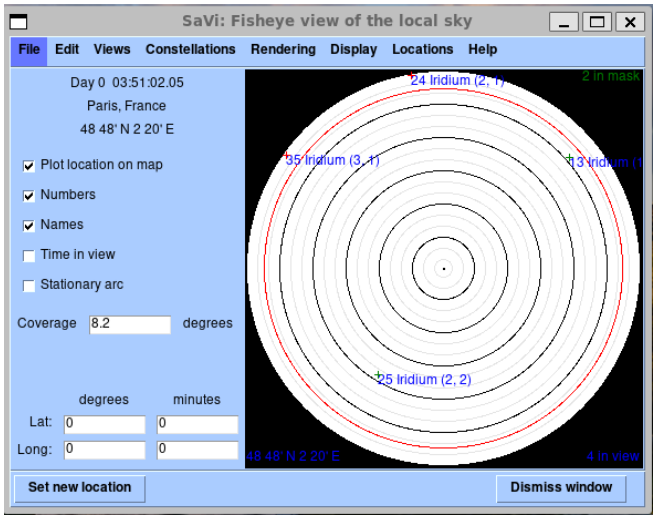
Edit a satellite in the constellation / Load satellites

- It provides a coverage cone for each satellite on the Earth map



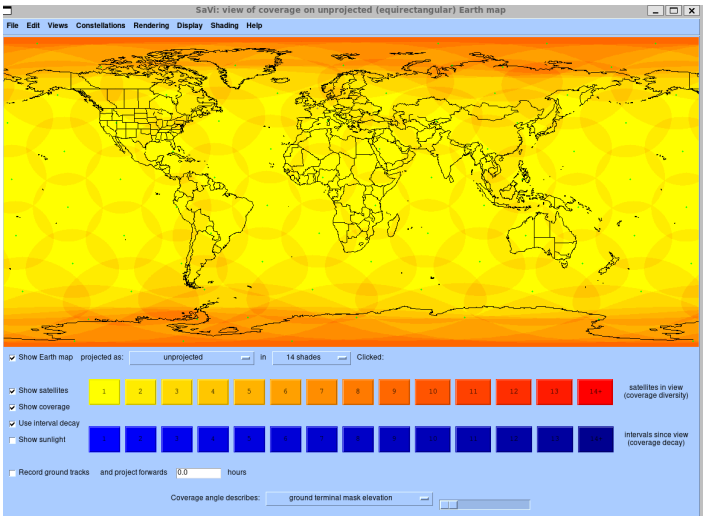
Coverage cone

- It provides a fisheye view of the sky to observe satellite movements over different Earth points.



Fisheye view of the local sky - Paris, France

- Satellite coverage can be shown.



View of coverage - Iridium Constellation

3. Objective

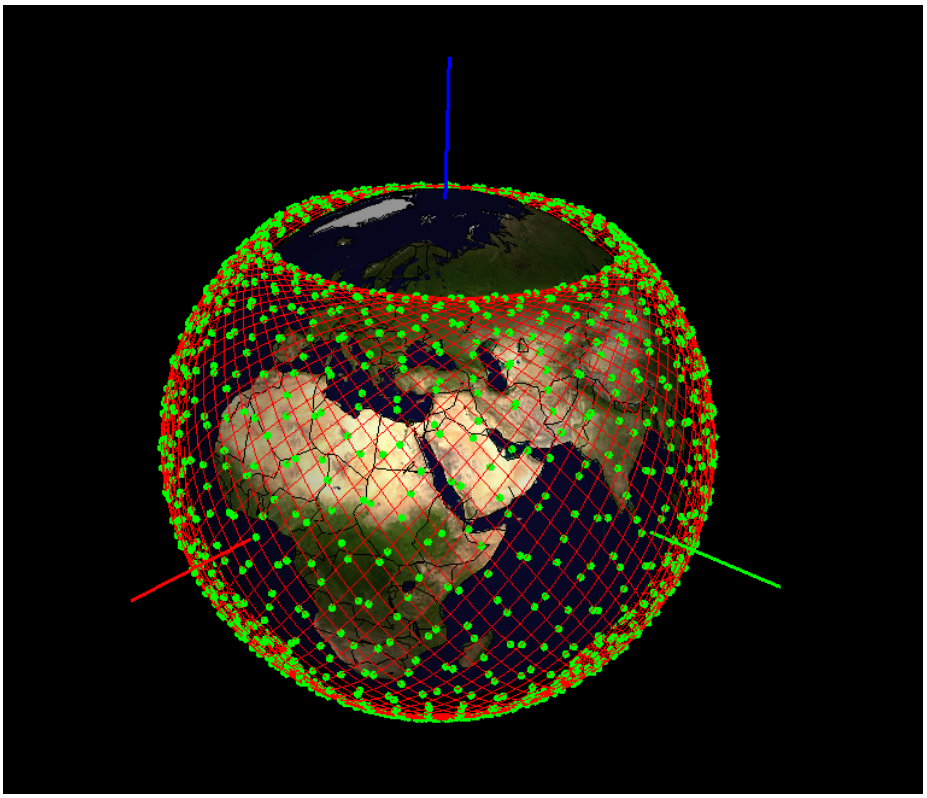
To test our concepts practically, we'll leverage the Savi Simulator, a tool for assessing our ideas under various conditions. Savi allows users to monitor and grasp the orbital trajectories of satellites in real-time.

Our aim is to try to build a fully connected grid showing the Inter Satellite Links (ISL) for various constellations, taking into account the extracted data from Savi. Then, using this grid, try to compute some metrics on it!

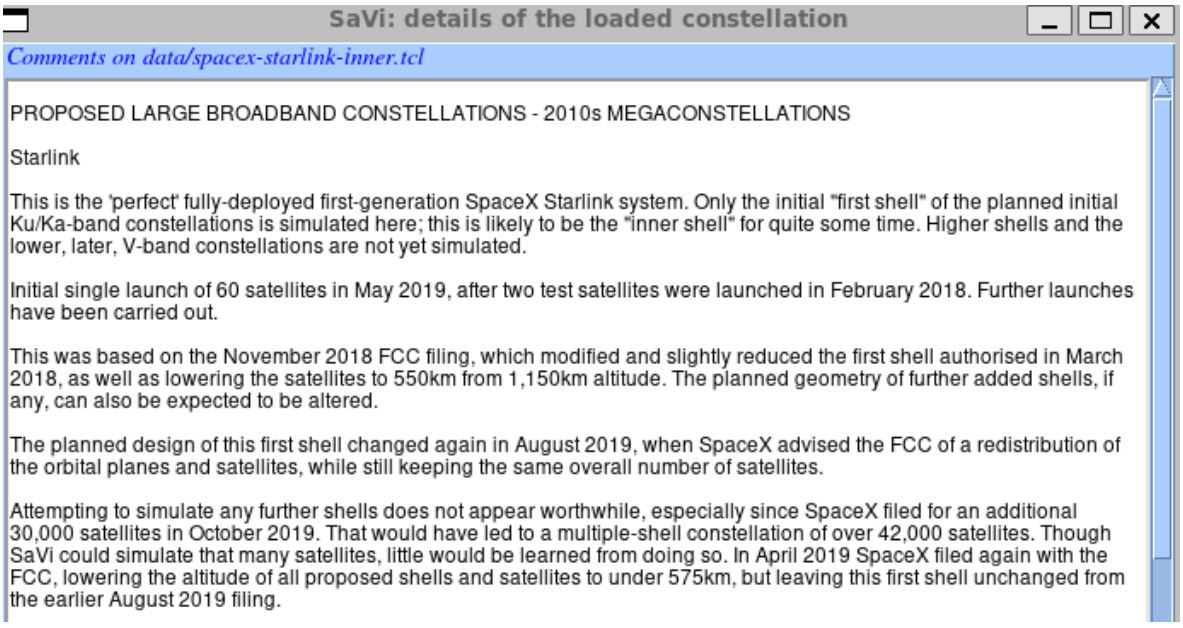
4- Required Tasks and Results

Task 1: Find 3 timeslots with 1 satellite from each of 3 constellations directly over Paris; Extract global satellite locations.

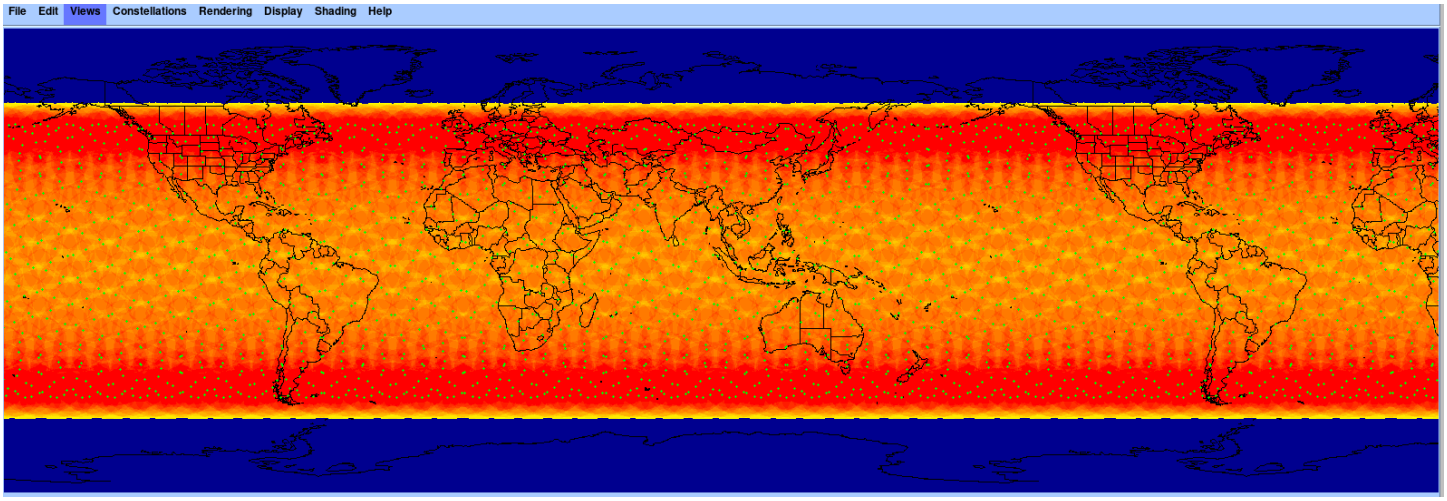
a) Starlink Constellation:



Starlink Constellation



Details of the Starlink Constellation



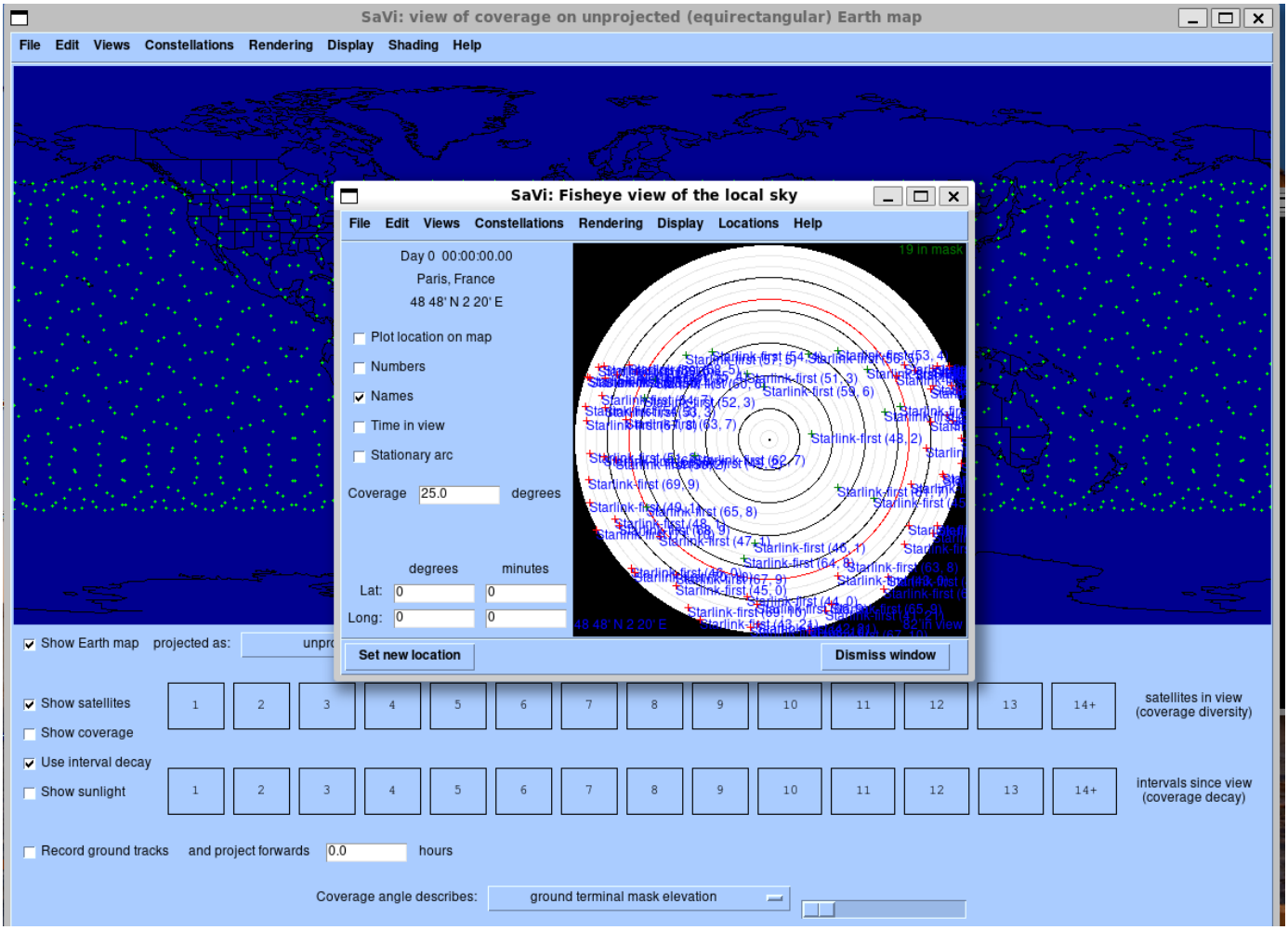
Coverage area of the Starlink Constellation

SaVi - data/spacex-starlink-inner.tcl - satellite constellation visualization

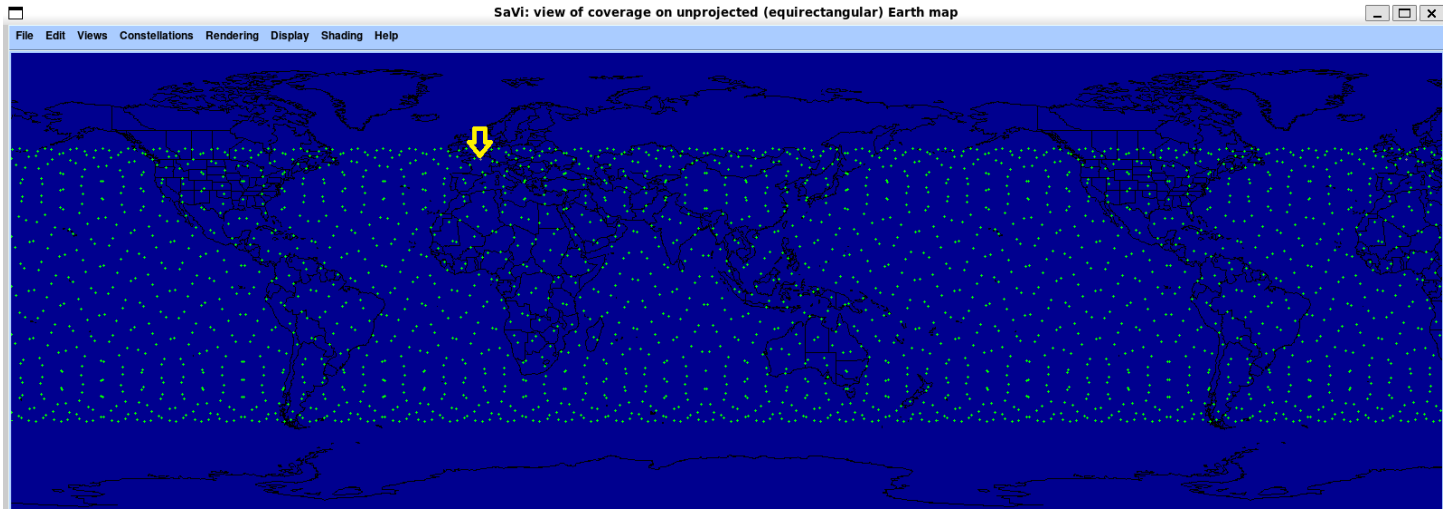
File Edit Views Constellations Rendering Help

no.	semi-major axis	eccentricity	inclination	longitude asc. node	arg. periapsis	time to periapsis	satellite name
1	6928.14	0.0000	53.000	0.000	0.000	0.000	Starlink-first (0, 0)
2	6928.14	0.0000	53.000	0.000	0.000	260.863	Starlink-first (0, 1)
3	6928.14	0.0000	53.000	0.000	0.000	521.726	Starlink-first (0, 2)
4	6928.14	0.0000	53.000	0.000	0.000	782.590	Starlink-first (0, 3)
5	6928.14	0.0000	53.000	0.000	0.000	1043.453	Starlink-first (0, 4)
6	6928.14	0.0000	53.000	0.000	0.000	1304.316	Starlink-first (0, 5)
7	6928.14	0.0000	53.000	0.000	0.000	1565.179	Starlink-first (0, 6)
8	6928.14	0.0000	53.000	0.000	0.000	1826.043	Starlink-first (0, 7)
9	6928.14	0.0000	53.000	0.000	0.000	2086.906	Starlink-first (0, 8)
10	6928.14	0.0000	53.000	0.000	0.000	2347.769	Starlink-first (0, 9)
11	6928.14	0.0000	53.000	0.000	0.000	2608.632	Starlink-first (0, 10)
12	6928.14	0.0000	53.000	0.000	0.000	2869.496	Starlink-first (0, 11)
13	6928.14	0.0000	53.000	0.000	0.000	3130.359	Starlink-first (0, 12)
14	6928.14	0.0000	53.000	0.000	0.000	3391.222	Starlink-first (0, 13)
15	6928.14	0.0000	53.000	0.000	0.000	3652.085	Starlink-first (0, 14)
16	6928.14	0.0000	53.000	0.000	0.000	3912.948	Starlink-first (0, 15)
17	6928.14	0.0000	53.000	0.000	0.000	4173.812	Starlink-first (0, 16)
18	6928.14	0.0000	53.000	0.000	0.000	4434.675	Starlink-first (0, 17)
19	6928.14	0.0000	53.000	0.000	0.000	4695.538	Starlink-first (0, 18)
20	6928.14	0.0000	53.000	0.000	0.000	4956.401	Starlink-first (0, 19)
21	6928.14	0.0000	53.000	0.000	0.000	5217.265	Starlink-first (0, 20)
22	6928.14	0.0000	53.000	0.000	0.000	5478.128	Starlink-first (0, 21)
23	6928.14	0.0000	53.000	5.000	0.000	-47.100	Starlink-first (1, 0)
24	6928.14	0.0000	53.000	5.000	0.000	213.763	Starlink-first (1, 1)
25	6928.14	0.0000	53.000	5.000	0.000	474.626	Starlink-first (1, 2)
26	6928.14	0.0000	53.000	5.000	0.000	735.489	Starlink-first (1, 3)
27	6928.14	0.0000	53.000	5.000	0.000	996.353	Starlink-first (1, 4)
28	6928.14	0.0000	53.000	5.000	0.000	1257.216	Starlink-first (1, 5)
29	6928.14	0.0000	53.000	5.000	0.000	1518.079	Starlink-first (1, 6)
30	6928.14	0.0000	53.000	5.000	0.000	1778.942	Starlink-first (1, 7)
31	6928.14	0.0000	53.000	5.000	0.000	2039.806	Starlink-first (1, 8)
32	6928.14	0.0000	53.000	5.000	0.000	2300.669	Starlink-first (1, 9)
33	6928.14	0.0000	53.000	5.000	0.000	2561.532	Starlink-first (1, 10)
34	6928.14	0.0000	53.000	5.000	0.000	2822.395	Starlink-first (1, 11)
35	6928.14	0.0000	53.000	5.000	0.000	3083.258	Starlink-first (1, 12)
36	6928.14	0.0000	53.000	5.000	0.000	3344.122	Starlink-first (1, 13)
37	6928.14	0.0000	53.000	5.000	0.000	3604.985	Starlink-first (1, 14)
38	6928.14	0.0000	53.000	5.000	0.000	3865.848	Starlink-first (1, 15)
39	6928.14	0.0000	53.000	5.000	0.000	4126.711	Starlink-first (1, 16)
40	6928.14	0.0000	53.000	5.000	0.000	4387.575	Starlink-first (1, 17)
41	6928.14	0.0000	53.000	5.000	0.000	4648.438	Starlink-first (1, 18)
42	6928.14	0.0000	53.000	5.000	0.000	4909.301	Starlink-first (1, 19)
43	6928.14	0.0000	53.000	5.000	0.000	5170.164	Starlink-first (1, 20)
44	6928.14	0.0000	53.000	5.000	0.000	5431.028	Starlink-first (1, 21)
45	6928.14	0.0000	53.000	10.000	0.000	-94.201	Starlink-first (2, 0)
46	6928.14	0.0000	53.000	10.000	0.000	166.663	Starlink-first (2, 1)
47	6928.14	0.0000	53.000	10.000	0.000	427.526	Starlink-first (2, 2)
48	6928.14	0.0000	53.000	10.000	0.000	688.389	Starlink-first (2, 3)
49	6928.14	0.0000	53.000	10.000	0.000	949.252	Starlink-first (2, 4)
50	6928.14	0.0000	53.000	10.000	0.000	1210.116	Starlink-first (2, 5)

List of Satellites



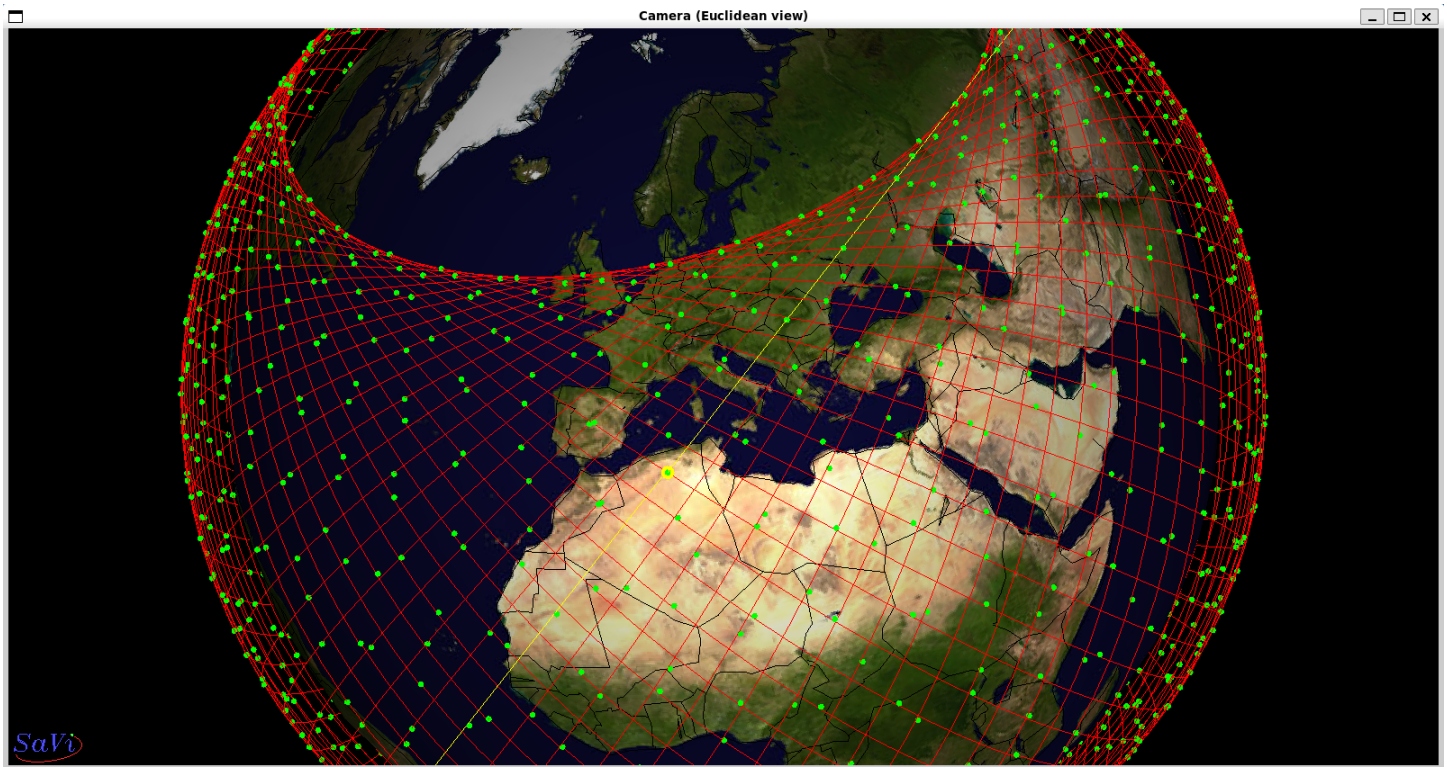
Fisheye view of the local sky - Paris



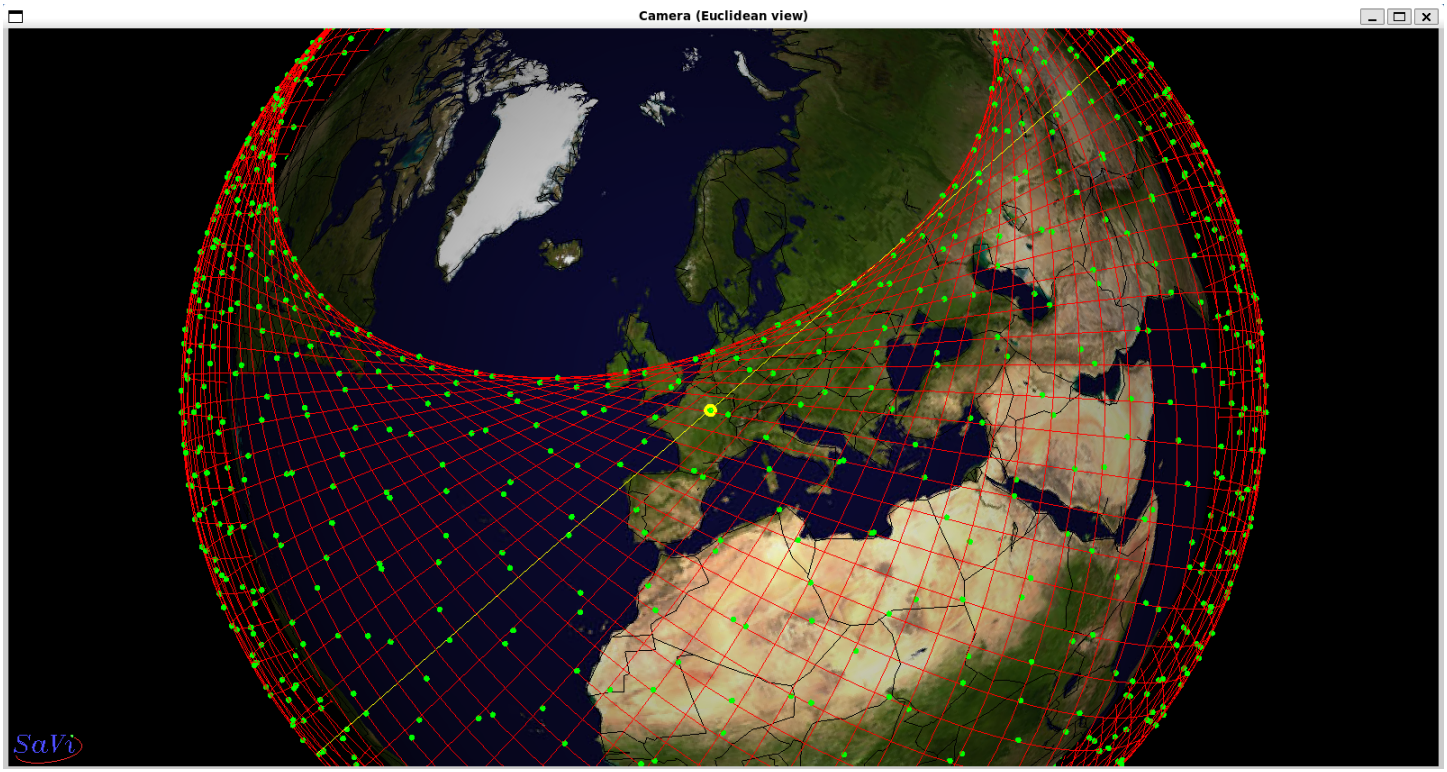
Plot Paris location on the map

no.	semi-major axis	eccentricity	inclination	longitude asc. node	arg. periapsis	time to periapsis	satellite name
1	6928.14	0.0000	53.000	0.000	0.000	0.000	Starlink-first (0, 0)
2	6928.14	0.0000	53.000	0.000	0.000	260.863	Starlink-first (0, 1)
3	6928.14	0.0000	53.000	0.000	0.000	521.726	Starlink-first (0, 2)
4	6928.14	0.0000	53.000	0.000	0.000	782.590	Starlink-first (0, 3)
5	6928.14	0.0000	53.000	0.000	0.000	1043.453	Starlink-first (0, 4)
6	6928.14	0.0000	53.000	0.000	0.000	1304.316	Starlink-first (0, 5)
7	6928.14	0.0000	53.000	0.000	0.000	1565.179	Starlink-first (0, 6)
8	6928.14	0.0000	53.000	0.000	0.000	1826.043	Starlink-first (0, 7)
9	6928.14	0.0000	53.000	0.000	0.000	2086.906	Starlink-first (0, 8)
10	6928.14	0.0000	53.000	0.000	0.000	2347.769	Starlink-first (0, 9)
11	6928.14	0.0000	53.000	0.000	0.000	2608.632	Starlink-first (0, 10)
12	6928.14	0.0000	53.000	0.000	0.000	2869.496	Starlink-first (0, 11)
13	6928.14	0.0000	53.000	0.000	0.000	3130.359	Starlink-first (0, 12)
14	6928.14	0.0000	53.000	0.000	0.000	3391.222	Starlink-first (0, 13)
15	6928.14	0.0000	53.000	0.000	0.000	3652.085	Starlink-first (0, 14)
16	6928.14	0.0000	53.000	0.000	0.000	3912.948	Starlink-first (0, 15)
17	6928.14	0.0000	53.000	0.000	0.000	4173.812	Starlink-first (0, 16)
18	6928.14	0.0000	53.000	0.000	0.000	4434.675	Starlink-first (0, 17)
19	6928.14	0.0000	53.000	0.000	0.000	4695.538	Starlink-first (0, 18)
20	6928.14	0.0000	53.000	0.000	0.000	4956.401	Starlink-first (0, 19)
21	6928.14	0.0000	53.000	0.000	0.000	5217.265	Starlink-first (0, 20)
22	6928.14	0.0000	53.000	0.000	0.000	5478.128	Starlink-first (0, 21)
23	6928.14	0.0000	53.000	5.000	0.000	-47.100	Starlink-first (1, 0)
24	6928.14	0.0000	53.000	5.000	0.000	213.763	Starlink-first (1, 1)
25	6928.14	0.0000	53.000	5.000	0.000	474.626	Starlink-first (1, 2)
26	6928.14	0.0000	53.000	5.000	0.000	735.489	Starlink-first (1, 3)
27	6928.14	0.0000	53.000	5.000	0.000	996.353	Starlink-first (1, 4)
28	6928.14	0.0000	53.000	5.000	0.000	1257.216	Starlink-first (1, 5)
29	6928.14	0.0000	53.000	5.000	0.000	1518.079	Starlink-first (1, 6)
30	6928.14	0.0000	53.000	5.000	0.000	1778.942	Starlink-first (1, 7)
31	6928.14	0.0000	53.000	5.000	0.000	2039.806	Starlink-first (1, 8)
32	6928.14	0.0000	53.000	5.000	0.000	2300.669	Starlink-first (1, 9)
33	6928.14	0.0000	53.000	5.000	0.000	2561.532	Starlink-first (1, 10)
34	6928.14	0.0000	53.000	5.000	0.000	2822.395	Starlink-first (1, 11)
35	6928.14	0.0000	53.000	5.000	0.000	3083.258	Starlink-first (1, 12)
36	6928.14	0.0000	53.000	5.000	0.000	3344.122	Starlink-first (1, 13)
37	6928.14	0.0000	53.000	5.000	0.000	3604.985	Starlink-first (1, 14)
38	6928.14	0.0000	53.000	5.000	0.000	3865.848	Starlink-first (1, 15)
39	6928.14	0.0000	53.000	5.000	0.000	4126.711	Starlink-first (1, 16)
40	6928.14	0.0000	53.000	5.000	0.000	4387.575	Starlink-first (1, 17)
41	6928.14	0.0000	53.000	5.000	0.000	4648.438	Starlink-first (1, 18)
42	6928.14	0.0000	53.000	5.000	0.000	4909.301	Starlink-first (1, 19)
43	6928.14	0.0000	53.000	5.000	0.000	5170.164	Starlink-first (1, 20)
44	6928.14	0.0000	53.000	5.000	0.000	5431.028	Starlink-first (1, 21)
45	6928.14	0.0000	53.000	10.000	0.000	-94.201	Starlink-first (2, 0)
46	6928.14	0.0000	53.000	10.000	0.000	166.663	Starlink-first (2, 1)
47	6928.14	0.0000	53.000	10.000	0.000	427.526	Starlink-first (2, 2)
48	6928.14	0.0000	53.000	10.000	0.000	688.389	Starlink-first (2, 3)
49	6928.14	0.0000	53.000	10.000	0.000	949.252	Starlink-first (2, 4)
50	6928.14	0.0000	53.000	10.000	0.000	1210.116	Starlink-first (2, 5)

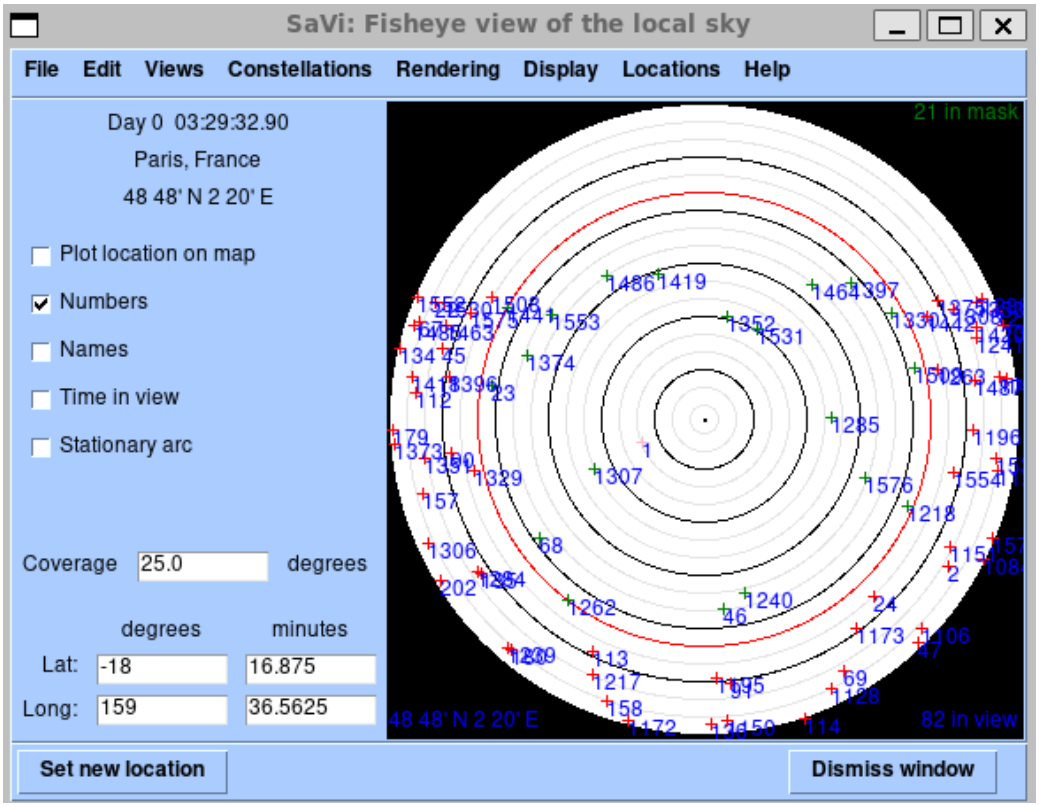
Select a satellite from the constellation - Satellite 0



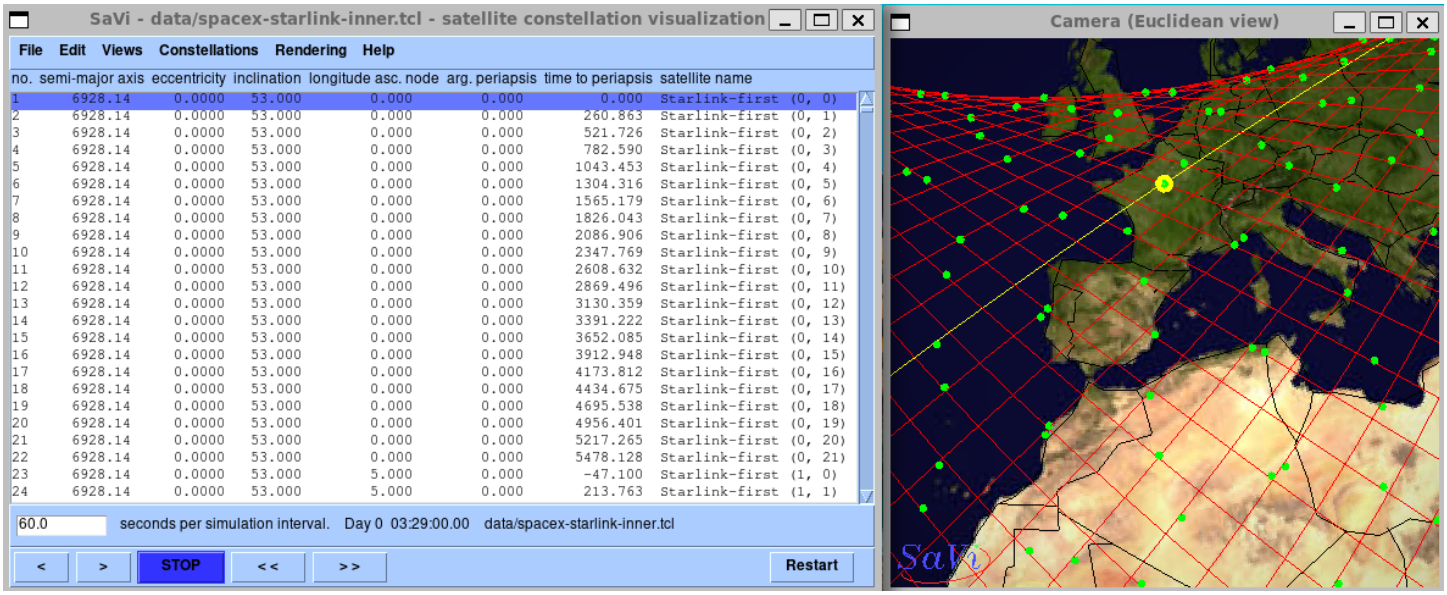
The orbit of the selected satellite



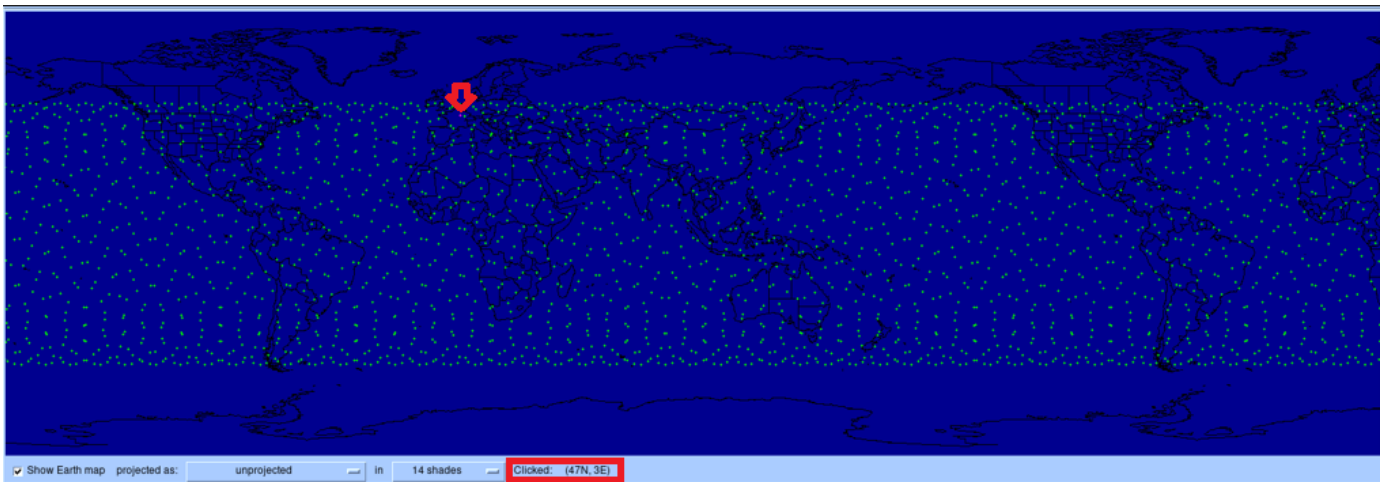
The selected Satellite reaches Paris Area



Fisheye view of the local sky - Paris (Satellite 0 is within this area)



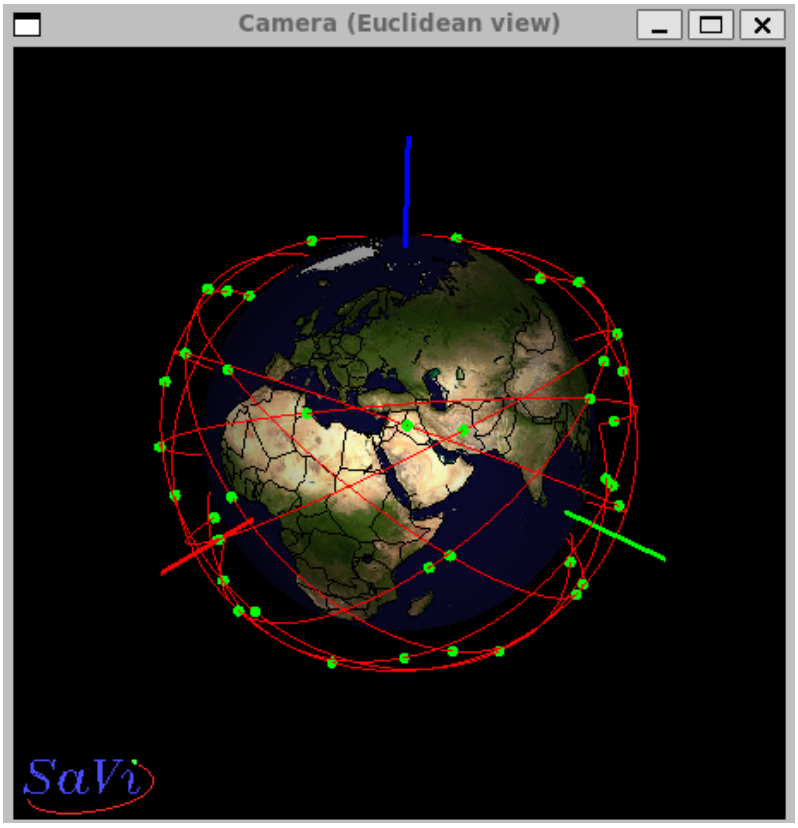
The timeslot for a satellite from the Starlink Constellation to reach Paris is about: 03:29:00 minutes.



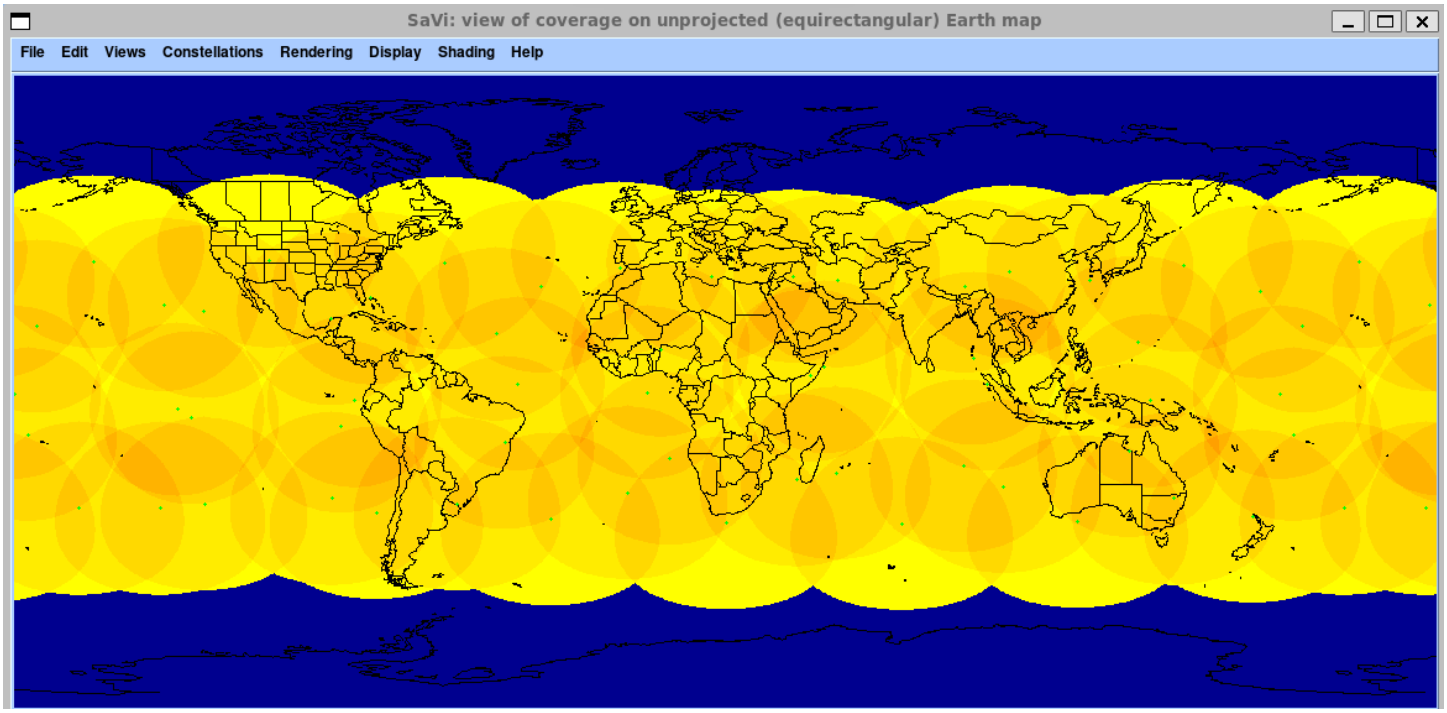
The position of the selected satellite is: (Latitude: 47N, Longitude: 3E)

b) Telesat Constellation:

Telesat54 (inclined Telesat):



Inclined Telesat Constellation



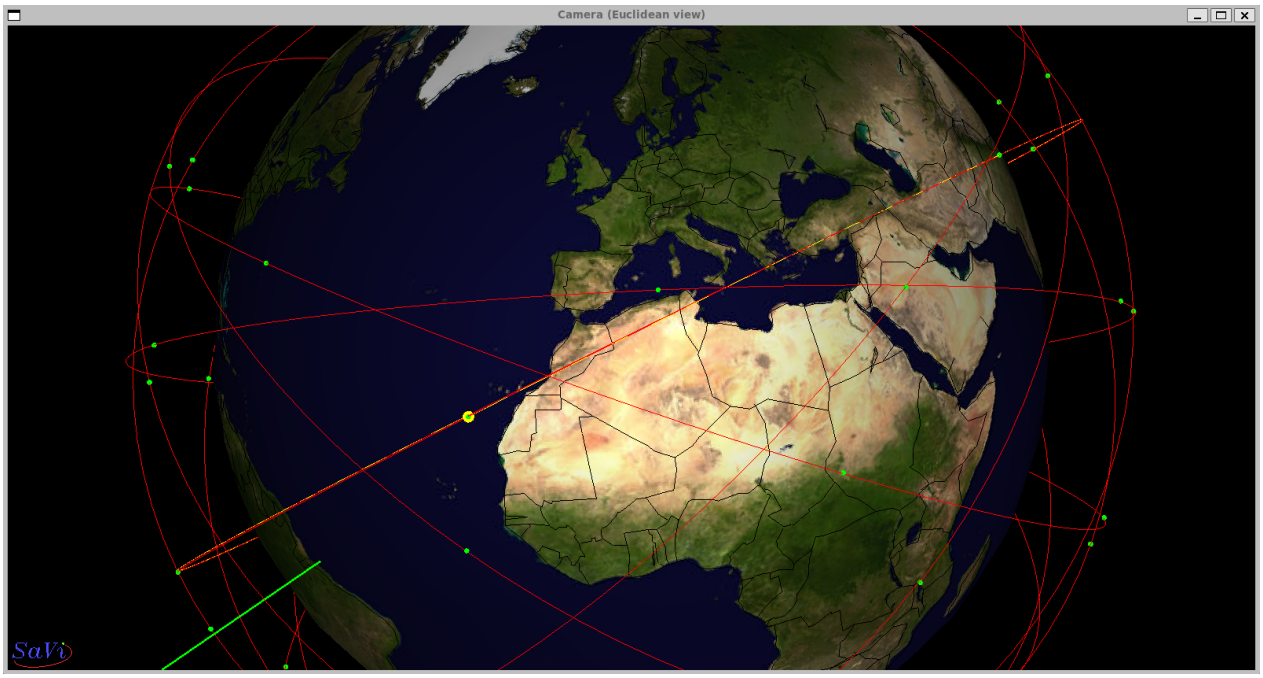
Coverage area of the Inclined Telesat Constellation

SaVi - data/telesat-incline

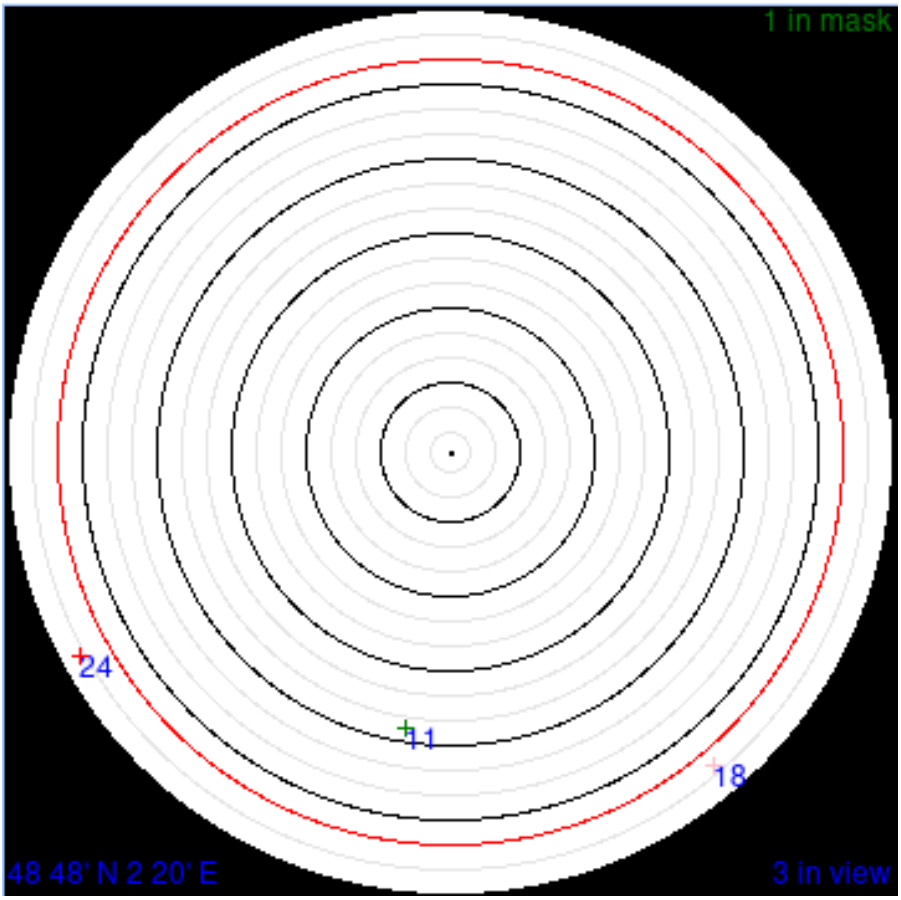
File Edit Views Constellations Rendering Help

no.	semi-major axis	eccentricity	inclination	longitude asc. node	arg. periapsis	time to periapsis	satellite name
1	7628.14	0.0000	37.400	0.000	0.000	0.000	Telesat-inclined (0, 0)
2	7628.14	0.0000	37.400	0.000	0.000	1105.064	Telesat-inclined (0, 1)
3	7628.14	0.0000	37.400	0.000	0.000	2210.127	Telesat-inclined (0, 2)
4	7628.14	0.0000	37.400	0.000	0.000	3315.191	Telesat-inclined (0, 3)
5	7628.14	0.0000	37.400	0.000	0.000	4420.255	Telesat-inclined (0, 4)
6	7628.14	0.0000	37.400	0.000	0.000	5525.318	Telesat-inclined (0, 5)
7	7628.14	0.0000	37.400	40.000	0.000	-147.342	Telesat-inclined (1, 0)
8	7628.14	0.0000	37.400	40.000	0.000	957.722	Telesat-inclined (1, 1)
9	7628.14	0.0000	37.400	40.000	0.000	2062.786	Telesat-inclined (1, 2)
10	7628.14	0.0000	37.400	40.000	0.000	3167.849	Telesat-inclined (1, 3)
11	7628.14	0.0000	37.400	40.000	0.000	4272.913	Telesat-inclined (1, 4)
12	7628.14	0.0000	37.400	40.000	0.000	5377.977	Telesat-inclined (1, 5)
13	7628.14	0.0000	37.400	80.000	0.000	-294.684	Telesat-inclined (2, 0)
14	7628.14	0.0000	37.400	80.000	0.000	810.380	Telesat-inclined (2, 1)
15	7628.14	0.0000	37.400	80.000	0.000	1915.444	Telesat-inclined (2, 2)
16	7628.14	0.0000	37.400	80.000	0.000	3020.507	Telesat-inclined (2, 3)
17	7628.14	0.0000	37.400	80.000	0.000	4125.571	Telesat-inclined (2, 4)
18	7628.14	0.0000	37.400	80.000	0.000	5230.635	Telesat-inclined (2, 5)
19	7628.14	0.0000	37.400	120.000	0.000	-442.025	Telesat-inclined (3, 0)
20	7628.14	0.0000	37.400	120.000	0.000	663.038	Telesat-inclined (3, 1)
21	7628.14	0.0000	37.400	120.000	0.000	1768.102	Telesat-inclined (3, 2)
22	7628.14	0.0000	37.400	120.000	0.000	2873.166	Telesat-inclined (3, 3)
23	7628.14	0.0000	37.400	120.000	0.000	3978.229	Telesat-inclined (3, 4)
24	7628.14	0.0000	37.400	120.000	0.000	5083.293	Telesat-inclined (3, 5)
25	7628.14	0.0000	37.400	160.000	0.000	-589.367	Telesat-inclined (4, 0)
26	7628.14	0.0000	37.400	160.000	0.000	515.696	Telesat-inclined (4, 1)
27	7628.14	0.0000	37.400	160.000	0.000	1620.760	Telesat-inclined (4, 2)
28	7628.14	0.0000	37.400	160.000	0.000	2725.824	Telesat-inclined (4, 3)
29	7628.14	0.0000	37.400	160.000	0.000	3830.887	Telesat-inclined (4, 4)
30	7628.14	0.0000	37.400	160.000	0.000	4935.951	Telesat-inclined (4, 5)
31	7628.14	0.0000	37.400	200.000	0.000	-736.709	Telesat-inclined (5, 0)
32	7628.14	0.0000	37.400	200.000	0.000	368.355	Telesat-inclined (5, 1)
33	7628.14	0.0000	37.400	200.000	0.000	1473.418	Telesat-inclined (5, 2)
34	7628.14	0.0000	37.400	200.000	0.000	2578.482	Telesat-inclined (5, 3)
35	7628.14	0.0000	37.400	200.000	0.000	3683.546	Telesat-inclined (5, 4)
36	7628.14	0.0000	37.400	200.000	0.000	4788.609	Telesat-inclined (5, 5)
37	7628.14	0.0000	37.400	240.000	0.000	-884.051	Telesat-inclined (6, 0)
38	7628.14	0.0000	37.400	240.000	0.000	221.013	Telesat-inclined (6, 1)
39	7628.14	0.0000	37.400	240.000	0.000	1326.076	Telesat-inclined (6, 2)
40	7628.14	0.0000	37.400	240.000	0.000	2431.140	Telesat-inclined (6, 3)
41	7628.14	0.0000	37.400	240.000	0.000	3536.204	Telesat-inclined (6, 4)
42	7628.14	0.0000	37.400	240.000	0.000	4641.267	Telesat-inclined (6, 5)
43	7628.14	0.0000	37.400	280.000	0.000	-1031.393	Telesat-inclined (7, 0)
44	7628.14	0.0000	37.400	280.000	0.000	73.671	Telesat-inclined (7, 1)
45	7628.14	0.0000	37.400	280.000	0.000	1178.735	Telesat-inclined (7, 2)
46	7628.14	0.0000	37.400	280.000	0.000	2283.798	Telesat-inclined (7, 3)
47	7628.14	0.0000	37.400	280.000	0.000	3388.862	Telesat-inclined (7, 4)
48	7628.14	0.0000	37.400	280.000	0.000	4493.926	Telesat-inclined (7, 5)
49	7628.14	0.0000	37.400	320.000	0.000	-1178.735	Telesat-inclined (8, 0)
50	7628.14	0.0000	37.400	320.000	0.000	-73.671	Telesat-inclined (8, 1)

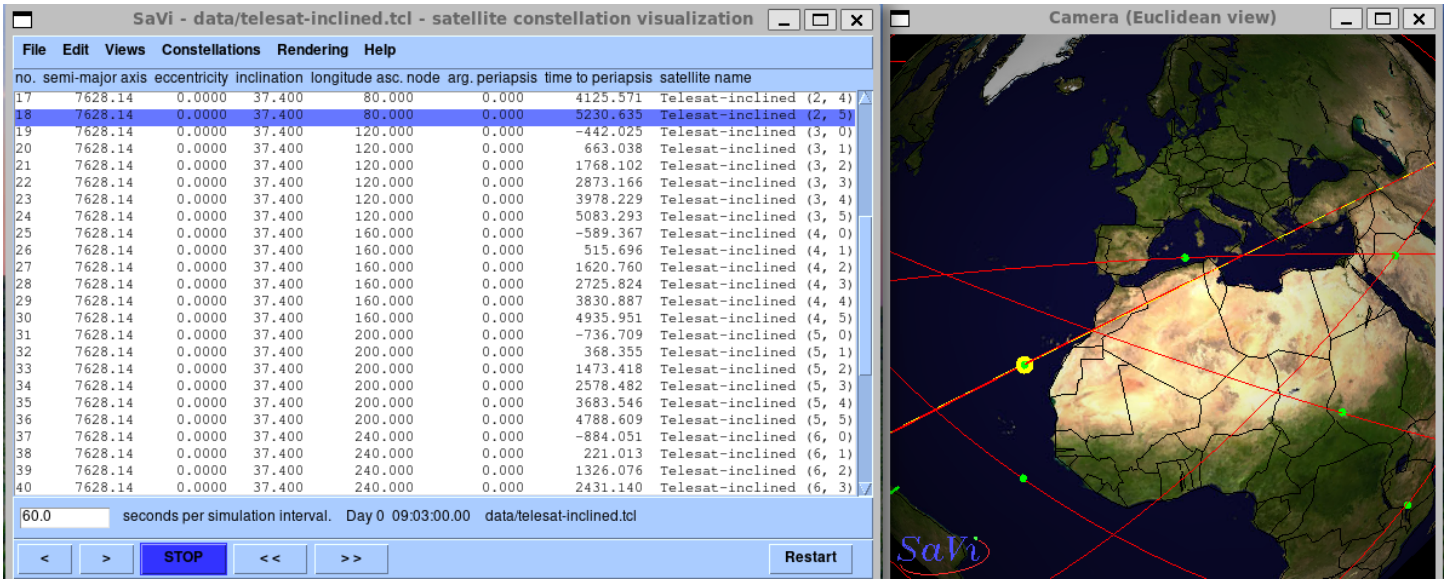
Select a satellite from the constellation - Satellite 18



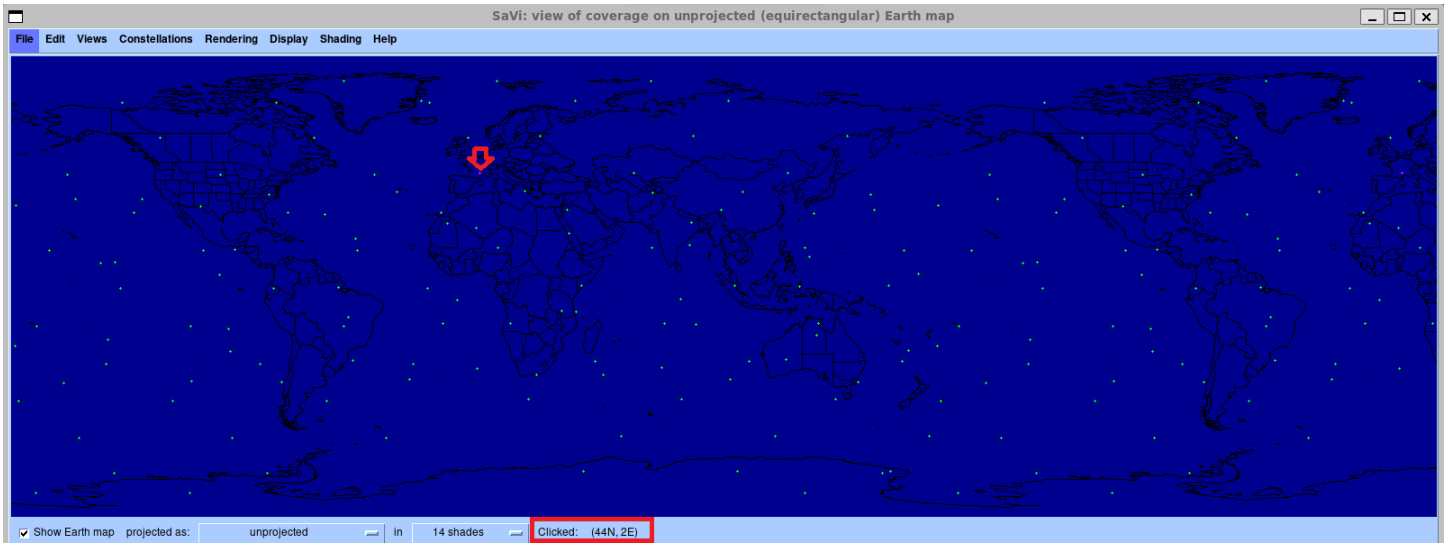
The selected satellite - Satellite 18



The selected Satellite reaches Paris Area (Satellite 18)

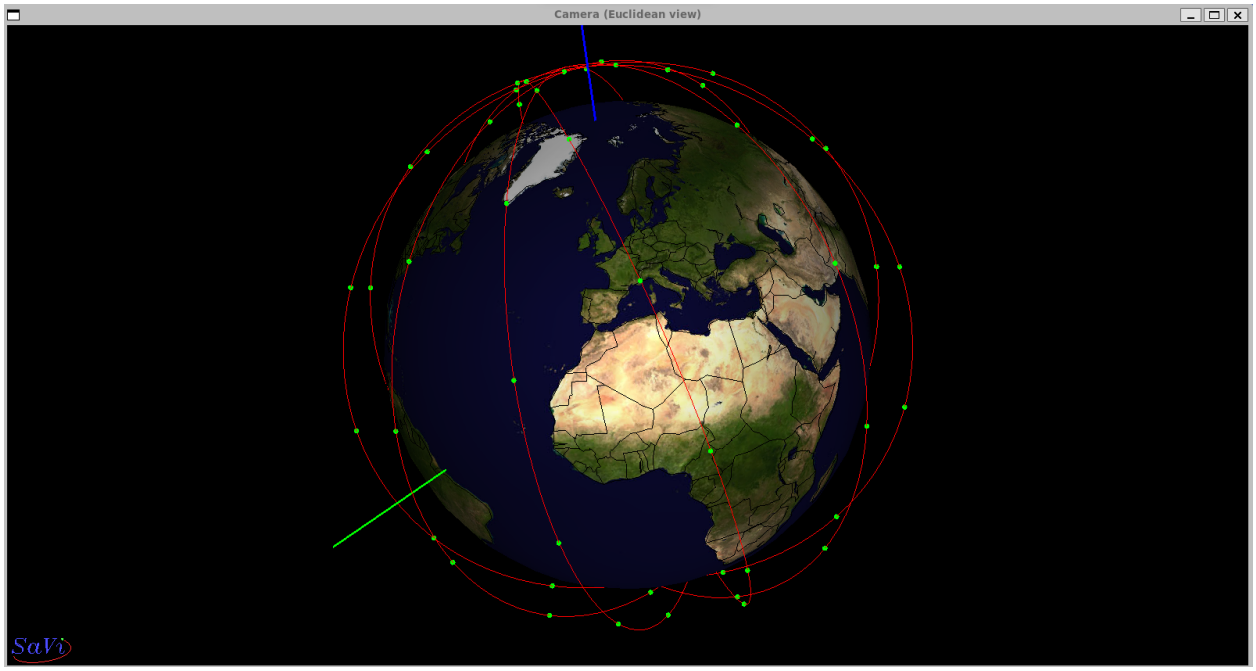


The timeslot for a satellite from the Telesat Constellation to reach Paris is about: 09:03:00.00 minutes.

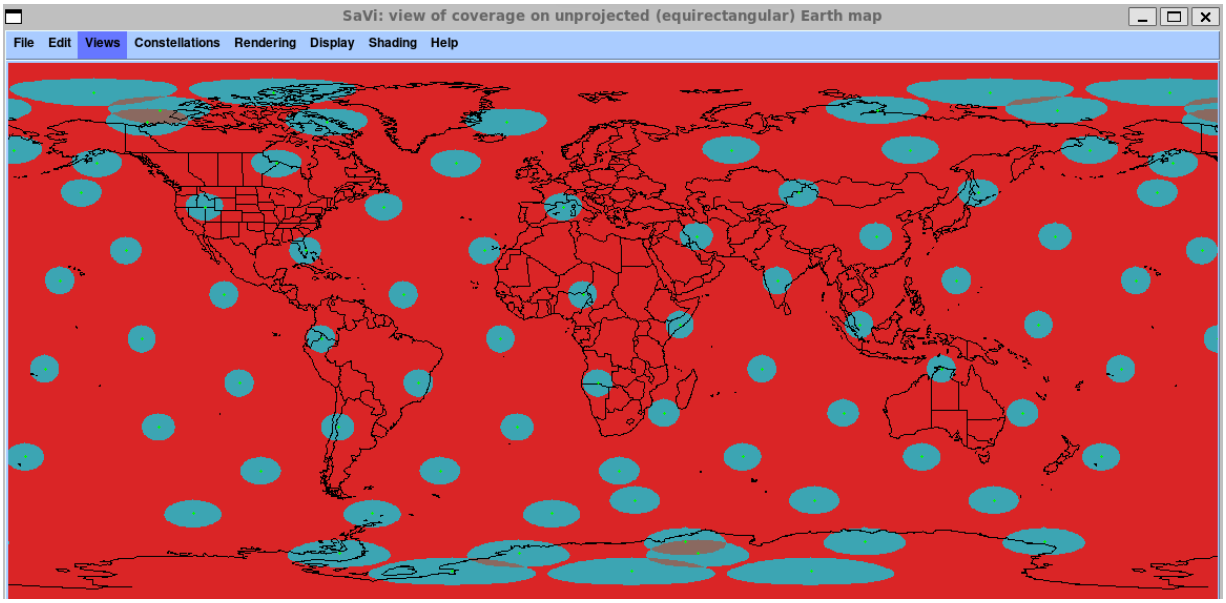


The position of the selected satellite is: (Latitude: 44N, Longitude: 2E)

Telesat74 (Polar Telesat):



Polar Telesat Constellation



Coverage area of the Polar Telesat Constellation

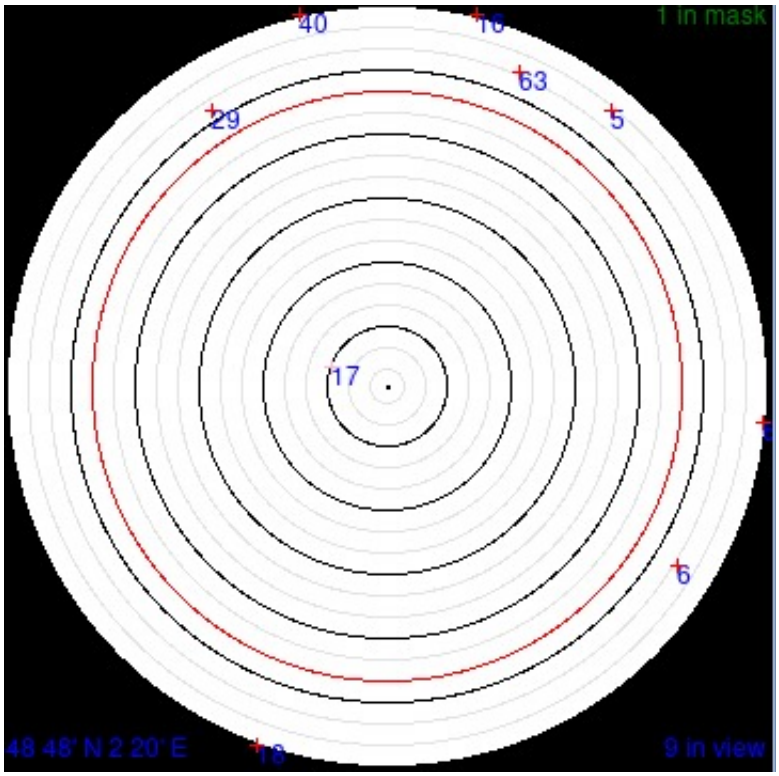
SaVi - data/telesat-pc

File Edit Views Constellations Rendering Help

no.	semi-major axis	eccentricity	inclination	longitude asc. node	arg. periapsis	time to periapsis	satellite name
1	7378.14	0.0000	99.500	0.000	0.000	0.000	Telesat-polar (0, 0)
2	7378.14	0.0000	99.500	0.000	0.000	525.593	Telesat-polar (0, 1)
3	7378.14	0.0000	99.500	0.000	0.000	1051.186	Telesat-polar (0, 2)
4	7378.14	0.0000	99.500	0.000	0.000	1576.779	Telesat-polar (0, 3)
5	7378.14	0.0000	99.500	0.000	0.000	2102.372	Telesat-polar (0, 4)
6	7378.14	0.0000	99.500	0.000	0.000	2627.966	Telesat-polar (0, 5)
7	7378.14	0.0000	99.500	0.000	0.000	3153.559	Telesat-polar (0, 6)
8	7378.14	0.0000	99.500	0.000	0.000	3679.152	Telesat-polar (0, 7)
9	7378.14	0.0000	99.500	0.000	0.000	4204.745	Telesat-polar (0, 8)
10	7378.14	0.0000	99.500	0.000	0.000	4730.338	Telesat-polar (0, 9)
11	7378.14	0.0000	99.500	0.000	0.000	5255.931	Telesat-polar (0, 10)
12	7378.14	0.0000	99.500	0.000	0.000	5781.524	Telesat-polar (0, 11)
13	7378.14	0.0000	99.500	30.000	0.000	262.797	Telesat-polar (1, 0)
14	7378.14	0.0000	99.500	30.000	0.000	788.390	Telesat-polar (1, 1)
15	7378.14	0.0000	99.500	30.000	0.000	1313.983	Telesat-polar (1, 2)
16	7378.14	0.0000	99.500	30.000	0.000	1839.576	Telesat-polar (1, 3)
17	7378.14	0.0000	99.500	30.000	0.000	2365.169	Telesat-polar (1, 4)
18	7378.14	0.0000	99.500	30.000	0.000	2890.762	Telesat-polar (1, 5)
19	7378.14	0.0000	99.500	30.000	0.000	3416.355	Telesat-polar (1, 6)
20	7378.14	0.0000	99.500	30.000	0.000	3941.948	Telesat-polar (1, 7)
21	7378.14	0.0000	99.500	30.000	0.000	4467.541	Telesat-polar (1, 8)
22	7378.14	0.0000	99.500	30.000	0.000	4993.134	Telesat-polar (1, 9)
23	7378.14	0.0000	99.500	30.000	0.000	5518.728	Telesat-polar (1, 10)
24	7378.14	0.0000	99.500	30.000	0.000	6044.321	Telesat-polar (1, 11)
25	7378.14	0.0000	99.500	60.000	0.000	0.000	Telesat-polar (2, 0)
26	7378.14	0.0000	99.500	60.000	0.000	525.593	Telesat-polar (2, 1)
27	7378.14	0.0000	99.500	60.000	0.000	1051.186	Telesat-polar (2, 2)
28	7378.14	0.0000	99.500	60.000	0.000	1576.779	Telesat-polar (2, 3)
29	7378.14	0.0000	99.500	60.000	0.000	2102.372	Telesat-polar (2, 4)
30	7378.14	0.0000	99.500	60.000	0.000	2627.966	Telesat-polar (2, 5)
31	7378.14	0.0000	99.500	60.000	0.000	3153.559	Telesat-polar (2, 6)
32	7378.14	0.0000	99.500	60.000	0.000	3679.152	Telesat-polar (2, 7)
33	7378.14	0.0000	99.500	60.000	0.000	4204.745	Telesat-polar (2, 8)
34	7378.14	0.0000	99.500	60.000	0.000	4730.338	Telesat-polar (2, 9)
35	7378.14	0.0000	99.500	60.000	0.000	5255.931	Telesat-polar (2, 10)
36	7378.14	0.0000	99.500	60.000	0.000	5781.524	Telesat-polar (2, 11)
37	7378.14	0.0000	99.500	90.000	0.000	262.797	Telesat-polar (3, 0)
38	7378.14	0.0000	99.500	90.000	0.000	788.390	Telesat-polar (3, 1)
39	7378.14	0.0000	99.500	90.000	0.000	1313.983	Telesat-polar (3, 2)
40	7378.14	0.0000	99.500	90.000	0.000	1839.576	Telesat-polar (3, 3)
41	7378.14	0.0000	99.500	90.000	0.000	2365.169	Telesat-polar (3, 4)
42	7378.14	0.0000	99.500	90.000	0.000	2890.762	Telesat-polar (3, 5)
43	7378.14	0.0000	99.500	90.000	0.000	3416.355	Telesat-polar (3, 6)
44	7378.14	0.0000	99.500	90.000	0.000	3941.948	Telesat-polar (3, 7)
45	7378.14	0.0000	99.500	90.000	0.000	4467.541	Telesat-polar (3, 8)
46	7378.14	0.0000	99.500	90.000	0.000	4993.134	Telesat-polar (3, 9)
47	7378.14	0.0000	99.500	90.000	0.000	5518.728	Telesat-polar (3, 10)
48	7378.14	0.0000	99.500	90.000	0.000	6044.321	Telesat-polar (3, 11)
49	7378.14	0.0000	99.500	120.000	0.000	0.000	Telesat-polar (4, 0)
50	7378.14	0.0000	99.500	120.000	0.000	525.593	Telesat-polar (4, 1)

Select a satellite from the constellation - Satellite 17

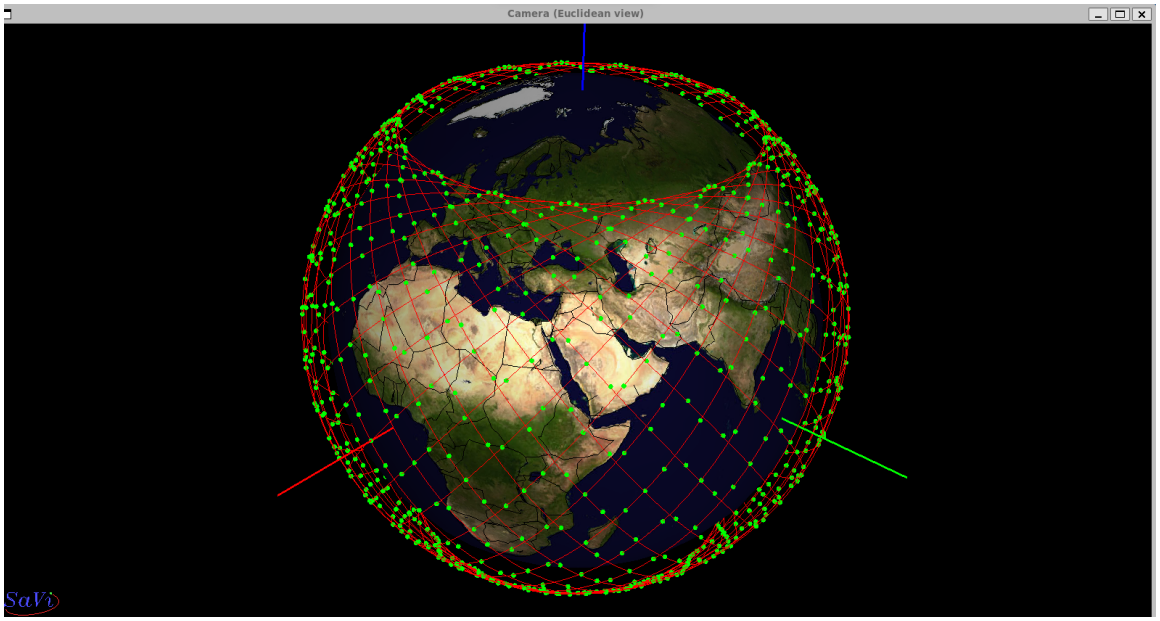
The selected satellite - Satellite 17



The selected Satellite reaches Paris Area (Satellite 17)

The timeslot for a satellite from the Telesat Constellation to reach Paris is about: 00:54:00.00 minutes.

c) Project Kuiper Constellation:



Project Kuiper Constellation (first shell)

Comments on data/amazon-project-kuiper-first.tcl

PROPOSED LARGE BROADBAND CONSTELLATIONS - 2010s MEGACONSTELLATIONS

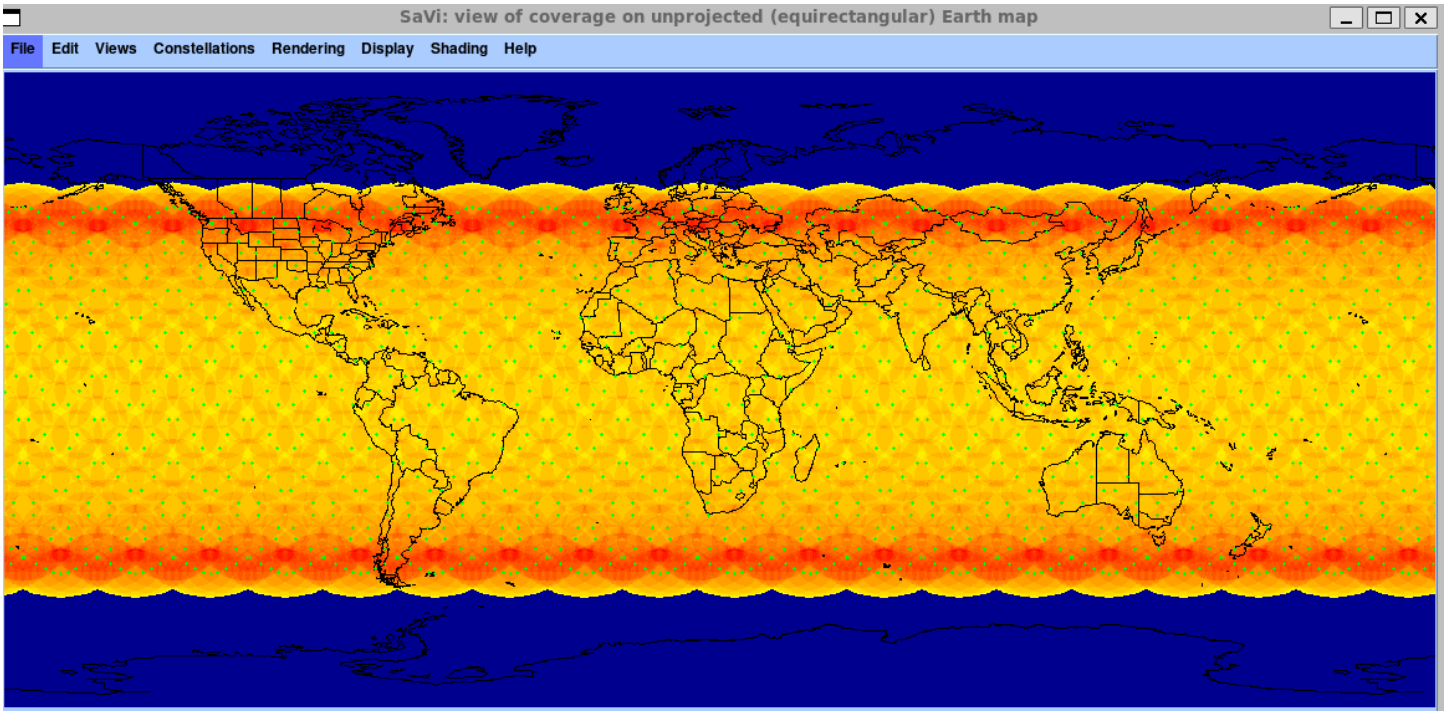
Project Kuiper

This is the 'perfect' fully-deployed first-generation Amazon Project Kuiper system. Only the initial "first shell" of the planned Ka-band constellations is simulated in this script.

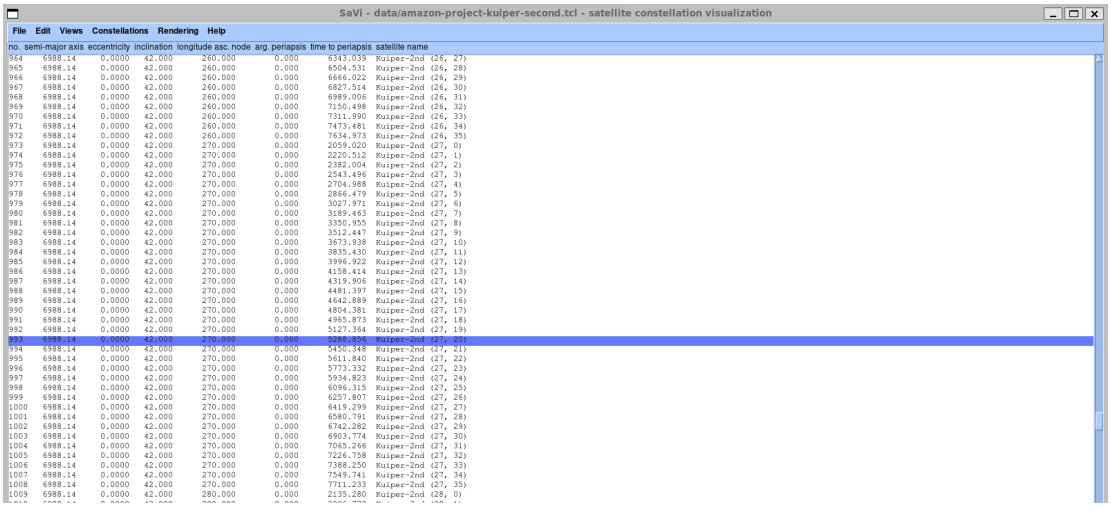
This is based on the 4 July 2019 FCC filing. This is at Ka-band - even though the name may suggest 'Ku'.

Despite the number of satellites, the filing does not appear to describe intersatellite links. The Technical Appendix says: "The number of United States gateways sites will be approximately equal to the number of active satellites serving U.S. territory."

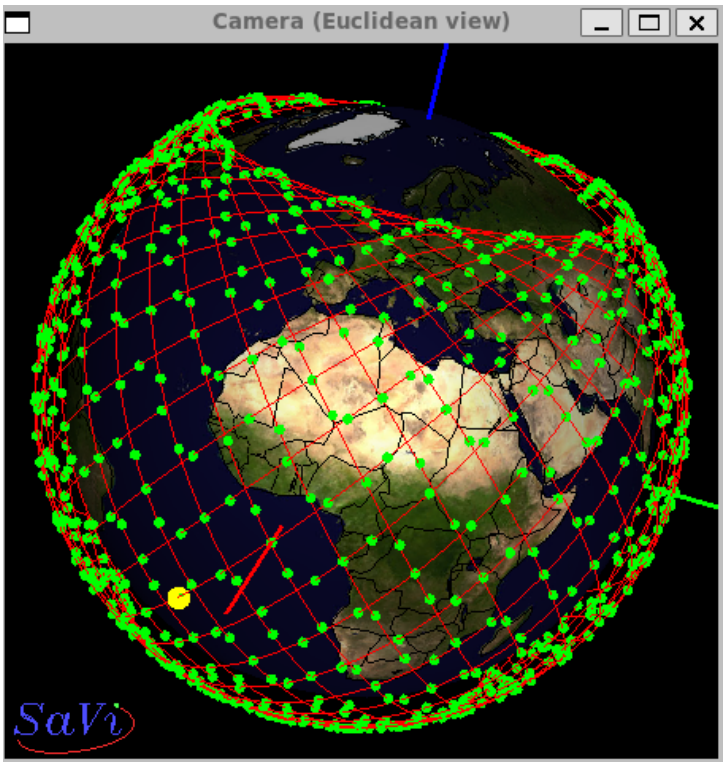
Details of the Project Kuiper Constellation



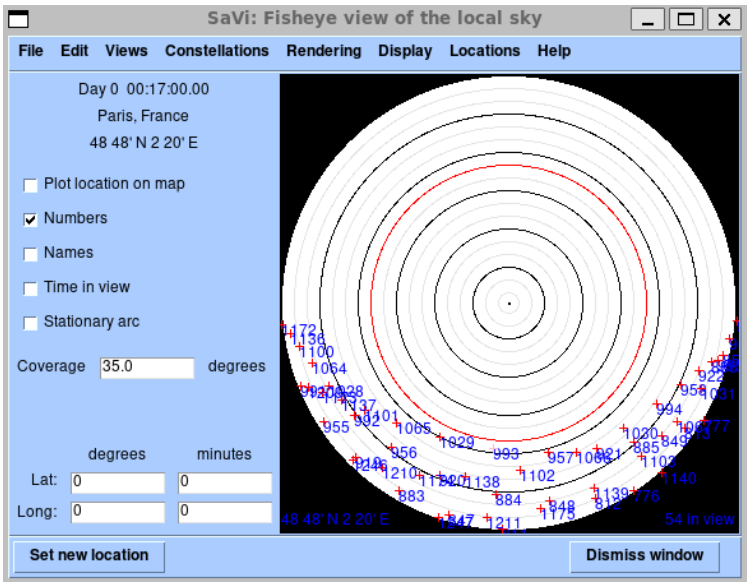
Coverage area of the Project Kuiper Constellation



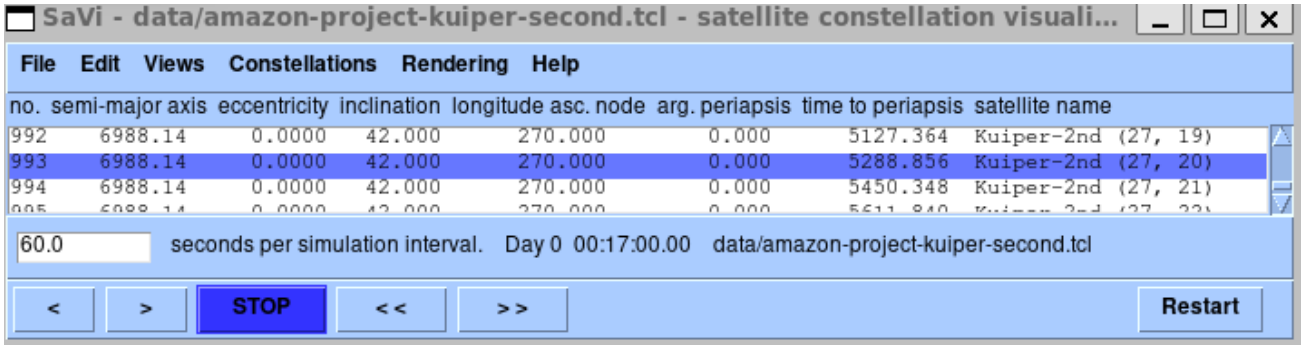
Select a satellite from the constellation - Satellite 993



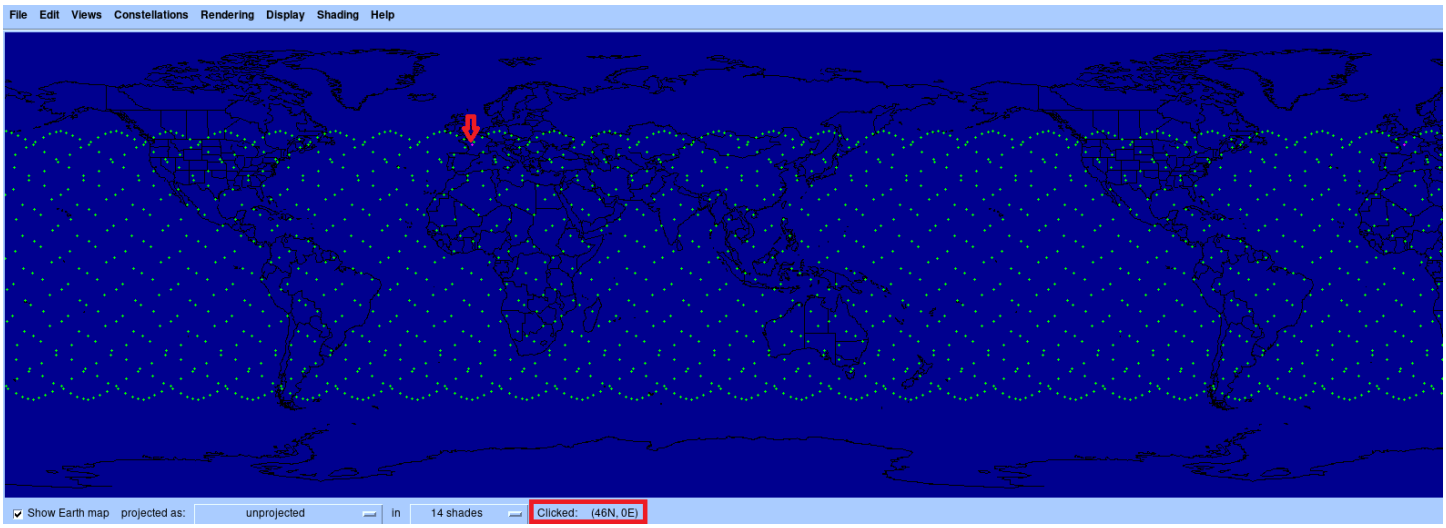
The selected satellite - Satellite 993



The selected Satellite reaches Paris Area (Satellite 993)



The timeslot for a satellite from the Project Kuiper Constellation to reach Paris is about: 00:17:00.00 minutes.



The position of the selected satellite is: (Latitude: 46N, Longitude: 0E)

Task 2: Input locations, create grid with neighboring satellites for 3 constellations, compute distances.

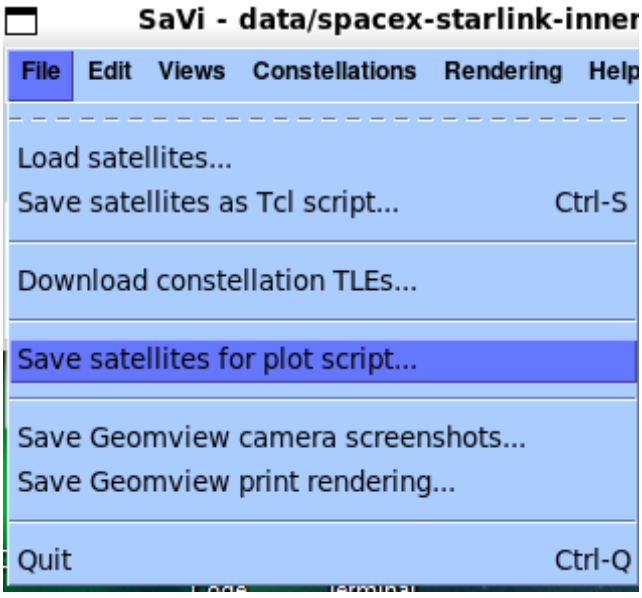
a) Starlink Constellation:

- First, save all the satellites at the specific timeslot, when satellite 1 is just above Paris, at 3:29:00 minutes.

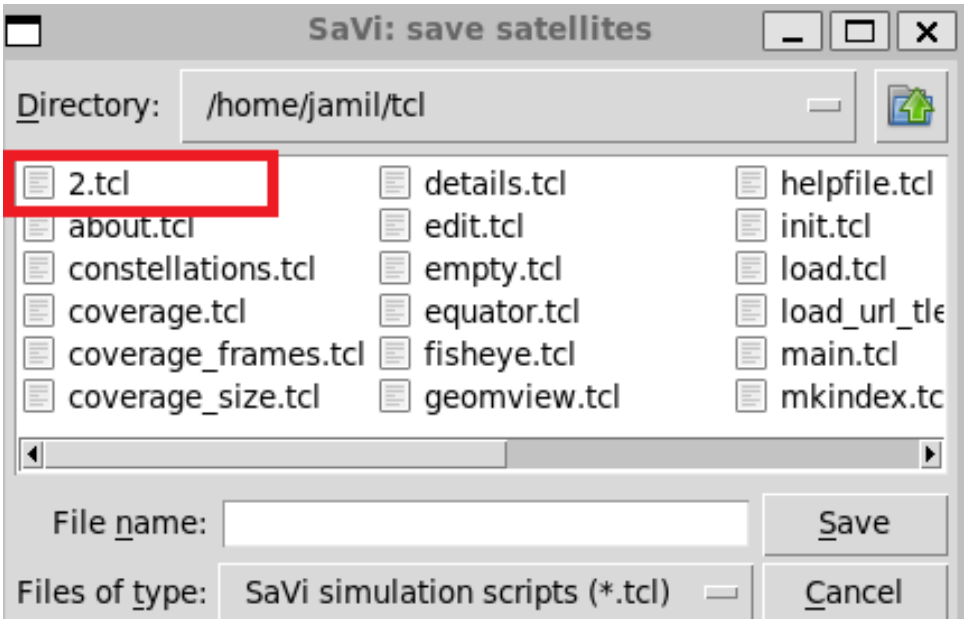
File Edit Views Constellations Rendering Help

no.	semi-major axis	eccentricity	inclination	longitude asc. node	arg. periapsis	time to periapsis	satellite name
1	6928.14	0.0000	53.000	0.000	0.000	0.000	Starlink-first (0, 0)
2	6928.14	0.0000	53.000	0.000	0.000	260.863	Starlink-first (0, 1)
3	6928.14	0.0000	53.000	0.000	0.000	521.726	Starlink-first (0, 2)
4	6928.14	0.0000	53.000	0.000	0.000	782.590	Starlink-first (0, 3)
5	6928.14	0.0000	53.000	0.000	0.000	1043.453	Starlink-first (0, 4)
6	6928.14	0.0000	53.000	0.000	0.000	1304.316	Starlink-first (0, 5)
7	6928.14	0.0000	53.000	0.000	0.000	1565.179	Starlink-first (0, 6)
8	6928.14	0.0000	53.000	0.000	0.000	1826.043	Starlink-first (0, 7)
9	6928.14	0.0000	53.000	0.000	0.000	2086.906	Starlink-first (0, 8)
10	6928.14	0.0000	53.000	0.000	0.000	2347.769	Starlink-first (0, 9)
11	6928.14	0.0000	53.000	0.000	0.000	2608.632	Starlink-first (0, 10)
12	6928.14	0.0000	53.000	0.000	0.000	2869.496	Starlink-first (0, 11)
13	6928.14	0.0000	53.000	0.000	0.000	3130.359	Starlink-first (0, 12)
14	6928.14	0.0000	53.000	0.000	0.000	3391.222	Starlink-first (0, 13)
15	6928.14	0.0000	53.000	0.000	0.000	3652.085	Starlink-first (0, 14)
16	6928.14	0.0000	53.000	0.000	0.000	3912.948	Starlink-first (0, 15)
17	6928.14	0.0000	53.000	0.000	0.000	4173.812	Starlink-first (0, 16)
18	6928.14	0.0000	53.000	0.000	0.000	4434.675	Starlink-first (0, 17)
19	6928.14	0.0000	53.000	0.000	0.000	4695.538	Starlink-first (0, 18)
20	6928.14	0.0000	53.000	0.000	0.000	4956.401	Starlink-first (0, 19)
21	6928.14	0.0000	53.000	0.000	0.000	5217.265	Starlink-first (0, 20)
22	6928.14	0.0000	53.000	0.000	0.000	5478.128	Starlink-first (0, 21)
23	6928.14	0.0000	53.000	5.000	0.000	-47.100	Starlink-first (1, 0)
24	6928.14	0.0000	53.000	5.000	0.000	213.763	Starlink-first (1, 1)
25	6928.14	0.0000	53.000	5.000	0.000	474.626	Starlink-first (1, 2)
26	6928.14	0.0000	53.000	5.000	0.000	735.489	Starlink-first (1, 3)
27	6928.14	0.0000	53.000	5.000	0.000	996.353	Starlink-first (1, 4)
28	6928.14	0.0000	53.000	5.000	0.000	1257.216	Starlink-first (1, 5)
29	6928.14	0.0000	53.000	5.000	0.000	1518.079	Starlink-first (1, 6)
30	6928.14	0.0000	53.000	5.000	0.000	1778.942	Starlink-first (1, 7)
31	6928.14	0.0000	53.000	5.000	0.000	2039.806	Starlink-first (1, 8)
32	6928.14	0.0000	53.000	5.000	0.000	2300.669	Starlink-first (1, 9)
33	6928.14	0.0000	53.000	5.000	0.000	2561.532	Starlink-first (1, 10)
34	6928.14	0.0000	53.000	5.000	0.000	2822.395	Starlink-first (1, 11)
35	6928.14	0.0000	53.000	5.000	0.000	3083.258	Starlink-first (1, 12)
36	6928.14	0.0000	53.000	5.000	0.000	3344.122	Starlink-first (1, 13)
37	6928.14	0.0000	53.000	5.000	0.000	3604.985	Starlink-first (1, 14)
38	6928.14	0.0000	53.000	.000	0.000	3865.848	Starlink-first (1, 15)
39	6928.14	0.0000	53.000	.000	0.000	4126.711	Starlink-first (1, 16)
40	6928.14	0.0000	53.000	5.000	0.000	4387.575	Starlink-first (1, 17)
41	6928.14	0.0000	53.000	5.000	0.000	4648.438	Starlink-first (1, 18)
42	6928.14	0.0000	53.000	5.000	0.000	4909.301	Starlink-first (1, 19)
43	6928.14	0.0000	53.000	5.000	0.000	5170.164	Starlink-first (1, 20)
44	6928.14	0.0000	53.000	5.000	0.000	5431.028	Starlink-first (1, 21)
45	6928.14	0.0000	53.000	10.000	0.000	-94.201	Starlink-first (2, 0)
46	6928.14	0.0000	53.000	10.000	0.000	166.663	Starlink-first (2, 1)
47	6928.14	0.0000	53.000	10.000	0.000	427.526	Starlink-first (2, 2)
48	6928.14	0.0000	53.000	10.000	0.000	688.389	Starlink-first (2, 3)
49	6928.14	0.0000	53.000	10.000	0.000	949.252	Starlink-first (2, 4)
50	6928.14	0.0000	53.000	10.000	0.000	1210.116	Starlink-first (2, 5)

List of all satellites (1584 satellite)



Save Satellites for plot script



The following file is the tcl file (contains the information of all satellites)

- Check the content of that file:

```
jamil@Jamil-AbouLtaif:/mnt/c/Users/dell/downloads/savi1.6.0$ cd /home/jamil/tcl
jamil@Jamil-AbouLtaif:~/tcl$ cat 2.tcl
```

```
Starlink-first (0, 0)   47.367918       1.896666
Starlink-first (0, 1)   38.197961       -16.678300
Starlink-first (0, 2)   26.804355       -30.666331
Starlink-first (0, 3)   14.298724       -41.971518
Starlink-first (0, 4)   1.317967        -52.051199
Starlink-first (0, 5)   -11.703034      -62.024901
Starlink-first (0, 6)   -24.346023      -72.980668
Starlink-first (0, 7)   -36.033328      -86.285391
Starlink-first (0, 8)   -45.775151      -103.779361
Starlink-first (0, 9)   -51.897239      -126.979447
Starlink-first (0, 10)  -52.509287      -153.816061
Starlink-first (0, 11)  -47.367918      -178.103334
Starlink-first (0, 12)  -38.197961      163.321700
Starlink-first (0, 13)  -26.804355      149.333669
Starlink-first (0, 14)  -14.298724      138.028482
Starlink-first (0, 15)  -1.317967       127.948801
Starlink-first (0, 16)  11.703034       117.975099
Starlink-first (0, 17)  24.346023       107.019332
Starlink-first (0, 18)  36.033328       93.714609
Starlink-first (0, 19)  45.775151       76.220639
Starlink-first (0, 20)  51.897239       53.020553
Starlink-first (0, 21)  52.509287       26.183939
Starlink-first (1, 0)   48.655812       10.872336
Starlink-first (1, 1)   40.061416       -8.722090
Starlink-first (1, 2)   28.967965       -23.389722
Starlink-first (1, 3)   16.606430       -35.057211
Starlink-first (1, 4)   3.676202        -45.269474
```

The Spherical parameters of 1584 satellites

- Next, Open Spider to run the code and build the grid.

The code is divided into several steps:

Step 1: Read the parameters of the satellites in “StarLink constellation” at the specific timeslot.

```
"""Libraries """
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import networkx as nx
from datetime import datetime, timedelta
import pytz
```

Used Libraries

```
"""Read the Starlink Satellites informations: Latitude and Longitude"""

# Replace 'your_file_path.xlsx' with the path to your Excel file
file_path = 'C:/Users/dell/Downloads/Starlink (1).xlsx'
df = pd.read_excel(file_path)
```

Read from an excel file

Step 2: Convert Spherical parameters to cartesian coordinates (x,y,z).

```
"""Function 1: Spherical to Cartesian Function"""

def spherical_to_cartesian(longitude, latitude, altitude):
    """
    Convert spherical coordinates (longitude, latitude, altitude) to Cartesian coordinates (x, y, z).
    """
    # Convert degrees to radians
    lon_rad = np.radians(longitude)
    lat_rad = np.radians(latitude)

    # Radius of the Earth
    R = 6371  # Assuming the Earth's radius in kilometers
    # Convert spherical coordinates to Cartesian coordinates
    x = (R  + altitude) * np.sin(lat_rad) * np.cos(lon_rad)
    y = (R  +  altitude) * np.sin(lat_rad) * np.sin(lon_rad)
    z = (R  +  altitude) * np.cos(lat_rad)

    return x, y, z
```

Convert the Spherical parameters to cartesian coordinates

Step 3: Once we have the spherical coordinates (longitude, latitude, altitude) for each satellite, we can compute the distances between all pairs of satellites using a distance formula appropriate for Cartesian coordinates.

One commonly used formula for computing straight-line distances is the Euclidian formula, which calculates the distance between two points given their cartesian coordinates.

```python
"""Function 2: Compute Euclidean distance Function"""

def euclidean_distance_3d(x1, y1, z1, x2, y2, z2):
    """
    Calculate the Euclidean distance between two points in 3D space.
    """
    return np.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2 + (z2 - z1) ** 2)
```

Calculate the Euclidean distance between any 2 points

```python
"""Search the neighboring satellites"""
def find_neighbors(df):
    neighbors = []

    for orbit in range(72):  # 72 orbits numbered from 0 to 71
        orbit_data = df[(df['Satellite-number'] > orbit * 22) & (df['Satellite-number'] <= (orbit + 1) * 22)]  # Satellites on the current orbit

        for i in range(len(orbit_data)):
            # Get indices of neighboring satellites within the same orbit
            prev_index = (i - 1) % len(orbit_data)
            next_index = (i + 1) % len(orbit_data)

            # Get Cartesian coordinates for current and neighboring satellites
            x_current, y_current, z_current = orbit_data.iloc[i]['X'], orbit_data.iloc[i]['Y'], orbit_data.iloc[i]['Z']
            x_prev, y_prev, z_prev = orbit_data.iloc[prev_index]['X'], orbit_data.iloc[prev_index]['Y'], orbit_data.iloc[prev_index]['Z']
            x_next, y_next, z_next = orbit_data.iloc[next_index]['X'], orbit_data.iloc[next_index]['Y'], orbit_data.iloc[next_index]['Z']

            # Compute distances between current satellite and neighboring satellites within the same orbit
            distance_to_prev = euclidean_distance_3d(x_current, y_current, z_current, x_prev, y_prev, z_prev)
            distance_to_next = euclidean_distance_3d(x_current, y_current, z_current, x_next, y_next, z_next)

            # Add neighboring satellites and their distances to the list
            neighbors.append({'Satellite': int(orbit_data.iloc[i]['Satellite-number']),
                              'Prev_Satellite': int(orbit_data.iloc[prev_index]['Satellite-number']),
                              'Next_Satellite': int(orbit_data.iloc[next_index]['Satellite-number']),
                              'Distance_to_prev': distance_to_prev,
                              'Distance_to_next': distance_to_next,
                              'Orbit': orbit})

    return pd.DataFrame(neighbors)
```

Searching for neighboring satellites

- [ ] **Let's take an example to check if our work till this point is well performed!**

The output:

Satellite	Prev_Satellite	Next_Satellite	Distance_to_prev	Distance_to_next	Orbit
1	22	2	1969.921938	1969.921999	0
2	1	3	1969.921999	1969.922060	0
3	2	4	1969.922060	1969.921936	0
4	3	5	1969.921936	1969.921983	0
5	4	6	1969.921983	1969.921990	0
6	5	7	1969.921990	1969.922019	0
7	6	8	1969.922019	1969.922046	0
8	7	9	1969.922046	1969.921965	0
9	8	10	1969.921965	1969.921978	0
10	9	11	1969.921978	1969.921992	0
11	10	12	1969.921992	1969.921938	0
12	11	13	1969.921938	1969.921999	0
13	12	14	1969.921999	1969.922060	0
14	13	15	1969.922060	1969.921936	0
15	14	16	1969.921936	1969.921983	0
16	15	17	1969.921983	1969.921990	0
17	16	18	1969.921990	1969.922019	0
18	17	19	1969.922019	1969.922046	0
19	18	20	1969.922046	1969.921965	0
20	19	21	1969.921965	1969.921978	0
21	20	22	1969.921978	1969.921992	0
22	21	1	1969.921992	1969.921938	0
23	44	24	1969.922012	1969.922033	1
24	23	25	1969.922033	1969.921954	1
25	24	26	1969.921954	1969.921950	1
26	25	27	1969.921950	1969.921978	1
27	26	28	1969.921978	1969.922062	1
28	27	29	1969.922062	1969.922048	1
29	28	30	1969.922048	1969.921893	1
30	29	31	1969.921893	1969.922079	1
31	30	32	1969.922079	1969.921944	1
32	31	33	1969.921944	1969.921953	1

Neighboring Satellites for each satellite within the same orbit

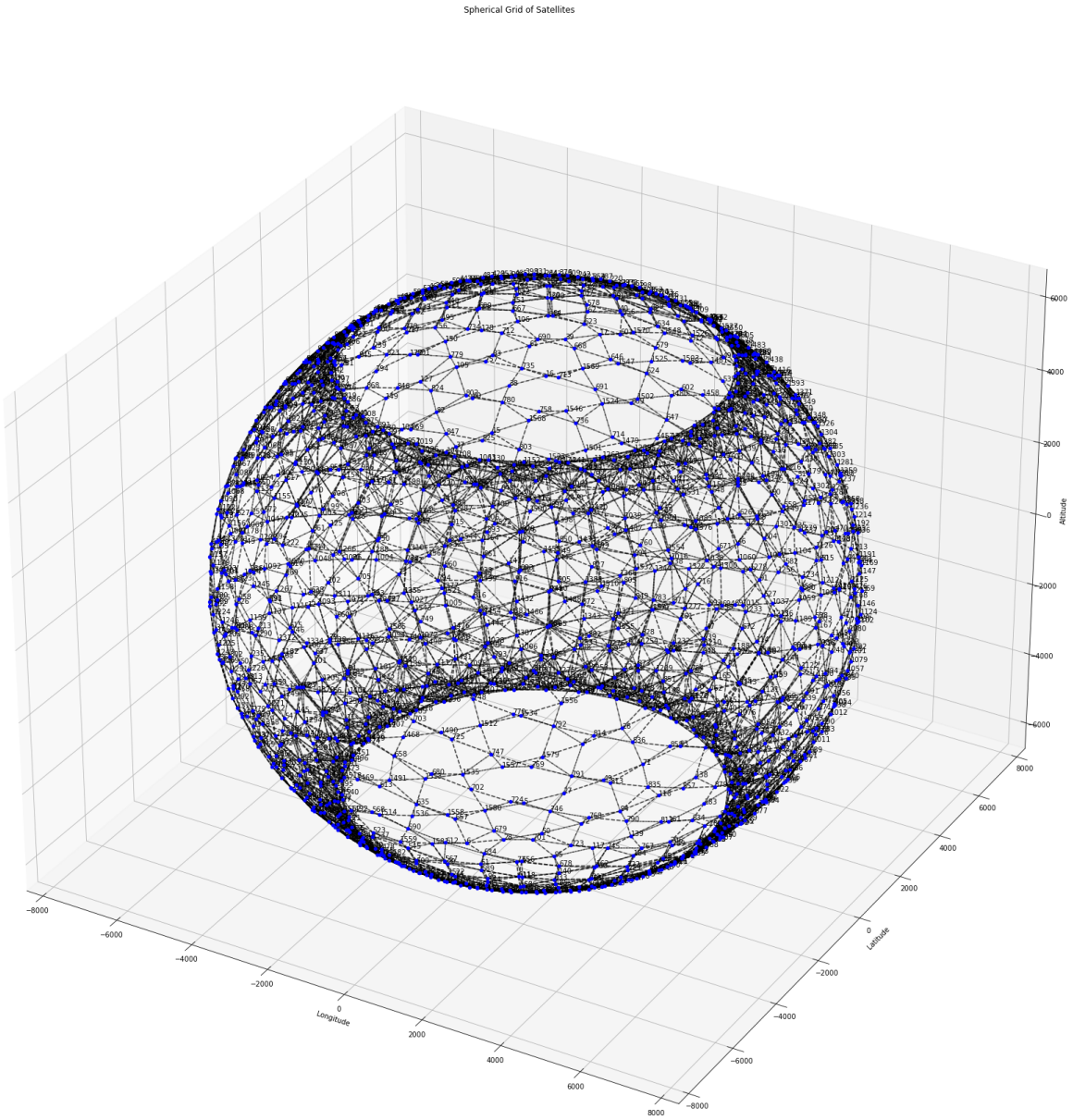
Step 4: Build a grid using the distances between satellites.

We can represent the satellites as nodes and the distances between them as edges in a graph. We can then use graph algorithms to construct the grid based on these distances.

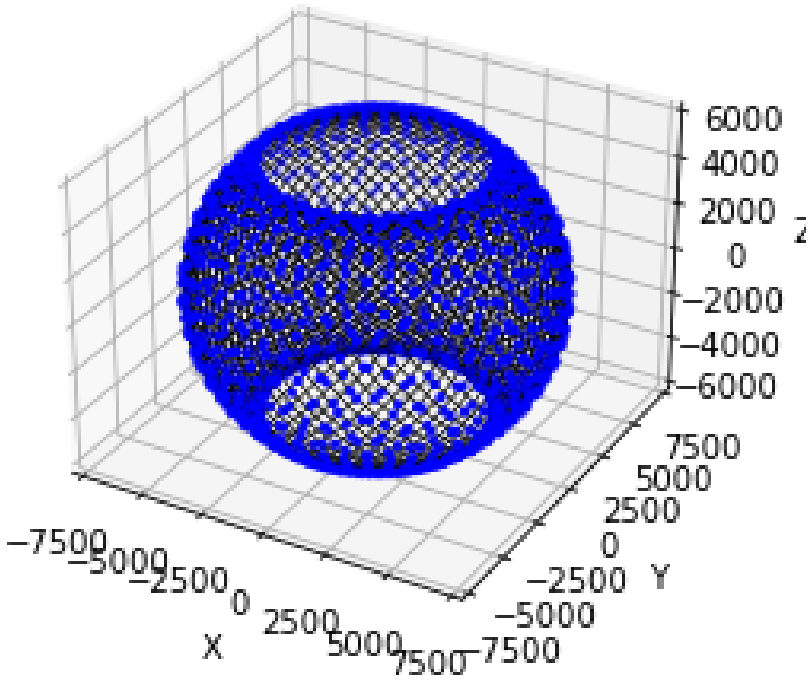
```
def build_3d_spherical_fully_connected_grid(df):
    connections = []

    # Iterate over neighboring satellites
    for index, row in df.iterrows():
        # Add connection to previous satellite
        if not pd.isnull(row['Prev_Satellite']):
            connections.append({'Satellite1': int(row['Satellite']),
                                'Satellite2': int(row['Prev_Satellite']),
                                'Distance': row['Distance_to_prev'],
                                'Orbit': int(row['Orbit'])})
        # Add connection to next satellite
        if not pd.isnull(row['Next_Satellite']):
            connections.append({'Satellite1': int(row['Satellite']),
                                'Satellite2': int(row['Next_Satellite']),
                                'Distance': row['Distance_to_next'],
                                'Orbit': int(row['Orbit'])})

    return pd.DataFrame(connections)

# Build the 3D spherical fully connected grid
fully_connected_grid = build_3d_spherical_fully_connected_grid(neighbor_distances)

# Plot the grid
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Plot nodes
ax.scatter(df['X'], df['Y'], df['Z'], color='b', marker='o', s=5)

# Plot edges
for index, row in fully_connected_grid.iterrows():
    x_values = [df.loc[df['Satellite-number'] == row['Satellite1'], 'X'].values[0],
                df.loc[df['Satellite-number'] == row['Satellite2'], 'X'].values[0]]
    y_values = [df.loc[df['Satellite-number'] == row['Satellite1'], 'Y'].values[0],
                df.loc[df['Satellite-number'] == row['Satellite2'], 'Y'].values[0]]
    z_values = [df.loc[df['Satellite-number'] == row['Satellite1'], 'Z'].values[0],
                df.loc[df['Satellite-number'] == row['Satellite2'], 'Z'].values[0]]
    ax.plot(x_values, y_values, z_values, color='k', linewidth=0.4)  # black lines (color='k'), smaller thickness (linewidth=0.5)

# Set labels
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
```

☐ Let’s build the grid for StarLink Constellation (1584 Satellites)

The output:



Star Link Constellation’s grid



Task 3:

I- Find the closest satellite in the constellation closest to Los Angeles:

To find the closest satellite to Los Angeles, there are several steps:

- Define Los Angeles Coordinates (Latitude = 34.0522-degree, Longitude = -118.243-degree)
- Convert Los Angeles' coordinates to Cartesian.
- Compute the Euclidean distance between Los Angeles and all the satellites in the grid.
- Choose the satellite with the minimum distance to Los Angeles.
- Extract the information about this closest satellite.

```
"""Find the closest satellite to Los Angeles"""
LA_latitude = 34.052235
LA_longitude = -118.243683

# Convert Los Angeles coordinates to Cartesian
LA_x, LA_y, LA_z = spherical_to_cartesian(LA_latitude, LA_longitude, 550)

# Calculate distances between Los Angeles and each satellite
distances_to_LA = np.sqrt((df['X'] - LA_x) ** 2 + (df['Y'] - LA_y) ** 2 + (df['Z'] - LA_z) ** 2)

# Find the index of the satellite with the minimum distance to Los Angeles
closest_satellite_index = distances_to_LA.idxmin()

# Get information about the closest satellite
closest_satellite_info = df.loc[closest_satellite_index]

print("Closest Satellite to Los Angeles:")
print("Satellite Number:", int(closest_satellite_info['Satellite-number']))
print("Distance to Los Angeles:", distances_to_LA.min(), "kilometers")
```

Find the closest satellite to LA

The output:

```
Closest Satellite to Los Angeles:
Satellite Number: 1178
Distance to Los Angeles: 69.2161594385602 kilometers
```

Closest satellite to Los Angeles

II- Find the closest satellite in the constellation closest to Moscow:

The output:

```
Closest Satellite to Moscow:
Satellite Number: 44
Distance to Moscow: 368.18574151630014 kilometers
```

Closest satellite to Moscow

Task 4:

I- Use Dijkstra's algorithm to find the shortest path between the satellite over Paris (Satellite 1) and over Los Angeles (Satellite 1178).

Dijkstra's algorithm is a fundamental algorithm used to find the shortest paths between nodes in a graph, where each edge has a non-negative weight.

Now, applying Dijkstra's algorithm to find the shortest path between the satellite over Paris (Satellite 1) and over Los Angeles (Satellite 1178) involves representing the satellites and their connections as a graph. Each satellite represents a node, and the connections between satellites represent edges with associated weights (possibly distances or costs).

Here's an outline of how we use Dijkstra's algorithm for this specific case:

Create Graph: Represent the satellites and their connections as a graph, where the nodes are the satellites, and the edges represent connections between satellites.

Run Dijkstra's Algorithm: Run Dijkstra's algorithm with the source vertex being the satellite over Paris (Satellite 1). This will calculate the shortest path from Satellite 1 to all other satellites in the graph.

Retrieve Shortest Path: Once Dijkstra's algorithm has been run, we can retrieve the shortest path from Satellite 1 to the satellite over Los Angeles (Satellite 1178). This path represents the shortest route between these two satellites.

```
"""Apply Dijkstra and compute the distance from the satellite 1 to the closest satellite to Los Angeles"""
G = nx.Graph()

# Calculate Euclidean distances and add edges to the graph
for i in range(len(df)):
    for j in range(i + 1, len(df)):  # Start from i + 1 to avoid duplicate edges
        distance = euclidean_distance_3d(df.at[i, 'X'], df.at[i, 'Y'], df.at[i, 'Z'],
                                         df.at[j, 'X'], df.at[j, 'Y'], df.at[j, 'Z'])
        # If the distance is within a certain threshold, consider them neighbors
        # Adjust this threshold according to your needs
        if distance < 2000:  # Assuming 2000 km as the threshold
            G.add_edge(df.at[i, 'Satellite-number'], df.at[j, 'Satellite-number'], weight=distance)
```

Build the graph to find the shortest path

```
# Apply Dijkstra's Algorithm
source_node = 1  # Satellite 1 (Paris)
target_node = 1178
shortest_path = nx.dijkstra_path(G, source=source_node, target=target_node, weight='weight')
shortest_path_distance = nx.dijkstra_path_length(G, source=source_node, target=target_node, weight='weight')

print("Shortest path:", shortest_path)
print("Shortest path distance (in kilometers):", shortest_path_distance)

# Print the latitude and longitude of each satellite in the shortest path
print("Satellite coordinates in the shortest path:")
for satellite_number in shortest_path:
    latitude = df.at[satellite_number - 1, 'Latitude']  # Adjust for 0-based indexing
    longitude = df.at[satellite_number - 1, 'Longitude']  # Adjust for 0-based indexing
    print(f"Satellite {satellite_number}: Latitude: {latitude}, Longitude: {longitude}")
```

Calculate the shortest path

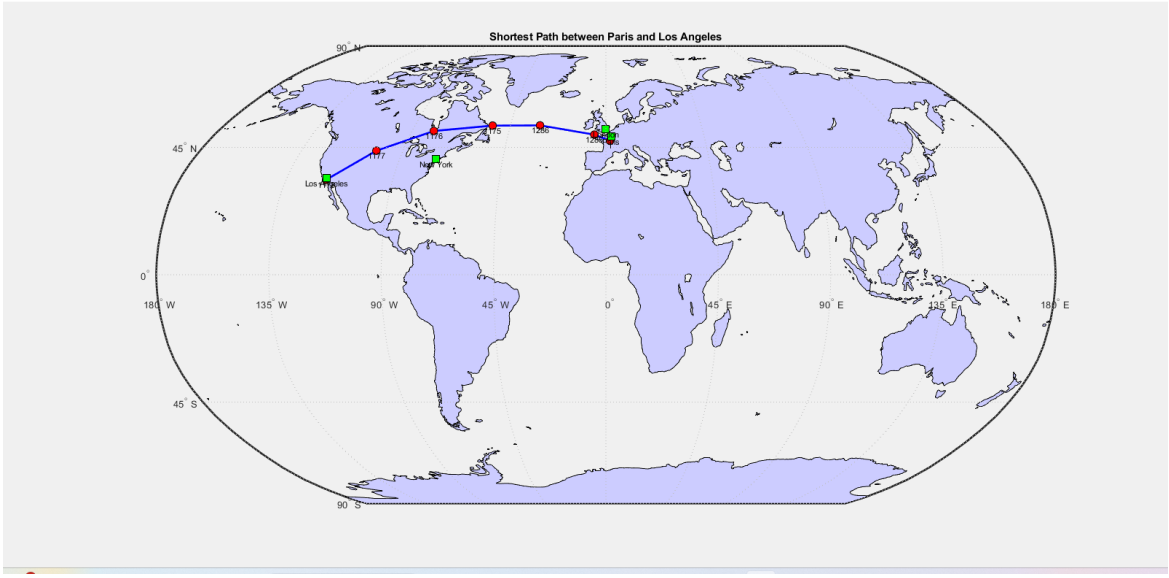
The output: The shortest path from Paris to Los Angeles

```
Shortest path: [1, 1285, 1286, 1175, 1176, 1177, 1178]
Shortest path distance (in kilometers): 10132.815335920992
Satellite coordinates in the shortest path:
Satellite 1: Latitude: 47.367918, Longitude: 1.896666
Satellite 1285: Latitude: 49.589776, Longitude: -5.308779
Satellite 1286: Latitude: 52.983444, Longitude: -31.058085
Satellite 1175: Latitude: 52.910178, Longitude: -53.42047
Satellite 1176: Latitude: 50.924695, Longitude: -79.90941
Satellite 1177: Latitude: 43.792809, Longitude: -101.787387
Satellite 1178: Latitude: 33.485162, Longitude: -118.144703
```

List of satellites in the shortest path list

```
Satellite 1 (Latitude: 47.367918, Longitude: 1.896666): Nearest city is Paris, France.
Satellite 1285 (Latitude: 49.589776, Longitude: -5.308779): Nearest city is Brest, France.
Satellite 1286 (Latitude: 52.983444, Longitude: -31.058085): No major city nearby; it's located in the Atlantic Ocean.
Satellite 1175 (Latitude: 52.910178, Longitude: -53.42047), the nearest major city is St. John's, Newfoundland and Labrador, Canada.
Satellite 1176 (Latitude: 50.924695, Longitude: -79.90941): Nearest city is Rouyn-Noranda, Quebec, Canada.
Satellite 1177 (Latitude: 43.792809, Longitude: -101.787387): Nearest city is Rapid City, South Dakota, United States.
Satellite 1178 (Latitude: 33.485162, Longitude: -118.144703): Nearest city is Los Angeles, California, United States.
```

Locations of the satellites in the shortest path list



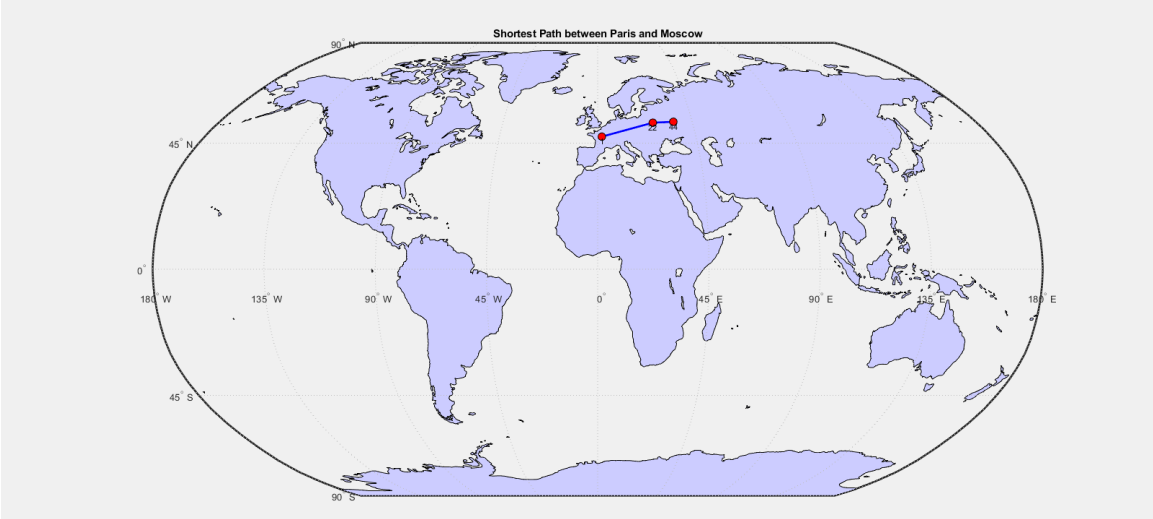
Shortest path between Paris and Los Angeles

II- Use Dijkstra's algorithm to find the shortest path between the satellite over Paris (Satellite 1) and over Moscow (Satellite 44).

The output: The shortest path from Paris to Moscow:

```
Shortest path: [1, 22, 44]
Shortest path distance (in kilometers): 2690.930238040792
Satellite coordinates in the shortest path:
Satellite 1: Latitude: 47.367918, Longitude: 1.896666
Satellite 22: Latitude: 52.509287, Longitude: 26.183939
Satellite 44: Latitude: 52.852134, Longitude: 36.0254
```

Shortest path between Paris and Moscow



Shortest path between Paris and Moscow

Task 5:

I- Compute the RTT between the start (Above Paris) and end satellite (Above Los Angeles).

First, compute the distance between Satellite 1 and Satellite 1178

Second, calculate the Round-Trip-Time: Where:

- distance is the physical distance between Satellite 1 and Satellite 1178.
- speed is the speed of transmission, typically measured in units like meters per second (In our case, the speed of light).

$$RTT = \frac{2 \times \text{distance}}{\text{speed}}$$

```
# Speed of light in km/s
SPEED_OF_LIGHT = 299792.458  # in km/s

"""Calculate the RTT"""
def compute_rtt(graph, source_node, target_node):
    # Find the shortest path
    shortest_path = nx.dijkstra_path(graph, source=source_node, target=target_node, weight='weight')

    # Initialize RTT
    rtt = 0

    # Calculate RTT
    for i in range(len(shortest_path) - 1):
        satellite1 = shortest_path[i]
        satellite2 = shortest_path[i + 1]
        distance = graph[satellite1][satellite2]['weight']
        # Add one-way time for each edge
        one_way_time = distance / SPEED_OF_LIGHT
        rtt += one_way_time

    # Double the total one-way time to get RTT
    rtt *= 2

    return rtt

# Compute RTT between satellite 1 and satellite 1178
rtt = compute_rtt(G, source_node, target_node)
print("Round-Trip Time (RTT):", rtt, "seconds")
```

- Add the time of signal traveling from the satellite to the ground station at both ends.

```
# Coordinates for Los Angeles
lat_los_angeles = 34.0522    # Latitude in degrees
lon_los_angeles = -118.2437  # Longitude in degrees
alt_los_angeles = 0          # Altitude in km

# Coordinates for Satellite 1178
lat_satellite1178 = 33.485162  # Latitude in degrees
lon_satellite1178 = -118.144703 # Longitude in degrees
alt_satellite1178 = 550        # Altitude in km

# Compute distance between Satellite 1178 and Los Angeles
distance_satellite1178_los_angeles = euclidean_distance_3d(lat_satellite1178, lon_satellite1178, alt_satellite1178,
                                                           lat_los_angeles, lon_los_angeles, alt_los_angeles)

# Compute one-way time
one_way_time = distance_satellite1178_los_angeles / SPEED_OF_LIGHT

# Compute RTT
rtt_satellite1178_los_angeles = 2 * one_way_time

print("Round-Trip Time (RTT) between Satellite 1178 and Los Angeles:", rtt_satellite1178_los_angeles, "seconds")


# Coordinates for Paris
lat_paris = 48.8566  # Latitude in degrees
lon_paris = 2.3522   # Longitude in degrees
alt_paris = 0        # Altitude in km

# Coordinates for Satellite 1
lat_satellite1 = 47.367918  # Latitude in degrees
lon_satellite1 = 1.896666    # Longitude in degrees
alt_satellite1 = 550         # Altitude in km

# Compute distance between Paris and Satellite 1
distance_paris_satellite1 = euclidean_distance_3d(lat_paris, lon_paris, alt_paris,
                                                  lat_satellite1, lon_satellite1, alt_satellite1)

# Compute one-way time
one_way_time = distance_paris_satellite1 / SPEED_OF_LIGHT

# Compute RTT
rtt_paris_satellite1 = 2 * one_way_time
```

The output:

```
Round-Trip Time (RTT) between Satellite 1178 and Los Angeles:
0.0036692070566450388 seconds
Round-Trip Time (RTT) between Paris and Satellite 1: 0.0036692197463211672 seconds
Round-Trip Time (RTT): 0.07493729438969512 seconds
```

Round-Trip Time (RTT): Paris - Los Angeles

II- Compute the RTT between the start (Above Paris) and end satellite (Above Moscow).

The output:

```
Round-Trip Time (RTT) between Satellite 44 and Moscow: 0.003669271555479599
seconds
Round-Trip Time (RTT) between Paris and Satellite 1: 0.0036692197463211672 seconds
Round-Trip Time (RTT): 0.02529044550366926 seconds
```

Round-Trip Time (RTT): Paris - Moscow

Task 6: Find the timestamp of each satellite in the time zone of France

Here's a breakdown of what we do for this task:

- It starts by retrieving a list of time to periapsis for each satellite, extracted from Savi as we have shown before. This time is likely in seconds.
- It then gets the current UTC time which is the reference time against which the time to periapsis will be calculated.
- Next, it converts each time to periapsis (which is in seconds) to a datetime object by adding it to the current UTC time. This step adjusts for the current time and calculates the time of periapsis passage for each satellite.
- After that, convert the UTC times of periapsis passage to France time (CET). This step accounts for the time zone difference between UTC and France.

```
"""Time zone: France"""

# Example list of time to periapsis in seconds for each satellite
time_to_periapsis_seconds = df['Time_to_periapsis'].tolist()

# Current UTC time
current_utc_time = datetime.utcnow()

# Convert time to periapsis to datetime objects and adjust for the current UTC time
satellite_time_of_periapsis_utc = [current_utc_time + timedelta(seconds=time) for time in time_to_periapsis_seconds]

# Convert UTC time to France time for each satellite
france_tz = pytz.timezone('Europe/Paris')
satellite_time_of_periapsis_france = [utc_time.astimezone(france_tz) for utc_time in satellite_time_of_periapsis_utc]

# Print the time of periapsis passage in France for each satellite
for i, france_time in enumerate(satellite_time_of_periapsis_france):
   print("Satellite", i+1, "Time of periapsis passage in France:", france_time.strftime("%Y-%m-%d %H:%M:%S %Z%z"))
```

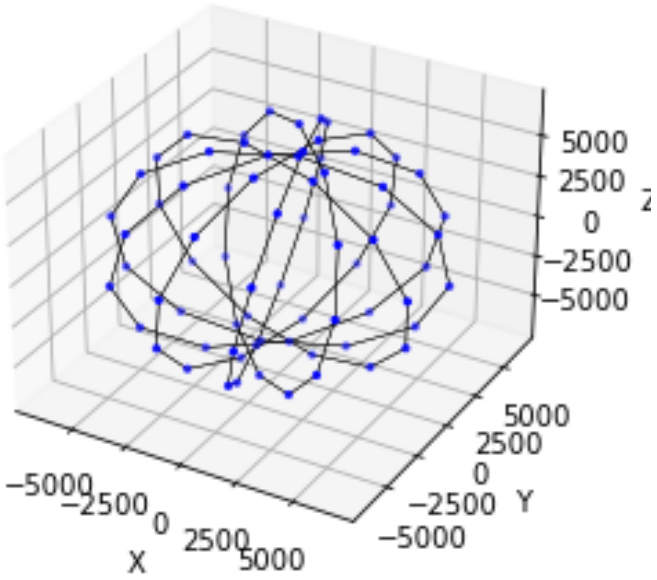
The output:

```
Satellite 1572 Time of periapsis passage in France: 2024-03-10 16:30:54 CET+0100
Satellite 1573 Time of periapsis passage in France: 2024-03-10 16:35:15 CET+0100
Satellite 1574 Time of periapsis passage in France: 2024-03-10 16:39:36 CET+0100
Satellite 1575 Time of periapsis passage in France: 2024-03-10 16:43:57 CET+0100
Satellite 1576 Time of periapsis passage in France: 2024-03-10 16:48:17 CET+0100
Satellite 1577 Time of periapsis passage in France: 2024-03-10 16:52:38 CET+0100
Satellite 1578 Time of periapsis passage in France: 2024-03-10 16:56:59 CET+0100
Satellite 1579 Time of periapsis passage in France: 2024-03-10 17:01:20 CET+0100
Satellite 1580 Time of periapsis passage in France: 2024-03-10 17:05:41 CET+0100
Satellite 1581 Time of periapsis passage in France: 2024-03-10 17:10:02 CET+0100
Satellite 1582 Time of periapsis passage in France: 2024-03-10 17:14:23 CET+0100
Satellite 1583 Time of periapsis passage in France: 2024-03-10 17:18:43 CET+0100
Satellite 1584 Time of periapsis passage in France: 2024-03-10 17:23:04 CET+0100
```

b) Telesat Constellation:

Telesat74 (Polar Telesat):

Task 2:



Polar Telesat grid

Task 3:

```
Closest Satellite to Los Angeles:
Satellite Number: 26
Distance to Los Angeles: 1760.000644492533 kilometers
```

Find the closest satellite to LA

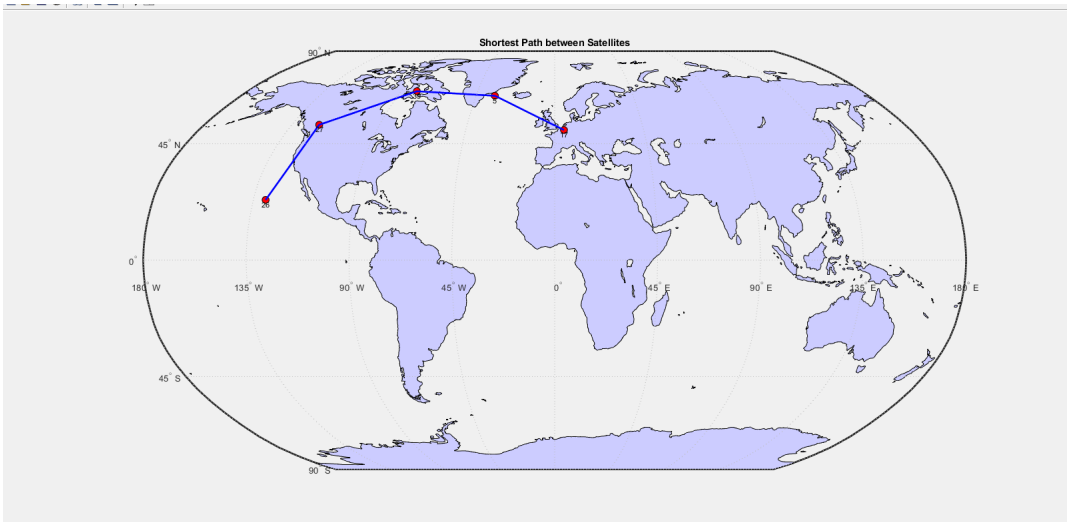
```
Closest Satellite to Moscow:
Satellite Number: 29
Distance to Los Angeles: 1285.977640140114 kilometers
```

Find the closest satellite to Moscow

Task 4:

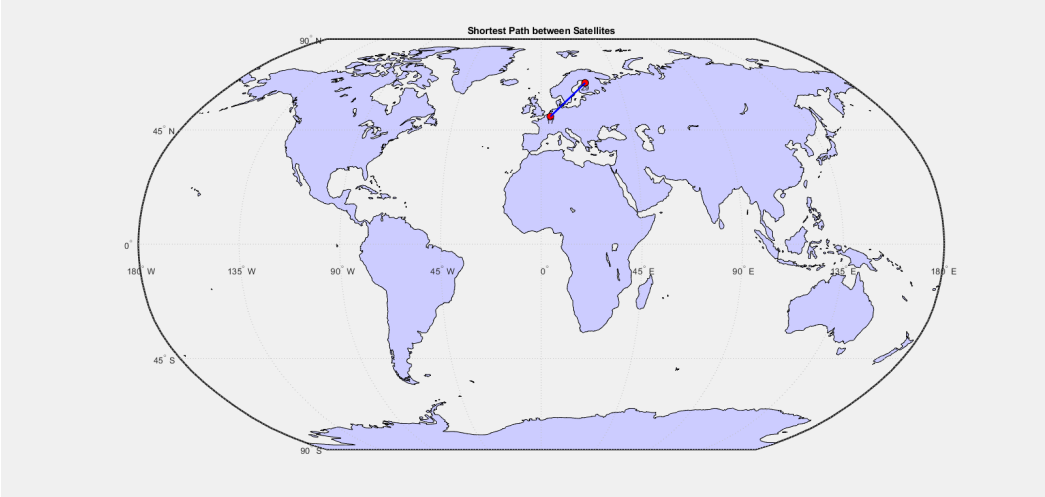
```
Shortest path: [17, 5, 39, 27, 26]
Shortest path distance (in kilometers): 11694.074439179756
Satellite coordinates in the shortest path:
Satellite 17: Latitude: 50.426371, Longitude: 4.696744
Satellite 5: Latitude: 64.661881, Longitude: -34.317558
Satellite 39: Latitude: 66.676565, Longitude: -80.78362
Satellite 27: Latitude: 52.546884, Longitude: -121.00339
Satellite 26: Latitude: 23.256768, Longitude: -129.497453
```

Shortest path from Paris to Los Angeles




```
Shortest path: [17, 29]
Shortest path distance (in kilometers): 2161.992367219343
Satellite coordinates in the shortest path:
Satellite 17: Latitude: 50.426371, Longitude: 4.696744
Satellite 29: Latitude: 64.661881, Longitude: 25.682442
```

Shortest path from Paris to Moscow



Task 5:

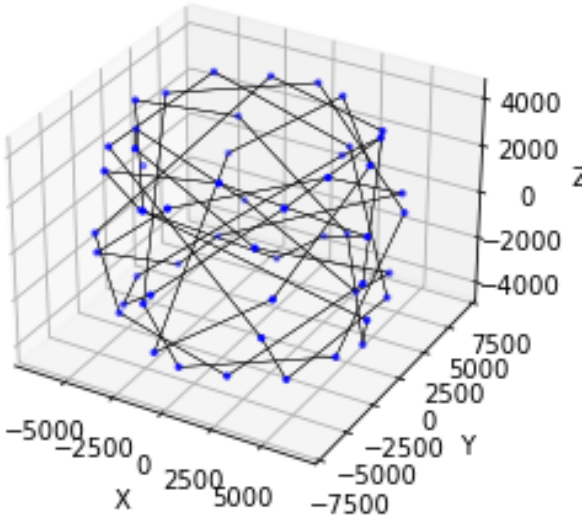
```
Round-Trip Time (RTT): 0.07801446718969665 seconds
```

Round-Trip Time (RTT, Paris-Los Angeles)

```
Round-Trip Time (RTT): 0.014423260555936622 seconds
```

Telesat54 (Inclined Telesat):

Task 2:



Inclined Telesat grid

Task 3:

```
Closest Satellite to Los Angeles:
Satellite Number: 27
Distance to Los Angeles: 698.3385841770886 kilometers
```

Find the closest satellite to LA

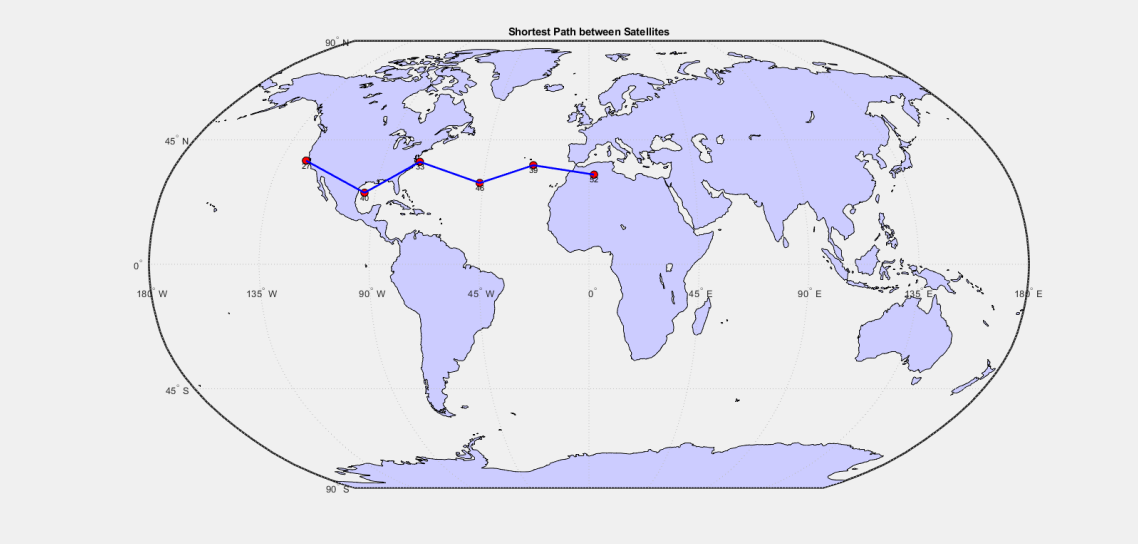
```
Closest Satellite to Moscow:
Satellite Number: 45
Distance to Los Angeles: 2831.5859748553767 kilometers
```

Find the closest satellite to Moscow

Task 4:

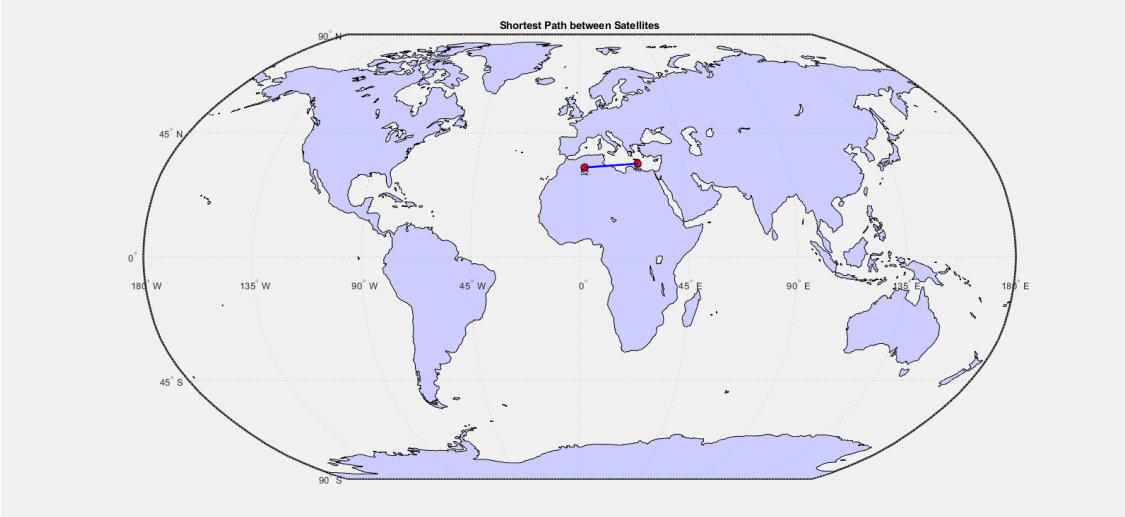
```
Satellite coordinates in the shortest path:
Satellite 52: Latitude: 32.366992, Longitude: 2.176065
Satellite 39: Latitude: 35.760233, Longitude: -24.199261
Satellite 46: Latitude: 29.353762, Longitude: -46.461264
Satellite 26: Latitude: 17.783212, Longitude: -58.622989
```

Shortest path from Paris to LA




```
Shortest path: [52, 45]
Shortest path distance (in kilometers): 2584.5331307682836
Satellite coordinates in the shortest path:
Satellite 52: Latitude: 32.366992, Longitude: 2.176065
Satellite 45: Latitude: 33.757217, Longitude: 25.232795
```

Shortest path from Paris to Moscow



Task 5:

```
Round-Trip Time (RTT): 0.08906108681497667 seconds
```

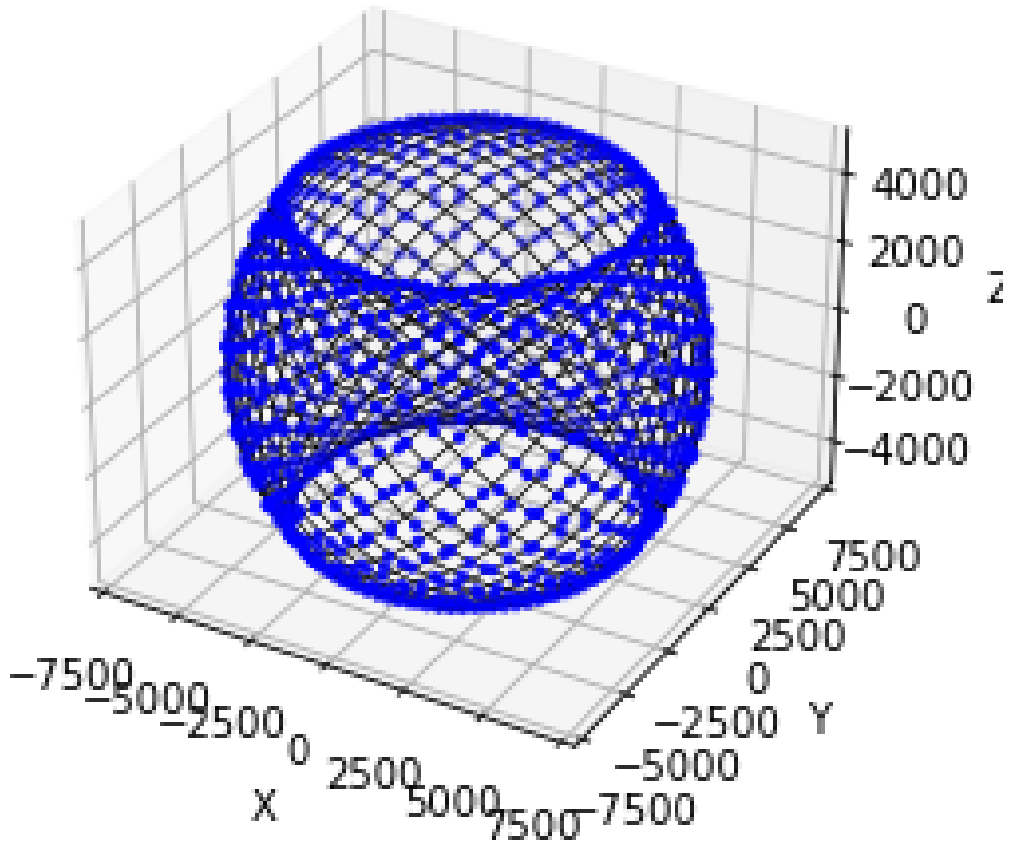
Round-Trip Time (RTT, Paris-LA)

```
Round-Trip Time (RTT): 0.017242175446764 98 seconds
```

Round-Trip Time (RTT, Paris-Moscow)

c) Project Kuiper Constellation:

Task 2:



Project Kuiper's Grid

Task 3:

```
Closest Satellite to Los Angeles:
Satellite Number: 712
Distance to Los Angeles: 337.175309076111 kilometers
```

Find the closest satellite to LA

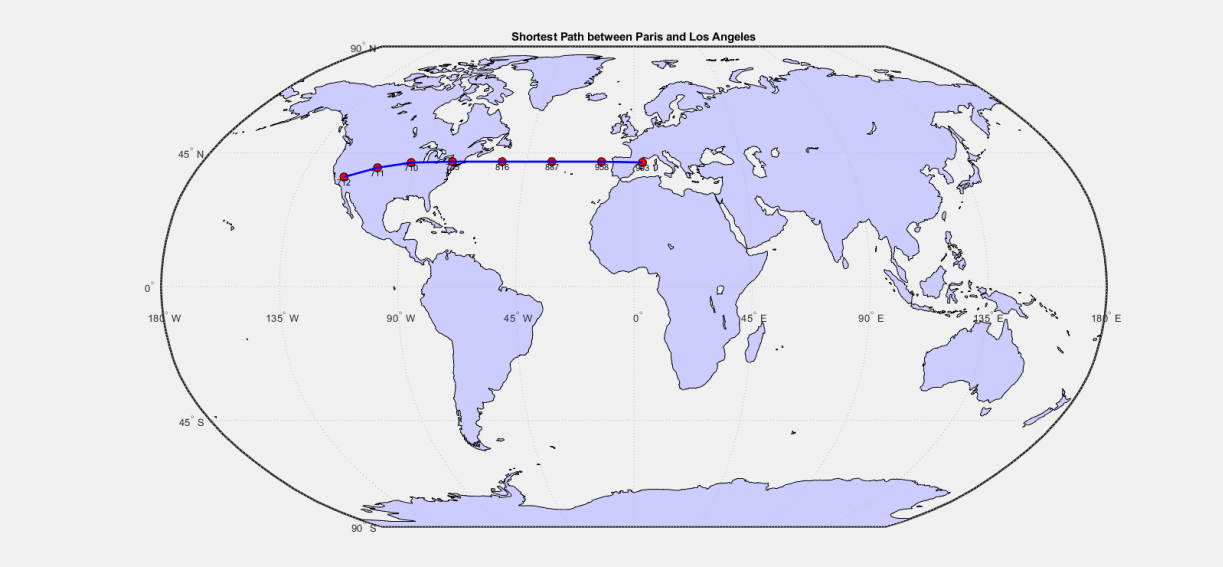
```
Closest Satellite to Moscow:
Satellite Number: 1172
Distance to Los Angeles: 1723.5667184400006 kilometers
```

Find the closest satellite to Moscow

Task 4:

```
Shortest path: [993, 958.0, 887.0, 816.0, 745.0, 710.0, 711.0, 712.0]
Shortest path distance (in kilometers): 11032.436929348927
Satellite coordinates in the shortest path:
Satellite 993: Latitude: 41.730744, Longitude: 3.534867
Satellite 958.0: Latitude: 41.997313, Longitude: -13.544945
Satellite 887.0: Latitude: 41.999993, Longitude: -34.292496
Satellite 816.0: Latitude: 41.997823, Longitude: -55.040054
Satellite 745.0: Latitude: 41.990804, Longitude: -75.787504
Satellite 710.0: Latitude: 41.683315, Longitude: -92.86134
Satellite 711.0: Latitude: 39.944049, Longitude: -105.886442
Satellite 712.0: Latitude: 36.838215, Longitude: -118.027084
```

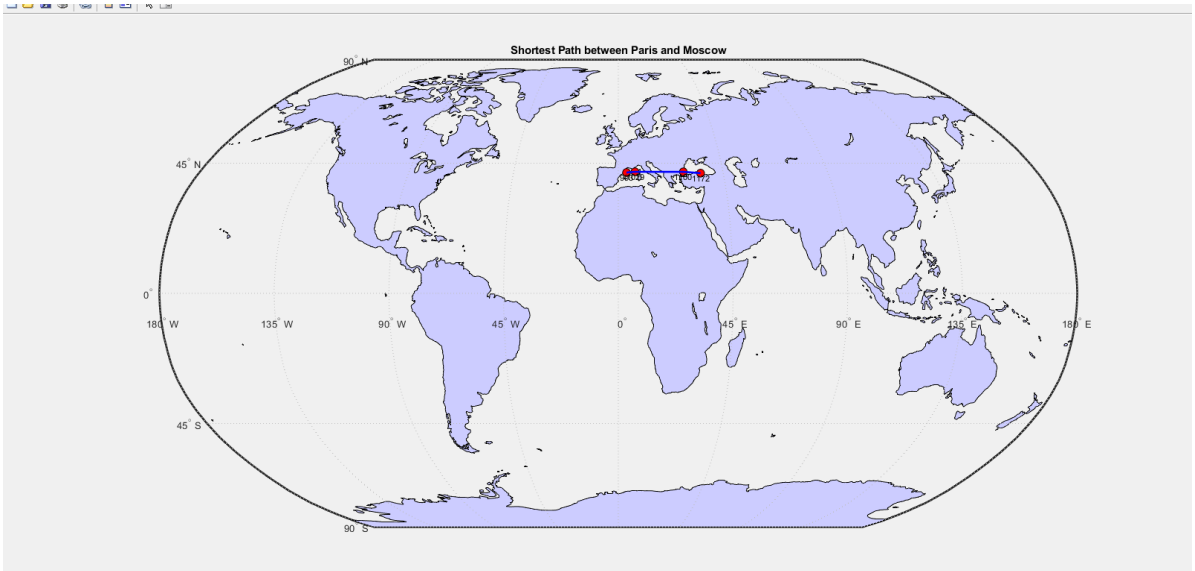
Shortest path from Paris to Los Angeles



Shortest path between Paris and Los Angeles

```
Shortest path: [993, 1029.0, 1100.0, 1172.0]
Shortest path distance (in kilometers): 2847.537399498334
Satellite coordinates in the shortest path:
Satellite 993: Latitude: 41.730744, Longitude: 3.534867
Satellite 1029.0: Latitude: 41.989783, Longitude: 7.202487
Satellite 1100.0: Latitude: 41.977407, Longitude: 27.949685
Satellite 1172.0: Latitude: 41.53068, Longitude: 35.292367
```

Shortest path from Paris to Moscow



Shortest path between Paris and Moscow

Task 5:

```
Round-Trip Time (RTT) between Satellite 712 and Los Angeles: 0.0040695246617835565
seconds
Round-Trip Time (RTT) between Paris and Satellite 993: 0.004069767266421898 seconds
Round-Trip Time (RTT): 0.08237785116294455 seconds
```

Round-Trip Time (RTT): Paris - Los Angeles

```
Round-Trip Time (RTT) between Satellite 1172 and Moscow: 0.0040706178858385695 seconds
Round-Trip Time (RTT) between Paris and Satellite 993: 0.004069767266421898 seconds
Round-Trip Time (RTT): 0.027301797563640325 seconds
```

Round-Trip Time (RTT): Paris - Moscow

5. Conclusion

Our focus on building a fully connected grid displaying Inter Satellite Links (ISL) for various constellations, utilizing data extracted from Savi, marks a significant step towards understanding and optimizing LEO satellites communication networks. As shown LEO satellite have better performance than terrestrial satellites where it results in smaller RTTs and the other metrics. By visualizing ISLs and computing metrics on this grid, we aim to

enhance our understanding of constellation connectivity and performance, ultimately contributing to the advancement of satellite communication technologies.

Overall, the state of the art in satellite communication networks is characterized by ongoing improvements in constellation design, ISL management, performance evaluation, and optimization techniques. Noting that now there are many ongoing researches on how to build the ISL grid where it is an open question and large field to discover and implement in the future.