```
https://soho.nascom.nasa.gov/data/realtime/eit_195/512/latest.jpg
# -*- coding: utf-8 -*-

from vispy.app.timer import *
from astropy.time import TimeDelta
from astropy.coordinates import solar_system_ephemeris
from poliastro.util import time_range
from data_functs import *
from simbody import SimBody
from astropy import units as u
from astropy.constants.codata2014 import G
from vispy.scene.visuals import Markers, Compound, Polygon, XYZAxis
import vispy.visuals.transforms as tr
import math

logging.basicConfig(filename="logs/sb_viewer.log",
                    level=logging.DEBUG,
                    format="%(funcName)s:\t\t%(levelname)s:%(asctime)s:\t%(message)s",
                    )
MIN_SYMB_SIZE = 5
MAX_SYMB_SIZE = 20
ST = tr.STTransform


class StarSystem:
    def __init__(self, view=None):
        self.INIT = False
        self.DATASET = setup_datastore()
        self.body_count = self.DATASET["BODY_COUNT"]
        self.body_names = self.DATASET["BODY_NAMES"]    # cast this to a tuple?
        self.body_data  = self.DATASET["BODY_DATA"]
        self.skymap     = self.DATASET["SKYMAP"]
        self.sim_params = self.DATASET["SYS_PARAMS"]
        self._sys_epoch = Time(self.DATASET["DEF_EPOCH"],
                               format='jd',
                               scale='tdb',
                               )
        self.simbods = self.init_simbodies(body_names=self.body_names)
        self.sb_list = [self.simbods[name] for name in self.body_names]
        self.sys_rel_pos = np.zeros((self.body_count, self.body_count), dtype=vec_type)
        self.sys_rel_vel = np.zeros((self.body_count, self.body_count), dtype=vec_type)
        self.body_accel = np.zeros((self.body_count,), dtype=vec_type)
        self._mainview = view
        self.cam = self._mainview.camera
        self.cam.auto_roll = False
        self.cam_rel_pos = np.zeros((self.body_count,), dtype=vec_type)
        self.cam_rel_vel = None
        self.bods_pos = None
        self._symb_sizes = self.get_symb_sizes()
        self.bod_symbs = [sb.body_symb for sb in self.sb_list]
        self._bods_viz = Markers(edge_color=(0, 1, 0, 1),
                                 size=self._symb_sizes,
                                 scaling=False, )
        self.trk_polys = []
        self.poly_alpha = 0.7
        self.orb_vizz = None
        self.frame_viz = None
        self.w_last = 0
        self.d_epoch = None
        self.avg_d_epoch = None
        self.end_epoch = None
        self.w_clock = Timer(interval='auto',
                             connect=self.update_epoch,  # change this
                             iterations=-1,
                             )
```

```
66          print("Target FPS:", 1 / self.w_clock.interval)
67          self.t_warp = 10000            # multiple to apply to real time in simulation
68          self.set_ephems()
69
70      def get_symb_sizes(self):
71          body_fovs = []
72          for sb in self.sb_list:
73              body_fovs.append(sb.dist2pos(pos=self.cam.center)['fov'])
74              sb.update_alpha()
75
76          raw_diams = [math.ceil(self._mainview.size[0] * b_fov / self.cam.fov) for b_fov in
            body_fovs]
77          pix_diams = []
78          for rd in raw_diams:
79              if rd < MIN_SYMB_SIZE:
80                  pix_diams.append(MIN_SYMB_SIZE)
81              else:
82                  pix_diams.append(rd)
83
84          return np.array(pix_diams)
85
86      def set_ephems(self, epoch=None, span=1):   # TODO: make default span to Time(1 day)
87          if epoch is None:
88              epoch = self._sys_epoch
89          else:
90              span = self.simbods["Earth"].orbit.period / 365.25
91
92          _t_range = time_range(epoch,
93                                periods=365,
94                                spacing=span,
95                                format="jd",
96                                scale="tdb",
97                                )
98          [sb.set_ephem(t_range=_t_range) for sb in self.sb_list]
99          self.end_epoch = _t_range[-1]
100         logging.info("END_EPOCH:\n%s\n", self.end_epoch)
101
102     def update_epoch(self, event=None):
103         if self.INIT:
104             w_now = self.w_clock.elapsed     # not the first call
105             dt = w_now - self.w_last
106             self.w_last = w_now
107         else:
108             w_now = 0                       # the first call sets up self.w_last
109             dt = 0
110             self.w_last = w_now - dt
111             self.INIT = True
112
113         d_epoch = TimeDelta(dt * u.s * self.t_warp)
114         if self.avg_d_epoch is None:
115             self.avg_d_epoch = d_epoch
116
117         new_epoch = self._sys_epoch + d_epoch
118         self.avg_d_epoch = (self.avg_d_epoch + d_epoch) / 2
119         self.do_updates(new_epoch=new_epoch)
120
121         if (self.end_epoch - new_epoch) < 2 * self.avg_d_epoch:
122             logging.debug("RELOAD EPOCHS/EPHEM SETS...")
123             self.set_ephems(epoch=new_epoch)               # reset ephem range
124
125         self._sys_epoch = new_epoch
126
127         logging.debug("AVG_dt: %s\n\t>>> NEW EPOCH: %s\n",
128                       self.avg_d_epoch,
129                       new_epoch.jd)
```

```
130
131     def init_simbodies(self, body_names=None):
132         solar_system_ephemeris.set("jpl")
133         sb_dict = {}
134         for name in self.body_names:
135             sb_dict.update({name: SimBody(body_name=name,
136                                           epoch=self._sys_epoch,
137                                           sim_param=self.sim_params,
138                                           body_data=self.body_data[name],
139                                           # add marker symbol to body_data
140                                           )})
141         logging.info("\t>>> SimBody objects created....\n")
142         return sb_dict
143
144     def do_updates(self, new_epoch=None):
145         for sb in self.sb_list:
146             sb.update_state(epoch=new_epoch)
147
148         # collect positions of the bodies into an array
149         self.bods_pos = []
150         self.bods_pos.extend([sb.pos for sb in self.sb_list])
151         self.bods_pos[4] += self.bods_pos[3]                          # add Earth pos to Moon pos
152         # self.trk_polys[3].transform = tr.STTransform(translate=self.bods_pos[3])  # move moon
            orbit to Earth pos
153         self.bods_pos = np.array(self.bods_pos)
154
155         i = 0
156         for sb1 in self.sb_list:
157             j = 0
158             # collect the position relative to the camera location
159             self.cam_rel_pos[i] = sb1.dist2pos(pos=self._mainview.camera.center)['rel_pos']
160             # if sb1.sb_parent is not None:
161             #     if sb1.sb_parent.name != self.sb_list[0].name:
162             #         pass
163             #         # sb1.trk_poly.transform =
                    ST(translate=list(self.simbods[sb1.sb_parent.name].state[0]))
164
165             # collect the relative position and velocity to the other bodies
166             for sb2 in self.sb_list:
167                 self.sys_rel_pos[i][j] = sb2.dist2pos(pos=sb1.pos)['rel_pos']
168                 self.sys_rel_vel[i][j] = sb2.vel - sb1.vel
169                 if i != j:
170
171                     # accumulate the acceleration from the other bodies
172                     self.body_accel[i] += (G * sb1.body.mass * sb2.body.mass) / (
173                             self.sys_rel_pos[i][j] * self.sys_rel_pos[i][j] * u.m * u.m)
174                 j += 1
175             i += 1
176
177         self._symb_sizes = self.get_symb_sizes()        # update symbol sizes based upon FOV of
            body
178         self._bods_viz.set_data(pos=self.bods_pos,
179                                 face_color=np.array([sb.base_color + np.array([0, 0, 0, self.
                                    poly_alpha])
180                                                      for sb in self.sb_list]),
181                                 edge_color=[1, 0, 0, .6],
182                                 size=self._symb_sizes,
183                                 symbol=self.bod_symbs,
184                                 )
185         logging.debug("\nSYMBOL SIZES :\t%s", self._symb_sizes)
186         logging.debug("\nCAM_REL_DIST :\n%s", [np.linalg.norm(rel_pos) for rel_pos in self.
            cam_rel_pos])
187         logging.debug("\nREL_POS :\n%s\nREL_VEL :\n%s\nACCEL :\n%s",
188                       self.sys_rel_pos, self.sys_rel_vel, self.body_accel)
189
```

```
190       def init_sysviz(self):
191           self.frame_viz = XYZAxis(parent=self._mainview.scene)       # set parent in
              MainSimWindow ???
192           self.frame_viz.transform = tr.STTransform()
193           self.frame_viz.transform.scale = [1e+05, 1e+05, 1e+05]
194           for sb in self.sb_list:
195               if sb.sb_parent is not None:
196                   new_poly = Polygon(pos=sb.o_track,
197                                      border_color=sb.base_color + np.array([0, 0, 0, self.
                                       poly_alpha]),
198                                      triangulate=False,
199                                      )
200                   sb.trk_poly = new_poly
201                   self.trk_polys.append(sb.trk_poly)
202
203           self.orb_vizz = Compound(self.trk_polys)
204           viz = Compound([self.frame_viz, self.bods_viz, self.orb_vizz])
205           viz.parent = self._mainview.scene
206           return viz
207
208       def run(self):
209           self.w_clock.start()
210
211       @property
212       def bods_viz(self):
213           return self._bods_viz
214
215
216   def main():
217       my_starsys = StarSystem()
218       my_starsys.run()
219
220
221   if __name__ == "__main__":
222       main()
223
```