Bijan Hamidi
5/8/2017
CPSC 484

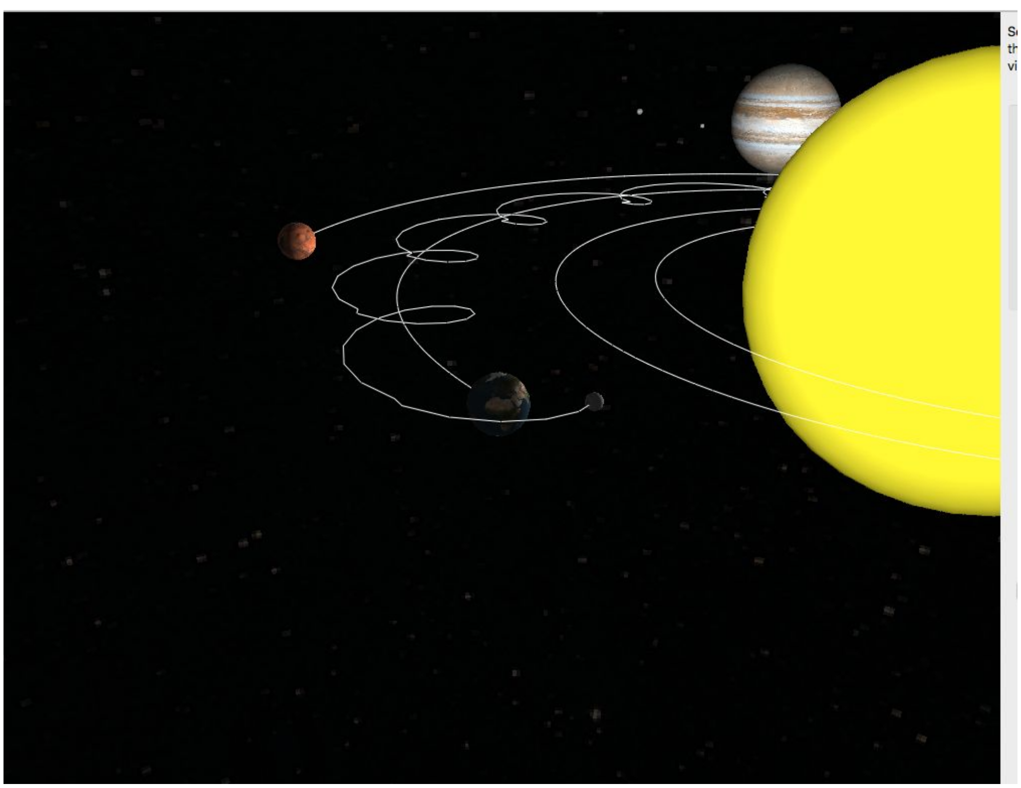
Final Project
Solar System Simulation

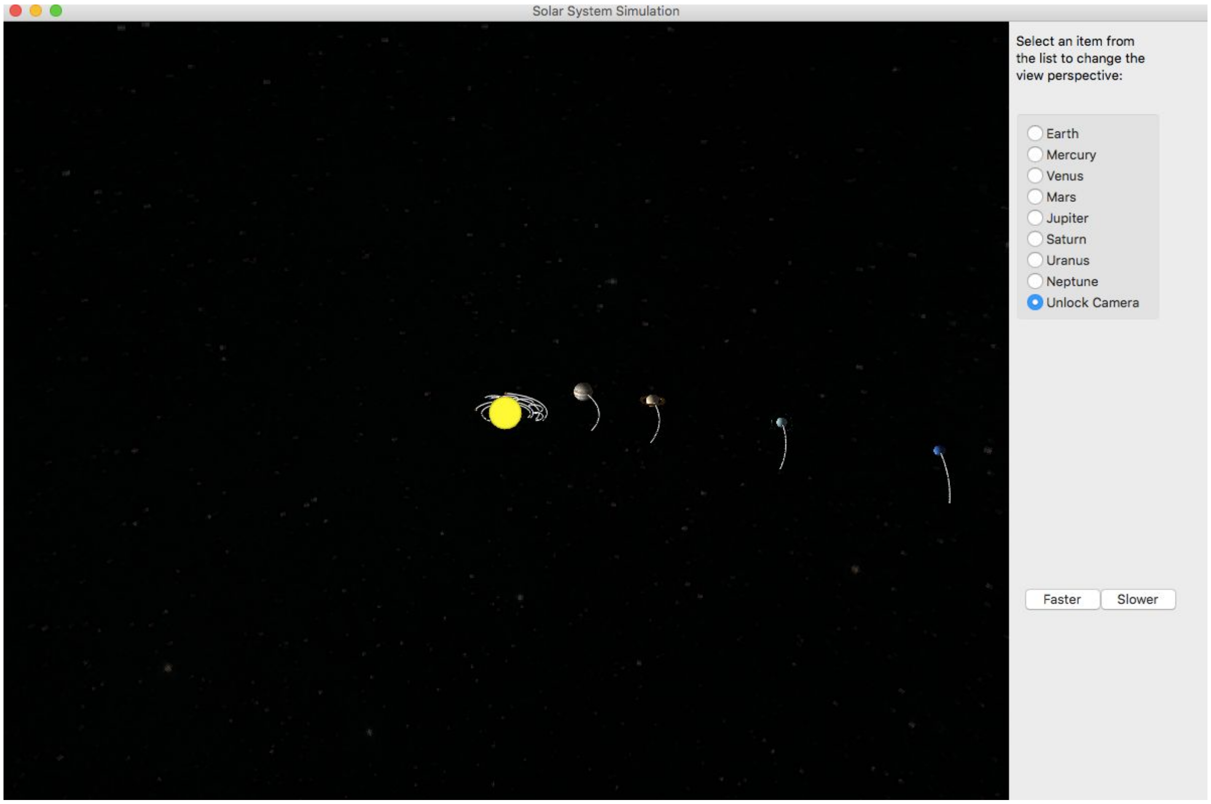
Features

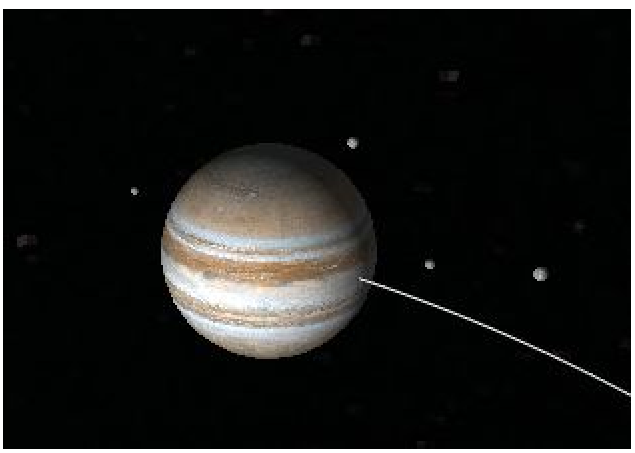
- Textures on 8 planets
- Local Light of emissive Sun
- Elliptical Planetary Orbits
- Moon Orbits with respective inclination
- Earth's Moon with Cyclonic Orbit
- Textured background with motion
- Zoom in view for each planet
- Speed up and Slow down animations
- Plant sizes and distances are respective to real life ratios within project scope
- Moon sizes and distances are respective to real life ratios within project scope

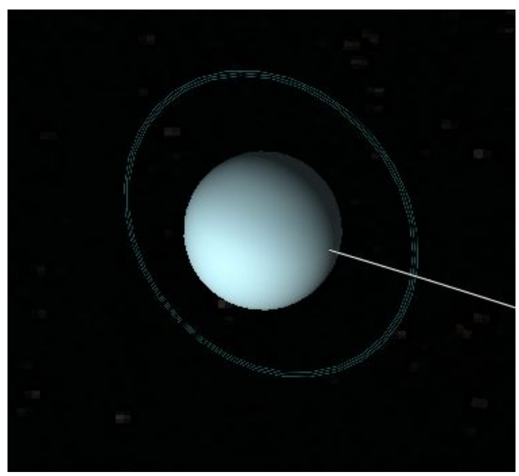
Note: Movements may appear to occasionally stutter due to performance issues during rendering.

Screen Shots




Solar System Simulation
Select an item from the list to change the view perspective:
Earth
Mercury
Venus
Mars
Jupiter
Saturn
Uranus
Neptune
Unlock Camera
Faster
Slower





```
#
# Solar System Simulation by Bijan A. Hamidi
#
from __future__ import division
from visual import *
import wx
import time

MAKE_TRAIL = true #set to false to improve performance, set to true to view orbits
cam_speed_div = 20

# adding side panel source code from camera.py
win = window(width=1200, height=800, menus=False, title='Solar System Simulation')
scene = display( window=win, width=1000, height=800, forward=-vector(1,1,2))
x1 = scene.width + 5
pan = win.panel   # addr of wx window object
wx.StaticText( pan, pos=(x1,10),
   label = "Select an item from \nthe list to change the \nview perspective:" )
# menu event handler
def hRadio(evt):
   global mode, fov, range_x
   mode =  ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', None][ bRadio.GetSelection()]
   if mode == None:  mode_lab.SetLabel('')
# scene camera definitions
mode = None
vss = scene
fov = pi/3.0  # 60 deg
range_x = 6  # simulates scene.range.x

# speed functions
def hCamSlower(evt):
   global cam_speed_div
   if cam_speed_div > 3:
      cam_speed_div = cam_speed_div -2

def hCamFaster(evt):
   global cam_speed_div
   if cam_speed_div < 140:
      cam_speed_div = cam_speed_div +2

# speed event handler
def Button12(y, label1, func1, label2, func2):
   # more code here
```

```
    bb = wx.Button( pan, label=label1, pos=(x1+5,y), size = (75,40))
    bb.Bind(wx.EVT_BUTTON, func1)
    bb = wx.Button( pan, label=label2, pos=(x1+80,y), size = (75,40))
    bb.Bind(wx.EVT_BUTTON, func2)
Button12(550, 'Faster', hCamFaster, 'Slower', hCamSlower )

# panel option List
bRadio = wx.RadioBox(pan, pos=(x1,90), choices = ['Earth', 'Mercury', 'Venus',
        'Mars', 'Jupiter', 'Saturn', 'Uranus','Neptune', 'Unlock Camera'], style= wx.RA_VERTICAL)
bRadio.SetSelection(8)   # Set Unlock Camera as intially selected.
bRadio.Bind(wx.EVT_RADIOBOX, hRadio)
mode_lab = wx.StaticText( pan, pos=(x1,310))



# load tga texture files
sun_texture = materials.texture(data = materials.loadTGA("images/sun_3.tga"), mapping =
'spherical', interpolate = False)
moon_texture = materials.texture(data = materials.loadTGA("images/moon_2.tga"), mapping =
'spherical', interpolate = False)
merc_texture = materials.texture(data = materials.loadTGA("images/mercury.tga"), mapping =
'spherical', interpolate = False)
ven_texture = materials.texture(data = materials.loadTGA("images/venus_2.tga"), mapping =
'spherical', interpolate = False)
mars_texture = materials.texture(data = materials.loadTGA("images/mars_3.tga"), mapping =
'spherical', interpolate = False)
jup_texture = materials.texture(data = materials.loadTGA("images/jupiter_3.tga"), mapping =
'spherical', interpolate = False)
sat_texture = materials.texture(data = materials.loadTGA("images/saturn_2.tga"), mapping =
'spherical', interpolate = False)
uranus_texture = materials.texture(data = materials.loadTGA("images/uranus_2.tga"), mapping
= 'spherical', interpolate = False)
nep_texture = materials.texture(data = materials.loadTGA("images/neptune_2.tga"), mapping =
'spherical', interpolate = False)
milky_way = materials.texture(data=materials.loadTGA("images/stars3.tga"),
mapping="spherical", interpolate = False)

# scene lighting & display
# Bruce Sherwood states that background images can not be applied
# work around method is using a large object with a texture
#
https://groups.google.com/forum/#!searchin/vpython-users/background$20image%7Csort:releva
nce/vpython-users/fQnMnCo8C98/f0TExDzyBAAJ
#scene.ambient = (1,1,1) #set ambient lighting to zero
```

```
scene.lights = []
scene.range = 3000
galaxy = sphere(radius=7000, material = milky_way)#.rotate(angle=180)#rotate 180 to show
most detail on bg

# sun lighting - uses base color with emissive properties
# Bruce Sherwood says objects with textures can not be a light source
# source - https://groups.google.com/forum/#!topic/vpython-users/YdYfsZBpCjE
# Bruce also states that no version of VPython supports realistic shadows..
# I have utilized local light to try and simulate night / day on planets to the best of VPython's
ability
# It doesnt seem possible to get intense shadows on the dark side of planets but I did manage
to get a shady tint
Sun = sphere(pos=(0,0,0), radius = 100, material = materials.emissive, color =
color.yellow)#material = sun_texture)
#lamplight = sphere(pos=(0,0,0), radius = 99.999, material = materials.emissive)
lamp = local_light(pos=(0,0,0), color=color.white)

# celestial Moon
Moon = sphere(pos = (215,0,0), radius = 3, material = moon_texture, make_trail = true, retain =
100)#color = color.white, make_trail = false)

# celestial bodies rocky
Earth = sphere(pos=(200,0,0), radius = 10, material = materials.earth, make_trail =
MAKE_TRAIL, retain = 190)
Mercury = sphere(pos = (150,0,0), radius = 2, material = merc_texture, make_trail =
MAKE_TRAIL, retain = 110)#color = color.yellow, make_trail = true)
Venus = sphere(pos = (165,0,0), radius = 9, material = ven_texture, make_trail = MAKE_TRAIL,
retain = 155)#material = materials.wood, make_trail = true)
Mars = sphere(pos = (245,0,0), radius = 10, material = mars_texture, make_trail =
MAKE_TRAIL, retain = 275)#color = color.red, make_trail = true)

# celestial bodies gas giants
Jupiter = sphere(pos = (550,0,0), radius = 60, material = jup_texture, make_trail =
MAKE_TRAIL, retain = 600)#color = color.red, make_trail = true)
Saturn = sphere(pos = (900,0,0), radius = 40, material = sat_texture, make_trail =
MAKE_TRAIL, retain = 900) #color = color.yellow , make_trail = true)
Uranus = sphere(pos = (1600,0,0), radius = 30, material = uranus_texture, make_trail =
MAKE_TRAIL, retain = 1600)#color = color.magenta, make_trail = true)
Neptune = sphere(pos = (2400,0,0), radius = 30, material = nep_texture, make_trail =
MAKE_TRAIL, retain = 2600) #color = color.cyan, make_trail = true)

# major celestial moons - Moons are accurate in size relation to Earth
```

```
ganymede = sphere(pos = (700,0,0), radius = 4.135, material = moon_texture, make_trail = true,
retain = 140)#jupiter
titan = sphere(pos = (1020,0,0), radius = 4.135, material = moon_texture, make_trail = true,
retain = 130) #saturn
callisto = sphere(pos = (730,0,0), radius = 3.783, material = moon_texture, make_trail = true,
retain = 140)#jupiter
io = sphere(pos = (640,0,0), radius = 2.859, material = moon_texture, make_trail = true, retain =
140)#jupiter
europa = sphere(pos = (670,0,0), radius = 2.45, material = moon_texture, make_trail = true,
retain = 140)#jupiter
triton = sphere(pos = (2470,0,0), radius = 2.124, material = moon_texture, make_trail = true,
retain = 130)#neptune

# velocity planets
Earth.v = vector(0,0,-7.5)
Mercury.v = vector(0,0,-8)
Venus.v = vector(0,0,-8)
Mars.v = vector (0,0,-7.5)
Jupiter.v = vector (0,0,-8)
Saturn.v = vector (0,0,-8)
Uranus.v = vector (0,0,-8)
Neptune.v = vector (0,0,-8)
#Moon.v = vector(0,0,-8)

#Populate list with rings for Saturn
rings = []
#ring_texture = materials.texture(data = materials.loadTGA("images/sun_3.tga"), mapping =
'cylinder', interpolate = False)
for x in xrange(200):
   rings.append(ring(pos=(Saturn.x,Saturn.y,Saturn.z), axis=(0,1,0), radius= 60+1.05*x/10,
thickness=.5, color = color.hsv_to_rgb((x/100%1,0.4,0.4))))#material = ring_texture))#color =
color.orange))
#Populate list with rings for Uranus
Urings = []
for x in xrange(3):
   Urings.append(ring(pos=(Uranus.x,Uranus.y,Uranus.z), axis=(0,0,1), radius= 60+x,
thickness=.2, color = color.cyan, opacity = 0.3))

#Moon Orbit Functions
#rad = 1.0
#fx = 2.0 - 2.0*min(abs((1-e*e)/(1-e)), abs(-(1-e*e)/(1+e)))
def move_moon(Moon_mov, planet, theta_degree, orbit_radius, ecliptic):
   ##for i in range(361):
```

```
    theta = radians(float(theta_degree))
##        r = (1-e*e)/(1-e*cos(theta))
##        nx = rad*(r*cos(theta) - fx)
##        ny = rad*r*sin(theta)
##        moon.pos[i]=vector(nx, ny, 0.0)
        #x_{P_2}=x_{P_1}+r\sin{\theta};\ \\ y_{P_2}=y_{P_1}-r(1-\cos{\theta})$$.
    r = orbit_radius
    nx = planet.x + r * sin(theta)
    nz = planet.z + r * cos(theta)
    ny = planet.y + ecliptic*cos(theta)
        ##ny = planet.y + r *(1 - cos(theta))
        #nx = rad*(r*cos(theta) - planet.x)
        #ny = rad*(r*sin(theta) - planet.y)
    Moon_mov.pos = vector(nx, ny, nz)
theta_count = 0
theta_countB = 180
theta_countC = 96
theta_countD = 240
# main loop
while (true):
    rate(50)
    time.sleep(1.0/cam_speed_div)

    if theta_count > 359:
        theta_count = 0
    else:
        theta_count += 24#cyclonic moon

    if theta_countB > 359:
        theta_countB = 0
    else:
        theta_countB += 3

    if theta_countC > 359:
        theta_countC = 0
    else:
        theta_countC += 3

    if theta_countD > 359:
        theta_countD = 0
    else:
        theta_countD += 3
```

```
    galaxy.rotate(angle = 0.001, axis=vector(0,-1,0))

    dist_earth = (Earth.x**2 + Earth.y**2 + Earth.z**2)**0.5
    radialVector = (Sun.pos - Earth.pos)/dist_earth
    Fgrav = 10000*radialVector/dist_earth**2
    Earth.v += Fgrav
    Earth.pos += Earth.v
    if dist_earth <= Sun.radius: break
    Earth.rotate(angle = 0.04, axis=vector(0,-1,0))

##    dist_moon = (Moon.x**2 + Moon.y**2 + Moon.z**2)**0.5
##    radialVector = (Earth.pos - Moon.pos)/dist_moon
##    Fgrav = 450000*radialVector/dist_moon**2
##    Moon.v += Fgrav
##    Moon.pos += Moon.v + vector(0,0,10)
##    #Moon.pos = (Earth.x, Earth.y, Earth.z + 60)
##    if dist_moon <= Earth.radius: break
    Moon.rotate(angle = 0.01, axis=vector(0,-1,0))
    move_moon(Moon, Earth, theta_count, 30, 5.15)
    #print(Moon.pos)
    #print(Earth.pos)

    dist_merc = (Mercury.x**2 + Mercury.y**2 + Mercury.z**2)**0.5
    radialVector = (Sun.pos - Mercury.pos)/dist_merc
    Fgrav = 10000*radialVector/dist_merc**2
    Mercury.v += Fgrav
    Mercury.pos += Mercury.v
    if dist_merc <= Sun.radius: break

    dist_ven = (Venus.x**2 + Venus.y**2 + Venus.z**2)**0.5
    radialVector = (Sun.pos - Venus.pos)/dist_ven
    Fgrav = 10000*radialVector/dist_ven**2
    Venus.v += Fgrav
    Venus.pos += Venus.v
    if dist_ven <= Sun.radius: break
    Venus.rotate(angle = 0.01, axis=vector(0,-1,0))

    dist_mars = (Mars.x**2 + Mars.y**2 + Mars.z**2)**0.5
    radialVector = (Sun.pos - Mars.pos)/dist_mars
    Fgrav = 12000*radialVector/dist_mars**2
    Mars.v += Fgrav
    Mars.pos += Mars.v
    if dist_mars <= Sun.radius: break
```

```
Mars.rotate(angle = 0.01, axis=vector(0,-1,0))

dist_jup = (Jupiter.x**2 + Jupiter.y**2 + Jupiter.z**2)**0.5
radialVector = (Sun.pos - Jupiter.pos)/dist_jup
Fgrav = 30000*radialVector/dist_jup**2
Jupiter.v += Fgrav
Jupiter.pos += Jupiter.v
if dist_jup <= Sun.radius: break
Jupiter.rotate(angle = 0.02, axis=vector(0,-1,0))

dist_sat = (Saturn.x**2 + Saturn.y**2 + Saturn.z**2)**0.5
radialVector = (Sun.pos - Saturn.pos)/dist_sat
Fgrav = 50000*radialVector/dist_sat**2
Saturn.v += Fgrav
Saturn.pos += Saturn.v
if dist_sat <= Sun.radius: break
Saturn.rotate(angle = 0.02, axis=vector(0,-1,0))

for i, val in enumerate(rings): #Update Saturn's ring position
    val.pos = Saturn.pos
    val.rotate(angle = 0.02, axis=vector(0,-1,0))


dist_uranus = (Uranus.x**2 + Uranus.y**2 + Uranus.z**2)**0.5
radialVector = (Sun.pos - Uranus.pos)/dist_uranus
Fgrav = 90000*radialVector/dist_uranus**2
Uranus.v += Fgrav
Uranus.pos += Uranus.v
if dist_uranus <= Sun.radius: break
Uranus.rotate(angle = 0.01, axis = vector(0,0,-1))

for i, val in enumerate(Urings): #Update Uranus' ring position
    val.pos = Uranus.pos
    val.rotate(angle = 0.01, axis = vector(0,0,-1))

dist_nep = (Neptune.x**2 + Neptune.y**2 + Neptune.z**2)**0.5
radialVector = (Sun.pos - Neptune.pos)/dist_nep
Fgrav = 130000*radialVector/dist_nep**2
Neptune.v += Fgrav
Neptune.pos += Neptune.v
if dist_nep <= Sun.radius: break
Neptune.rotate(angle = 0.01, axis=vector(0,-1,0))
```

```
#move major moons
ganymede.rotate(angle = 0.01, axis=vector(0,-1,0))
move_moon(ganymede, Jupiter, theta_countC, 150, 2)
titan.rotate(angle = 0.01, axis=vector(0,-1,0))
move_moon(titan, Saturn, theta_countB, 120, 3)
callisto.rotate(angle = 0.01, axis=vector(0,-1,0))
move_moon(callisto, Jupiter, theta_countB, 180, 16.69)
io.rotate(angle = 0.01, axis=vector(0,-1,0))
move_moon(io, Jupiter, theta_countC, 90, 0.4)
europa.rotate(angle = 0.01, axis=vector(0,-1,0))
move_moon(europa, Jupiter, theta_countD, 120, 0.5)
triton.rotate(angle = 0.01, axis=vector(0,-1,0))
move_moon(triton, Neptune, theta_countB, 100, 156.87)

if mode == 'a':  # scene.center
   vss.autoscale = vss.autocenter = False
   saved_pyvars = [ tuple(vss.forward), tuple(vss.center), vss.fov ]
   vss.center =  Earth.pos#+vector(0,0,30)
   vss.fov = fov
   vss.range = range_x + 150
elif mode == 'b':
   vss.autoscale = vss.autocenter = False
   saved_pyvars = [ tuple(vss.forward), tuple(vss.center), vss.fov ]
   vss.center =  Mercury.pos
   vss.fov = fov
   vss.range = range_x + 70
elif mode == 'c':  # scene.center
   vss.autoscale = vss.autocenter = False
   saved_pyvars = [ tuple(vss.forward), tuple(vss.center), vss.fov ]
   vss.center =  Venus.pos
   vss.fov = fov
   vss.range = range_x + 150
elif mode == 'd':  # scene.center
   vss.autoscale = vss.autocenter = False
   saved_pyvars = [ tuple(vss.forward), tuple(vss.center), vss.fov ]
   vss.center =  Mars.pos
   vss.fov = fov
   vss.range = range_x + 150
elif mode == 'e':  # scene.center
   vss.autoscale = vss.autocenter = False
   saved_pyvars = [ tuple(vss.forward), tuple(vss.center), vss.fov ]
   vss.center =  Jupiter.pos
   vss.fov = fov
```

```
    vss.range = range_x + 550
elif mode == 'f':  # scene.center
    vss.autoscale = vss.autocenter = False
    saved_pyvars = [ tuple(vss.forward), tuple(vss.center), vss.fov ]
    vss.center =  Saturn.pos
    vss.fov = fov
    vss.range = range_x + 350
elif mode == 'g':  # scene.center
    vss.autoscale = vss.autocenter = False
    saved_pyvars = [ tuple(vss.forward), tuple(vss.center), vss.fov ]
    vss.center =  Uranus.pos
    vss.fov = fov
    vss.range = range_x + 250
elif mode == 'h':  # scene.center
    vss.autoscale = vss.autocenter = False
    saved_pyvars = [ tuple(vss.forward), tuple(vss.center), vss.fov ]
    vss.center =  Neptune.pos
    vss.fov = fov
    vss.range = range_x + 250
```