

A Design of application program

In the following description of the design, for clarity we use pseudocode:

Indentation denotes block structure.

'=' denotes assignment; '==' denotes equality.

Pointers to objects are implicit.

An object's fields and methods are referenced and invoked by the object name suffixed with a dot, followed by the field name or method call.

Mathematical structures – strings, sets, and maps (sets of ordered pairs) – are treated as primitives, i.e. not referenced by pointers.

For a map m we define domain $m = \{ d \mid (d, x) \in m \}$ and range $m = \{ r \mid (x, r) \in m \}$, and for $d \in \text{domain } m$, $m(d) =$ the unique r such that $(d, r) \in m$.

And to allow a separation of this description of the application at large from the details of the implementation, we state certain assumptions:

GUI objects *menus*, *slider dials*, and *dialogue boxes* are the interaction techniques described in [FOL98]. A *menu item* is an element of a menu choice set – a basic interaction technique which can produce an event when selected. A *button* is the same thing outside of a menu. We call a drop-down menu that maintains a display of its currently selected item a *choice box*. A *text field* is a part of the application window containing a string that can be selected with the cursor and then edited using the keyboard.

We will allow a differentiation of event types at this stage: buttons and menu items produce *action events* on their interaction tasks, slider dials produce *adjustment events*. The mouse produces *mouse pressed events* and *mouse dragged events*; mouse events have fields $x_position$ and $y_position$. The other interaction techniques produce no events. An event has a *source* field pointing to its producer and a *command* field of unspecified type that is used to convey information to listeners.

We assume that the graphics library requires initialisation and display sequences to be defined: the former being executed only at the birth of the drawable, the latter whenever the drawable needs to be redisplayed (requested by the window manager or the application itself). For a drawable gld , $gld.display()$ causes gld to call the display sequences of its listeners.

The graphics library holds matrices for transformations from world coordinates to camera coordinates and from camera coordinates to screen coordinates, and holds a matrix stack for each of them. Material properties of surfaces are defined by 4 4-element *float* (32-bit floating point number) arrays specifying ambient, diffuse, and specular reflection, and emission, and a float specifying shininess.

As implementation specific points crop up, the requirements will be written in English if necessary and marked with either [JS1], [JS2], ... (Java Specifics) or [GS1], [GS2], ... (Graphics library Specifics), to be dealt with later.

A.1 Overview

Our GUI objects are mutable. We divide their properties into:

- (1) Variable state, not including the basic interaction techniques
- (2) Basic interaction techniques (objects that take interaction tasks and fire events)
- (3) Composite interaction techniques (inner-classed objects, themselves GUI objects)
- (4) Listener methods (called when certain events happen)
- (5) Construction

In this chapter we describe these properties of *Orrery* objects – instances of our application.

A.2 Variables of an Orrery object

For each abstract variable we choose a concrete variable to represent it (a *double* is a 64-bit floating point number, a *long* is a 64-bit integer):

$A1$	double <i>time</i> storing days measured from the beginning of year 4712, i.e. the standard Julian Day scale
$A2, A3$	strings <i>camera</i> , <i>lookat</i> storing unique names of bodies in the model
$A4, A5$	3-element double arrays <i>absolutecameraoffset</i> , <i>absolutelookatoffset</i> storing Cartesian coordinates in AU
$A6, A7$	3-element double arrays <i>relativecameraoffset</i> , <i>relativelookatoffset</i> storing Cartesian coordinates in AU where the camera position (after the absolute offset) is the origin, the camera-lookat line is the start of z^- and y^+ is up.
$A8$	double <i>rotate</i> storing degrees where positive degrees rotate the view anticlockwise.
$A9, A10$	2-element double arrays <i>angularcameraoffset</i> , <i>angularlookatoffset</i> storing longitude, latitude in degrees. For the former, positive longitude moves the camera to the right and positive latitude moves the camera up around the offset lookat point; for the latter positive longitude moves the

direction of view left and positive latitude moves the direction of view up around the offset lookat point.

A11 double *angleofview* storing degrees

A12 boolean *schematicnotrealistic*

A13 boolean *staticnotanimated*

A14 double *rate*

We keep an index for each type of satellite (see Section C), and one for all the satellites:

string sets *kepleriansatelliteindex*, *sunsatelliteindex*, *moonsatelliteindex*, *plutosatelliteindex* [JS1]

The model:

Satellites *satellites*

States *states*

4-element float arrays *starlight*, *light_position* initialised to [0.2, 0.2, 0.2, 1], [0, 0, 0, 1] respectively

Animation variables:

doubles *requestedframespersecond*, *timejump* initialised to 30, 0

suitable object *animator* that runs in its own thread and fires action events at regular intervals [JS2]

longs *lastanimatorevent*; *currentanimatorevent*

Application window:

window *f* initialised with title 'Orrery'

graphics library drawable *gld*, initialised [GS1]

A.3 Basic interaction techniques of an Orrery object

We will implement mappings from slider values to the corresponding variables where the linear increments of the sliders represent constant factors:

slider dials *angleofviewtechnique*, *ratetechnique* initialised: minimum value 0, maximum value 43, selection bar width 2 and initial value 43 for the former; minimum value 0, maximum value 64, selection bar width 4 and initial value 0 for the latter

buttons *schematicnotrealistictechnique*, *staticnotanimatedtechnique* initialised with inscriptions 'Schematic/realistic', 'Static/animated'

menu items *gregoriandatetechnique*, *juliandatetechnique*, *juliandaytechnique*, *trackingtechnique*, *perspectivetechnique* initialised with the inscriptions 'Gregorian date', 'Julian date', 'Julian day', 'Tracking', and 'Perspective'

A.4 Composite interaction techniques of an Orrery object

The composite interaction techniques launched by *gregoriandatetechnique*, *juliandatetechnique*, *juliandaytechnique* are instances of a parameterised class (note that an object of this class listens only to its *settechnique*, so no examination of events is necessary):

Variables: integers *type*, *year*, *month*
 doubles *day*, *hour*, *minute*

Interaction: text fields *yeartechnique*, *monthtechnique*, *daytechnique*, *hourtechnique*, *minutetechnique*
 button *settechnique* initialised with inscription 'Set'

Listeners: **actionPerformed(event)**

case *type* of

0 or 1 : *year*, *month*, *day*, *hour*, *minute* = texts of *yeartechnique*, *monthtechnique*, *daytechnique*, *hourtechnique*, *minutetechnique* parsed into doubles [JS3]

2 : *day* = text of *daytechnique* parsed into a double

case *type* of

0 : *time* = Julian Day value corresponding to *year*, *month*, *day*, *hour*, *minute* interpreted as Gregorian date

1 : *time* = Julian Day values corresponding to *year*, *month*, *day*, *hour*, *minute* interpreted as Julian date

2 : *time* = *day*

Construction: **TimeDialog(f, type)**

create dialogue box as child of *f*

register to listen to *settechnique*

case *type* of

0 : dialogue box title = 'Gregorian calendar'

display dialogue box

A.5 Listener methods of an Orrery object

For events from the graphics library. Since each satellite registers display lists with the graphics system when constructed, the construction of the satellites must be initiated by the graphics system:

```

init()
satelliteset = {}
for name in kepleriansatelliteindex do satelliteset = satelliteset ∪ {KeplerianSatellite(name)}
for name in sunsatelliteindex do satelliteset = satelliteset ∪ {SunSatellite(name)}
for name in moonsatelliteindex do satelliteset = satelliteset ∪ {MoonSatellite(name)}
for name in plutosatelliteindex do satelliteset = satelliteset ∪ {PlutoSatellite(name)}
satellites = Satellites(satelliteset)
states = satellites.statesAt(time)
Set rendering modes needed for both schematic and realistic rendering [GS2]
if schematicnotrealistic then set remaining rendering modes needed for schematic rendering [GS3]
else set remaining rendering modes needed for realistic rendering [GS4]

display()
set camera-to-window matrix to display zoom vertical degrees in the display width and display height, with farpoint and
nearpoint set to include the lookat point and be close enough to avoid z-buffer aliasing problems [GS5]
set world-to-camera matrix to reflect the camera position specified by the camera, lookat, and offsets [GS6]
clear the display of gld to black [GS7]
states.toDisplay(gld, schematicnotrealistic)

```

For events from buttons, menu items, and the animator: In the case of an event from the animator, we look at the last time an animator event was picked from the event queue, to see if we are maintaining the required frame rate. We aim to keep the framerate from 0.8 to 0.9 the requested framerate, and assume that the displayed rate is on average 0.85 the requested rate:

```

actionPerformed(event)
case event.source of
  animator :
    changed = false
    lastanimatorevent, currentanimatorevent = currentanimatorevent, current
    system time in milliseconds
    if 1000/(currentanimatorevent-lastanimatorevent) <
    0.8*requestedframespersecond then
      requestedframespersecond = requestedframespersecond -1
      changed = true
    if 1000/(currentanimatorevent-lastanimatorevent) >
    0.9*requestedframespersecond then
      requestedframespersecond = requestedframespersecond + 1
      changed = true
    if changed then
      timejump = rate/(86400*0.85*requestedframespersecond)
      stop the animator
      start a new animator with requestedframespersecond
    states = satellites.statesAfter(time, timejump, states)
    time = time + timejump
    gld.display()
  gregoriandate :
    TimeDialog(f, 0)
  juliandate :
    TimeDialog(f, 1)
  julianday :
    TimeDialog(f, 2)
  tracking :
    TrackingDialog(f)
  perspective :
    PerspectiveDialog(f)
  schematicnotrealistictechnique :
    schematicnotrealistic = ¬schematicnotrealistic
    gld.display()
  animatednotstatictechnique :
    animatednotstatic = ¬animatednotstatic
    if ¬animatednotstatic then stop the animator
    else start a new animator

```

For events from slider dials: (There are 86400 seconds in a day. Our dynamic framerate code keeps the framerate around 0.85 the requested frame rate.)

adjustmentValueChanged(event)

case *event*.source of

```

    angleofviewtechnique :    angleofview = event.command
    ratetechnique :          rate = event.command
                             timejump = rate / (86400*0.85*requestedframespersecond)
                             stop the animator
                             if animatednotstatic then start a new animator
  
```

For events from the mouse:

mousePressed(event)

dragx = *event*.x_position

dragy = *event*.y_position

mouseDragged(event)

angularcameraoffset[0] = *angularcameraoffset*[0] + (*dragx* - *event*.x_position)/5

angularcameraoffset[1] = *angularcameraoffset*[1] - (*dragy* - *event*.y_position)/5

dragx = *event*.x_position

dragy = *event*.y_position

gl.display()

A.6 Construction of an Orrery object

Each index of Satellite names will correspond to a set of Satellite objects of a certain class. Satellites will be constructed from human-readable data files in the local directory that have their names:

Model: read *time*, *camera*, *lookat*, *absolutecameraoffset*, *absolutelookatoffset*, *relativecameraoffset*, *relativelookatoffset*, *viewup*, *angularcameraoffset*, *angularlookatoffset*, *angleofview*, *schematicnotrealistic*, *staticnotanimated*, *rate*, *kepleriansatelliteindex*, *fixedsatelliteindex* from file *Variables*
satelliteindex = *kepleriansatelliteindex* ∪ *sunsatelliteindex* ∪ *moonsatelliteindex* ∪ *plutosatelliteindex*

Listen: register to listen to *gl*, *gregoriandate*, *juliandate*, *julianday*, *tracking*, *perspective*, *schematicnotrealistictechnique*, *staticnotanimatedtechnique*, *angleofviewtechnique*, *ratetechnique*

GUI: build menu 'Set time' including *gregoriandate*, *juliandate*, *julianday*
 build menu 'Set view' including *tracking*, *perspective*
 set *f*'s menubar to consist of 'Set time', 'Set view'
 set *f*'s background colour to black
 create panel including *schematicnotrealistictechnique*, *staticnotanimatedtechnique* and place at the bottom of *f*
 place *gl* in the centre of *f*
 place *angleofviewtechnique* on the left of *f*, smallest value at the top
 place *ratetechnique* on the right of *f*, smallest value at the top [JS4]
 register to listen to the window events of *f*
 display *f*

B Design of application model

B.1 Overview

Our model objects are immutable. We divide their properties into:

- (1) Constant state
- (2) Observation methods (not needed when public fields can be inspected)
- (3) Display functions

(4) Construction (absent, of course, in descriptions of abstract classes)

In this chapter we describe these properties of the four model classes specified in Chap. 3.

B.2 The State class

Since we do not intend to implement Satellite objects which produce State objects of varying size, mass, or visual properties, these fields are simply copied from the Satellite which produces the State, reducing the number of arguments to the constructor. The reason that a circle of fixed pixel diameter is drawn at the centre of each body is so that it can be seen when the 3D sphere becomes too small to be drawn (these circles are not 3D: they are always drawn on the screen as circles of the specified diameter). The spheres never change, so can be kept in display lists:

Constants: strings *name*, *parent*
doubles *mass*, *spherediameter*, *ringwidth* the first to be interpreted as kg, others as AU
4-element float arrays *ambient*, *diffuse*, *specular*, *emission*
float *shininess*
double *time* to be interpreted in the Julian Day scale
3-element double arrays *position*, *velocity*
Drawable *drawable*
integers *texture*, *slist*, *rlist*

Display: **toDisplay(*schematicnotrealistic*)**
push world-to-camera matrix [GS8]
push world-to-camera matrix
change the world-to-camera matrix so that the parent body is at the origin, satellite is on the positive *x*-axis, and the *xy* plane is the orbital plane of the satellite [GS9]
if *schematicnotrealistic* then
draw ring of width 1 pixel in colour corresponding to *ambient* as RGBA, in the *xy* plane, centered on the origin, with radius equal to the satellite-parent distance [GS10]
else
if *ringwidth* ≠ 0 then
draw a ring in colour corresponding to *ambient* as RGBA, in the *xy* plane, centered on the origin, with radius equal to the satellite-parent distance, and ring width equal to *ringwidth* [GS11]
pop world-to-camera matrix [GS12]
change world-to-camera matrix so that the satellite's position is at the origin
if *schematicnotrealistic* then execute the display list *slist* in *drawable*
else execute the display list *rlist* in *drawable*
pop world-to-camera matrix

Construction: **State(*satellite*, *time*, *position*, *velocity*, *drawable*, *texture*, *slist*, *rlist*)**
copy the variables *name*, *parent*, *mass*, *spherediameter*, *ringwidth*, *ambient*, *diffuse*, *specular*, *shininess*, *emission* from the Satellite argument, set *time*, *position*, *velocity*, *drawable*, *texture*, *slist*, *rlist* to their corresponding arguments

B.3 The States class

The State objects are stored in the ranges of two maps: *m* maps a State's *parent* string (which can be 'Origin') to a set containing the State. *n* maps a State's *name* string to the State itself, and is only used for extracting a State from a States.

The component State objects of a States are rendered in a depth-first traversal of *m*.

Constants: maps *m*, *n* [JS5]

Observations: **extract(*name*)**
return *n(name)*

Display: **toDisplay(*drawable*, *schematicnotrealistic*)**
renderTraverse('Origin', *drawable*, *schematicnotrealistic*)

renderTraverse(*parent*, *drawable*, *schematicnotrealistic*)

```

v = m(parent)
for state in v do
    state.toDisplay(drawable, schematicnotrealistic)
    push world-to-camera matrix
    move to position of state
    renderTraverse(state.name, drawable, schematicnotrealistic)
    pop world-to-camera matrix

```

Construction: **States(collection)**
 $m = \{ \}$
for state in collection do
 if $s.parent \in \text{domain } m$ **then** $v = m(state.parent)$
 else $v = \{ \}$
 $w = v \cup \{state\}$
 $m = (m \setminus \{(state.parent, v)\}) \cup \{(state.parent, w)\}$
 $n = \{ \}$
for state in collection do $n = n \cup \{(state.name, state)\}$

B.4 The Satellite abstract class

Constants: strings *name*, *parent*
 double *mass*, *spherediameter*, *ringwidth* the first to be interpreted as kg, others as AU
 4-element float arrays *ambient*, *diffuse*, *specular*, *shininess*, *emission*
 byte array *raw* initialised to be of size $3 \times 200 \times 100$
 Drawable *drawable*
 integers *width* (initialised 200), *height* (initialised 100), *slist*, *rlist*
 integer array *texture* initialised [0]

Initialisation: **makeLists()**
 load the bytes of .raw file with name *name* into *raw*
 register *texture*[0] as the 2D texture of *raw* and the current texture to be drawn with *drawable* [GS13]
 register *slist* as a display list performing
 if *spherediameter* $\neq 0$ **then**
 draw a filled circle of diameter 10 pixels at the origin, in colour corresponding to *ambient* as
 RGBA [GS14]
 draw a sphere of radius *spherediameter*/2 at the origin, in colour corresponding to *ambient* as
 RGBA [GS15]
 register *rlist* as a display list performing
 if *spherediameter* $\neq 0$ **then**
 draw a filled circle of diameter 1 pixel at the origin in colour corresponding to *ambient* as
 RGBA
 draw a sphere of radius *spherediameter*/2 in material corresponding to the origin in colour
 corresponding to *ambient*, *diffuse*, *specular*, *emission* as RGBA and *shininess* [GS16]

Observations: **stateAt(time)**
 stateAfter(time, quantum, states)

B.5 The Satellites class

The Satellite objects are stored in the range of a map *m*, which allows them to be looked up by name (we require that Satellite objects in the set passed to the constructor have unique names). The interrogation function build sets of states and then wraps them in a State object.

Constants: map *m*

Observations: **statesAt(time)**
 $c = \{ \}$
 for satellite in range m do $c = c \cup \{(satellite.stateAt(time))\}$
 $minimum = time$

```

for state in c do minimum = min{state.time, minimum}
s = States(c)
if minimum < time then
    currenttime = minimum;
    while currenttime < time do
        s = statesAfter(currenttime, 1/24, s)
        currenttime = currenttime + 1/24
    return s

statesAfter(time, quantum, states)
c = { }
for satellite in range m do c = c ∪ {satellite.stateAfter(time, quantum, states)}
return States(c)

```

Construction: **Satellites**(*collection*)
 m = { }
 for *satellite* **in** *collection* **do** *m* = *m* ∪ {(*satellite.name*, *satellite*)}

C Example data objects

We present concrete Satellite classes that can be used to build a simulation of the solar system. A general implementation of the Keplerian model is shown here in pseudocode; it cannot model the motion of the Moon or Pluto, so special (and somewhat larger) classes are defined for those bodies, based on [MEU98]. In this simulation we fix the Sun at the world coordinate origin: it can therefore be served by a far simpler class. The Java code can be found in Section F.

C.1 KeplerianSatellite

In our calculation of the mean elements we follow Meeus [MEU98]; the conversion to heliocentric Cartesian coordinates is one described by P. Schlyter of the Swedish Amateur Astronomer's Society.

The functions $\text{dsin}(d)$, $\text{dcos}(d)$ return the sine and cosine of d interpreted as degrees. The function $\text{datan2}(a, b)$ takes 2D coordinates and returns the corresponding angle in degrees. The function $\text{normalise}(d)$ corrects a degree value so that it lies within $[0, 360]$.

We include three private methods in the observations section because they support stateAt and stateAfter. The function $\text{positionAt}(time)$ returns the position of the body at $time$ (every new variable in this method is a local double); sumPolynomial uses Horner's rule to evaluate a polynomial in $time$ specified by a list (which we assume is indexed from 0 and has a length field) of coefficients from lowest to highest order; eccentricAnomaly solves Kepler's equation iteratively¹ (our accuracy requires less than five iterations).

Extra state: lists *inclinationlist*, *longitudeofascendingnodelist*, *semimajoraxislist*, *eccentricitylist*, *longitudeofperiastronlist*, *meanlongitudelist*

Observation: **stateAt**(*time*)
 position1 = positionAt((*time* - 2451545)/36525)
 position2 = positionAt((*time* + 1/24 - 2451545)/36525)
 velocity = [*position2*[0] - *position1*[0],
 position2[1] - *position1*[1],
 position2[2] - *position1*[2]]]
 return State(**this**, *time*, *position1*, *velocity*)

 stateAfter(*time*, *quantum*, *states*)
 steppedtime = *time* + *quantum*
 position1 = positionAt((*steppedtime* - 2451545)/36525)

¹ Kepler's equation $\text{eccentric anomaly} = \text{mean anomaly} + \text{eccentricity} \sin(\text{eccentric anomaly})$ is a transcendental function which cannot be solved directly.

```

steppedtimeplushour = steppedtime + 1/24
position2 = positionAt(steppedtimeplushour - 2451545)/36525)
velocity = [    position2[0] - position1[0],
              position2[1] - position1[1],
              position2[2] - position1[2]    ]
return new State(this, steppedtime, position1, velocity);

positionAt(time)
inclination, longitudeofascendingnode, semimajoraxis, eccentricity, longitudeofperiastron, meanlongitude =
normalise(sumPolynomial(time, inclinationlist)), normalise(sumPolynomial(time, longitudeofascendingodelist)),
sumPolynomial(time, semimajoraxislist), sumPolynomial(time, eccentricitylist), normalise(sumPolynomial(time,
longitudeofperiastronlist)), normalise(sumPolynomial(time, meanlongitudelist)
meananomaly = normalise(meanlongitude - longitudeofperiastron);
argumentofperiastron = normalise(longitudeofperiastron - longitudeofascendingnode)
eccentricanomaly = eccentricAnomaly(meananomaly, eccentricity)
orbitalx = semimajoraxis*(dcos(eccentricanomaly) - eccentricity)
orbitaly = semimajoraxis*sqrt(1 - eccentricity*eccentricity)* dsin(eccentricanomaly)
trueanomaly = datan2(orbitaly, orbitalx)
distance = sqrt(orbitalx*orbitalx + orbitaly*orbitaly)
truelongitude = normalise(trueanomaly + argumentofperiastron)
x = distance*( dcos(longitudeofascendingnode)*dcos(truelongitude)-
dsin(longitudeofascendingnode)*dsin(truelongitude)* dcos(inclination) )
y = distance*( dsin(longitudeofascendingnode)*dcos(truelongitude)+
dcos(longitudeofascendingnode)*dsin(truelongitude)* dcos(inclination) )
z = distance*dsin(truelongitude)*dsin(inclination)
return [x, y, z]

sumPolynomial(time, coefficientlist)
sum = coefficientlist[coefficientlist.length-1]
for i = coefficientlist.length-2 to 0 do sum = sum*time + coefficientlist[i]
return sum

eccentricAnomaly(meananomaly, eccentricity)
approximation1 = meananomaly
approximation2 = meananomaly+(180/π)*eccentricity*
dsin(approximation1)*(1+eccentricity*dcos(approximation1))
while |approximation1 - approximation2| > 0.005 do
    approximation1 = approximation2
    approximation2 = meananomaly + (180/π)*eccentricity*
dsin(approximation1)*(1+eccentricity*dcos(approximation1))
return approximation2

```

Construction: **KeplerianSatellite**(*path*)
read all superclass and additional constants from the file specified by *path*

D Java specifics

- [JS1] The utility class `HashSet` implements sets, but since its `add` method requires time to ensure that the 'no duplicates' property of the set is maintained, we will choose to use the 'extensible array' class `Vector` instead, ensuring maintenance of the set property ourselves.
- [JS2] The Swing utility `Timer` provides this functionality, and has a convenient `Timer(interval in milliseconds , listener)` constructor.
- [JS3] The Java Abstract Windowing Toolkit layout class `BorderLayout` maintains 4 panels of minimal size surrounding a central panel of maximal size – ideal for our purposes.
- [JS4] The wrapper class `Double` has a method `parseDouble` which does this, throwing an exception if it is impossible. Of great value to us is that it can parse scientific form, e.g. 23.4E-6.

[JS5] The utility class `TreeMap` implements maps.

E Graphics library specifics

[GS1] We use the JOGL binding to OpenGL (a reference implementation hosted on java.net), which implements OpenGL 1.4 and the Utility Library. One thing lacking until Sun releases the official binding is a *lightweight* drawable – currently the only hardware-accelerated drawable extends the AWT class `Canvas`. This is of no great consequence for us, but to avoid the problems of mixing light- and heavy-weight GUI components, we will use AWT exclusively in our application.

A drawable is initialised in JOGL with

```
GLDrawable.getFactory.createGLCanvas(new GLCapabilities());
```

A `GLDrawable gld` has GL, GLU objects associated with it, referenced with `gld.getGL()`, `gld.getGLU()`; these are instances of OpenGL and GL Utility Library systems. A GL object has a method with the name of each OpenGL instruction; OpenGL constants are simply static fields of the GL class.

[GS2] We always want the following settings:

```
gl.glClearColor(0,0,0,0);
gl.glCullFace(GL.GL_BACK);
gl.glLightModelfv(GL.GL_LIGHT_MODEL_AMBIENT, starlight);
gl.glLightModeli(GL.GL_LIGHT_MODEL_LOCAL_VIEWER, 1);
gl.glBlendFunc(GL.GL_SRC_ALPHA, GL.GL_ONE_MINUS_SRC_ALPHA);
gl.glEnable(GL.GL_DEPTH_TEST);
gl.glEnable(GL.GL_CULL_FACE);
gl.glEnable(GL.GL_LIGHT0);
gl.glEnable(GL.GL_POINT_SMOOTH);
gl.glEnable(GL.GL_LINE_SMOOTH);
```

The first sets the colour that will be used whenever a command to clear the display is executed. The second ensures that the GL system doesn't bother drawing the inside faces of spheres. The third and fourth set up our ambient lighting and ensure that specular reflections take the viewpoint into account. The blend command specifies how a translucent fragment (a component of a final pixel) should affect the final colour if blending is enabled. We then enable depth testing, so that a fragments are discarded if they are further away than an existing fragment, and enable the culling we specified earlier. The default light source is enabled, and the GL system is instructed to draw points and lines smoothly (i.e. with 'feathered' edges) – note that these only have an effect if blending is enabled.

[GS3] In the schematic rendering will use a simple antialiasing scheme, because on colour displays antialiased representations of curves give a much clearer impression of the true curve, and under OpenGL's simple antialiasing scheme points are drawn as smooth circles rather than squares. We simply enable blending to allow the previous settings to take effect:

```
gl.glEnable(GL.GL_BLEND);
```

[GS4] We enable the lighting that we have previously defined with:

```
gl.glEnable(GL.GL_LIGHTING);
```

[GS5] The GL Utility Library function `gluPerspective` applies a viewport transformation [FOL98] appropriate to a given angle of view, aspect ratio, and front and back clipping planes. We are making use of OpenGL's depth-buffer in the realistic rendering; it is important we do not require more resolution of the buffer than it has, so in the case of a realistic rendering we use the precalculated distance from the camera to the lookat point to set appropriate cutoff distances:

```
viewdistance =
    Math.sqrt(
        (finalcamera[0]-finallookat[0])*
        (finalcamera[0]-finallookat[0])      +
        (finalcamera[1]-finallookat[1])*
        (finalcamera[1]-finallookat[1])      +
        (finalcamera[2]-finallookat[2])*
        (finalcamera[2]-finallookat[2])
    );
```

```

gl.glMatrixMode(GL.GL_PROJECTION); gl.glLoadIdentity();

if (schematicnotrealistic)
{   glu.gluPerspective
    (   angleofview,
        (float)glc.getWidth() / (float)glc.getHeight(),
        0.0001f,
        100
    );
}
else
{   glu.gluPerspective
    (   angleofview,
        (float)glc.getWidth() / (float)glc.getHeight(),
        viewdistance/25,
        viewdistance*4
    );
}

gl.glMatrixMode(GL.GL_MODELVIEW); gl.glLoadIdentity();

```

[GS6] Calculate *absolutecamera* and *absolutelookat*, by adding up the displacement-from-parent chains from the camera body and the lookat body to the origin:

```

if (camera!=null&&!camera.equals("Origin"))
{   State tempstate = states.extract(camera);
    double[] temparray =
    {   tempstate.position[0] + absolutecameraoffset[0],
        tempstate.position[1] + absolutecameraoffset[1],
        tempstate.position[2] + absolutecameraoffset[2]
    };
    while (!(tempstate.parent.equals("Origin")))
    {   tempstate = states.extract(tempstate.parent);
        temparray[0] += tempstate.position[0];
        temparray[1] += tempstate.position[1];
        temparray[2] += tempstate.position[2];
    }
    finalcamera = temparray;
}
else finalcamera = absolutecameraoffset;

if (lookat!=null&&!lookat.equals("Origin"))
{   State tempstate = states.extract(lookat);
    double[] temparray =
    {   tempstate.position[0] + absolutelookatoffset[0],
        tempstate.position[1] + absolutelookatoffset[1],
        tempstate.position[2] + absolutelookatoffset[2]
    };
    while (!(tempstate.parent.equals("Origin")))
    {   tempstate = states.extract(tempstate.parent);
        temparray[0] += tempstate.position[0];
        temparray[1] += tempstate.position[1];
        temparray[2] += tempstate.position[2];
    }
    finallookat = temparray;
}
else finallookat = absolutelookatoffset;

```

Apply relative offsets. This Utility Library function applies a transformation that maps the first three parameters (as coordinates) to the origin, the second three onto the z- axis and the third into the yz plane (In OpenGL additional transformations are multiplied on the right of the current transformation):

```

glu.gluLookAt
(   relativecameraoffset[0],
    relativecameraoffset[1],
    relativecameraoffset[2],
    relativelookatoffset[0],
    relativelookatoffset[1],
    -viewdistance + relativelookatoffset[2],
    0,1,0
);

```

Apply angular camera offsets:

```

gl.glTranslated(0,0,-viewdistance);
gl.glRotated(angularcameraoffset[1],1,0,0);
gl.glRotated(-angularcameraoffset[0],0,1,0);
gl.glTranslated(0,0,viewdistance);

```

Apply angular lookat offsets:

```
gl.glRotated(-angularlookatoffset[1],1,0,0);
gl.glRotated(-angularlookatoffset[0],0,1,0);
```

Apply view rotate (the image rotates clockwise for positive viewrotate):

```
gl.glRotated(-viewrotate,0,0,1);
```

Apply viewpoint and viewrotate:

```
glu.gluLookAt
(   finalcamera[0],finalcamera[1],finalcamera[2],
    finallookat[0],finallookat[1],finallookat[2],
    viewup[0],viewup[1],viewup[2]
);
```

[GS7] We also clear the depth buffer:

```
gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
```

[GS8] As one might expect:

```
gl.glPushMatrix();
```

[GS9] We can achieve this with an orthogonal matrix. We wish to map the planet's normalised position vector onto $x+$, the unit vector perpendicular to its position and velocity onto $z+$, and the unit vector perpendicular to those onto $y+$. We construct a matrix in which the unit vectors mentioned form the first three rows:

```
double normposition[] = normalise(position);
double normvelocity[] = normalise(velocity);
double row2[] = crossproduct(normposition,normvelocity);
double row1[] = crossproduct(row2,normposition);
double matrix[] = {
    normposition[0],    normposition[1],    normposition[2],    0,
    row1[0],           row1[1],           row1[2],           0,
    row2[0],           row2[1],           row2[2],           0,
    0,                 0,                 0,                 1
};
gl.glMultMatrixd(matrix);
```

[GS10] We use the OpenGL Utility Library to draw a 'silhouetted' (only external edges drawn) ring of zero width (inner and outer radii equal). It is actually a regular 128-gon. The parameter 1 means that there are no internal rings of vertices.

```
glu.gluQuadricDrawStyle(gluq, GLU.GLU_SILHOUETTE);
gl.glColor4fv(ambient);
glu.gluDisk
(   gluq,
    xyzdistance,
    xyzdistance,
    128,
    1
);
```

[GS11] The specification `GL_SMOOTH` means that normals are generated for each vertex. Our material properties are loaded into OpenGL in the obvious way. Note that here we draw a ring of width *ringwidth*, and we must in fact draw another one with normals facing the other way, so that side can reflect light too. There is no problem with one ring occluding the other because we have enabled culling.

```
glu.gluQuadricNormals(gluq, GL.GL_SMOOTH);
gl.glMaterialfv(GL.GL_FRONT, GL.GL_AMBIENT, ambient);
gl.glMaterialfv(GL.GL_FRONT, GL.GL_SPECULAR, specular);
gl.glMaterialfv(GL.GL_FRONT, GL.GL_SPECULAR, specular);
gl.glMaterialfv(GL.GL_FRONT, GL.GL_SHININESS, shininess);
gl.glMaterialfv(GL.GL_FRONT, GL.GL_EMISSION, emission);
glu.gluDisk
(   gluq,
    xyzdistance-ringwidth/2,
    xyzdistance+ringwidth/2,
    128,
    1
);
glu.gluQuadricOrientation(gluq, GLU.GLU_INSIDE);
glu.gluDisk
```

```
(   gluq,
    xyzdistance-ringwidth/2,
    xyzdistance+ringwidth/2,
    128,
    1
);
```

[GS12] The expected:

```
gl.glPopMatrix();
```

[GS13] We get a name for a texture in the similar way to a display list, *bind* it as the current 2D texture to be drawn, then instruct the GL system to create a collection of textures of various sizes, from which an appropriate size will be automatically selected when the texture is mapped (this is called *mipmapping*). We wish textures and underlying materials to both be factors in the final fragments, which is default:

```
gl.glGenTextures(1, texture);
gl.glBindTexture(GL.GL_TEXTURE_2D, texture[0]);
glu.gluBuild2DMipmaps
(GL.GL_TEXTURE_2D, GL.GL_RGB, width, height, GL.GL_RGB, GL.GL_UNSIGNED_BYTE, raw);
```

[GS14] OpenGL requires that points are specified in a list:

```
gl.glPointSize(10);
gl.glColor4fv(ambient);
gl.glBegin(GL.GL_POINTS);
    gl.glVertex3f(0,0,0);
gl.glEnd();
```

[GS15] Again we use the Utility Library. The second parameter is the radius, the third the number of segments the sphere would have were it an orange, and the fourth the number of horizontal layers it is split into (the edges of the faces making up the sphere look like lines of longitude and latitude). We can get away with relatively small values for these because the sphere is being drawn in flat colour.

```
glu.gluQuadricDrawStyle(gluq, GLU.GLU_FILL);
glu.gluSphere(gluq, spherediameter/2, 24, 24);
```

[GS16] Note that when drawing a shaded sphere we require many segments so that the terminator on a planet can be approximated well as the planet circles its parent.

```
glu.gluQuadricOrientation(gluq, GLU.GLU_OUTSIDE);
glu.gluSphere(gluq, spherediameter/2, 256, 24);
```

F Java code

Java code for the Orrery class

```
import net.java.games.jogl.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.Timer;
import java.io.*;
public class Orrery extends WindowAdapter
    implements GLEventListener, ActionListener, AdjustmentListener, MouseListener,
    MouseMotionListener
{
    //Concrete variables:
    private double time;
    private String camera, lookat;
    private double absolutecameraoffset[], absolutelookatoffset[], relativecameraoffset[],
        relativelookatoffset[], viewup[], viewrotate, angularcameraoffset[],
        angularlookatoffset[], angleofview;
    private boolean schematicnotrealistic, staticnotanimated;
    private double rate;
    //An index for each type of satellite, and one for all the satellites:
    private Vector kepleriansatelliteindex, sunsatelliteindex, moonsatelliteindex,
        plutosatelliteindex, satelliteindex = new Vector();
    //The model:
    private Satellites satellites; private States states;
    private float starlight[] = {0.15f,0.15f,0.15f,1}, light_position[] = {0,0,0,1};
    //Animation variables:
```

```

private double requestedframespersecond = 40, timejump = 0;
private Timer animator;
long lastanimatorevent, currentanimatorevent;
//Mouse drag record:
int dragx, dragy;
//Application window:
private Frame f = new Frame("Orrery");
private GLCanvas gld =
    GLDrawableFactory.getFactory().createGLCanvas(new GLCapabilities());
GL gl = gld.getGL(); GLU glu = gld.getGLU();
//Basic interaction techniques:
Scrollbar angleofviewtechnique = new Scrollbar(Scrollbar.VERTICAL,43,2,0,43);
Button schematicnotrealistictechnique = new Button("Schematic / Realistic");
Button staticnotanimatedtechnique = new Button("Static / Animated");
Scrollbar ratetechnique = new Scrollbar(Scrollbar.VERTICAL,0,4,0,64);
MenuItem gregoriandate techniq ue = new MenuItem("Gregorian Date");
MenuItem juliandate techniq ue = new MenuItem("Julian Date");
MenuItem juliandaytechnique = new MenuItem("Julian Day");
MenuItem trackingtechnique = new MenuItem("Tracking");
MenuItem perspectivetechnique = new MenuItem("Perspective");
private double absolutecamera[], absolutelookat[], viewdistance;
Label angleofviewlabel = new Label(""), ratelabel = new Label("");
//Composite interaction techniques:
public class TimeDialog extends Dialog
    implements WindowListener, ActionListener
{
    private int year, month, type;
    private double day, hour, minute;
    private TextField
        yeartechnique = new TextField("0",10),
        monthtechnique = new TextField("0",10),
        daytechnique = new TextField("0",10),
        hourtechnique = new TextField("0",10),
        minutetechnique = new TextField("0",10);
    private Button settechnique = new Button("Set");
    public TimeDialog(Frame f, int type) //Gregorian = 0, Julian = 1, Julian day = 2
    {
        super(f,true);
        this.type = type;
        switch (type)
        {
            case 0: setTitle("Gregorian Date"); break;
            case 1: setTitle("Julian Date"); break;
            case 2: setTitle("Julian Day"); daytechnique.setText(time+"");
        }
        addWindowListener(this); settechnique.addActionListener(this);
        switch (type)
        {
            case 0: case 1:
                setLayout(new GridLayout(6,2));
                add(new Label("Year")); add(yeartechnique);
                add(new Label("Month")); add(monthtechnique);
                add(new Label("Days")); add(daytechnique);
                add(new Label("Hours")); add(hourtechnique);
                add(new Label("Minutes")); add(minutetechnique);
                add(new Label("")); add(settechnique); break;
            case 2:
                setLayout(new GridLayout(2,2));
                add(new Label("Julian Day")); add(daytechnique);
                add(new Label("")); add(settechnique);
        }
        pack(); setResizable(false); show();
    }
    public void windowOpened(WindowEvent e) {};
    public void windowClosing(WindowEvent e) {dispose();};
    public void windowClosed(WindowEvent e) {};
    public void windowIconified(WindowEvent e) {};
    public void windowDeiconified(WindowEvent e) {};
    public void windowActivated(WindowEvent e) {};
    public void windowDeactivated(WindowEvent e) {};
    public void actionPerformed(ActionEvent e)
    {
        try
        {
            switch (type)
            {
                case 0: case 1:
                    year = Integer.parseInt(yeartechnique.getText());
                    month = Integer.parseInt(monthtechnique.getText());
                    day = Double.parseDouble(daytechnique.getText());
                    hour = Double.parseDouble(hourtechnique.getText());
                    minute = Double.parseDouble(minutetechnique.getText());
                case 2:
                    day = Double.parseDouble(daytechnique.getText());
            }
        } catch (Exception x) {return;}
        if (month<=2) {year -= 1; month += 12;}
        int a = year/100, b = 2 - a + a/4;
        switch (type)
        {
            case 0: time = (int) (365.25*(year + 4716)) +
                (int) (30.6001*(month + 1)) + day + hour/24.0 + minute/1440.0 + b - 1524.5;
                break;
            case 1: time = (int) (365.25*(year + 4716)) +
                (int) (30.6001*(month + 1)) + day + hour/24d + minute/1440d - 1524.5;
        }
    }
}

```

```

        break;
        case 2: time = day;
    }
    dispose(); states = satellites.statesAt(time); gld.display();
}
}
public class TrackingDialog extends Dialog
implements WindowListener, ActionListener
{
    private String camera, lookat;
    private double
        absolutecameraoffset[] = new double[3], absolutelookatoffset[] = new double[3],
        relativecameraoffset[] = new double[3], relativelookatoffset[] = new double[3];
    private Choice cameratechnique = new Choice(), lookattechnique = new Choice();
    private TextField
        absolutecameraoffset0technique =
            new TextField(Orrery.this.absolutecameraoffset[0]+"", 5),
        absolutecameraoffset1technique =
            new TextField(Orrery.this.absolutecameraoffset[1]+"", 5),
        absolutecameraoffset2technique =
            new TextField(Orrery.this.absolutecameraoffset[2]+"", 5),
        absolutelookatoffset0technique =
            new TextField(Orrery.this.absolutelookatoffset[0]+"", 5),
        absolutelookatoffset1technique =
            new TextField(Orrery.this.absolutelookatoffset[1]+"", 5),
        absolutelookatoffset2technique =
            new TextField(Orrery.this.absolutelookatoffset[2]+"", 5),
        relativecameraoffset0technique =
            new TextField(Orrery.this.relativecameraoffset[0]+"", 5),
        relativecameraoffset1technique =
            new TextField(Orrery.this.relativecameraoffset[1]+"", 5),
        relativecameraoffset2technique =
            new TextField(Orrery.this.relativecameraoffset[2]+"", 5),
        relativelookatoffset0technique =
            new TextField(Orrery.this.relativelookatoffset[0]+"", 5),
        relativelookatoffset1technique =
            new TextField(Orrery.this.relativelookatoffset[1]+"", 5),
        relativelookatoffset2technique =
            new TextField(Orrery.this.relativelookatoffset[2]+"", 5);
    private Button settechnique = new Button("Set");
    public TrackingDialog(Frame f)
    {
        super(f, "Tracking", true);
        addWindowListener(this); settechnique.addActionListener(this);
        cameratechnique.add("Origin");
        for (int i = 0; i < satelliteindex.size(); i++)
            cameratechnique.add((String) (satelliteindex.elementAt(i)));
        cameratechnique.select(Orrery.this.camera);
        lookattechnique.add("Origin");
        for (int i = 0; i < satelliteindex.size(); i++)
            lookattechnique.add((String) (satelliteindex.elementAt(i)));
        lookattechnique.select(Orrery.this.lookat);
        setLayout(new GridLayout(7, 4));
        add(new Label("Camera")); add(cameratechnique);
        add(new Label("Lookat")); add(lookattechnique);
        add(new Label(""));
        add(new Label("x in AU")); add(new Label("y in AU")); add(new Label("z in AU"));
        add(new Label("Absolute camera offset")); add(absolutecameraoffset0technique);
        add(absolutecameraoffset1technique); add(absolutecameraoffset2technique);
        add(new Label("Absolute lookat offset")); add(absolutelookatoffset0technique);
        add(absolutelookatoffset1technique); add(absolutelookatoffset2technique);
        add(new Label("Relative camera offset")); add(relativecameraoffset0technique);
        add(relativecameraoffset1technique); add(relativecameraoffset2technique);
        add(new Label("Relative lookat offset")); add(relativelookatoffset0technique);
        add(relativelookatoffset1technique); add(relativelookatoffset2technique);
        add(new Label("")); add(new Label("")); add(new Label("")); add(settechnique);
        pack(); setResizable(false); show();
    }
    public void windowOpened(WindowEvent e) {};
    public void windowClosing(WindowEvent e) {dispose();};
    public void windowClosed(WindowEvent e) {};
    public void windowIconified(WindowEvent e) {};
    public void windowDeiconified(WindowEvent e) {};
    public void windowActivated(WindowEvent e) {};
    public void windowDeactivated(WindowEvent e) {};
    public void actionPerformed(ActionEvent e)
    {
        try
        {
            camera = cameratechnique.getSelectedItem();
            lookat = lookattechnique.getSelectedItem();
            absolutecameraoffset[0] = Double.parseDouble(
                absolutecameraoffset0technique.getText());
            absolutecameraoffset[1] = Double.parseDouble(
                absolutecameraoffset1technique.getText());
            absolutecameraoffset[2] = Double.parseDouble(
                absolutecameraoffset2technique.getText());
            absolutelookatoffset[0] = Double.parseDouble(
                absolutelookatoffset0technique.getText());
            absolutelookatoffset[1] = Double.parseDouble(
                absolutelookatoffset1technique.getText());

```

```

        absolutelookatoffset[2] = Double.parseDouble(
            absolutelookatoffset2technique.getText());
        relativecameraoffset[0] = Double.parseDouble(
            relativecameraoffset0technique.getText());
        relativecameraoffset[1] = Double.parseDouble(
            relativecameraoffset1technique.getText());
        relativecameraoffset[2] = Double.parseDouble(
            relativecameraoffset2technique.getText());
        relativelookatoffset[0] = Double.parseDouble(
            relativelookatoffset0technique.getText());
        relativelookatoffset[1] = Double.parseDouble(
            relativelookatoffset1technique.getText());
        relativelookatoffset[2] = Double.parseDouble(
            relativelookatoffset2technique.getText());
    } catch (Exception x) {return;}
    Orrery.this.camera = camera; Orrery.this.lookat = lookat;
    Orrery.this.absolutecameraoffset = absolutecameraoffset;
    Orrery.this.absolutelookatoffset = absolutelookatoffset;
    Orrery.this.relativecameraoffset = relativecameraoffset;
    Orrery.this.relativelookatoffset = relativelookatoffset;
    dispose(); gld.display();
}
}
public class PerspectiveDialog extends Dialog
implements WindowListener, ActionListener
{
    private double
    viewrotate,
    angularcameraoffset[] = new double[2],
    angularlookatoffset[] = new double[2];
    private TextField
    viewrotatetechnique = new TextField(Orrery.this.viewrotate+"",5),
    angularcameraoffset0technique =
        new TextField(Orrery.this.angularcameraoffset[0]+"",5),
    angularcameraoffset1technique =
        new TextField(Orrery.this.angularcameraoffset[1]+"",5),
    angularlookatoffset0technique =
        new TextField(Orrery.this.angularlookatoffset[0]+"",5),
    angularlookatoffset1technique =
        new TextField(Orrery.this.angularlookatoffset[1]+"",5);
    private Button settechnique = new Button("Set");
    public PerspectiveDialog(Frame f)
    {
        super(f,"Perspective",true);
        addWindowListener(this); settechnique.addActionListener(this);
        setLayout(new GridLayout(5,3));
        add(new Label("Rotate")); add(viewrotatetechnique); add(new Label(""));
        add(new Label("")); add(new Label("longitude in degrees"));
        add(new Label("latitude in degrees"));
        add(new Label("Angular camera offset")); add(angularcameraoffset0technique);
        add(angularcameraoffset1technique);
        add(new Label("Angular lookat offset")); add(angularlookatoffset0technique);
        add(angularlookatoffset1technique);
        add(new Label("")); add(new Label("")); add(settechnique);
        pack(); setResizable(false); show();
    }
    public void windowOpened(WindowEvent e) {};
    public void windowClosing(WindowEvent e) {dispose();};
    public void windowClosed(WindowEvent e) {};
    public void windowIconified(WindowEvent e) {};
    public void windowDeiconified(WindowEvent e) {};
    public void windowActivated(WindowEvent e) {};
    public void windowDeactivated(WindowEvent e) {};
    public void actionPerformed(ActionEvent e)
    {
        try
        {
            viewrotate = Double.parseDouble(
                viewrotatetechnique.getText());
            angularcameraoffset[0] = Double.parseDouble(
                angularcameraoffset0technique.getText());
            angularcameraoffset[1] = Double.parseDouble(
                angularcameraoffset1technique.getText());
            angularlookatoffset[0] = Double.parseDouble(
                angularlookatoffset0technique.getText());
            angularlookatoffset[1] = Double.parseDouble(
                angularlookatoffset1technique.getText());
        } catch (Exception x) {System.out.println(x);}
        Orrery.this.viewrotate = viewrotate;
        Orrery.this.angularcameraoffset = angularcameraoffset;
        Orrery.this.angularlookatoffset = angularlookatoffset;
        dispose(); gld.display();
    }
}
//Listeners:
public void init(GLDrawable gld)
{
    Vector satelliteset = new Vector();
    try
    {
        for (int i = 0; i < kepleriansatelliteindex.size(); i++)
            satelliteset.add(
                new KeplerianSatellite((String)kepleriansatelliteindex.elementAt(i), gld));
    }
}

```

```

    for (int i = 0; i < sunsatelliteindex.size(); i++)
        satellitese.set.add(
            new SunSatellite((String)sunsatelliteindex.elementAt(i), gld));
    for (int i = 0; i < moonsatelliteindex.size(); i++)
        satellitese.set.add(
            new MoonSatellite((String)moonsatelliteindex.elementAt(i), gld));
    for (int i = 0; i < plutosatelliteindex.size(); i++)
        satellitese.set.add(
            new PlutoSatellite((String)plutosatelliteindex.elementAt(i), gld));
} catch (Exception e) {System.out.println(e);}
satellites = new Satellites(satellitese.set);
states = satellites.statesAt(time);
gl.glClearColor(0,0,0,0);
gl.glCullFace(GL.GL_BACK);
gl.glLightModelfv(GL.GL_LIGHT_MODEL_AMBIENT, starlight);
gl.glLightModeli(GL.GL_LIGHT_MODEL_LOCAL_VIEWER, 1);
gl.glBlendFunc(GL.GL_SRC_ALPHA, GL.GL_ONE_MINUS_SRC_ALPHA);
gl.glEnable(GL.GL_DEPTH_TEST); gl.glEnable(GL.GL_CULL_FACE);
gl.glEnable(GL.GL_LIGHT0); gl.glEnable(GL.GL_POINT_SMOOTH);
gl.glEnable(GL.GL_LINE_SMOOTH);
if (schematicnotrealistic) gl.glEnable(GL.GL_BLEND); else gl.glEnable(GL.GL_LIGHTING);
gl.glFlush();
}
public void reshape(GLDrawable gldrawable, int x, int y, int w, int h) {}
public void display(GLDrawable gldrawable)
{
    if (camera!=null&&!camera.equals("Origin"))
    {
        State tempstate = states.extract(camera);
        double[] temparray =
        {
            tempstate.position[0] + absolutecameraoffset[0],
            tempstate.position[1] + absolutecameraoffset[1],
            tempstate.position[2] + absolutecameraoffset[2]
        };
        while (!(tempstate.parent.equals("Origin")))
        {
            tempstate = states.extract(tempstate.parent);
            temparray[0] += tempstate.position[0];
            temparray[1] += tempstate.position[1];
            temparray[2] += tempstate.position[2];
        }
        absolutecamera = temparray;
    }
    else absolutecamera = absolutecameraoffset;
    if (lookat!=null&&!lookat.equals("Origin"))
    {
        State tempstate = states.extract(lookat);
        double[] temparray =
        {
            tempstate.position[0] + absolutelookatoffset[0],
            tempstate.position[1] + absolutelookatoffset[1],
            tempstate.position[2] + absolutelookatoffset[2]
        };
        while (!(tempstate.parent.equals("Origin")))
        {
            tempstate = states.extract(tempstate.parent);
            temparray[0] += tempstate.position[0];
            temparray[1] += tempstate.position[1];
            temparray[2] += tempstate.position[2];
        }
        absolutelookat = temparray;
    }
    else absolutelookat = absolutelookatoffset;
    viewdistance =
        Math.sqrt(
            (absolutecamera[0]-absolutelookat[0])*(absolutecamera[0]-absolutelookat[0])+
            (absolutecamera[1]-absolutelookat[1])*(absolutecamera[1]-absolutelookat[1])+
            (absolutecamera[2]-absolutelookat[2])*(absolutecamera[2]-absolutelookat[2])
        );
    gl.glMatrixMode(GL.GL_PROJECTION); gl.glLoadIdentity();
    if (schematicnotrealistic)
    {
        glu.gluPerspective
        (
            angleofview,
            (float)gld.getWidth()/ (float)gld.getHeight(),
            0.0001f,
            100
        );
    }
    else
    {
        glu.gluPerspective
        (
            angleofview,
            (float)gld.getWidth()/ (float)gld.getHeight(),
            viewdistance/25,
            viewdistance*4
        );
    }
}
gl.glMatrixMode(GL.GL_MODELVIEW); gl.glLoadIdentity();
//Apply relative offsets:
glu.gluLookAt
(
    relativecameraoffset[0],
    relativecameraoffset[1],
    relativecameraoffset[2],
    relativelookatoffset[0],
    relativelookatoffset[1],
    relativelookatoffset[2]
);

```

```

        relativelookatoffset[1],
        -viewdistance + relativelookatoffset[2],
        0,1,0
    );
    //Apply angular camera offsets:
    gl.glTranslated(0,0,-viewdistance);
    gl.glRotated(angularcameraoffset[1],1,0,0);
    gl.glRotated(-angularcameraoffset[0],0,1,0);
    gl.glTranslated(0,0,viewdistance);
    //Apply angular lookat offsets:
    gl.glRotated(-angularlookatoffset[1],1,0,0);
    gl.glRotated(-angularlookatoffset[0],0,1,0);
    //Apply view rotate (the image rotates clockwise for positive viewrotate):
    gl.glRotated(-viewrotate,0,0,1);
    //Apply viewpoint:
    glu.gluLookAt
    (
        absolutecamera[0],absolutecamera[1],absolutecamera[2],
        absolutelookat[0],absolutelookat[1],absolutelookat[2],
        0,0,1
    );
    //Place light at world origin:
    gl.glLightfv(GL.GL_LIGHT0, GL.GL_POSITION, light_position);
    gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
    states.toDisplay(gldrawable, schematicnotrealistic); gl.glFlush();
}
public void displayChanged(GLDrawable gld, boolean modechanged, boolean devicechanged){}
//ActionListener:
public void actionPerformed(ActionEvent e)
{
    if (e.getSource() == animator)
    {
        //We ensure that the displayed frame rate does not drop below
        //0.8 the requested animation rate. Note that we assume
        //that the displayed rate is on average 0.85 the requested rate.
        boolean changed = false;
        lastanimatorevent = currentanimatorevent;
        currentanimatorevent = System.currentTimeMillis();
        //
        System.out.println( //
        // "animator rate = "+1000.0/(currentanimatorevent-lastanimatorevent)); //
        if
        (
            1000.0/(currentanimatorevent-lastanimatorevent) <
            0.8*requestedframespersecond
        )
        {
            requestedframespersecond--; changed = true; }
        if
        (
            1000.0/(currentanimatorevent-lastanimatorevent) >
            0.9*requestedframespersecond
        )
        {
            requestedframespersecond++; changed = true; }
        //
        System.out.println("framespersecond = "+requestedframespersecond); //
        if (changed)
        {
            timejump = rate/(86400f*0.85*requestedframespersecond);
            animator.stop();
            animator = new Timer((int)(1000/requestedframespersecond), this);
            animator.start();
        }
        states = satellites.statesAfter(time, timejump, states); time += timejump;
        gl.glFlush(); gld.display();
    }
    if (e.getSource() == gregoriandatetechnique) new TimeDialog(f,0);
    if (e.getSource() == juliandatetechnique) new TimeDialog(f,1);
    if (e.getSource() == juliandaytechnique) new TimeDialog(f,2);
    if (e.getSource() == trackingtechnique) new TrackingDialog(f);
    if (e.getSource() == perspectivetechnique) new PerspectiveDialog(f);
    if (e.getSource() == schematicnotrealistictechnique)
    {
        schematicnotrealistic = !schematicnotrealistic;
        if (schematicnotrealistic)
        {
            gl.glDisable(GL.GL_LIGHTING); gl.glEnable(GL.GL_BLEND); }
        else
        {
            gl.glDisable(GL.GL_BLEND); gl.glEnable(GL.GL_LIGHTING); }
        gl.glFlush(); gld.display();
    }
    if (e.getSource() == staticnotanimatedtechnique)
    {
        staticnotanimated = !staticnotanimated;
        if (!staticnotanimated)
        {
            animator = new Timer(1000/(int)requestedframespersecond, this); animator.start();
        }
        else animator.stop();
    }
}
public void adjustmentValueChanged(AdjustmentEvent e)
{
    if (e.getSource() == angleofviewtechnique)
    {
        angleofview = 0.0001*Math.pow(1.41421356,e.getValue());
        angleofviewlabel.setText(angleofview+"");
    }
    if (e.getSource() == ratetechnique)
    {
        rate = Math.pow(1.41421356,e.getValue());
        //We assume that the actual frame rate will be about 0.85 the
        //requested frame rate:
    }
}

```

```

        timejump = rate/(86400f*0.85*requestedframespersecond);
        if (!staticnotanimated)
        {
            animator.stop();
            animator = new Timer(1000/(int)requestedframespersecond, this);
            animator.start();
        }
        ratelabel.setText(rate+"");
    }
    gld.display();
}
public void mouseClicked(MouseEvent e) {}
public void mousePressed(MouseEvent e) {dragx = e.getX(); dragy = e.getY();}
public void mouseReleased(MouseEvent e) {}
public void mouseEntered(MouseEvent e){} public void mouseExited(MouseEvent e){}
public void mouseMoved(MouseEvent e){}
public void mouseDragged(MouseEvent e)
{
    angularcameraoffset[0] += (dragx - e.getX())/5.0;
    angularcameraoffset[1] -= (dragy - e.getY())/5.0;
    dragx = e.getX(); dragy = e.getY();
    gld.display();
}
}
public Orrery()
{
    //Build model:
    try{
        LineNumberReader r = new LineNumberReader(new FileReader("Variables"));
        time = Read.readdouble(r,"time");
        camera = Read.readString(r,"camera"); lookat = Read.readString(r,"lookat");
        absolutecameraoffset = Read.readdoubleArray(r,"absolutecameraoffset",3);
        absolutelookatoffset = Read.readdoubleArray(r,"absolutelookatoffset",3);
        relativecameraoffset = Read.readdoubleArray(r,"relativecameraoffset",3);
        relativelookatoffset = Read.readdoubleArray(r,"relativelookatoffset",3);
        viewrotate = Read.readdouble(r,"viewrotate");
        angularcameraoffset = Read.readdoubleArray(r,"angularcameraoffset",2);
        angularlookatoffset = Read.readdoubleArray(r,"angularlookatoffset",2);
        angleofview = Read.readdouble(r,"angleofview");
        schematicnotrealistic = Read.readboolean(r,"schematicnotrealistic");
        staticnotanimated = Read.readboolean(r,"staticnotanimated");
        rate = Read.readdouble(r,"rate");
        kepleriansatelliteindex = Read.readStringVector(r,"kepleriansatellitenames");
        sunsatelliteindex = Read.readStringVector(r,"sunsatellitenames");
        moonsatelliteindex = Read.readStringVector(r,"moonsatellitenames");
        plutosatelliteindex = Read.readStringVector(r,"plutosatellitenames");
        satelliteindex.addAll(kepleriansatelliteindex);
        satelliteindex.addAll(sunsatelliteindex);
        satelliteindex.addAll(moonsatelliteindex);
        satelliteindex.addAll(plutosatelliteindex);
    } catch (Exception e) {System.out.println(e);}
    //Listen:
    gld.addGLEventListener(this);
    gld.addMouseListener(this);
    gld.addMouseMotionListener(this);
    gregoriandatetechnique.addActionListener(this);
    juliandatetechnique.addActionListener(this);
    juliandaytechnique.addActionListener(this);
    trackingtechnique.addActionListener(this);
    perspectivetechnique.addActionListener(this);
    angleofviewtechnique.addAdjustmentListener(this);
    schematicnotrealistictechnique.addActionListener(this);
    staticnotanimatedtechnique.addActionListener(this);
    ratetechnique.addAdjustmentListener(this);
    //Build GUI:
    gld.setSize(800,600);
    Menu t = new Menu("Set Time");
        t.add(gregoriandatetechnique); t.add(juliandatetechnique); t.add(juliandaytechnique);
    Menu v = new Menu("Set View");
        v.add(trackingtechnique); v.add(perspectivetechnique);
    MenuBar mb = new MenuBar();
        mb.add(t); mb.add(v);
    angleofviewlabel.setText(angleofview+"");
    ratelabel.setText(rate+"");
    Panel p = new Panel(new GridLayout(1,7));
        p.add(new Label("Angle of view in deg.));
        p.add(angleofviewlabel);
        p.add(schematicnotrealistictechnique);
        p.add(new Label("Drag to rotate", Label.CENTER));
        p.add(staticnotanimatedtechnique);
        p.add(ratelabel);
        p.add(new Label("Rate"));
    f.addWindowListener(this);
    f.setMenuBar(mb);
    f.setLayout(new BorderLayout());
        f.add(p, BorderLayout.PAGE_END); f.add(gld, BorderLayout.CENTER);
        f.add(angleofviewtechnique, BorderLayout.LINE_START);
        f.add(ratetechnique, BorderLayout.LINE_END);
    f.pack();
    f.show();
}
}

```

```

    public static void main(String[] args) {new Orrery();}
    public void windowClosing(WindowEvent e){System.exit(0);}
}

```

Java code for the Satellites class

```

import java.util.*;
public class Satellites
{
    final Map m;
    public Satellites(Collection c)
    {
        m = new TreeMap();
        Iterator i = c.iterator();
        Satellite s;
        while (i.hasNext())
        {
            s = (Satellite)i.next();
            m.put(s.name,s);
        }
    }
    public States statesAt(double time)
    {
        Collection c;
        Iterator i;
        //Try to build set of states at time t:
        c = new Vector();
        i = m.values().iterator();
        while (i.hasNext())
            c.add(((Satellite)i.next()).stateAt(time));
        //Check if it was possible (Satellites
        //return the latest time not after the requested time
        //that they can give exactly):
        double minimum = time;
        i = c.iterator();
        while (i.hasNext())
        {
            State s = (State)i.next();
            if (s.time < minimum) minimum = s.time;
        }
        States s = new States(c);
        //If it was not possible, iterate up from the minimum time:
        if (minimum < time)
        {
            double currenttime = minimum;
            while (currenttime < time)
            {
                s = statesAfter(
                    currenttime, 1/24.0, s
                );
                currenttime += 1/24.0;
            }
        }
        return s;
    }
    public States statesAfter(double time, double quantum, States s)
    {
        Collection c;
        Iterator i;
        c = new Vector();
        i = m.values().iterator();
        while (i.hasNext())
            c.add(((Satellite)i.next()).stateAfter(time, quantum, s));
        return new States(c);
    }
}

```

Java code for the Satellite abstract class

```

import net.java.games.jogl.*;
import java.io.*;
public abstract class Satellite
{
    //Constants:
    public String name;
    public String parent;
    public double mass;
    public double spherediameter;
    public double ringwidth;
    public float ambient[] = new float[4];
    public float diffuse[] = new float[4];
    public float specular[] = new float[4];
    public float shininess;
    public float emission[] = new float[4];
    public GLDrawable gld;
    //Interrogation:
    public abstract State stateAt(double time);
    public abstract State stateAfter(double time, double quantum, States s);
    byte[] raw;
    int width = 200, height = 100;
    int texture[] = {0}, slist, rlist;
    public void makeLists()
    {
        GL gl = gld.getGL();
        GLU glu = gld.getGLU();
        raw = new byte[width*height*3];
    }
}

```

```

try{
    FileInputStream input = new FileInputStream(name+".raw");
    input.read(raw,0,width*height*3);
} catch (Exception e) {}
gl.glGenTextures(1, texture);
gl.glBindTexture(GL.GL_TEXTURE_2D, texture[0]);
glu.gluBuild2DMipmaps
(GL.GL_TEXTURE_2D, GL.GL_RGBA, width, height, GL.GL_RGB, GL.GL_UNSIGNED_BYTE, raw);
//Make schematic list:
GLUquadric gluq = glu.gluNewQuadric();
slist = gl.glGenLists(1);
gl.glNewList(slist, GL.GL_COMPILE);
    if (spherediameter != 0)
    {
        gl.glPointSize(10); gl.glColor4fv(ambient);
        gl.glBegin(GL.GL_POINTS); gl.glVertex3f(0,0,0); gl.glEnd();
        glu.gluQuadricDrawStyle(gluq, GLU.GLU_FILL);
        glu.gluSphere(gluq, spherediameter/2, 24, 24);
    }
gl.glEndList();
glu.gluDeleteQuadric(gluq);
//Make realistic list:
gluq = glu.gluNewQuadric();
rlist = gl.glGenLists(1);
gl.glNewList(rlist, GL.GL_COMPILE);
    gl.glEnable(GL.GL_TEXTURE_2D);
    glu.gluQuadricTexture(gluq, true);
    gl.glMaterialfv(GL.GL_FRONT, GL.GL_AMBIENT, ambient);
    gl.glMaterialfv(GL.GL_FRONT, GL.GL_SPECULAR, specular);
    gl.glMaterialfv(GL.GL_FRONT, GL.GL_SPECULAR, specular);
    gl.glMaterialf(GL.GL_FRONT, GL.GL_SHININESS, shininess);
    gl.glMaterialfv(GL.GL_FRONT, GL.GL_EMISSION, emission);
    if (spherediameter != 0)
    {
        gl.glPointSize(1);
        gl.glColor4fv(ambient);
        gl.glBegin(GL.GL_POINTS);
            gl.glVertex3f(0,0,0);
        gl.glEnd();
        gl.glEnable(GL.GL_LIGHTING);
        glu.gluQuadricOrientation(gluq, GLU.GLU_OUTSIDE);
        glu.gluSphere(gluq, spherediameter/2, 128, 24);
        gl.glDisable(GL.GL_LIGHTING);
    }
    gl.glDisable(GL.GL_TEXTURE_2D);
gl.glEndList();
glu.gluDeleteQuadric(gluq);
}
}
}

```

Java code for the States class

```

import java.util.*;
import net.java.games.jogl.*;
public class States
{
    //Constants:
    final Map m;
    final Map n;
    //Interrogation:
    public State extract(String name)
    {
        return (State)n.get(name);
    }
    //Display:
    void toDisplay(GLDrawable gld, boolean schematic)
    {
        //Display in immediate mode, restoring the coordinate system:
        renderTraverse("Origin", gld, schematic);
    }
    private void renderTraverse(String parent, GLDrawable gld, boolean schematicnotrealistic)
    {
        GL gl = gld.getGL(); GLU glu = gld.getGLU();
        Vector v = (Vector)m.get(parent);
        if (v == null) return;
        for(int j = 0; j < v.size(); j++)
        {
            State state = (State)v.get(j);
            //Draw the planet (remember that the coordinate system is restored):
            state.toDisplay(schematicnotrealistic);
            gl.glPushMatrix(); // - Save position.
            //Move to planet:
            gl.glTranslated
            (
                state.position[0],
                state.position[1],
                state.position[2]
            );
            renderTraverse(((State)v.get(j)).name, gld, schematicnotrealistic);
            // - Render all children.
            gl.glPopMatrix(); // - Move away from planet.
        }
    }
}
//Construction:

```

```

public States(Collection c)
{
    m = new TreeMap();
    Iterator i = c.iterator();
    State s;
    Vector v;
    while (i.hasNext())
    {
        s = (State)i.next();
        if (m.containsKey(s.parent))
            v = (Vector)m.get(s.parent);
        else
        {
            v = new Vector();
            m.put(s.parent, v);
        }
        v.add(s);
    }
    n = new TreeMap();
    Iterator j = c.iterator();
    State q;
    while (j.hasNext())
    {
        q = ((State)j.next());
        n.put(q.name, q);
    }
}
}

```

Java code for the State class

```

import net.java.games.jogl.*;
public class State
{
    //Constants:
    public final String name;
    public final String parent;
    public final double mass;
    public final double spherediameter;
    public final double ringwidth;
    public final float ambient[], diffuse[], specular[], shininess, emission[];
    public final double time;
    public final double[] position, velocity;
    public final GLDrawable gld;
    public final int texture, rlist, rlist;
    //Display:
    public void toDisplay(boolean schematicnotrealistic)
    {
        GL gl = gld.getGL(); GLU glu = gld.getGLU();
        GLUquadric gluq = glu.gluNewQuadric();
        gl.glPushMatrix();
        gl.glPushMatrix();
        double normposition[] = normalise(position);
        double normvelocity[] = normalise(velocity);
        double row2[] = crossproduct(normposition, normvelocity);
        double row1[] = crossproduct(row2, normposition);
        double matrix[] = {
            normposition[0],      normposition[1],      normposition[2],      0,
            row1[0],              row1[1],          row1[2],              0,
            row2[0],              row2[1],          row2[2],              0,
            0,                    0,                0,                    1
        };
        gl.glMultMatrixd(matrix);
        double xyzdistance =
            Math.sqrt
            (
                position[0]*position[0] +
                position[1]*position[1] +
                position[2]*position[2]
            );
        if (schematicnotrealistic)
        {
            glu.gluQuadricDrawStyle(gluq, GLU.GLU_SILHOUETTE);
            gl.glColor4fv(ambient);
            glu.gluDisk
            (
                gluq, xyzdistance, xyzdistance, 128, 1 );
        }
        else{
            glu.gluQuadricNormals(gluq, GL.GL_SMOOTH);
            gl.glEnable(GL.GL_TEXTURE_2D);
            gl.glBindTexture(GL.GL_TEXTURE_2D, texture);
            glu.gluQuadricTexture(gluq, true);
            gl.glMaterialfv(GL.GL_FRONT, GL.GL_AMBIENT, ambient);
            gl.glMaterialfv(GL.GL_FRONT, GL.GL_SPECULAR, specular);
            gl.glMaterialfv(GL.GL_FRONT, GL.GL_SPECULAR, specular);
            gl.glMaterialfv(GL.GL_FRONT, GL.GL_SHININESS, shininess);
            gl.glMaterialfv(GL.GL_FRONT, GL.GL_EMISSION, emission);
            gl.glEnable(GL.GL_LIGHTING);
            glu.gluDisk
            (
                gluq, xyzdistance-ringwidth/2, xyzdistance+ringwidth/2, 64, 1 );
            glu.gluQuadricOrientation(gluq, GLU.GLU_INSIDE);
            glu.gluDisk
            (
                gluq, xyzdistance-ringwidth/2, xyzdistance+ringwidth/2, 64, 1 );
            gl.glDisable(GL.GL_LIGHTING);
        }
    }
}

```

```

    }
    gl.glPopMatrix();
    gl.glTranslated
    ( position[0], position[1], position[2] );
    if (spherediameter != 0)
    {   if (schematicnotrealistic) gl.glCallList(slist);
        else gl.glCallList(rlist);
    }
    gl.glPopMatrix();
    glu.gluDeleteQuadric(gluq);
    gl.glDisable(GL.GL_TEXTURE_2D);
}
//Construction:
public State
(   Satellite s,
    double time,
    double[] position,
    double[] velocity,
    GLDrawable gld,
    int texture,
    int slist,
    int rlist
)
{   this.name = s.name;
    this.parent = s.parent;
    this.mass = s.mass;
    this.spherediameter = s.spherediameter;
    this.ringwidth = s.ringwidth;
    this.ambient = s.ambient;
    this.diffuse = s.diffuse;
    this.specular = s.specular;
    this.shininess = s.shininess;
    this.emission = s.emission;
    this.time = time;
    this.position = position;
    this.velocity = velocity;
    this.gld = gld;
    this.texture = texture;
    this.slist = slist;
    this.rlist = rlist;
}
private static double[] crossproduct(double[] a, double[] b)
{   double c[] = new double[3];
    c[0]=a[1]*b[2]-a[2]*b[1];
    c[1]=-a[0]*b[2] + a[2]*b[0];
    c[2]=a[0]*b[1] - a[1]*b[0];
    return c;
}
private static double[] normalise(double[] a)
{   double c[] = new double[3];
    double length = Math.sqrt(a[0]*a[0] + a[1]*a[1] + a[2]*a[2]);
    c[0]=a[0]/length;
    c[1]=a[1]/length;
    c[2]=a[2]/length;
    return c;
}
}
}

```

Java functions for reading labelled values from input streams

```

import java.util.*;
import java.io.*;
public class Read
{   static public String readString(LineNumberReader r, String declaration)
    throws Exception
    {   if (!r.readLine().equals(declaration))
        throw new Exception(declaration+" NOT FOUND AT LINE "+r.getLineNumber());
        return r.readLine();
    }
    static public boolean readboolean(LineNumberReader r, String declaration)
    throws Exception
    {   if (!r.readLine().equals(declaration))
        throw new Exception(declaration+" NOT FOUND AT LINE "+r.getLineNumber());
        return (new Boolean(r.readLine())).booleanValue();
    }
    public static float readfloat(LineNumberReader r, String declaration)
    throws Exception
    {   if (!r.readLine().equals(declaration))
        throw new Exception(declaration+" NOT FOUND AT LINE "+r.getLineNumber());
        return Float.parseFloat(r.readLine());
    }
    public static float[] readfloatArray(LineNumberReader r, String declaration, int n)
    throws Exception
    {   if (!r.readLine().equals(declaration))
        throw new Exception(declaration+" NOT FOUND AT LINE "+r.getLineNumber());
        float a[] = new float[n];
    }
}

```

```

        for (int i = 0; i < n; i++) a[i] = Float.parseFloat(r.readLine());
        return a;
    }
    static public double readdouble(LineNumberReader r, String declaration)
        throws Exception
    {
        if (!r.readLine().equals(declaration))
            throw new Exception(declaration+" NOT FOUND AT LINE "+r.getLineNumber());
        return Double.parseDouble(r.readLine());
    }
    static public double[] readdoubleArray(LineNumberReader r, String declaration, int n)
        throws Exception
    {
        if (!r.readLine().equals(declaration))
            throw new Exception(declaration+" NOT FOUND AT LINE "+r.getLineNumber());
        double a[] = new double[n];
        for (int i = 0; i < n; i++) a[i] = Double.parseDouble(r.readLine());
        return a;
    }
    static public Vector readDoubleVector(LineNumberReader r, String declaration)
        throws Exception
    {
        if (!r.readLine().equals(declaration))
            throw new Exception(declaration+" NOT FOUND AT LINE "+r.getLineNumber());
        Vector v = new Vector();
        if(!r.readLine().equals("BEGIN"))
            throw new Exception("BEGIN NOT FOUND AT LINE "+r.getLineNumber());
        String line;
        while (!(line = r.readLine()).equals("END")) v.add(new Double(line));
        return v;
    }
    static public Vector readStringVector(LineNumberReader r, String declaration)
        throws Exception
    {
        if (!r.readLine().equals(declaration))
            throw new Exception(declaration+" NOT FOUND AT LINE "+r.getLineNumber());
        Vector v = new Vector();
        if(!r.readLine().equals("BEGIN"))
            throw new Exception("BEGIN NOT FOUND AT LINE "+r.getLineNumber());
        String line;
        while (!(line = r.readLine()).equals("END")) v.add(line);
        return v;
    }
    public static int[][] readintMatrix(LineNumberReader r, String declaration, int m, int n)
        throws Exception
    {
        if (!r.readLine().equals(declaration))
            throw new Exception(declaration+" NOT FOUND AT LINE "+r.getLineNumber());
        int a[][] = new int[m][n];
        for (int i = 0; i < m; i++)
            for (int j = 0; j < n; j++) a[i][j] = Integer.parseInt(r.readLine());
        return a;
    }
}

```

Java code for KeplerianSatellite:

```

import java.util.*;
import java.io.*;
import net.java.games.jogl.*;
class KeplerianSatellite extends Satellite {
    private Vector inclinationlist,longitudeofascendingnodelist, semimajoraxislist,
        eccentricitylist,longitudeofperiastronlist, meanlongitudelist;
    public KeplerianSatellite(String path, GLDrawable gld) throws Exception
    {
        LineNumberReader r = new LineNumberReader(new FileReader(path));
        name = r.readLine(); parent = r.readLine(); mass = Read.readdouble(r,"mass");
        spherediameter = Read.readdouble(r,"spherediameter");
        ringwidth = Read.readdouble(r,"ringwidth");
        ambient = Read.readfloatArray(r,"ambient",4); diffuse = Read.readfloatArray(r,"diffuse",4);
        specular = Read.readfloatArray(r,"specular",4); shininess = Read.readfloat(r,"shininess");
        emission = Read.readfloatArray(r,"emission",4);
        inclinationlist = Read.readDoubleVector(r,"inclinationlist");
        longitudeofascendingnodelist = Read.readDoubleVector(r,"longitudeofascendingnodelist");
        semimajoraxislist = Read.readDoubleVector(r,"semimajoraxislist");
        eccentricitylist = Read.readDoubleVector(r,"eccentricitylist");
        longitudeofperiastronlist = Read.readDoubleVector(r,"longitudeofperiastronlist");
        meanlongitudelist = Read.readDoubleVector(r,"meanlongitudelist");
        this.gld = gld;
        makeLists();
    }
    public State stateAt(double time)
    {
        double[] position1 = positionAt((time - 2451545)/36525);
        double[] position2 = positionAt((time + 1.0/24.0 - 2451545)/36525);
        double[] velocity = {
            position2[0] - position1[0], position2[1] - position1[1],
            position2[2] - position1[2],
        };
        return new State(this, time, position1, velocity, gld, texture[0], slist, rlist);
    }
    public State stateAfter(double time, double quantum, States s)
    {
        double steppedtime = time + quantum;
        double[] position1 = positionAt((steppedtime - 2451545)/36525);
        double steppedtimeplushour = steppedtime + 1.0/24.0;
    }
}

```

```

    double[] position2 = positionAt((steppedtimeplushour - 2451545)/36525);
    double[] velocity = {position2[0]-position1[0], position2[1]-position1[1],
                        position2[2]-position1[2],
                        };
    return new State(this, steppedtime, position1, velocity, gld, texture[0], slist, rlist);
}
private double[] positionAt(double time)
{
    double x,y,z,
        inclination, longitudeofascendingnode, semimajoraxis, eccentricity,
        longitudeofperiastron, argumentofperiastron, meanlongitude, meananomaly,
        eccentricanomaly, orbitalx, orbitaly, trueanomaly, distance, truelongitude;
    inclination=normalise(sumPolynomial(time, inclinationlist));
    longitudeofascendingnode=normalise(sumPolynomial(time, longitudeofascendingnodelist));
    semimajoraxis=sumPolynomial(time, semimajoraxislist);
    eccentricity=sumPolynomial(time, eccentricitylist);
    longitudeofperiastron=normalise(sumPolynomial(time, longitudeofperiastronlist));
    meanlongitude=normalise(sumPolynomial(time, meanlongitudelist));
    meananomaly = normalise(meanlongitude - longitudeofperiastron);
    argumentofperiastron = normalise(longitudeofperiastron - longitudeofascendingnode);
    eccentricanomaly=eccentricAnomaly(meananomaly, eccentricity);
    orbitalx=semimajoraxis*(dcos(eccentricanomaly)-eccentricity);
    orbitaly=semimajoraxis*Math.sqrt(1-eccentricity*eccentricity)*dsin(eccentricanomaly);
    trueanomaly=atan2(orbitaly, orbitalx);
    distance=Math.sqrt(orbitalx*orbitalx+orbitaly*orbitaly);
    truelongitude=normalise(trueanomaly+argumentofperiastron);
    x=distance*(
        dcos(longitudeofascendingnode)*dcos(truelongitude)-
        dsin(longitudeofascendingnode)*dsin(truelongitude)*
        dcos(inclination)
    );
    y=distance*(
        dsin(longitudeofascendingnode)*dcos(truelongitude)+
        dcos(longitudeofascendingnode)*dsin(truelongitude)*
        dcos(inclination)
    );
    z=distance*dsin(truelongitude)*dsin(inclination);
    return new double[] {x, y, z};
}
private double sumPolynomial(double time, Vector coefficientlist)
{
    //Horner's rule. We require that length > 1:
    double sum = ((Double)coefficientlist.get(coefficientlist.size()-1)).doubleValue();
    for (int i = coefficientlist.size()-2; 0<=i; i--)
        sum = sum*time + ((Double)coefficientlist.get(i)).doubleValue();
    return sum;
}
private double dsin(double degrees) {return Math.sin(Math.toRadians(degrees));}
private double dcos(double degrees) {return Math.cos(Math.toRadians(degrees));}
private double atan2(double a, double b) {return Math.toDegrees(Math.atan2(a,b));}
private double eccentricAnomaly(double meananomaly, double eccentricity)
{
    double approximation1=meananomaly;
    double approximation2=meananomaly+(180/Math.PI)*eccentricity*
        dsin(approximation1)*(1+eccentricity*dcos(approximation1));
    while (Math.abs(approximation1-approximation2)>0.005)
    {
        approximation1 = approximation2;
        approximation2=meananomaly+(180/Math.PI)*eccentricity*
            dsin(approximation1)*(1+eccentricity*dcos(approximation1));
    }
    return approximation2;
}
private double normalise(double degrees)
{
    while (degrees > 360) degrees -= 360; while (degrees < 0 ) degrees += 360;
    return degrees;
}
}

```

Java code for MoonSatellite

```

import java.util.*;
import java.io.*;
import net.java.games.jogl.*;
class MoonSatellite extends Satellite {
    private Vector moonmeanlongitudelist, moonmeanelongationlist, sunmeananomalylist,
        moonmeananomalylist, moonargumentoflatitudelist, allist, a2list, a3list;
    private int table47a[][] , table47b[][];
    public MoonSatellite(String path, GLDrawable gld) throws Exception
    {
        LineNumberReader r = new LineNumberReader(new FileReader(path));
        name = r.readLine(); parent = r.readLine(); mass = Read.readDouble(r, "mass");
        spherediameter = Read.readDouble(r, "spherediameter");
        ringwidth = Read.readDouble(r, "ringwidth");
        ambient = Read.readFloatArray(r, "ambient", 4);
        diffuse = Read.readFloatArray(r, "diffuse", 4);
        specular = Read.readFloatArray(r, "specular", 4);
        shininess = Read.readFloat(r, "shininess");
        emission = Read.readFloatArray(r, "emission", 4);
        moonmeanlongitudelist = Read.readDoubleVector(r, "moonmeanlongitudelist");
        moonmeanelongationlist = Read.readDoubleVector(r, "moonmeanelongationlist");
        sunmeananomalylist = Read.readDoubleVector(r, "sunmeananomalylist");
        moonmeananomalylist = Read.readDoubleVector(r, "moonmeananomalylist");
        moonargumentoflatitudelist = Read.readDoubleVector(r, "moonargumentoflatitudelist");
        allist = Read.readDoubleVector(r, "allist");
        a2list = Read.readDoubleVector(r, "a2list");
    }
}

```

```

a3list = Read.readDoubleVector(r,"a3list");
//We use only the first 32 terms:
table47a = Read.readintMatrix(r,"table47a",32,6);
table47b = Read.readintMatrix(r,"table47b",32,5);
this.gld = gld;
makeLists();
}
public State stateAt(double time)
{
    double[] position1 = positionAt((time - 2451545)/36525);
    double[] position2 = positionAt(((time + 1.0/24.0) - 2451545)/36525);
    double[] velocity = {
        position2[0] - position1[0], position2[1] - position1[1],
        position2[2] - position1[2],
    };
    return new State(this, time, position1, velocity, gld, texture[0], slist, rlist);
}
public State stateAfter(double time, double quantum, States s)
{
    double steppedtime = time + quantum;
    double[] position1 = positionAt((steppedtime - 2451545)/36525);
    double steppedtimeplushour = steppedtime + (1.0/24.0);
    double[] position2 = positionAt((steppedtimeplushour - 2451545)/36525);
    double[] velocity = {
        position2[0]-position1[0], position2[1]-position1[1],
        position2[2]-position1[2],
    };
    return new State(this, steppedtime, position1, velocity, gld, texture[0], slist, rlist);
}
private double[] positionAt(double time)
{
    double x,y,z, moonmeanlongitude, moonmeanelongation, sunmeananomaly,
        moonmeananomaly, moonargumentoflatitude, a1, a2, a3, sigmal, sigmar, sigmab,
        longitude, latitude, distance;
    moonmeanlongitude=normalise(sumPolynomial(time,moonmeanlongitudelist));
    moonmeanelongation=normalise(sumPolynomial(time,moonmeanelongationlist));
    sunmeananomaly=normalise(sumPolynomial(time,sunmeananomalylist));
    moonmeananomaly=normalise(sumPolynomial(time,moonmeananomalylist));
    moonargumentoflatitude=normalise(sumPolynomial(time,moonargumentoflatitudelist));
    a1=normalise(sumPolynomial(time,allist));
    a2=normalise(sumPolynomial(time,allist));
    a3=normalise(sumPolynomial(time,allist));
    //We use only the first 32 terms:
    sigmal = 0; sigmar = 0;
    double argument = 0;
    for (int i = 0; i < 32; i++)
    {
        argument =
            table47a[i][0]*moonmeanelongation +
            table47a[i][1]*sunmeananomaly +
            table47a[i][2]*moonmeananomaly +
            table47a[i][3]*moonargumentoflatitude;
        sigmal += table47a[i][4]*dsin(argument);
        sigmar += table47a[i][5]*dcos(argument);
    }
    sigmab = 0;
    for (int i = 0; i < 32; i++)
    {
        argument =
            table47b[i][0]*moonmeanelongation +
            table47b[i][1]*sunmeananomaly +
            table47b[i][2]*moonmeananomaly +
            table47b[i][3]*moonargumentoflatitude;
        sigmab += table47b[i][4]*dsin(argument);
    }
    //The additives:
    sigmal +=
        3958*dsin(a1) +
        1962*dsin(moonmeanlongitude-moonargumentoflatitude) +
        318*dsin(a2);
    sigmab +=
        -2235*dsin(moonmeanlongitude) +
        382*dsin(a3) +
        175*dsin(a1 - moonargumentoflatitude) +
        175*dsin(a1 + moonargumentoflatitude) +
        127*dsin(moonmeanlongitude - moonmeananomaly) -
        115*dsin(moonmeanlongitude + moonmeananomaly);
    longitude = moonmeanlongitude + sigmal/1000000;
    latitude = sigmab/1000000;
    distance = 385000.56 + sigmar/1000;
    //Convert distance to AU:
    distance = distance*6.68458134E-9;
    x=distance*dsin(90-latitude)*dcos(longitude);
    y=distance*dsin(90-latitude)*dsin(longitude);
    z=distance*dcos(90-latitude);
    return new double[] {x, y, z};
}
private double sumPolynomial(double time, Vector coefficientlist)
{
    //Horner's rule. We require that length > 1:
    double sum = ((Double)coefficientlist.get(coefficientlist.size()-1)).doubleValue();
    for (int i = coefficientlist.size()-2; 0<=i; i--)
        sum = sum*time + ((Double)coefficientlist.get(i)).doubleValue();
    return sum;
}
private double dsin(double degrees) {return Math.sin(Math.toRadians(degrees));}
private double dcos(double degrees) {return Math.cos(Math.toRadians(degrees));}

```

```

private double datan2(double a, double b) {return Math.toDegrees(Math.atan2(a,b));}
private double eccentricAnomaly(double meananomaly, double eccentricity) {
    double approximation1=meananomaly;
    double approximation2=approximation1+(180/Math.PI)*eccentricity*
        dsin(approximation1)*(1+eccentricity*dcos(approximation1));
    while (Math.abs(approximation1-approximation2)>0.005)
    {
        double temp=approximation2;
        approximation2=approximation1+(180/Math.PI)*eccentricity*
            dsin(approximation1)*(1+eccentricity*dcos(approximation1));
        approximation1=temp;
    }
    return approximation2;
}
private double normalise(double degrees)
{
    while (degrees > 360) degrees -= 360;
    while (degrees < 0 ) degrees += 360;
    return degrees;
}
}

```

Java code for PlutoSatellite

```

import java.util.*;
import java.io.*;
import net.java.games.jogl.*;
class PlutoSatellite extends Satellite {
    private Vector jlist, slistx, plist;
    private int table37a[][];
    public PlutoSatellite(String path, GLDrawable gld throws Exception
    {
        LineNumberReader r = new LineNumberReader(new FileReader(path));
        name = r.readLine(); parent = r.readLine(); mass = Read.readdouble(r,"mass");
        spherediameter = Read.readdouble(r,"spherediameter");
        ringwidth = Read.readdouble(r,"ringwidth");
        ambient = Read.readfloatArray(r,"ambient",4);
        diffuse = Read.readfloatArray(r,"diffuse",4);
        specular = Read.readfloatArray(r,"specular",4);
        shininess = Read.readfloat(r,"shininess");
        emission = Read.readfloatArray(r,"emission",4);
        jlist = Read.readDoubleVector(r,"jlist");
        slistx = Read.readDoubleVector(r,"slistx");
        plist = Read.readDoubleVector(r,"plist");
        //We use only the first 10 terms:
        table37a = Read.readIntMatrix(r,"table37a",10,9);
        this.gld = gld;
        makeLists();
    }
    public State stateAt(double time)
    {
        double[] position1 = positionAt((time - 2451545)/36525);
        double[] position2 = positionAt((time + 1.0/24.0) - 2451545)/36525);
        double[] velocity =
        {
            position2[0] - position1[0], position2[1] - position1[1], position2[2] - position1[2], };
        return new State(this, time, position1, velocity, gld, texture[0], slist, rlist);
    }
    public State stateAfter(double time, double quantum, States s)
    {
        double steppedtime = time + quantum;
        double[] position1 = positionAt((steppedtime - 2451545)/36525);
        double steppedtimeplushour = steppedtime + (1.0/24.0);
        double[] position2 = positionAt((steppedtimeplushour - 2451545)/36525);
        double[] velocity =
        {
            position2[0]-position1[0],
            position2[1]-position1[1],
            position2[2]-position1[2],
        };
        return new State(this, steppedtime, position1, velocity, gld, texture[0], slist, rlist);
    }
    private double[] positionAt(double time)
    {
        double x,y,z,
            j, s, p, sigmal, sigmab, sigmar,
            longitude, latitude, distance;
        j=normalise(sumPolynomial(time,jlist));
        s=normalise(sumPolynomial(time,slistx));
        p=normalise(sumPolynomial(time,plist));
        //We use only the first 10 terms:
        sigmal = 0; sigmab = 0; sigmar = 0;
        double argument = 0;
        for (int i = 0; i < 10; i++)
        {
            argument =
                table37a[i][0]*j +
                table37a[i][1]*s +
                table37a[i][2]*p;
            sigmal += table37a[i][3]*dsin(argument) + table37a[i][4]*dcos(argument);
            sigmab += table37a[i][5]*dsin(argument) + table37a[i][6]*dcos(argument);
            sigmar += table37a[i][7]*dsin(argument) + table37a[i][8]*dcos(argument);
        }
        longitude = 238.958116 + 144.96*time + sigmal/1000000;
        latitude = -3.908239 + sigmab/1000000;
    }
}

```

```

distance = 40.7241346 + sigmar/10000000;
x=distance*dsin(90-latitude)*dcos(longitude);
y=distance*dsin(90-latitude)*dsin(longitude);
z=distance*dcos(90-latitude);
return new double[] {x, y, z};
}
private double sumPolynomial(double time, Vector coefficientlist)
{ //Horner's rule. We require that length > 1:
double sum = ((Double)coefficientlist.get(coefficientlist.size()-1)).doubleValue();
for (int i = coefficientlist.size()-2; 0<=i; i--)
sum = sum*time + ((Double)coefficientlist.get(i)).doubleValue();
return sum;
}
private double dsin(double degrees) {return Math.sin(Math.toRadians(degrees));}
private double dcos(double degrees) {return Math.cos(Math.toRadians(degrees));}
private double datan2(double a, double b) {return Math.toDegrees(Math.atan2(a,b));}
private double eccentricAnomaly(double meananomaly, double eccentricity) {
double approximation1=meananomaly;
double approximation2=approximation1+(180/Math.PI)*eccentricity*
dsin(approximation1)*(1+eccentricity*dcos(approximation1));
while (Math.abs(approximation1-approximation2)>0.005)
{ double temp=approximation2;
approximation2=approximation1+(180/Math.PI)*eccentricity*
dsin(approximation1)*(1+eccentricity*dcos(approximation1));
approximation1=temp;
}
return approximation2;
}
private double normalise(double degrees)
{ while (degrees > 360) degrees -= 360; while (degrees < 0 ) degrees += 360; return degrees; }
}

```

Java code for SunSatellite

```

import java.io.*;
import net.java.games.jogl.*;
class SunSatellite extends Satellite {
private double position[] = new double[3];
private double velocity[] = new double[3];
public SunSatellite(String path, GLDrawable gld) throws Exception
{ LineNumberReader r = new LineNumberReader(new FileReader(path));
name = r.readLine(); parent = r.readLine(); mass = Read.readdouble(r,"mass");
spherediameter = Read.readdouble(r,"spherediameter");
ringwidth = Read.readdouble(r,"ringwidth");
ambient = Read.readfloatArray(r,"ambient",4);
diffuse = Read.readfloatArray(r,"diffuse",4);
specular = Read.readfloatArray(r,"specular",4);
shininess = Read.readfloat(r,"shininess");
emission = Read.readfloatArray(r,"emission",4);
position = Read.readdoubleArray(r,"position",3);
velocity = Read.readdoubleArray(r,"velocity",3);
this.gld = gld;
makeLists();
}
public State stateAt(double time)
{ return new State(this, time, position, velocity, gld, texture[0], slist, rlist);
}
public State stateAfter(double time, double quantum, States s)
{ return new State(this, time, position, velocity, gld, texture[0], slist, rlist);
}
}

```