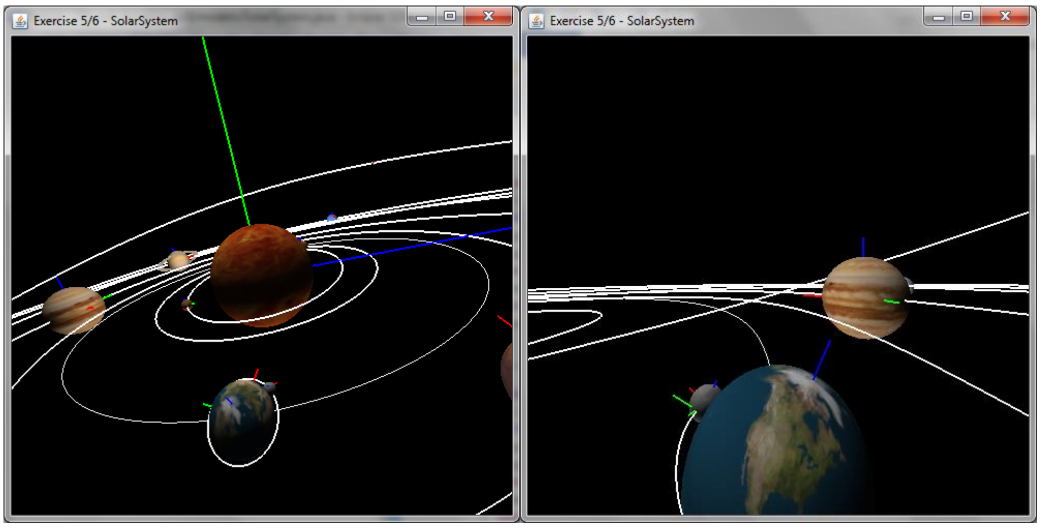
Exercise 6 – Animated Solar System



Contents

Overview

In this exercise you will implement an OpenGL 3D animation that will consist of animated textured Solar System. To do this you will practice textures, lighting, transformations and most of what we've learnt.

The implementation of this exercise will be an extension of ex5.

A reference executable can be downloaded from moodle. Note that you will need to place it in a folder along with the JOGL dlls.

Usage

The same as ex5, plus the following:

- Pressing 'c' toggles the camera position (explained later).
- The planets are animated, as they move along their orbits and revolve about their axis.
- Pressing 's' will toggle current planet (1 of 9 planets), when 'c' is pressed the camera will be placed above and behind the current planet and move along with it (explained later)

Requirements

1. General
    1.1. A planet should move along its orbit and revolve about itself.
    1.2. The moon should revolve around earth.
    1.3. The implementation of ex5 should be used as the starting point.
2. View
    2.1. The camera will be placed in several positions (toggled by the 'c' key):
        2.1.1. As in ex5 – rotated/zoomed using the mouse.
        2.1.2. Above and behind each moving planet (along its orbital motion direction without the axial tilt and not revolving about the planet's axis).
    2.2. In both cases, as before, a perspective projection will be used.
3. **Bonus1** -implement colliding asteroids (you can make a new model class for that): place 2 asteroids on the same orbit moving in opposite directions. After collision they will move in opposite directions (each one to the other side), till next collision and so on.
4. **Bonus2** - implement collision of all planets into the sun- instead of moving along the orbits performing a spiral into the sun and disappearing when hitting the sun. Spiral is implemented as a circle with changing radius.
5. **Bonus3**- make the orbits elliptic and move the planets along them. Do the adaptation needed for the camera view when the orbit is elliptic. (More explanations will follow in the bonus part in the appendix)

Placing the camera behind the planet

To do this you should first uncomment the handler of the 'c' key in App class. Then you can use IRenderable.setCamera() of your model to gain control of the viewing transformation inside the model. Pressing ‘c’ will bypass the mouse rotation/zoom controls, and use your own IRenderable.setCamera() instead.

For switching planets, use the same flow implemented for toggling light spheres in ex5:

- Add in a same way an ‘s’ key mode in the app, which will change the current “sub model”.
- Add a “toggleSubModel” method for the Viewer that will be called from the app (just like “toggleLightSpheres” method). Call the control method of the model from there (control is the method of the interface IRenderable as well, which your model is implementing).
- Add a `public static final int` constant in IRenderable for toggling sub-model (planets in the solar system) just like TOGGLE_LIGHT_SPHERES, and give it a different value than TOGGLE_LIGHT_SPHERES.
- Pass this constant to the IRenderable .control() method of your model from the Viewer
- Set the current planet state in the control method. This is the planet that the camera will move along with.

Tip: use gluPerspective for projection and play with the near and angle parameters. Also see if you scaled the scene to match your projection clipping. Keep these in mind when you set the camera in a different place- it might be that the problem is not the camera position or direction that you’ve set, but the projection that is attached to it and clipping which leaves some of the desired scene not rendered.

Textures

Textures should be used for all the planets and the Saturn ring, you can add a 1D texture to the orbit of you like. Image files are supplied for your convenience and are already flipped for you (texturing quadrics through glu flips the images back).

Files should be placed in such a location where the system will find them

```
File texFile = new File("texture.bmp");
```

In Eclipse this mean relative to the project folder. In ex6 put the Bitmaps folder supplied by us in your project folder and call

```
File texFile1 = new File("Bitmaps/texture1.bmp");
```

Recommended Milestones

Write incrementally. We suggest the following implementation milestones:

1. Add Textures to all planets.
2. Implement animation by returning true in SolarSystem.isAnimated and set an parameter for time you will increment each time you call render method you override from IRenderable in your model

3. Move Planets along their orbits.
4. Rotate planets about their axis
5. Rotate the moon about Earth
6. Implement SolarSystem .setCamera(). Implement it for one specific planet first. You can disable the motion of the scene first. Make sure that pressing ‘c’ shows the planet from behind as in the jar.
7. Implement setCamera more generically and apply the flow for using ‘s’ key to switch between the planets.
8. Check that you didn’t forget anything.

Submission

- Submission is in pairs
- Zip file should include
    - All the java source files, including the files you didn't change in a JAR named “ex6-src.jar”
    - Compiled runnable JAR file named “ex6.jar”
        - This JAR should run after we place JOGL DLLs in its directory
        - And textures in Bitmaps folder
        - Make sure the JAR doesn’t depend on absolute paths – test it on another machine before submitting
        - **Points will be taken off for any JAR that fails to run!**
- A short readme document where you can **briefly** discuss your implementation choices.
- Zip file should be submitted to moodle.
- Name it:
    - <Ex##> <FirstName1> <FamilyName1> <ID1> <FirstName2> <FamilyName2> <ID2>
- Submission deadline is : 16/06/2015
- Before submission be sure to check the discussion board for updates

Appendix

A Planetary Day - duration of self-revolution

Planet	A Planetary Day (Rotational Period in Earth Day or Hours)
Mercury	58.65 days
Venus	243.01 days
Earth	1 day
Mars	~1 day
Jupiter	0.41 days
Saturn	0.44 days
Uranus	0.72 days
Neptune	0.67 days
Pluto	6.38 days

A Planetary Year - Orbital Period

Use in logarithmic scale for getting a nice animation (log(number of days))

Mercury: 87.97 days (0.2 years)
Venus : 224.70 days (0.6 years)
Earth: 365.26 days(1 year)
Mars: 686.98 days(1.9 years)
Jupiter: 4,332.82 days (11.9 years)
Saturn: 10,755.70 days (29.5 years)
Uranus: 30,687.15 days (84 years)
Neptune: 60,190.03 days (164.8 years)

Pluto: 90800 days (248.8 years)

Bonus

Many improvements can be added to make this application more interesting. Such improvements will be graded with anything between 0-20 additional points, depending on creativity, difficulty, aesthetics and amount of fun gained by the user. ☺

You can also add some user interaction to manipulate the scene (add objects to the scene by key commands, move them by key commands or mouse and so on)

To get bonus points for the additional effects, they have to look aesthetic.

Elliptic orbits bonus

Before You Start- Good Practice Tip

For implementing this part it’s easier to disable all the effects but the rotation along the orbit. This means disabling the axial and orbital tilt, disabling self-revolution to debug this feature independently. Use show Axis to understand if your orientation is correct. After checking this feature, enable the rest incrementally and make it play.

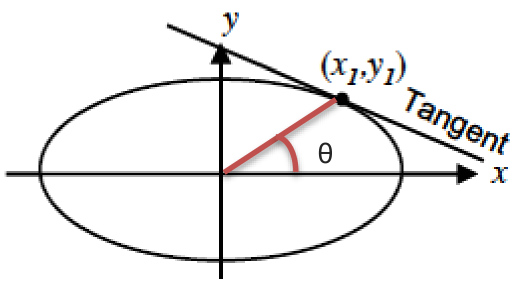
Some theory

2D ellipse equation is similar to the circle, but the radius is not constant.

$$x = a\cos\theta.$$
$$y = b\sin\theta.$$

You can choose **a** as orbit radius and **b** as 0.8 times orbit radius (or find something that’s closer to the physics).

So basically you can translate the planet to the position given by the equation. Additionally, you need to keep the direction correct. Imagine that the planet wasn’t a sphere in your model but a car. Then it was obvious that you wish the car front to face the direction of the orbit. For that you need to specify a rotation. This is done by making the planet face the direction of the **tangent of the ellipse**. We calculate it by taking a derivative of the equation above:



Ellipse

x_tangent=-a*sin (theta)
y_tangent=b*cos(theta)

Calculate the tangent of the ellipse in the initial position you placed your planet on the orbit (this is actually determined by θ). Normalize this vector ((x,y) is a 2D vector) and find its angle to the x direction. Make one axis of the planet that is in the orbit plane face the direction of the orbit at any time by rotating by this angle.

Place the camera just like in the circular orbit case- behind and above the planet, moving along with it. The camera is looking at a direction parallel to the orbits direction, described here.