SlingShot Flight

Jason Higuchi

Project Report

CS 491AB

Spring 2010

Prof. Elaine Eun-Young Kang

California State University, Los Angeles

College of Engineering, Computer Science, and Technology

Department of Computer Science

# screen-Thu-Jun-10-22-13-43-2010-1765.jpg

# Abstract

The Integrated Training Pipeline for Scientific Visualization (SciVi) is an NSF funded project at Cal State LA in association with JPL which collaborates with students and faculty to create scientifically accurate visualizations of the Universe and the physical laws that govern it. Slingshot Flight is one of these projects specifically created to scientifically demonstrate the physical laws governing simple planetary motion and the complexity of simple orbital changes by means of user interaction with a simulation space ship.

 This specific project, Slingshot Flight, is a space simulator game, specifically a simulator of simple two-body orbits governed by Kepler’s laws. The reason it was created was for the purpose of educating the ordinary, non-astrophysics educated persons about the general motion of the planets and the effects of impulse driven orbital changes that are most commonly associated with that of present spaceships and artificial satellites. Plus, the general lack of such simple information/education-driven games dealing with astrophysics is quite small.

# Introduction

The orbital simulator game Slingshot Flight is the visual representation of the basic physical laws governing orbital paths. Created under the SciVi project at Cal State LA, this game was made under the collaboration between undergraduates and graduates of computer science, physics, and art. The game itself fills in a niche gap in terms of orbital simulator games. Where certain space simulator games are created to be completely realistic, using rendered three dimensional representations of the solar system governed by complex multi-body orbital algorithms and providing accurate internal mock-ups of the interiors of current and possibly future manned space vehicles within their games’ ship models, they are highly complex and do not represent the nature of orbital movement as a whole as it is difficult for the user to understand the effects of their actions since they cannot see the entire path of the ship and the effects of gravity in a space or time scale they can completely mentally grasp. On the other hand, there are simpler games which use Newtonian definition of gravity to implement a sort of gravitational slingshot game, with the user able to launch a body around large gravity effecting planetary objects which remain still (such games can be easily found on the internet with names such as slingshot something or gravity golf something), however, these are games for the purpose of killing time, and are not highly complex, meaning not scientific in its representation of orbital paths. Between these two types of space game extremes, Slingshot Flight’s purpose is to be a simple game with accurate scientific representation of the perceived physical laws governing orbital movement while still providing good interaction with the user and allowing for a short learning curve.

 Slingshot Flight is the continuation of a previous project, which was formerly undertaken by Julia Yefimenko. Being of the same premise, the purpose of extending the project was to create a new set of levels, specifically that of asteroid mitigation (gravity tractor/deflection) and possibly level involving travel to the outer solar system utilizing a gravity assist (slingshot maneuver). But to undertake such changes, the code had to be changed to accommodate orbits which were to have full degrees of freedom within the 3D space of the game, and a new launch system to take advantage of such orbits. The project mainly revolved around the latter of these implementations.

# Game Background

To implement a fully three dimensional game, it is easiest to use an already complete 3D game engine. For all intents and purposes, the Panda3D Engine (<http://www.panda3d.org/>) was used to render the game scenes and implement the logic:

## Panda3D

Panda3D is a an open source game engine which was developed on C++, but can be used for game development and 3D scene graph rendering on C++ or the Python programming language. The Panda3D Engine is quite reliable for game development due to the large amounts of documentation on the website and in the Panda3D forums. Not only does it provide full and effective rendering of a 3D scene graph, the Panda 3D Engine is fully capable of using shaders with special maps (glow maps, texture maps, gloss maps, etc.), collision detection with its own collision solids, and a wide selection of its own library classes including a fully developed finite state machine transition graph. Its completeness and usability is evidenced by the fact that the Disney Corporation has used it to develop several online massively-multiplayer online games.

 While the Panda3D Engine’s libraries can be used in C++, it is more convenient to code and develop in the Pyton programming language, given the purpose of the Python language is to be a general-purpose language which is easily read and coded in, despite the type of programmer who is developing it.

## Python

The Python programming language was chosen as the primary developing language for the Panda3D Engine because of its versatile nature. Python as a language is mainly object-oriented and imperative, however, it does support functional programming styles, meaning that certain code can be written shorter and more easily readable, though that is in terms of mathematical computation or array functions. Also, with its dynamic type system, the necessity to consistently typecast variables is gone, and the ability to reuse variable names is a bit more flexible, and with Python’s automated memory allocation (i.e. garbage collection) the programmer does not need to allocate and de-allocate memory manually within the code. Also, being a dynamic language, Python has the ability to modify existing code and objects, extending them, allowing for a better level of debugging specifically within a game.

 The downside of using Python is efficiency. Being implemented with automated garbage collection and as a dynamic language with easy readability, Python is slower than C++. Though the Panda3D Engine is coded in C++ and can perform memory operations much faster than if it were written in Python, all code for the game written in the higher level language of Python can ultimately slow down the game’s speed if there too many memory or mathematically intensive operations within the code that is directly coded in Python.

# Implementation

The algorithms which implement some of the fundamental behaviors in Slingshot Flight are as follows:

## Basic Orbital Launch

The launch sequence for each ship body within the game used to be implemented as the start to a Hohmann Transfer Maneuver, a change in velocity which was executed perpendicular to the normal of the orbit of Earth. However, the launch is now an additive velocity in any direction of the 3D Cartesian plane to the Earth's current velocity, which is recalculated to reform the orbit in respect to these changes in velocity, now implemented in this way:



Figure : Orbital Plane (yellow) intersects Reference Plane (gray), which will be the Ecliptic Plane (parallel to the Earth’s orbit around the Sun) with the Sun as the Origin.

r = distance from origin in terms of a heliocentric frame of reference (radius)

i = initial inclination (from launch platform or current position)

w = initial Argument of Perihelion

a = initial semi-major axis

e = initial eccentricity

Ω = initial Longitude of Ascending Node

μ = Gravitational constant \* Mass of the Sun (Gravitational Potential)

v = initial velocity

, given

, vis-viva equation

h is the angular momentum from initial state vector



 , cross product of velocity and radius is angular momentum;  is the component of velocity that is perpendicular to , which does not equate to zero when taking its cross product with h.

, radial component of velocity





is the x component of velocity in the orbital frame (where the Sun is the origin, but the body’s perihelion is the reference direction of the x-axis)

 is the y component of velocity in the orbital frame

Change velocity components to those in the perifocal frame (where the Sun is the Origin and is parallel to the ecliptic plane):

$$v\_{x}^{'}= v\_{x}\*\cos(\left(-Ω\right))- v\_{y}\* sin⁡(-Ω)$$

$$v\_{y}^{'}= v\_{x}\*sin⁡(-Ω)- v\_{y}\*cos⁡(-Ω)$$

$$v\_{y}^{'}= v\_{y}\*\cos(\left(-i\right))- v\_{z}\*\sin(\left(-i\right))$$

$$v\_{z}^{'}= v\_{y}\*\sin(\left(-i\right))- v\_{z}\* cos(-i)$$

$$v\_{x}^{'}= v\_{x}\*\cos(\left(-ω\right))- v\_{y}\*\sin(\left(-ω\right))$$

$$v\_{y}^{'}= v\_{x}\*\sin(\left(-ω\right))- v\_{y}\* cos(-ω)$$

Add change in velocity and new velocity parameters are found:









v' is the new velocity transformed to the ecliptic plane



h' is the angular momentum for new orbit state vectors



U' is the orbital energy of new orbit



a' is the semi-major axis of new orbit



e’ is the eccentricity of the new orbit



i’ is the inclination of the new orbit



 is the argument of perihelion of the new orbit

however, if <0:



if i' is equal to zero, then Ω' is equal to zero



or



if i' is 0 then:



else:



if e\_z < 0 then:



from the new state vectors, Ω,w, and I was found, using these angles and using them to transform the heliocentric coordinates in reverse, perifocal coordinates X and Y can be found, True Anomaly, ϴ can be found:



if Y < 0:



if e' < 1:



else:

 , for hyperbolic orbits



P is the period

pT is the time after periapsis



if the orbit is hyperbolic:

, period is purely symbolic for hyperbolic orbits in calculating time after periapsis

After all the new parameters for the orbit are calculated, they are assigned to the ship's orbit and the orbit is recalculated and the pT assigned as the new current time for the ship.

If the ship's orbit is hyperbolic, is calculated as follows:



where is the angle at which a hyperbolic orbit's radius is infinite (with respect to the true anomaly)

## Basic Orbit Update

The orbit of each body is updated by finding the true anomaly at the given time,using the state parameters e, a, μ, Ω,ω, and i (eccentricity, semi-major axis, gravitational potential, longitude of ascending node, argument of perihelion, and inclination) to rotate the orbit into the ecliptic plane from the body’s own orbital frame of reference:

$m= sqrt(\frac{μ}{a^{3}})$ \* t – T, m is the mean anomaly, T is the period of the orbit and t is the current time since passing the perihelion (beginning new period T)



Figure : Eccentric Anomaly, E, in relation to True Anomaly, θ (C is the center of the ellipse, and F is the Focus)

The mean anomaly, m, is used to solve for the eccentric anomaly, E, by using Newton’s method:

m is approximated as a guess, usually in relation to the true anomaly.

$$ratio=\frac{\left(E-e\*\sin(\left(E\right))- m\right)}{1-e\*cos⁡(E)}$$

$E=E-ratio$, where E is constantly adjusted until ratio is sufficiently small

Then the true anomaly is found by:

$$θ=atan2(\sqrt{1-e^{2}}\*\sin(\left(E\right)),\cos(\left(E\right))- e)$$

Which can be used to find the radius, r, or distance from the focus:

$$r=a\*\frac{\left(1-e^{2}\right)}{(1+e\*\cos(\left(θ\right)))}$$

From which the position of the body can be positioned from

## Gravity Tractor

The gravity tractor works by increasing the semi-major axis, a, of the asteroid body which the ship body is orbiting, allowing for an even expansion of the asteroid body orbit.

## Gravity Assist (Slingshot)

The gravity assist works by reparenting the ship body to the planet’s (the one it is getting the boost from) frame of reference and adjusting the gravitational potential, µ, of the ship body’s to be that of the planet’s, as that is the only parameter which changes, so the update of the ship body’s orbit is within the frame of reference to the planet (with the planet being the origin) and will update around it as if it were still within the perifocal frame (heliocentric frame, with the Sun as the origin, and parallel to the ecliptic plane)

# Design



Speed Slider

Accept

Ships

Velocity Sliders

The design of the game right now is simple. The user uses the ship buttons to buy a ship with credits (indicated on the top left of the screenshot) and uses the velocity sliders to adjust the orbit of the ship, which is indicated by the red dotted lines as seen in the screenshot below.



When the desired trajectory is found, the user presses the accept button and the ship is sent into its new orbit.



The user uses these orbital maneuvers to achieve the goals set at the beginning of the level.

# Overview

In terms of completeness, the game is quite underdeveloped in terms of interactivity. For most games, there is quite a degree in freedom or repeatability involved, but Slingshot Flight lacks such freedom in controls and repeatability (repeatability meaning changes in the gaming experience for the users due to certain elements within the game changing with each play). The entire concept of Slingshot Flight as a game is lacking because of the fact it is not very playable at the moment.

 Other than the gaming aspects, Slingshot Flight has the basic logic in it already implemented, as seen from the algorithms shown in the implementation section. The bodies within the game are able to update their positions and launch, along with having certain algorithms implemented for a couple of the levels. However, in terms of expanding the game itself, the structure of the logic should be changed to fit a more general layout rather than having specific classes and methods for each of the level’s functions.

# Conclusion (Future Work)

Slingshot Flight is still in development, but the most basic of tools are now implemented, those being the orbital updates, the launch sequence, and the basic 3D layout of the entire game. With restructuring of the current code to improve efficiency and readability, the next set of levels can be implemented, such as travel to the outer solar system or interception of multiple near Earth objects. Also, the re-implementation of the levels can be done, such as changing the premise of the asteroid level from that of using a gravity assist to nudge the asteroid, to that of using a nuclear device to change the orbit of or even destroy the asteroid, which at the present is a more realistic and viable strategy.

# References

Curtis, Howard. “Orbital Mechanics for Engineering Students”. Burlington,MA: Elsevier Butterworth-Heinemann, 2005.

Fraenz, Markus. “Planetary Orbits”. Heliospheric Coordinate Systems. **Astronomy Unit, Queen Mary, University of London, London, E1 4NS - D. Harper
The Sanger Centre, Wellcome Trust Genome Campus, Hinxton, Cambs CB10 1SA.** [http://www.mps.mpg.de/homes/fraenz/systems/systems2art/node15.html]. 2002-03-12

Gravitational Slingshot. MathPages. [http://www.mathpages.com/home/kmath114.htm]

Panda3D. [http://www.panda3d.org/]. 2010

Python. [http://www.python.org/]. 2010

# Acknowledgements

This project was originally coded by Julia Yefimenko 2009 at CalState LA. The physics advisor was also Dr. Milan Mijic of CalState LA’s Department of Physics and Astronomy. Advising Professor was Dr. Elaine Eun-Young Kang