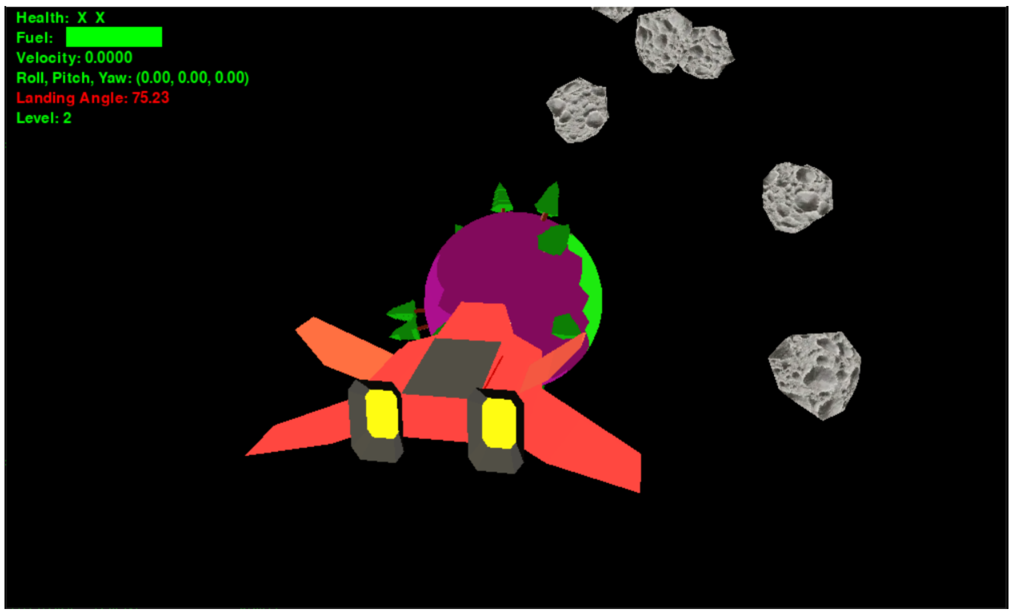
Interplanetary Space Jaunt Specifications and Sofware Architecture

Jay Kmetz

November 29, 2020

1 Description

You control a spaceship which has deliberately hard controls: pitch, yaw, roll, thrust, and stabilizers for all controls (See Section 4), who's sole purpose is to make it from planet to planet to refuel while flying through an asteroid field. The spaceship will have three health and a certain amount of fuel. Thrusting will use up fuel. Hitting an asteroid which is floating between you and the planet, running out of fuel, or crashing into the planet (see Section 3) will remove a health point and remove the obstruction. Successfully landing the ship on the planet (see Section 5) will advance the player to the next level. Each level will be procedurally generated and will get harder as you progress through the game. There will be a HUD which displays useful information such as your health, fuel, velocity, pitch, yaw, roll, landing angle and level counter.



Health: X X
Fuel:
Velocity: 0.0000
Roll, Pitch, Yaw: (0.00, 0.00, 0.00)
Landing Angle: 75.23
Level: 2

Figure 1: Screen grab from Interplanetary Space Jaunt

2 Planet

The Planet will have a procedurally generated landing spot marked in green. In order to complete a level, the spaceship must correctly land on the green landing spot. The other area will be too densely packed with trees for a safe landing. See Section 3 for more details.

3 Damage

Throughout the game, there will be many opportunities to take damage. The damage will be displayed in your onscreen HUD. Damage can come from three sources:

- Asteroids
  - Hitting into an asteroid will automatically damage the ship. Any asteroid that is hit by the ship will be instantly pulverized by the ship's impressive asterolysis technology.
- Landing incorrectly (Crashing into the planet)
  - If the ship takes damage from the planet while it has more than one health, the pilot will black out and be ejected off the planet. If the ship held together, the pilot will miraculously wake up close to the landing zone of the planet.
  - When the ship is going too fast for a proper landing, the Velocity readout on the HUD will display in a red color. If the ship collides with the planet while the velocity is red, the ship will take damage.
  - When the ship's landing angle is too steep, it will appear in red in the HUD. The landing angle is the angle between the ship's up vector and the vector from the center of the planet to the ship. Basically, try to land as flat as possible on the planet.

  - Landing on anything nut the green landing zone will cause the ship to crash and the ship's autopilot will take over to try and steer the ship back over the landing zone.

- Running out of fuel

  - The ship's hull is made out of a metal/hydrazine mixture. Conveniently, the ship runs on hydrazine! If you run out of fuel, you can re-purpose some of the ship's hull and use it to fill up your tanks. Be careful though as doing this will cost you a point of health.

4 Ship Controls

The pilot will follow his ships guidance system (represented by an undulating arrow) to the nearest planet.

The pilot will be able to control each Euler component of it's angular acceleration directly due to advancements in spaceship handling technology. That is, the pilot will have to control the spaceship's roll, pitch, and yaw in order to accomplish their goal. See Figure 2 for details.

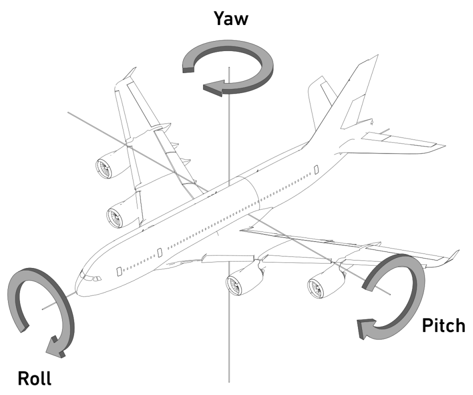


Figure 2: Explanation of Roll, Pitch, and Yaw

The pilot will also be able to centralize each of these using the ship's correction software. Pressing the corresponding button will attempt to apply a force in a direction to diminish one of the angular counterparts.

The pilot will be able to move the ship through space (or rather move space around the ship to achieve Faster Than Light relative speeds) by thrusting either forwards or backwards. The pilot can also create a dampening force to slow down the ship, no matter the direction or orientation the ship currently possesses by using the ThrustCorrect^{TM} technology developed by JavaCorp.

5 Level Win Condition

A level is won and passed by landing on the planet in the designated landing spot with a safe velocity and landing angle. Once the pilot lands, the planet begins to rumble and jostle the ship around. When the pilot regains consciousness, they realize that they are further away from the planet than when they started and there are more asteroids in their way!

"I always thought something was fundamentally wrong with the universe" [1]

6 The Start

This project started out very simply by rendering handmade cubes. See Figure 3.

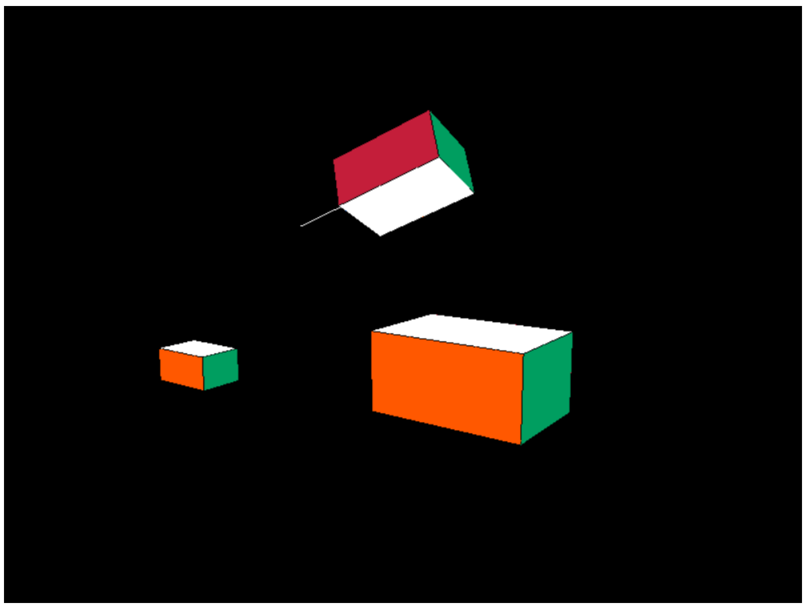


Figure 3: How the project started

From a project which started with only 18 faces, it has come a long way. Problems with rotating objects and views separately made the decision to use Quaternions instead of abusing OpenGL's matrix operations and Matrix Stack. Objects which were drawn in immediate mode moved to call lists, lighting and materials were added, Blender import code was conceptualized and made, and a fairly decent game was born.

7 Software Architecture

The project combines OOP (Object Oriented Python) and Procedural Python to abstract each object. The dependencies are listed here:

- PyOpenGL 3.1.5
- pygame 1.9.6
- numpy 1.19.2
- Pillow 7.2.0

The code and assets are split into 4 main parts:

7.1 pyobjs

Contains all of the objects which are rendered on screen. Each object is in charge of rendering itself.

7.1.1 Asteroid

Keeps track of its velocity and loads its DisplayObj.

7.1.2 ColObj

Superclass of all collision-enabled objects. Keeps track of the objects position, collision radius, and display object

7.1.3 Planet

Creates its display object, initializes its landing position based on input, populates its surface with trees, and houses logic to check the spaceship landing

7.1.4 Spaceship

Houses all logic for user input enabled movement and orientation using quaternions, handles its own health / fuel calculations, draws an arrow which points to a designated point in space, and makes all of its orientation/position data accessible by callers.

7.2 utils

Contains all of the files which are used as utilities all over the program

7.2.1 DisplayObj

Responsible for loading in geometries from .obj, .mtl, and .bmp files. Handles rendering, registering call lists and texture lists, and processes textures and materials to be rendered by OpenGL.

7.2.2 quat

Houses all the quaternion related math which is need to do orientation calculations all throughout the program

7.2.3 util

Has utility functions which are used throughout the program to do math calculations. Some examples are any vector calculation (add_vecs, sub_vecs,dot_vecs, etc.), level generation functions like a logistic curve to asymptotically approach a certain value as levels increase, and coordinate system conversions for easy coordinate abstractions like spherical to cartesian.

7.2.4 View

Handles all view calculations. There are three different modes: VT_STATIC, VT_SHIP_ RELATIVE, and VT_ORBIT. VT_STATIC represents a view which stays where it was but looks at a certain point, VT_SHIP_RELATIVE represents a view which is always moving with the ship to make it appear that everything else is moving around the ship, and VT_ORBIT which remains relative to the ship but orbits around the top at a specified height

7.3 wfobjs

Houses all of the .obj, .mtl, and .bmp files which are used by the DisplayObj. All of the .obj and .mtl files were created in Blender.

7.4 game.py

Serves as the entry point for the application. This handles User input, Views, Game initialization, integration between pygame and OpenGL, initializes lighting, draws the HUD, handles win and lose conditions, and handles all the collisions. This is truly where the magic happens.

References

[1] D. Adams. *The Hitchhiker's Guide to the Galaxy.* San Val, 1995.