Paraglider Physics Simulation with Bullet Physics

Pudniks, K. and Stephen, T.

5 Feb 22

1 Introduction

The process of designing and testing paraglider wings is inherently dangerous. This project attempted to lay the groundwork for system to design and test wings *in silica*, in realtime. This has the key advantages of

- allowing rapid iteration of wing design parameters
- simulating a range of scenarios, including "worst case" situations (such as turbulent weather, forced cravattes, tangled lines, rapidly changing winds, etc) that would not be safe for experienced pilots to test the wing in
- recording extremely fine data regarding positions, headings, velocities, etc to analyse
- allowing the tester to react to the movement of the wing, as opposed to being forced to program in moves

In doing this, we aimed to create a tool that is safer, cheaper, and more informative than previous practices.

2 Method

The simulation has been built using the *Bullet Physics* C++ SDK, with graphics done in OpenGL. Bullet Physics (hereafter referred to as "Bullet") is a realtime physics library, commonly used in games, machine learning, visual effects, etc. The project itself is an extension of the "SoftBody" demo included with Bullet 2.71.

The demo that this simulation is extended from handles most of the work of creating the application, applying forces, drawing to the screen, and so forth. To extend the demo, we created a function to initialize a paraglider wing (this function is discussed below). Once the paraglider is initialized and the program is started, Bullet handles most of the remaining work. We appended some code to the `SoftDemo::renderme` function to assist with visualisation of forces (affectionately named "echidna mode"), and to the `SoftDemo ::clientMoveAndDisplay` function (which is called every timestep, and calls functions to step the simulation and render the world) to log the current time and position of the pilot to a file.

Actual paraglider wings are complex designs, especially when considering the two planes, fine riser plans, ribbing within the wing, etc. For this simulation, we approximate the wing as a single plane, with a basic riser plan of main lines to "brakes" (only the rearmost set are treated as brakes), where the lines then split only once up to the wing. When generating the wing, the user has the options of varying its width (x dimension) and depth (z), as well as the resolution (expressed as number of nodes) in each dimension[1]. There are also parameters to adjust the spread riser attachment points on the wing, allowing, for example, to bias risers to the front of the wing while leaving the brakes attached to the rear.

The first iteration of the physics model applied hard-coded forces to specific nodes[2]. This led to a "floaty" wing, due to a precise balance between these forces, which was not influenced by it's own velocity or load. Also, due to the hard-coding of parameters, the resolution of the wing could not be changed without also moving where the forces are applied.

The second, and current, iteration of the

[1] During testing on a fairly low-power laptop (2015 MacBook Pro, i5, integrated graphics), we found that we could simulate a wing with resolution of at least 62×30 in real time.

[2] Specifically, these forces included an upwards force to nodes near the center of wing, and forces away from the wing to the edges to pull the wing open, etc.

physics model defers all work to Bullet. In this iteration, the set up is more complex, involving the fine-tuning of many parameters (relating to lift and drag force, properties of the wing material, etc), however the results are much more realistic. For the main forces on the wing, Bullet 2.71 has the notion of "Aero models", a list models to apply aerodynamic forces to soft bodies. When the wing is initialized, we apply the appropriate aero model, and realistic lift and drag forces are calculated and applied by Bullet.

Finally, a key component of the program design is the file inputs and outputs. Although the simulation is well suited for interactive use (i.e., allows keyboard and mouse interactions, and running in real time), we have added the capability to read in:

1. a config file, containing as many or as few parameters that the user wishes to set. For example, the mass, dimensions and resolution of the wing, mass of the pilot, lift and drag coefficients, etc.

2. an input file, containing a list of times and brake commands. This allows us to pre-program certain manuevers

The combination of these two files, alongside an output file which logs the current time and position of the pilot at every timestep, allows us to design automated testing to analyse the effects of changing parameters and situations. In the below *Results* section, we use these inputs and outputs to automate "batch" tests, involving many runs with slightly varied parameters.

The paths in the below *Results* section are plotted using Matplotlib and Python3. Since compiling the simulation simply produces an executable file, one may use the `subprocess` library in Python to run the simulation from a Python script. The workflow for each simulation then is to write a Python script to

1. clear the directory of any unnecessary files

2. create `config.txt` and `input.txt` to suite the test to be completed

3. call the simulation executable

4. read in the output file (Pandas is well suited to this)

5. plot the path using Matplotlib

Some other code is added for command line inputs, theming of the output plots, etc. Please reach out if you'd like to see the Python scripts.

3 Results

3.1 Basic Testing Parameters

All of the following tests are completed under the same conditions. This includes factors such as:

- gravity, air density, aerodynamics lift and drag coefficient, etc

- the wing starts at rest, with the front left corner 400m above the origin

- the simulation starts with no inputs for the first five seconds, to allow for the wing to settle[3]

3.2 Batch Testing

3.2.1 Hold Left Brake

In this batch run, we set all parameters as outlined above, allow the wing to settle for five seconds, and then apply the left brake for varying lengths of time, as labelled (in seconds) in the legend.

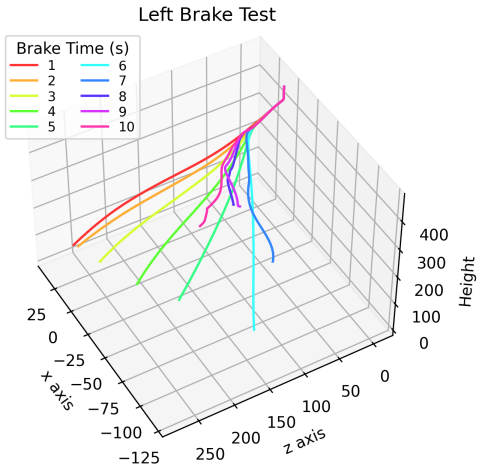

Figure 1: Comparison of paraglider flight paths effected by applying left brake for varying lengths of time

[3]The wing starts motionless, so begins to fall, then generate lift and in turn forwards movement. If we were to apply inputs immediately, we would be braking on a stationary wing, and this causes it to collapse.

We observe that the wing behaves identically for the initial few seconds each run, before diverging as the brake is applied for longer as the runs continue. The wing tends to the right (discussed below), however it is deflected further left, proportional to the time that the left brake is held, until the left brake is held for too long (between 6-7 seconds), causing the wing to slow too much, leading to a collapse and then spiral into the ground.

3.2.2 Gentle Left Brake

The brakes are "pulled" by simply applying an impulse. The C code for this looks something like

```
if (left_brake_pulled) {
    // [sic]

    left_brake -> applyImpulse(
    btVector3(0, -1.5, 0), left_brake
    -> getCenterOfMassPosition());
}
```

The `applyImpulse()` function applies an impulse (the first vector) to the body it is called on, about some position (then second argument). In the previous test, we applied the full impulse for varying times. In this test, we pull the left brake for 10 seconds, and vary the impulse for each run (specifically, we vary a coefficient from 0 to 1).

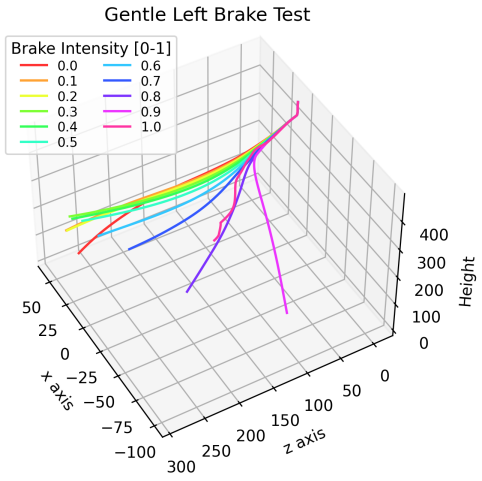

Figure 2: Comparison of paraglider flight paths effected by applying left brake for ten seconds at varying magnitudes

As expected, applying very little brake has a small effect on the path of the wing. As the force increases, the wing is deflected further left, until again collapsing at the highest intensities. Notice that the path with maximum intensity matches the 10s path in the previous test.

3.2.3 Hold Both Brakes

In this batch run, we allow the wing to settles then vary the time *both* brakes are held.

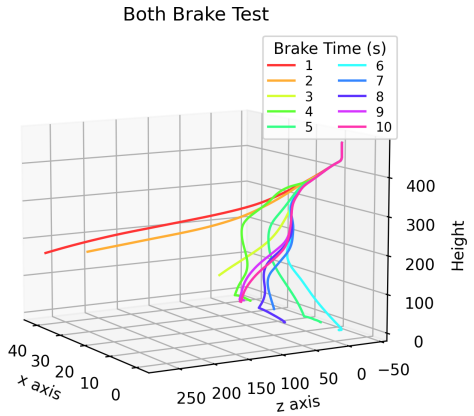

Figure 3: Comparison of paraglider flight paths effected by applying both brakes for varying lengths of time

When both brakes are applied for short lengths of times, the wing slows down then returns to normal flight. Notice that the 1s flight travels further than the 2s flight - the extra second of braking leads to a significant difference in ground speed. When the brakes are applied for moderate lengths of time (2-4s), the wings groundspeed is greatly reduced, leading to a greatly decreased glide ratio. In some cases, the wing may still resume steady flight (albeit at a much lower altitude), or the wing collapses and dives. For the remaining, longer time periods, the wing is slowed too far, causing the pilot to swing forwards, and the trailing edge of the wing to tip backwards while being braked, leading to the wing accelerating backwards.

3.2.4 Vary Pilot Mass

In this batch run, there are no inputs, however the pilot mass was different for each run.

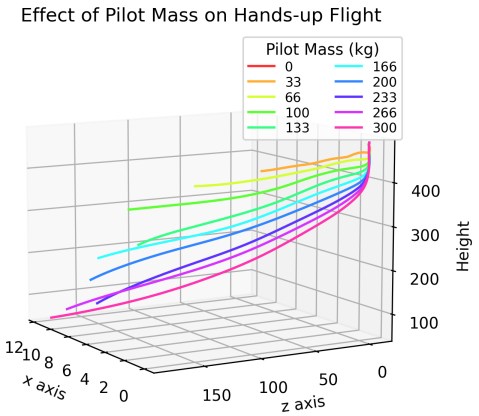

Figure 4: Comparison of paraglider flight paths with differing pilot masses

The path for the pilot with mass 0kg is not visible, as Bullet treats objects with zero mass as immovable, and so the wing does not move. As the pilot mass increases, the wing falls further to reach steady flight. Once the wing has reached steady flight, however, the glide ratios appear to be similar - for example, the glide ratio at the end of the flight for the 33kg and 300kg pilots appears similar. The heavier pilots cause the wing to accelerate to a much higher speed than the lighter pilots.

3.3 Behavior Observations

This section will give a couple examples of observed behaviors with brief explanations. It is suggested that the reader *try the simulation themselves*. It is interactive and real time, so provides an excellent sense of how the wing behaves and "feels".

3.3.1 Initialisation through to Steady Flight

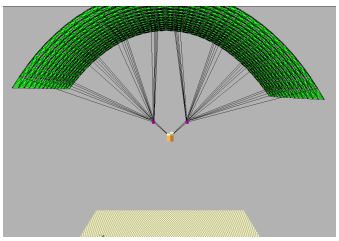

Figure 5: State of the default wing, as it is initialized. The simulation is paused before resetting the wing, so this is exactly as it appears before any forces have been applied.

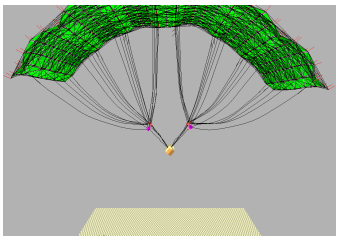

Figure 6: State of the default wing less than a second after initialization

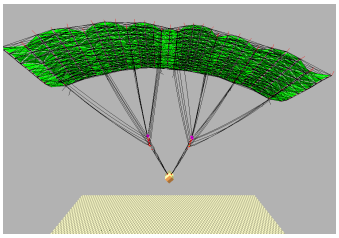

Figure 7: State of the default wing a few seconds after initialization, as it begins to enter steady flight

Figures 5, 6 and 7 show the wing from when it is initialized to when it begins to settle into steady flight. When the wing is created (figure 5), the shape of the wing and angles of the risers are arbitrary - the way it is created is not done with any sense of the final shape. For example, the lowest risers are at 45 deg, whereas they become steeped once the wing has settled. This leads to a lack of tension and loss of shape immediately after creation; this is seen in figure 6, as the risers loose shape. Finally, the load is spread throughout the risers evenly, and the wing is deflected (typically flattened) to match, and the wing displays steady flight.

3.3.2 Braking to Stall

Earlier, in reference to the "both brakes" test (figure 3.2.3), we mentioned that pulling both brakes for too long causes the wing to dive *backwards*. To demonstrate, in figure 8 you can both brakes being pulled, the pilot beginning to swing forwards and the wing pitch up. Figure 9 is taken a few seconds later, where the wing is now perpendicular to the ground, and the rear of the wing is towards the ground.

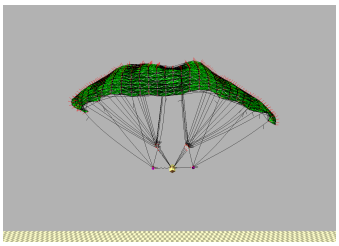

Figure 8: Both brakes being applied to default wing after resetting and settling in to steady flight.

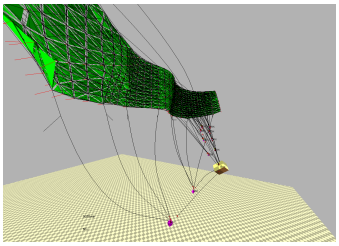

Figure 9: Both brakes still applied, however pilot has swung forward and wing pitched up, and now falling to ground.

4 Discussion

This discussion is written from the perspective of one of the authors - Tom Stephen, 3rd student studying BMath/BSci at the University of Queensland.

4.1 Retrospect

This project was completed[4] for a summer work experience program run through the University of Queensland. Thus, there were two purposes for completing the project:

- to lay the groundwork for a tool that, hopefully, will one day be used to help improve the sport of paragliding in terms of safety, accessibility and education
- to practice and develop my abilities in software design, physics, scientific and professional communication, etc.

For the second point, I believe this project has been a success. I am *far* more capable and experienced with C++, as well as working with large, complex and powerful libraries (Bullet and OpenGL). Additionally, in writing this report and my frequent summary emails, I've had of practice explaining my work in succinct and dense ways. Finally, this has been a large project, with many features to investigate, research, implement and test, and so has been a good test of my time management and planning.

In relation to the first point, I believe the tests above show that the simulation is exhibiting behaviors similar to those of a real paraglider wing. Although not entirely my own work, the project is still built in such a way that it is extensible, and particularly the work for initializing the wing and physics model is accessible.

The simulation is observed to be repeatable as well. For example, in all tests with the default wing (so all tests excluding the varying mass runs), the first section of the run is identical up until brakes are applied/etc. This suggests no significant numerical/computational errors are appearing immediately. This repeatability may also be seen across the first two tests, where the 10s brake test and 1.0x brake test had the same path - in both tests, the left brake is applied for 10s at full intensity. This consistency and repeatability suggests the simulation is rather precise.

Also, I believe the system for inputting configuration and command input files, and outputting position log files is well designed. The code to implement it is easily extensible, so the files may be extended to take more inputs (such as environment or wing parameters), as well as log additional positions (discussed below).

4.2 A look forward

As mentioned, the work done so far simply lays the groundwork for the potential software ahead. For example, the wing-customization aspect could be expanded, implementing features such as:

- a system to design more complex riser patterns, or imitate riser patterns from real wings
- extended the wing offset code to create more complex wing shapes
- consider two-skin wings, as a more realistic model of real wings

[4] "completed" is not the best word - if I had more time, I've got a lot I'd like to add!

Additionally, more testing is required to determine the precise, practical effects of parameters such as bending constraints, linear, angular and volumetric resistance forces, wing resolutions[5] and so forth.

Currently, the simulation *must* run in realtime, and render the scene. However, when completing batch testing, it would be ideal for the scene to not be rendered, and the simulation to run as fast as possible - otherwise the time to run batches is limited by the simulated length of each test.

If more positions were logged (for example, the positions of the wing tips, or the position of the center of the wing), analyses of spiral drives and other situations where the relation in position between the wing and the pilot is important would be possible. Also, it should be possible to find and log the force through each riser - this would allow for determining which risers are loaded more and under which scenarios (for example, perhaps a wing is safe for cruising, but the force in a specific riser would exceed its tensile strength).

A key component to paragliding is thermals and wind, which we have made very little attempt at simulating so far. Perhaps implementing a basic terrain system, and realistic thermals and wind based off of real-life locations would improve the experience. This may also allow users to compare the accuracy of the simulation to real, recorded flights.

Finally, for a consumer-facing product, significant work should be done to the UI. Since it is built with OpenGL, anything should be possible, particularly and update to the graphics, more detailed help information and options that does not obstruct the simulation, and perhaps real-time plotting of positions/forces/velocities/etc.

5 Conclusion

In this project, we attempted to design and build a tool to simulate the flight of paraglider wings. Using C++, the Bullet physics library, and OpenGL for graphics, we were able to simulate in realtime what appears to be a realistic paraglider wing. The wing has been shown to react realistically to braking and changing rider mass. There are many improvements recommended for this project, particularly customizability of the wing, and more realistic flight environments.

[5] an idea that has been noted for some time is to dynamically adjust the wing resolution in different sections of the wing - for example, if the wing tips are bending and move more than the center of the wing, the resolution could be increased there to allow for more realistic fabric simulations