MOSI Science Festival – Very Low Earth Orbit Drag and Gas Surface Interactions Exhibit

Project Documentation

Created as a part of the SOE Summer Internship 2024

Intern – Anshika Singh

Line Manager – Dr. Katherine Smith

Contents

Introduction

This project aims to visualize the effects of Atomic Oxygen particles on satellites in Very Low Earth Orbit, specifically the drag force caused by particle collisions and gas-surface interactions between the particles and the satellite's surface. Unity Game Engine was used to create this visualisation and some external processing was done using MATLAB.

My aim with this project was to make it as interactive and game-like as possible to increases user engagement while also retaining technical accuracy to ensure the most educational impact. Future development of this project could involve applying more sophisticated methods in various steps of the programme. This is discussed further in the "Future Scope" section and in the "Future Scope Flowchart".

This document contains information about the development of this visualisation including various features that I created but were discarded from the final version of the project. Appendices B & C have screenshots of the Hierarchy and Inspector Settings that will be a useful reference for each section of this document. Furthermore, information about the GitHub repository, flowcharts and the link to a brief video I created explaining how everything works in the project are provided in Appendix D.

1. Satellite Models

Satellite models are created in Solidworks (or other CAD software) and imported into MeshLab or another software for meshing. The .obj files are then imported as assets into the Unity project. At this stage it is important to ensure that different satellite parts are meshed and exported separately to allow Unity to apply different materials on different parts of the satellite. For the project, initially, I worked with simple cylinders with rectangular solar panels, cubes and spheres. Then, I progressed to using CAD models inspired by the SkimSat, Starlink satellites, and the SabreSat.

1.1 Assembling parts in Unity

The simple in-built 3D objects in Unity were used for basic testing purposes and were connected with fixed joints with infinite break force and infinite break torque. This was done to ensure that the forces exerted by the oncoming particles don't break apart the parts. It was more efficient to test out code and different features of the simulation on simple objects before applying these to more detailed objects.

1.2 Mesh Colliders

Satellite objects need to have mesh colliders that are convex and non-kinematic rigid bodies for showing the drag effects of ATOX. However, for showing different GSI types and collision locations the mesh needs to be concave and rigid body needs to be kinematic. Hence, combining both of these is a non-trivial task.

An alternative to this is found by having multiple stationary "tiles" of different materials that display the varying GSI interactions. Furthermore, an alternative to UV colouring (discussed in detail later) is found by using sub-emitters in the main particle system.

1.3 Materials

Satellites can use multiple custom made materials with different colours and textures. The textures to be used for collision colour changes are discussed in depth in later sections.

Satellites also have different Physic Materials which control the physics of rigid bodies in Unity. The Dynamic Friction and Static Friction were found to not be important when showing the drag effects of ATOX particles. The Bounciness of the physic material is more important than the other properties but is still made irrelevant when working with particle systems. This is because the particle system collision model has a variety of in-builts tools that I have found to be more useful in achieving the desired results. Hence, I decided to not focus too much on different physic materials and chose to stick with one Physic Material with constant properties for the project.

As mentioned earlier, Unity does not allow the same mesh to have different materials. Hence, I found that the best way to achieve this is to import the parts of the satellite such as the main body, solar panels, nose, etc. as separate .obj files and connect them in Unity. I created four different materials that were used in my final scene. Two of these were programmed to respond to particles by giving "diffuse emissions" while the other two used Unity's in-built engine that gives "specular emissions". This is discussed in more detail later.

1.4 Physics Settings

The satellite Game Objects are rigidbodies with gravity turned off. They also need to have convex mesh colliders with "Is Trigger" turned off. The drag and angular drag should be set to 0 to avoid unwanted motion of the satellite.

1.5 Scripts

The satellite's forward motion is controlled by a C# script that gives it a constant forward velocity. This velocity can be set in the Inspector of the Satellite Object in the "translation speed" space. There is also an additional component that reduces the satellite speed by a certain fraction when particles collide with it. This further illustrates the drag effect of the ATOX particles as the satellite would need to use more propulsion to compensate for the drag force given to it as ATOX particles lose kinetic energy upon collision. This, too, can be controlled using the "collision energy reduction" property in the Inspector of the Satellite Object.

2. Particle System

Particle Systems are created to represent Atmoic Oxygen particles and to display the thrusters on the satellite. They ATOX particles are used primarily for two purposes – showing the drag effects and de-orbiting of VLEO satellites and showing the different Gas-surface interactions on VLEO satellites' surfaces.

2.1 Thrusters

The thrusters have been created using three particle systems. The thrusters do not have colliders. They have trails and reducing size over lifetime. All three particle systems have a cone shape emitter. The main thruster is always active while the thrusters on the additional two axes are activated by a key press. This is discussed further in later sections.

2.2 Scripts

The additional 4 particle systems that collide with stationary tiles of different materials have a Particle Collision Handler script attached to them. The details of the UI elements and GSI modes controlled by this script are discussed in later sections.

2.3 Trails

Particle Systems need to have trails to show the path followed by the particles and also have a Colour by Speed property. The latter ensures that when particles hit another collider (satellite) and experience a change in speed due to loss of kinetic energy to the satellite they are represented by different colours. The trails are in particle mode and are set to inherit particle colour to indicate different speeds and also die with particles. The render mode is set to Billboard as it gives the most visually appealing results.

2.4 Collision

Particle systems also have colliders with high collider forces that allow the much heavier satellite game object to experience movement after being struck by particles. It is to be noted that “send collision messages” must be enabled to ensure that C# scripts can be used to control/monitor collision events.

2.5 Particle index issue

Particle Systems are set with Looping turned off and a much higher Start Lifetime than the simulation run time. This was done because I found that whenever I was using particle indices to measure/monitor velocities or forces, the indices would start to repeat after some time. For example, Particles would be indexed 0 to 19 and then re-start from 0. This is probably because Unity re-assigns the numbers of particles after their lifetime has ended. Since I couldn’t find any simple fix for this, increasing the particle lifetime is the best way to mitigate this issue.

The final scene includes 4 additional particle systems that have components to represent different Gas-surface interactions. These have different settings and scripts attached to them to display the GSIs in the best way possible. This is discussed in detail later.

3. Gameplay

Gameplay is an important part of the project that adds an interactive user experience. Creating this part of the project was very exciting for me as it allowed me to combine my theoretical knowledge on Orbital Mechanics, Propulsion Systems and Attitude Control and Determination Systems (ADCS) with my programming skills and creative ideas.

3.1 Keys

All gameplay features are controlled using key presses. I have also created a flowchart of exactly how these key presses, scripts and methods connect with each other. I believe this will be extremely useful to understand the control flow of the code.

The different keys and what they control are also briefly explained here –

Spacebar – Add force to thrust forward and reduce the height of the fuel gauge

R – Add torque to rotate the satellite back to the original rotation

T – Activate additional thrusters that move the satellite back to its original rotation and reduce the height of the fuel gauge

U – Activate the upwards thruster to increase the satellite altitude and reduce the height of the fuel gauge

3.2 Thrust and Satellite Speed

As the satellite gets hit by particles, it loses forward speed due to getting force in the opposite direction. Displaying this effect is very useful and, so, I added a satellite speed indicator on screen in display 2. The loss of speed is controlled by the HandleCollision Method in the Translate Satellite Script. The different parts of the satellite that are children of the main body call this method through the Child Object Collision Handler script. The fraction of speed lost can be edited in the Inspector. Adding this feature was difficult since changing the speed on any of the satellite children objects would mess up the speed of the whole satellite assembly, but I also needed to change the speed if any collisions occurred with any of the children objects.

Satellite propulsion systems need to burn more fuel to counteract this drag force and maintain speed. This additional thrust force is added by pressing the spacebar. This gives the satellite some forward thrust but also lowers the fuel gauge. The amount of thrust force added for each press can also be edited in the Inspector. Once the fuel gauge has reached a height of 0 no more thrust can be added and “No fuel left” is indicated on screen when space is pressed.

3.3 ADCS – Reaction Wheels and Thrusters

I created an ADCS system that uses reaction wheels to provide torque to move the satellite back to its original orientation. This system works when the R key is pressed and is controlled by the Rotate Satellite script. The rotation speed can be altered in the Inspector and determines how fast the satellite is able to rotate back to its original location. When the ADCS is operative, “ADCS turned on” is displayed on screen. GUI is discussed in detail in a later section.

Additionally, I have added a feature that counts the number of times the ADCS is used and for how long. When either the maximum count value or the maximum press time have been exceeded the ADCS is saturated and does not work. This is indicated to the user and they are prompted to activate the thrusters. The max. count value and max. press time can both be easily altered in the script.

```
if (Input.GetKey(resetKey))
{
    if (!resetKeyHeld){
        resetKeyPressCount++;
        resetKeyHeld = true;
        //resetKeyPressDuration = 0f; // Reset duration on each new press
        //Debug.Log("ADCS Press Count: " + resetKeyPressCount);
    }

    if(resetKeyPressCount < 10 && resetKeyPressDuration < saturationTime)
    {
    //Debug.Log("turning!");
    transform.rotation = Quaternion.Lerp(transform.rotation, targetRotation, rotationSpeed * Time.deltaTime);
    textMeshPro.text = $"<color=green>ADCS Turned on";
    }
}
else
{
    resetKeyHeld = false;
    textMeshPro.text = $"<color=red>ADCS Turned off";
}
```

If the user presses T to activate thrusters on all three axes, the satellite immediately returns to its original orientation. However, this consumes fuel faster than the reaction wheels and the fuel gauge lowers quickly. A desaturation timer is active and when the T key has been pressed for this duration, the reaction wheels become de-saturated and can be used for attitude control.

The desaturation timer can be controlled in the Inspector. The thrusters being operative is shown in the simulation by activating additional particle systems from the Thruster script.

3.4 Satellite Altitude and $\Delta \mathbf{V}$

As the satellite loses kinetic energy it also experiences a loss in altitude. In VLEO, this effect is amplified by the high density of ATOX particles.Hence, I chose to take advantage of the zoomed out satellite view to show this effect. The Drag Altitude Change script controls this feature.

When the satellite gets hit by a particle, a small amount of downwards force is applied to it and the on screen altitude decreases. As the altitude drops another indicator displays the ΔV needed to perform a Hohmann transfer manoeuvre back to the original orbit. I chose the Hohmann transfer to calculate ΔV as it is the most propellant efficient manoeuvre. The image below is a screenshot of the method that calculates the ΔV.

```
public void CalculateDeltaV(float newAltitude)
{
    //Calculating delta V needed
    float radiusEarth = 6371; //km
    float gravitationalParameter = 398800; //km3/s2
    float a = (newAltitude + altitude + (2*radiusEarth))/2;
    float r_p = radiusEarth + newAltitude;
    float r_a = radiusEarth + altitude;
    deltaV = Mathf.Abs(Mathf.Sqrt(((2*gravitationalParameter)/r_p)-(gravitationalParameter/a))-Mathf.Sqrt(gravitationalParameter/r_p)) + Mathf.Abs(Mathf.Sqrt(gravitationalParameter/r_a)-Mathf.Sqrt(((2*gravitationalParameter)/r_a)-(gravitationalParameter/a)));
    textMeshProDeltaV.text = $"<color=yellow>Delta V needed to manoeuvre = {deltaV} km/s";
}
```

When U is pressed the satellite receives a slight upwards force that increases the altitude. As the satellite altitude changes an indicator gives the user a warning if the altitude is outside a certain range. I chose a starting altitude of 400 km and display a warning when altitude is below 395 km or above 405 km.

4. Colour Change Upon Collision

One of the most important things to visualize about interaction between ATOX particles and the surfaces of satellites in VLEO is the locations where these collisions occur. The motion of the satellite depends on the location of these collisions relative to the centre of mass of the satellite. The centre of mass is discussed in detail later. Hence, I spent a lot of time focusing on a way to display these collision points on the satellite in real-time.

4.1 Sub-emitters

The first method I used for this was to have sub-emitters on the particle system that trigger upon collision. These sub-emitters are created with a new material that has a circular texture (initially, the shape was a splatter but was changed later). The sub-emitters have looping turned off and have 0 start speed to ensure they "stick" to the satellite's surface. The simulation space of the sub-emitter particle system is set to be custom. This is done so they stay relative to the satellite object's transform.

The emission is set to 0 rate over time and uses bursts as collision mode sub-emitters need burst events to work. The cycles are set to infinite and the count is set to 10. For unknown reasons when the count was less than this the sub-emitters weren't working properly.

The collisions for these are turned on so that collision messages can be used in scripts, however, since I do not want sub-emitters to impact the physics of the satellite the collider force is set to 0.

The sub-emitter particle system has a script attached to it that makes the individual sub-emitter particles children of the satellite object upon collision. This is done so that they "Stick" to the satellite surface until the end of the simulation and are indicative of the collision locations. However, despite these settings due to the nature of the sub-emitters emitting after a collision has occurred they do, sometimes, appear a bit off the actual satellite surface.

These sub-emitters are eliminated from the additional 4 particle systems that are used to visualise gas-surface interactions as the location of collisions is not significant.

4.2 UV colouring with raycasts

An alternative to the sub-emitter method discussed above is to use UV colouring on satellite material textures. This is a much more complicated method that makes the use of raycasts to colour in specific pixels on the satellite object. This method is highly accurate but only works with non-convex mesh colliders. In Unity, only kinematic rigid bodies can have non-convex mesh colliders which, then, makes the object non-responsive to physics as stated in Unity's documentation. "If isKinematic is enabled, Forces, collisions or joints will not affect the rigidbody anymore. The rigidbody will be under full control of animation or script control by changing transform.position. Kinematic bodies also affect the motion of other rigidbodies through collisions or joints." Due to its high accuracy this method is very useful for a stationary object where the collision locations and angles of incidence and reflection are to be looked at closely.

This method makes use of a raycast (invisible to the user) from the main camera to the satellite surface and then uses the position of the collision to colour in the pixel at that point. I, initially, created this method to respond to the user clicking the surface at the point of collision and then, integrated it with the GSI scripts I had already made. This method, in its final state, detects the collision point between the particle and the satellite surface and fires a raycast along that direction, uses the UV coordinates to find the point of collision and, then, colours in a small square of pixels around that point.

For this method to work, the satellite materials must have a texture on them. I added this to the Base Map in the Universal Render Pipeline. The C# script has four properties that can be set in the Inspector of the Particle System the script is attached to. The first one is the particle system

itself, the second one called “Hit Color” is the colour to change the collision point to, the third property called “Collision Target” is the game object that the particles should collide with and the last one is to select the camera to do the raycasts from. Here, it must be noted that the camera to do raycasts from should be parallel to the particle stream and close to the object so the raycast does not collide with any other object.

Furthermore, the configuration of the material textures is very important when using this method. The format must be set to RGBA 32 bit for both Default and PC settings. Also, the Read/Write in the Advanced settings must be enabled to ensure the texture is “colourable”. Another issue I encountered while creating this method was that the texture was retaining the coloured pixels from previous simulations and was never blank. Hence, I created another C# script to reset the texture of the satellite object every time a new simulation is instantiated. This creates a copy of the texture on the surface of the satellite and then uses that to apply the method stated above ensuring that each time a new “blank copy” of the texture is used.

5. Gas Surface Interactions - Diffuse and Specular Emissions

Visualising different Gas-Surface Interactions is another key part of the project. Hence, I decided to spend a lot of time understanding the GSI model of Unity. To do this, I created three different scripts to monitor/measure the velocity and rotation of the satellite, the velocity of the particles in the particle system and the location and normal to the surface of the collision events. These scripts called “ParticleCollisionDetector” and “SatelliteVelocityMonitor” output data into three separate .csv files. I then sorted through the data to create tables that describe the collision model used by Unity.

The first part of this analysis was to identify how the change in particle velocities corresponded to the change in the velocity and rotation of the satellite. I found that based on the settings of drag and bounciness in the particle system collision model, the satellite object got a fraction of the kinetic energy lost by the particles.

I, further, created a script in MATLAB called “Plot_vectors.mlx” to plot the velocity vectors for each of the particles before and after the collision at the point of collisions as well as the normals to the surface at the points of collision. This script also calculated the angles of incidence and reflection of these particles based on this data. I repeated this process several times for a variety of different angles of incidence. An example of cleaned tabulated data from the three separate .csv files that can be directly imported into the MATLAB script is provided below and is also available in GitHub repository.

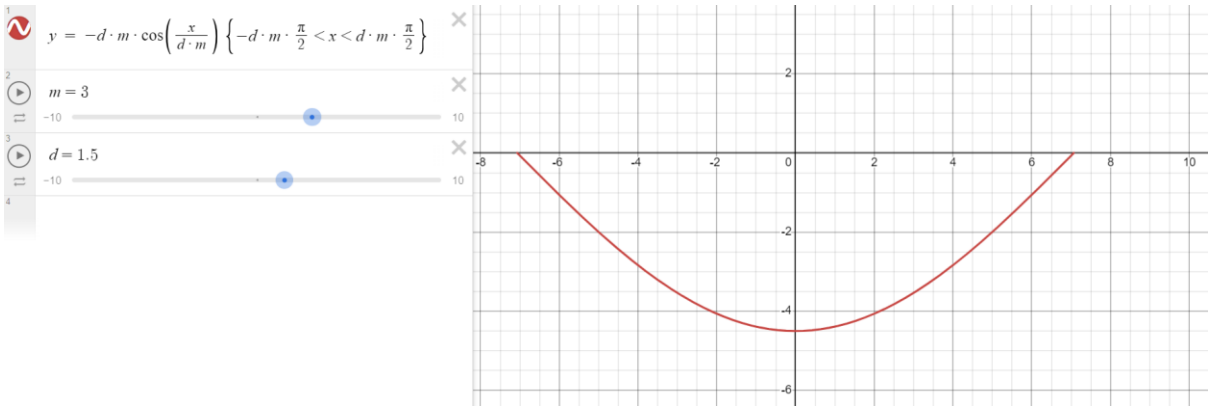
Particle No.	Velocity Before Collision X	Velocity Before Collision Y	Velocity Before Collision Z	Velocity after collision X	Velocity after collision Y	Velocity after collision Z	Collision Location X	Collision Location Y	Collision Location Z	Collision Normal X	Collision Normal Y	Collision Normal Z
0	11.63092	5.36E-06	27.65394	-25.62306	5.72E-06	-15.26393	-5.306361	5.650336	-1.883453	-0.6574581	-2.1E-05	-0.7534314
1	11.63092	5.36E-06	27.65394	-23.63035	6.64E-06	4.343866	-5.306397	5.497451	-1.779145	-0.8703056	2.05E-06	-0.4913974
2	11.63092	5.36E-06	27.65394	-11.62911	5.72E-06	27.65411	-5.453763	4.668428	-0.8364611	-1	4.76E-06	7.36E-06
3	11.63092	5.36E-06	27.65394	-23.61281	-3.62E-05	6.603778	-5.460618	4.646633	-1.775245	-0.8747201	-2.15E-06	-0.4846286
4	11.63092	5.36E-06	27.65394	11.62928	5.72E-06	-27.65431	-5.002934	5.235395	-1.93039	-1.52E-05	-1.09E-06	-1
5	11.63092	5.36E-06	27.65394	-11.62909	5.72E-06	27.65428	-5.453773	5.436071	-1.092104	-1	1.51E-06	1.86E-05
6	11.63092	5.36E-06	27.65394	-26.10604	3.00E-06	14.77706	-5.431041	5.422973	-1.706521	-0.3464236	3.73E-06	-0.9228279
7	11.63092	5.36E-06	27.65394	-11.62666	7.62E-06	27.65456	-5.453765	5.295426	-0.7137513	-1	8.31E-06	2.63E-05
8	11.63092	5.36E-06	27.65394	-28.42676	1.50E-05	9.597452	-5.449253	4.64358	-1.743193	-0.31513	2.21E-07	-0.4911595
9	11.63092	5.36E-06	27.65394	-6.305801	1.91E-05	-29.19436	-5.360883	5.219094	-1.972947	-0.309397	2.42E-07	-0.9507378
10	11.63092	5.36E-06	27.65394	-11.62803	7.63E-06	27.65402	-5.453766	4.532594	-0.8054352	-1	9.41E-06	3.71E-05

To conduct this analysis I created a simple scene to launch one particle at a time on a stationary wall. I found that Unity uses a specular approach with the angles of reflection only changing based on the damping and bounciness settings of the particle system. With damping = 0 and bounciness = 1, the angle of incidence and reflection are equal. Thus, Unity uses an almost quasi-specular approach. However, this cannot be considered diffuse emission as the particle are never reflected on the same side of the normal as the incoming particles.

Hence, I decided to create a C# script called Particle Collision Handler that implements a diffuse approach. This script is attached to the additional 4 particle systems that fire one particle at a time on 4 stationary “tiles” of different materials.

This script has five properties that can be changed in the Inspector – the particle system to be controlled, the damping fraction which defines how much kinetic energy the particles lose upon collision, the text mesh pro UI element to display velocities and the colours to display the initial and final velocities. The damping fraction is used as a representation of the thermal accomodation factor and different materials can be given different damping fractions. This script, thus, changes the magnitude of the velocity after the collision and assigns a new random velocity in the x direction within limits set using the damping fraction. This x velocity is then used to calculate a velocity in the y direction that plots the velocity vectors on a cosine curve as characteristic to diffuse emissions. These calculations use bespoke mathematical functions that I created. For a clearer understanding the function is plotted on a 2D graph as shown below. Here, d represents the damping fraction and m represents the initial magnitude of the velocity. A value of x velocity is assigned within the limits shown and the corresponding y velocity is then calculated.

The UI elements and displays are discussed in detail in later sections.



6. Visual Elements

The project focuses of creating a visualisation tool that conveys the research on VLEO satellites and GSI models to the general public at the MOSI Science Festival. Hence, adding features which make the tool visually appealing and clearly understandable is important. Additionally, GUI elements have been added to all three displays and these are discussed in more detail in the next section.

6.1 Displays

The latest version of the simulation has 3 displays – zoomed out satellite view, zoomed in satellite view, and GSI view for different materials. The zoomed out satellite view showcases the particle stream and the motion of the satellite. In this display mode, identifying the effects of drag, change in altitude and changes in orientation of the satellite are very easy. The zoomed in satellite view showcases the different materials on the satellite the collision locations and intricacies of the energy exchange between the particles and the satellite. The GSI view has a

number of different 3D “tile” objects of different materials with their own particle systems. In this display, the different types of emissions – specular and diffuse – and loss of kinetic energy of particles after the collisions with different materials are really easily identifiable. All three displays have UI elements associated with them that are discussed in a section below.

6.2 Cameras

The latest version of the simulation has six different cameras for the three different displays. It is to be noted here that the “Audio Listener” must be turned off for all except one camera to avoid any errors. The four cameras named “Material_cam”, “Material_cam(1)”, “Material_cam(2)” and “Material_cam(3)” are stationary and used for the GSI view. They are placed in front of the four materials and all output to display 3.

Display 3 has a split screen view between all 4 cameras and the “Viewport Rect” settings have been manipulated to achieve this. This is done by setting the width and height to 0.5 each, and X and Y are set to (0,0), (0,0.5),(0.5,0) and (0.5,0.5).

The “Zoomed Camera” is used for the zoomed in satellite view. The “Main Camera” is used for a zoomed out satellite view. These cameras use the “Move Setup” C# script to move and the speed must be changed in the Inspector if the satellite’s initial translation speed is changed. “Zoomed Camera” moves at the same speed as the satellite’s initial translation speed, while the “Main Camera” moves at a slightly slower speed.

6.3 Lighting

The latest version of the simulation has two directional lights – Directional Light and Satellite Light. The former is used to light up the whole scene and make everything visible. It has a high intesity and is placed much above on the global y-axis than the other game objects. The Satellite Light is placed slightly below and in front of the satellite and has a lower intensity. This provides a clearer view of the collision location and illuminates the areas of the satellite shadowed by the main light source. The satellite light also moves with the same speed as the camera described previously using the same C# script.

The background view uses a “Sky_Midnight” skybox that I downloaded when undertaking the Unity Learn – Essentials pathway.

7. User Interface

All the features discussed in the earlier sections need to be clearly communicated to the users. Hence, I focused a lot on adding appropriate UI elements. Display1 has Satellite Altitude, ΔV and a Altitude Warning indicators. Display 2 has Satellite speed, ADCS and Thrust indicators and a fuel gauge. Display 3 has Particle Velocities and Particle colour indicators. Appendix A has screenshots of all three displays.

All the UI elements are displayed on a Canvas set to Screen Space – Camera render mode.

7.1 Satellite Altitude and $\Delta \mathbf{V}$

Satellite Altitude is displayed in Display 1. As the satellite altitude drops this indicator updates the new altitude. The ΔV needed to perform the manoeuvre to the original orbit is also displayed on the top right of the screen. The altitude warning indicator is on the bottom centre of the

screen and displays “Maintaining 400 km altitude” initially. If the altitude drops below 395 km “Altitude too low” is displayed and if the altitude is above 405 km “Altitude too high" is displayed. If the user manoeuvres back to the original altitude “Original altitude attained” is displyed.

7.2 Fuel Gauge & Thrust

A fuel gauge is displayed on the left side of the screen in display 2. I created this using 3 images – one to form the white outline, one to have the gray empty colour, and one to have the orange fuel colour. The fuel gauge is set to have a (0,0.5) anchor point. The fuel gauge controls are discussed in earlier sections. Whenever fuel gauge reduces, the height of the orange fuel colour image is decreased. The amount the height decreases by can be edited in the Inspector.

When thrust is added to the satellite, “Increasing thrust force!” is displayed on the bottom centre and if no thrust is added “No additional thrust force” is displayed.

7.3 Satellite Speed

Display 2 also shows the satellite’s current speed on the top left corner of the screen. This is controlled by Translate Satellite script and the Display speed method inside the script.

7.4 ADCS

If the ADCS is ineoprative, “ADCS turned off” is displayed on the top-right corner of the screen, else “ADCS turned on” is displayed. When the ADCS is saturated, “Reaction wheels saturated… Press T to desaturate” is displayed. While additional thrusters are used “Desaturating reaction wheels…” is displayed and once the desaturation time is up “Reaction wheels desaturated” is displayed.

7.5 Particle Speed & Colour by Speed Gradient

Display 3 is a split screen view that displays the initial and final velocities of particles on each material. Here, either the velocity vectors can be displayed or the velocity magnitudes. I chose to display the magnitudes as they are easier to understand for people and, also, because the directions of the particles are clearly visible. An additional colour by speed gradient is displayed on the bottom-right of the screen to allow people to easily associate the varying particle and trail colours with the speed they represent. The Particle Collision Handler script controls the displaying of the velocities and the GSI modes. The GSI modes have been discussed in detail in an earlier section.

This gradient was added to an Image using the Gradient Image Script. The top, middle and bottom colours for this gradient can be chosen in the Inspector to match the particle system settings.

8. Centre of Gravity

The centre of gravity of the satellite is crucial to the way the satellite responds to collision with ATOX particles. Hence, I decided to spend some time seeing how the centre of gravity affects the response of the satellite to collisions.

However, when automatic centre of gravity is disabled in Unity, it does not display the COG on the object itself. This means that when setting up the COG location you cannot see where on

the body of the object it is located. So I decided to create my a small red sphere at the location of the centre of gravity that I place at the coordinates of the COG. This allows you to visualise the COG location much better.

I then tested out how the locations of COG affected the StarLink satellite model. I found that when the COG acts as a pivot point for the object and controls the direction of its rotation. Furthermore, the object is more resistant to rotation when the COG is at the centre of the body. However, changing the mass of individual parts of the satellite also has an impact on this. Since this idea wasn't too significant in the final version of the simulation, I chose to stick with an automatic COG.

Future Scope

Future Scope for this project could include developing and implementing more advanced algorithms at various steps. Firstly, possibilities of dividing an assembly into different sub-meshes that can have different materials in other meshing softwares can be explored further. This would reduce the effort needed to import satellite parts separately and join them using fixed joints.

Secondly, instead of reducing the satellite speed by a fixed fraction upon every collision real drag values based on altitude, angle of attack and angle of side slip found from ADBSat could be used. A lookup table can be created with these values and the appropriate one can be used inside the Translate Satellite script. This would also require adding a reference to the angle between the local satellite axes and global axes/particle system local axes.

Thirdly, the translation speed of the satellite could be calculated using the actual orbital velocity for the chosen orbit type. This would require adding an Earth game object that acts as a source of gravity for the satellite. Additionally, this would allow us to showcase realistic orbital decay and drag effects as well as have a more realistic thruster that combines both upward and forward thrust.

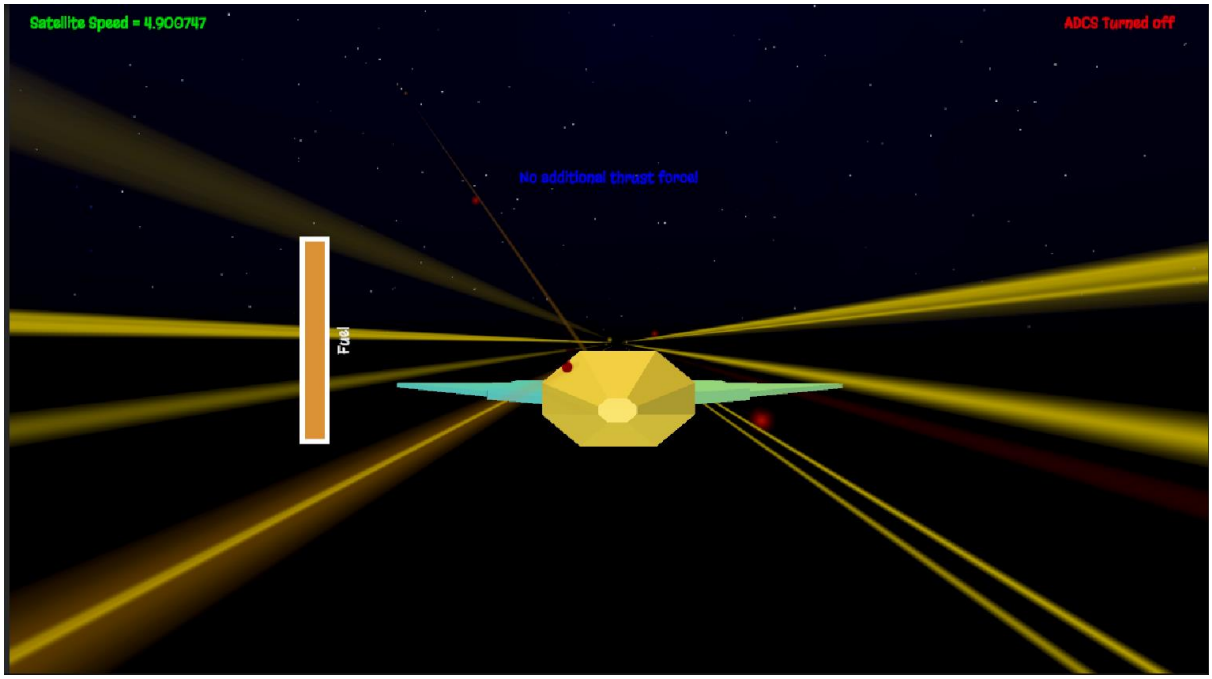
Fourthly, integrating the different gas surface interaction types with the main satellite would be extremely useful. However, this would require implementing a way to ensure that the different GSI types are apparent to the user even when the satellite isn't in its original orientation.

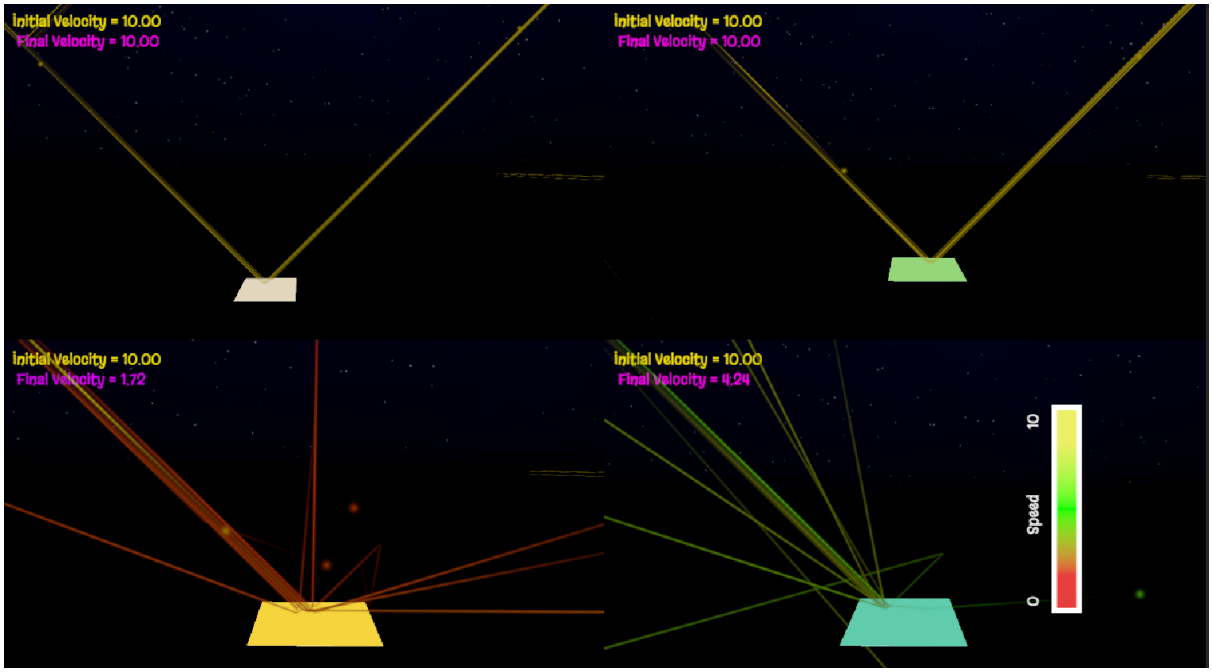
Another thing to consider is a more sophisticated script to handle the saturation of the reaction wheels in ADCS would make the visualisation more realistic and technically accurate.

The visualisation can be enhanced by finding a way to use UV colouring on convex mesh colliders, perhaps, without raycasts. I was unable to find a simple way to implement this but this would replace the sub-emitters and highly improve the accuracy of the collision locations that are shown on the satellite. This could also open up discovery of secondary collisions on the object.

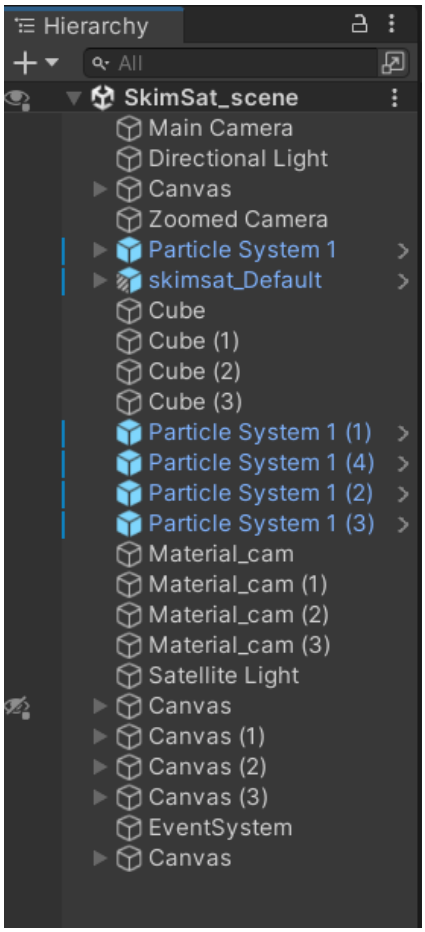
Finally, the link between the Unity Visualisation and the MATLAB live script for post-processing of the collision data currently involves several manual steps including combing through the data to create a final table that can be imported into the MATLAB script. Adding a C# or MATLAB script to automate this process would greatly improve the user experience. If such a method is implemented, analyses beyond what is currently available can be done with this data.

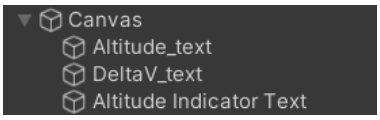
Appendix A – Different Displays

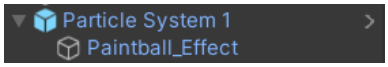


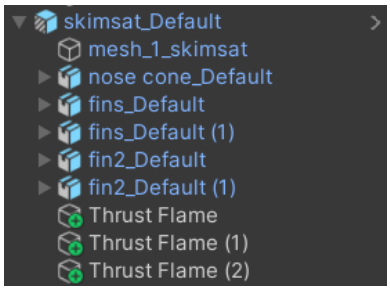

Initial Velocity = 10.00
Final Velocity = 10.00
Initial Velocity = 10.00
Final Velocity = 10.00
Initial Velocity = 10.00
Final Velocity = 1.72
Initial Velocity = 10.00
Final Velocity = 4.24
10
Speed
0

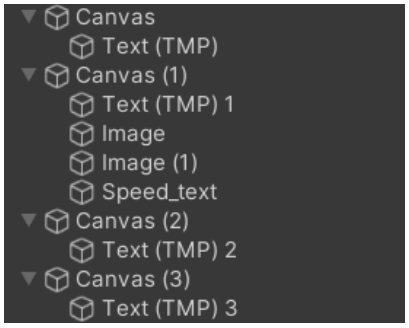
Appendix B – Hierarchy

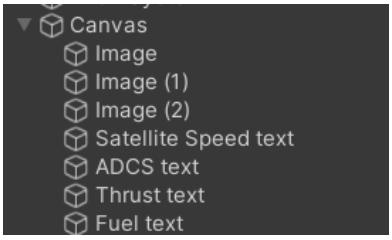






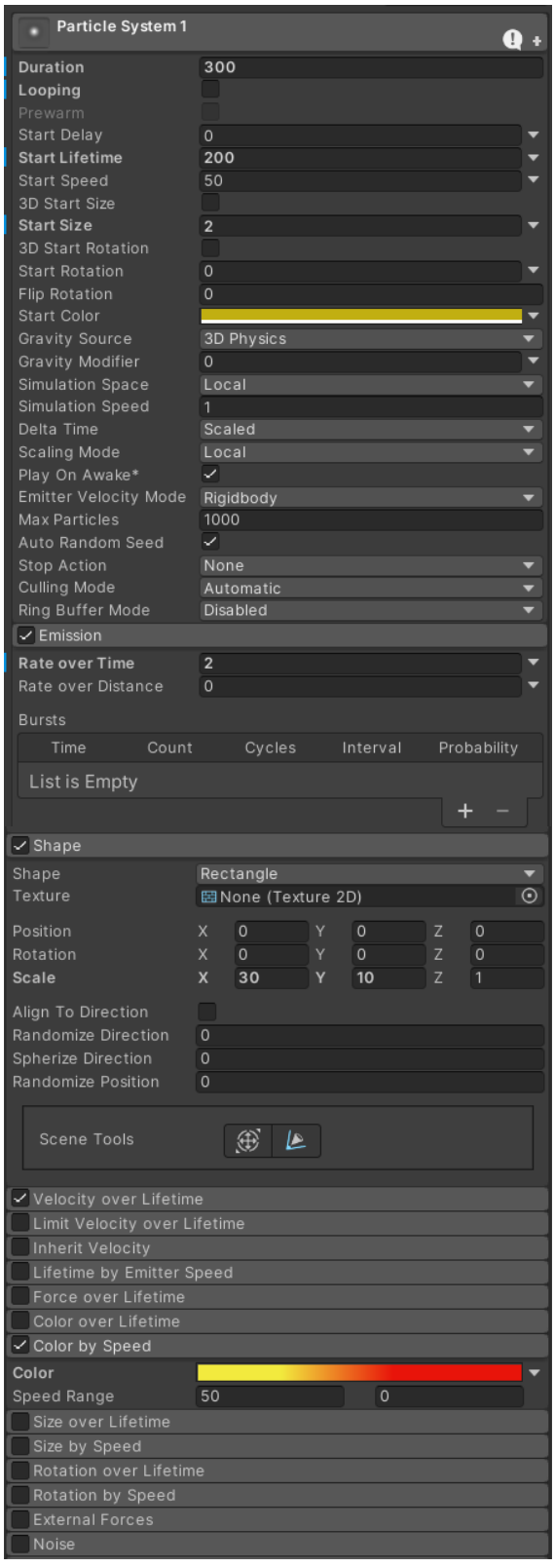


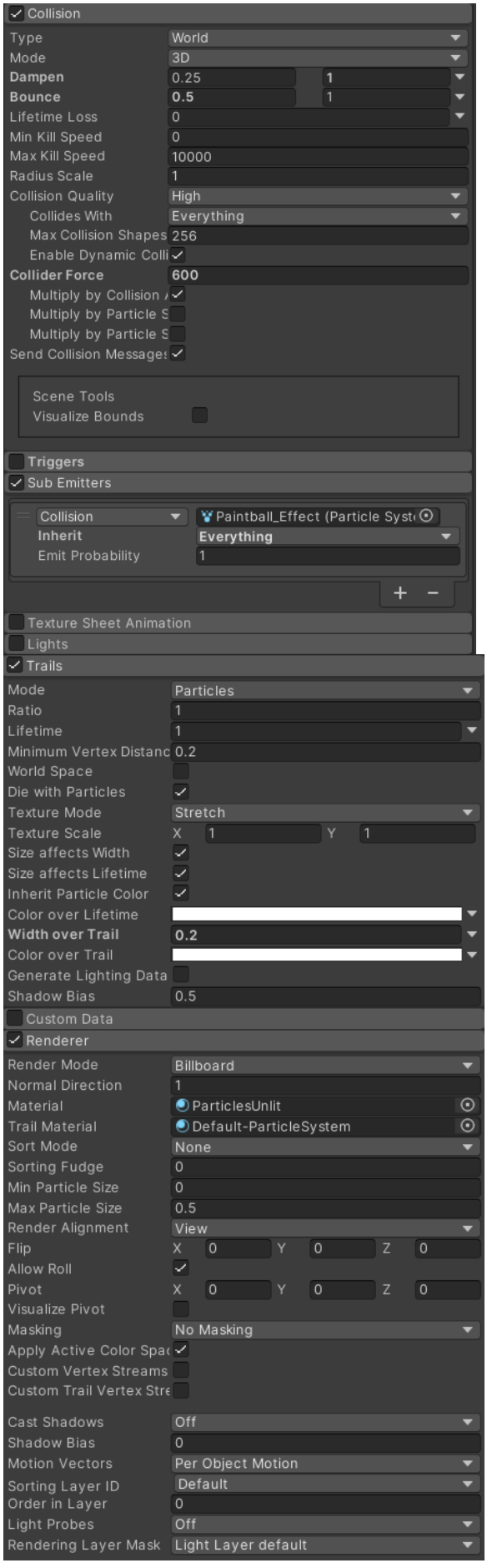




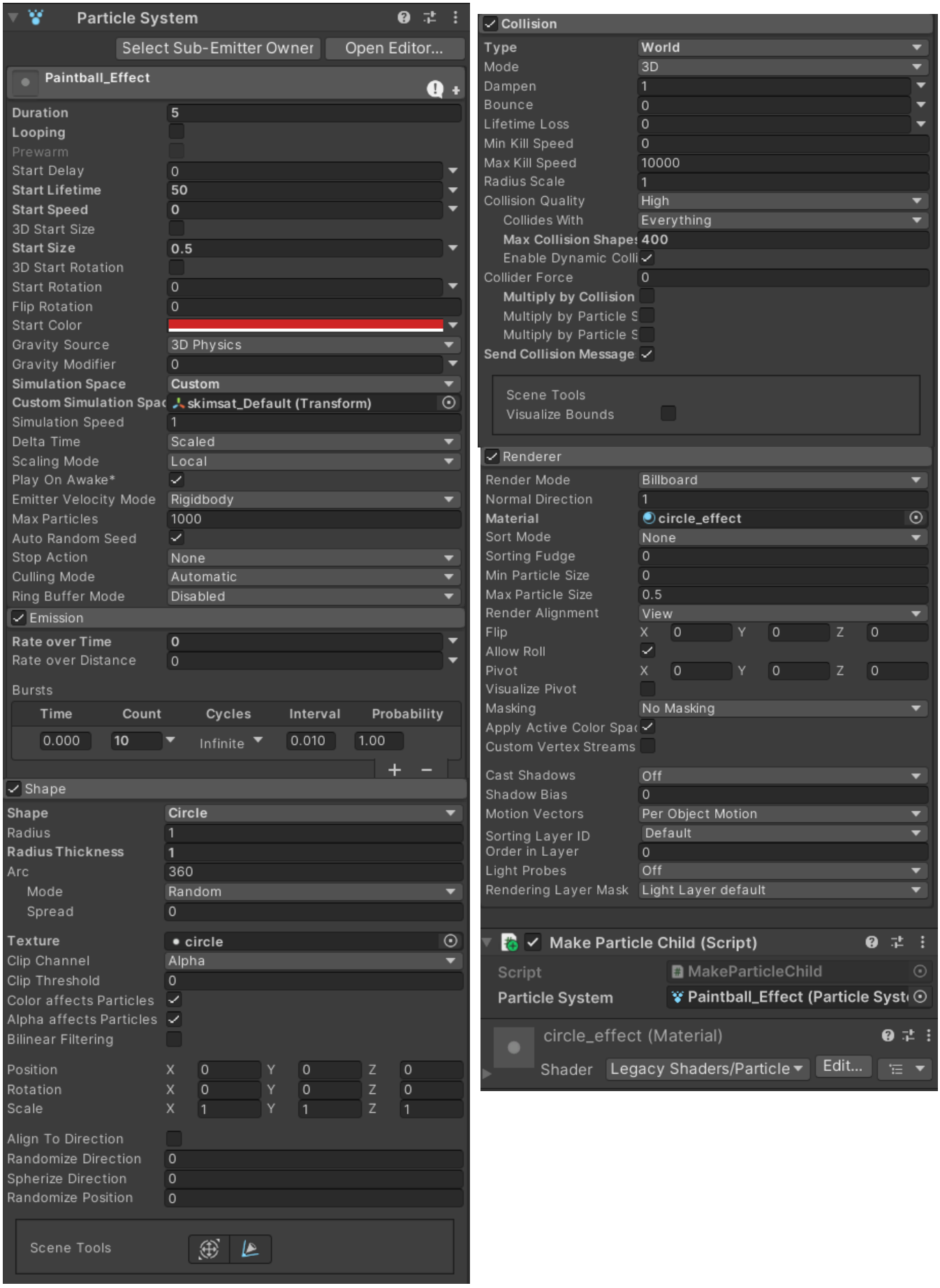
Appendix C – Inspector Settings

Main Particle System

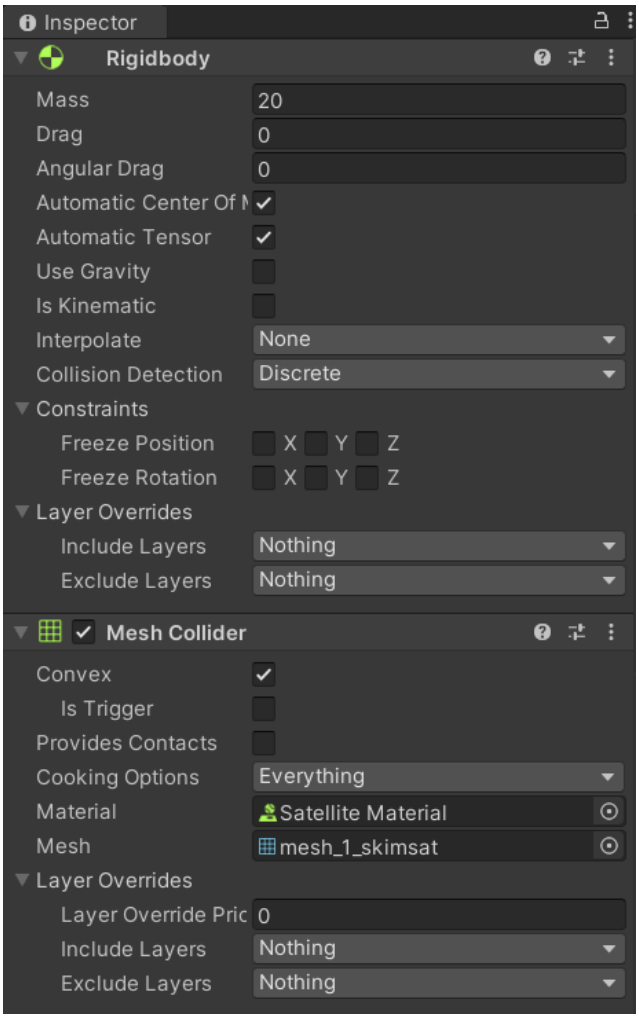


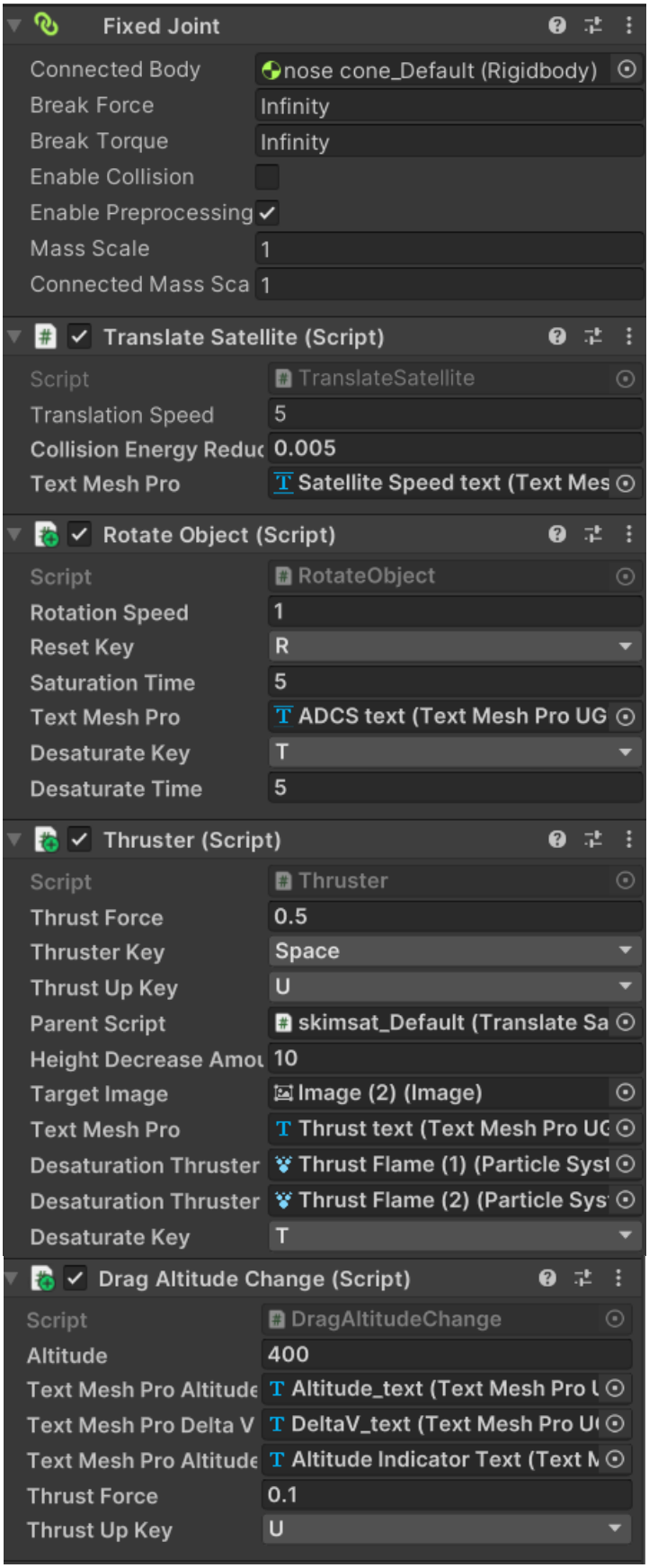


Sub-emitter Particle System

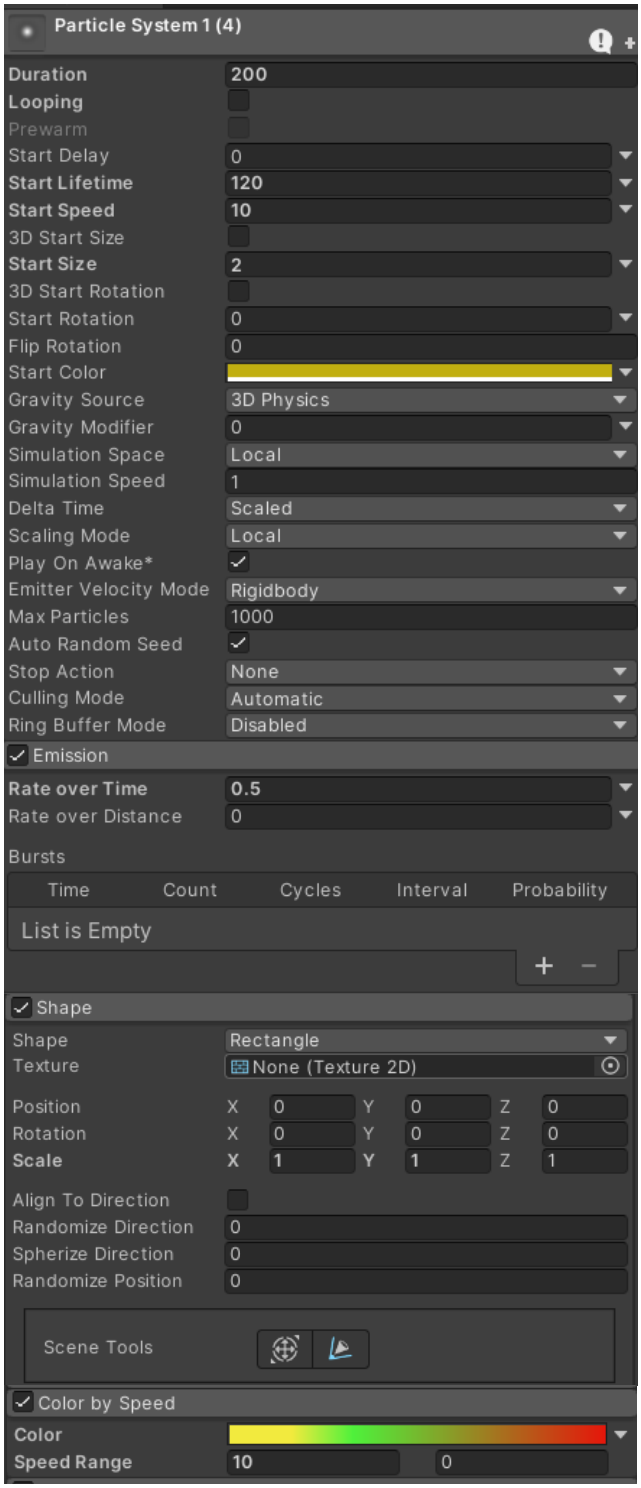


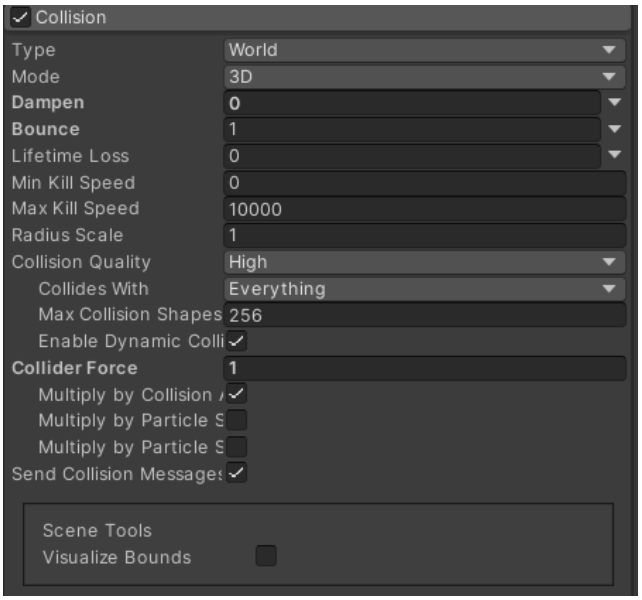
Satellite Game Object – Main Body





GSI Particle System





Appendix D – GitHub Repository & Video

A public repository has been created on GitHub with all necessary scripts, prefabs, materials, textures, etc. This can be accesed at https://github.com/ANS2806/VLEO-exhibit.

GitHub also has some useful flowcharts that help in understanding the project flow. The first flowchart titled “Overview Flowchart” is useful to get a zoomed out view of how the various inputs, applications and outputs are connected to each other. Another flowchart titled “Code Flowchart” provides a more detailed interaction of different game objects, scripts, methods, etc. and is useful if trying to make changes within the scripts. Finally, the “Future Scope Flowchart” showcases all the spaces where advanced features or sophistication can be added. These flowcharts can also be accessed here in OneDrive.

A short video showcasing the different features of the satellite is available at https://youtu.be/ySsJiKzwlRg.