Virtual Physics: Modelling Anisotropic Particles in Fluids using Unity Game Engine

Cristina D. Gonzalez, Greg Voth Department of Physics, Wesleyan University

Introduction

One of the most challenging steps in the 3D reconstruction of complex particle motion in turbulent flows is **measuring particle orientation from multiple images**. Standard methods, which numerically project the object onto multiple cameras, fall short when tracking the more complex movements of particles affected by specific lighting conventions.

However, Unity, a popular video game engine, is highly optimized to project 3D objects onto images, allowing us to simulate the translation and rotation of helical ribbon particles through 500 centistokes fluid (silicon oil).

To best simulate the particle's movement, we first set up **virtual versions of the objects** involved in the experiment, including the helical ribbon particle, in which light is rendered using Unity's **High Definition Render Pipeline** and movement is simulated using Unity's RigidBody engine.

Figure I. Real-life CAM72 image of helical ribbon

Original Set Up

Firstly, we implemented 3D models of each object and camera involved, including their exact scale and position. Unlike conventional physics and mathematics, Unity uses a left-handed coordinate system and the Euler "ZXY" convention, where the positive rotation moves clockwise from the axis of rotation.

This system caused challenges throughout the development of this project, which we tackled using MATLAB scripts translating from real-life information to the Unity system.

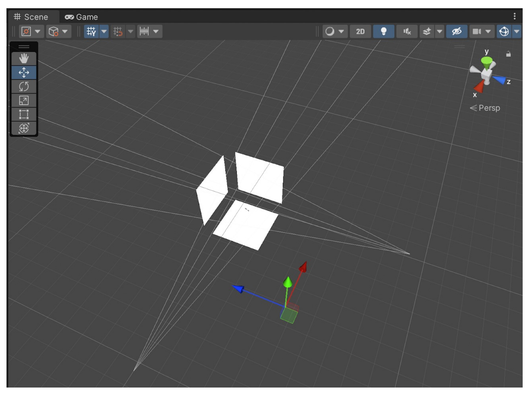


Figure II. Experimental Scene Set Up

Camera Calibration

To properly recreate the real-life camera images, we had to calibrate the camera's extrinsic parameters (converting the coordinate system onto the camera coordinate system) and its intrinsic parameters (converting camera coordinates onto pixel projections).
Unity does not have a built-in internal camera calibration system, so we computed the calibration outside of Unity and implemented it for the virtual cameras using parameters in the **Unity physical camera component.**

Intrinsic parameters: The field of view in Unity's cameras, or what the camera sees at a point in time (which follows the pinhole model) is inversely proportional to the focus length.

Therefore, we used Python's OpenCV library and its camera calibration systems to compute the focal length (f_x , f_y) and other necessary parameters used in camera calibration:

$$camera\ matrix = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

We ran images of a checkerboard from each camera on a script using the OpenCV library, which gave us each camera's focal length and image distortion based on the angles between the checkerboard corners.

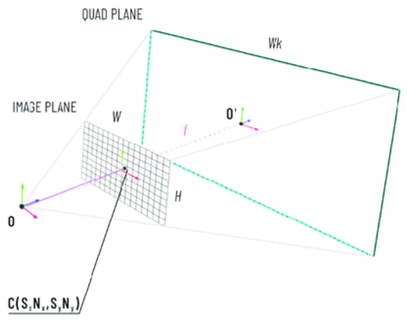
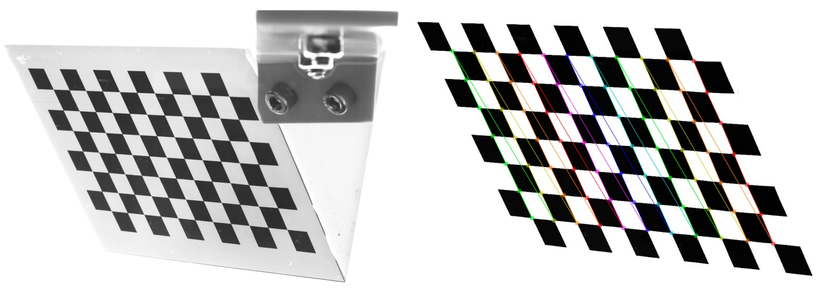


Figure III. Unity Camera Pinhole model

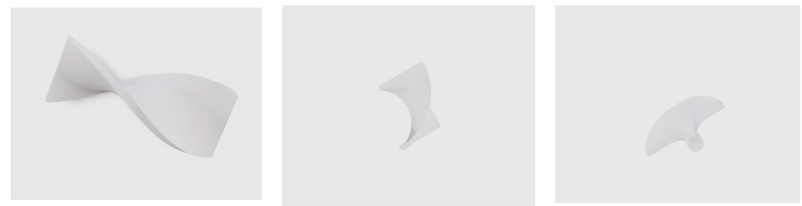


Figures IV,V. Original image of the checkerboard calibration rig on the tub. The holder was edited out of the picture, and the image's exposure was increased. Later, the calibration rig would be drawn and computed onto the edited image.

Extrinsic Parameters: For more accuracy, we used lab-calculated values (including position and camera rotation matrices) to finalize the setup of the cameras.

Lighting

Unity lights its virtual world through a system of pre-computed 'global illumination' and 'real-time' lights, which simulate the complex behavior of light as it bounces and interacts with the world. Unity's High Definition Render Pipeline gave us greater control to apply complex lighting conditions to the virtual world.
When considering the visual aspects of the particle simulation, there are two main concerns: how light scatters off objects and what light sources there are.



Figures VI,VII,VIII. Three camera images of rotated particle

First, we set up large lights which simulated the big LED panels present while filming the real-life particles dropping, as well as complementary spotlights.

To control the light scattering on the particle, we modified the "material" of the helix to match its real-life counterpart best, including setting up its metallic value and smoothness.

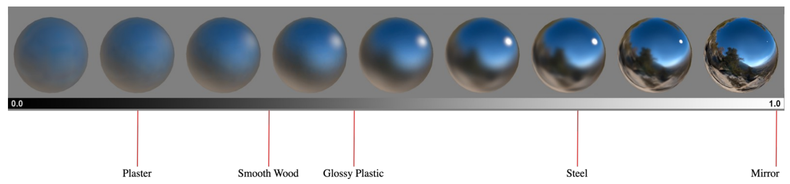


Figure XV. Unity Smoothness Manual

However, even if Unity's lighting system is quite powerful, it is not equipped with the actual material of the real-life particle (translucency), as it cannot compute varying degrees of internal light scattering.

This issue affects the particle visuals and how the light bounces from it within the scene. Unity also has trouble recreating the shadows of complex particles, oversimplifying the shadow the object makes within the panels in the background.

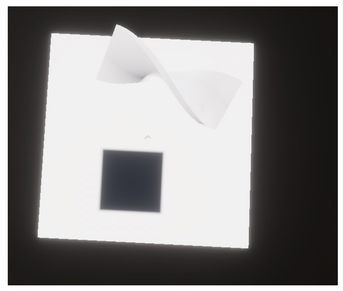


Figure X. Error computing shadows!

Movement

Finally, we simulated the position and rotation of the particle by using Unity's built-in 3D physics engine by measuring the particle's starting and ending position. The particle was translated using Unity's Transform.position() function.

For the rotation of the particle, we used the Quaternion.Slerp() function, which spherically interpolates between the two angles. Both functions ran concurrently and are updated every frame to create the final result.

Scan me to watch the particle move!

Conclusion

By exploring Unity's Physics engine and optic systems, we could better understand the inner mechanisms of game engines and how they relate to real-life physics, as well as the strengths and limitations of Unity as a tool for the simulation of fluid dynamics, turbulence, and other scientific models.

Unity provides an excellent system for modeling real-life objects and offers a robust lighting system that can compute extensive computations faster. However, Unity is not always precise with its calculations and may not fully represent the exact physics to simplify its implementation process. In the future, it would be beneficial to use Unity's built-in features as a stepping-stone and build a custom version of their Physics engine to account for more complex movement and object shapes.

Acknowledgements

I would like to thank Professor Voth for his guidance and mentorship in this program as well as the QAC Summer Apprenticeship and the RIS for making this opportunity possible through funding and support.

[1] Condino, Sara, et al. "Hybrid Spine Simulator Prototype for X-Ray Free Pedicle Screws Fixation Training." Applied Sciences, vol. 11, no. 3, 2021, p. 1038, https://doi.org/10.3390/app11031038.