**The purpose of this document is to act as a workspace for text that I eventually want to be a part of the paper. I’ve been switching between different versions of MSWord on Mac and PC.**

# Intro

# System

## Software Options

/\*Continuing from Unreal to talking about LAMMPS\*/

Additionally, the engine supports scripting in C++, making it an attractive option for scientists looking to reuse existing code and quickly incorporate VR-based data visualization into their research endeavors [45]. Furthermore, Unreal Engine 4 offers a unique form of visual scripting called Blueprints. These allow users to program for the engine using a graph system where functions and variables are represented with nodes and relationships between them are represented with edges; theoretically, entire games can be created without using C++ and only using the Blueprint system. For research purposes, this makes high-level/interface features – such as controls, menu options, and HUDs – incredibly simple to develop, accelerating the data acquisition process. However, we ultimately wish to gravitate our platform towards real-time simulation, meaning full control over the engine via C++ seems to be a necessity. We still wish to minimize the amount of “new” coding necessary to get a researcher up and running with the system. As a solution, we chose to integrate the existing library, LAMMPS, into our Unreal GEARS visualization system.

# Use Cases

# Discussion

# Abstract

## How to use the VFC Shader

The single simulation data prefab is actually split into pieces. Each of these pieces has a material associated with it. You need to make sure all these pieces’ materials use the shader we provided. Additionally, each of these pieces needs to have a box collider associated with it to detect the plane when it intersects. You can do this by highlighting each piece in the Hierarchy window, then attaching a box collider component to them. Then make sure that the collider’s “Is Trigger” box is checked. Attach the VFCDetection trigger script we provide. This handles the collision detection for the pieces of the simulation data. When the plane intersects, this script will update the shader’s inputs with the current values for the plane normal and viewing distance.

That’s just how we set up the simulation data shader. Now we need to setup the actual viewing plane that slices through the data. To do this, we need to make a canvas for the in-game camera. Create a campus from the GameObjects menu, and then set it to “Screen Overlay – Camera” mode. This allows the canvas to follow the head movement of the user. We then need to place a plane on the canvas that will trigger the box colliders, and be the basis for our normal vector that we supply to the shader. Create a plane, attaching it to the Canvas, adding a rigid body component to it. If one feels it beneficial to their data acquisition, they can increase/decrease the dimensions of the plane to encompass all or some of the user’s vision. Attach a rigid body component to the canvas. Now you should be good to go.

## Using our provided LAMMPS animation system in UnrealGEARS

To demonstrate the utility of LAMMPS as not only a simulator, but also an animation tool, we use it to rerun and render simulations based off LAMMPS dump files. This does, however, involve proper setup of the DUMP files and input script.

Theoretically, one could also create their own XYZ file interpreter in the Unreal Engine, and animate simulations just based on that. But using LAMMPS allowed us to reduce the amount of coding needed to go from obtaining simulation data to viewing it run in Unreal.