2022

A Reynolds Number Based Sampling Technique for 3-D Vector Fields in Computational Fluid Dynamic Environments

Trent James Schweitzer

A REYNOLDS NUMBER BASED SAMPLING TECHNIQUE FOR 3-D VECTOR FIELDS IN COMPUTATIONAL FLUID DYNAMIC ENVIRONMENTS

By:
Trent SCHWEITZER

Bachelor of Science - Health & Human Performance, University of Montana, Missoula, MT, 2015
Associates of Applied Science - Photographic Communications, Northwest College, Powell, WY, 2009

Master of Science in Computer Science

University of Montana

Missoula, MT

May, 2022

Approved by:

Scott Whittenburg, Dean of The Graduate School

Graduate School

Jesse Johnson, Chair

Computer Science

Yolanda Reimer

Computer Science

Russell Parsons

Forest Service

Schweitzer, Trent, M.S., May 2022 Computer Science

A Reynolds Number Based Sampling Technique for 3-D Vector Fields in Computational Fluid Dynamic Environments

Chairperson: Jesse Johnson

Effective visualization of unsteady, time-dependent vector fields in a virtual environment is not a trivial task. This is due to the fact that most visualization techniques require the user to have a prior understanding of how the vector field will behave to set the parameters used to create the visualization. In this thesis we will take air flow data from a computational fluid dynamic simulations to calculate the amount of turbulence (represented as Reynolds numbers) to identify regions of interest. We then calculate wind pathlines that will intersect with these points sampled from these regions. We address the issue of optimizing the appropriate number of pathlines relative to the size and resolution of the simulation. We are then able to implement the ability to interact with the simulations using a modern video game engine with virtual reality capabilities. By comparing the results with results that do not involve the turbulence based sampling methods, we conclude that our method provides more detail where detail is demanded.

Acknowledgements

I am deeply grateful to Jesse Johnson for their assistance at every stage of this research project. I am also extremely thankful for the rest of our research team, Anthony, Fred, Brad, and Esther that provided vital feedback and support throughout this project.

Contents

List of Figures

List of Abbreviations

CFD	**C**omputational **F**luid **D**ynamics
VR	**V**irtual **R**eality
FDS	**F**ire **D**ynamic **S**imulator
LIC	**L**ine **I**ntegral **C**onvolution
USFA	**U**nited **S**tates **F**ire **A**dministration
ODE	**O**rdinary **D**ifferential **E**quation
HMD	**H**ead **M**ounted **D**isplay
HDF5	**H**ierarchical **D**ata **F**ormat version **5**
JSON	**J**ava**S**cript **O**bject **N**ation

List of Symbols

Re	Reynolds Number	
p	fluid density	$\mathrm{kg\,m^{-1}}$
u	flow speed	$\mathrm{m\,s^{-1}}$
L	characteristic linear dimension	m
μ	dynamic viscosity of fluid	kg/(ms)

Dedicated to my wife without her I would forget to eat.

Chapter 1

Introduction

1.1 Motivation

Simulation modeling has become an invaluable tool for emulating smoke and heat release from fires. Its use for training urban and wildland firefighters has the capability to be immeasurably useful as it eliminates the danger of training in hazardous conditions. For example, in 2020 the United States Fire Administration (U.S.F.A.) reported that there were a total of 66 firefighter fatalities (this does not include SARS CoV-2 related deaths). Twelve of those fatalities were at the scene of wildland or outdoor fires [USF22].

When looking at wildland fire incidents with multiple fatalities related to fire behavior, wind is noted to be a major contributing factor to the fatalities. Some notable examples of this are the Yarnell Hill Fire which had 19 fatalities [Ser13], the El Dorado Incident which had one fatality [Ser20], the Twisp River Fire which had three fatalities [Ser16], and the Mann Gulch Fire which had 13 fatalities [Ser49]. While it is impossible to definitively state that additional training could have prevented these fatalities, it has been shown by B. Wiederhold et. all that stress responses in virtual reality had a high level of correlation to real life stimulus [Wie+03].

A training environment in virtual reality will allow anyone ranging from brand new firefighters to fire bosses to experience dynamic fire environments. An advantage with utilizing virtual environments is that we are able to create an unlimited number of unique situations to train firefighters. The more unique environments and situations we are able to expose firefighters to, the better prepared they will be to deal with a fire in an emergency situation. Additionally, virtual reality (VR) is rapidly improving to the point where some VR head mounted displays (HMD) can cost under $300 and only need to be connected to a phone application to run [Fac22].

Steven G Wheeler, Hendrik Engelbrecht, and Simon Hoermann reviewed six articles pertaining to the use of firefighter training in virtual reality [WEH21]. In each experiment they showed that VR training out performed the control groups and performed at least as well as other traditional training methodologies.

Current visualization software, such as Smokeview or Paraview, allow the users to change their vantage point and manually filter out what information may be important. A variety of types of data are needed to build an accurate simulation such as physical obstructions, topography, fuel types, ignition points, wind speed, and wind direction.Topography and fuel type information can all be directly pulled from databases like LANDFIRE. Wind speed and direction data can be obtained from NOAA and visualized directly or put into a simulation model to estimate future

wind data [FF17]. While local wind behavior during a fire is complex and dynamic, we are able to model it using the well validated reaction chain and computational fluid dynamic simulator from the National Institute of Standards and Technology (NIST) called Fire Dynamic Simulator (FDS) [McG+21]. FDS allows us to generate discrete estimates on a structured grid of wind vectors, temperature, and air density. Improvements in computational hardware and available algorithms has made visualizing these simulations in a digestible manner a tangible goal.

1.2 Graphing Techniques

Currently there are many unique techniques used for visualizing vector fields in 2-D or 3-D spaces. These can be sub-categorized as textured-based techniques (line integral convolution), glyph based (arrow graphs), or particle tracing (streamlines). All of these approaches have unique benefits but come with a similar set of drawbacks that our methodology looks to resolve.

Line integral convolution (LIC) simulates surface oil patterns in wind tunnel experiments. To do this, a texture of white noise is applied to the domain of an area. This can be a flat plane at any orientation, or more commonly, it is a texture mapped onto a 3-D object. Next a one dimensional convolution with a kernel filter, that is in the direction of the vector field is applied to the white noise, then finally the kernel output is normalized [CL93]. The resulting intensity of LIC pixels is recorded and this provides a visualization of strongly correlated streamlines. A large drawback when it comes to LIC is that line brightness is not indicative of velocity due to the local normalization that occurs. Additionally, the texture will occlude any part of the object behind it; as well as, the small stream lines do not indicate directionality, i.e., a line going left to right looks identical to a line going right to left [Rez+99].

Hedgehog, or arrow plots, are visualized by inserting glyphs as each cell of data in a vector field. Glyphs can be represented by a variety of 3-D objects, but arrows are primarily used. The orientation of the glyphs can be used to indicate directionality, while the objects scale and color can represent other scalar values. These plots can also be visualized in 2-D or 3-D. The main advantage to this type of graph is the ease of implementation and ability to be understood quickly. The drawback with this technique is when represented in a 3-D area, the visual becomes quite busy if each point in a vector field is represented. Additionally, when the data being represented is time dependent, then the previous time-step glyphs are just replaced with the new glyph. This forces the user to mentally compare the changes and make assumptions based on that.

Particle tracing tends to be a larger subcategory as the techniques used change drastically based on several factors. If the vector field has steady state flow, meaning it does not change in time, we refer to these as streamlines. Pathline is the term used if the vector field is time dependent. We also have streaklines, where a line is created from all particles that pass through a given point. Placement of these points can be predetermined, random, or placed to form a designated shape like a ribbon or a cylinder. To calculate the flow of a particle through a volume, the ordinary differential equation solvers (ODE's) are used to move the particles through velocity fields. ODE solvers with dynamic time stepping, like Runge-Kutta 4(5), ensures that we have a more accurate model then one with a discrete time step like Euler's [TGE97]. These visualizations can be textured to indicate velocity at a set point along the line. They have the same demerits, as previously discussed with LIC; there is no clear

start or end to each line and resolving areas of interest is difficult without previous knowledge of the vector field.

All of these visualization techniques generally have a large drawback, all information in the field is visualized at once, forcing the user to mentally compare the change between time steps, all while having other data being occluded by data closer to the user's vantage point. Solutions often involve the user having to preselect where they assume that the most important information will be, which is difficult to do without prior visualization.

Our goal is to develop an optimized way to visualize pathlines in vector fields focused on improving firefighter training. For fire simulations the areas we will be interested in are areas with turbulent air flow that can cause unpredictable fire behavior. Using the reynolds number allows us to have a unitless value to indicate air turbulence, with this we can calculate pathlines that cross these areas of interest.

Chapter 2

Methods

2.1 Reynolds Number

Our first goal is to identify a set of points of interest. Because turbulent flow is where we need to study, we need to calculate the reynolds number (RE) of each voxel in every time step of the simulation. We take the product of p, u and L where p is the density of the fluid, u is the flow speed of the fluid, and L is the characteristic linear dimension. Then using the dynamic viscosity of fluid represented as μ as the divisor to get the RE for that voxel in that time-step(Eqn. 2.1). It should be noted that for consistency in our simulations L is calculated from the average length of the sides of each voxel ($\sqrt[3]{length * width * height}$). Due to the scale being a constant in each calculation, changing this value can affect the scale of values obtained, but it does not alter the distribution of the values.

$$RE = \frac{puL}{\mu} \tag{2.1}$$

Now we have the reynolds number of every voxel in each time-step RE_t where t is an index number referencing a time value. Next, we find the mean value for each voxel 2.2 then threshold the data into two categories; $RE >= 150$ as turbulent flow and $0 < RE <= 40$ as laminar flow. We chose to take the mean over the median or mode as we don't expect our data to have any outliers to the distribution; and with reynolds numbers being 32-bit floating point values any numerical repetition is inconsequential. We exclude RE values of zero because they are indicative of a voxel that is within an obstruction (e.g. underground, inside of a tree). Next we RE threshold our 3-D points in space to locate points of interest.

$$\langle RE \rangle_{mean} = \frac{1}{T} \sum_{t=1}^{T} RE_t = \frac{1}{T} \left(RE_1 + \cdots + RE_T \right) \tag{2.2}$$

Because the number of voxels with $\langle RE \rangle$ greater than the threshold is large we run a clustering algorithm on voxels with $\langle RE \rangle$ above the threshold, in this case we use k-means due to its ease of implementation and speed when clustering large sets of points [Bar22]. This groups all points into k groups while minimizing the distance between a point and that group's centroid. We calculate the value of k by taking the $\frac{1}{2}\sqrt[3]{I * J * K}$ where I, J and K are the number of voxels in the x, y and z axis respectively. We then are able to take the centroid from each cluster and refer to these as our points of interest reducing the points of interest by approximately 7000.

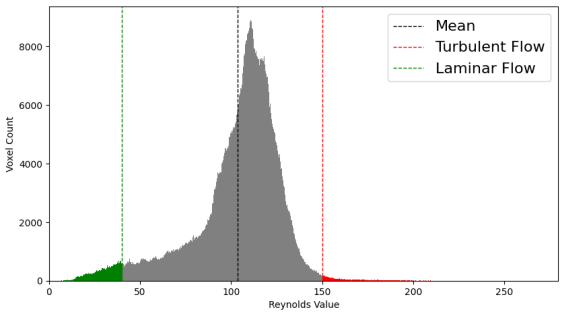

FIGURE 2.1: Mean Reynolds Values Histogram

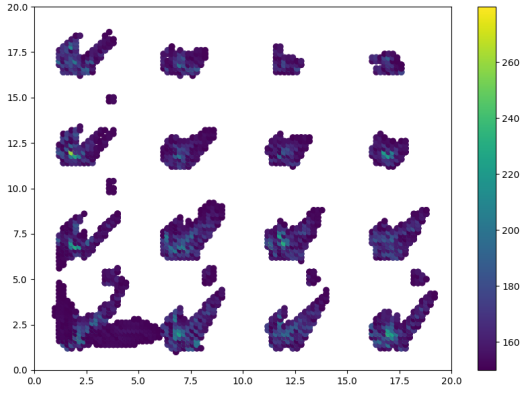

FIGURE 2.2: A top down view of a plot showing areas of turbulent air flow containing 7705 points

2.2 Pathline Calculations

Now that we have calculated the points of interest we can use an ordinary differential equation solver to calculate the pathlines. We will use a fifth-order Runge-Kutta method for adaptive time steps and to minimize errors in calculations [TGE97]. We are able to use forward and reverse integration to calculate pathlines for every time step that passes through these points of interest. After integrating in both directions we then combine the posting and time data for each pathline while also adding in the reynolds number for each point as well. We chose to store all of this data in formatting so it can be efficiently loaded in parallel into our visualization pipeline [].

2.3 Unity Visualization

Once we read in pathline and scene data into Unity, we determine the largest ($REmax$) and smallest ($REmin$) value within the data set. Now having the minimum and maximum values we are able to calculate a color value for each segment of the pathlines. To do so we have to calculate the interpolant value within the range [$REmin$, $REmax$] (Eqn. 2.3). This provides us with a value between [0,1] allowing us to map this value to a color on a perceptually uniform color map. We selected Vega's magma color map with eight colors [Veg22]. For each time step, we load in the respective data for all k pathlines. To resolve the issue of being able to see orientation but not true directionality of a pathline we segment each pathline into n-1 line segments (e.g. $P_0 \rightarrow P_1$,$P_1 \rightarrow P_2$, ... ,$P_{n-2} \rightarrow P_{n-1}$). We then adjust the starting width of each line segment to be the average length of a voxel and the ending width zero. Figure 2.3 illustrates the benefits of this visual change. Using cones resolves the issues with directionality of pathlines as previously discussed. We then build a color gradient texture and apply it to the texture from point to point with the precalculated color map. We are able to compile and build our Unity system into an executable file that allows for the end user to run on any compatible system. We have the user interface (UI) designed to allow for viewing the simulation using a head mounted display, a virtual reality headset, and controller or a computer monitor with a keyboard and mouse for movement. Users just need to type in the directory of the files output discussed previously and the data is cached for quick loading into the simulation. Users are then able to move around the environment in either virtual reality or just by using the mouse and keyboard.

$$f(RE_{min}, RE_{max}, RE_{value}) = \frac{RE_{value} - RE_{min}}{RE_{max} - RE_{min}} \tag{2.3}$$

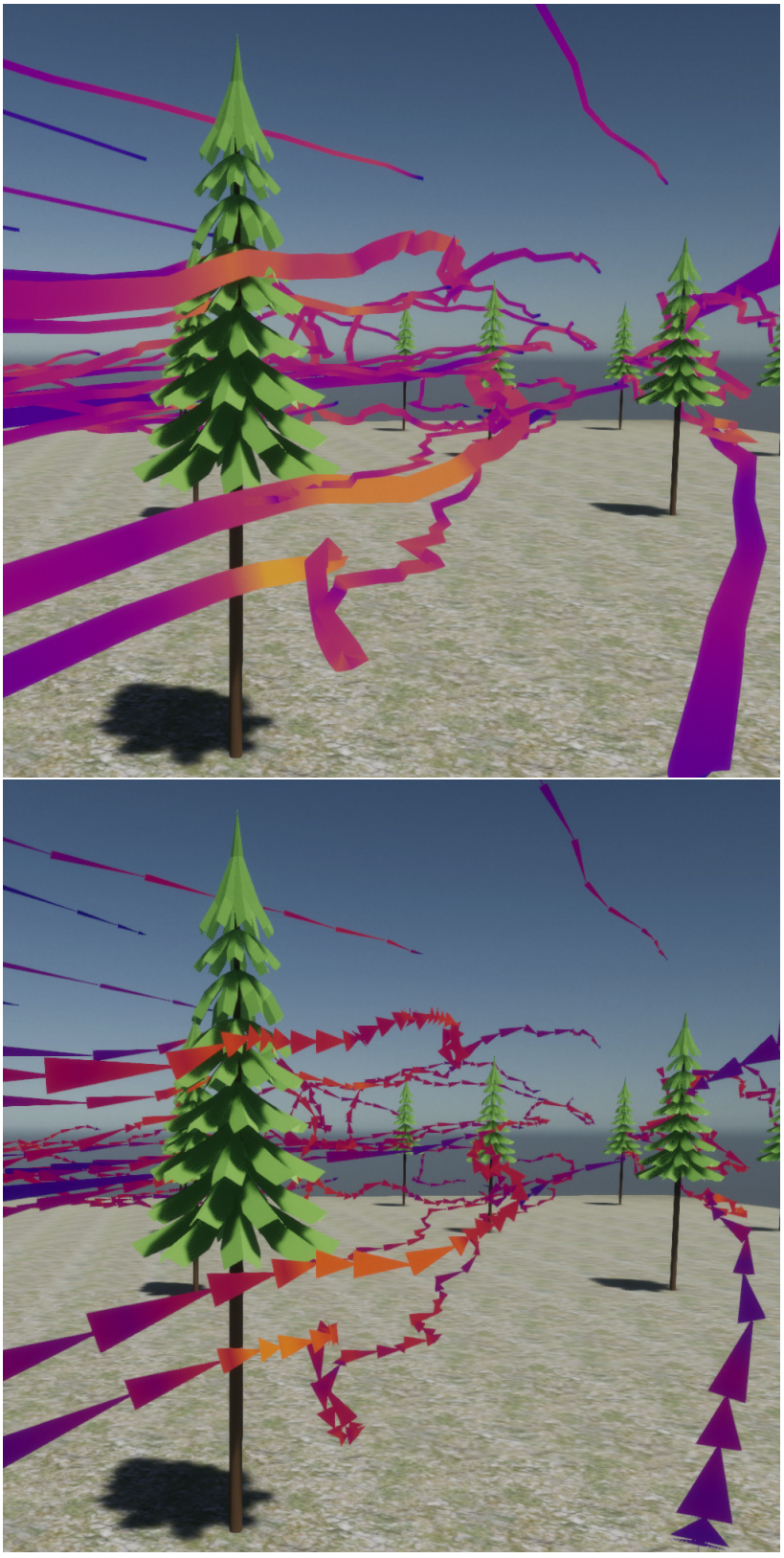

FIGURE 2.3: **Top:** Pathlines visualized as a single line **Bottom:** Pathlines broken in to line segments

Chapter 3

Results

3.1 Setup

In this paper, we described a pipeline implemented using `python` then `C#` and Unity's real time development platform; as the visualization engine. To simulate fire behavior, we ran a Level set 4 FDS simulation, set to output plot3D files every 0.5 seconds on a 20m x 20m x 20m mesh, containing voxels 0.2m x 0.2m x 0.2m in size, with open boundary conditions. The simulation was 100 seconds in length with a ground fire igniting ten seconds into the simulation, a wind of 5.6 m/s originating from the south west (215°) and trees evenly distributed on the mesh. Figure 3.1 shows a top down view of our visualization rendered in SmokeView, while Figure 3.2 is the same visualization rendered in unity.

The sample FDS input file and code used in this paper can be accessed at `https://github.com/tjschweitzer/ReynoldsNumberSampling`.

3.2 Analyses

The CFD simulation was run using FDS 7.6.0, the total run time for this simulation took 252 (4 Hours 12 Min) minutes to complete running 4 cores of an i7 4790k in parallel. The calculations discussed in this paper were able to complete in three minutes, with no parallelization. Additionally, the size of the CFD output files was 5.0 Gigabytes, while the saved data needed for full virtual reality using our methodology is 28.9 Megabytes, a 99.4% reduction in file size. When visualized inside of unity using a HMD (HTC Vive and HTC Vive Pro was used during testing) we were able to average 15-30 frames per second with a resolution of 1440 x 1660 pixels per eye 3.3, with the lowest frame rate during the transition between timesteps. This was achieved using an Intel i7 4790k CPU with a RTX 2070-Super GPU.

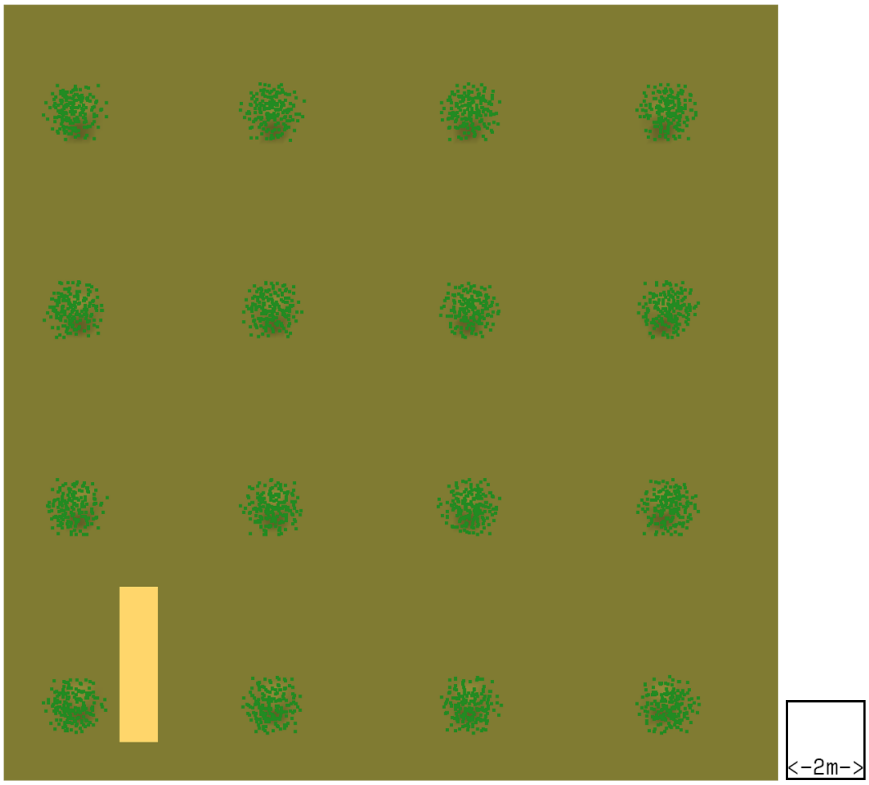

FIGURE 3.1: A top-down view of the simulation in SmokeView with a two meter by two meter square on the right for scale

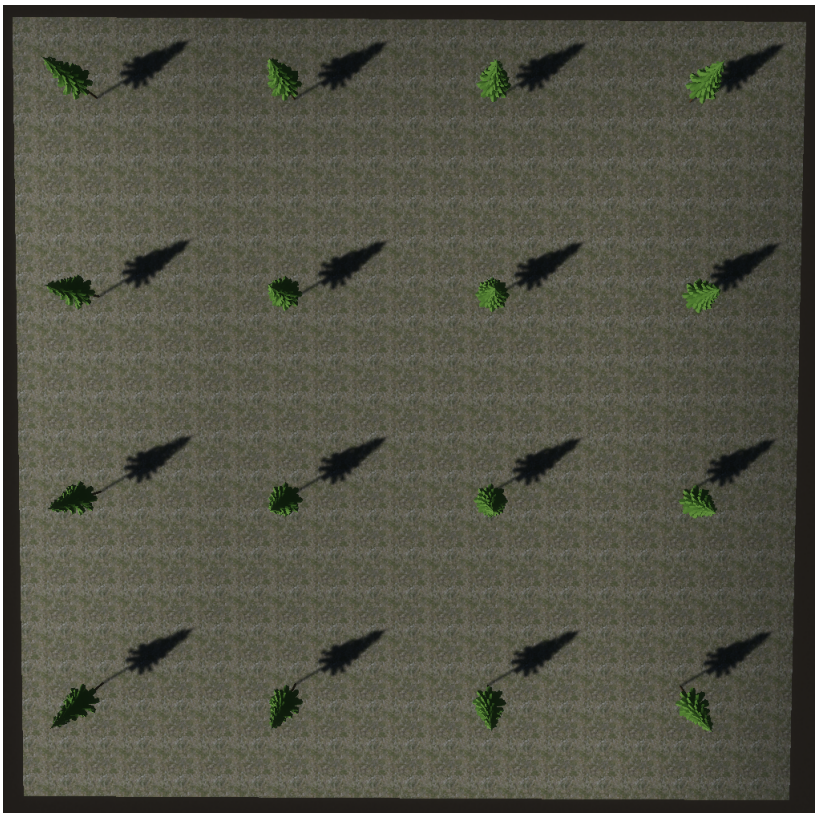

FIGURE 3.2: A top-down view of the simulation in Unity

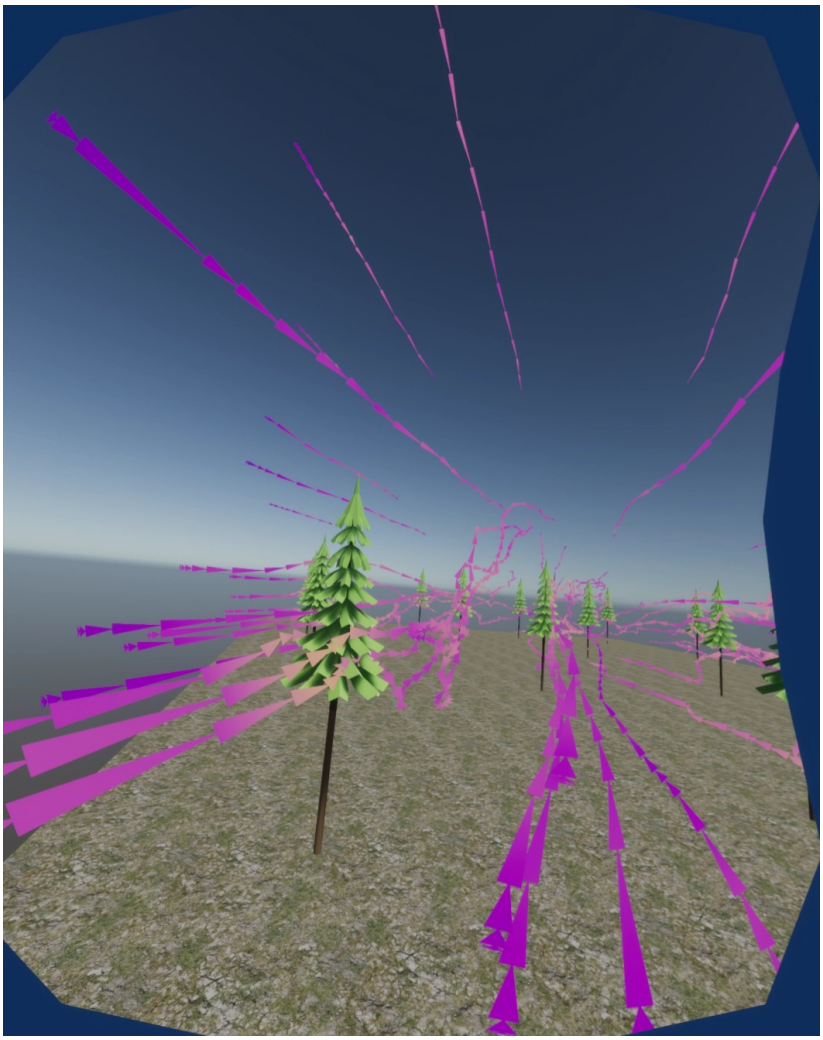

FIGURE 3.3: A view of the video output to the left eye of a HMD from Unity

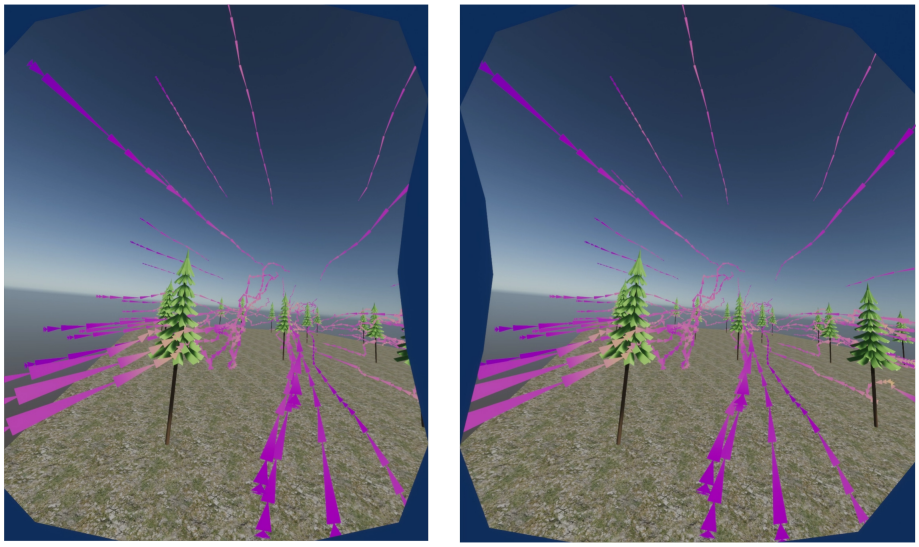

FIGURE 3.4: A view of the video output to the eyes of a HMD from Unity

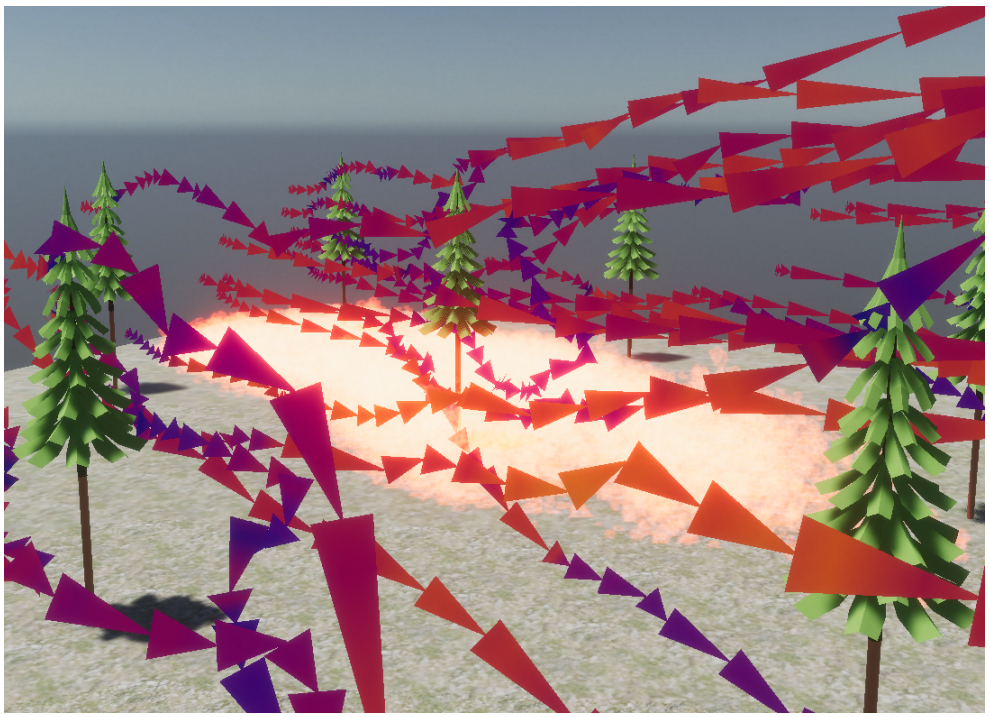

FIGURE 3.5: Fire and wind vectors visualized with Unity, color mapped based on the heat release rate and air velocity respectively

Chapter 4

Conclusion and Future Work

Updating the tools available for fire fighter training is a current objective for the U.S. Forestry Service. New approaches have the capability to save lives if used effectively. Our goal was to design a generalized proof of concept that would be able to be built on to improve training and safety for firefighters. The use of Unity's development platform allows us to compile and provide a self-contained system to improve training for firefighters.

A necessary focus for this work was the visualization of 3-D wind fields that transport oxygen to and heat away from fires. Improving the fidelity of the visualization we are able to allow firefighters to experience simulations without the risks of a real fire. This allows each user to have a larger mental set of fire environments they have experienced before being put into a dangerous situation. This will allow them to make more accurate assessments in real life situations and be safer overall.

We were able to substantially lower the total amount of memory needed to store pertinent data, with only a trivial increase to the amount of processing time, for the current pipeline of running and visualizing a simulation.

The Unity stage of our pipeline is currently designed for use with HTC, Steam's HMD, or no headset. A separate version of the system could be compatible with other brands of HMD with minimal changes to the configuration; further research into this would be needed.

In the future, further research into improving or implementing new thresholding and clustering techniques may be able to further optimize our pipeline. As for future capabilities, unity implementation of additional libraries would allow for these projects to run on the Oculus platform, which would lower the cost of equipment needed for each training facility. Additionally, further research into utilizing Unity's WebVR would allow for the system to be hosted online and accessed remotely; again lowering the total cost of the equipment needed for each training facility.

Within the VR simulation, allowing the users to select what scalar value is being shown along the pathlines would also help in the training process. This would require the simulation to be run multiple times to get different values like temperature, heat release rate, smoke density, etc. Visualizing the temperature along the paths for example would allow for firefighters to see how the temperature effects the flow over ignited areas.

Appendix A

Software Documentation

The following is a list of the repositories of code used to visualize the simulations, including descriptions of each script contained within. Additionally, we list and describe each function/method contained within the scripts as well as the file structure for each data type used.

- **FDSPreUnityTool**

  Converts the plot3D output from Fire Dynamic Simulator (FDS) into a more optimized data format to be visualized inside of unit.

  - **fds2ComplexGeom.py**

    Converts FDS input file into a JSON with tree location data and complex terrain information

  - **fdsOutput2Unity.py**

    Converts plot3D data from FDS into binary data to be loaded into unity

  - **fdsPathLines.py**

    Takes plot3D data and calculates the amount of turbulence (represented as Reynolds numbers) to identify regions of interest. Then calculates pathlines that will intersect with points sampled from these regions

  - **main.py**

    Creates the proper file structure and ensures all functions save data into the proper location

- **FDS2UnityVR**

  Visualizes the data from **FDSPreUnityTool** in an immersive VR environment using Unity's Real-Time Development Platform

  - **ConfigData.cs**

    Saves all information when transitioning from one scene to another

  - **hrrLoader.cs**

    Loads in a color mapped and animated fire particle engine in each voxel per timestep

  - **MainMenu.cs**

    Verifies menu check boxes are selected and verifies fire and directory structure for input data

  - **PauseMenu.cs**

    Checks pause button input and tracks time in pause menu to allow for accurate timesteps after pausing

  - **smokeLoader.cs**

    Loads in a color mapped and animated smoke particle engine in each voxel per timestep

  - **smvReader.cs**

    Loads in and caches heat release rate and smoke density data while tracking the current timestep

  - **tduController.cs**

    Uses calulations of thermal dosage from Parsons, Russell A., Butler, Bret. W., and Mell, William "Ruddy" to visualize burn injury potential [PBM14]

  - **TerrainBuilder.cs**

    Loads a JSON of terrain data from fds2ComplexGeom.py A.1.1 and maps to a 3-D mesh with texturing

  - **TreeLoader.cs**

    Loads a JSON of tree data from fds2ComplexGeom.py A.1.1 and maintains scale regardless of the size of the 3-D tree object

  - **WindStreams.cs**

    Loads in and caches pathline data from fdsPathLines.py A.1.3 to be visualized each timestep

A.1 FDSPreUnityTool

A.1.1 fds2ComplexGeom.py

Converts FDS input file into a JSON with tree location data and complex terrain information

- **Class instantiation**
  - Parameters
    - **fds_input_location:** Complete path to FDS input file (.fds)
    - **tree_id:** Name of the tree label
    - **non_terrain_obsts:** List of any obstacles that are not classified as terrain
- **read_in_fds_obst**

  Reads in FDS input file and generates a dictionary of dictionaries containing the elevation data for the terrain
- **read_in_tree_locations**

  Reads in all the locations of trees in the FDS input file and parses other tree information
- **read_in_fds_mesh**

  Reads in FDS input file and parses the MESH information
- **complex_geom**

  Reads in topographical information and formats it for a 3-D mesh
- **save_to_json**

  Saves all parsed data into a JSON file

  JSON File Structure
  - **meshData**
    - **X_MIN:** x value of the origin point of the mesh
    - **Y_MIN:** y value of the origin point of the mesh
    - **Z_MIN:** z value of the origin point of the mesh
    - **X_MAX:** x value of the opposite corner
    - **Y_MAX:** y value of the opposite corner
    - **Z_MAX:** z value of the opposite corner
    - **I:** number of cells in the x direction
    - **J:** number of cells in the y direction
    - **K:** number of cells in the z direction
  - **verts**

    List of all vertices of the topographical map

  - **faces**

    List of indices of vertices to form triangle faces (one indexed values)

  - **treeList**

    * **x:** x position
    * **y:** y position
    * **height:** Ground height at position x,y
    * **crownBaseHeight:** Height from ground level where tree crown begins
    * **crownRadius:** Radius of tree crown
    * **crownHeight:** Height of tree crown

- **write_hdf5**

  Saves all parsed data into a HDF5 file

  Note: Reading in the complex geometry as a HDF5 file has not been implemented in to the current Unity project

  HDF5 File Structure

  - **meshData** Array of nine, 32-bit floats

    [X_MIN,Y_MIN, Z_MIN, X_MAX, Y_MAX, Z_MAX, I, J, K]

    * **X_MIN:** x value of the origin point of the mesh
    * **Y_MIN:** y value of the origin point of the mesh
    * **Z_MIN:** z value of the origin point of the mesh
    * **X_MAX:** x value of the opposite corner
    * **Y_MAX:** y value of the opposite corner
    * **Z_MAX:** z value of the opposite corner
    * **I:** number of cells in the x direction
    * **J:** number of cells in the y direction
    * **K:** number of cells in the z direction

  - **verts**

    List of all vertices of the topographical map

  - **faces**

    List of indices of vertices to form triangle faces (one indexed values)

  - **treeList** 2-D array, Size: N (number of trees) by six

    [x, y, height, crownBaseHeight, crownRadius, crownHeight]

    * **x:** x position
    * **y:** y position
    * **height:** Ground height at position x,y

    * **crownBaseHeight:** Height from ground level where tree crown begins
    * **crownRadius:** Radius of tree crown
    * **crownHeight:** Height of tree crown

A.1.2 fdsOutput2Unity.py

Converts plot3D data from FDS into HDF5 data to be loaded into Unity

- **Class instantiation**
  - Parameters
    * **fds_output_directory:** Full path to where FDS data was output
    * **fds_input_location:** Full path to FDS input file
    * **save_location:** Full path to where data should be saved once processed
    * **save_type:** Allows for selection for output file type .
- **read_in_fds**

  Reads in FDS input file to generate list of data being saved as plot3D
- **get_file_timestep**

  Extracts the timestep from the filename
- **group_files_by_time**

  Groups filenames by timestamp to allow for multi-mesh simulations
- **find_max_values_parallel**

  Parallel function to find the maximum values for each time of plot3D data
- **get_values**

  Calculates minimum and maximum values for all types of plot3D data
- **runParallel**

  Reads in and saves all plot3D data in parallel.
- **get_mesh_number**

  Pulls mesh number from filename
- **q_file_to_dict**

  Reads in plot3D data and saves it to a dictionary to allow all threads to access the data when running in parallel
- **write_to_json**

  Saves all data to JSON format

  JSON File Structure
  - **fire:** 2-D array, size: N number of voxels by four

[x, y, z, q] where x, y, and z represent the position while q represents the heat release rate

- **smoke:** 2-D array, size: N number of voxels by four

  [x, y, z, q] where x, y, and z represent the position while q represents the smoke density

- **configData:**
  - **min:** 1-D Array, Minimum value for [fire, smoke]
  - **max:** 1-D Array, Maximum value for [fire, smoke]

- **write2bin**

  Saves all data in a custom binary format

  Binary File Structure

  - **Header**

    Five 32-bit integers corresponding to the length of each of the five scalar values from the plot3D output

  - **2-D array with [x,y,z,q] as quantities, where x, y, and z represent the position while q represents the respective scalar value**

    Note: this is a space matrix version of the output from plot3D files

- **write_to_hdf5**

  Saves all data in HDF5 format

  HDF5 File Structure

  - **fire:** 2-D array, size: N number of voxels by four

    [x, y, z, q] where x, y, and z represent the position while q represents the heat release rate

  - **smoke:** 2-D array, size: N number of voxels by four

    [x, y, z, q] where x, y, and z represent the position while q represents the smoke density

  - **min:** 1-D Array, Minimum value for [fire, smoke]
  - **max:** 1-D Array, Maximum value for [fire, smoke]

A.1.3 fdsPathLines.py

Takes plot3D data and calculates the amount of turbulence (represented as reynolds numbers) to identify regions of interest. Then calculates pathlines that will intersect with points sampled from these regions

- **Class instantiation**
  - Parameters
    - **directory:** Location of the FDS output files
    - **fds_input_location:** Location of the FDS input file.

- **__check_valid_file**

  Verifies file path is valid

- **__set_voxel_size**

  Calculates voxel size

- **get_position_from_index**

  Converts index value into a 3-D position

- **__set_minimum_pathline_length**

  Sets the minimum length of pathlines

- **filter_streams_by_length**

  Generates list of pathlines longer than minimum length

- **__add_ribbon_points**

  Generates a set number of pathline starting points between two given points

  - Parameters
    - **starting_point:** starting point for calculation
    - **ending_point:** ending point for calculation
    - **number_of_points** total number of starting points to be calculated

- **run_ode**

  Runs the ode solver for all saved starting points beginning at a specified timestep's index, can also run reverse integration

  - Parameters
    - **time_step_index:** indexed value of what timestep to start the ode solver from
    - **reverse_integration** A boolean that if set to true the ode solver will solve from designated timestep backwards until time is at zero

- **start_ode**

  Calls run_ode on all timesteps including reverse integration if needed

  - Parameters
    - **reverse_integration** A boolean that if set to true the ode solver will solve from designated timestep backwards until time is at zero

- **combine_ode_frames**

  Combines data frames from the forwards and backward integration

  - Parameters
    - **all_forward_data** Array of ode solver data frames
    - **all_backwards_data** Array of ode solver data frames

- **write_hdf5**

Saves all data in HDF5 format

HDF5 File Structure

- **maxValue:** Maximum magnitude of wind represented as a pathline in the current timestep
- **number_of_wind_streams:** Number of pathlines visualized in the current timestep
- **length_of_wind_streams:** Array the size of the number of wind streams indicating the number of line segments per pathline
- **windstream_{N}:** 2-D array, size: N length of current pathlines by five

  [Time, Velocity, x, y, z]

- **get_velocity**

  Returns the magnitude of the wind vector at a set point and time

  - Parameters
    - **re_time:** current time to be analyzed
    - **x:** 3-D array of position

      [x, y, z]

- **get_index_values**

  Returns the vector field index of a set point

  - Parameters
    - **x:** 3-D array of position

      [x, y, z]

- **add_reynolds_number**

  Adds the reynolds number to the ode solver data frame

- **__add_velocity_to_ode_data_frame**

  Adds the velocity to the ode solver data frame

- **draw_stream_lines**

  Uses matplotlib to draw pathlines for each timestep

- **__get_closest_time_step_index**

  Takes any time value and returns the index value of the nearest plot3D file

- **__get_reynolds_matrix**

  Returns a 3-D array of reynolds numbers for a specified timestep

- **__get_reynolds_number**

  Returns a reynolds number for a specified timestep

- **get_data_from_time**

  Returns a dataframe from the ode solver for a specified timestep

- **__get_max_re**

  Returns a maximum reynolds numbers from all dataframes

- **get_average_re_over_time**

  Returns a 3-D array of the mean reynolds number over a specified time range

- **get_mean_std**

  Calculates data ranges for histogram plots

- **__plot_points_re_range**

  Returns indices of all points that are within a set range of values

- **get_starting_positions**

  Uses k-means to cluster all points from **__plot_points_re_range** to calculate and return a list of centroids

- **get_all_starting_points**

  Calls **get_average_re_over_time, get_mean_std, __plot_points_re_range,** then **get_starting_positions** to generate all starting points of interest for the ode solver for a set range of reynolds numbers

- **set_turbulent_laminar_poi**

  Calls **get_all_starting_points** for the ranges of laminar and turbulent flow then combines the data sets

- **set_random_distro_poi**

  Generates a random set of starting points for the ode solver

- **set_even_distro_poi**

  Generates an evenly distributed set of starting points for the ode solver

A.1.4 main.py

Creates the proper file structure and ensures all functions save data into the proper location

- **main**

  Creates a file structure for Unity to read in, copies FDS input file, runs **fds2ComplexGeom.py, FdsPathLines.py,** then **fdsOutputToUnity.py**

A.2 FDS2UnityVR

Visualizes the data from **FDSPreUnityTool** in an immersive VR environment using Unity's Real-Time Development Platform

A.2.1 ConfigData.cs

Saves all information when transitioning from one scene to another

- **Awake**

  Checks for any data gathered from the main menu scene and saves them to the correct variables

- **getFireSmokeOption**

  Returns the selected option for fire, smoke, or none

- **setFireSmokeOption**

  Sets an option for fire, smoke, or none

- **getLoadTrees**

  Returns the selected option for trees or none

- **setLoadTrees**

  Sets an option for trees or none

- **getWindOption**

  Returns the selected option for windlines, windvectors, or none

- **setWindOption**

  Sets an option for windlines, windvectors, or none

A.2.2 hrrLoader.cs

Loads in a color mapped and animated fire particle engine in each voxel per timestep

- **Start**

  Creates a pointer to the particle engine that will be used for fire animation and saves all mesh data needed for scaling the size of the particles

- **LateUpdate**

  Places all particle engines mapped to correct color relative to heat release rate

A.2.3 MainMenu.cs

Verifies menu check boxes are selected and verifies fire and directory structure for input data

- **Start**

  Initializes progress bar

- **FireSmokeToggleChange**

Saves list of what fire, smoke, or none option is selected

- **TreeToggleChange**

  Saves changes when tree option is toggled from on/off

- **WindToggleChange**

  Saves changes when tree option is changed from wind path, wind vector, or none

- **PlayGame**

  When play button is clicked it preloads the next scene as long as the directory input is valid

- **Update**

  Moves the loading bar as long as the next scene is loading

- **UpdateDirTexField**

  Updates text field when VR keyboard is used

- **FieldInput**

  Validates that all file types are in input directory

- **FileExists**

  Checks if files exist in specific directory

A.2.4 PauseMenu.cs

Checks pause button input and tracks time in pause menu to allow for accurate timesteps after pausing

- **Start**

  Checks if pause option was preselected

- **Update**

  If pause button is pressed, change timescale from zero to one or one to zero

- **updateTime**

  Tracks time inside of pause menu to keep accurate timing once it is unpaused

A.2.5 smokeLoader.cs

Loads in a color mapped and animated smoke particle engine in each voxel per timestep

- **Start**

  Creates a pointer to the particle engine that will be used for fire animation and saves all mesh data needed for scaling the size of the particles

- **LateUpdate**

  Places all particle engines mapped to correct color relative to heat release rate

A.2.6 smvReader.cs

Loads in and caches heat release rate and smoke density data while tracking the current timestep

- **Start**

  Checks if fire, smoke, or none is selected, loads in binary file of sparse matrix data

- **getMeshData**

  Returns mesh data

- **sortedFileArray**

  Converts list of file paths to dictionary: where the keys are timesteps and the values are the file path

- **getFileTime**

  Parses file path to timestep

- **Update**

  Calls function to load in plot data

- **optimizedFDSLoader**

  Reads in smoke and fire data, saving to dictionaries to be accessed later

A.2.7 TerrainBuilder.cs

Loads a JSON of terrain data from fds2ComplexGeom.py A.1.1 and maps to a 3-D mesh with texturing

- **Start**

  Builds smooth 3-D mesh or cubic terrain based on input settings

- **MovePlayer**

  Moves player to the highest point on the topography

- **buildTerrainCubes**

  Builds topography with cubes

- **ParseFds**

  Reads FDS input file to parse mesh data

- **GetVerts**

  Reads in vertices from JSON file saves to mesh and saves as UVs for texturing

- **GetFaces**

  Reads in faces from JSON

A.2.8 TreeLoader.cs

Loads a JSON of tree data from fds2ComplexGeom.py A.1.1 and maintains scale regardless of the size of the 3-D tree object

- **Start**

  Checks if trees should be loaded, if so calls **fdsReader** function

- **fdsReader**

  Reads in tree data from JSON, places trees and scales them based on the size of the 3-D tree object to keep scale correct

A.2.9 WindStreams.cs

Loads in and caches pathline data from fdsPathLines.py A.1.3 to be visualized each timestep

- **Start**

  Checks if windlines or windvectors will be visualized, if not script is deactivated, wind data is loaded into dictionary

- **GetFileTime**

  Parses file name to get timestep

- **SortedFileArray**

  Sorts list of file paths based on timestep

- **readInData**

  Reads in binary file data

  Note: This is a legacy function that is no longer used

- **loadNextWindLines**

  Loads in the new pathlines

- **loadNextWindVector**

  Loads in the new path vectors

- **setMaxVelovity**

  Updates maximum velocity value if needed

- **setMinVelovity**

  Updates minimum velocity value if needed

- **getMaxVelovity**

  Returns maximum velocity

- **readInData2Dict**

  Reads in binary file data

  Note: This is a legacy function that is no longer used

- **readInDataHDF5**

Reads in all data from HDF5 files

- **Update**

  Checks if next timestep needs to be loaded in, if so it calls corresponding function based on menu selection

Bibliography

[Ser49] USDA Forest Service. *Mann Gulch Fire - wildfirelessons.net*. 1949. URL: https://www.wildfirelessons.net/HigherLogic/System/DownloadDocumentFile.ashx?DocumentFileKey=d648798c-a650-4596-ba81-7566696848d2&forceDialog=0.

[CL93] Brian Cabral and Leith Casey Leedom. "Imaging vector fields using line integral convolution". In: 1993, pp. 263–270.

[TGE97] Christian Teitzel, Roberto Grosso, and Thomas Ertl. "Efficient and Reliable Integration Methods for Particle Tracing in Unsteady Flows on Discrete Meshes". In: 1997, pp. 31–41. DOI: 10.1007/978-3-7091-6876-9_4.

[Rez+99] C. Rezk-Salama et al. "Interactive Exploration of Volume Line Integral Convolution Based on 3D-Texture Mapping". In: *Proceedings of the Conference on Visualization '99: Celebrating Ten Years*. VIS '99. San Francisco, California, USA: IEEE Computer Society Press, 1999, pp. 233–240. ISBN: 078035897X.

[Wie+03] Brenda K. Wiederhold et al. "10 An Investigation into Physiological Responses in Virtual Environments: An Objective Measurement of Presence". In: 2003.

[Ser13] USDA Forest Service. *Yarnell Hill fire - wildfirelessons.net*. 2013. URL: https://www.wildfirelessons.net/HigherLogic/System/DownloadDocumentFile.ashx?DocumentFileKey=4c98c51d-102c-4e04-86e0-b8370d2beb27&forceDialog=0.

[PBM14] Russell A. Parsons, Bret. W. Butler, and William "Ruddy" Mell. *Safety zones and convective heat: numerical simulation of potential burn injury from heat sources influenced by slopes and winds*. 2014. DOI: 10.14195/978-989-26-0884-6_165.

[Ser16] USDA Forest Service. *United States Department of Agriculture Twisp River Fire Fatalities and Entrapments*. 2016. URL: https://www.wildfirelessons.net/HigherLogic/System/DownloadDocumentFile.ashx?DocumentFileKey=177a62fd-9d28-1d94-707c-cb3ee4bce050&forceDialog=0.

[Ser20] USDA Forest Service. *El Dorado Incident – Learning Review Narrative*. 2020. URL: 1https://www.wildfirelessons.net/HigherLogic/System/DownloadDocumentFile.ashx?DocumentFileKey=9d91a48a-554c-d6e1-012c-95496d902494&forceDialog=0.

[McG+21] Kevin B McGrattan et al. *Fire Dynamics Simulator Technical Reference Guide Volume 3: Validation*. National Institute of Standards and Technology, 2021. DOI: 10.6028/NIST.SP.1018.

[WEH21] Steven G Wheeler, Hendrik Engelbrecht, and Simon Hoermann. "Human Factors Research in Immersive Virtual Reality Firefighter Training: A Systematic Review". In: *Frontiers in Virtual Reality* 2 (Oct. 2021). DOI: 10.3389/frvir.2021.671664.

[Bar22] Chris L. Barnes. HDBScan 28. 2022. URL: https://github.com/scikit-learn-contrib/hdbscan.

[Fac22] FaceBook. *Quest 2:* 2022. URL: https://store.facebook.com/quest/products/quest-2.

[USF22] United States Fire Administration USFA. *Firefighter Fatalities in the United States in 2020*. Mar. 2022. URL: https://www.usfa.fema.gov/downloads/pdf/publications/firefighter-fatalities-2020.pdf.

[Veg22] Vega. *Color schemes*. 2022. URL: https://vega.github.io/vega/docs/schemes/.