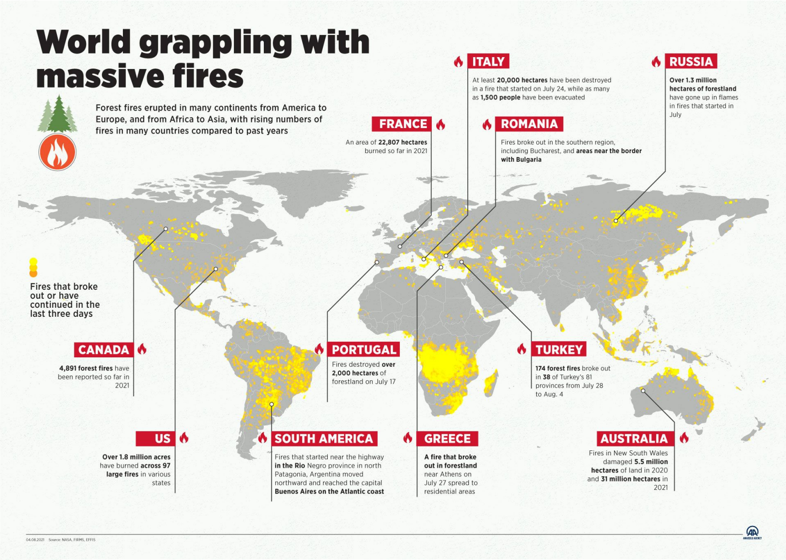
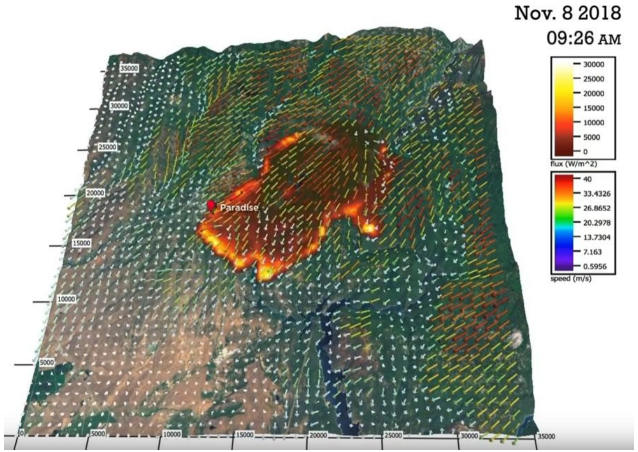
Simulating Wildfire Spread Using Physically Accurate Models

Stephen Lee, Anthony Mansur, and Lindsay Smith

Need and Overview

- As **climate change** continues to exacerbate the frequency and intensity of **wildfires** around world, it is becoming increasingly important to find new ways to **accurately predict** and combat their spread
- The behavior of wildfires is incredibly difficult to predict since accurate models require **many variables** that have traditionally been **computationally expensive**
- We want to develop a **GPU-based** tree burning **simulation** that leverages **parallelism** and **modular** tree designs to both efficiently and accurately simulate the spread of wildfires





Our Schedule

- **<u>Milestone 1</u>:** Setting up the **code framework** required for the simulation and render
  - What are the kernels and data structures needed to both define our system state and how that state will be updated
  - Initial, basic computation and visualization of key steps as proof of concept
  - A good starting point for the rest of the project
- ~~Milestone 2: Working **implementation** and **integration** of simulation + render~~
  - ~~“Hello world” sandbox visualizing 1-2 trees catching on fire~~
- ~~Milestone 3:~~
  - ~~Creation of the **forest** and **wildfire effects** (“fire” clouds, rain, wind, etc.)~~
- ~~Final: **Performance evaluation** + **parameter fine-tuning** + **Key findings**~~
  - ~~Wow factors, stress testing, making it unique~~

Code Framework

- C++/CUDA and OpenGL Project
  - Used Project 1: Boids simulation as a guide to our framework
- Main.cpp
  - Initiates CUDA and OpenGL pipeline
  - Creates window
  - Calls our simulation's step function
- Kernel.cu
  - Includes header files to our device function pointers and kernel prototypes
  - Contains our InitSimulation(), StepSimulation(), and EndSimulation() functions

Simulation Overview

What's does our step function do?

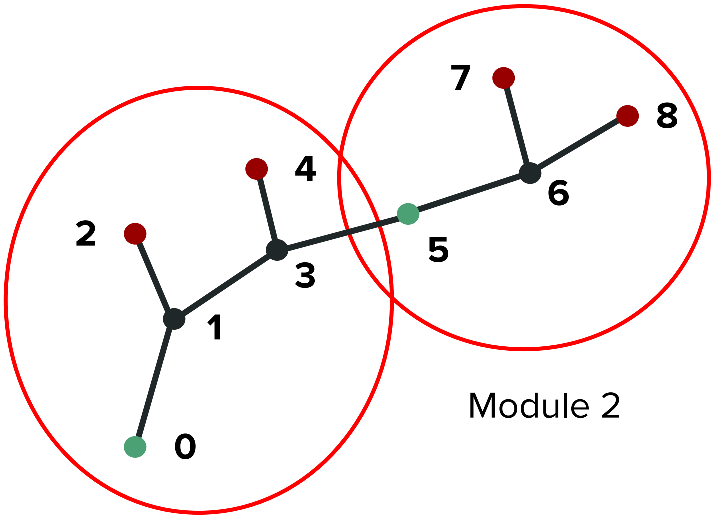
ALGORITHM 1: Overview of our simulator's numerical procedure.
*Please note that $\psi_{\mathcal{M}}$ can be precomputed for all $\mathcal{M} \in \mathcal{F}$.

Input: Current system state.
Output: Updated system state.

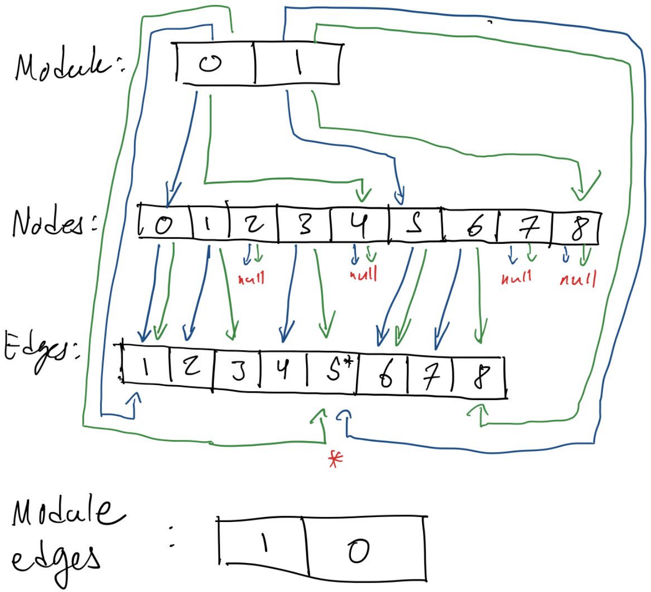
for each module $\mathcal{M} \in \mathcal{F}$ **do**
  | Update mass $M := M(\mathcal{M})$ according to Eq. (1).
  | Perform radii update according to Eq. (11–12)*.
  | Update temperature $T_{\mathcal{M}}$ according to Eq. (30).
  | Update released water content $W := W(\mathcal{M})$ according to Eq. (17).
end
for each grid point $\boldsymbol{x} \in \mathcal{D}(\Omega)$ **do**
  | Update $M := M(\boldsymbol{x})$ and $W := W(\boldsymbol{x})$ as described in Section 5.1.
end
Update temperature T according to Eq. (21).
Update drag forces $\boldsymbol{f}_d$ according to Eq. (15).
Update q_v, q_c, q_r, q_s, and $\boldsymbol{u}$ according to Hädrich et al. [2020] including vorticity confinement with intensity ϵ.
for each module $\mathcal{M} \in \mathcal{F}$ **do**
  | **if** $M := M(\mathcal{M}) = 0$ **then** $\mathcal{F} \leftarrow \mathcal{F} \setminus (\{\mathcal{M}\} \cup \text{children}(\mathcal{M}))$
end

MESOSCALE SIMULATION OF WILDFIRES

Part 1: Module-level Combustion

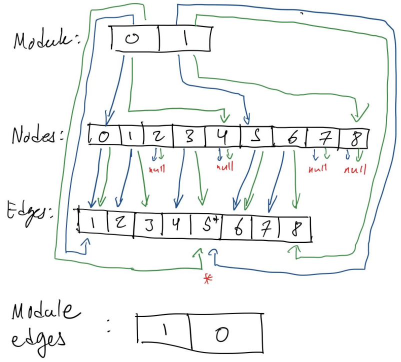


Data structure (Graph)



Graph representation paper: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.102.4206&rep=rep1&type=pdf

Part 1: Module-level Combustion

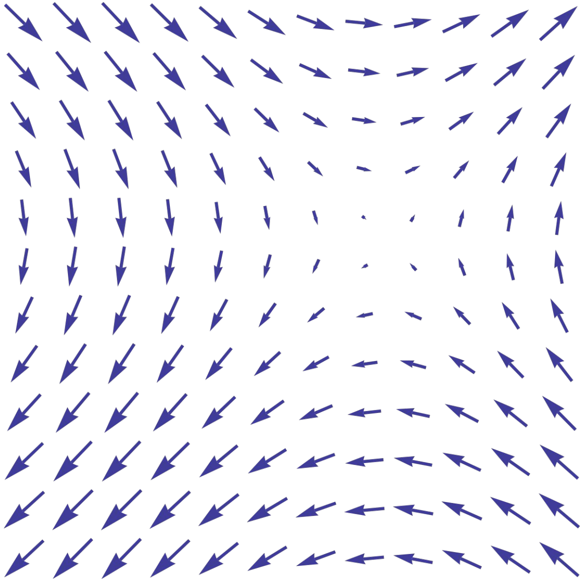


- For each module in the forest **(embarrassingly parallel)**
  - Update its mass due to combustion
  - Update the radius of each branch within the module
  - Update its surface temperature
  - Update the released water content due to mass loss
- <--- Fluid Solver goes here --->
- For each module in the forest **(culling!)**
  - Delete modules in the forest if its mass is zero

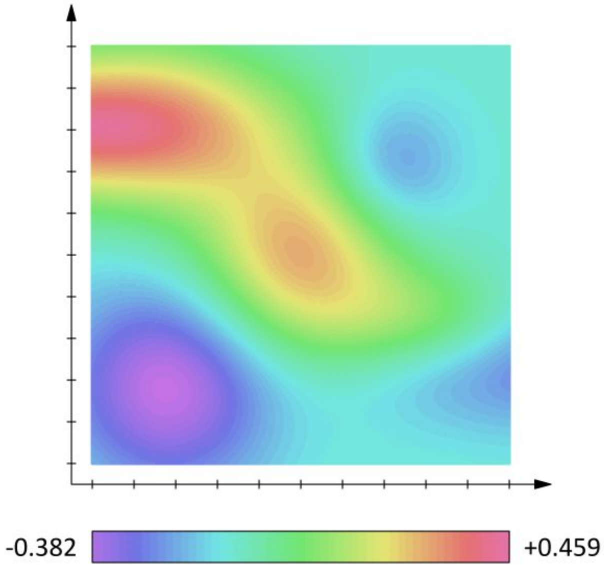
Part 2: Fluid Solver

- For each grid cell in the simulation space **(also embarrassingly parallel!)**
  - Update the vorticity
  - Update the velocity wind field
  - Update the air pressure
  - Update the air temperature
- 3D grid represented as 1D array of grid values with (x, y, z) parameters to lookup and flatten to determine location in 1D array

Part 2: Fluid Solver



Vector fields: vorticity and velocity



Scalar fields: pressure and temperature

Part 2: Fluid Solver

Account for external confinement forces due to turbulence

Update temperature scalar values based on advection from wind flow

Vorticity → Velocity → Temperature

Compute pressure changes from last time step and update for force calculations on velocity

Pressure

Rendering Overview

How are we going to see

Rendering

- Based on the first few projects we created a simple camera and plane using OpenGL that will serve as the base of our terrain
- We will render each branch of the tree by storing two 4D-vectors at the beginning and end of the branch, with radius as the fourth component
    - Will use a tessellation shader to transform the points to truncated cones
    - Textures will be applied to each branch to make it look more realistic
- Simulation will be updating the radius component as the tree burns
- For rendering the fire, we are thinking about mapping the temperature at each piece of the grid and interpolating between them to get the desired effect
- We also are looking into how OpenGL particles work and whether that would be applicable here

Our Schedule

Milestone 2: Working **implementation** and **integration** of simulation + render

- "Hello world" sandbox visualizing 1-2 trees catching on fire
- What needs to be done:
    - Generate a (very simple) forest and pass that to our simulation and rendering buffers for us to test our kernels and rendering pipeline
    - Add kernels for smoke generation and tracking of water content within grids for smoke cloud effects
    - Fully integrate all grid-level effects with all module-level effects (for example module burning rate is dependent on grid temperature and wind velocity parameters)