StormSynth

September 29, 2024

1 StormSynth: AI-Driven Storm Simulation and Visualization

Owen Kroeger **-Project Video Link-**

1.1 Final High-Level Mock Version

1.1.1 Project Overview

The goal of this project is to create a 3D visualization that allows users to observe the full lifecycle of a storm based on environmental input parameters. This project leverages neural networks to model storm behavior, while Unreal Engine provides real-time 3D rendering of the storm lifecycle.

Users will be able to adjust environmental conditions (e.g., temperature, humidity, wind speed), and the system will generate storm simulations that evolve in real-time. The aim is to combine the power of machine learning and real-time graphics to provide an interactive educational experience that showcases storm formation, development, and dissipation.

1.2 Refined UI Sketches

The user interface (UI) will allow users to interact with the storm simulation system through a variety of controls and data displays. Below are the key UI components:

1.2.1 Key UI Components:

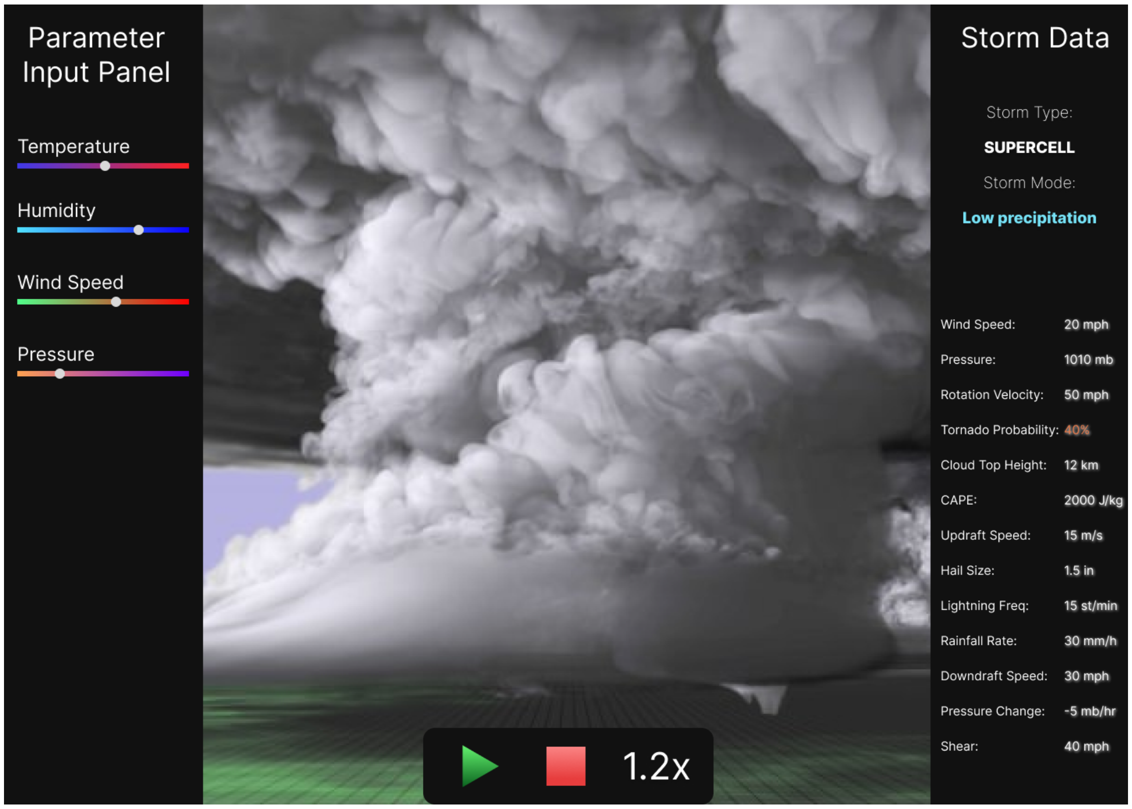
1. **Parameter Input Panel**:
   - Users will be able to adjust environmental parameters such as:
     - Temperature
     - Humidity
     - Wind Speed
     - Pressure Levels
   - The input will be provided via sliders or text input fields for precise control.
2. **3D Storm Visualization Window**:
   - A 3D view of the storm lifecycle will be displayed, showing cloud formations, rain, and other visual elements of the storm.
   - The user can rotate, zoom, and pan within this window to explore the storm from different angles.
3. **Control Panel**:
   - Includes buttons to:
     - Play/Pause the simulation
     - Adjust the simulation speed (slow down/speed up storm development)

    - Reset the simulation to its initial state.

4. **Storm Data Display**:
    - Real-time data related to the storm will be displayed in a side panel, including:
        - Current wind speed (in miles per hour)
        - Pressure levels (in millibars)
        - CAPE (in J/kg)
        - Tornado Probabililty

1.2.2 Example UI Wireframe:

Below is a wireframe representation of the planned user interface.



Storm Type: **SUPERCELL**

Storm Mode: **Low precipitation**

Wind Speed:	20 mph
Pressure:	1010 mb
Rotation Velocity:	50 mph
Tornado Probability:	40%
Cloud Top Height:	12 km
CAPE:	2000 J/kg
Updraft Speed:	15 m/s
Hail Size:	1.5 in
Lightning Freq:	15 st/min
Rainfall Rate:	30 mm/h
Downdraft Speed:	30 mph
Pressure Change:	-5 mb/hr
Shear:	40 mph

1.3 Detailed Functionality

1.3.1 1. Parameter Input Panel

- **Function**:
    - Allows users to adjust environmental parameters, such as:
        * **Temperature** (°F/°C)
        * **Humidity** (%)
        * **Wind Speed** (mph/km/h)
        * **Pressure Levels** (mb)
    - Each slider updates the storm simulation in real-time by sending the selected values to the storm model.
- **How It Works**:

    - User adjusts a slider, and the corresponding value is updated in the system's data structure.
    - For example, moving the **temperature slider** to a higher value will trigger the storm simulation to modify cloud height and intensity accordingly.

[ ]:
```
# Example code for handling slider input
temperature = slider_value  # Value from slider input
update_storm_model(temperature=temperature)
```

1.3.2 2. 3D Storm Visualization Window

Function:

- Displays a real-time, interactive 3D visualization of the supercell storm, allowing users to view the storm from different angles.
- Users can rotate, zoom, and pan to explore different parts of the storm.

How It Works:

- The visualization window continuously updates to reflect changes in the storm model as the environmental parameters are adjusted.
- Users can interact using mouse controls:
    - **Click-and-drag**: Rotate the storm model.
    - **Scroll wheel**: Zoom in and out to focus on specific parts of the storm.
    - **Right-click and drag**: Pan the view.

1.3.3 3. Control Panel

Function:

Allows users to control the simulation using: - **Play/Pause**: Start or stop the storm simulation. - **Speed Control**: Adjust the simulation speed to view the storm development in fast-forward or slow motion. - **Reset**: Reset the simulation to the initial state.

How It Works:

- **Play/Pause**: Starts or stops the simulation loop that updates the storm model every frame.
- **Speed Control**: Adjusts the time step in the simulation to speed up or slow down the evolution of the storm.
- **Reset**: Resets all parameters (temperature, humidity, etc.) to their default values and restarts the simulation from the beginning.

1.3.4 4. Storm Data Display

Function:

Displays real-time storm data, including: - Wind Speed - Pressure Levels - Tornado Probability - Rotation Velocity - Cloud Top Height

How It Works:

- As the storm evolves, real-time data from the simulation model is sent to the **data display panel**.

- Values are updated dynamically, reflecting the current state of the storm.
- The color coding of the data (e.g., red for high wind speeds) provides visual feedback to indicate the severity of the storm.

1.3.5 5. Interaction with Storm Simulation Model

Function:

The entire interface is built to interact with the underlying storm simulation model, which processes environmental inputs and updates the 3D visualization and data display.

How It Works:

- The storm model uses the user-provided inputs (temperature, wind speed, etc.) to calculate the storm's current state. This model is updated frame by frame to simulate the evolution of the supercell storm.
- **Example of storm model**: The model could be built using neural networks that use mathematical equations to represent storm physics. These calculations are tied to the input parameters, and the resulting storm behavior (e.g., cloud height, intensity) is reflected in the visualization.

[ ]:
```python
# Pseudo-code for the storm model update
def update_storm_model(temperature, humidity, wind_speed, pressure):
    # Apply physics or neural network calculations to generate storm behavior
    storm_data = calculate_storm(temperature, humidity, wind_speed, pressure)
    return storm_data
```

1.4 Data Structures and Computations

1.4.1 1. Input Data Structures

These are the structures that hold the environmental parameters (like temperature, humidity, wind speed, etc.) provided by the user.

[ ]:
```python
# Input data structure to store environmental parameters
input_data = {
    "temperature": 75,  # in degrees Fahrenheit
    "humidity": 80,     # in percentage
    "wind_speed": 15,   # in miles per hour
    "pressure": 1010    # in millibars
}
```

1.4.2 2. Storm Data Structures

These structures will hold the evolving storm data that gets updated as the simulation progresses.

[ ]:
```python
# Data structure to store storm-related variables
storm_data = {
    "cloud_height": 12,       # height of clouds in kilometers
    "wind_velocity": 50,       # wind speed inside the storm in mph
    "rotation_velocity": 20,   # mesocyclone rotation velocity in mph
    "pressure_drop": 15        # pressure difference in millibars
}
```

1.4.3 3. Computations

Storm Model Calculations: The key computations that simulate the storm's behavior based on the user's input. This could involve physics-based models or neural networks.
For instance, a formula might be used that adjusts the cloud height based on changes in temperature or pressure.

[ ]:
```
# Example function to simulate storm dynamics
def update_storm_model(temperature, humidity, wind_speed, pressure):
    # Apply physics-based calculations to model storm behavior
    cloud_height = calculate_cloud_height(temperature, pressure)
    rotation_velocity = calculate_rotation_velocity(wind_speed, pressure)

    # Update the storm data
    storm_data["cloud_height"] = cloud_height
    storm_data["rotation_velocity"] = rotation_velocity
    return storm_data
```

1.4.4 4. Real-Time Updates

Each time the user adjusts a parameter or the simulation progresses, these data structures and computations will be updated in real-time to reflect changes in the storm simulation.

[ ]:
```
# Main simulation loop to continuously update storm model
while simulation_running:
    storm_data = update_storm_model(
        input_data["temperature"],
        input_data["humidity"],
        input_data["wind_speed"],
        input_data["pressure"]
    )
    update_visualization(storm_data)
```

1.5 APIs and Development Tools

1.5.1 1. APIs

OpenWeatherMap API (or a different weather API)

- **Purpose**: Fetch real-time or historical weather data (such as temperature, wind speed, humidity, etc.) to provide accurate input parameters for the storm simulation.

- **Usage**: This API will allow the system to pull real-time environmental data, which can then be fed into the storm simulation model to create more realistic storm behavior.

[ ]:
```python
# Example API request using OpenWeatherMap API
import requests

api_key = "your_api_key"
city = "your_city"
url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}"

response = requests.get(url)
weather_data = response.json()

# Example usage: extract temperature, wind speed, etc.
temperature = weather_data['main']['temp']
wind_speed = weather_data['wind']['speed']
```

1.5.2 Other Possible APIs:

- **NOAA Weather API**: A government-provided weather API that offers real-time and historical data, ideal for detailed weather modeling.
- **Custom API for Climate Data**: May have custom APIs to fetch additional environmental data.

1.5.3 2. Development Tools

Unreal Engine

- **Purpose**: Unreal Engine will be used for real-time 3D visualization of the storm simulation. It allows the creation of realistic storm animations and provides user interaction tools (e.g., camera controls).
- **Key Features**:
  - 3D rendering of the storm lifecycle.
  - Integration with Python for controlling simulation parameters and storm behavior.

Python

- **Purpose**: Python will be used for all the backend logic, including:
  - API requests to gather real-time weather data.
  - Processing storm models (either physics-based or neural network-based).
  - Communicating with Unreal Engine for visual updates.
- **Key Libraries**:
  - **Requests**: For API requests to fetch weather data.
  - **NumPy**: For handling computations and data manipulation.
  - **TensorFlow/PyTorch**: Using machine learning for storm behavior prediction.

Jupyter Notebooks

- **Purpose**: Used for documentation, prototyping, and running simulations during the development process.

1.6 Final Outputs

1.6.1 1. Visual Output

- The project will generate a real-time 3D visualization of a storm's lifecycle, starting from the storm's initial formation to its dissipation.
- Users can interact with the 3D environment, allowing them to:
  - Rotate, zoom, and pan around the storm.
  - Observe changes in storm characteristics such as cloud height, rotation velocity, and rain intensity.
- The sky and cloud formations will change visually to reflect storm intensity.

1.6.2 2. Data Output

- The simulation will provide real-time data which will be updated continuously and displayed in a side panel, giving users real-time insight into the storm's behavior.

1.6.3 3. Performance Metrics

- **Frame Rate (FPS)**: The system will display the frame rate (FPS) to ensure smooth visualization of the storm.
- **Simulation Speed**: Users will be able to control the speed of the simulation and see how this impacts the storm's lifecycle. For example, they can fast-forward the storm to see how it evolves quickly, or slow it down to observe detailed behavior.

1.6.4 4. User Interaction

- Users will have control over environmental inputs (such as temperature, wind speed, and pressure) and see how these changes affect the storm in real time.
- Each user interaction will update both the visualization and the data output dynamically.
- The control panel will allow users to:
  - Play/Pause the storm simulation.
  - Reset the storm to its initial state.
  - Speed up or slow down the simulation.

1.7 Challenges and Solutions

1.7.1 1. Performance Bottlenecks

- **Challenge**: Rendering complex storm systems with many particles and effects in real-time may cause performance slowdowns.
- **Solution**:
  - Optimize the 3D model and particle systems to minimize the number of real-time calculations.
  - Use LOD (Level of Detail) techniques to reduce complexity when the user is zoomed out.
  - Implement performance profiling to ensure smooth operation and address any FPS issues.

1.7.2 2. Large Data Handling

- **Challenge**: Handling large datasets for weather data and storm parameters can strain memory and computation time.
- **Solution**:
  - Use efficient data structures and minimize the number of stored variables by computing values only when necessary.
  - Store preprocessed storm data for specific conditions to reduce real-time calculations.

1.7.3 3. Neural Network Training

- **Challenge**: Training a neural network to accurately model storm behavior may be difficult without extensive weather data.
- **Solution**:
  - Start with simplified weather datasets and gradually refine the model.
  - Perform validation using known storm patterns and adjust the model to avoid overfitting.

1.7.4 4. Realistic Storm Visuals

- **Challenge**: Accurately representing storm visuals (clouds, rain, lightning) in real-time can be resource-intensive and complex.
- **Solution**:
  - Use Unreal Engine's built-in weather and particle systems, and tweak them for storm-specific visuals.
  - Adjust transparency and lighting dynamically to simulate realistic changes in weather as the storm evolves.

1.7.5 5. User Interface Complexity

- **Challenge**: Managing a large number of input sliders, buttons, and real-time data displays can overwhelm the user interface.
- **Solution**:
  - Keep the interface minimalistic and only display essential information.
  - Use tooltips or collapsible panels to hide advanced options until needed.