May 4, 2025

Anjali Nuggehalli - Project Report, Computer Graphics

Rain and Grass Build

Alex Kalis, **Anjali Nuggehalli**, Will Deley, Seo-Eun Kim

This project simulates a dynamic outdoor environment featuring falling rain and animated grass using Babylon.js and custom GLSL shaders. The most important features include generated geometry for both the grass and raindrops, real-time GPU animation of grass swaying in the wind, and looping rainfall rendered with custom fragment shaders for shape and coloring. Our goal was to explore the connection between real-time graphics and shader programming by building a scene that feels realistic and dynamic, using minimal geometry and maximum GPU-based effects. We were inspired by our own observations of nature - how grass bends in the breeze and how rain visually distorts our perception of the environment. We wanted to replicate that kind of motion in a digital scene using WebGL.

I was responsible for implementing the rain system in our Babylon.js scene, which included procedural geometry creation, real-time animation using custom GLSL shaders, and ambient sound integration. To simulate falling rain, I created triangle-based raindrop geometry directly in JavaScript. Each raindrop was positioned randomly within a defined range, and the geometry was kept lightweight to allow many drops to be rendered efficiently. I wrote the vertex shader to animate the rain by moving each drop downward based on a time uniform and used modulus logic to loop the motion continuously. To make the effect more realistic, I adjusted transparency in the shader based on vertical position, fading drops out near the ground or outside the camera's view. The accompanying fragment shader discards pixels outside a circular center, creating soft-edged drops, and applies a subtle blue tint to match the mood of the scene.

One of the main challenges was understanding how to structure vertex and fragment shaders in Babylon.js. We initially struggled with writing correct GLSL code and passing time-based animations through uniforms. Debugging shaders also posed a challenge, as shader compilation errors can be convoluted to trace. Another unexpected challenge came from adding ambient rain sound to the scene using an mp3 file. We downloaded and locally hosted rainsounds.mp3, and used Babylon's Babylond.Sound class to load and autoplay it. While it worked locally in some browsers, others (especially Chrome) blocked autoplay due to user interaction restrictions. We explored multiple solutions, including autoplay : true and setting the sound to play on scene load, but none worked consistently without user input.

Overall, we are proud of our final project, as the grass and rain both look natural. The grass has a gradient color scheme to look like realistic blades, and the rain has the proper effect due to custom fragment shaders of lighting and color. We learned a lot about GPU programming, shader architecture, and how WebGL rendering is orchestrated through Babylon.js. If we had more time, we would have added environmental lighting effects like lightning flashes, fog, or reflections on the ground to create a more atmospherically realistic scene.

Sources

- Babylon.js Documentation: used for understanding scene setup, materials, and ShaderMaterial usage.
- WebGL Fundamentals Website: clarify how vertex and fragment shaders work
- The Book of Shaders: understanding of how to build and animate GLSL shaders
- Mozilla Web Documentation on HTML and JavaScript
- Babylon.js Playground Examples/Past Class Assignments
- Babylon.js Tutorial Youtube Playlist

May 4, 2025

Alex Kalis - Project Report, Computer Graphics

Rain and Grass Build

Alex Kalis, Anjali Nuggehalli, Will Deley, Seo-Eun Kim

Introduction

The project we decided on was to simulate a natural outdoor environment with animated grass swaying in the wind as well as falling rain which we built using Babylon.js and custom GLSL shaders. We focused on GPU accelerated animation and procedural mesh generation, so we could create a more responsive scene. This project combines shader programming with real time rendering techniques to help us model rain and grass efficiently. Our goal was to explore the connection between geometry, shader programming, and modeling outdoor environments. Lastly, our inspiration for this project was exploring the idea of modeling how rain and wind could affect outdoor scenery like grass. My specific contributions included troubleshooting and implementing the rain audio, reviewing and refining the rain droplet shader code for appearance and motion, and assisting with adjustments in the grass fragment shader to improve the color gradient.

Challenges

A major challenge that we faced was the learning curve involved with writing and debugging the GLSL shaders, specifically with how to structure and pass uniforms like time into the shaders. Also, the shader compilation errors didn't have complete clarity, so we needed to experiment and isolate code segments to debug. Integrating the rain sound also caused some issues because we had issues getting the sound to play, which we figured is most likely due to the inconsistency between browser restrictions. To fix this I added a click event trigger which allows us to start the sound manually. I also wasn't able to get the sound file to play for a sustained period of time, so I looped the track in order to avoid this. I also tried to explore the idea of adding terrain as well to create a more realistic nature scene but eventually due to time constraints focused solely on the grass, rain, and sound.

Discussion

In conclusion, our group did a great job in our goal of simulating an outdoor environment with rain and sound, especially the fact that we were able to include a large number of grass elements with simulated movement. I loved that we were also able to include rain with sound to try and simulate the raindrops falling on the grass. With more time I would've loved to explore the idea of adding terrain to make a more complete environment and also working with the time and shape of the rain drops to try and simulate snow as well. By adding snow in some areas like terrain and then rain over the grass I could definitely see how we could make an even better scene.

Sources

-Babylon.js Documentation - for scene setup, ShaderMaterial, audio
-https://thebookofshaders.com/ - GLSL syntax + animation
-https://webglfundamentals.org/ - vertex/fragment shader structure
-MDN Web Docs
-Babylon.js Playground and Youtube tutorials
-https://pixabay.com/sound-effects/search/rain%20hitting%20grass/ - audio mp3 file
-https://doc.babylonjs.com/features/featuresDeepDive/audio/intro - audio

May 4, 2025

Will Deley - Project Report, Computer Graphics

Animating Rain and Grass

Will Deley, Alex Kalis, Seo-eun Kim, Anjali Anjali Nuggehalli

Introduction

This project simulates a natural scene of rain falling onto grass. Our goal was to make it look as realistic as possible, with a full scene encompassing a rainy day. We used Babylon.js and custom GLSL shaders to make the swaying grass and falling rain come to life. We created custom messes for the grass blades, and field as a whole, along with a custom rain mesh. These were then animated in the GPU. We took a lot of inspiration from the outside world, and figured combining the grass, rain, and some wind would make the scene look as close to nature as possible. My specific contributions came in working with the GLSL rain shaders to make the rain look more realistic and rain shaped. I also worked on making the rain, and the entire scene smoother while running, this included smoothing grass movement and rain descension.

Challenges

One large challenge that I addressed in this project was the issue of lag in our grass and rain scene. When opened in a browser the window would move way too slow, making rain and grass movements appear fake. Using some documentation I found I could use mesh.freezeWorldMatrix() to not recompute the matrix of the mesh unnecessarily. This made the lag much more manageable, and after some changes in the movement of the grass and rain I was able to make their movement much more smooth and realistic. I also ran into many issues when editing the custom GLSL shaders. An issue in these shaders does not provide much output, making it difficult to debug. I used stack overflow articles of similar custom shaders in order to pinpoint what little mistake could be causing the shader to not compile. One challenge I wish I had been able to solve was maintaining a large level of grass blades in the final implementation. I downscaled the number of blades to 6000, enough to make the grass patch full, but not enough to cause immense lag. Ideally this number would be much closer to 10000, but with our implementation and my low level graphics on my computer that was a difficult task.

Discussion

We did a fantastic job at simulating a real-world environment that looks realistic and combines both the moving grass and rain into one scene, along with sound. We all overcame issues with the project, one of which was the sound of the rain and it really brings the whole project together in the final implementation. With some more time I would have loved to expand the scene to fully fill an entire environment. It would have been nice to have some camera movement where you can move around the scene where there are hills of grass and rain falling all over. We also would have implemented occasional wind gusts that would affect the rain and grass together, improving realism.

Sources

- Babylon.js Documentation - used for basic setup of the scene and normal functions
- WebGL Fundamentals Website - used to setup and debug shaders
- Babylon.js Playground Examples/Past Class Assignments - used for reference, especially when creating shaders
- Stack Overflow - various posts about debugging GLSL shaders to pinpoint issues
- https://doc.babylonjs.com/features/featuresDeepDive/scene/optimize_your_scene - babylon.js used to optimize the scene

May 4, 2025

Kim, Seo-Eun - Project Report, Computer Graphics

Rain and Grass Build

Seo-Eun Kim, Alex Kalis, Anjali Nuggehalli, Will Deley

Introduction

For our project, we aimed to replicate a realistic scene of swaying grass on a rainy day outdoors using babylon.js and shaders. By using real-time rendering techniques and randomization for custom mesh generation, our project had the natural variation in motion and shape that exists in real-life environments. We were inspired by the sway and movements that exist in nature, to take notice of small details and qualities that add towards realism that typically might not be noticed. My contribution was to make the grass mesh generation, from making the single blade as well as making an algorithm to generate the larger grass field with randomized blades of grass. I used randomization to add variation to the height and width of each blade of grass, as well as randomizing the grass blade position and angle. The angle variation required the use of basic trigonometric functions which added a layer of complexity to the field generation.

Challenges

We encountered a large learning curve with writing the GLSL shaders, as making sense of the compilation errors was difficult. There were also some challenges adding the rain sound to the project, as browser restrictions on chrome blocked the mp3 file from automatically loading with the html file. To get around this issue, we had to add a click event trigger which allows the user to manually start the sound. One personal challenge I came across was finding the proper degree of randomization to use for the grass field. I wanted to make sure there was enough variation to make the field look not too much like a grid with the same blade mesh copied and pasted to the scene, but not too much variation that would look unnatural. I found that using a small range of size variation and even spacing/angularity with a high blade count generated a realistic scene.

Discussion

I believe our group was able to make a cohesive, complete outdoor scene that incorporates two different kinds of motion, rain and grass. With more time, I think we could've added a more realistic shader to the grass that simulates sunlight shining onto the field and partially through the grass. This would've added a degree of realism by making the grass material look less plastic and more organic, as opposed to the straightforward gradient that we finished the project with.

Sources

- Custom mesh generation
- Babylon mesh documentation
- Mesh Transformation, Parents and pivots