applied sciences

MDPI

Article

CWD-Sim: Real-Time Simulation on Grass Swaying with Controllable Wind Dynamics

Namil Choi and Mankyu Sung *

Department of Computer Engineering, Keimyung University, Daegu 42601, Republic of Korea; chnamil21@gmail.com
* Correspondence: mksung@kmu.ac.kr

Abstract: In this paper, we propose algorithms for the real-time simulation of grass deformation and wind flow in complex scenes based on the Navier–Stokes fluid. Grasses play an important role in natural scenes. However, accurately simulating their deformation due to external forces such as the wind can be computationally challenging. We propose algorithms that minimize computational cost while producing visually appealing results. We do this by grouping the grass blades and then applying the same force to the group to reduce the computation time. We also use a quadratic equation to deform the blades affected by the wind force rather than using a complicated spline technique. Wind force is fully modeled by the Navier–Stokes fluid equation, and the blades react to this force as if they were being swept by the wind. We also propose the AGC interface (Arrow-Guided wind flow Control), which allows the direction and intensity of the wind to be manipulated using an arrow-shaped interface. Through this interface, users can have grass sway in response to user-defined wind forces in a real-time rate. We verified that the proposed algorithms can simulate 900% more grass blades than the compared paper's algorithms.

Keywords: interactive visualization; natural scene visualization; grass animation; real-time simulation; fluid dynamics in graphics

Citation: Choi, N.; Sung, M. CWD-Sim: Real-Time Simulation on Grass Swaying with Controllable Wind Dynamics. *Appl. Sci.* **2024**, *14*, 548. https://doi.org/10.3390/app14020548

Academic Editor: João M. F. Rodrigues

Received: 29 November 2023
Revised: 1 January 2024
Accepted: 6 January 2024
Published: 8 January 2024

1. Introduction

Simulating natural phenomena presents a significant challenge but is essential in computer graphics, especially for creating realistic scenes in applications like video games and virtual environments. Grass, ubiquitous in natural landscapes, plays a pivotal role. The accurate simulation of grass swaying in the wind necessitates a detailed modeling of each blade and an in-depth understanding of the wind flow dynamics. Achieving such realism requires sophisticated physics algorithms capable of simulating intricate wind patterns and blade deformation along with substantial computing resources to simulate and render a large number of blades effectively.

In this paper, we introduce the Controllable Wind Dynamics (CWD) techniques, which were designed to facilitate the real-time simulation of numerous grass blades interacting with external forces. This approach leverages the parallel computation capabilities of GPUs for the simulation, deformation, and rendering of grass blades. To minimize unnecessary transfer overhead between the CPU and GPU, all data updates are confined to the GPU memory buffer. The computation of blade deformation is contingent upon the direction and magnitude of the artificially generated wind. We achieve a precise representation of wind force and its interaction with the blades through fluid simulation governed by the Navier–Stokes equations, which are fundamental to fluid dynamics. The methodology for implementing fluid simulation using the Navier–Stokes equations is extensively documented. In our research, we have adopted the methods delineated in [1–5].

The reason why the CWD-Sim algorithm uses minimal computational resources compared to previous methods is that it uses a combination of techniques specifically

designed to optimize simulation steps. First, unlike the method proposed in [6], which uses Bezier curves to deform the grass blades, our method uses a simple quadratic equation to stretch the grass blade model vertically and bend it in all directions. This approach requires fewer operations than spline curves, although both produce similar results. Second, instead of simulating individual blades, we group them based on their world positions and place them in a grid structure. All blades in a group can have different deformation effects, even if they are exposed to the same wind force because they have slightly different initial physical properties. This grouping significantly reduces the computation time without causing any noticeable visual artifacts. Through experiments, we have found that the computation speed remains almost constant regardless of the number of blades and objects. Essentially, the value of a cell on the grid computed by the fluid simulation determines the curvature, orientation, and shadow of the blade through specific separate equations. In particular, we use the quadratic equation to deform the blade model into a curved shape, as if it were under the influence of gravity. The curved shape of the blade model can also be bent or stretched by external wind forces.

An important problem to be addressed is how to efficiently specify the direction and force of the wind in the environment. Our method proposes the AGC (Arrow-Guided wind flow Control) interface, which allows users to intuitively control wind flow. The interface adds a set of 2D arrows that represent wind directions for a given time period directly into the environment. These arrows are connected to control the flow. Using this interface, users can manage complex flows, such as branching and merging of the wind.

The remaining sections consist of the following. Section 2 provides an overview of related work and a comparison with the proposed algorithm. Section 3 describes the technical details of the CWD-Sim algorithms. Section 4 presents the experimental results and performance graphs. Finally, Section 5 concludes the paper with a discussion and outlines future work that could improve our CWD method.

2. Related Works

2.1. Static Grasses

In recent years, several methods have been proposed for real-time grass simulation. For example, ref. [7] proposed a non-dynamic method to render more than 627,000,000 virtual grass blades in real time at 18 fps. However, this method could not simulate the deformation of grass by external forces, such as the wind or objects, and could only render a static grass model without dynamic grass deformation. Similarly, Deussen et al. proposed a method that did not focus on rendering time [8]. It showed the most colorful plant composition among the papers referenced, but it could only render a static grass model and takes 75 min to render the scene.

2.2. Grass Deformation with External Forces

Habel focused on real-time vegetation rendering and animation [9] but did not specifically address the aspects of wind interaction and manipulation in detail. Chen et al. presented a 2D approach to animate 3D vegetation in real time [10]. While their previous method proposed a simple method to animate vegetation with billboard images based on simulation-guided grid-based warping, the methods did not provide specific features for the wind interaction. Qiu et al. proposed a rendering system for large-scale grass [11]. The three-layer framework separated the rendering task from the data logic, making it convenient to add new vegetation simulation methods on the data layer, but it did not propose an interaction with external forces. Max et al. proposed a method for rendering grasses blowing in the wind with global illumination [12] using a lattice Boltzmann model, a mass-spring system and multiple scattering. However, since the simulation and rendering were performed on the CPU, performance was limited. Fan et al. utilized physical laws to simulate the movement of grasses deformed by a rolling ball [13]. The authors were able to reduce the computational load by activating and deactivating tile groups, which is the subdivision of the environment, as the ball passes over them for a

certain period of time. Although this approach showed highly dynamic grass interactions, it did not account for interactions with the wind. Furthermore, if global wind affecting the entire scene or interactions with rigid body objects was required, then this method would result in a significant computational burden. Similarly, Wang et al. proposed a GPU-based grass simulation with accurate blade reconstruction [14], which focused on improving the grass blade representation. But it still did not address the wind interaction and manipulation extensively.

2.3. Grass Deformation with Fluid Dynamics

In [6], Lo et al. used a 60 × 60 × 20 3D Navier–Stokes simulation for wind dynamics, and each grass blade calculated four control points of the parametric spline to represent a curved shape swaying by the wind. Although their approach was able to produce highly realistic grass animation, simulating 3D fluids and finding four control points of each blade of grass were computationally intensive for large scenes.

Our method proposes a 1000 × 1000 2D Navier–Stokes simulation for wind dynamics instead. Complex wind dynamics created by the proposed method and its interaction with grasses in Figure 1. Our method produces more detailed wind interaction than [6] and is able to cover larger complex scenes due to a more detailed and highly optimized wind dynamic control scheme. For instance, our quadratic equation for the deformation of the grass blade offers an alternative approach that can represent natural movement in all directions within a three-dimensional space while reducing the computational complexity involved in deforming the blades. Please refer to the accompanying video clip (Supplementary Materials) for more details.

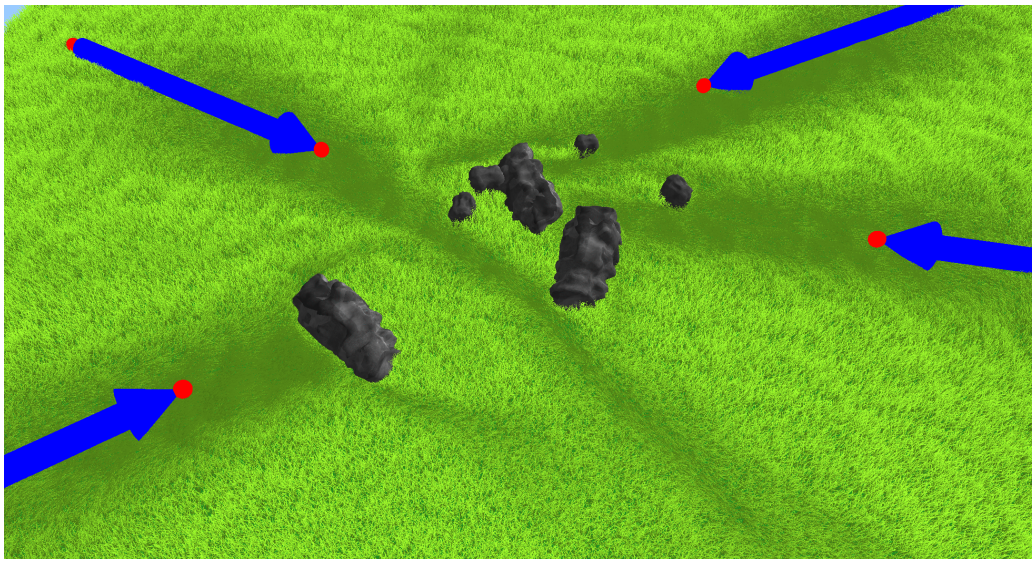


Figure 1. Complex wind dynamics created by the proposed method and its interaction with grasses. The blue arrows splat the wind and can be moved through the red colored control point.

Another point that makes our approach different from all the other work is the wind force authoring technique. Our method includes the ability to control the flow of the wind in a way that designers intend. All previous work [8,12,13,15–18] did not address the problem of wind authoring. For comparison, ref. [6] provides only a one-way wind generator. However, in our proposed method, the designer can place and modify the wind flow directly in the environment with the AGC interface. The designer can also adjust the strength of the wind and the area affected by the wind. To put a wind force, the AGC interface allows users to put a starting point and an arrow guideline in front and behind the starting point. It is also possible for multiple arrows to be branched out from a single

starting point, showing that various wind dynamics can be designed according to the designer's intent.

3. Proposed Algorithms

The CWD-Sim method describes a computationally efficient technique to realistically simulate the sway of the grass by the wind. It involves grouping grass blades into a two-dimensional grid, simplifying the forces affecting the grass, on the vertex shaders to deform the grass model, and allowing the designer to control the flow of wind using arrow guides. We are going to explain all steps in detail in the following sections.

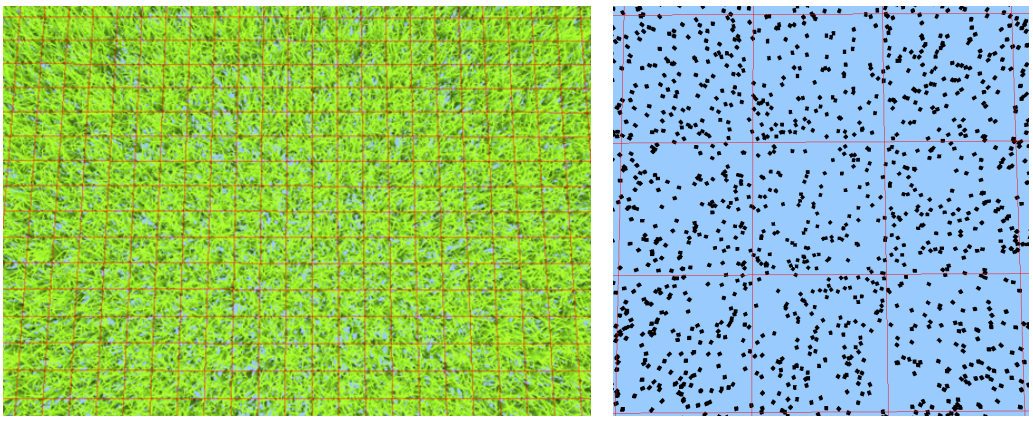
3.1. Grouping of Grasses

Performing individual fluid simulation calculations for every grass blade increases the computational load. It blocks the real-time performance required for interactive applications. To solve this problem, the grass blades are grouped and assigned to a grid structure. To do so, the world positions of the blade groups are converted to a group index. The group index, $G \in \mathbb{Z}$, is calculated in Equation (1).

$$G = \left(\frac{P_x}{w} + 0.5, \frac{P_z}{h} + 0.5 \right) \tag{1}$$

where $G \in \mathbb{R}^2$, w is the width of the grid, h is the height of the grid, P_x and P_z are the x and z world coordinates of the blade.

This equation divides the whole world into a 2D grid with a fixed cell size. Each cell contains a group of grass blades within its range. The grid, which has a 1000×1000 resolution in our case, is used for fluid simulation of wind dynamics. However, this grid resolution can be reduced to obtain faster simulation speeds. Our experiments indicate that reducing it to 200×200 would not make a big difference in visual quality. The 1000×1000 grid size means that there would be a total of 1,000,000 groups of grass blades. Using the instance ID, which is the ID number of the instance when we use the GPU Instancing technique [19], we can calculate the appropriate grid position for each grass blade based on its world coordinates and then assign it to the appropriate group. Once we determine the cells of all blade groups, we can make all blades in a group receive the same force instead of applying a different force to each individual blade. This approach greatly reduces the computational load because all blades within a group receive the same force. However, the visual quality does not decrease because there are so many grasses with different sizes and orientations. Figure 2 represents the 2D grid structure and the positions where the grass blades are placed. Note that the grass blades are randomly distributed on the cell.



(**a**) (**b**)

Figure 2. (**a**): Visualization of the 2D grid. (**b**): Grass blades represented as black points in the (**a**) cell.

3.2. Wind Force Modeling

Simulating wind on a computer is commonly achieved using the Navier–Stokes equations. These can be effectively solved through computational fluid dynamics methods, as detailed in [1]. The wind force in our simulation is modeled by a real-time fluid simulation algorithm grounded in the theory of Stable Fluid introduced by Jos Stam in [1,3]. In this section, we will briefly summarize the basic fluid simulation algorithms. This algorithm provides a stable numerical solution to solve the Navier–Stokes equation, which is denoted in Equation (2).

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u}\cdot\nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \nu\nabla^2\mathbf{u} + \mathbf{F} \tag{2}$$

$$\nabla\cdot\mathbf{u} = 0 \tag{3}$$

where ∂ is partial derivative, $\mathbf{u}$ is fluid velocity, t is time, ∇ is gradient operator, ν is the kinematic viscosity, ∇^2 is the Laplacian operator quantifying the diffusion, p is pressure, $\frac{\partial \mathbf{u}}{\partial t}$ is the local or temporal acceleration, reflecting the changes in velocity at a specific point over time, and the term $(\mathbf{u}\cdot\nabla)\mathbf{u}$ is the convective acceleration that represents the transport of momentum by the fluid. The term $\nu\nabla^2\mathbf{u}$ represents the viscous diffusion of momentum. The term $-\nabla p$ represents the pressure gradient, which is responsible for driving or opposing fluid motion. Finally, $\mathbf{F}$ represents any external forces acting on the fluid, such as the wind. Most air movement in the atmosphere is considered incompressible, and Equation (3) embodies the assumption of incompressibility for the fluid. Our implementation is based on the procedures proposed by Dobryakov et al. [3]. The procedures consist of multiple steps given a 2D grid to obtain the velocity grid V, where $\mathrm{V}_{i,j} \in \mathbb{R}^2$ is a cell in the ith row and the jth column. To obtain the final updated velocity grid V''', the algorithm performs the following processes from (4) to (9) in order. First, we calculate the curl of the velocity field as shown in Equation (4) that provides a quantification of the rotation at each point.

$$\mathrm{C}_{i,j} = \mathrm{V}_{i+1,j} - \mathrm{V}_{i-1,j} + \mathrm{V}_{i,j+1} - \mathrm{V}_{i,j-1} \tag{4}$$

where $\mathrm{C}_{i,j}$ is a 2D curl value at the ith row and jth cell of the grid. The subtraction term, $\mathrm{V}_{i+1,j} - \mathrm{V}_{i-1,j}$, approximates the median difference for the derivative of the velocity. The term $\mathrm{V}_{i+1,j}$ represents a single step speed to the right cell from the current position and $\mathrm{V}_{i-1,j}$ represents a single step speed for the left cell. Also, $\mathrm{V}_{i,j+1} - \mathrm{V}_{i,j-1}$ indicates the vertical speed. The calculation of these two directions gives a rotation measurement at (i, j) points. Next, we apply the vorticity confinement as described in Equation (5). This process helps to improve the smaller swirls that are noticeable in the fluid flow.

$$\begin{gathered} \mathrm{f}_{i,j} = \left(\mathrm{C}_{i,j+1} - \mathrm{C}_{i,j-1}, \mathrm{C}_{i+1,j} - \mathrm{C}_{i-1,j}\right)\cdot\lambda \\ \mathrm{V}'_{i,j} = \mathrm{V}_{i,j} + \mathrm{f}_{i,j}\cdot\Delta t \end{gathered} \tag{5}$$

where $\mathrm{V}'_{i,j}$ is the first updated velocity, $\mathrm{f}_{i,j} \in \mathbb{R}^2$ is the force at (i, j), Δt is the time step and λ is the vorticity confinement factor. The divergence of the velocity field is then computed as in Equation (6) in the next step. In fluid dynamics, this calculation gauges the rate at which the density leaves a specific region of space.

$$\mathrm{D}_{i,j} = \left(\mathrm{V}'_{i,j+1} - \mathrm{V}'_{i,j-1} + \mathrm{V}'_{i+1,j} - \mathrm{V}'_{i-1,j}\right)/2 \tag{6}$$

where $\mathrm{D}_{i,j} \in \mathbb{R}^2$ is the divergence value. This step is followed by the projection of the pressure, which is described in Equation (7). This step eliminates the component of the velocity that does not contribute to the advection along the vector field, leaving only the divergence-free component.

$$\mathrm{P}_{i,j} = \left(\mathrm{P}_{i,j+1} + \mathrm{P}_{i,j-1} + \mathrm{P}_{i+1,j} - \mathrm{P}_{i-1,j} - \mathrm{D}_{i,j}\right)/4 \tag{7}$$

where $P_{i,j} \in \mathbb{R}^2$ is the pressure and $D_{i,j}$ is the divergence at the $g_{i,j}$. Next, the pressure gradient is subtracted from the velocity field as indicated in Equation (8). This step ensures the conservation of mass within our fluid system.

$$V''_{i,j} = V'_{i,j} - \left(P_{i+1,j} - P_{i-1,j}, P_{i,j+1} - P_{i,j-1}\right) \tag{8}$$

where $V''_{i,j}$ is the second updated velocity and $V'_{i,j}$ the first updated velocity obtained in Equation (5). In the final step, the velocity field is then advected along itself. This stage creates the illusion of motion and fluidity, which is a critical aspect of fluid dynamics visualization. Let us say that the 2D coordinates of cell is $\alpha = (i, j)$. Then, the updated coordinate α' is first calculated from the second updated velocity and the grid size s. Note that the grid has a square shape where the width and height are equal to s.

$$\alpha' = \alpha - V''_{i,j} \cdot s \cdot \Delta t \tag{9}$$

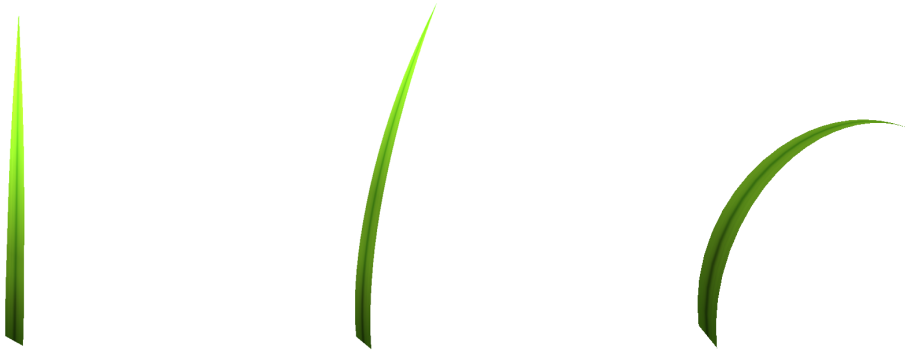
Once the advection is complete, the final velocity $V'''_{i,j}$ is obtained through Equation (10).

$$V'''_{i,j} = V''_{\alpha'} / (1.0 + \lambda \cdot \Delta t) \tag{10}$$

The calculated V''' in Equation (9) is used to model the deformation of the grass group. Each blade in a grass group calculates the deformation vector with Equation (12) based on V''' in the next Section 3.3.

3.3. Deformation of the Grass Model

From real-world observations of grass swaying in the wind, we propose a basic grass deformation model. It replicates grass dynamics through a blend of the two most significant grass motions, as shown in Figure 3. Bending is due to the influence of gravity, and the swaying of the grass is due to the wind force.



(**a**) (**b**) (**c**)

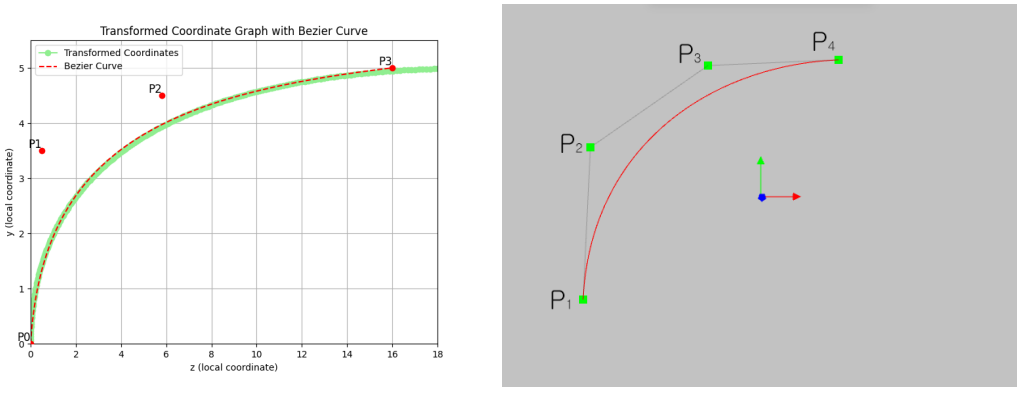
Figure 3. Shows the detailed bending effect of a grass blade due to the wind force. (**a**): Default state. (**b**): Only gravity. (**c**): Gravity with external wind force.

The deformation of the grass is carried out in the vertex shader. Initially, before the wind force is applied, the only force that acts on the grass is gravity. This force consistently bends the blade downward, and the amount of bending depends on the weight of the blade in the absence of wind force. This process is divided into gravity deformation and external force deformation. In the first step, we apply an initial deformation based on the elevation value $P_y \in \mathbb{R}$ of the position of the vertex. This step modifies the original position of the vertex $P \in \mathbb{R}^3$ to a new position P', as shown in Figure 4. The second step converts the external force into a translation vector using a quadratic equation, as shown in Figure 5.

This calculation of a quadratic equation eliminates the computational overhead of using a Bezier curve in [6] and provides a similar translation result.

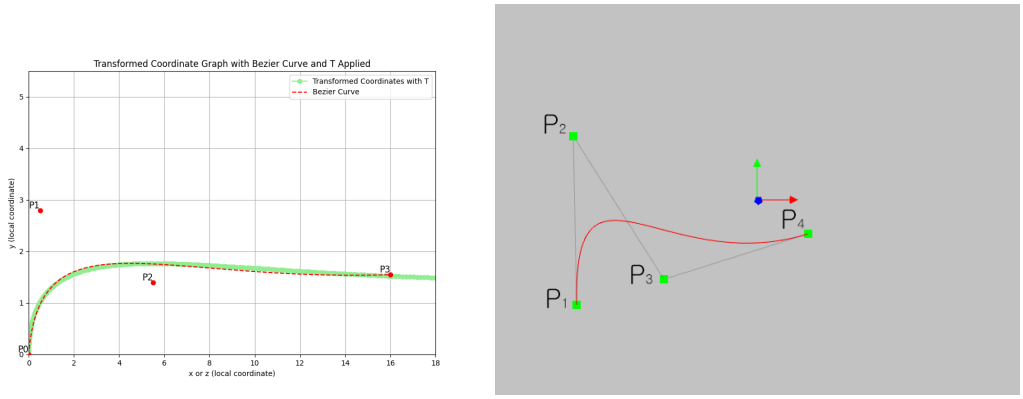
$$P' = \left(P_x, P_y - k_1 \cdot (P_y)^2, P_z + k_2 \cdot (P_y)^2\right) \tag{11}$$

where k_1 and k_2 are parameters to control the shape of the curve. For comparison, Figure 4a,b show an example of bending of a grass blade. Figure 4a is the result when we apply our simple quadratic equation, whereas Figure 4b shows the case when we apply the Bezier curve. For comparison, we put two graphs together to check the similarity for both Figures 4a and 5a where the dotted curves are the Bezier curves and the green curves are our proposed methods. We also show the red dots for control points for the Bezier curves. As we can see from the picture, the bending result is quite similar for both cases, although our equation needs fewer computations. We also add numerical comparisons in Table 1.



(**a**) (**b**)

Figure 4. Comparison of grass's default state due to gravity. (**a**): Proposed deformation Equation (11) is shown as a green line, the Bezier curve is shown as a red dotted line superimposed on our equation. (**b**): Bezier curve equation $\left(P = (1-t)^3 P_1 + 3(1-t)^2 t P_2 + 3(1-t)\, t^2 P_3 + t^3 P_4, 0 \le t \le 1\right)$ proposed in [20].



(**a**) (**b**)

Figure 5. Comparison of grass's swaying state due to external force. (**a**): Proposed deformation Equations (11) and (12) applied are shown as a green line, the Bezier curve is shown as a red dotted line superimposed on our equation. (**b**): Bezier curve equation $\left(P = (1-t)^3 P_1 + 3(1-t)^2 t P_2 + 3(1-t) t^2 P_3 + t^3 P_4, 0 \le t \le 1\right)$ proposed in [20].

Table 1. Comparative analysis of algorithmic efficiency in processing vertex points.

# of Vertex Points	Computation Time of Equation (11) (ms)	Computation Time of Bezier Curve (ms)
1000	1.9	6.8
5000	5.9	37.9
10,000	13.0	75.8

Table 1 shows the evaluation of up to 10,000 virtual vertex points. Our proposed algorithm (11) shows a speed faster than that of using the Bezier curve in terms of computation times, which is approximately 82.8% faster, with a time savings of 62.8 ms. This efficiency difference is quite important when we are dealing with a large set of vertex points such as grasses because it underscores the impact of computational complexity on processing speed and therefore highlights the importance of choosing the right algorithm for time-sensitive computational tasks.

In the second step of our process, we take into account the impact of the wind force on the grass blades. We calculate the wind translation vector T from the wind direction vector W and its magnitude F. This vector T essentially quantifies how the wind force should alter the position of the grass blades. The elevation value of the deformed vertex P'_y is again used to calculate the wind translation. Specifically, we calculate T, which encapsulates both the direction vector of the wind W and its magnitude F. The height of the deformed vertex, which we refer to as P'_y, plays a critical role in this calculation. The effect of the wind changes depending on the height of the blade, and this is captured in the height value. For example, the wind may have a stronger impact on the top of the blade than on the lower base part. Therefore, we use P'_y to adjust the strength of the wind translation vector T. Equation (12) describes how these computations are performed.

$$T = F \cdot \left(V'''_x \left(P'_y\right)^2, -\left|V'''\right| \left(P'_y\right)^2, -V'''_y \left(P'_y\right)^2 \right) \tag{12}$$

Figures 4 and 5 show another comparison between our equation proposed in (12) and the Bezier curve. As we can see, these two curves are almost identical, which proves that our equation can be used to bend the grass blade influenced by wind force. The final step involves updating the vertex positions by applying the wind translation T to the initial deformed positions P'. Transformation of the positions of the vertex positions is facilitated by the model matrix M. As shown in Equation (13), the final position of the vertex, P'', is calculated.

$$P'' = M\left((1-\lambda)T + \lambda P'\right) \tag{13}$$

where λ is the weighting parameter. The λ is a weighting parameter that represents the degree of effect that wind translation T and initial deformation P' have on the final position P''. When λ is closer to 0, the wind translation T has more influence on the final position, and when λ is closer to 1, the initial deformation P' has more influence.

3.4. Shadows between Grasses

Without the shadows, realism is greatly reduced, and blade interaction is difficult to perceive. However, calculating the shadows between all blades of grass can be computationally expensive. In particular, if we use a conventional method such as shadow mapping, which requires multi-pass rendering, it would not be effective to generate the map considering a large number of geometry data to render.

To solve this problem, we propose a simplified self-shadow calculation technique, as shown in Figure 6. We use a simplified equation to handle the shadows between all the grass blades. When a blade is in shadow, its color becomes dark. The brightness of the grass is adjusted based on the highest height of every group of grasses. The vertex of the highest position has the lightest color, while the color becomes dimmer as it goes down. This principle is based on the fact that when a blade of grass is pushed downward, it has a

high chance of being obscured by other blades of grass. Equation (14) represents the color adjustment formula. Figure 3 shows the detailed bending effect of a grass blade due to the wind force. Note that the x axis is the x or z offset from the local origin, while the y axis indicates the y offset from the origin, which shows the amount of bending. The original upright grass blade is also shown for comparison. As we can see in the figure, there were no unnatural artifacts on the mesh. As shown in Figure 7, the difference in naturalness with and without shadows is significant.

$$c_f = c_t \cdot \max(m_{min}, \min(P_y'' - |F| \cdot c_1 + c_2, m_{max})) \tag{14}$$

where $c_f \in \mathbb{R}^4$ is the color of a vertex, $c_t \in \mathbb{R}^3$ is a diffuse color, m_{min} and m_{max} are the darkest and brightest values, c_1 and c_2 are control parameters and p_y'' is the height of the blade. Through experimentation, we believe that this approach is sufficient for grasses in a large meadow where a large number of homogeneous grasses are packed. We have shown the comparison results in Section 4.

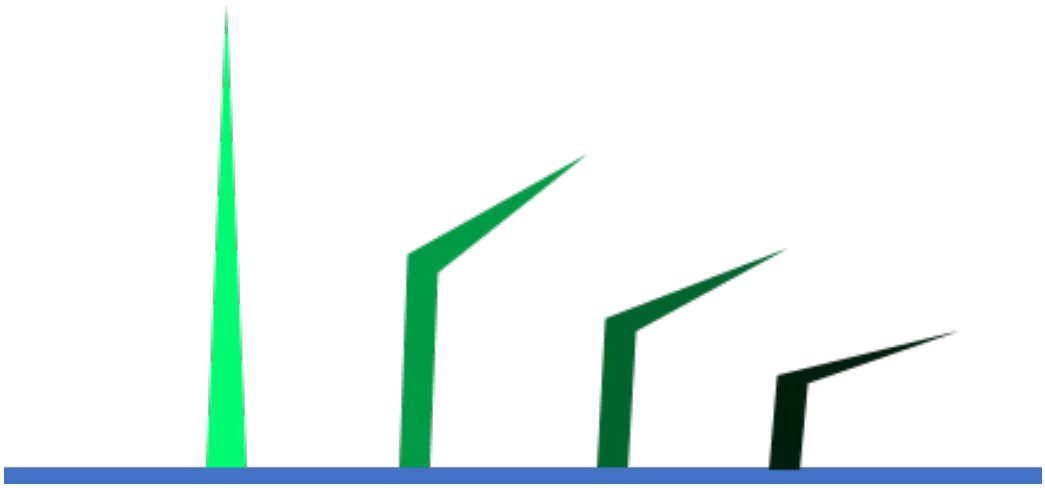
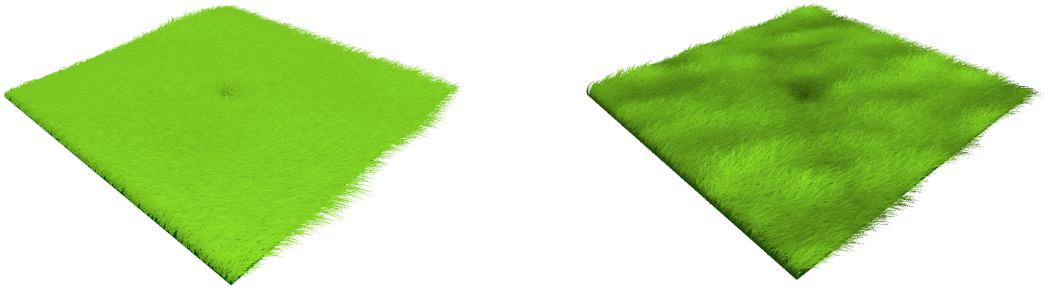


Figure 6. As the bending of the blade goes deeper due to the wind force, vertex colors become darker.



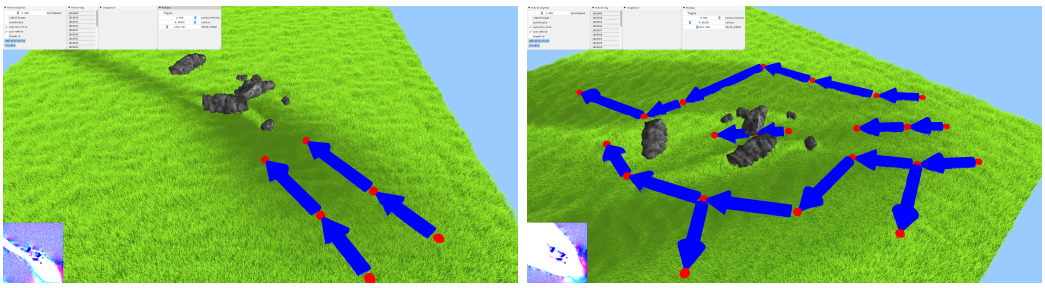
(**a**) (**b**)

Figure 7. (**a**): Without the shadow between grasses. (**b**): After applying the proposed shadow generation technique to grasses.

3.5. Arrow-Guided Wind Flow Control

One of the problems with using fluid for wind dynamics is how we can specify the wind the way the designer wants. Our algorithm gives designers the ability to control the wind flow in a scene using the so-called AGC (Arrow-Guided wind flow Control) interface. These arrow guides consist of a root point and multiple ending points, which can be added or removed as needed. The root point acts as the starting point for the wind flow. Clicking the points also opens the inspector window. In this window, the force strength can be adjusted by changing sliders or by entering a number. Setting an end point determines the

direction of the flow from the root point, which automatically changes to an arrow. Because all points can be added or removed directly anywhere in the environment, the designer has complete control over editing the wind forces, as shown in Figure 8.



(**a**) (**b**)

Figure 8. Starting with the state of (**a**) and adding as shown in (**b**) using the controllable arrow guide wind editing tool.

One of advantages of our proposed AGC interface is that multiple arrows can be connected to build more complicated wind dynamics. Thus, the wind flow can be a simple line or can be designed to resemble a tree structure or other complex patterns. By changing the position and length of the arrows, designers can adjust the direction of the wind flow. Once the design is complete, the wind forces are generated from the root to the end point along the series of arrows. Each point, which is the end point of the arrow, applies a force to the fluid simulation in the direction of the arrow from the start point. In the case of a tree structure, the forces are applied in a sequence based on the direction of the arrow's flow to make it appear continuous.

4. Experiments

To verify our algorithms, we built a system and performed a set of experiments. Hardware specifications include an E3-1230 v2 CPU and GTX 660 2GB GPU. For 3D rendering, we used the OpenGL and GLSL version 4.5. The grass model that we used in the experiments was in Autodesk's FBX format. Please see the accompanying video clip that we submitted (Supplementary Materials) and the Youtube video (https://youtu.be/uV0CFSqszJE (accessed on 5 January 2024)).

For fluid simulation, we used a 2D texture grid size of 1000×1000 to simulate fluid dynamics, applying Equations (4)–(10). In Equation (5), we set the vorticity confinement factor λ to 50. Regarding grass deformation, in Equation (11), we set the deformation parameters k_1 to 0.05 and k_2 to 0.1. These values were used to control the initial shape of the grass, which represented the weight of a grass blade due to gravity. Furthermore, in Equation (13), we set 0.2 for λ to control the flexibility of the grass blade under external force.

In the first experiment, we checked the performance of our algorithm. As we increase the number of grass blades, we checked its fps. Note that all computations and rendering are performed on the GPU side. The result is shown in Figure 9. As we can see in the figure, our algorithm maintained the real-time performance even if we increased the number of grasses up to 1,200,000. For comparison with other algorithms, we picked [6], which we believe to be one of the complete solutions for grass rendering and animation. Figure 9 shows the performance comparison between our algorithm and [6]. Note that the narrow blue and orange bands represent the trends of the graph. For this test, we used the same GPU to obtain an unbiased result. From this test, we knew that our algorithm did not significantly reduce performance as we increase the number of grasses. On the contrary, the algorithm proposed in [6] had a substantial decrease in fps. It turned out that our simulation can achieve speeds $10\times$ to $50\times$ faster than [6] in a similar hardware environment.

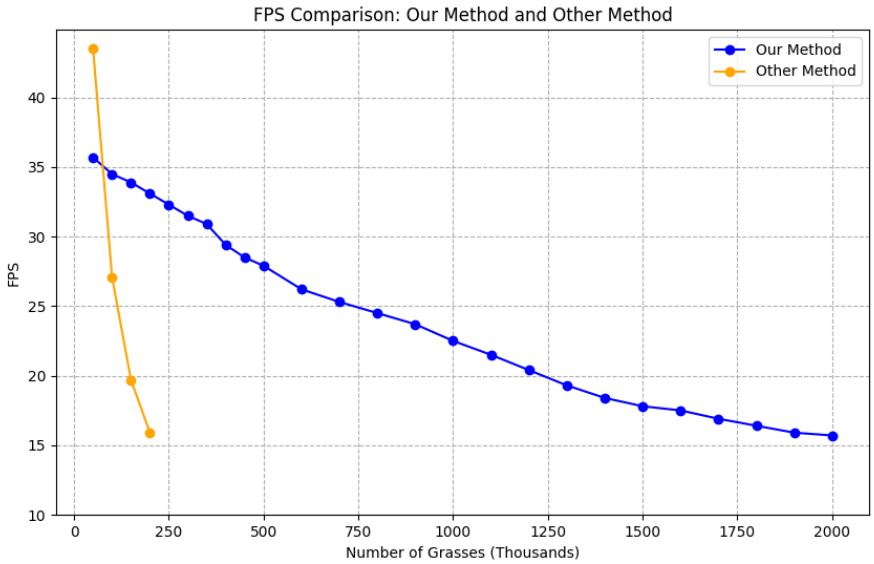
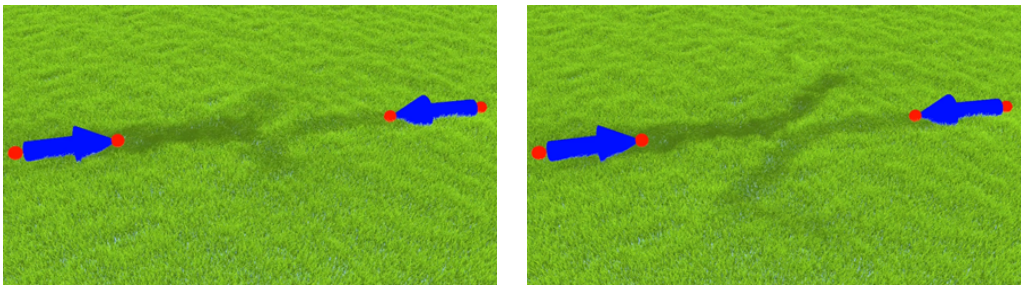


Figure 9. Performance comparison between our algorithms and the method proposed in [6].

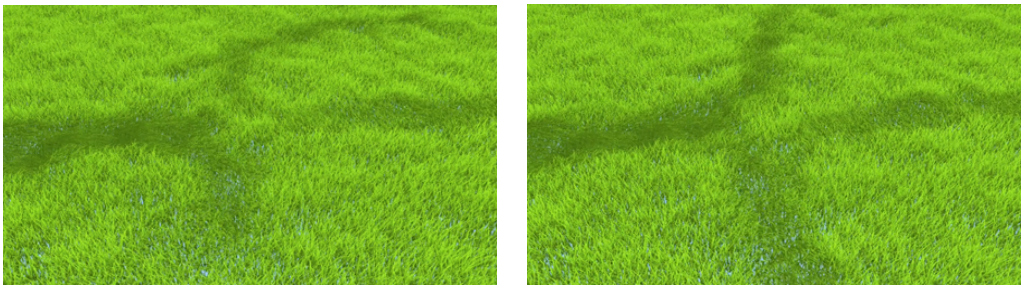
In the second experiment, we tested how efficient our algorithms are in designing complicated wind dynamics. Figure 1 shows the case where winds coming from multiple sources must interact with static obstacles. Our method could generate a realistic bump and churn in a very realistic way between wind and obstacles. Figure 10 shows two winds colliding in the middle of the environment. You can see that the two winds are deflecting and changing direction smoothly as shown in Figure 11. Please refer to the accompanying video of the result for more details. Figure 7 compared two cases in which we applied the shadow generation technique proposed in Section 3.5 and not. We can easily tell that shadowing between grasses improves visual quality. Finally, Figure 8 shows the wind-editing process with the proposed AGC interface. Root points and end points are added directly to the environment to form the arrow guides, and those guides are connected to each other to create complicate tree-like wind forces, which improves controllability.



(**a**) (**b**)

Figure 10. The two winds interact in the middle and then turn from the other direction (**a**) to (**b**).

The data in Table 2 present additional performance metrics obtained using an Intel Core i7-10700KF CPU and an NVIDIA RTX 2080 8 GB GPU. The simulations were conducted with a varying number of grass blades, up to a maximum of 7,000,000, to evaluate real-time performance. The optimal frame rate achieved under these conditions was 29 fps. The grid size for wind simulation was 1000×1000. The whole simulation time includes the processes time described in Equations (11)–(13). The time for the grass shadow indicates the performance of the shading algorithm, as illustrated in Figure 7b. The grass rendering time includes both the grass simulation and shadow rendering step.



(**a**) (**b**)

Figure 11. Two winds are changing direction over time after bending. (**a**) has been changed to (**b**).

Table 2. Performance metrics of grass simulation.

Grass Count	Wind Simulation (ms)	Grass Simulation (ms)	Grass Shadow (ms)	Grass Rendering (ms)	FPS
1,000,000	5.9	0.1	0.1	3.3	87
2,000,000	5.9	0.1	0.1	7.5	69
3,000,000	5.9	0.1	0.1	11.4	51
4,000,000	5.9	0.3	0.2	15.6	42
5,000,000	5.9	0.6	0.4	19.4	36
6,000,000	5.9	0.7	0.5	23.2	32
7,000,000	5.9	0.7	0.5	27.4	29

5. Conclusions

In this paper, we presented CWD-Sim, a real-time simulation algorithm for grass deformation and wind dynamic control in complex scenes. Our algorithm is capable of naturally simulating the effects of wind on grasses while allowing designers to have control over the wind flow in complex scenes with obstacles or other structures. By grouping grass blades and simplifying the force calculation, our algorithm significantly reduces computational load and achieves faster and more efficient simulations. Our method also allows for grass-model variation and efficient shadowing, which further enhances the realism of the simulation.

However, we acknowledge some limitations of our method. While our algorithm is well suited for animating large numbers of homogeneous grass blades, it focuses on the aggregate behaviors, such as wind-induced swaying, and therefore may not be appropriate for real-world physics-based animation, which would require a physics-based simulation technique. Another drawback of our method is 2D wind dynamics. Our proposed grass deformation is based on a 2D fluid simulation. Therefore, it is impossible to reproduce certain 3D fluid behaviors, such as the three-dimensional vortex observed in the real world. However, we believe that the 3D deformation can be approximated with the 2D simulation with simple quadratic equations that we proposed.

Also, our method did not take into account collisions between grass blades. To solve this problem, a more complex calculation method is needed. If our quadratic equation is to reflect the deformation of the adjacent grass blades, the collision information can be extracted and used. We will need to discuss this further in the future to incorporate the collision of many grasses into our processing simulations.

According to experiments, our methods appeared a little slower than certain prior methods such as [6] in performance, which had 43.5 fps for 50,000 grass blades compared to our 35 fps. However, our method did not downgrade much in performance as the number of blades increased. For example, while the [6] drops to 15.9 fps at 200,000 blades, our method maintains a frame rate of 28 fps even with 500,000 blades as shown in Figure 9, showing its advantage in large-scale simulations.

Additionally, we have also conducted experiments on the latest hardware specification and can see that it shows excellent real-time performance at 29 fps at 7,000,000 of grass count as shown in Table 2.

In future research, we would like to incorporate level of detail (LOD) and culling techniques for optimization and complement them with different types of models, such as flowers, and different types of grasses.

In the course of our current experiments, we have encountered a challenge in simulating the effects of strong winds on grass blades. We found that too much wind can cause grass blades to become too dark and flat. Although allowing the user to adjust the wind strength could potentially mitigate this problem, it could also lead to tedious control by the user. An alternative approach was considered instead, such as limiting the maximum wind strength, but this may cause the grass blades to appear unnaturally rigid. We also carried out an experiment with interpolation methods to smoothly limit the wind intensity, but this did not effectively solve the problem in the cases of very strong winds. Furthermore, our attempts to use periodic functions such as cosine and sine to maintain constant motion in grass blades were not successful, either. Identifying and solving this problem represents a significant opportunity for future research, as it is critical to achieving more realistic and dynamic simulations of natural environments.

Supplementary Materials: The following supporting information can be downloaded at: https://www.mdpi.com/article/10.3390/app14020548/s1.

Author Contributions: Conceptualization and methodology, N.C. and M.S.; software, N.C.; validation, N.C. and M.S.; formal analysis, N.C. and M.S.; investigation, N.C.; resources, N.C. and M.S.; data curation, N.C.; writing—original draft preparation, N.C. and M.S.; writing—review and editing, N.C. and M.S.; visualization, N.C.; supervision, M.S.; project administration, M.S. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2021R1A2C1012316) and was supported 2023 Cultural Heritage Smart Preservation & Utilization R&D Program by Cultural Heritage Administration, National Research Institute of Cultural Heritage (Project Name: A smart H-BIM modeling technology of wooden architecture for the conservation of Historical and Cultural Environment, Project Number: 2023A02P01-001, Contribution Rate: 50%).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data is contained within the article or Supplementary Materials.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Stam, J. Stable fluids. In Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, Los Angeles, CA, USA, 8–13 August 1999; pp. 121–128.
2. Harris, M.J. Fast Fluid Dynamics Simulation on the GPU. GPU Gems. 2005; Chapter 38. Available online: https://developer.nvidia.com/sites/all/modules/custom/gpugems/books/GPUGems/gpugems_ch38.html (accessed on 12 April 2023).
3. Dobryakov, P. WebGL Fluid Simulation. Available online: https://github.com/PavelDoGreat/WebGL-Fluid-Simulation (accessed on 12 April 2023).
4. haxiomic. Cross-Platform GPU Fluid Simulation. Available online: https://github.com/haxiomic/GPU-Fluid-Experiments (accessed on 12 April 2023).
5. angeluriot. 2D Fluid Simulation. Available online: https://github.com/angeluriot/2D_fluid_simulation (accessed on 12 April 2023).
6. Lo, Y.; Chu, H.K.; Lee, R.R.; Chang, C.F. A simulation on grass swaying with dynamic wind force. In Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, Redmond, DC, USA, 27–28 February 2016; p. 181.
7. Boulanger, K.; Pattanaik, S.N.; Bouatouch, K. Rendering Grass in Real Time with Dynamic Lighting. *IEEE Comput. Graph. Appl.* **2009**, *29*, 32–41. [CrossRef] [PubMed]
8. Deussen, O.; Hanrahan, P.; Lintermann, B.; Měch, R.; Pharr, M.; Prusinkiewicz, P. Realistic modeling and rendering of plant ecosystems. In Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, Orlando, FL, USA, 19–24 July 1998; pp. 275–286.
9. Habel, R. Real-Time Rendering and Animation of Vegetation. Ph.D. Thesis, Technischen Universität Wien, Vienna, Austria, 2010.

10. Chen, K.; Johan, H. Animating 3D vegetation in real-time using a 2D approach. In Proceedings of the 19th Symposium on Interactive 3D Graphics and Games, San Francisco, CA, USA, 27 February–1 March 2015; pp. 69–76.
11. Qiu, H.; Chen, L. Rendering System for Large-Scale Grass. In Proceedings of the 2009 International Conference on Computational Intelligence and Software Engineering, Wuhan, China, 11–13 December 2009; pp. 1–4. [CrossRef]
12. Max, N.; Saito, S.; Watanabe, K.; Nakajima, M. Rendering grass blowing in the wind with global illumination. *Tsinghua Sci. Technol.* **2010**, *15*, 133–137. [CrossRef]
13. Fan, Z.; Li, H.; Hillesland, K.; Sheng, B. Simulation and Rendering for Millions of Grass Blades. In Proceedings of the 19th Symposium on Interactive 3D Graphics and Games, i3D '15, San Francisco, CA, USA, 27 February–1 March 2015; pp. 55–60. [CrossRef]
14. Wang, S.; Ali, S.G.; Lu, P.; Li, Z.; Yang, P.; Sheng, B.; Mao, L. GPU-based Grass Simulation with Accurate Blade Reconstruction. In Proceedings of the Advances in Computer Graphics: 37th Computer Graphics International Conference, CGI 2020, Geneva, Switzerland, 20–23 October 2020; pp. 288–300.
15. Jahrmann, K.; Wimmer, M. Interactive Grass Rendering Using Real-Time Tessellation. In *WSCG 2013 Full Paper Proceedings*; TU Wien: Vienna, Austria, 2013.
16. Bakay, B.; Lalonde, P.; Heidrich, W. Real-Time Animated Grass. In *Eurographics (Short Presentations)*; TU Wien: Vienna, Austria, 2002.
17. Jens, O.; Salama, C.R.; Kolb, A. GPU-based responsive grass. *J. WSCG* **2009**, *17*, 65–72.
18. Belyaev, S.Y.; Laevsky, I.; Chukanov, V.V. Real-Time Animation, Collision and Rendering of Grassland. In Proceedings of the GraphiCon2011, Moscow, Russia, 26–30 September 2011.
19. JoeyDeVries. LearnOpenGL-Instancing. Available online: https://github.com/JoeyDeVries/LearnOpenGL/tree/master/src/4.advanced_opengl/10.1.instancing_quads (accessed on 12 April 2023).
20. Dobryakov, P. NURBS Demo-Evaluator for Non Uniform Rational B-Splines. Available online: http://nurbscalculator.in (accessed on 12 April 2023).