Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики

РЕНДЕРИНГ ТРАВИ У РЕАЛЬНОМУ ЧАСІ ТА ЇЇ ДИНАМІЧНА ВЗАЄМОДІЯ З ОБ'ЄКТАМИ

Текстова частина
магістерської роботи
за спеціальністю „Інженерія програмного забезпечення" 121

Керівник магістерської роботи
ст.в., к.т.н. Бучко О. А.
_____
(підпис)
"____" _____ 2020 р.

Виконав студент
Ярошепта О.П.
"____" _____ 2020 р.

Київ 2020

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
Зав. кафедри інформатики,
к.ф-м.н.
_____ С. С. Гороховський
(підпис)
7 листопада 2019 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на дипломну роботу

студенту 2 року ІПЗ факультету інформатики
Ярошепті Олександру Павловичу
Дослідити Рендеринг трави у реальному часі та її динамічна взаємодія з об'єктами

Зміст ТЧ до магістерської роботи:

Зміст
Анотація
Вступ
1 Генерація трави.
1.1 Огляд
1.2 Методи генерації трави
1.3 Поширення та шейдінг
2 Анімація трави.
2.1 Алгоритми симуляції
2.2 Симуляція вітру
3 Взаємодія об'єктів у віртуальному світі
3.1 Фізична поведінка
3.2 Гравітація
3.3 Взаємодія об'єктів
4 Методи оптимізації
5 Порівняння алгоритмів
Висновки
Список літератури

Дата видачі „25" жовтня 2019 р. Керівник _____
(підпис)
Завдання отримав _____
(підпис)

Тема:_____

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на дипломну роботу.	7.11.2019	
2.	Огляд технічної літератури за темою роботи.	01.12.2019	
3.	Виконати аналіз сучасних методів	25.01.2020	
3.	Пошук та вивчення готових реалізацій	07.02.2020	
4.	Підготовка реалізацій до тестування	01.03.2020	
5.	Виконання порівняльного аналізу результатів роботи алгоритмів з різними параметрами застосування.	30.03.2020	
6.	Написання пояснювальної записки	20.04.2020	
7.	Створення слайдів для доповіді та написання доповіді.	22.05.2020	
8.	Аналіз отриманих результатів з керівником, написання доповіді та попередній захист магістерської роботи.	02.06.2020	
9.	Корегування роботи за результатами попереднього захисту.	3.06.2020	
10.	Остаточне оформлення пояснювальної записки та слайдів.	5.06.2020	
11.	Захист магістерської роботи (проекту)	16.06.2020	

Студент: Ярошепта Олександр Павлович

Керівник старший викладач, к.т.н. Бучко Олена Андріївна

“_____”_____

Зміст

Анотація

В даній роботі були розглянуті технології рендеренгу трав’яного покрову, а також методи їхньої оптимізації. В якості методів рендерингу трави у віртуальному світі досліджено та проаналізовано способи описані Fan та Jahrmann.

Вступ

Висока видова різноманітність трави є важливою складовою рослинності світу. Траву можна знайти у будь-якій наземні екосистемі: тропічному лісі, широколистяному лісі помірної зони (один із найтиповіших видів екосистем в Україні), савані та навіть у пустелі. Таким чином незалежно від того, яким розробник хоче створити свій віртуальний світ, повстане питання рендеренгу зеленого покрову. В наш час є багато методів, які здатні швидко та якісно вирішити це завдання. Але задача рендеренгу трави залишається актуальною задачею через високу видову різноманітність та велику кількість геометричних елементів. Прикладом таких алгоритмів є реалізація трави у вигляді:

- однієї суцільної текстури,
- у вигляді набору простих 2D текстур, що з певною орієнтацією розміщені на земляній поверхні,
- у вигляді маленьких зубців, кожен з яких може мати свій власний розмір та форму.

Кожен з алгоритмів має свої переваги та недоліки, які потрібно враховувати під час створення віртуального світу. В залежності від того, який алгоритм був обраний, необхідно також врахувати фізичні явища та властивості, які відповідатимуть за взаємодію між об’єктами.

Розробник, мета якого зімітувати реальний світ, стикається з проблемами пов’язаними з балансом між оптимізацією, реалізмом та особливостями, які з ними пов’язані, але, зазвичай, досить важко визначити, що має більше значення в тій чи іншій ситуації. Саме тому важливо розуміти переваги та недоліки технології рендеренгу.

У наш час медіа інтерактивність набуває все більшого значення у повсякденному житті. Однією з основних складнощів створення таких додатків є те, що кожне представлене зображення має бути обчислено та намальовано в

режимі реального часу відповідно до дій користувача. Таким чином рендеринг в режимі реального часу часто потребує швидкості та наближення, особливо для візуалізації складних сцен. Цьому сприяє швидкий розвиток апаратних можливостей та графічних інтерфейсів програмування, який дозволяє досягти вищого ступеня фотореалізму під час рендерингу сцени в реальному часі. Сцени на відкритому повітрі зазвичай є складнішими порівняно зі сценами закритого простору через високу геометричну складність рослинності. Згідно з доповіддю Продовольчої та сільськогосподарської організації Організації Об'єднаних Націй про зелений покров, до сорока відсотків земельної площі Землі займають луки, включаючи пасовища та поля. Таким чином трава є на більшості відкритих місцевостей.

1. Генерація трави

1.1 Огляд

Трава є важливим елементом, який допомагає відображати сцени природи. Особливо у віртуальній реальності та комп'ютерних іграх природні сцени повинні включати належним чином відтворену рослинність. Для дерев і рослин вже існує багато рішень, але генерування трави вже давно є складною задачею через велику кількість необхідної геометрії. Ось чому найчастіше трава виводиться у вигляді текстури, намальованою на землі, або у вигляді прозорих білбордів (вертикальна 2D текстура). Обидва підходи мають різні небажані артефакти: трава як текстура має малі кути огляду, оскільки земля має статичну позицію, тоді як білборди мають проблеми, коли на них дивляться зверху - виглядають дуже плоскими.

Ще одним недоліком традиційних підходів до малювання трави є те, що вони не дають змоги реагувати на зміни в середовищі. Наприклад, на траву не впливають вітер чи інші сили. Це проблематично, оскільки інтерактивність вкрай необхідна для віртуального досвіду користувача. Елісон Мак-Махан [25] визначила три основні правила тривимірної віртуальної реальності для занурення глядача:

1. Очікування користувачів щодо гри чи оточення повинні відповідати умовам середовища.

2. Дії користувача повинні мати нетривіальний вплив на навколишнє середовище.

3. Конвенції світу повинні бути послідовними.

Аналізуючи ці правила, жодне з них насправді не залежить від фотореалізму візуального вигляду.

Моделювання, рендеринг та анімація трави є складними процесами. Усі три ці проблеми були визначеними Вільямом Рівзом та Рікі Блаумом[5]. Вони моделювали траву за допомогою стохастичного процесу, створюючи леза для візуалізації кіпами, і цим самим їм не довелося зберігати повну геометрію всієї

трави одразу. Вплив вітру було змодельовано методом процедурної анімації. Їхня система не працювала в режимі реального часу, але створені ними основні поняття все ще використовуються, включаючи системи реального часу. Однією з областей, яку Рівз та Блау не досліджували, є реакція трави на зіткнення з твердими предметами. Деякі інші роботи [26] розглядають взаємодію трави-об'єкта як процес, згідно з яким вона процедурно відсувається вбік. У деяких попередніх роботах є моделювання руху за допомогою пружинно-масової системи або хвильового моделювання.

1.2 Методи генерації трави

1.2.1 Картинкові методи

Найдавніші методи рендерингу трави використовують плоскі текстури, нанесені на землю. Навіть у наш час ці методи часто використовуються, незважаючи на те, що вони не реалістичні та не реагують на зміни в світі. Більш складний текстурний метод був запропонований Shah в [9]. Він використовує двонаправлену функцію текстури для анімування текстури трави та відображення зміщення силуетів. Таким чином він зміг досягти кращого вигляду, ніж проста текстура, витрачаючи на рендеринг більше часу.

Навіть в наш час майже кожен метод все ще використовує текстурну техніку для рендерингу трави, що знаходиться далеко від камери. Для трав'яного покрову, що знаходиться ближче до глядача, розроблені різні підходи: Orthmann в [8] представив техніку візуалізації трави, яка використовує два білборди, щоб зобразити купу трави і яка може взаємодіяти з її оточенням. Перевагою даного методу є ефективна візуалізація, яку можна виконати на графічній карті, в той час як недоліком є малі кути огляду, через які трава виглядає плоскою, якщо дивитися з більш високого кута. Також траві згенерованій таким способом не вистачає візуальної глибини. Для покращення результату Хабель в [12] використовує

напівпрозорі текстурні фрагменти, які розміщуються на звичайній сітці. Для вирішення проблеми пов’язаної з кутом зору, Нейрет в [11] звертається до існуючих методів візуалізації волосся за допомогою 3D-текстур і розширює їх для створення геометрії з високим рівнем деталізації для великих сцен, як трава. Цей метод дозволив йому виводити траву хорошої якості з різних кутів, яка все ще має штучний вигляд.

Zhao в [10] використовує геометричний шейдер для малювання кіп трави у вигляді геометричних фігур, але внаслідок слабкої продуктивності результат досить розріджений.

На відміну від методів, згаданих раніше, запропонована методика Wimmer’ом в [2] промальовує всі леза трави у вигляді згладженої геометрії, використовуючи апаратну тесселяцію. Крім того, кожну травинку можна окремо деформувати силами, такими як вітер або інші предмети. Особливістю цієї методики є те, що вона зберігає геометричні дані кожної травинки окремо. Це призводить до ідеально випадкового та рівномірного трав’янистого поля, яке має хороші показники продуктивності, так як до нього можна застосувати багаторівневу систему деталізації. Однак, оскільки кожне лезо зберігається окремо, даний підхід погано масштабується для великих сцен. Щоб вирішити цю проблему Wimmer розширив основну техніку, використовуючи інстанціювання: лише один патч із травою зберігається в пам'яті та малюється кілька разів, так, що анімовані сцени з травою практично довільних розмірів можуть бути відображені в режимі реального часу. Отриманий результат досить часто має повторювані візерунки, які зазвичай з'являються внаслідок копіювання, але оскільки кожне лезо генерується всередині шейдера, на його вигляд можна легко вплинути змінивши просторове освітлення, яке приховує переходи між латками.

1.2.3 Геометричні методи

Більш новий підхід, який малює кожну траву як геометричний об'єкт, запропонував Jahrmann в [2]. Автор використовує тесселяційний конвеєр OpenGL для моделювання травинок з динамічним рівнем деталізації безпосередньо на графічній карті. Одинарне лезо представлене тесилязованим квадом та альфа-текстурою, що формує його форму. Для великих полів одну латку трави малюють кілька разів, використовуючи екземплярну візуалізацію. Крім того, кожна травинка анімована статичним вітром. Колізії також наближені з використанням текстури, що вказує на перехід.

Рис. 1.1 Приклад роботи алгоритму описаний Jahrmann [2]

Фан та ін. [1] запропонували один із останніх підходів до генерації трави у 2015 році. Алгоритм схожий на роботу Jahrmann [2], але автори більше звернули свою увагу на взаємодію кожної травинки з тривимірними об'єктами. Моделювання відбувається на основі плиток, де кожна плитка, на якій розміщується об'єкт, протягом останніх двох секунд оновлюється. Після закінчення цього періоду травинки повертаються до свого початкового положення. Даний алгоритм здатний відображати та імітувати щільні поля трави в режимі реального часу, але продуктивність значною мірою залежить від кількості та розміру плиток.

Запропонований алгоритм Jahrmann в [3] базується на попередній роботі його та Wimmer [2], з якої було взято лише основні ідеї процесу рендерингу. Швидкі та точні методи обрізання, що застосовуються до кожної травинки, дозволяють замінити грубі наближення фізично обґрунтованими розрахунками. Для того, щоб відповідати правилам віртуального занурення, які описані в [26], Jahrmann розраховує всі чинники впливу для кожної травинки окремо. Крім того, форма леза визначається аналітичною функцією, яка дозволяє мати більш реалістичний та гетерогенний вигляд трав'яного поля. Перевагами даного алгоритму є можливість розташування трави на різних об'єктах, а не тільки на рівнинах та поверхнях, що базуються на картах висот, використання для кожної травинки окремої фізичної моделі, форма трави обраховується аналітичними функціями, а також використання динамічного рівня деталізації за допомогою апаратної тестелизації.

Рис. 1.2 Приклад роботи алгоритму описаний Фаном в [1]

1.2.2 Гібридні методи

Boulanger та ін. [4] представили гібридну техніку, яка використовує вищезазначені методи як різні рівні деталізації. Як геометричне зображення вони

використовують єдиний трав'яний патч, що складається з травинок, які формуються траєкторіями частинок під час етапу попередньої обробки. Для отримання великого трав'яного поля використовується екземплярна візуалізація з випадковою орієнтацією латки для кожного екземпляра аби зменшити кількість артефактів сітки, що надходять з екземпляру. Трава, яка знаходиться далі від камери, виводиться за допомогою техніки, що використовує 3D-текстури, у вигляді вертикальних та горизонтальних текстурних фрагментів. Для рендерингу найвіддаленішої трави виводиться лише один горизонтальний зріз, який формує текстуру відображену на землю за допомогою наявних різних методів візуалізації як різних етапів статичного рівня деталізації. Для відображення трави, що знаходиться поруч із глядачем, алгоритм використовує геометричні об'єкти, наближені до латки трави. На наступному рівні деталізації використовуються вертикальні та горизонтальні фрагменти текстури, які утворюють тривимірну сітку, яка схожа на двовимірну сітку Хабеля. Для подачі трави на більшу відстань враховується лише один горизонтальний фрагмент текстури, що відповідає простому відображенню текстури. Щоб подолати ступінчасті артефакти статичного рівня деталізації, автори інтерполювали між різними рівнями, щоб приховати межі. З одного боку, візуалізація в цій техніці дуже ефективна і має реалістичний вигляд. З іншого боку, представлення трави не анімоване і не взаємодіє зі своїм оточенням. На малюнку 1.3 показана візуалізація різних рівнів деталізації.

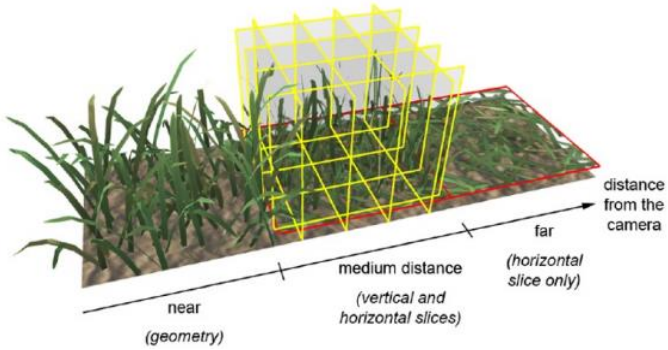


Рис. 1.3 Приклад роботи алгоритму описаний Boulanger в [4]

1.3 Поширення та шейдінг

В цій частині буде розглянуто різні способи поширення трав’яного покрову, а також методи шейдінгу в моделях, що були описані в попередньому підрозділі. Те як відбувається розповсюдження трави зазвичай залежить від обраного методу генерації.

Наприклад генерування трави як двовимірної текстури поверхні не вимагає ніяких додаткових складних обчислень від користувача, оскільки все, що відбувається це мапінг (накладання) текстури на поверхні землі. Цей процес проходить наступним чином: або одне зображення з травою “натягується” на поверхню, або ж використовується багато копій однієї текстури, які тайлами заповнюють необхідне місце. Є і більш складний спосіб, який генерує текструри і потім за допомогою тайлового мапінгу покриває ними необхідну поверхню. Таким чином можна досягти простого, але ефективного поширення трави. Недоліком даного методу є не інтерактивний і статичний результат, з яким мало що можна зробити.

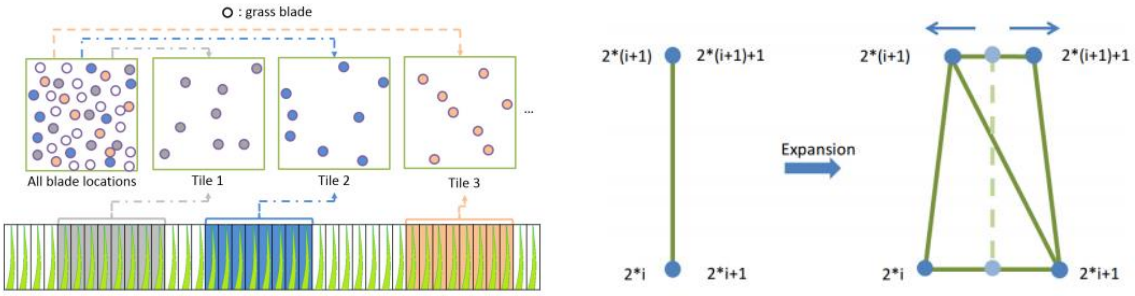
Wang в своїй роботі [7] рендерить все трав’яне поле, використовуючи геометрію однієї травинки, яку він відтворює кілька разів. Хоч лезо і моделюється декількома вершинами та скелетом для анімації, але для отримання хорошого результату для досить рідких трав’янистих полів необхідно додатково використовувати складні підходи для реалізації різнорівневої деталізації та відсікання.

Спосіб описаний Fan [1] дозволяє генерувати поля, які складаються з дуже великої кількості травинок. Щоб отримати геометричні дані трави, він адаптував процедурний метод моделювання описаний Boulanger та ін. в [14], шляхом попереднього генерування списку травинок. Спершу потрібно мануально створити кілька типів леза трави. Потім створюються екземпляри цих типів та випадковим

чином розсаджуються в прямокутну площу з випадковою орієнтацією та довжиною. Отриманий результат автор називає патчем. Його розмір становить 16384 травинок.

Після цього травинки промальовуються за допомогою модифікованої моделі шейдінга Фона. Для отримання реалістичного результату автори додають помірний ступінь ефекту розсіювання під поверхнею [15], змішуючи значення затінення задньої грані з часткою значення затінення передньої грані.

Після цього травинки промальовуються за допомогою модифікованої моделі шейдінга Фона. Для отримання реалістичного результату автори додають помірний ступінь ефекту розсіювання під поверхнею [15], змішуючи значення затінення задньої грані з часткою значення затінення передньої грані.



А) Б)

Рис. 1.4 на А) показано як виглядає попередньо згенерована латка трави; Б) показує процес формування травинки.

Після цього травинки промальовуються за допомогою модифікованої моделі шейдінгу Фона [27]. Для отримання реалістичного результату автори додають помірний ступінь ефекту розсіювання під поверхнею [15], змішуючи значення затінення задньої грані з часткою значення затінення передньої грані.

Генерація трави в алгоритмі запропонованому Jahrmann в [3] відбувається наступним чином: виходячи з представленої нижче ієрархії (див. схему 1.5),

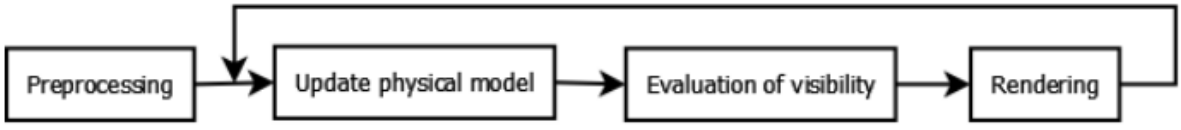


Рис. 1.5 Схема процесу генерації трави описана Jahrmann в [3]

алгоритм рендерингу трави розділено на два загальні етапи. Перший крок обчислює всі необхідні дані та готує структури даних. Цей крок розподіляє леза трави та генерує патчі з одиничних травинок. Даний крок виконується лише один раз на початку програми. Другий крок обробляється під час рендерингу кожного кадру. На цьому кроці необхідно виконати три завдання, які зазначені у списку нижче.

1. Визначається фізична модель визначається для кожної травинки. Сюди входить обчислення колізій зі складними об'єктами, впливу вітру та гравітації.
2. Перевіряється видимість кожної травинки, після чого частина травинок видаляється з малювального конвеєру. Рішення про те, чи відбивається лезо, ґрунтується на його напрямку, відстані до камери та оклюзії інших предметів. Цей крок є критичним для роботи цієї методики візуалізації.
3. Кожне видиме лезо трави рендериться як геометричний об’єкт. Форма травинки визначається аналітичною функцією. Динамічний рівень деталізації, що досягається апаратною тесселяцією (замощення паралелограмами) забезпечує гладкі краї форми.

На відміну від методів, згаданих раніше, запропонована методика Wimmer та Jahrmann в [2] рендерить усі леза густого трав’янистого поля повністю як гладку геометрію, використовуючи апаратну тесселяцію. Крім того, кожна травинка може бути окремо деформувана різними силами, такими як вітер або рух інших предметів предмети.

Особливістю цієї методики є те, що вона зберігає геометричні дані кожної травинки окремо. Це призводить до ідеально випадкового та рівномірного згенерованого трав’янистого поля та має хороші показники продуктивності, так як до нього можна застосувати різні способи реалізації багаторівневої деталізації. Однак, оскільки кожне лезо зберігається таким чином, даний метод має погану масштабованість для великих сцен.

Дана методика ґрунтується на трав'яному полі, яке формує межі простору, де трава малюється. Поле виступає контейнером для текстур та параметрів, визначених користувачем, і представляється як звичайна сітка, де кожна комірка сітки містить патч трави. Розмір кожної клітинки істотно впливає на продуктивність, але оптимальний розмір загалом не може бути визначений, оскільки він сильно залежить від загальної сцени та та інших чинників. Кожен патч трави зберігає масив вершин, який містить всі необхідні дані, щоб намалювати травинки, що ростуть на ньому. Крім того, йому призначається обмежувальний чотирикутник, щоб можна було застосувати обрізання по видовому фрустуму.

На кроці ініціалізації, опрацьовуються текстури та параметри, щоб згенерувати вхідні дані для процесу рендерингу.

- Щільність: вказує, скільки травинок потрібно створити на квадраті розміром 1x1.
- Ширина: мінімальна та максимальна ширина леза.
- Висота: мінімальна та максимальна висота леза.
- Фактор згинання: визначає наскільки лезо може згинатись.
- Фактор згладжування: визначає максимальний рівень тесселяції травинок.

Рендеринг трави здійснюється повністю на стороні відеокарти, яка повинна мати можливість виконувати шейдери тесселяції. Якщо патч трави видно у видовому фрактумі, малюються всі включені леза з відповідним рівнем деталізації. Оскільки кожна травинка - це просто двовимірний квадратик, для повного візуального вигляду потрібно вимкнути обрізання задньої поверхні.

Спочатку кожна травинка представлена прямокутником випадкового розміру, вирівняним в площині x-y, який потім тесселюється на підряди, використовуючи шейдер тесселяції. Отримані вершини вирівнюються на двох паралельних сплайнах, які утворені верхніми та нижніми вершинами вхідного

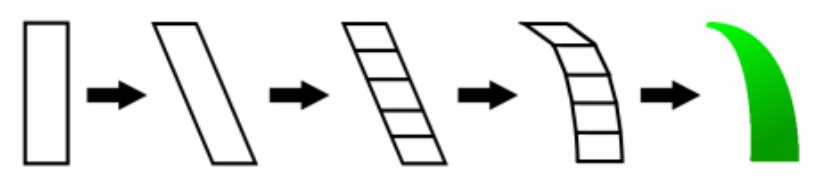


Рис. 1.7 Процес створення травинки, перший є результатом роботи вершинного шейдера, шейдер управління тесселяцією, шейдер обрахування тесселяції і останній - фрагментний шейдер

квадратика та двома додатковими контрольними точками, що визначають кривизну леза. Кінцева форма леза визначається альфа-текстурою. Процедура також проілюстрована на рисунку 1.3. Весь процес описується наступними кроками:

1. Під час стадії вершинного шейдера визначається орієнтація леза шляхом обертання навколо своєї осі леза з одним із випадкових значень в якості кута.
2. У шейдері управління тесселяцією обчислюється відстань між лезом і положенням камери.
3. Дві додаткові контрольні точки формуються в шейдері управління тесселяцією для визначення форми леза на наступному кроці.
4. У шейдері оцінки тесселяції отриманий на попередньому кроці чотирикутник формується шляхом вирівнювання вершин уздовж двох паралельних квадратичних сплайнів.
5. У фрагментальному шейдері формується остаточна форма леза шляхом застосування маски в якості альфа текстури.

Для того, щоб досягти більшої варіативності, компоненти кольору можна додатково міняти трьома випадковими значеннями. В реальному світі через тіньові ефекти трава зазвичай є темнішою біля землі та світліша на її кінчику. Для імітації цього ефекту можна помножити колір на вертикальну координату текстури і таким чином досягти кращого ефекту паралаксу.

2. Анімація трави

В цьому розділі ми розглянемо як можна реалізувати анімацію трави. Дослідимо способи симуляції вітру та гравітації, а також алгоритми, що описують взаємодію трави з іншими фізичними об’єктами. Велику увагу буде приділено методам описаними в [3], як одним з найновіших та найкращих.

2.1 Алгоритми симуляції

Для симуляції реалістичної ідилії природи крім дерев та кущів, води та неба, також необхідно мати високоякісні ефекти трави. В минулому цей процес вважався досить складним та ресурсовитратним, але бенчмарк (тести продуктивності) “Codecreatures Benchmark” опублікований в 2002 показав, що це не так.
Як було сказано раніше, ми визначили наступні вимоги до реалістичної симуляції трави:

1. Вона повинна реалістично реагувати на контакт, включаючи деформацію, на рівні окремих лез.
2. Мільйони травинок мають обчислюватись в інтерактивному режимі.
3. Симуляція повинна бути сумісною з процедурними методами анімації.

Для анімації дерев та кущів досить часто використовують систему куль для твердих тіл, яка використовує CPU для симуляції. Її перевагою є простий та ефективний спосіб обчислення взаємодій між об’єктами. Кожен об’єкт заповнюється віртуальними кулями, за допомогою яких проводяться всі обчислення.

Для генерації рослинних систем застосовують процедурні методи. Вітер впливає на всі травинки, і в більшості реалізацій - це процедурний метод, який є частиною вершинного шейдера. Одним із можливих способів оптимізації даного

методу є наближення обчислень, тому що цей рух є адитивним при моделюванні через реакцію зіткнення з об'єктами.

Далі ми опишемо три алгоритми пошуку випадкової точки на поверхні тривимірної моделі. Після вибору точки проводиться процес генерування одного леза, або ж пучка трави. Процес триває до тих пір поки кількість лез менше *n* або якщо згенерований об’єкт є відхиленим за відбором диску Пуассона, що визначає посів трави.

2.1.1 Розподіл з випадковою направленістю граней (Random-face distribution)

Перший алгоритм розподілу, який буде розглянуто є розподіл з випадковою направленістю граней. Він випадковим чином обирає сторону моделі та обчислює випадкову точку всередині трикутника, що їй належить. Кожну точку трикутника можна виразити за допомогою барицентричних координат. Ці тривимірні координати можна також використовувати для інтерполяції між вершинами трикутника. Таким чином, точка, представлена барицентричними координатами, знаходиться всередині трикутника, якщо кожна барицентрична координата є додатною, а сума координат не перевищує одиниці. Формула 2.1 показує обчислення випадкової точки всередині трикутника з використанням барицентричних координат. У цьому рівнянні b посилається на барицентричні координати, а r_1, r_2 і r_3 - випадкові величини. Крім того, P_B вказує на інтерпольоване положення, а P_i - i-та вершина трикутника.

$$\mathbf{b} = \frac{[r_1, r_2, r_3]}{r_1 + r_2 + r_3}$$
$$\mathbf{P}_B = \sum_{i=0}^{3} b_i * \mathbf{P}_i \qquad (2.1)$$

Результат роботи даного алгоритму можна побачити на рисунку 2.1.

2.1.2 Розподіл площею (area distribution)

На відміну від попереднього алгоритму, розподіл площею намагається заповнити кожну грань моделі з однаковою щільністю точок. Для цього він обчислює максимальну кількість точок для кожної грані відповідно до її площі та заданої щільності. Крім того, протягом усього процесу для кожної сторони підраховується кількість точок, які утворюються всередині неї. Це число не повинно перевищувати її ємність, до тих пір поки інше сторона має місце для розташування травинок. Розрахунок ємності показаний у формулі (2.2), де n_i відповідно до A_i відноситься до ємності та площі i-тої грані, а d - параметр щільності.

$$n_i = [d\ A_i] \quad (2.2)$$

Під час процесу генерації, в алгоритмі розподілу площею обчислюється випадкова точка на передній стороні, починаючи з будь-якої грані моделі. Для наступних ітерацій та сама лицева сторона використовується для генерації випадкової точки. Як тільки всі точки розподілились відповідно до його ємності сторони, вибирається наступна лицева сторона. Цей процес триває до тих пір, поки всі сторони не заповняться. Однак, оскільки кількість точок які може вмістити кожна сторона напряму залежить від площі, в результаті округлення ми отримаємо менше очок, ніж бажано. Таким чином, решта точок генеруються випадковим чином на гранях, на яких не було згенеровано леза. Якщо ж таких сторін немає, то решта лез генеруються

$$n = d \sum \quad A_i \quad (2.3)$$

випадковим чином на будь-якій стороні. Схему розподілу площ можна побачити в формулі (2.3). Два приклади роботи цього алгоритму представлені в середньому стовпчику рисунка 2.1.

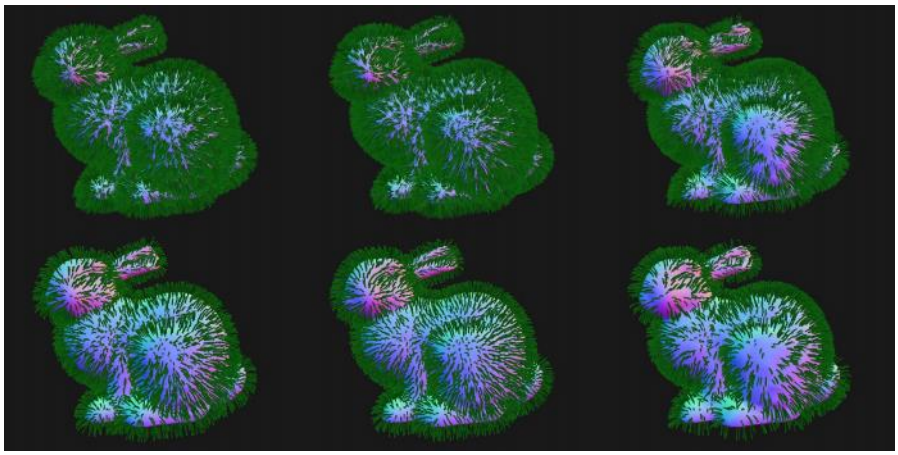


Рис. 2.1 Порівняння різних методів поширення трави разом з різними способами генерації трави, рядки показують способи генерації трави (зверху по одному лезу, знизу насіннєвий спосіб), а стовпчики розповсюдження (зліва на право: random-face distribution, area distribution, volumetric distribution з використанням дерева октантів). Даний малюнок було взято в [3].

2.1.3 Об'ємний розподіл (volumetric distribution)

На відміну від розподілів, про які було сказано вище, об'ємний розподіл спрямований на формування точок з рівномірною щільністю в тривимірному просторі. Це може бути корисно для формування трави на рельєфі, який має велику різницю кривизни. Якщо всі грані будуть заповнені однаково рівномірно, утвориться багато точок у вузьких місцях, в яких не можна розмістити траву. Цей випадок проілюстрований на малюнку 2.2, де верхнє зображення показує розподіл площею, а нижнє зображення об'ємний розподіл.

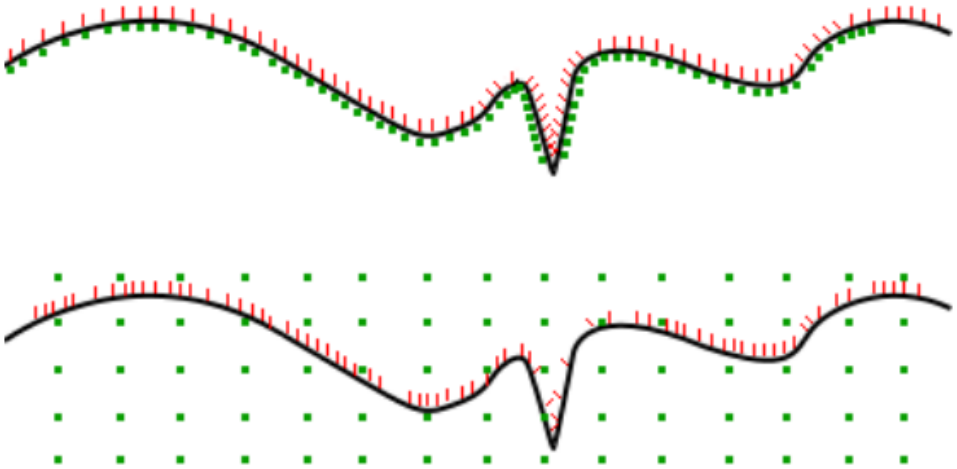


Рис. 2.2 Порівняння розподілу між алгоритмами площинного (верх) та об’ємного (низ) поширення. Малюнок взято з [3]

Крім того, червоні лінії цієї фігури відповідають за згенеровані трав’яні леза, а зелені точки - розподіл випадкових наміток. Під час кожної ітерації процесу генерації об'ємного розподілу обирається випадкова точка всередині обмежувального поля тривимірної моделі та проводиться пошук найближчої точки до поверхні. Для того щоб ми мали змогу зробити це за прийнятний час, можна використати таку структуру даних як дерево октантів. Приклад такої реалізації описаний у вигляді псевдокоду в алгоритмі 2.1.

Спочатку, будується дерево октантів для вершин моделі та зберігаються відповідні сторони у кожній вершині. Після цього в обмежувальній зоні випадковим чином вибирається точка і знаходяться найближчі вершини, використовуючи отримане на попередньому кроці дерево октантів. Це дає нам список можливих сторін, на яких буде проводитись пошук найближчої точки до поточної випадкової. Після цього обчислюється відстань до кожної з них та обирається найближча.

Результат можна побачити в правій колонці на рисунку 2.1. Аналізуючи алгоритм, можна легко помітити, що щільність розповсюдження не однакова для всієї моделі. Наприклад, сторони в регіонах з великим порожнім об'ємом отримали більше в процесі висіву насіння. Це не є недоліком об'ємного розподілу, так як він

спрямований на рівномірне розповсюдження в тривимірному просторі, а не на поверхні моделі. У правому стовпчику рисунка 2.1 цей прояв більшої щільності видно на задній частині та грудях зайчика, тоді як ніс, вуха та стопи мають меншу.

Алгоритм 2.1: Distribution - Face area описаний в [3]

```
points ← 0;
for each face f do
    N_p ← f.area * density;
    if N_p ≥ 1 then
        generate ⌊N_p⌋ points;
        points ← points + ⌊N_p⌋;
    end
end
if points < n then
    generate rest on faces without points or random if there are no faces without
     points;
end
```

2.2 Симуляція вітру

Вітер є важливим елементом у створенні реалістичної та візуально красивої рослинності. Проблема імітації руху дерев у вітровому полі є складною науковою темою. Більшість наукових підходів використовують методи фізичного моделювання, які базуються на рівняннях руху [28] , які застосовують силу вітру до окремих гілок. Такі імітації, як правило, є непомірними для обмежених часом застосувань, і, крім того, вони можуть дати менш природний вигляд через притаманну складність основної проблеми.

У природі трава завжди перебуває в русі, або під впливом вітру, або від предметів, які залишають після себе сліди. Вітер може бути застосований за допомогою текстури, обчисленої певним чином, наприклад, фізичним моделюванням [29], або ж це можна обчислити, використовуючи функцію вітру всередині шейдера.

В роботі [2] автори реалізували функцію вітру $w(\mathrm{p})$, яка приймає положення світового простору $p = (p_x, p_y, p_z)$ та поточний час t як параметр, і потім обчислює вітер як дві перекриваючі синусоїдальні і косинусоїдні хвилі вздовж осі x та косинусоїда вздовж осі z. Функцію можна побачити в формулі (2.2), де константи c_i визначають форму вітрових хвиль разом з невеликим числом ε, щоб уникнути поділу на нуль. Результат функції вітру може бути застосований безпосередньо до зміщення верхніх вершин чотирикутника у вершинному шейдері.

$$\begin{aligned} w(\mathbf{p}) &= \sin(c_1 \cdot a(\mathbf{p})) \cdot \cos(c_3 \cdot a(\mathbf{p})) \\ a(\mathbf{p}) &= \pi \cdot \mathbf{p}_x + t + \frac{\frac{\pi}{4}}{|\cos(c_2 \cdot \pi \cdot \mathbf{p}_z)| + \varepsilon} \end{aligned} \quad (2.2)$$

Разом з вітром, на кожну травинку також можуть впливати окремо зовнішні сили. Для цього можна використовувати карту сил, яка вказує напрямок, в якому лезо штовхатиметься у певне положення. Для імітації слідів вектори областей, які повністю перебувають під тиском об’єкту, вказують у негативному напрямку осі *y*. На краях об'єкту вектори сили направлені від центральної точки об'єкта, яка може бути наближена нормальним вектором на границях. Вектори регіонів навколо об'єкта також вказують від його центру, але із поступовим зменшенням довжини. Розмір цієї зони залежить від значення прикладеної сили.

В описаній вище реалізації досить важко відслідковувати розсіювання енергії зіткнення. Таким чином автори в [1] запропонували більш інтуїтивний процедурний метод. Спершу вони додають часову позначку до кожної активованої клітинки, в яку записують час її активації. Ця мітка буде оновлюватися кожного разу, коли об’єкт потрапляє в плитку. На кожному кадрі проводиться порівняння позначки з поточним часом. Якщо різниця перевищує поріг, тобто клітинка не торкалася жодних предметів протягом останніх секунд, припускається, що її енергія зникла і плитка стає не активованою. Приклад роботи даного підходу можна побачити на рисунку 2.3.

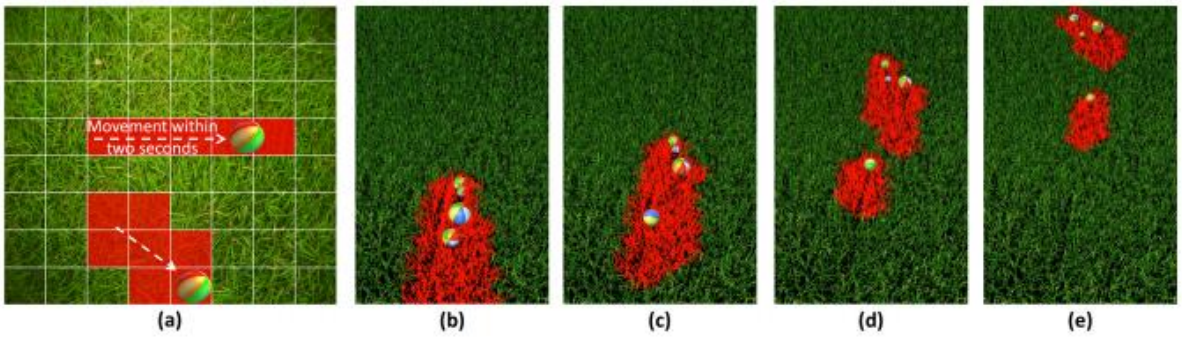


Рис. 2.3 (a) Активована плитка оновлюється відповідно до позицій об'єктів на сцені. (b) - (e) - послідовні скріншоти, які червоним кольором показують процес активації плиток в сцені під час руху об’єктів

3. Взаємодія об’єктів у віртуальному світі

В цьому розділі ми розглянемо моделі фізичної взаємодії у віртуальному світі. Приділимо увагу тому як їх можна застосовувати до анімації трави, а також їхні переваги та недоліки в поєднанні з алгоритмами описаними в попередніх розділах.

3.1 Фізична поведінка

Однією з найважливіших властивостей симуляцій є фізична модель. В своїй роботі Jahrmann та інші [2] для анімації трави використовували просте зміщення вершин. Пізніше Jahrmann разом з Wimmer адаптували більш складний алгоритм руху, який базується на фізичній моделі. Це гарантувало, що довжина травинок не буде змінюватись незалежно від того, які сили на неї діятимуть. Крім того, всі сили об’єднуються в єдину фізичну модель. Такий підхід є кращою версією моделі, яку запропонували Fan і співавт. в [1], де зіткнення та обчислення вітру виконуються незалежно один від одного.

Одним з загальних підходів до реалізації фізичної моделі є використання карт сили. Карта сили є загальною структурою даних для фізичної моделі, яка є двовимірною текстурою. Кожен текстель (одиниця текстурного простору [30]) карти містить тривимірний вектор, що вказує на зміщення v_2 від вихідного положення. Четверта координата називається силою колізії і описує супротив виниклого зіткнення. З часом, значення цієї сили зменшується, поки знову не досягне нуля.

Для того, щоб зв’язати тексель з травинкою, кожен патч з інформацією про об'єкт має квадратну площу, яка призначена для його травинок. З цієї причини рекомендується мати максимальну кількість лез на одному патчі близькою до квадрату натурального числа. Положення квадрата в карті визначається

двовимірним вектором зміщення *o*. Цей вектор залежить від його індексу та від загальної кількості патчів з інформацією.

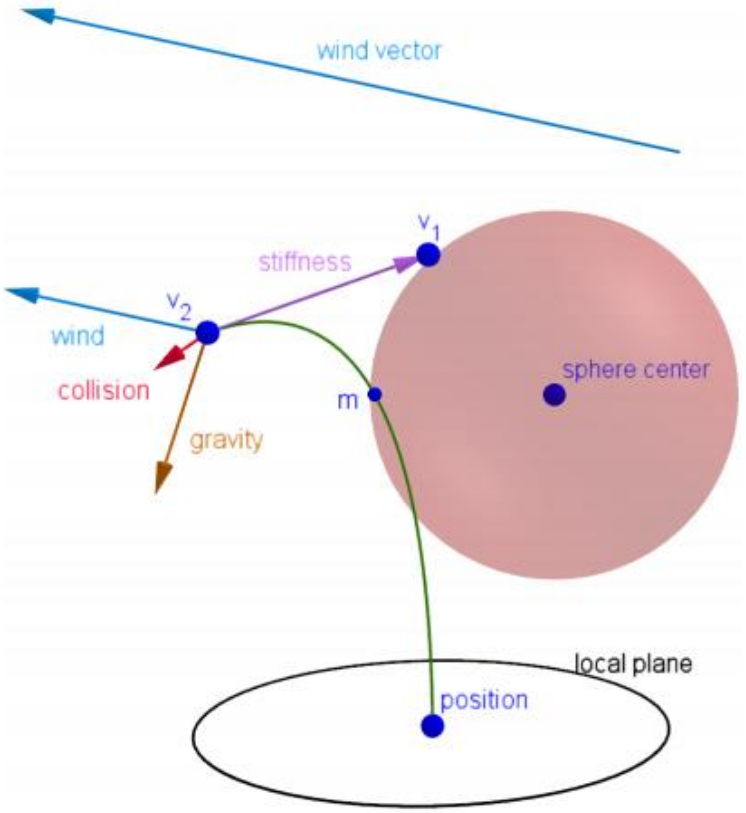


Рис. 3.1 Даний рисунок демонструє фізичну модель. Вектори відповідають за різні сили, які враховуються при зміщенні контрольної точки v_2. Сфера відповідає за об'єкт, а верхній вектор вказує напрямок вітру.

Розрахунок вектора зміщення показано в формулі (3.1), де *id* - індекс об'єкту з патчем інформації, а *p* їх загальна кількість.

$$\mathbf{o} = \begin{pmatrix} id \mod \left\lceil \frac{p}{\lfloor \sqrt{p} \rfloor} \right\rceil \\ \left\lceil \frac{id}{\frac{p}{\lfloor \sqrt{p} \rfloor}} \right\rceil \end{pmatrix} \quad (3.1)$$

Для того щоб травинка змогла прочитати задане значення з карти сили, необхідно обчислили координати текселя. Це можна зробити, знаючи індекс травинки, вектор зміщення відповідного патчу з інформацією та довжину сторони

квадрата. У формулі (3.2) представлено обчислення координат текстури **t**, де *i* вказує на індекс леза, а *s* позначає розмір квадрата.

$$\mathbf{t} = \begin{pmatrix} t_1 \\ t_2 \end{pmatrix}$$
$$t_1 = s\ o_1 + (i \mod s)$$
$$t_2 = s\ o_2 + \frac{i}{s} \tag{3.2}$$

3.2 Гравітація

Гравітація - це константна сила, яка застосовується до v_2. Її вплив базується на висоті травинки, на низькі леза менше, на високі більше. Напрямок та інтенсивність сили тяжіння можна обчислити двома різними способами. З одного боку, напрямок та інтенсивність можуть бути однаковими для всієї сцени, як у реальному світі в місцевих районах. З іншого боку, гравітація може бути спрямована на певну точку в просторі. Це явище існує і в реальному світі, якщо ми будемо розглядати гравітацію з глобальної точки зору. Крім того, обчислення вектора сили тяжіння може використовувати обидва варіанти, які ми визначаємо як гравітацію довкілля, g_E. Ця інтерполяція показана у формулі (3.3), де G відповідно L являє собою глобальний і локальний вектори тяжіння з інтенсивністю в якості четвертого виміру. Крім того, *t* - параметр інтерполяції між локальною та глобальною гравітацією.

$$\mathbf{g}_E = \left(\frac{\mathbf{G}_{xyz}}{\|\mathbf{G}_{xyz}\|} \mathbf{G}_w (1-t) + \frac{\mathbf{L}_{xyz}}{\|\mathbf{L}_{xyz}\|} \mathbf{L}_w t \right) \tag{3.3}$$

Якщо до травинки застосовується лише сила тяжіння навколишнього середовища, то може так статись, що ця сила тяжіння буде спрямована протилежно її верх-вектору і лезо не покидатиме своє початкове положення. Це пов'язано з тим, що корекція довжини знову встановлює v_2 у початкове положення, якщо воно просто зміщене в негативному напрямку верх-вектора. Однак, оскільки трава є

еластичною, її кінчик в реальному світі буде штовхатись вниз. Для моделювання цього ефекту додається додаткова частина до сили тяжіння, яка називається передньою силою тяжіння, g_F. Ця додаткова сила визначає фронтальний напрямок травинки, який обчислюється крос-добутком між верх-вектором леза та вектором, що йде по ширині леза. Значення цієї сили залежить від значення гравітації навколишнього середовища. Формула (3.4) показує обчислення передньої сили тяжіння, де f - передній напрямок травинки.

$$\mathbf{g}_F = \frac{1}{4} \|\mathbf{g}_E\| \mathbf{f} \quad (3.4)$$

Кінцевий вектор сили тяжіння може бути обчислений нормалізацією суми сили тяжіння навколишнього середовища і передньої сили тяжіння. Нормалізація проводиться з використанням висоти леза, коефіцієнта гнучкості та часу останнього кадру. Формула 3.5 показує обчислення вектора сили тяжіння g, де h - висота леза, *b* коефіцієнт гнучкості.

$$\mathbf{g} = (\mathbf{g}_E + \mathbf{g}_F)\, h\, b\, \Delta t \quad (3.5)$$

3.3 Взаємодія об’єктів

Для імітації природної поведінки трави, необхідно, щоб травинки могли реагувати на зміни в середовищі. У попередніх розділах були описані природні ефекти, такі як вітер та сила тяжіння, які можна виразити простими функціями. Крім цих ефектів трава також має бути здатна реагувати на зіткнення між лезами трави та іншими об’єктами. Одним із способів досягнення цього ефекту є використання обмежувальної сфери об'єкта, яка демонструє реалістичні результати зіткнення для опуклих об’єктів.

Крім зіткнення простих сфер, дана фізична модель також здатна керувати зіткненнями зі складними моделями. Складність роботи зі складними моделями полягає у їхньому представленні, що використовується для обчислень. Ця

складність випливає з того, що обчислення зіткнення кожного полігону моделі з кожною травинкою не може бути здійснено за розумний час. Тому в [8] пропонується використання сфер як представлення складного об'єкту. Автори обрали сфери, оскільки зіткнення зі ними можна обчислити швидше, як було показано у попередніх розділах.

Представлення тривимірного об'єкта як набору сфер є NP-складною проблемою оптимізації, оскільки її можна розглядати як проблему упаковки в корзину або як проблему покриття набором [17], які теж є NP-складними. Існує кілька варіантів цієї проблеми представлення. Взагалі кажучи, варіанти утворюють дві групи, які відрізняються обмеженням, чи можуть згенеровані сфери перекриватися чи ні. Надалі буде розглядатись подання сфер, яке не дозволяє їм перетинатися. Цю проблему можна вирішити за допомогою методики пакування сфер [18]. Для пошуку сфер можна використовувати алгоритм ProtoSphere запропонований Weller та іншими в [19], який можна реалізувати з використанням графічного процесору. Він складається з ітеративного процесу оптимізації, де кожна ітерація складається з трьох етапів. Однак перед тим, як виконати алгоритм, тривимірну воксельну сітку потрібно попередньо обчислити та надіслати на відеокарту. У цій сітці кожен воксель зберігає найближчу точку на поверхні моделі. Ця структура даних представляє модель під час циклу оптимізації. У кожній ітерації необхідно виконати три етапи, які перелічені нижче.

1. Для кожної сторони моделі генерується одна прототипна точка, даний процес відбувається на стороні програми.
2. На графічній карті позиції прототипів переміщують n разів. Для кожного переміщення прототип знаходить найближчу поверхневу точку, використовуючи сітку вокселів. Коли точка знайдена, він відсувається від найближчої точки на відстань, що є між цими точками. Це переміщення множиться на коефіцієнт згладжування, який стає меншим з кожною ітерацією. Після обчислення n переміщень

вимірюється кінцева відстань до найближчої поверхні для кожного прототипу.

3. Останній крок робиться ще раз на стороні програми. Прототипи сортують за їх відстанню до найближчої точки поверхні у порядку зменшення. Після цього генерується сфера для кожного прототипу, якщо ця сфера не перетинає жодну іншу сферу. Після обробки всіх прототипів воксельна сітка оновлюється новими сферами, які розглядаються як додаткові межі об'єкта.

У випадку рендерингу трави, модель повинна бути щільно упакована на поверхні, а не у внутрішній частині. Отже, отримані сфери фільтруються таким чином, що лише сфери на межі моделі залишалися для представлення об'єкта. Це представлення можна попередньо обчислити для кожної моделі один раз і зберегти разом із нею, щоб уникнути тривалих періодів завантаження для дуже складних моделей. Коли дана техніка готує дані для оновлення сил патчу, сфери, які представляють об'єкт, перевіряються на перетин з обмежувальною коробкою патча. Усі сфери, які не перетинають патч, не надсилаються на відеокарту. Крім того, сфери, які значно менші, ніж лозини трави пластиру, також відсікаються.

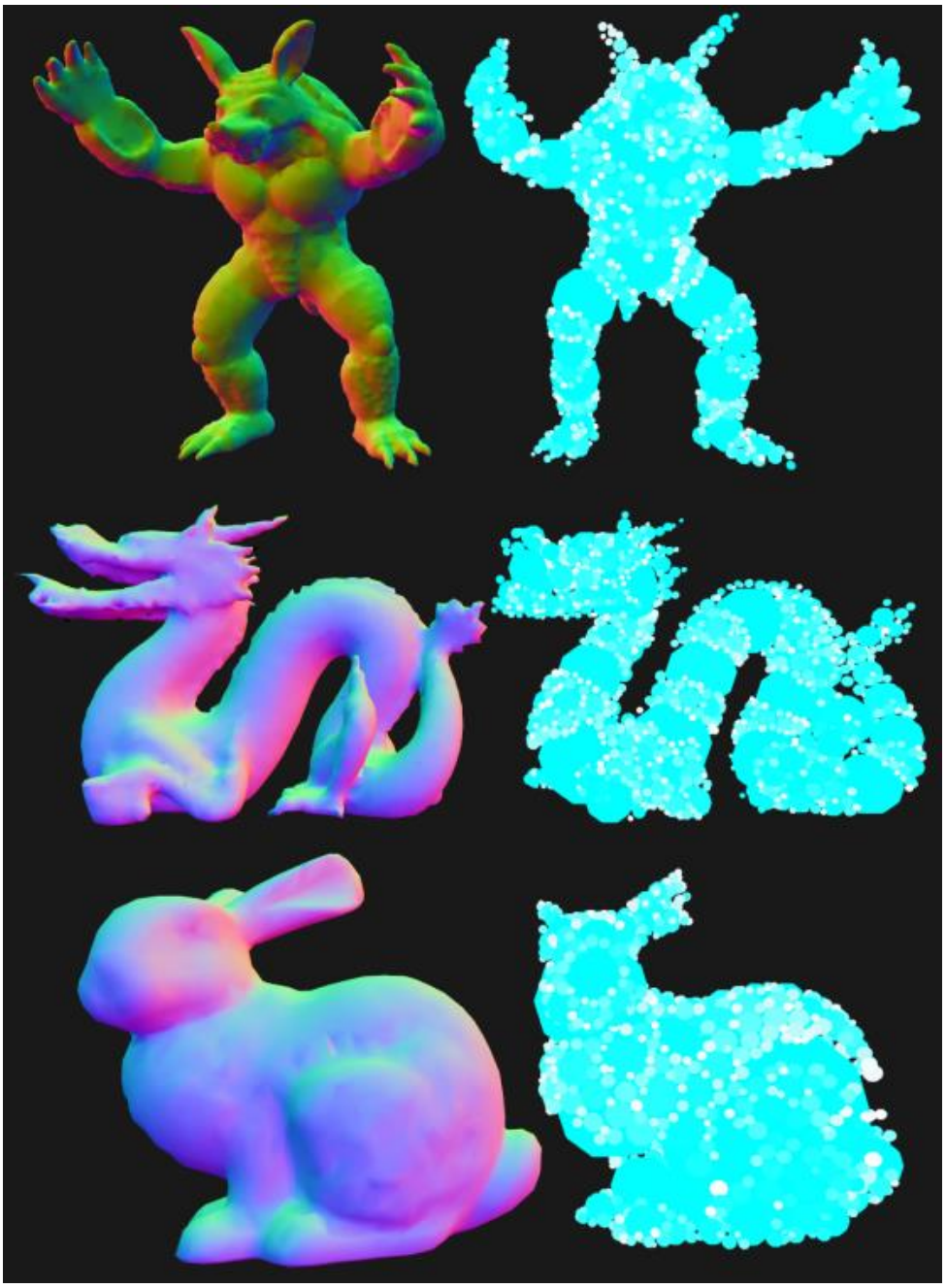


Рис. 3.2 Приклад заповнення 3D моделі сферами; малюнок взято з роботи [3]

4. Методи оптимізації

Візуалізація великої кількості лез трави є складним завданням для відеокарти. Особливо, якщо більшості з них не видно під час фінального рендерингу, значна кількість ресурсів витрачається даремно. Існують різні методи оптимізації, що відрізняються за точністю та часом роботи. Їхня основна мета полягає в тому, щоб передати ті об’єкти на рендеринговий конвеєр, які видно на кінцевому зображенні. Це досягається шляхом оцінки видимості кожної травинки безпосередньо на відеокарті за допомогою обчислювальних шейдерів.

В цьому розділі розглянемо такі методи оптимізації, як різнорівневі обрізання:

- обрізання за орієнтацією;
- обрізання по видовому фрустуму (зрізане геометричне тіло);
- обрізання за відстанню;
- обрізання за оклюзією.

Обрізання - є одним з найефективніших способів оптимізації графічних застосунків. Обрізання на основі одиночних травинок є інноваційним підходом [8], особливо якщо ми будем порівнювати його з іншими видами, де обрізання може здійснюватися лише на кіпах[13, 15], латках[6,2] або інших структурах даних високого рівня, таких як сітки [1].

Обрізання високого рівня виконується на стороні застосунку та відповідає за фільтрацію об'єктів, які знаходяться повністю поза зоною видимості камери. Видовий фрустум, як правило, одержують шляхом взяття фрустуму (обрізання паралельними площинами) видової піраміди, що є адаптацією (ідеалізованого) видового конусу. Це відсікання обчислюється на двох різних рівнях ієрархії об'єктів.

Кожне поле трави представлене обмежувальним вікном, орієнтованим на осі, яке обчислюється об'єднанням обмежувальних зон відповідних латок. По-перше, це

поле перевіряється проти фрустуму камери. Якщо обмежувальної зони не видно, все поле відсікається. В іншому ж випадку проводиться обчислення перетину між видовим фрустумом та обмежвальними зонами кожної відповідної латки трави. Всі латки, обмежувальні зони яких знаходяться поза межами видового фрустуму, відтинаються.

4.1 Відсікання за орієнтацією

Метод відсікання за орієнтацією відтинає травинки на основі їх орієнтації відносно камери. Це є важливим фактором, так як воно напряму залежить від псевдо тривимірності травинки - вона не має товщини. Коли лезо рендерять збоку, це може спричинити небажані артефакти, оскільки його зпроектована ширина менша за розмір пікселя. Через це вектор напрямку камери порівнюється з вектором, що йде по ширині леза. Таке порівняння досягається шляхом обчислення абсолютного значення косинуса кута векторів напрямку, яке називається відносною орієнтацією. Це можна побачити в формулі 4.1, де υ - відношення орієнтації, $\mathbf{dir_c}$ вказує напрямок від камери до травинки, а $\mathbf{dir_b}$ - напрямок травинки. Якщо коефіцієнт орієнтації υ перевищує 0,9, лезо відсікається.

$$\upsilon = |\mathbf{dir_c} \cdot \mathbf{dir_b}| \quad (4.1)$$

4.2 Обрізання за видовим фрустумом

Під час відтинання за видовим фрустумом кожну травинку перевіряють, чи знаходиться вона всередині спроектованої рамки. Оскільки проводити обчислення для кожної точки на кривій леза неможливо, на ньому обираються три опорні точки: точка місцезнаходження леза, v_2 та середня точка кривої, яка отримується інтерполяцією сплайну з параметром 0,5. Кожна з цих точок проектується на

нормалізовані координати шляхом множення на видово-проекційну матрицю. Ці нормалізовані координати можна перевірити на те, чи знаходяться вони всередині видового фрустуму, шляхом порівняння кожного виміру з однорідною координатою, яка представлена четвертим виміром. Крім того, проводиться розрахунок того, чи розташовані координати між переднім та заднім планом камери, використовуючи однорідну координату. Може так статись, що обчисливши опорні точки, сегменти фактичного леза буде видно навіть у тому випадку, якщо всі три точки знаходяться поза видовим фрустумом. Для врахування таких моментів автори [8] пропонують додавати деяке значення допуску до змінних, що використовуються в наступних розрахунках. Обчислення видимості v для однієї точки **p** показано в формулі 4.2, де **VP** позначає видово-проекційну матрицю, t - значення допуску, а n відповідно до f позначає передню та задню площину камери.

$$\begin{aligned}
\mathbf{p}_{NDC} &= \mathbf{VPp} \\
\mathbf{p}_{NDC_w} &= \mathbf{p}_{NDC_w} + t \\
n_t &= n - 2t \\
f_t &= f + 2t \\
v &= \mathbf{p}_{NDC_x} > -\mathbf{p}_{NDC_w} \wedge \mathbf{p}_{NDC_x} < \mathbf{p}_{NDC_w} \\
v &= v \wedge \mathbf{p}_{NDC_y} > -\mathbf{p}_{NDC_w} \wedge \mathbf{p}_{NDC_y} < \mathbf{p}_{NDC_w} \\
v &= v \wedge \mathbf{p}_{NDC_w} > n_t \wedge p_{NDC_w} < f_t
\end{aligned} \tag{4.2}$$

Якщо результат v для кожної з трьох точок є хибним (false), травинка відсікається.

4.3 Відсікання за відстанню

Якщо розглядати поле трави в перспективній проекції, воно буде більш щільним біля горизонту. Під час рендерингу, це збільшення щільності створює два небажані ефекти:

1. Точність обчислень відеокарти є кращою для малих значень глибини.
2. Травинки, що знаходяться на відстані, зменшуються в розмірах внаслідок перспективи.

Однак, якщо дивитись на поле трави зверху, щільність не збільшується так само як при збільшенні відстані, внаслідок чого видно менше артефактів. Метод відсікання за відстанню вирішує ці проблеми шляхом відсікання стількох лез, скільки необхідно для уникнення появи артефактів. Таким чином відстань від глядача до трави проектується на площину, визначену верх-вектором травинки. Це не впливає на відстань, якщо ми будемо дивитись вздовж поля трави. Але воно зменшуватиме відстань, якщо дивитись перпендикулярно до трав'яного поля. Ця прогнозована відстань обчислюється в два етапи. Спочатку вектор, що вказує від камери до положення землі, обчислюється та проектується на площину, визначену верх-вектором травинки. Потім обчислюється довжина зпроектованого вектора, що дає нам відстань d_{proj}. Цей процес показаний у формулі (4.4), де v_{proj} є проектованим вектором, а **c** позначає положення камери. Крім того, **p** відноситься до положення травинки та до її верх-вектора.

$$\begin{aligned} \mathbf{v}_{proj} &= (\mathbf{p} - \mathbf{c}) - \mathbf{up}\,((\mathbf{p} - \mathbf{c}) \cdot \mathbf{up}) \\ d_{proj} &= \|\mathbf{v}_{proj}\| \end{aligned} \tag{4.4}$$

Метод відсікання за відстанню передбачає, що всі леза є відсортованими таким чином, що травинки поблизу один одного мають схожі показники. Якщо цього не буде, то на густих полях будуть прогалини.

4.4 Відсікання за прихованістю

Зазвичай сцени, окрім трави, також містять й інші геометричні об'єкти. Таким чином, травинки, які розташовуються за іншими предметами, також повинні бути прихованими. Є два загальних способи виявлення прихованих лез. В кожному з методів використовуються ті ж самі точки, що і в обрізанні за видовим фрустумом.

4.4.1 Відсікання внутрішніми сферами

Алгоритм відсікання внутрішньою сферою перевіряє, чи трава прихована максимальною сферою, що міститься в тривимірному об'єкті. Цей метод відсікання найкраще працює для опуклих предметів. Максимальні внутрішні сфери попередньо розраховуються при генерації об'єкта. В [8] автори розглядають методи пошуку максимальної внутрішньої сфери.

Використання сфер для обчислення дає можливість використовувати прості рівняння видимості, які можна обчислити швидко та з використанням невеликого обсягу пам'яті. Так само, як і сфери зіткнення, що були описані в попередньому розділі, кожна сфера представлена чотиривимірним вектором, де три виміри позначають положення центру, а останній визначає радіус сфери. Тест на те, чи прихована травинка за внутрішньою сферою, складається з двох етапів. Цей тест виконується для кожної з трьох точок на траві. Якщо всі ці точки приховані за внутрішньою сферою, лезо вважається невидимим.

4.4.2 Відсікання за текстурою глибини

Цей метод відтинання використовує інформацію про глибину попередньо виведених об'єктів, для того, щоб з’ясувати, чи травинка розташована за якимось з об’єктів. Основна перевага цього методу полягає в тому, що він може обробляти оклюзію будь-якого непрозорого об’єкта незалежно від його геометричних властивостей. Однак, порівняно з відсіканням за внутрішньою сферою, для розрахунку потрібно більше часу та необхідна додаткова текстура глибини. У цьому методі три точки кожної травинки проектуються на екран. Ще разом з розміром екрану в пікселях обчислюються пошукові координати текстури і відповідні значення глибини виводяться з текстури глибини. Нарешті, значення

глибини порівнюються з відстанню леза до камери. Якщо значення глибини менше, травинка відсікається.

Подібно до проблем тіньового мапінгу [20], небажані артефакти можуть з'являтись із згладжуванням, якщо значення глибини стосуються поверхонь, які є не перпендикулярними напрямку зору. Тому їх бажано збільшити на невелике відхилення.

5. Порівняння алгоритмів

В цьому розділі розглядається порівняння та тестування методів генерації трави описані Фаном [1] та Jahrmann [8]. В якості їхньої реалізації використовуються проекти [21] та [22] відповідно. Між цими реалізаціями є певні відмінності, які описані нижче.

Реалізація [21] зроблена з використанням ігрового двигуна Unity [23] та має наступні властивості:

1. Вбудовано шість режимів роботи для тестування алгоритму;
2. Генерація трьох типів об’єктів для демонстрації властивостей трави: куля, куб та кролик;
3. Два режими мапінгу трави;
4. Чотири рівні роздільної здатності текстур.
5. Дві сцени для тестування.

Експериментально було перевірено, що друга сцена працює некоректно, тому в нашому порівнянні було використано лише першу.

Реалізація [22] була розроблена з використанням графічного фреймворку SharpDX, який в свою чергу є обгорткою над графічним API DirectX 11 [24] для мови програмування C#. Дана програма не мала достатнього функціоналу для порівняння, тому була модифікована. В результаті застосунок має наступні властивості:

1. Зміна кількості травинок в одній латці.
2. Чотири рівня деталізації.
3. Динамічний вітер.
4. Зміна роздільної здатності зображення.

Операційна система	Microsoft Windows 10 Version 10.0.19041.264
Процесор	AMD Ryzen 7 3700X
Оперативна пам’ять	HyperX DDR4-3200 32768MB
Відеокарта	AMD RX 5700 XT в режимі роботи PCIe 3.0 x16, драйвер 20.5.1
Накопичувач	Samsung 970 Evo Plus 500GB M.2 PCIe 3.0 x4

Таблиця 5.1 Характеристики системи, на якій проводилось тестування

5.1 Тестування

Для тестування алгоритмів були проведені тести за наступними параметрами:

1. Роздільна здатність зображення.
2. Кількість трави.
3. Деталізація трави.

Всі тести проводились на системі, параметри якої описані в таблиці 5.1. Кожна з програм мала свою власну сцену, але оскільки їхня складність є мінімальною, це практично не вплинуло на результати.

Один з найпростіших тестів, але важливий - перевіряє, як роздільна здатність зображення впливає на продуктивність програми. Для тестування були обрані наступні формати: 1920x1080, 1280x720, 800x600.

Наступний параметр, який був врахований - це кількість травинок на латку. Було використано наступні значення: 10, 100.

Для тестування [21] за деталізацією були використані вбудовані рівні деталізації. Всього їх три плюс дві (загалом 6): високий, середній, низький, два рівні

тесселяції трави - високий та низький. Для [22] були використані наступні рівні деталізації: 0, 20, 45, 70 / 0, 10, 20, 30 / 0, 10, 10, 10.

	0,10,10,10	0,10,20,30	0,20,45,70
1920x1080	120	115	102
1280x720	137	126	92
800x600	230	181	160

Таблиця 5.2 Результати тесту для [22], кількість травинок в патчі 10, кадри за секунду

	0,10,10,10	0,10,20,30	0,20,45,70
1920x1080	130	115	94
1280x720	153	147	130
800x600	163	143	102

Таблиця 5.3 Результати тесту для [22], кількість травинок в патчі 100, кадри за секунду

Приклади роботи застосунків можна побачити на Рис. 5.1-5.2. Тести проводились для різних комбінацій тестувальних параметрів, щоб максимально точно оцінити можливості алгоритмів. Результати зображені в таблицях 5.2-5.4.

5.2 Результати

Провівши тестування [22] можемо зробити висновок, що він є досить ефективним методом генерації трави. Але в той же час в нього є ряд недоліків. В першу чергу, досить важко досягти стабільного значення кількості кадрів за секунду для всієї сцени, досить часто цей показник падав до 20-30 і потім знову

повертався до початкового рівня. По-друге, якість отриманого результату досить низька. Як видно на рис. 5.3 при певному куті зору можна побачити лінії трави та прогалини між нею.

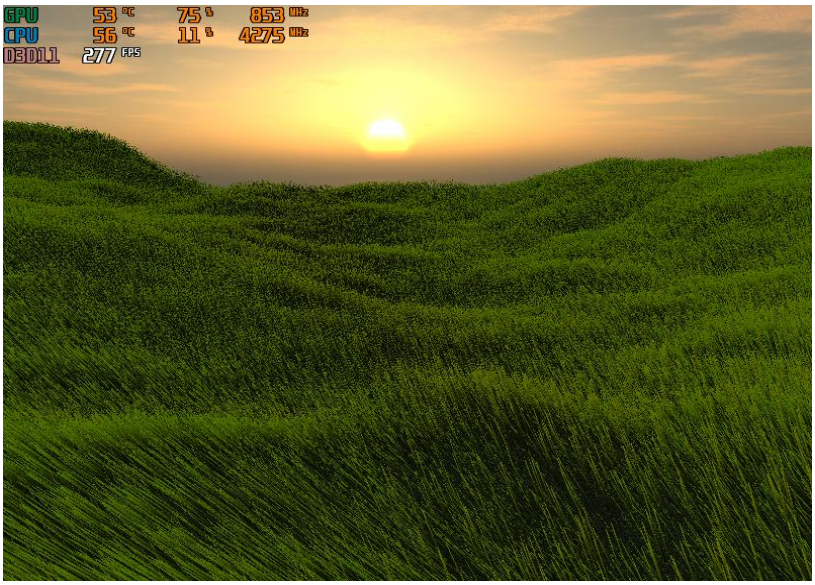


Рисунок 5.1 Приклад роботи [22]

Для вирішення цієї проблеми можна рандомізувати розмір та позицію травинок на рівні латки і таким чином ми досягнемо кращого, більш природнього вигляду поля.

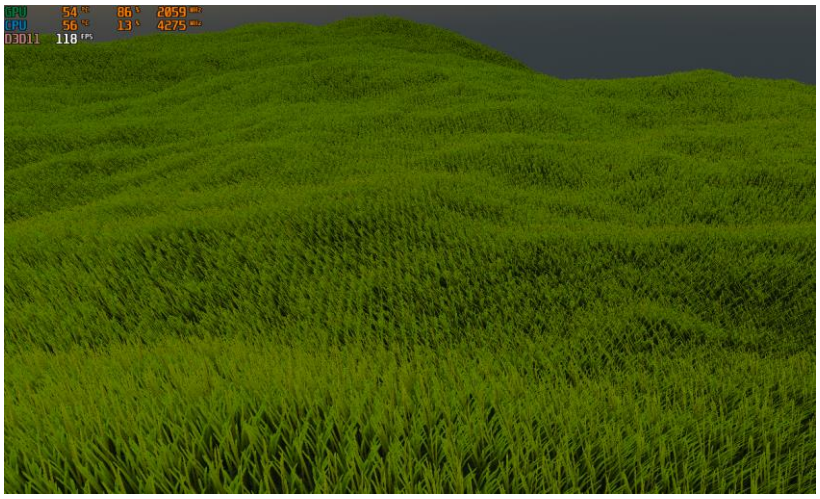


Рисунок 5.3 Артефакти алгоритму [22]

Реалізація [21] продемонструвала меншу продуктивність, але в той же час виглядає більш реалістично за рахунок використання повноцінних текстур, в той час як в [22] генеруються процедурно (Рис. 5.2).

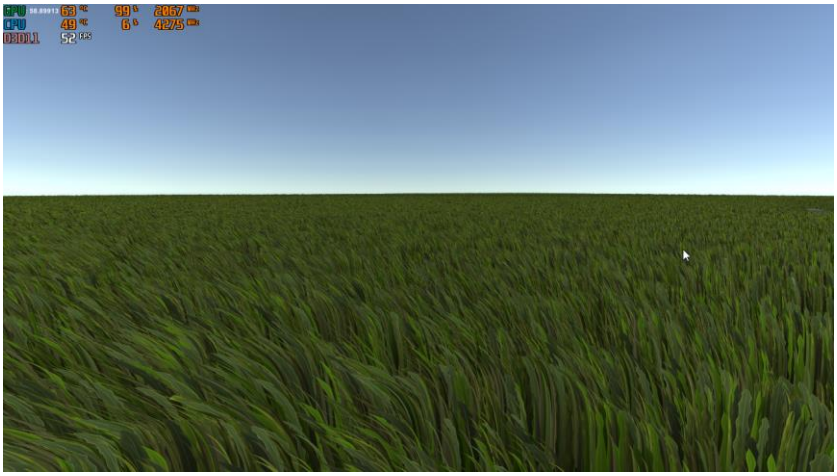


Рисунок 5.2 Приклад роботи [21]

Суттєвими недоліком цієї реалізації є її вимога до ресурсів системи. Кількість травинок у всіх рівнях деталізації відносно менша за [22], загальний результат вийшов мінімум в два рази гірший (Таблиця 5.3).

	Low	Medium	High
1920x1080	33	9	4
1280x720	39	10	4
800x600	64	17	5

Таблиця 5.4 Результати тесту для [21], обробка травинок Low, кадри за секунду

Алгоритм має схожі проблеми з розташуванням трави, які ми бачили в [22]. Відмінність полягає в тому, що замість ліній ми бачимо межі латок. Це добре видно на найвищому рівні деталізації (див. рис. 5.3).

Рисунок 5.3 Артефакти розташування трав’яних латок в [21]

Другий недолік було знайдено в системі оптимізації. Якщо швидко рухати камерою зі сторони в сторону, то на краях вікна програми буде видно як повільно оновлюється невидима частина сцени.

Обидва підходи є інноваційними для вирішення проблеми генерації реалістичної трави, кожен з яких має свої переваги та недоліки. Експериментально було перевірено, що [8] є більш продуктивним, в той же час як [1] дає візуально кращий результат.

Висновки

Генерація трави є складною та актуальною задачею в комп’ютерній графіці. Багато розробників займалися цією задачею, але, на жаль, жодного ефективного рішення так і не було знайдено.

Всі рішення, які існують на сьогоднішній день, генерації трави можна об’єднати в три групи: картинкові, геометричні та гібридні. Картинкові методи використовують для генерації трави текстури з різною орієнтацією в просторі. Їхня перевага полягає в ефективності. Геометричні - дають кращий результат, але й при цьому вимагають більше ресурсів для своєї роботи. Гібридні методи поєднують підходи з попередніх двох груп для досягнення високопродуктивного та реалістичного результату.

В роботі проаналізовано наявні методи оптимізації генерації трави. Найвідоміші з них - це обрізання: за орієнтацією, по видовому фрустуму (зрізане геометричне тіло), за відстанню та за оклюзією. Вони дозволяють зменшити кількість об’єктів, які потрапляють на графічний конвеєр і цим самим суттєво покращити продуктивність програми.

Досліджено та протестовано два найвідоміших на сьогодні методи генерації трави: метод описаний Fan та метод описаний Jahrmann. Проведено тестування алгоритмів за параметрами: роздільна здатність зображення, кількість трави та деталізація трави. Особливістю першого є хороший, реалістичний результат, але в той же час його продуктивність для великих сцен досить слабка. В той же час, другий має значно кращу продуктивність, але при цьому результат гірший.

Список літератури

1. Zengzhi Fan, Hongwei Li, Karl Hillesland, and Bin Sheng. Simulation and rendering for millions of grass blades. In Proceedings of the 19th Symposium on Interactive 3D Graphics and Games, i3D ’15, New York, NY, USA, 2015. ACM.
2. Klemens Jahrmann and Michael Wimmer. “Interactive grass rendering using real-time tessellation.” In Manuel Oliveira and Vacllogy and Computer Science - Volume 01. IEEE Computer Society, 2009.
3. Fabrice Neyret. A General and Multiscale Model for Volumetric Textures. In Graphics Interface. Canadian Human-Computer Communications Society, 1995.
4. Ralf Habel, Michael Wimmer, and Stefan Jeschke. Instant Animated Grass. Journal of WSCG, 2007.
5. Alison McMahan. Immersion, engagement and presence. The video game theory reader, 2003.
6. Boulanger, K., Pattanaik, S., and Bouatouch, K. 2009. Rendering grass in real time with dynamic lighting. Computer Graphics and Applications, IEEE 29, 1 (Jan).
7. Sousa, T. 2007. Vegetation procedural animation and shading in Crysis. In GPU Gems 3, Pearson Education, Inc., Boston, H. Nguyen, Ed.av Skala, editors, WSCG 2013 Full Paper Proceedings, June 2013
8. BSc Klemens Jahrmann: Interactive Grass Rendering in Real Time Using Modern OpenGL Features
9. Kévin Boulanger, Sumanta N. Pattanaik, and Kadi Bouatouch. Rendering grass in real time with dynamic lighting. IEEE Comput. Graph. Appl., January 2009

10. Reeves, W. T., and Blau, R. 1985. Approximate and probabilistic algorithms for shading and rendering structured particle systems. In Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques, ACM, New York, NY, USA, SIGGRAPH ’85.
11. Changbo Wang, Zhangye Wang, Qi Zhou, Chengfang Song, Yu Guan, and Qunsheng Peng. Dynamic modeling and rendering of grass wagging in wind: Natural phenomena and special effects. Comput. Animat. Virtual Worlds, July 2005.
12. Viewing frustum [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Viewing_frustum
13. J. Orthmann, C. Rezk-Salama, and A. Kolb. GPU-based Responsive Grass. Journal of WSCG, 2009.
14. Musawir A. Shah, Jaakko Kontinnen, and Sumanta Pattanaik. Real-time rendering of realistic-looking grass. In Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia, GRAPHITE ’05, New York, NY, USA, 2005. ACM.
15. Xiangkun Zhao, Fengxia Li, and Shouyi Zhan. Real-time animating and rendering of large scale grass scenery on gpu. In Proceedings of the 2009 International Conference on Information Techno.
16. Kurt Pelzer. Rendering countless blades of waving grass. In Randima Fernando, editor, GPU Gems. Addison-Wesley, 2004.
17. O. Aichholzer, F. Aurenhammer, B. Kornberger, S. Plantinga, G. Rote, A. Sturm, and G. Vegter. Recovering structure from r-sampled objects. Computer Graphics Forum, 2009.
18. Mhand Hifi and Rym M’hallah. A literature review on circle and sphere packing problems: models and methodologies. Advances in Operations Research, 2009.

19. Rene Weller and Gabriel Zachmann. Protosphere: A gpu-assisted prototype guided sphere packing algorithm for arbitrary objects. In ACM SIGGRAPH ASIA 2010 Sketches, SA ’10, New York, NY, USA, 2010. ACM.
20. Cass Everitt, Ashu Rege, and Cem Cebenoyan. Hardware shadow mapping. White paper, nVIDIA, 2, 2001.
21. Проект реалізації алгоритму Фана[Електронний ресурс] – Режим доступу до ресурсу: https://github.com/nicoell/Grass-Simulation
22. Проект реалізації алгоритму Джахрманна [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/mreinfurt/Grass.DirectX
23. Unity [Електронний ресурс] – Режим доступу до ресурсу: https://docs.unity3d.com/Manual/index.html
24. SharpDX [Електронний ресурс] – Режим доступу до ресурсу: http://sharpdx.org/
25. Alison McMahan. Immersion, engagement and presence. The video game theory reader, 2003.
26. Guerraz, S., Perbet, F., Raulo, D., Faure, F., and Cani, M.-P. 2003. A procedural approach to animate interactive natural sceneries. In Proceedings of the 16th International Conference on Computer Animation and Social Agents (CASA 2003), IEEE Computer Society, Washington, DC, USA, CASA ’03.
27. Модель шейдінгу Фона [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Phong_shading
28. Рівняння руху в класичній механіці [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D0%B8%D1%87%D0%BD%D0%B0_%D0%BC%D0%B5%D1%85%D0%B0%D0%BD%D1%96%D0%BA%D0%B0

29. Фізичне моделювання [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%A4%D1%96%D0%B7%D0%B8%D1%87%D0%BD%D0%B5_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8E%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F
30. Andrew Glassner, An Introduction to Ray Tracing, San Francisco: Morgan—Kaufmann, 1989.