УДК 519.683.8
Гринюк С.В., Мельник В.М., Равенець А.В.
Луцький національний технічний університет

БУЛЕВІ ОПЕРАЦІЇ НАД ПОЛІГОНАМИ ПРИ СТВОРЕННІ МАПИ ШЛЯХІВ У ТРИВИМІРНОМУ ПРОСТОРІ ДЛЯ КОМП'ЮТЕРНИХ ІГОР

Гринюк С.В., Мельник В.М., Равенець А.В. Булеві операції над полігонами при створенні мапи шляхів у тривимірному просторі для комп'ютерних ігор. Ігри жанру RTS (real time strategy) використовують 3d компонент Mesh для визначення областей призначених для переміщення рухомих ігрових одиниць (мапа шляхів). Зазвичай це статична мапа яка не змінюється з часом гри. В данні статті розглядається проблема динамічної модифікації мапи шляхів з плином ігрового процесу, що дозволяє гравцю змінювати мапа шляхів розміщуючи різні ігрові об'єкти як перешкоди або новий шлях. Код представлений на мові програмування C#.
Ключові слова: ігри, 3d, булеві операції, мапа шляхів, c#, стратегії.

Гринюк С.В., Мельник В.М., Равенец А.В. Булевые операции над полигонами при создании карты путей в трехмерном пространстве для компьютерных игр. Игры жанра RTS используют 3d компонент Mesh для определения областей предназначенных для перемещения подвижных игровых единиц. Обычно это статическая карта которая не меняется со временем игры. В данной статье рассматривается проблема динамической модификации карты путей с течением игрового процесса, что позволяет игроку изменять карту путей размещая различные игровые объекты как препятствия или новый путь. Код представлен на языке програмирования C#.
Ключевые слова: игры, 3d, булевые операции, карта путей, c#, стратегии.

Hrynyuk S., Melnyk V., Ravenets A. Boolean operations between polygons for creating paths map in tree-dimensional space for computer games. RTS games uses 3d component Mesh for defining zones designed for moving mobile game objects. Often it is a static map which is not changing during the gameplay. In here we analyzing problem of dynamic modification of paths map during gameplay which allows player change paths map by placing different game objects as obstacles or new path. Code is written using C# programing language.
Keywords: games, 3d, boolean operations, paths map, c#, strategies.

Постановка проблеми. Жанр відеоігор, в якому запорукою досягнення перемоги є планування і стратегічне мислення називаються – стратегії. Основною характеристикою цього жанру є керування великою кількістю об'єктів як: солдати, військова техніка, фантастичні звірі і т.д. Для того щоб підконтрольний об'єкт зміг дійти до пункту призначення використовують різні алгоритми пошуку шляху. Одним із найвідоміших є алгоритм A*. Однак, для 3d середовищ є більш ефективний метод який називається Navigation Mesh. Основними перевагами цього алгоритму є те що він працює не тільки в одній площині та замість розподілу усієї мапи шляхів на клітини (Nodes) опуклі полігони цього компоненту виступають в ролі клітин, але це не означає що вони містять в собі лише один крок, вони є зонами в яких об'єкт може вільно пересуватись у різних площинах. Ми розглядаємо випадок коли гравець може вносити зміни у мапу. Будувати різні будівлі які містять зони по яким можна пересуватись або ж є перешкодами. Оскільки мапа представлена як 3d компонент Mesh нам потрібно реалізувати зміну шляху, а саме такі логічні операції як сума та різниця.

Аналіз досліджень. За основу взятий ігровий двигун Unity3d. 3d об'єкти у ньому представлені у вигляді масиву точок та масиву трикутників, які в свою чергу складаються з перелічених за годиниковою стрілкою трьох точок. Для виконання логічних операції був взятий алгоритм для полігонів [1]. Щоб реалізувати алгоритм нам потрібно утворити зв'язні контури використовуючи точки та трикутники. Після цього потрібно реалізувати процес триангуляції щоб вихідний об'єкт був коректний для використання у 3d.

Мета. Метою статті є успішне виконання булевих операцій уми та різниці для 3d компоненту Mesh на основі алгоритму для полігонів у 2d вимірі.

Виклад основного матеріалу й обґрунтування отриманих результатів.

Утворення контуру. Для того щоб виконати операцію суми чи різни, нам потрібно підготувати вхідні данні. Алгоритм опирається на полігон що являє собою список точок та список контурних ребер [1]. Тому оскільки список точок у нас уже є, потрібно використовуючи трикутники сформувати список контурних ребер (контур). Для початку підготуємо типи які ми надалі будемо використовувати. Основними будуть три типи:

1. Точка (Vertex)
2. Ребро (Edge)
3. Трикутник (Triangle)

Оскільки уже існуючий в Unity3d клас який представляє собою точку у 3d просторі Vector3 є структурою, ми створимо свій тип який буде класом. Для точки нам потрібно зберігати об'єкт Vector3, для ребра його дві точки, для трикутника - три токи і три ребра. Після того як ми сформували типи нам потрібно сформувати ці всі елементи. Точки у нас є тому ми просто використовуючи цикл формуєм список з нашими об'єктами Vertex. для трикутників виконуєм аналогічну операцію. Для ребер, нам потрібно з усіх утворених нами трикутників, вибрати такі ребра які зявляються лише в одному трикутнику, оскільки внутрішні ребра є суміжними для трикутників. Дані для контура готові. Головним моментом є те що ребра контура полігона повинні бути перераховані в списку за годинниковою стрілкою. Але дірки також виступають контуром для полігона тільки напрямок ребер проти годинникової стрілки.

Знаходження перетину ребер у просторі. Після того як ми сформували данні для двох контурів над якими ми будемо виконувати операції суми та різниці нам потрібно знайти усі можливі перетини між ними. Для початку нам потрібно визначити чи перетинаються ребра. Ми будемо використовувати матриці повороту для спрощення операцій з 3d виміру до 2d. Для підтвердження їхнього перетину нам достатньо довести наступне для кожного ребра:

- Кінцеві точки одного з ребер лежать в різних півплощинах які утворяться в результаті поділу площини на дві іншим ребром.

Ми знаємо що векторний добуток двох векторів дає нам третій вектор, напрямок якого залежить від того, додатній чи від'ємний кут між першим та другим векторами, відповідну така операція антикоммутативна. А так як вектори будуть лежати в двох осях (**перед перевіркою перетину нам потрібно повернути ребра на вісь Z так щоб усі точки мали однакове значення по осі Z щоб ми змогли ігнорувати це значення для подальших операцій та виконувати їх уже не в 3d, а в 2д просторі. А після отримання бажаного результату ми повернемо його в зворотньому напрямку на ті ж самі кути. Даний процес буде описаний нижче**) векторний добуток буде мати не нульову компоненту тільки Z.

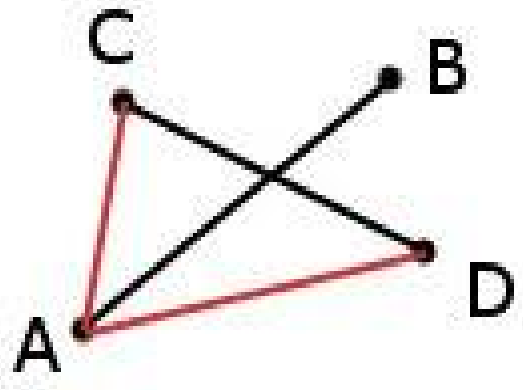

Рис. 1. Новоутворені вектори.

Відповідно відмінність векторних добутків буде тільки в цій компоненті. Тому ми можемо помножити попарно-векторно вектор ребра що розділяє площину на півплощини і нові вектора направлені від початку розділяючого вектора до обох кінці вектора що перевіряється. Якщо компоненти Z обох добутків буде мати різний знак, це означає що один з кутів менше 0, але не більше -180, а другий більше 0 і не менше 180 градусів. Відповідно точки лежать по різні сторони від прямої. Якщо ми маєм одну компоненту Z рівну нулю це означає що ребро дотикається іншого. Після цього на потрібно перевірити і інше ребро. Якщо ж два відрізка задовільняють усі умови то перетин існує.

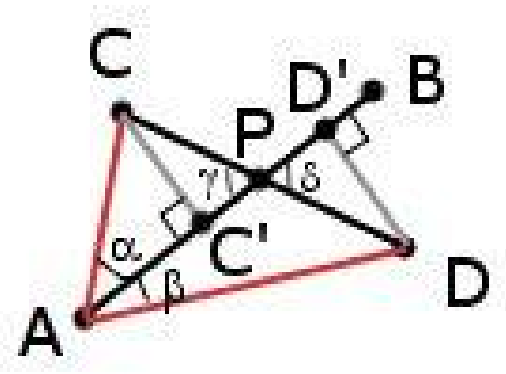

Рис 2. Висоти які є результатом скалярного добутку векторів.

Довжина векторного добутку векторів a і b (як ми з'ясували, що чисельно дорівнює його компоненті Z) дорівнює добутку модулів цих векторів на синус кута між ними.

$|a|\ |b|\ \sin ab$

Відповідно, для конфігурації на малюнку ми маємо наступне:

$$|AB \times AC| = |AB||AC| \sin\alpha, \text{ і } |AB \times AD| = |AB||AD| \sin\beta$$

$|AC| \sin\alpha$ є перпендикуляром, опущеним з точки C на відрізок AB, а $|AD| \sin\beta$ є перпендикуляром, опущеним з точки D на відрізок AB (катетом ADD'). Так як кути γ і δ - вертикальні кути, то вони рівні, а значить трикутники PCC' і PDD' подібні, а відповідно і довжини всіх їх сторін пропорційні в рівне ставлення.
Маючи $Z1$ ($AB \times AC$, а значить $|AB||AC| \sin\alpha$) і $Z2$ ($AB \times AD$, а значить $|AB||AD| \sin\beta$), ми можемо розрахувати $CC' \div DD'$ (яка дорівнюватиме $Z1 \div Z2$), а також знаючи що $CC' \div DD' = CP \div DP$ легко можна вирахувати місце розташування точки P. Ось кінцеві формули:

$$Px = Cx + (Dx - Cx) * |Z1| \div |Z2 - Z1|$$
$$Py = Cy + (Dy - Cy) * |Z1| \div |Z2 - Z1|$$

Вигляд методу на C#:
(Алгоритм виконується для попередньо повернутих ребер.)

```
public Vertex GetIntersection(Edge edge1, Edge edge2, ref bool bound)
{
        bool boundLocal = false;

        Vertex Vector1 = edge1.GetVector();
        Vertex Vector2 = edge2.GetVector();

        Vertex Prod1 = CrossProduct(Vector1, new Edge(
                edge1.Start,
                edge2.Start
        ).GetVector());

        Vertex Prod2 = CrossProduct(Vector1, new Edge(
                edge1.Start,
                edge2.End
        ).GetVector());
```

```
        if (!boundLocal && (Prod1.y == 0f || Prod2.y == 0f))
        {
                boundLocal = true;
        }

        // Граничні значення рахуються перетином
        if ((Prod1.y == 0f && Prod2.y == 0f) || Math.Sign(Prod1.y) == Math.Sign(Prod2.y))
        {
                return null;
        }

        Prod1 = CrossProduct(Vector2, new Edge(
                edge2.Start,
                edge1.Start
        .GetVector());

        Prod2 = CrossProduct(Vector2, new Edge(
                edge2.Start,
                edge1.End
        ).GetVector());

        if (!boundLocal && (Prod1.y == 0f || Prod2.y == 0f))
        {
                boundLocal = true;
        }

        // Граничні значення рахуються перетином
        if ((Prod1.y == 0f && Prod2.y == 0f) || Math.Sign(Prod1.y) == Math.Sign(Prod2.y))
        {
                return null;


        // Опрацювання ділення на нуль.
        if (Prod2.y == Prod1.y)
        {
                return null;
        }

        float x = edge1.Start.x + Vector1.x * Math.Abs(Prod1.y) / Math.Abs(Prod2.y - Prod1.y);
        float z = edge1.Start.z + Vector1.z * Math.Abs(Prod1.y) / Math.Abs(Prod2.y - Prod1.y);

        bound = boundLocal;

        return new Vertex(x, 0f, z);
}


public Vertex GetVector()
{
        return new Vertex(
                End.x - Start.x,
                End.y - Start.y,
                End.z - Start.z
        );
}

public Vertex CrossProduct(Vertex Vector1, Vertex Vector2)
{
        return new Vertex(
                (Vector1.y * Vector2.z - Vector1.z * Vector2.y),
                (Vector1.z * Vector2.x - Vector1.x * Vector2.z),
                (Vector1.x * Vector2.y - Vector1.y * Vector2.x)
        );
}
```

Поворот точок в просторі. Перед тим як виковувати поворот точки нам потрібно визначити

нормалію площини в якій лежать наш контур, що являє собою току в просторі яка належить лощині і через яку проходить перпендикуляр до цієї площини від початку координат (0, 0, 0). Ми не будемо зупинятися на цьому кроці, в мережі доступно багато способів визначення цієї точи. Коли ми отримали нормалію площини ми можемо визначити кут для повороту площини спочатку по осі Y, а потім по осі X. Для першого випадку це буде косинус кута між проекцією перпендикуляра на осі X-Z і відрізком що лежить на осі Z. Для другого випадку косинус кута між перпендикуляром та відрізком що лежить на осі Y. Після того як ми за допомогою векторів знайшли косинуси кутів, нам потрібно дізнатися їх синус, але з правильним знаком. Синус ми знайдемо легко за допомогою формули:

$$\cos\alpha + \sin\alpha = 1$$

А знак ми дізнаємось з того в якій четверті лежать проекція перпендикуляра на осі X-Z відносно цих осей і проекція перпендикуляра на осі Y-Z відносно цих осей для знайдених косинусів відповідно.

Якщо проекції лежать на одній прямій з відповідними осями до яких визначається кут або лежать в четвертях 2 і 3 тоді напрямок за годинниковою стрілкою, тобто знак плюс. Якщо ж вони лежать в четвертях 1 і 4 тоді напрямок проти годинникової стрілки тобто знак мінус. На даному етапі ми знайшовши два кути по яких ми будемо виконувати поворот точок площини. Данні значення варто обчислити один раз і зафіксувати для усіх компонентів що знаходяться в тій самій площині. Значення синусів і косинусів використовуються у формулах матриць повороту.

Маркування ребер. Перед тим як розпочати маркування ребер, нам потрібно розбити ребра які містять в собі точки перетину для двох контурів. Далі слід маркувати ребра, для цього ми будемо використовувати два показника для кожного ребра fl і fr. Для усіх ребер з першого контуру присвоюєм значення fl = 1, а fr = 0. Для усіх ребер з другого контуру які ще не марковані присвоюєм значення fl = 2, а fr = 0. Якщо ребро з другого контуру існує в першому у нас є два варінта. Якщо їх напрямок співпадає (напрямок можна перевірити по точкам ребер, start і end) то ми збільшуєм для нього fl на 2 (отримуєм fl = 3 fr = 0), якщо напрямок різний присвоюєм fr = 2 (отримуєм fl = 1 fr = 2). Після цього нам потрібно опрацювати ребра які знаходяться всередині контурів. Для цього ми використаєм алгоритм належності точки полігону (**описаний далі**). Для всіх ребер з першого контуру перевіряєм чи існує таке ж ребро не зважаючи на напрямок в другому контурі. Якщо ні знаходим центральну точку ребра і перевіряєм чи належить вона другому контуру. Якщо вона належить збільшуєм показники fl і fr ребра на 2 (отримаєм fl = 3 fr = 2. було fl = 1 fr = 0). Теж саме виконуєм аналогічно для ребер другого контуру тільки якщо умови виконуються збільшуєм показники не на 2 а на 1 (отримаєм fl = 3 fr = 1. було fl = 2 fr = 0). На даному етапі усі ребра марковані нам залишилось тільки вибрати ті які потрібні за допомогою таблиці:

Таблиця 1. Вибірка ребер

fl	fr	Сума	Переріз	Різниця	Симетрична різниця
1	0	+	-	+	+
2	0	+	-	-	+
1	2	-	-	+	-
3	0	+	+	-	-
3	1	-	+	<>	<>
3	2	-	+	-	<>

Позначення в таблиці:
+ додаєм ребро до нового контура

- не додаєм ребро до нового контура

<> додаєм ребро до нового контура, але змінюєм його напрямок

Опираючись на таблицю утворюєм нові контури визначаєм чи виступає якийсь контур діркою, та формуємо новий полігон. Ось і все, на даному етапі маємо новий полігон що є результатом булевої операції над двома іншими полігонами.

1. Леонов М. В., Никитин А. Г. Эффективный алгоритм, реализующий замкнутый набор булевых операций над множествами многоугольников на плоскости / Препринт Института систем информа- тики СО РАН № 46. 1997. — 20 с.
2. Скворцов А.В. Построение объединения, пересечения и разности произвольных многоугольников в среднем за линейное время с помощью триангуляции // Вычислительные методы и программирование, 2002, Т. 3. С. 116-123.
3. Скворцов А. В. Линейно-узловой алгоритм построения оверлеев двух полигонов // Вестник ТГУ. 2002. № 275. С. 99—103.
4. Ченцов О.В., Скворцов А.В. Обзор алгоритмов построения оверлеев многоугольников // Вестник ТГУ. 2003. № 280. С. 338–345.