Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ *Робототехника и комплексная автоматизация*

КАФЕДРА *Системы автоматизированного проектирования (РК-6)*

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ

«Разработка системы сбора и исследования информации о фондовом рынке на Unreal Engine 4»

Студент РК6-81Б — М.А.Афиногенов
(Группа) (подпись, дата) (инициалы и фамилия)

Руководитель КП — Ф.А.Витюков
(подпись, дата) (инициалы и фамилия)

Москва, 2024 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой *РК-6*
(индекс)

_____ *А.П. Карпенко*
(инициалы и фамилия)

«___» _____ *2024* г.

З А Д А Н И Е
на выполнение НИРС

Студент группы РК6-81Б

Афиногнеов Михаил Алексеевич
(фамилия, имя, отчество)

Тема НИРС

Разработка системы сбора и исследования информации о фондовом рынке на Unreal Engine 4

Источник тематики (кафедра, предприятие, НИР): НИР

Тема утверждена распоряжением по факультету РК № _____ от «___» _____ *2023* г.

Часть 1. *Аналитический обзор литературы*

В рамках аналитического обзора литературы должны быть изучены методы, применяемые для оптимизации отрисовки большого количества элементов на сцене.

Часть 2. П*рограммная реализация*

Должен быть разработан определенный функционал, позволяющий быстро и эффективно отобразить графическое представление свечного финансового графика в рамках Unreal Engine 4.

Часть 3. *Отладка и тестирование*

Отладка и тестирование функционала должно проводиться с использованием самодельного программного обеспечения, необходимо достичь удовлетворительных показателей производительности программы при выполнении определенных действий.

Оформление выпускной квалификационной работы:

Расчетно-пояснительная записка на … листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.):

4 графических листа

Дата выдачи задания «03» октября 2023 г.

Студент М.А.Афиногенов

(Подпись, дата) (И.О.Фамилия)

Руководитель выпускной квалификационной работы Ф.А.Витюков

(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

РЕФЕРАТ

Работа посвящена исследованию функционала игрового движка Unreal Engine 4, позволяющего реализовать программу по исследованию и сбору информации о финансовых рынках. В работе будут рассмотрены возможности оптимизации отрисовки и представления финансовых графиков, методы и инструменты исследования финансовых графиков и примеры разработанных функциональных интерактивных элементов, позволяющих осуществлять анализ и сбор информации с представленных пользователю графиков.

Тип работы: НИРС.

Тема работы (проект темы): *Исследование методов оптимизации отрисовки трехмерных объектов на базе игрового движка Unreal Engine 4 и разработка функционала для исследования свечных финансовых графиков.*

Объект исследований: *Свечные финансовые графики.*

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современных игровых движках, таких как Unreal Engine, оптимизация отрисовки графики является ключевым аспектом, определяющим производительность и качество визуального представления. Instance Static Mesh Component (ISM) предоставляет мощный механизм для оптимизации путем инстанцирования (instancing), что позволяет значительно снизить количество вызовов отрисовки (draw calls) и уменьшить нагрузку на центральный процессор (CPU) и графический процессор (GPU). В этом документе рассматриваются принципы работы ISM, механизмы его функционирования и степень, до которой можно оптимизировать отрисовку.

В рамках выпускной квалификационной работы (ВКР) необходимо реализовать отрисовку большого количества однотипных трехмерных объектов – в данном случае финансовых свечей. Для их грамотного представления реализованы 2 подтипа объектов: тело свечи и тени свечи. В данной конкретной работе используются подготовленные графики в формате CSV, данные из которых используются для отрисовки свечей. В свою очередь графики представлены в виде таблиц, где одна строка – одна свеча. Из таблицы получаются значения минимального и максимального значения свечи, точки открытия и закрытия свечи, ее время открытия и закрытия. В таблице 250 строк – соответственно, на выходе будет реализовано 250 свечей в рамках одного графика. Каждая свеча представлена 2 конструктивными элементами, и суммарно получается, что для того, чтобы отрисовать один график, необходимо обработать информацию по 250-и записям в таблице, после чего быстро отрисовать 500 элементов графика.

Основные проблемы с которыми можно столкнуться при решении данной задачи – это большая нагрузка на компьютер как при разработке, так и при использовании готовой работы, так как имеется необходимость в быстрой отрисовке большого количества элементов, а также необходимость быстрой и четкой обработки данных с полученных графиков. Обработка графиков была решена посредством перевода графиков, полученных с сайта, в формат CSV -

Comma-Separated Values — значения, разделённые запятыми) — текстовый формат, предназначенный для представления табличных данных. Строка таблицы соответствует строке текста, которая содержит одно или несколько полей, разделенных запятыми. – а затем при помощи специальной структуры данных преобразована в таблицу с данными, где одна строка соответствует одной свече на графике. Большую ресурсоемкость задачи было решено оптимизировать с помощью функционала игрового движка Unreal Engine 4, в частности с использованием технологии Instance Static Mesh Component.

1. ПОСТАНОВКА ЗАДАЧИ

1.1.Концептуальная постановка задачи

Объект исследования – методы отрисовки свечных финансовых графиков.

Цель исследования – оптимизировать отрисовку большого количества однотипных элементов на сцене в Unreal Engine 4.

Требуется получить – определенный функционал, используемый для отрисовки большого количества однотипных элементов.

2. АРХИТЕКТУРА ПРОГРАММНОЙ РЕАЛИЗАЦИИ

В данном случае Instance Static Mesh Component является наиболее оптимальным решением, так как эта технология позволяет объединить некоторое количество вызовов отрисовки (Draw Calls) в один, что снижает нагрузку на видеокарту и повышает скорость исполнения необходимых вызовов отрисовки.

Instance Static Mesh Component - это компонент, который используется для отрисовки повторяющихся объектов в игровом движке. Он позволяет создавать копии одного и того же статического меша на сцене без дополнительного расхода памяти на каждую копию. Это особенно важно при отрисовке большого количества объектов, так как позволяет существенно уменьшить нагрузку на процессор и GPU. Instance Static Mesh Component повышает производительность игры и позволяет создавать более крупные и детализированные сцены без потери в качестве графики.

Среди принципов работы Instance Static Mesh Component можно отметить следующее:

1. Группировка объектов и их отрисовка - в традиционной системе отрисовки каждый объект на сцене требует отдельного draw call, что означает отправку команды на GPU для отрисовки этого объекта. При большом количестве объектов, особенно идентичных, этот процесс создает значительную нагрузку на CPU, так как каждый draw call требует обработки и подготовки данных. Instance Static Mesh Component позволяет объединить все экземпляры одного и того же меша в один draw call. Это достигается путем создания одного набора данных меша, который используется для всех экземпляров, а также таблицы трансформаций, определяющей позицию, вращение и масштаб каждого экземпляра.
2. Снижение числа draw calls - Draw call — это команда, отправляемая от CPU к GPU для рендеринга конкретного объекта или группы объектов. При большом количестве объектов draw calls становятся узким местом в производительности, так как каждая команда требует значительных

ресурсов для обработки. Использование ISM позволяет сократить количество draw calls с тысяч до единиц. Например, вместо 1000 draw calls для 1000 объектов, ISM позволяет отправить всего 1 draw call, что значительно уменьшает нагрузку на CPU и увеличивает общую производительность.

3. Индивидуальные параметры трансформации - Одним из преимуществ ISM является возможность задания индивидуальных параметров трансформации для каждого экземпляра объекта. Это включает в себя позицию, вращение и масштаб, что позволяет создавать визуально разнообразные сцены с использованием одного и того же меша. Эти трансформации хранятся в виде таблицы, которая используется GPU для применения соответствующих изменений к каждому экземпляру во время отрисовки.

Использование Instance Static Mesh Component (Instanced Static Mesh или ISM) в Unreal Engine позволяет значительно оптимизировать отрисовку множества одинаковых объектов на сцене. Это достигается за счет использования техники instancing, которая уменьшает нагрузку на процессор и видеокарту. Вот основные аспекты, как это работает и какие выгоды это дает:

1. Группировка объектов: Вместо того чтобы обрабатывать каждый объект отдельно, Instance Static Mesh Component позволяет объединить все экземпляры одного и того же меша в один draw call. Draw call — это команда, отправляемая на GPU для отрисовки объектов. С уменьшением числа draw call, уменьшается и нагрузка на процессор. Чем больше количество экземпляров одного и того же меша на сцене, тем больше выгода от использования ISM. В сценах с тысячами идентичных объектов (например, леса, толпы, здания) уменьшение количества draw calls может быть значительным, приводя к существенному увеличению производительности.
2. Сокращение Draw calls: Если у вас есть 1000 одинаковых элементов, которые нужно отрисовать, использование Instance Static Mesh вместо

стандартного Static Mesh сократит количество draw call с 1000 до 1 (в идеальном случае).

3. Параметры трансформации: Instance Static Mesh позволяет каждому экземпляру объекта иметь свою трансформацию (позиция, вращение, масштаб), что делает каждое дерево уникальным, но все они обрабатываются одной командой на GPU.

Сложные меши и материалы требуют больше ресурсов для обработки и рендеринга. При использовании ISM выигрыш в производительности будет более заметен для сложных объектов, так как каждый draw call требует значительных ресурсов для подготовки и обработки данных. Для простых мешей выигрыш может быть менее заметен, но все равно ощутим.

Оптимизация отрисовки с использованием Instance Static Mesh зависит от многих факторов, но вот некоторые примеры и ориентировочные оценки:

1. Количество экземпляров: Чем больше количество экземпляров, тем больше выгода от использования Instance Static Mesh. Например, для сцены с 1000 и более одинаковых объектов, снижение количества draw calls может быть весьма значительным.
2. Сложность объектов: Если меши простые, то выигрыш может быть менее заметным, так как процессор и так быстро обрабатывает команды. Для более сложных объектов и сложных материалов выгода будет более ощутимой.
3. Тип сцены: В сценах, где множество мелких деталей (например, леса, городские сцены), использование Instance Static Mesh позволит существенно улучшить производительность.
4. GPU и CPU: Современные GPU отлично справляются с instancing, и в большинстве случаев производительность возрастет на порядок, если использовать Instance Static Mesh Component для повторяющихся объектов.

Также использование Instance Static Mesh Component обеспечивает положительное влияние на компоненты рабочей машины пользователя и разработчика, так как:

1. Происходит снижение времени на CPU - Оптимизация в плане снижения нагрузки на CPU может достигать 70-90%, так как уменьшается количество draw call и пересчетов трансформаций. В традиционной системе рендеринга каждый draw call требует значительных ресурсов для подготовки и отправки данных на GPU. При большом количестве объектов это становится узким местом, ограничивающим производительность. ISM позволяет сократить количество draw calls, что снижает нагрузку на CPU и освобождает ресурсы для других задач, таких как обработка логики игры и физики.
2. Происходит снижение времени на GPU: На GPU можно ожидать улучшения производительности в диапазоне от 20% до 80%, в зависимости от сложности сцены и количества объектов. Современные GPU оптимизированы для работы с большими объемами данных и параллельной обработкой. Техника инстанцирования использует эти возможности, позволяя GPU обрабатывать множество объектов за один цикл отрисовки. Это достигается путем передачи одного набора данных меша и таблицы трансформаций, что уменьшает объем данных, передаваемых между CPU и GPU, и снижает количество draw calls.
3. Инстанцирование позволяет более эффективно использовать кэш и память GPU, так как данные меша хранятся в едином экземпляре и повторно используются для всех экземпляров. Это уменьшает количество операций чтения-записи и повышает общую производительность системы.

Instance Static Mesh Component также обладает удобным интерфейсом для управления и настройки каждой копии меша на сцене. Это позволяет программистам и художникам быстро и эффективно создавать сложные сцены с большим количеством деталей. Кроме того, использование данного компонента

позволяет легко изменять параметры каждой копии меша, такие как масштаб, поворот, положение, что делает процесс настройки сцены более гибким и удобным.

Еще одним преимуществом Instance Static Mesh Component является его эффективное управление памятью и ресурсами. Поскольку компонент использует один и тот же меш для всех копий, это существенно снижает объем занимаемой памяти и ускоряет процесс загрузки игры. Благодаря этому игровые сцены могут быть более крупными и детализированными, не теряя при этом в производительности.

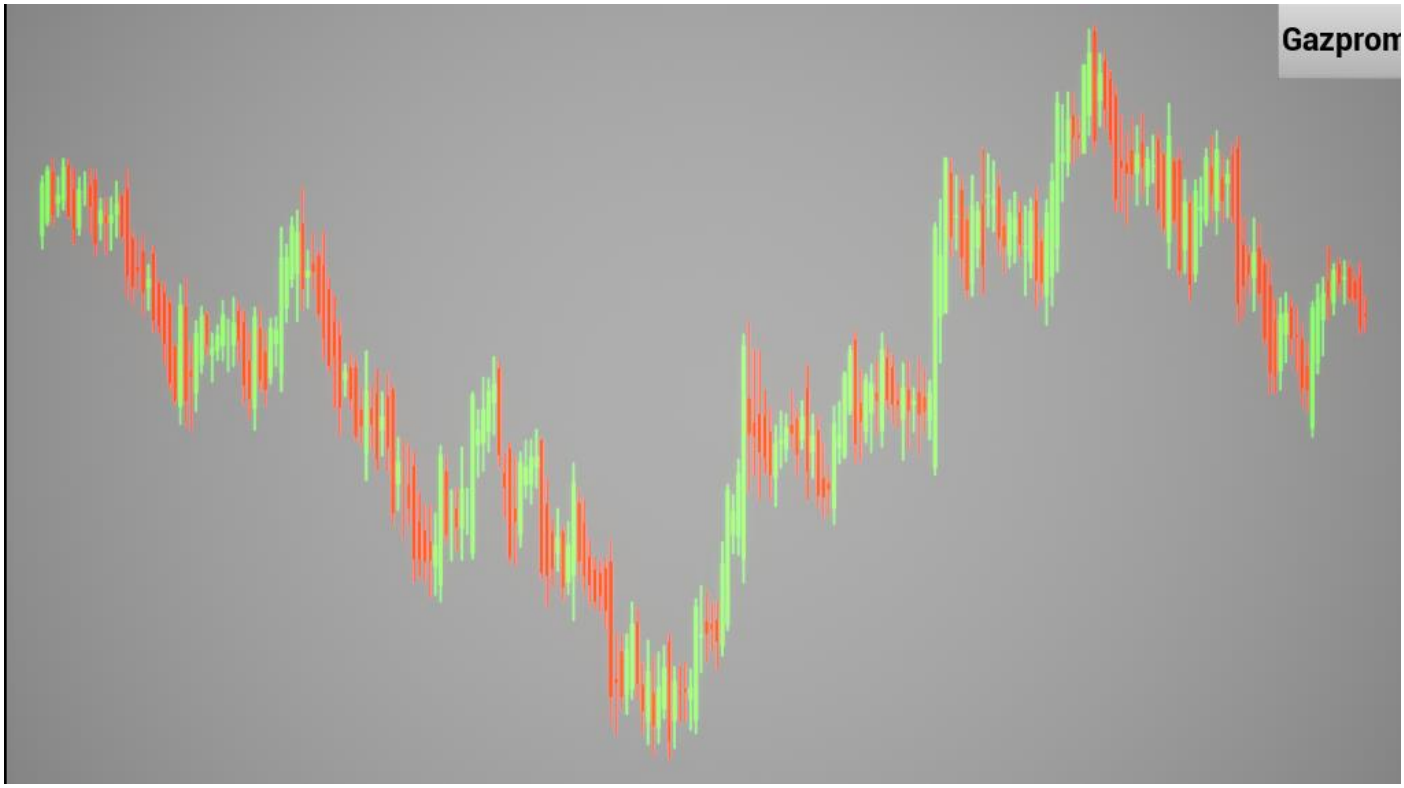


Рисунок 1. Пример использования Instance Static Mesh Component

Также технология Instance Static Mesh Component позволяет работать со Static Mesh Component-ами, входящими в множество одинаковых элементов отрисовки, по отдельности, что позволяет присваивать каждому элементу Instance Static Mesh Component свой цвет, положение на сцене, размер, масштаб, прозрачность и так далее.

Еще одно удобство технологии Instance Static Mesh Component заключается в том, что общий набор одинаковых элементов можно представить

в виде массива, в котором можно добавлять, удалять и редактировать объекты по своему усмотрению. На этой возможности основана реализация смены отображаемых свечных графиков в рамках данной работы.

3. ТЕСТИРОВАНИЕ И ОТЛАДКА

В ходе данной работы необходимо реализовать грамотное отображение и размещение финансовых свечных графиков с использованием интерактивных элементов и средств отображения игрового движка Unreal Engine 4.

В первую очередь необходимо было подобрать и отформатировать данные по финансовому состоянию акций некоторых крупных компаний, в данном случае APPLE, Камаз и ГазПром. Для этого были скачаны из общего доступа графики акций вышеперечисленных компаний и переведены в формат CSV – формат файла, в котором данные разделяются запятой либо точкой с запятой. Данные отформатированные файлы необходимо было загрузить в Unreal Engine 4, для чего были созданы соответствующие таблицы для размещения там информации по графикам в понятной и легкодоступной форме, а также специальная структура, благодаря которой CSV-файлы были успешно разделены на блоки данных и размещены в соответствующие строки таблиц.

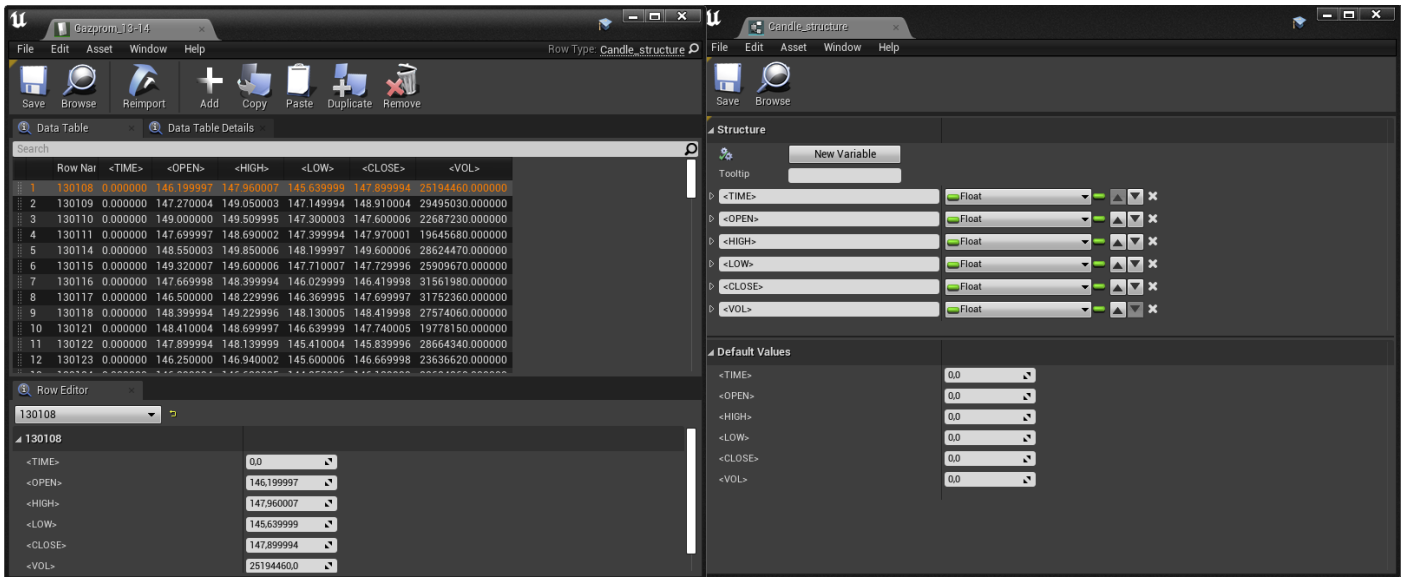


Рисунок 2. Пример размещения данных в таблице (слева) и специальная структура для размещения данных (справа)

Для отрисовки данных из таблицы на сцене необходимо было реализовать следующее:

1. Выбор необходимой таблицы в зависимости от того, какой график выберет пользователь в выпадающем меню выбора графика.

Реализовано данное решение было с использованием интерактивных виджетов, в частности с использованием ComboBox, являющим собой настраиваемый выпадающий список предложенных элементов.

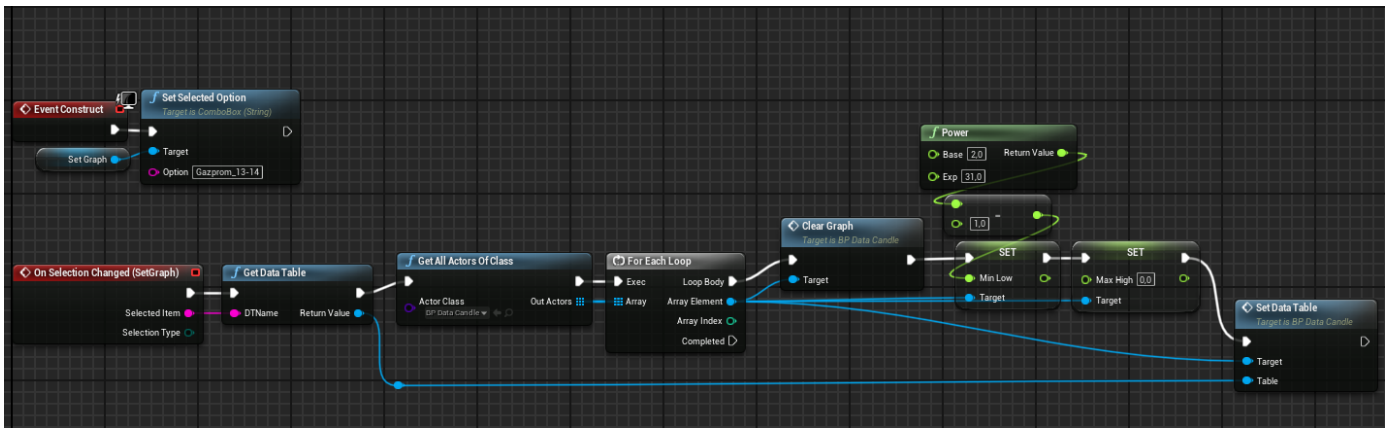


Рисунок 3. Графическое представление алгоритма действий при выборе пользователем отображаемого графика

Данный алгоритм реализует следующее:

1. При создании виджета (то есть при запуске программы) автоматически выбирается предложенный вариант – в данном случае будет отрисован график акций ГазПром.
2. При смене пользователем желаемого отображаемого графика запускается событие смены графика (On Selected Change (SetGraph)), в котором происходит чтение из элемента виджета ComboBox значения, которое было в нем установлено пользователем, на основании полученного значения по имени таблицы выбирается таблица, из которой будет происходить чтение данных, после чего сама таблица передается на вход функции SetDataTable, которая, в свою очередь, была разработана для отрисовки свечного графика из таблицы, поступающей на вход как аргумент функции. До вызова функции SetDataTable алгоритм вызывает функцию, подготавливающую сцену для отрисовки нового свечного графика – функцию ClearGraph, которая очищает массив Instance Static Mesh Component, тем самым удаляя более не использующиеся данные со сцены, а также очищая саму сцену. Также в функцию SetDataTable

передаются константы, необходимые для просчета значений для дальнейшего определения масштаба и размещения свечей графика.

3. Чтение данных из таблицы и заполнение необходимых для построения констант – общая высота свечей, необходимая для того, чтобы масштаб свечей по вертикали соответствовал соотношению масштабов самих свечей; Ширина свечей, необходимая для правильного отображения свечей как отдельных друг от друга элементов, с учетом отступов и количества самих свечей. Для этого была было разработано следующее:
    1. При вызове функции SetDataTable происходит чтение полученной как аргумент функции таблицы с данными о свечном графике, затем каждая строка разбивается на отдельные значения переменных в соответствии со структурой строки таблицы данных, после чего происходит вызов функции просчета горизонтального масштаба свечей графика и их размещения (SetCandlePositions).

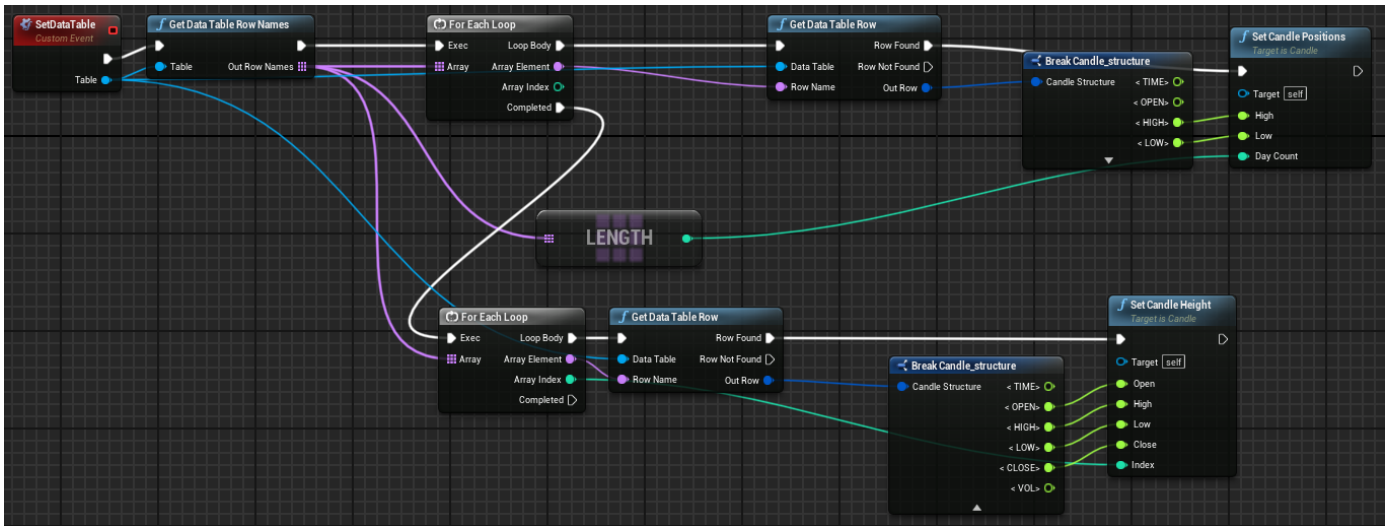


Рисунок 4. Графическое представление функции SetDataTable

2. При вызове функции SetCandlePositions происходит следующее – функция принимает как аргументы крайние точки теней свечей и количество свечей, затем происходит расчет констант, необходимых для соблюдения масштаба свечей и их взаимного расположения, после чего свечи размещаются на одной высоте с соблюдением своих масштабов и отступов свечей друг от друга.

Листинг 9 – файл Candle.cpp – функция SetCandlePositions

```
void ACandle::SetCandlePositions(float _high, float _low, int
_dayCount)
{
    //time = _time;
    //open = _open;
    high = _high;
    low = _low;
    //close = _close;
    //volume = _volume;
    dayCount = _dayCount;

        if (high > maxHigh)
        {
            maxHigh = high;
        }
        if (low < minLow)
        {
            minLow = low;
        }

        // candle placement here
        candleWidth = 100 * 0.8f / dayCount;
        FVector StartPos = FVector(-100*0.4f/2.f, -100*0.8f/2.f,
0.f);
        FTransform newCandle = FTransform(FRotator(0.f, 0.f,
0.f), FVector(StartPos.X + 0.f, StartPos.Y + CandleBody-
>GetInstanceCount() * candleWidth + candleWidth/2.f, 0.03f),
FVector(candleWidth/100.f, candleWidth/100.f - 0.001f, 1.f));
        FTransform newCandleLine =
FTransform(newCandle.GetRotation(), newCandle.GetLocation() +
/*FVector(-20.f, -50.f, 0.f)*/FVector(0.f, 0.f, 0.f),
FVector(candleWidth / 100.f, candleWidth / 200.f - 0.0005f, 1.f));

        CandleBody->AddInstance(newCandle);
        CandleLine->AddInstance(newCandleLine);
}
```

После завершения цикла горизонтального размещения свечей графика для каждой строки таблицы с данными о графике вызывается функция SetCandleHeight, отвечающая за размещение свечей во вертикали с соблюдением масштабов сцены и взаимного расположения и масштаба свечей графика.

Листинг 10 – файл Candle.cpp – функция SetCandleHeight

```
void ACandle::SetCandleHeight(float _open, float _high, float
_low, float _close, int index)
{
// counting candle body's placement
float XCandleCenter = (_open + _close) / 2.f;

float XCandleDataPercent = (XCandleCenter - minLow) * 100.f /
(maxHigh - minLow);
float XCandleCoordinates = (100.f * 0.4f * XCandleDataPercent) /
100.f;

float XCandleDataScale;
```

```
8. if (_open > _close)
9. {
10.     XCandleDataScale = (_open - _close) * 100.f / (maxHigh -
   minLow);

11.     CandleBody->SetCustomDataValue(index, 0, 1.f);
12.     CandleBody->SetCustomDataValue(index, 1, 0.f);

13.     CandleLine->SetCustomDataValue(index, 0, 1.f);
14.     CandleLine->SetCustomDataValue(index, 1, 0.f);
15.     }
16.     else
17.     {
18.     XCandleDataScale = (_close - _open) * 100.f / (maxHigh -
   minLow);

19.     CandleBody->SetCustomDataValue(index, 0, 0.f);
20.     CandleBody->SetCustomDataValue(index, 1, 1.f);

21.     CandleLine->SetCustomDataValue(index, 0, 0.f);
22.     CandleLine->SetCustomDataValue(index, 1, 1.f);
23.     }
24.     float XCandleCoordinatesScale = ((100.f * 0.4f *
   XCandleDataScale) / 100.f) / (100.f * 0.4f);

25.     // counting candle line's placement
26.     float XCandleLineCenter = (_high + _low) / 2.f;

27.     float XCandleLineDataPercent = (XCandleLineCenter - minLow)
   * 100.f / (maxHigh - minLow);
28.     float XCandleLineCoordinates = (100.f * 0.4f *
   XCandleLineDataPercent) / 100.f;

29.     float XCandleLineDataScale = (_high - _low) * 100.f /
   (maxHigh - minLow);
30.     float XCandleLineCoordinatesScale = ((100.f * 0.4f *
   XCandleLineDataScale) / 100.f) / (100.f * 0.4f);

31.     // setting everything in their places
32.     FTransform oldBodyTransform;
33.     CandleBody->GetInstanceTransform(index, oldBodyTransform,
   false);
34.     CandleBody->UpdateInstanceTransform(index,
   FTransform(oldBodyTransform.GetRotation(),
   oldBodyTransform.GetLocation() + FVector(XCandleCoordinates, 0.f,
   0.f), FVector(XCandleCoordinatesScale,
   oldBodyTransform.GetScale3D().Y,
   oldBodyTransform.GetScale3D().Z)));

35.     FTransform oldLineTransform;
36.     CandleLine->GetInstanceTransform(index, oldLineTransform,
   false);
37.     CandleLine->UpdateInstanceTransform(index,
   FTransform(FRotator(0.f, 0.f, 0.f), oldLineTransform.GetLocation()
   + FVector(XCandleLineCoordinates, 0.f, 0.f),
   FVector(XCandleLineCoordinatesScale,
   oldLineTransform.GetScale3D().Y,
   oldLineTransform.GetScale3D().Z)));
38. }
```

В данной функции происходит определение цвета свечи посредством сравнения значений точек открытия и закрытия торгов – если

значение точки открытия торгов за определенный промежуток времени было меньше значения точки закрытия торгов, то такая свеча будет иметь зеленый цвет, что является символом положительной тенденции, в противном случае свеча будет окрашена в красный цвет. Далее происходит расчет вертикального положения каждой свечи, для чего необходимо рассчитать процентное замощение тела и теней свечи пространства необходимого графика, затем определяется общее расстояние сцены по вертикали для того, чтобы через пропорцию преобразовать процентное замощение пространства свечой в ее величину в координатах Unreal Engine. Подобным образом вычисляется масштаб тела свечи, длина и масштаб теней свечи и их взаимное положение на сцене.

4. Отрисовка свечей с соблюдением их взаимного размещения, масштаба, расчеты высоты теней свечей и тел свечей, присвоение свечам определенного цвета в зависимости от данных, полученных из таблицы. Это происходит посредством использования технологии Instance Static Mesh Component, заполнение которого идет подобно массиву – в данном случае свеча, заранее помещенная в массив Instance Static Mesh Component (CandleBody предназначен для тел свечей, CandleLine – для теней свечей, соответственно), может быть изменена посредством обращения к ней по ее индексу в структуре Instance Static Mesh Component и непосредственному внесению изменений через специальные команды. В ходе выполнения функций SetCandlePositions и SetCandleHeight поэлементно вносятся изменения в каждое тело и тени свечи графика, вследствие чего все представленные в структуре массива элементов Instance Static Mesh Component свечи будут отрисованы за меньшее количество DrawCall-ов, являя собой структуру идентичных друг другу объектов, но, в то же время, каждая свеча будет самостоятельной единицей представления графической информации о финансовом графике.

ЗАКЛЮЧЕНИЕ

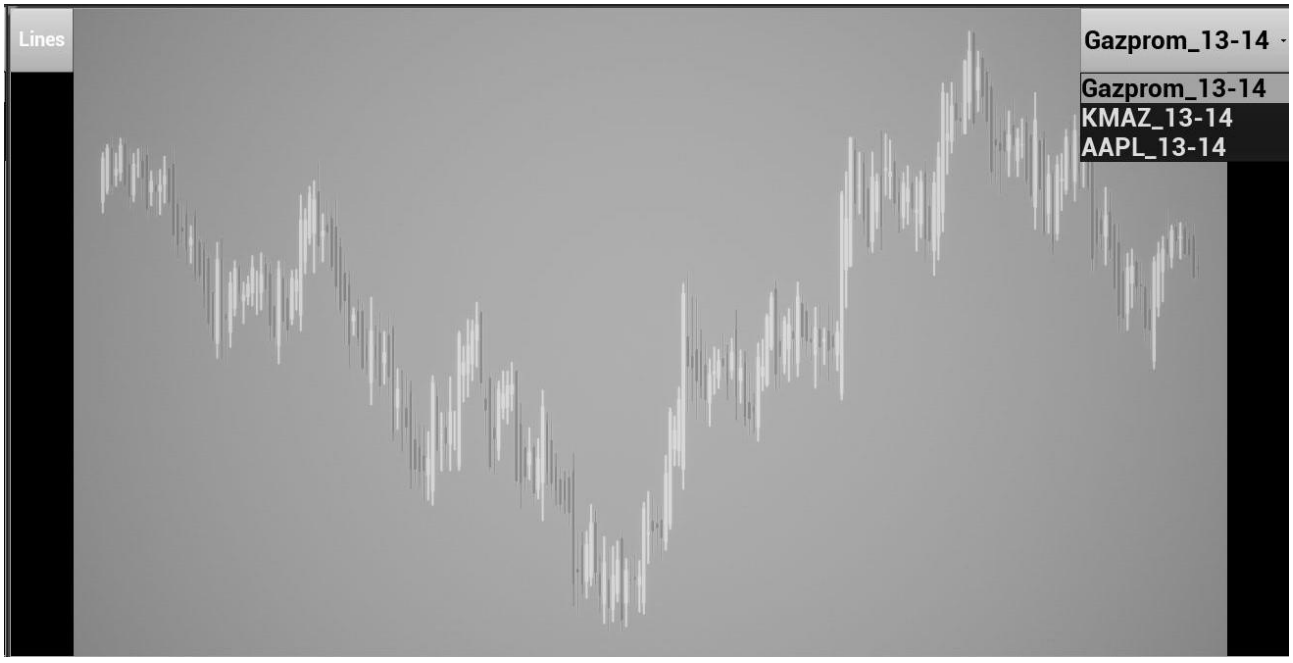
Instance Static Mesh Component в Unreal Engine является мощным инструментом для оптимизации отрисовки, особенно в сценах с большим количеством идентичных объектов. Техника инстанцирования позволяет значительно уменьшить количество draw calls, снизить нагрузку на CPU и улучшить общую производительность системы. Оптимизация отрисовки с использованием ISM зависит от множества факторов, включая количество экземпляров, сложность объектов, тип сцены и характеристики оборудования. В большинстве случаев использование ISM приводит к значительному улучшению производительности, что делает его ключевым инструментом для создания высокоэффективных и визуально насыщенных сцен в современных играх.

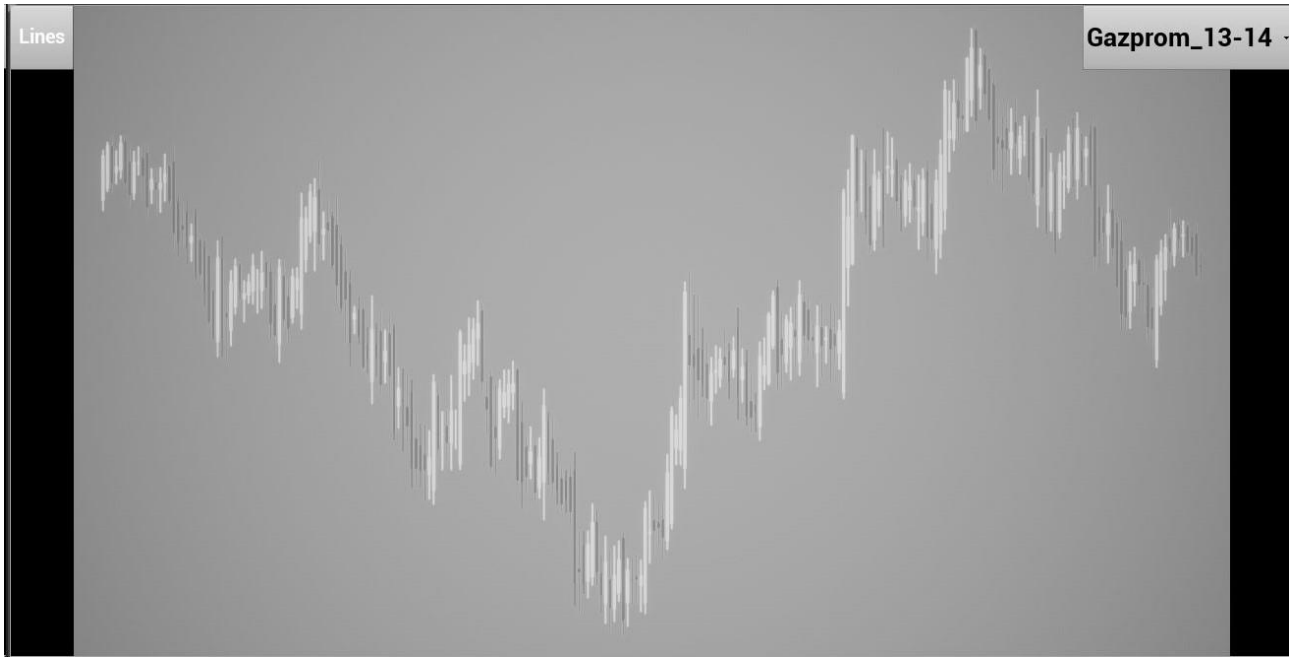
В сценах, содержащих множество мелких деталей и объектов, таких как леса, городские ландшафты или крупные массовки, использование ISM может значительно улучшить производительность. Такие сцены обычно требуют большого количества draw calls, что значительно нагружает CPU и GPU. Использование ISM позволяет уменьшить количество draw calls и повысить эффективность обработки.

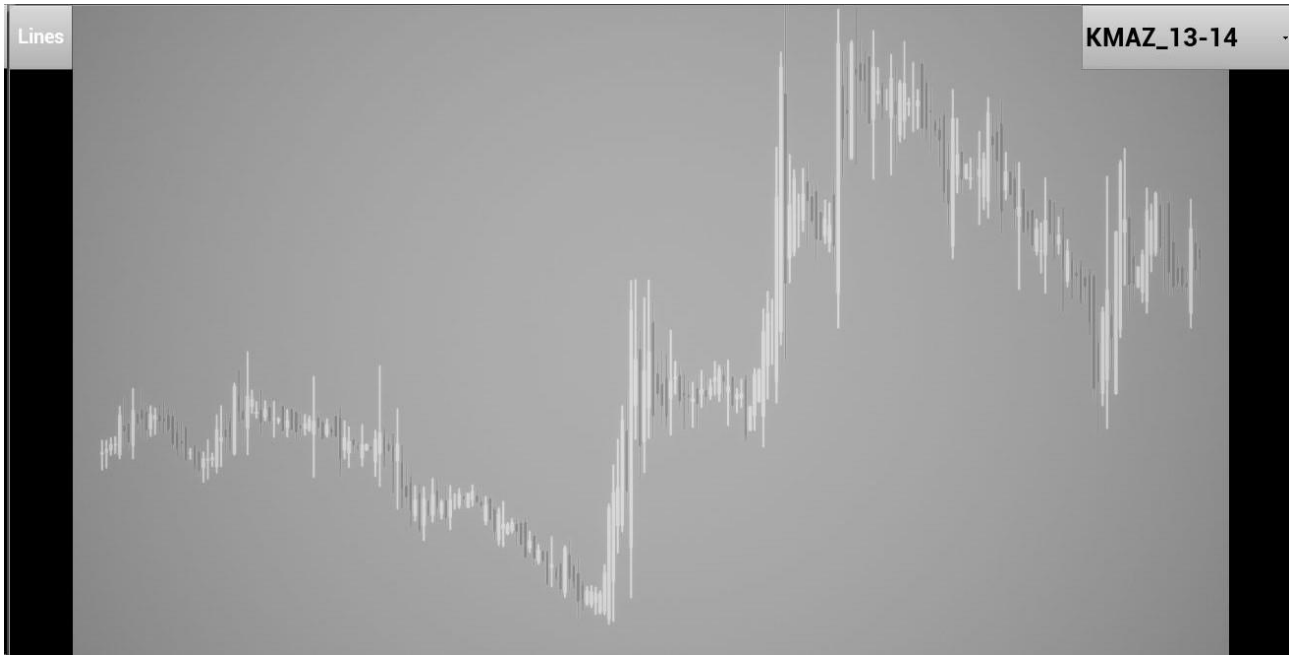
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

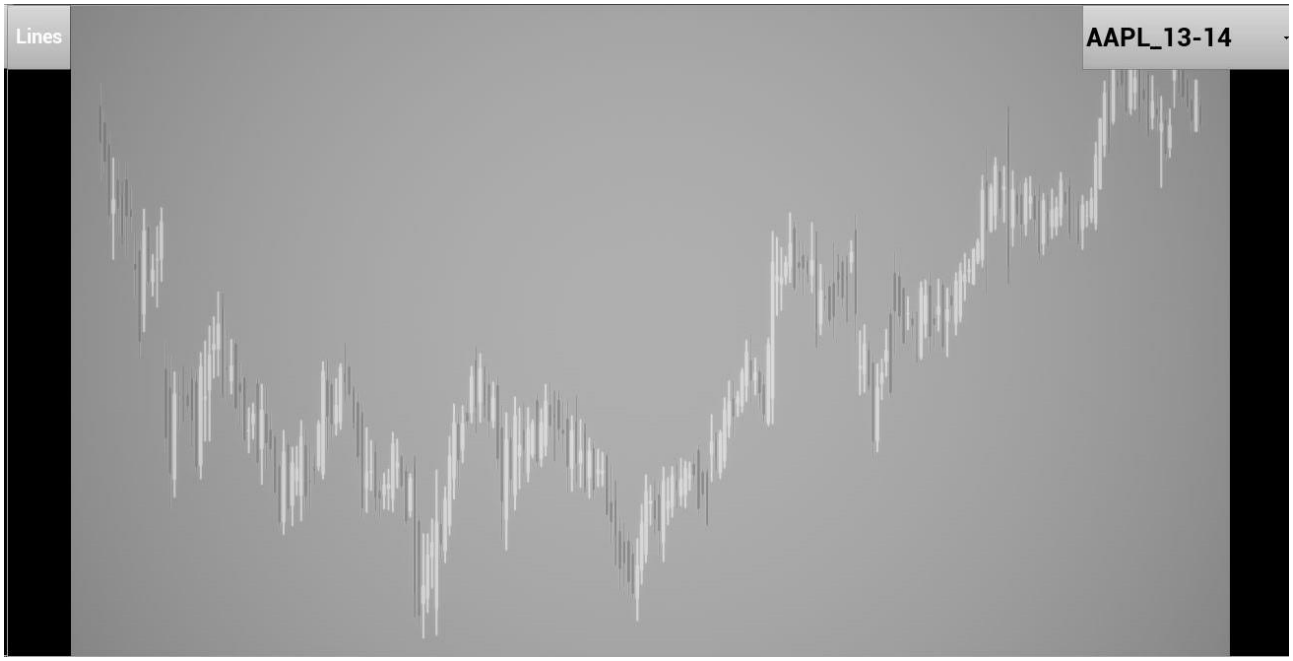
1. Instanced Static Mesh Component // Unreal Documentation URL : https://dev.epicgames.com/documentation/en-us/unreal-engine/API/Runtime/Engine/Elements/SMInstance?application_version=5.1. Дата обращения : 17.04.2024.
2. Экспорт финансовых графиков в формате CSV // Tradingview site URL: https://www.tradingview.com/blog/ru/export-chart-data-in-csv-14395/. Дата обращения: 13.04.2024.

ПРИЛОЖЕНИЕ А






Lines
KMAZ_13-14


Lines
AAPL_13-14