МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по научно-исследовательской работе

Тема: Генератор датасетов для задач машинного зрения

Студент гр. 6304 _____ Пискунов Я.А.

Руководитель _____ Заславский М.М.

Санкт-Петербург

2021

ЗАДАНИЕ

НА НАУЧНО-ИССЛЕДОВАТЕЛЬСКУЮ РАБОТУ

Студент Пискунов Я.А.

Группа 6304

Тема НИР: Генератор датасетов для задач машинного зрения

Задание на НИР:

- изучить и сравнить существующие генераторы датасетов;
- выделить преимущества и недостатки разных генераторов;
- получить доступ к средствам разработки Unreal Engine и изучить их;
- реализовать прототип среды с 3D сценой и интерфейсом пользователя, используя эти средства разработки;
- реализовать на основе прототипа генератор, который позволит проводить
- комплексное моделирование с параметризацией;
- протестировать полученный генератор на предмет качества данных,
- которые он предоставляет.

Сроки выполнения НИР: 01.09.2021 – 27.12.2021

Дата сдачи отчета: 27.12.2021

Дата защиты отчета: 28.12.2021

Студент _____ Пискунов Я.А.

Руководитель _____ Заславский М.М.

АННОТАЦИЯ

При всех преимуществах получения данных для задач машинного зрения через симуляцию, их основным недостатком является невозможность гарантировать точность и достоверность этих данных. В работе производится исследование существующих сред для симуляции, производится анализ пригодности данных сред для генерации датасетов, которые могли бы быть применены в задачах машинного зрения, а также описывается процесс разработки среды симуляции, которая позволит генерировать данные для таких задач. В данном документе уделено большее внимание разработке устройств сбора данных.

SUMMARY

With all the advantages of obtaining data for machine vision tasks through simulation, their main disadvantage is the inability to guarantee the accuracy and reliability of this data. The work examines existing environments for simulation, analyzes the suitability of these environments for generating data sets that could be used in machine vision tasks, and also describes the process of developing a simulation environment that will generate data for such tasks. Attention is also paid to the development of data collection devices

СОДЕРЖАНИЕ

ПОСТАНОВКА ЗАДАЧИ

С развитием информационных технологий все больше внимания уделяется искусственному интеллекту (ИИ). Среди всех направлений, связанных с ИИ, одним из наиболее быстро развивающихся является машинное зрение.

Технологии, в работе которых применяется машинное зрение, уже достаточно прочно вошли в нашу жизнь. Это, например, системы распознавания лиц, беспилотные системы управления транспортом, системы построения трехмерных объектов. Совершенствование этих технологий важно, так как благодаря этому улучшается качество выполнения задач, которые возложены на искусственный интеллект в целом, а также на ту область его развития, где применяется машинное зрение.

Немалую роль в совершенствовании технологий машинного зрения играет качественная отладка и тестирование новых алгоритмов, которые зависят от того, какие данные используются. Для целого спектра задач машинного зрения необходимы данные из трехмерного пространства. Собирать такие данные можно разными способами. Первый – сбор реальных данных. Он влечет за собой большие временные затраты и различные трудности по доставке оборудования, а также ограничивает спектр возможных параметров окружения. Второй способ – проведение симуляции. Этот способ позволяет проводить много измерений в короткие сроки, регулировать различные параметры системы для проведения сбора большего количества различных данных, которые больше отличаются друг от друга. Однако, не всегда можно гарантировать точность собранных данных.

В настоящее время существует множество различных виртуальных окружений. Они создаются для разных целей, а цель определяет то, что и как будет моделироваться. Прежде всего может отличаться сама моделируемая среда (окружение). Глобально среды можно разделить на наружные (outdoor) и внутренние (outdoor). Внутренние среды представляют собой модели закрытых

6 помещений и применяются, например, для задач ориентирования и перемещения роботов, распознавания объектов и поиска. В свою очередь внешние среды – это модели более открытых пространств, таких как улицы городов или дороги. Такие среды также можно применять для вышеописанных целей, а также возможно их использование в качестве моделей для задач навигации и тестов беспилотного транспорта. Также существуют среды смешанные, в которых на сцене присутствуют как наружные, так и внутренние элементы. Однако они редки, вероятно потому, что зачастую для решения конкретной задачи, под которую создается то или иное окружение, нет необходимости иметь оба типа среды, а также из-за возможной ресурсоемкости таких систем.

Задача также определяет и набор сенсоров, датчиков и иных средств сбора данных, которые моделируются в окружении. Видов таких устройств достаточно много. Их можно разбить на определенные категории по выполняемой функции. Среди них камеры, снимающие в разных диапазонах от видимого до ультрафиолетового и инфракрасного, лидары, датчики освещенности и иные устройства, не относящиеся напрямую к применимым в решении задач машинного зрения, такие как датчики движения, звука.

Как уже упоминалось выше, важную роль в целесообразности применения таких окружений играет возможность обеспечения ими точности и вариативности данных. Вариативность данных подразумевает возможность получить данные, предоставляемые разными средствами сбора, а также с разных точек сцены, в том числе и с тех, с которых их нельзя было бы получить в аналогичной ситуации при сборе данных в реальности.

В свою очередь точность – это соответствие данных, которые предоставляются в симуляции, данным, которые могли бы быть получены с такого прибора в реальности. Сюда можно отнести как соответствие данных адекватности (например, объект между кадрами, заснятыми на камере с разницей в доли секунды, может или не может перемещаться на значительные расстояния, тем самым якобы имея скорость света или выше, или соответствие

7 получаемого поведения теплового фона объекта в инфракрасном диапазоне его предполагаемому или фактическому поведению в реальности), так и непосредственно точность измерений (например, показания радара в сравнении с заданной в симуляции скоростью объекта или соответствия цветов объекта в симуляции и на записи с камеры). Задача обеспечения и, что более важно, проверки точности уже не является настолько тривиальной и иногда рассматривается некачественно или не рассматривается вовсе в существующих виртуальных окружениях для симуляции. Поэтому, разработка сред симуляции, которые предоставляют точные и вариативные данные, является актуальной задачей.

Цель данной работы – разработка среды симуляции, которая позволит собирать данные для задач машинного зрения, при этом повышая точность и вариативность данных, а также свою масштабируемость за счет возможности проведения комплексного моделирования и широкой параметризацией всего от параметров окружения до свойств, возможностей и положения измерительных приборов (средств сбора данных). Для достижения цели необходимо выполнить ряд задач:

- изучить и сравнить существующие генераторы датасетов;
- выделить преимущества и недостатки разных генераторов;
- получить доступ к средствам разработки Unreal Engine и изучить их;
- реализовать прототип среды с 3D сценой и интерфейсом пользователя, используя средства разработки Unreal Engine;
- реализовать на основе прототипа полноценный генератор, который позволит проводить комплексное моделирование с параметризацией, а также будет обладать удобным интерфейсом пользователя;
- разработать методологию тестирования полученного генератора на предмет качества и точности предоставляемых им данных;
- произвести тестирование по разработанной методологии и сделать вывод о качестве созданного генератора датасетов.

1. РЕЗУЛЬТАТЫ РАБОТЫ В ОСЕННЕМ СЕМЕСТРЕ

1.1. План

Период	Задачи
09.06 – 01.08	Доработка устройств сбора, добавление возможности получения и сохранения данных с устройств, создание и проведение эксперимента по проверке точности камеры, создание docker-контейнера для проекта
02.08 – 01.10	Добавление возможности управления положением сенсора в пространстве во время проведения симуляции, проведение тестовой симуляции облета управляемым дроном 3D объекта с записью данных камеры и лидара
02.10 – 31.12	Добавление новых видов и подвидов устройств сбора данных, отладка и исправление ошибок по итогам проведенных ранее экспериментов

1.2. Доработка устройств сбора данных

В прошедшем семестре основное внимание было уделено исследованию возможностей движка Unreal Engine (UE) в области моделирования устройств сбора данных. Так, была начата реализация камеры. Обновленная UML диаграмма классов представлена на рис. 1. Следует отметить, что на данной диаграмме представлены только классы и методы, которые непосредственно описаны на языке C++. Однако, помимо этого, большая часть поведения

устройств реализована также с использованием встроенной технологии Unreal Engine под названием Blueprints [1].

Основа для реализации устройств была создана ранее в виде общего суперкласса для всех устройств ABasedevice, унаследованного от встроенного в движок AActor, который представляет собой класс, описывающий базовое поведение объекта, помещаемого в сцену.

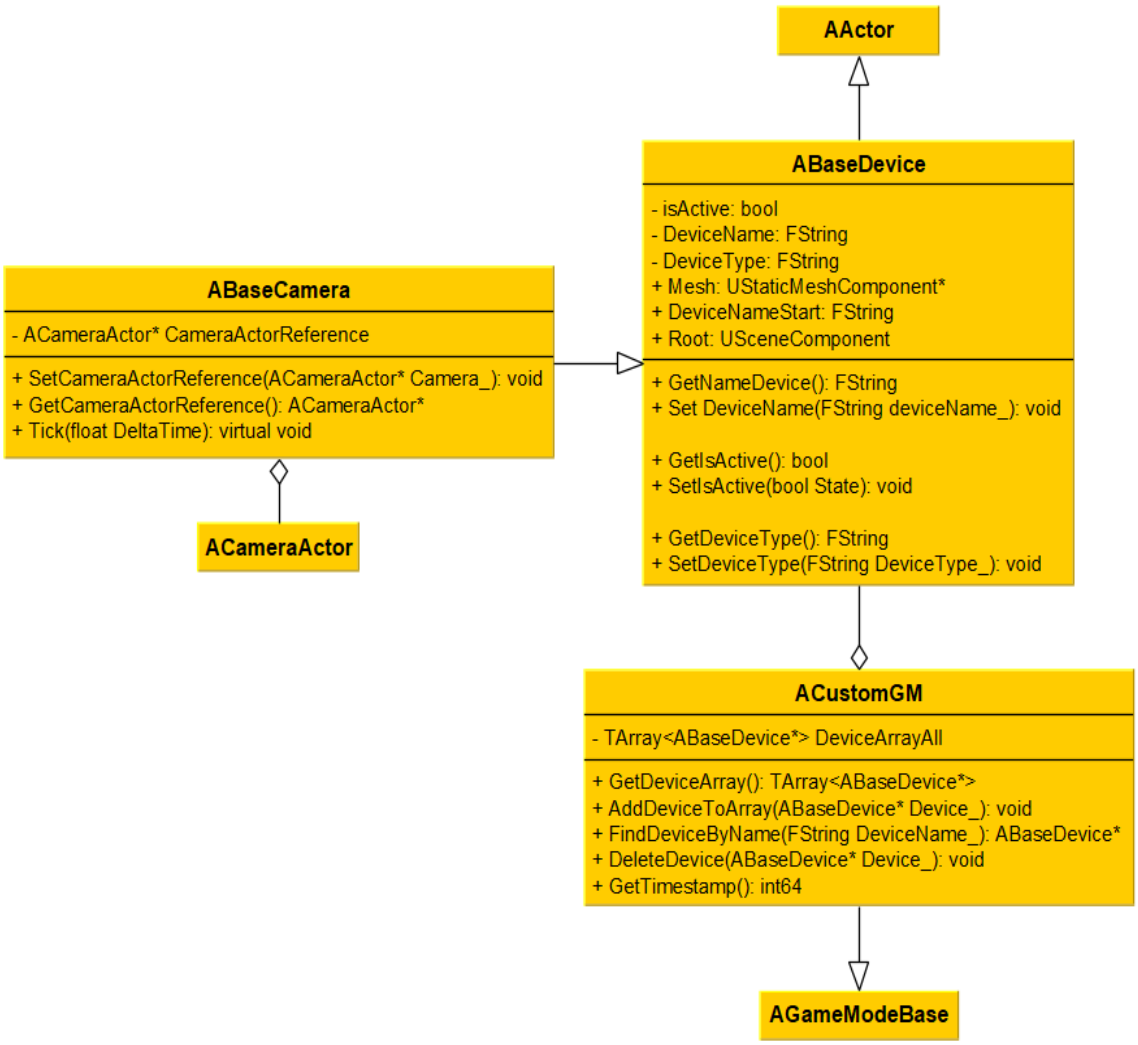

Рисунок 1 – UML диаграмма классов

Для обеспечения работы камеры класс ABaseDevice был расширен до ABaseCamera путем наследования. Изначально планировалось хранить все параметры камеры в данном классе отдельно, но UE предоставляет готовый класс для управления камерами, который уже содержит в себе все необходимые настройки и методы для изменения параметров камеры [2]. Таким образом, в

унаследованном классе сохраняется ссылка на объект типа CameraActor, который создается в момент размещения камеры на сцене. Процесс привязки реализован с помощью Blueprints и представлен на рис. 2. Как можно заметить, для привязки используется метод SetCameraActorReference, который описан на C++. Код, отвечающий за BaseCamera представлен в приложении А.

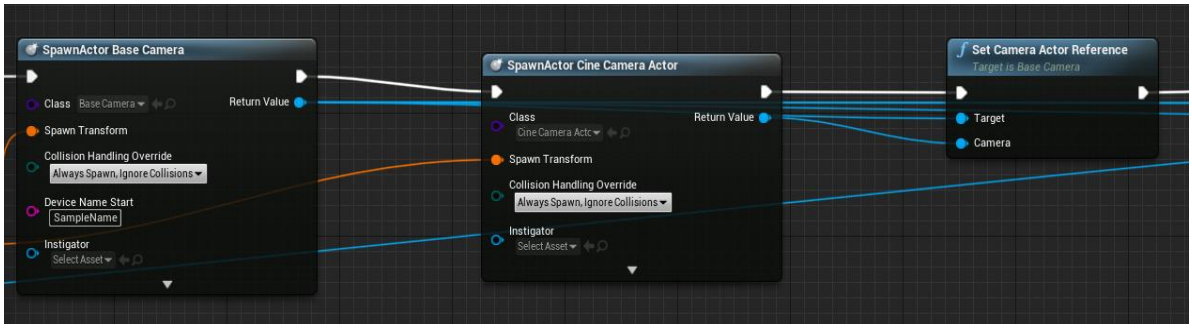

Рисунок 2 – Привязка объекта «CameraActor» к «BaseCamera»

1.3. Сохранение данных с устройств

Помимо реализации работы камеры также были предприняты шаги по исследованию возможностей по сохранению результатов работы устройств в файлы. Так, камера предоставляет набор снимков. Для снятия этих снимков и их сохранения. На данный момент сохранение происходит в папку по умолчанию внутри каталога проекта. Процесс получения снимков также реализован с помощью Blueprints.

Сам по себе объект CameraActor не имеет возможности сохранить снимок со своей перспективы, он лишь выступает своего рода перспективой, с которой можно смотреть на сцену. Таким образом, для получения снимка с какой-либо камеры необходимо на нее переключиться. Недостатком в данном случае является тот факт, что мгновенное или близкое к нему переключение между камерами не позволяет снять снимки со всех камер (процесс выполнения команды снятия снимка занимает некоторое время). С другой стороны, при наличии значимого времени между переключением с камеры на камеру теряется возможность получения снимков одного и того же момента времени с разных камер. Возможное решение данной проблемы будет предложено позже.

Сам же процесс переключения между существующими камерами и получение снимков представлен на рис. 3 [4].

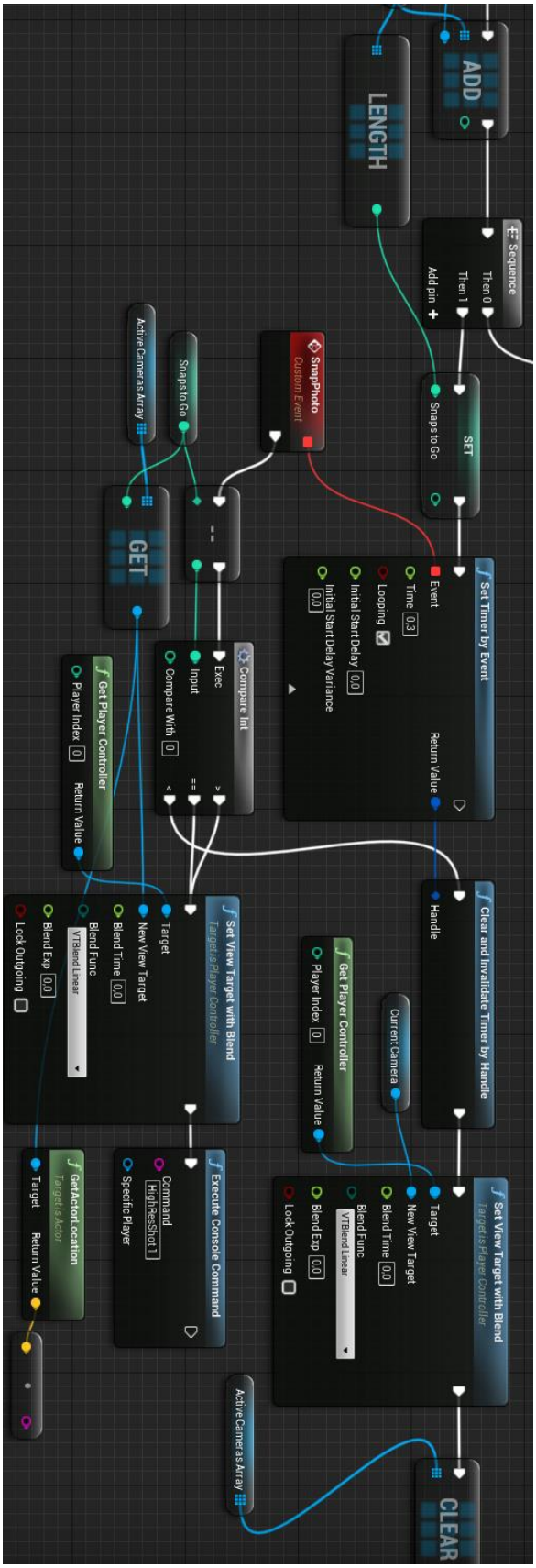

Рисунок 3 – Процесс получения снимков из массива активных камер

1.4. Положение сенсора в пространстве

Реализовано два различных способа размещения сенсора в пространстве – статичный и прикрепленный к объекту. Размещение статичного сенсора происходит в месте, где была произведена команда на размещение. Процесс размещения объекта также реализован с помощью Blueprints и уже частично представлен на рис. 1. Помимо представленного на этом рисунке также происходит определение позиции и ротации создаваемого объекта через позицию пользователя (представлено на рис. 4) и добавление так называемого меша, который позволяет визуально увидеть расположение камеры на сцене (представлено на рис. 5). В качестве меша пока что используется временный вариант из библиотеки движка, внешний вид которого представлен на рис. 6. Также во время создания происходит добавление имени устройства (на данный момент берется текущая дата и время). После создания устройство добавляется в массив устройств. На данном этапе все добавленные устройства считаются активными по умолчанию [4].

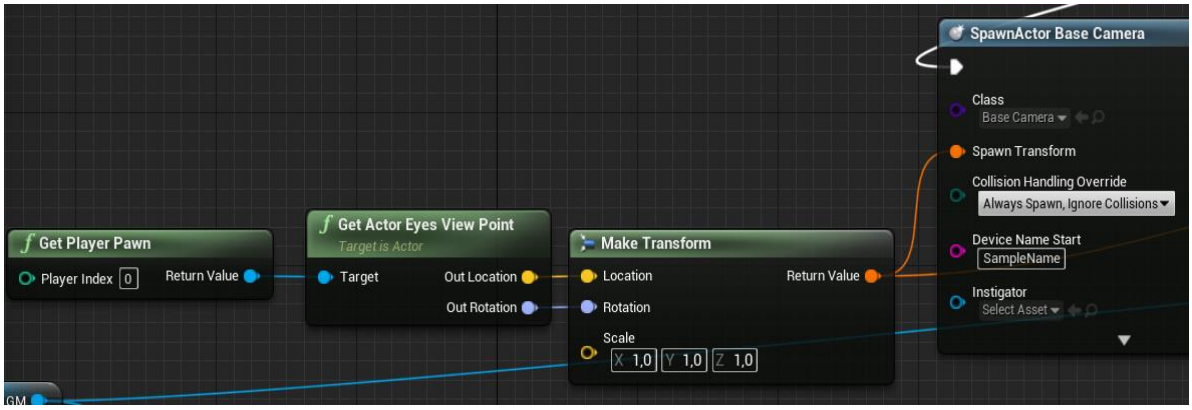

Рисунок 4 – Определение позиции размещаемой камеры

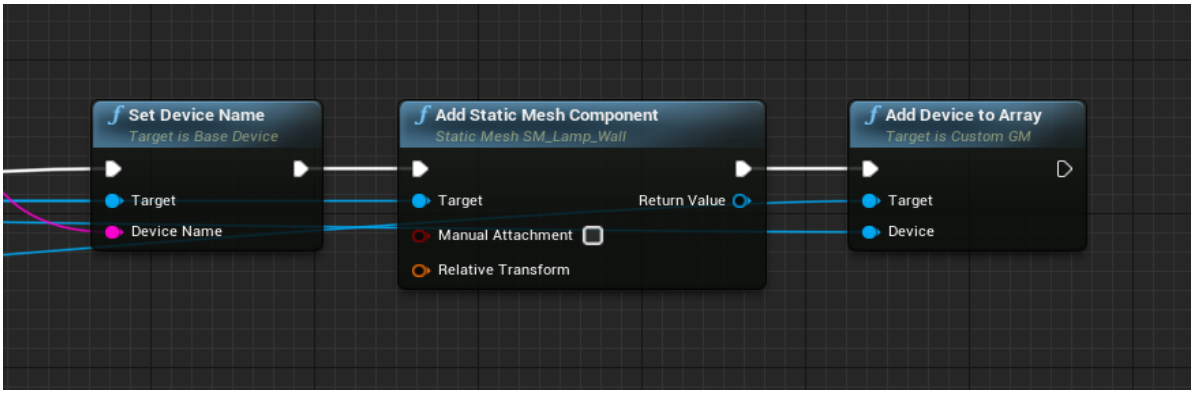

Рисунок 5 – Добавление меша, имени устройства и сохранение в массив

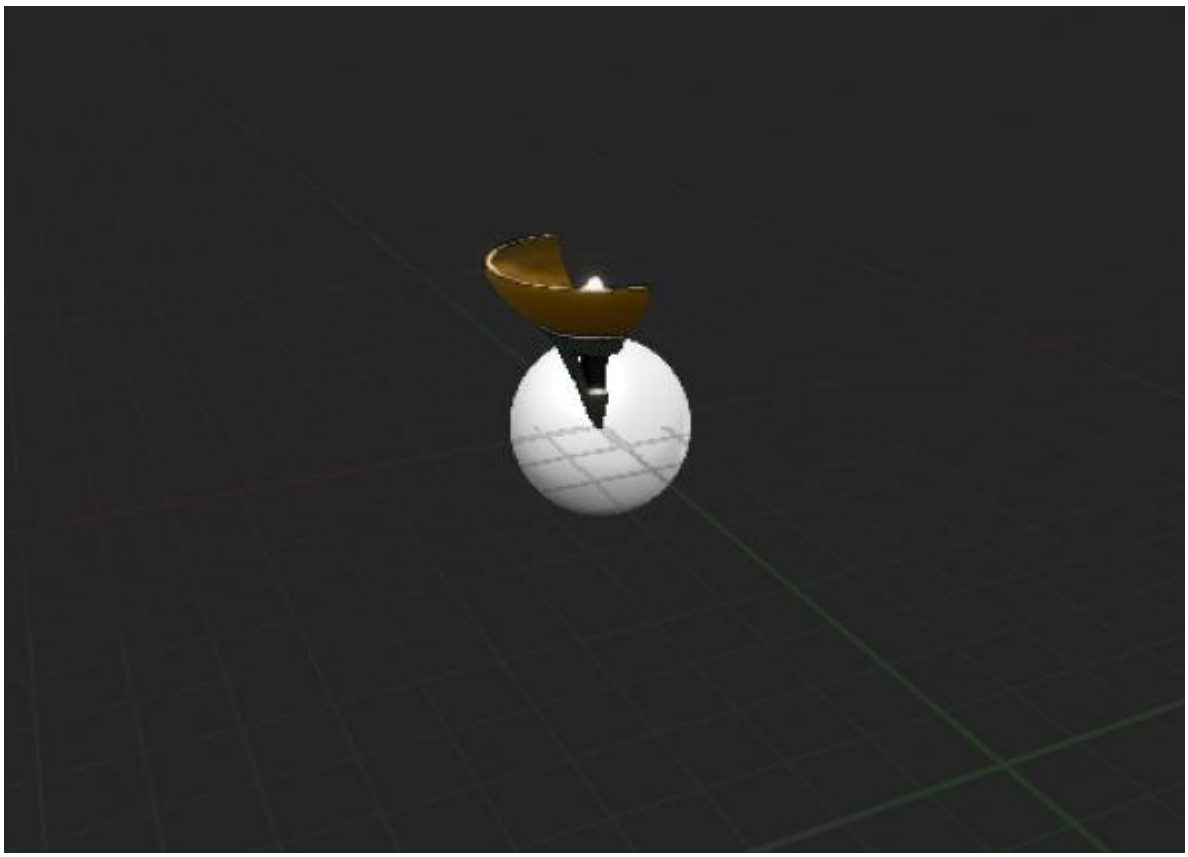

Рисунок 6 – Временный меш, используемый для отображения устройств

Второй способ размещения сенсоров – привязать их к объекту. На данный момент данный способ реализован, однако для того, чтобы он работал, необходим объект, к которому привязан объект камеры, BaseCamera или CameraActor. Создание подобных объектов возможно в редакторе UE. Возможность присоединения камеры к произвольному объекту во время выполнения собранной программы исследуется. Процесс получения данных в целом аналогичен тому, что применяется при получении свободных данных [4].

1.5. Тестовые симуляции

Для проведения тестовых симуляций создан демонстрационный уровень в виде поверхности, по которой движется объект в виде шара. В редакторе к этому шару также присоединена камера. Для того, чтобы различать углы камер и положение шара, по краям поверхности расположены дополнительные объекты. Внешний вид демонстрационного уровня представлен на рис. 7. На этом уровне также можно размещать камеры в произвольных точках путем нажатия клавиши «Space». Также, можно собрать снимки со всех

существующих камер с помощью нажатия клавиши «Z». Также имеется возможность на 5 секунд переключиться на камеру, которая привязана к движущемуся объекту.

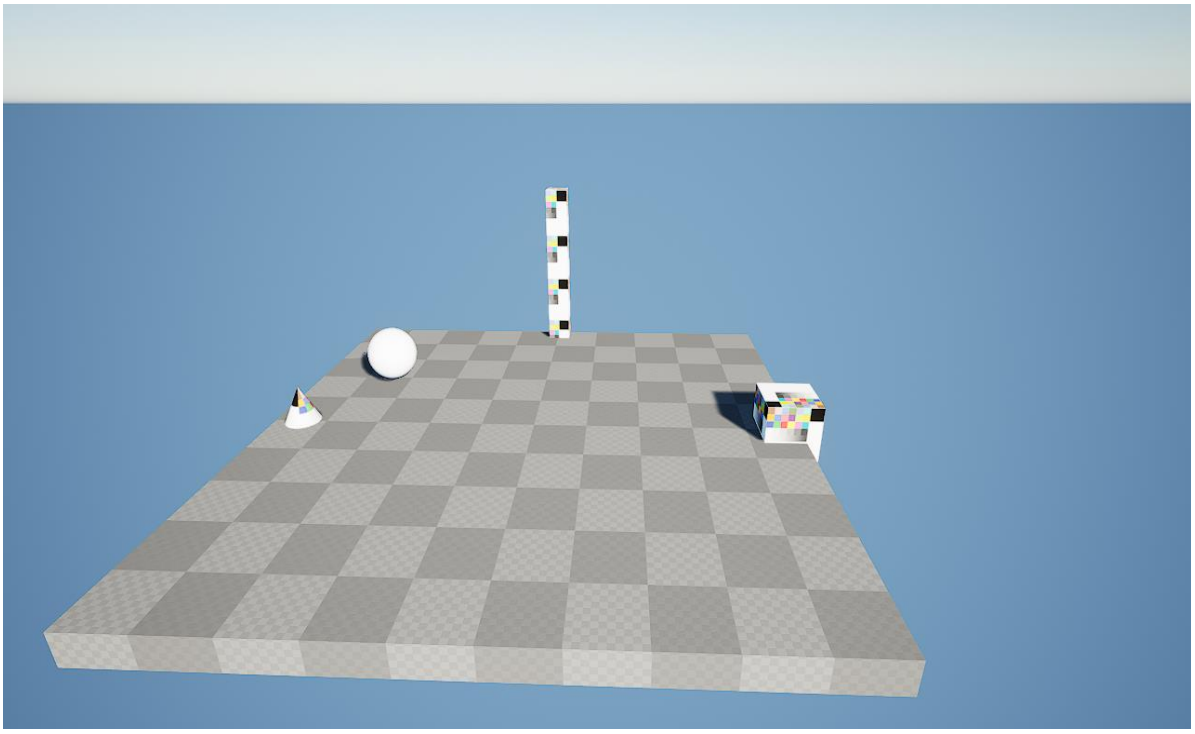

Рисунок 7 – Демонстрационный уровень

Демонстрационные скринкасты, а также полученные снимки можно найти в репозитории проекта в папке doc [3].

2. ОПИСАНИЕ ПРЕДЛАГАЕМОГО МЕТОДА РЕШЕНИЯ

2.1 Настройки камеры

На данный момент камера реализована, но использует значения всех настроек по умолчанию. Для того, чтобы реализовать изменение этих настроек, имеется улучшенный вариант CameraActor, который называет CineCameraActor. Этот вид камеры обладает тем же поведением, что и обычный, но при этом его можно более гибко настраивать. Наиболее полезные настройки из тех, что можно использовать, представлены на рис. 8. На рисунке также представлены структуры данных, соответствующие настройкам в случаях, когда содержание настроек неочевидно. Также данный вид камеры обладает адаптивным фокусом и функцией трекинга движущихся объектов. Поведение данного вида камеры должно быть идентично текущей камере, так что переход скорее всего не будет трудным.

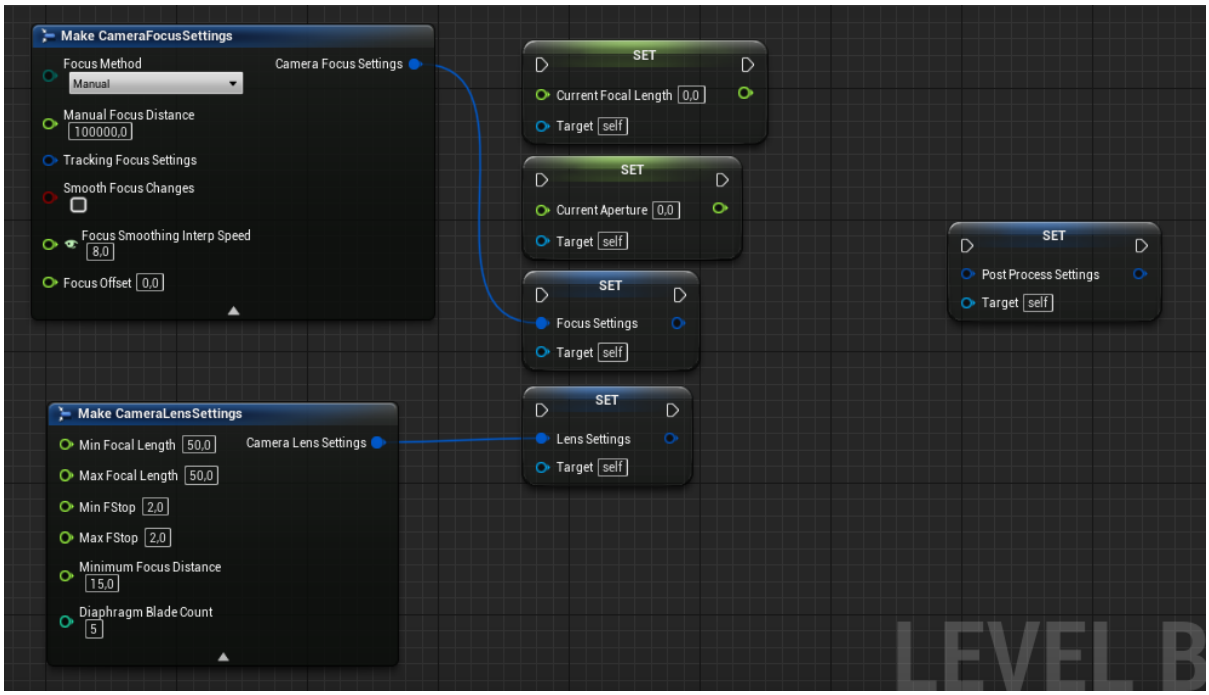

Рисунок 8 – Настройки камеры

2.2 Другие устройства сбора данных

После полноценной реализации камер также рассматривается возможность добавления других устройств сбора данных. Прежде всего лидаров. Для реализации лидаров в библиотеке UE имеется бесплатное

решение. Однако, оно доступно только на более старых версиях движка. Для более новых версий имеется платная библиотека. Ее главная проблема в том, что она платная. Поиск других библиотек ведется. Принимая во внимание вышеописанное, рассматривается также возможность написания кода данного устройства вручную, опираясь на существующие наработки коллег в данной области [4].

Учитывая специфику работы лидаров (подсчет расстояния до препятствий), можно с высокой долей уверенности заявить, что после успешной реализации данного вида устройств также можно будет на их основе реализовать и RGBD камеры, отличительным свойством которых от реализованных уже RGB камер является как раз наличие кроме обычного снимка еще и карты глубины кадра, которую можно будет построить на основе данных, которые предоставляет лидар.

2.3 Возможное решение проблемы одновременности снимков

Как уже было отмечено выше, имеет место проблема с одновременным снятием снимков с нескольких камер. Процесс получения снимка занимает некоторое время, за которое сцена изменяется. Это может быть значительной проблемой для некоторых датасетов. Одним из возможных решений этой проблемы может служить разделение процесса симуляции некоторой «жизни» на сцене и процесса непосредственного снятия данных с устройств. Ко второму пункту в целом можно отнести и непосредственное размещение устройств в сцене. Более того, такой подход позволит с легкостью использовать уникальные для каждого пользователя сцены без необходимости пересобирать наше приложение каждый раз для изменения сцены.

Ключевым пунктом в предлагаемом подходе является использование встроенной в ue4 системы реплеев. Данная система предоставляет возможность записи происходящего на сцене, сохранение этого в файл с определенным расширением, а также воспроизведение. Причем в процессе воспроизведения, помимо возможности перематывать, возвращаться в некоторый момент времени, менять скорость воспроизведения и, что важно в нашей ситуации,

ставить воспроизведение на паузу. Постановка воспроизведения на паузу не прерывает процессов, происходящих в сцене, у нас все также будет иметься возможность размещения средств сбора данных и получения с них результатов [5].

Применение реплеев позволит обойти описанную выше проблему, а также упростит пользователям и разработчикам процесс взаимодействия с программой. Так, не будет необходимости загружать в программу уровни, ассеты, акторов, размещать их. Все это можно сделать в других программах или даже просто в редакторе UE, записать реплей, а после загрузить его в нашу программу, разместить средства сбора данных и получить с них требуемую информацию, вернувшись к интересующему моменту времени несколько раз если это необходимо.

2.4 Формат данных

Формат экспортируемых данных сильно зависит от типа датаеста и целей, для которых он создается. Для датасетов с камер существует множество применений [6]. В целом, данные, которые хранятся в этих датасетах не являются чем-то сложным. Зачастую это те же картинки и числа, отвечающие за настройки камеры или ее позицию [7].

На основе состава типичных датасетов камер [7] было принято решение о том, какие именно данные будет предоставлять программа. Так, помимо непосредственно снимка, будет необходим файл, который сопоставляет данному снимку метку времени (время в секундах с 1 января 1970 по настоящий момент [8]). Обычно также метка времени сохраняется в имени файла изображения. Аналогичные файлы нужны для сопоставления времени с позиццией камеры, которая определяется семью числами. Три числа – координаты по осям X, Y, Z и четыре числа – кватернион, однозначно определяющий поворот объекта относительно этих же осей. Также в некоторых случаях может понадобиться файл, содержащий скорость по этим же осям. Для получения метки времени модифицирован CustomGM – туда добавлен метод,

который возвращает метку времени по запросу. Код CustomGM представлен в приложении Б.

3. ПЛАН НА ВЕСЕННИЙ СЕМЕСТР

Целью работы в весеннем семестре 2022 года является доработка прототипа, реализация других видов устройств, помимо камеры, а также объединение всех наработок в единую систему, которая позволит выполнить описанные ранее сценарии использования.

Период	**Задача**
01.01 – 13.01	Подготовка публикации по теме работы
13.01 – 14.02	Завершение разработки прототипа, который сможет позволить выполнить основной сценарий использования для RGB камеры
14.02 – 01.04	Реализация дополнительных видов устройств
01.04 – 01.05	Объединение всех элементов программы и наработок в единую систему
01.05 – 01.06	Финальная отладка

ЗАКЛЮЧЕНИЕ

В результате работы в осеннем семестре 2021 года были исследованы возможности UE по симуляции различных средств сбора данных, реализованы RGB камеры. Также реализованы два вида размещения средств сбора данных – статичные и прикрепленные к объекту.

Глобальная цель осеннего семестра выполнена частично. Разработан демонстрационный прототип, в котором есть возможность получить данные с камеры. Не выполнены следующие пункты плана – добавление новых видов и подвидов устройств сбора данных, создание docker-контейнера для приложения, проведение эксперимента по проверке точности камеры. Выполнены частично (только для камеры) – доработка устройств сбора, добавление возможности получать и сохранять данные с устройств, добавление возможности управления положением сенсора во время проведения симуляции (статично, прикреплен к объекту).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Blueprint Visual Scripting | Unreal Engine Documentation // unrealengine. URL: https://docs.unrealengine.com/4.26/enUS/programmingAndScripting/Blueprints (дата обращения 27.12.2021).

2. Using Cameras | Unreal Engine Documentation // unrealengine. URL: https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/UsingCameras/ (дата обращения 14.08.2021).

3. moevm/cv_dataset_generator at devices_dev // github. URL: https://github.com/moevm/cv_dataset_generator/tree/devices_dev (дата обращения 27.12.2021).

4. moevm/bsc_pavlov // github. URL: https://github.com/moevm/bsc_pavlov (дата обращения 27.12.2021).

5. Replay System | Unreal Engine Documentation // unrealengine. URL: https://docs.unrealengine.com/4.27/en-US/TestingAndOptimization/ReplaySystem/ (дата обращения 27.12.2021).

6. Computer Vision Group – Dataset Download // tum. URL: https://vision.in.tum.de/data/datasets/rgbd-dataset/download (дата обращения 01.10.2021).

7. Computer Vision Group – File Formats // tum. URL: https://vision.in.tum.de/data/datasets/rgbd-dataset/file_formats (дата обращения 01.10.2021).

8. Unix Time Stamp – Epoch Converter // unixtimestamp. URL: https://www.unixtimestamp.com/ (дата обращения 27.12.2021).

9. Vicci L., Quaternions and Rotations in 3-Space: The algebra and its Geometric Interpretation // UNC Chappel Hill, Department of Computer Science. USA, North Carolina. 2001.

ПРИЛОЖЕНИЕ А

КОД BASECAMERA.CPP

```
include "BaseCamera.h"

ABaseCamera::ABaseCamera() : ABaseDevice()
{
	this->SetDeviceType("BaseCamera");
}

void ABaseCamera::Tick(float DeltaTime)
{
	Super::Tick(DeltaTime);
}

void ABaseCamera::SetCameraActorReference(ACameraActor* Camera_)
{
	this->CameraActorReference = Camera_;
}

ACameraActor* ABaseCamera::GetCameraActorReference()
{
	return this->CameraActorReference;
}

void ABaseCamera::BeginPlay()
{
	Super::BeginPlay();
}
```

ПРИЛОЖЕНИЕ Б

КОД CUSTOMGM.CPP

```
#include "CustomGM.h"

void ACustomGM::AddDeviceToArray(ABaseDevice* Device_)
{
	this->DeviceArrayAll.Add(Device_);
}

ABaseDevice* ACustomGM::FindDeviceByName(FString DeviceName_)
{
	for (int i = 0; i < this->DeviceArrayAll.Num(); i++)
	{
		if ((this->DeviceArrayAll[i]->GetNameDevice()).Compare(DeviceName_) == 0)
		{
			return this->DeviceArrayAll[i];
		}
	}

	return nullptr;
}

void ACustomGM::DeleteDevice(ABaseDevice* Device_)
{
	if (Device_ != nullptr)
	{
		Device_->Destroy();
		this->DeviceArrayAll.Remove(Device_);
	}
}

int64 ACustomGM::GetTimeStamp()
{
	FDateTime Time = FDateTime::Now();
	int64 Timestamp = Time.ToUnixTimestamp();
	return Timestamp;
}

TArray<ABaseDevice*> ACustomGM::GetDeviceArray()
{
	return this->DeviceArrayAll;
}
```