FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Import and post-processing of BIM files in Unreal Engine
Student:	Denis Dubin
Supervisor:	Ing. Petr Pauš, Ph.D.
Study program:	Informatics
Branch / specialization:	Web and Software Engineering, specialization Computer Graphics
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2023/2024

Instructions

BIM files contain information about the structure of buildings. The goal of this work is to create a project in Unreal Engine that allows importing them, assigns appropriate textures and sets the necessary parameters of the imported models.

Instructions:
1. Analyse the BIM format and its capabilities.
2. Analyze the capabilities of Unreal Engine to import BIM files and assign appropriate properties.
3. Using software engineering methods, design a project in Unreal Engine that will enable the import.
4. Implement the model import. Implement the ability to browse the imported building from a first-person view.
5. Perform appropriate testing of the project on the provided sample files.

Electronically approved by Ing. Radek Richtr, Ph.D. on 4 February 2023 in Prague.

Bachelor's thesis

IMPORT AND PROCESSING OF BIM FILES INTO UNREAL ENGINE

Denis Dubin

Faculty of Information Technology
Department of software engineering
Supervisor: Ing. Petr Pauš, Ph.D.
May 16, 2024

Citation of this thesis: Dubin Denis. *Import and processing of BIM files into Unreal Engine.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2024.

Contents

List of Figures

Abstract

This undergraduate work aims to explore the optimal methods for importing BIM files into Unreal Engine 5 and provide basic functionality for viewing the final result. The main focus will be on an off-the-shelf solution from Unreal Engine developers called DataSmith. Also demonstration of the functionality of the project to view the test files provided to me.

Keywords UE5, Unreal Engine 5, Datasmith, BIM, Building Information Modeling, Building Information Management.

Abstrakt

Cílem této bakalářské práce je prozkoumat optimální metody importu souborů BIM do Unreal Engine 5 a poskytnout základní funkce pro zobrazení konečného výsledku. Hlavní zaměření bude věnováno hotovému řešení od vývojářů Unreal Engine s názvem DataSmith. Také ukázka funkčnosti projektu pro zobrazení testovacích souborů, které mi byly poskytnuty.

Klíčová slova UE5, Unreal Engine 5, Datasmith, BIM, Building Information Modeling, Building Information Management.

First and foremost I would like to thank my presenter of this work. Without Ing. Petr Pauš, Ph.D., this work would not have been possible and was completed only thanks to his support.
In addition, I would like to thank Ing. Radek Richter Ph.D. and Ing. Jiří Chludil for being able to interest the students in the material they teach and for having an individual approach to each of their students, being always open to discussion and not sparing their time for the students.
I would also like to thank the school for providing me with the skills used in this work, as well as the developers of Epic Games for making the documentation quite nice and convenient.
I also want to express my gratitude to my family and relatives for the moral and emotional support during the process of this work.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis. I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Czech Technical University in Prague has the right to conclude a licence agreement on the utilization of this thesis as a school work pursuant of Section 60 (1) of the Act

In accordance with Section 2373(2) of Act No. 89/2012 Coll., Civil Code, as amended, I hereby grant a non-exclusive authorisation (licence) to use this copyright work, including all computer programs and all their documentation (hereinafter collectively referred to as "the Work"), to all persons who wish to use the Work. Such persons shall be entitled to use the Work in any manner that does not diminish the value of the Work, but only for non-profit purposes. This authorisation is unlimited in time, territory and quantity.

In Prague on May 16, 2024

Introduction

There is a lot of software for rendering scenes, objects, animations and other things, but, at the moment, the most popular is Unreal Engine 5 because of the presence of advanced technologies for rendering light with Lumen, geometry thanks to Nanite and other technologies that at the time of writing are unrivaled. Among those interested in using this software there are representatives from a variety of parties, but in context of this paper we will deal exclusively with users of the format BIM, and specifically the problem of transferring files from the format BIM to formats accepted Unreal Engine 5 through the official plugin from the developers of Unreal Engine called Datasmith. The range of accepted formats of Datasmith includes a huge number of formats such as: 3ds Max. Cinema 4D, Revit and SketchUp Pro, IFC, Rhino 3DD, SolidWorks and CATIA, VRED and DELTAGEN, a solid number of other formats from the CAD/CAID spectrum.

Chapter 1
Goals

The first task of this work is to create a description of BIM files, their properties, and describe their transformation stages during the import process.

The second task is to find and describe existing methods of importing these files into Unreal Engine and provide information regarding the characteristic features of these methods.

Based on one of the selected methods, using methods of software engineering, provide a project capable of importing the provided test data on the basis of one of the described methods and the corresponding documentation on the creation of such a project.

Analyze the resulting project containing imported test data and describe the final results.

Chapter 2

Unreal Engine

Unreal Engine is a game engine owned and developed by Epic Games. Initially, this game engine was developed exclusively for shooters, but in the wake of changes in the company's policy has been repeatedly revised, and the goal is to make it universal. Written in C++ has ready functionality for the development of products on most existing operating systems and platforms. To simplify porting to different operating systems and platforms, the engine uses a modular system of dependent components: it supports various rendering systems, audio playback, voice text playback, speech recognition, modules for networking, and support for various input devices.[4]

Thanks to its universal functionality, Unreal Engine has a large and diverse audience, including representatives of the film industry, game industry, architecture and many others. Thanks to this, the community actively interacts on forums and other platforms to share knowledge and solve problems. The high-level functionality provides the ability to produce complex renders using a variety of development tools, including level editors, material editors, animation editors, sound editors and many others, making it relatively self-contained and avoiding the use of third-party editors.

2.1 Understanding File Structures

The parent of any class used inside Unreal Engine is UObject. This does not mean that it is impossible to create a class that is not a descendant of UObject, but it does mean that you will lose all the benefits of UObject and your class will exist in isolation from the internal Unreal Engine ecosystem. The four main gameplay classes are most often extended. These are UObject, AActor, UActorComponent and UStruct. Each of them will be described in detail below. You can create types that do not extend any of these classes, but they will prevent you from using most of the engine's built-in features. As a rule, classes that are created outside the UObject hierarchy (they are not inherited at any depth from UObject) exist for the following purposes: integration of third-party libraries, wrapping of operating system features, etc.[5]

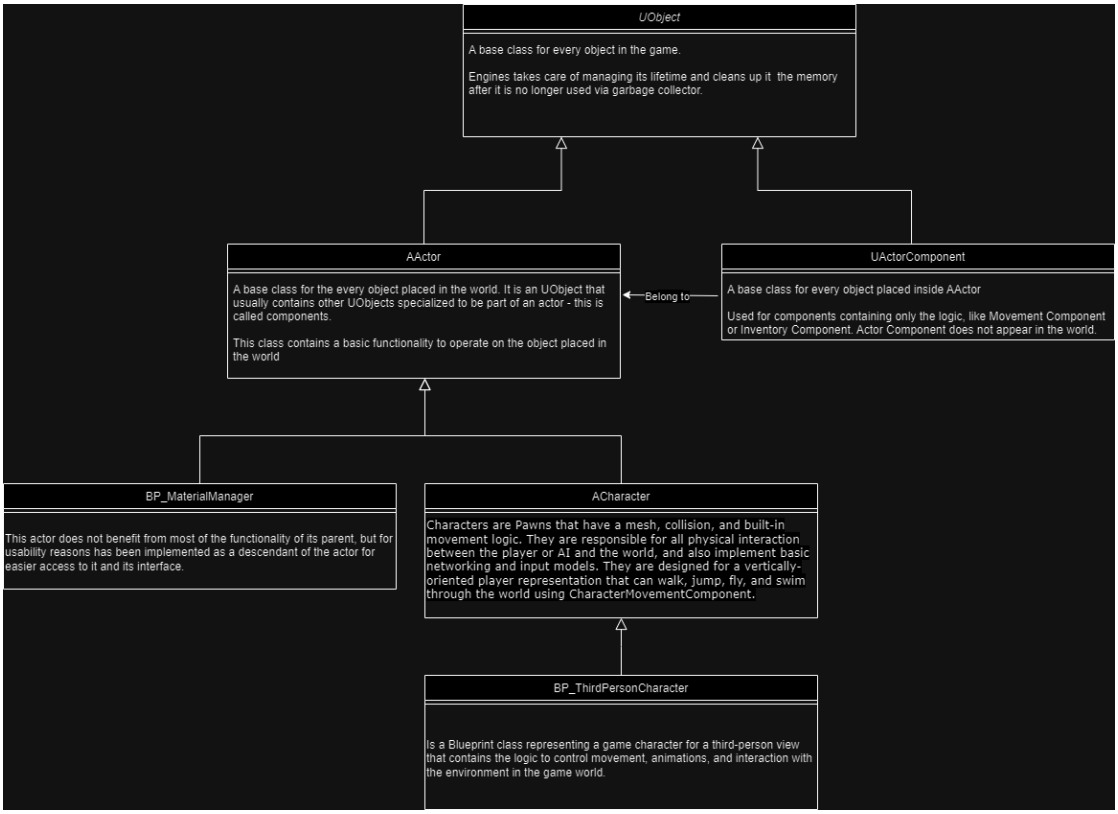

Figure 2.1 Simplified diagram of native and created classes with description.

UObject

As mentioned earlier, UObject is a Unreal Engine base class that provides you with basic functionality such as:

- Reflection of properties and methods
- Serialization of properties
- Garbage collector
- Search UObject by name
- Customizing values for properties
- Network mode support for properties and methods

Each class inherited from UObject contains a UClass singleton created for that class. This class contains all the metadata about the class instance. The functionality of the UObject and UClass (together) is the basis for everything a gameplay object does during its lifecycle. The best way to understand the difference between UClass and UObject is that UClass describes exactly what a UObject instance is and what properties are available for things like serialization and networking. [6]

AActor

The AActor class represents the base object of the game space (every object in the scene is an AActor or its descendant). An AActor inherits from a UObject, that is, it uses the standard functions that were listed in the previous section. AActor can be destroyed manually (C++ or

Blueprint) or by using the standard garbage collection mechanism (when a level is unloaded from memory). It provides high-level behavior for our game objects, eliminates the need for low-level memory management, and provides basic functionality that enables network mode replication. When replicating (in multi-user mode), an AActor also provides access to information about any of its UActorComponents, which is necessary to support network mode operation. AActors have their own behavior (specialized by inheritance), but in addition they are containers for a hierarchy of UActorComponents (specialized by composition). Within the actor is usually located all the logic of game mechanics and basic animations like spline movement.

It's important to mention that AActor contains a lot of basic events that give you the functionality you need to implement their logic. For example, the Tick(float Delta) event is updated with each tick of the game cycle and returns the time from the last Tick, which allows you to bind, for example, the animation of movement along the curve to the delta and thus make it smoother and more correct relative to real time. The BeginPlay event is called at the moment of AActor construction completion and thus notifies about the beginning of its existence and execution of its logic. [7]

UActorComponent

UActorComponent are not independent objects with their own implemented logic and are used as a rule to give this logic to AActors that initially do not have a common parent to avoid duplication of code and logic. It is assumed that each component solves a separate unique high-level task, so for example each NPC, quest object, city, etc. needs to have its own marker on the map, but between the quest object and NPC initially common parent is only AActor. This problem can be solved by implementing this logic in UActorComponent and adding this component to these objects. [8]

ACharacter

ACharacter in Unreal Engine is a base class designed to represent game characters in the virtual world. It is part of the Unreal Engine game engine and provides a number of basic features and characteristics that can be used to create different types of characters in the game. ACharacter contains functionality for controlling movement, interaction with the environment, animation, and handling input from the player or other sources.

This class is typically used as a base class to create specific character types such as heroes, enemies, NPCs, etc. Developers can extend the functionality of ACharacter by adding unique features and characteristics depending on the needs and requirements of their project.

All classes that will be used in this paper are derived from the above classes.

2.2 Blueprints

Blueprint is a visual programming system in Unreal Engine 4 and Unreal Engine 5 based on nodes with data: events and functions. They can be linked to each other and form gameplay elements. There are several types of Blueprints designed for specific tasks, from creating a level event to interfaces and macros that can be used as the basis for another Blueprint. Even though Blueprints don't have the same amount of functionality as C++, their capabilities are quite extensive. They can dynamically modify geometry, materials and their configuration, particle system behavior, and more.

In practice, it usually works like this: developers provide all the necessary functionality that can be used in the game, and game designers, technical artists and others use the provided

functionality at the Blueprint level to implement the final concepts, so they do not need to know C++ and its api in the context of Unreal Engine.

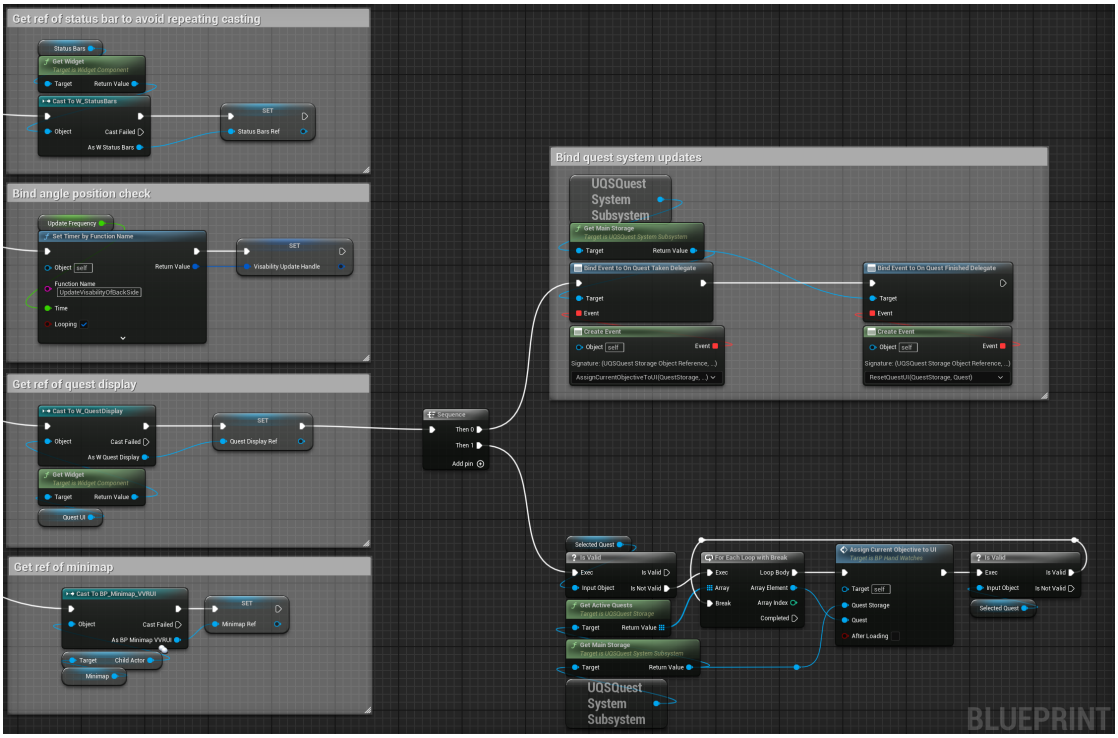

Figure 2.2 Blueprints example

The original parent of Blueprint is believed to be Kismet, but it is generally not used at the moment due to high qualification requirements on the developer side. Blueprints due to its abstractness and multiple encapsulation, has comparatively lower performance than the same code written in C++.

Within Blueprint it is worth to arrange rather basic logic like: actor move to a point, play animation, make requests to other actors or subsystem, but not calculations or other mathematical logic.

2.2.1 BP_ThirdPersonCharacter

BP_ThirdPersonCharacter was created and developed by the development team at Epic Games, the company behind the Unreal Engine. They created this class as part of the standard set of functionality provided by the engine to simplify the process of creating third-person game characters.

There are several reasons why BP_ThirdPersonCharacter is the way it is:

- Versatility: BP_ThirdPersonCharacter is designed to provide basic functionality that can be used to create a wide range of characters in a variety of games. It provides the basic control, animation, and interaction features that are often required in third-person view games.
- This class is designed to be relatively easy to use and customize for game developers, including those without deep programming knowledge. It can be customized and extended using Blueprint's visual interface, making it accessible to a wide range of developers.

- Flexibility: BP_ThirdPersonCharacter provides basic functionality, but remains flexible and customizable. Developers can augment it with their own logic and functionality to customize it to meet the specific requirements and design of their games.
- Compatibility: This class integrates well with other Unreal Engine components and features, making it part of the vast ecosystem of game development tools and resources.

The above-mentioned features make it indispensable and alternative at least at the early stages of development, and some solutions used in already implemented projects are custom versions of this class.[9]

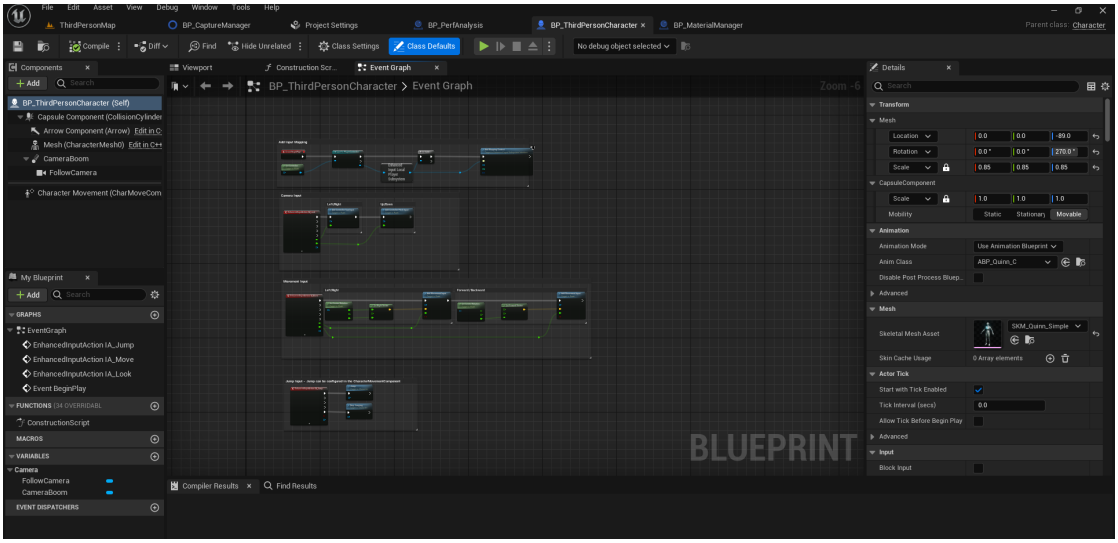

Figure 2.3 Third person character blueprint, detail panel and other.

The above image shows the standard interaction interface and settings of this class. In the details panel you can change the default values of this actor, for example the blueprint animations used, the response to a particular collision channel, the mass of the actor and more. The space in the middle is the Blueprints themselves. In them the logic of the object is realized. The interface on the top left shows the hierarchy of actor components within the actor itself. You can add or remove components and customize the values of their attributes. The lower left tab stores a list of all the events and functions as well as the variables of the actor.

2.2.2 In-game controls

The basic functionality of third-person character control will be described below. In case of need, they can be changed using project settings and changing mapping of inputs.

- **W** – Moves the character forward.
- **A** – Moves the character left.
- **S** – Moves the character back.
- **D** – Moves the character right.
- **Spacebar** – Jump.
- **Mouse Rotation** – Rotating camera around character via mapping 2D movement onto sphere.

2.3 Materials

Materials are one of the most important elements for creating compelling interactive content. Materials determine how each individual pixel in a scene reacts to light, shadow, and reflection.

Creating materials in real-time can be very different from creating materials in 3D applications and renderers. Like most Unreal Engine processes, materials work focuses on interactivity and performance. Unreal Engine has WYSIWYG[1], real-time material previewing and - through the visual Material Editor - the ability to program, literally based on every pixel and vertex, how materials behave. Because of its interactive nature, Unreal Engine provides instant feedback on stage about how materials look.[10]

There is no material creation within the scope of this work. The main interaction with materials is the application of materials to meshes based on the method of syntactic analysis of mesh names, which will be described in the next chapters. The materials that will be used in this work are taken from the Starter Content[2].

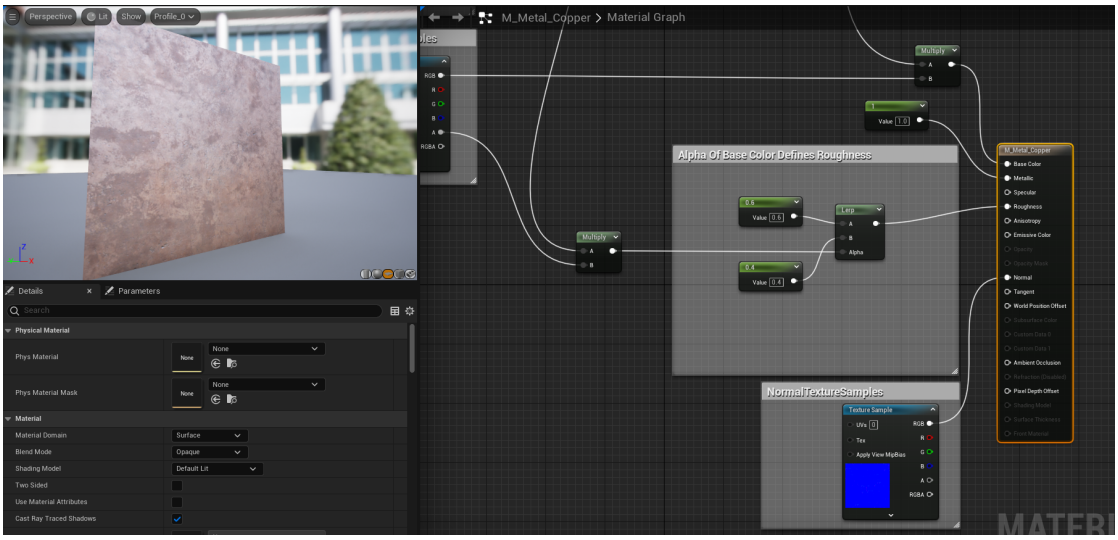

Figure 2.4 Node graph of metal material imported from Starter Content.

In the above image 2.4 material is collected from a set of parameters such as Base Color (the basic color of the material collected from several textures), maximum value Metallic (a parameter describing the presence of metallic properties in the range from 0 to 1), average value Roughness (a parameter describing the roughness or smoothness of the material) and Normal (the map of material normals). This set of parameters creates the material that can be seen in the image 2.4 in the upper left corner. Most of the materials used within the work use this preset or are even more primitive. Due to the fact that Unreal Engine solves most of the implementation on its own, it will define the object normals itself if you don't specify them and so on.

[1] WYSIWYG is an acronym for What You See Is What You Get.

[2] Starter Content pack is a list of basic content such as meshed, material, etc provided by Unreal Engine developers.

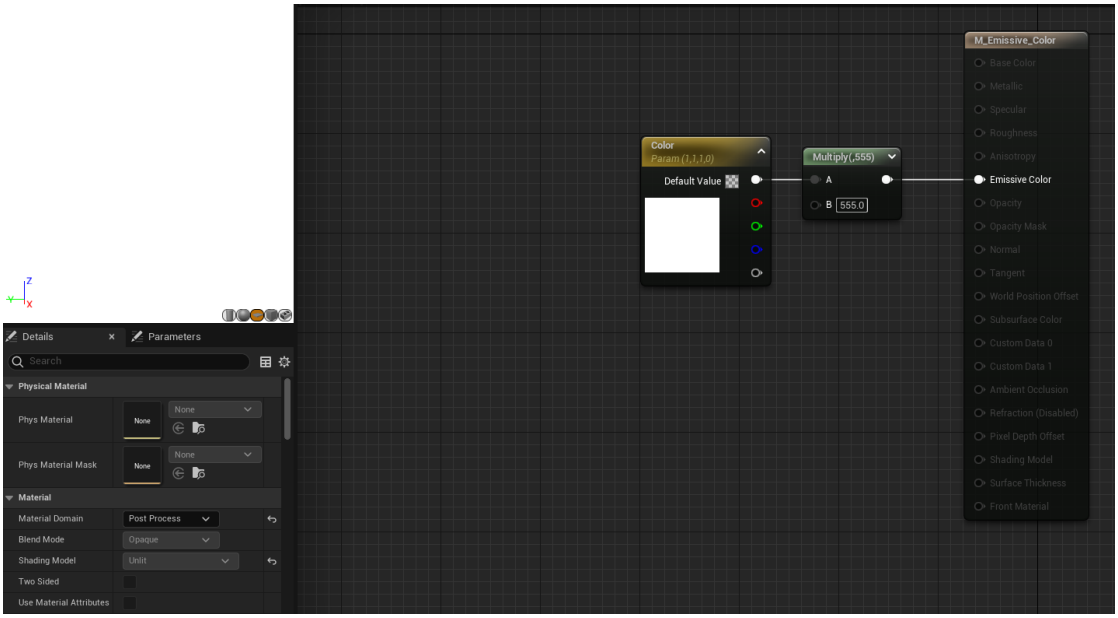

Figure 2.5 Node graph of simple emissive material.

An emissive material in computer graphics is a material that is itself a light source. Unlike materials that reflect or transmit light, emissive materials create light themselves, making them useful for creating luminous or glowing objects in a scene. While it is technically difficult to implement such materials, it is quite easy within the Material Editor. In the figure2.5, may be observed the implementation of such a material by changing the material domain and passing a large value to Emissive Color. This material is subsequently used as part of the work to realize light sources within the imported IFC files.

2.4 Collision

Collision is a very broad term, in engines and visualization it means the interaction between objects, and more precisely, their collision (you can also meet the notion of intersection) and its result. There is such a process as collision detection. This process is the result of mathematical calculations, its task is to determine the total number of objects in the frame, sift out the overlapping ones and determine collisions between them to calculate further interaction. You can often meet the concept of collider - it is some invisible object, a simplified shell that is assigned to the object and sets its shape allowing the engine to understand whether the object in the frame collided with something or not. This is a narrower term, although in essence it works with the same functions and is basically a collision shell. We will touch on it a bit further in the text.[11]

2.4.1 Manual collision creation

There are two ways to create collision on your own. Using primitives in the engine itself in the Static Mesh Editor - Collision section, or in a 3D editor (Maya, Bldender, etc.). When manually creating, the collision must match the name of the asset for which it was made, usually it has some additional prefixes to the main naming. For example, if your asset is named SM_Wood_Chair_A, then its collision will be named UCX_SM_Wood_Chair_A.

For the example, a collision of the specified mesh will be created by means of the Collision Editor.

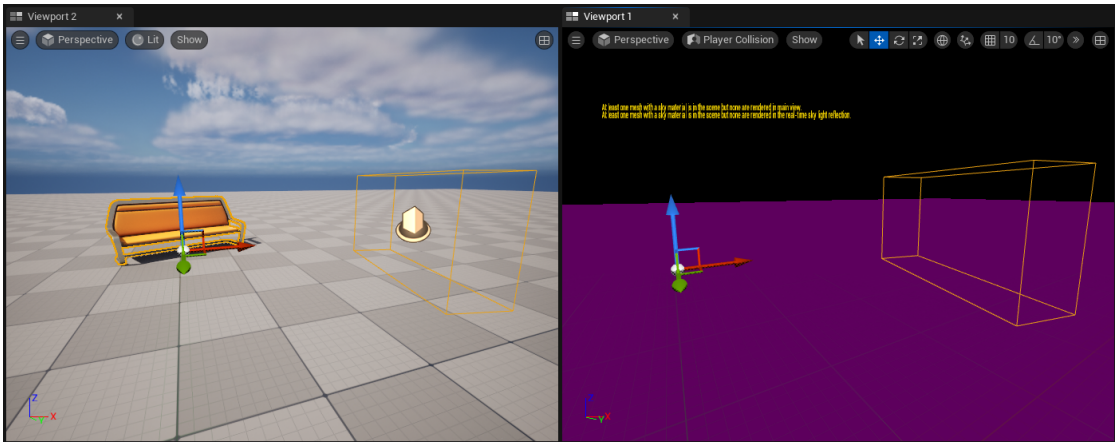

Figure 2.6 Mesh without collision and basic collider. Lit on the left side and Collision on the right side

The above image2.6 shows that the mesh has no collisions. Expanding the mesh itself as an asset will take you to the collision editor.

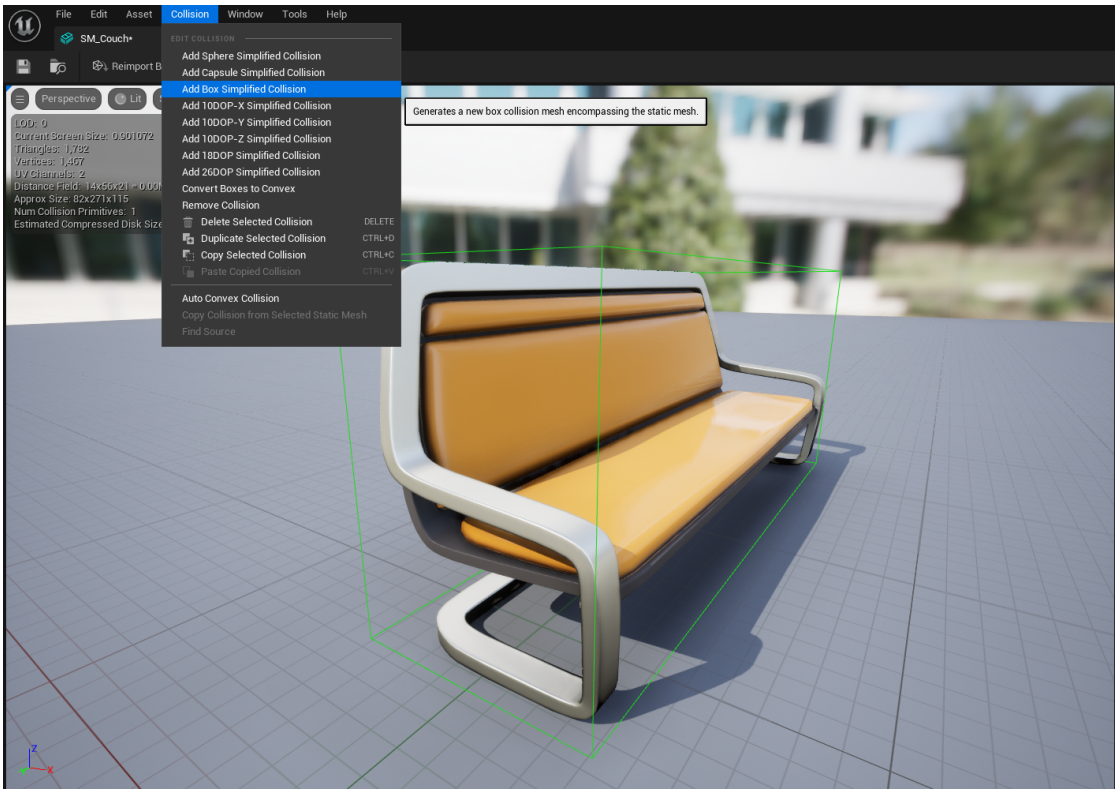

Figure 2.7 Collision Editor. The context menu with basic collisions is opened in the upper part of the image.

Clicking on the collision tab at the top will open the context menu.

Among the existing options for creating a collision, the following variants can be found:

- Basic shapes – Primitive forms to which all kinds of transformations are applied.
  - **Box**
  - **Sphere**
  - **Capsule**
- KDOP – Are simple collision generators, where K is the number of planes projected along the selected axis (X, Y, Z).
  - **10** – box with 4 beveled edges[3] (faces) on one of the axes.
  - **18** – box with all beveled edges.
  - **26** – box with all beveled edges and corners.
- Auto Convex Collision – Another option for creating collisions. And although it is called "Auto", there is a lot of manual work to be done with it. Selecting it opens an additional window with three parameters:
  - **Hull Count** – Is the total number of primitives of the final collider.
  - **Max Hull Vertex** – The total number of vertices that the collision will have.
  - **Hull Precision** – coefficient of collision projection accuracy.

  This solution is imperfect and will require repeated generation of the collider in order to obtain a suitable result. It is important to pay attention to the fact that the final collision of this method is complex, not simple.

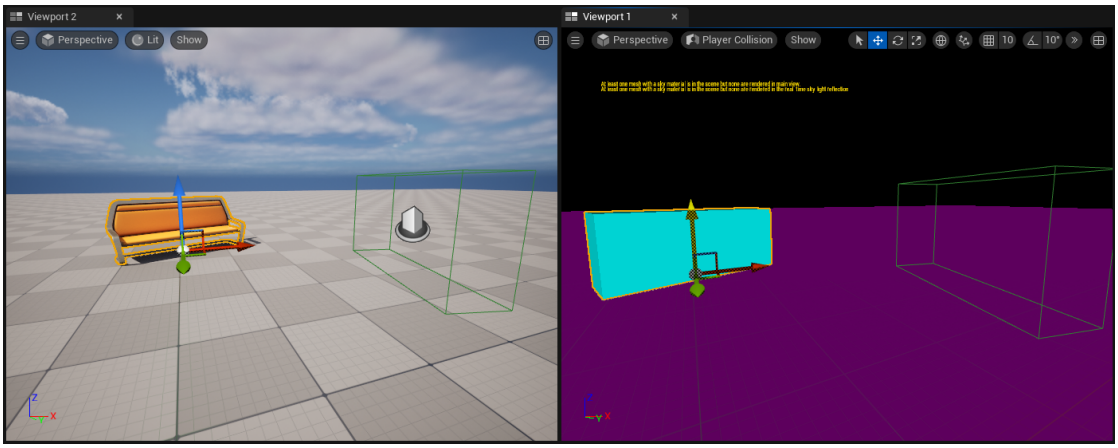

Figure 2.8 Result mesh collision

In the image2.8, may be observed the added basic cube collision to our meshes.

Collisions can also be added within the scene itself as primitive shapes. This is a quicker and simpler solution that may allow, for example, to traverse several meshes with a single collider, however, adding collisions at the mesh level will apply the collision to the instances of the mesh located at the level.

[3]Beveled edges means that the edges (faces) of a primitive collision shape have rounded corners or smoothed edges. This makes the collision shape more suitable for handling collisions with other objects in the scene

2.4.2 Common

Any collision also has Collision Presets, the presets help customize how the asset interacts with various environment elements and events. This is a separate complex topic that will not be dealt with in this paper. It is enough to know that by selecting the BlockAll collision preset the object will be an obstacle or a surface on which you can move.

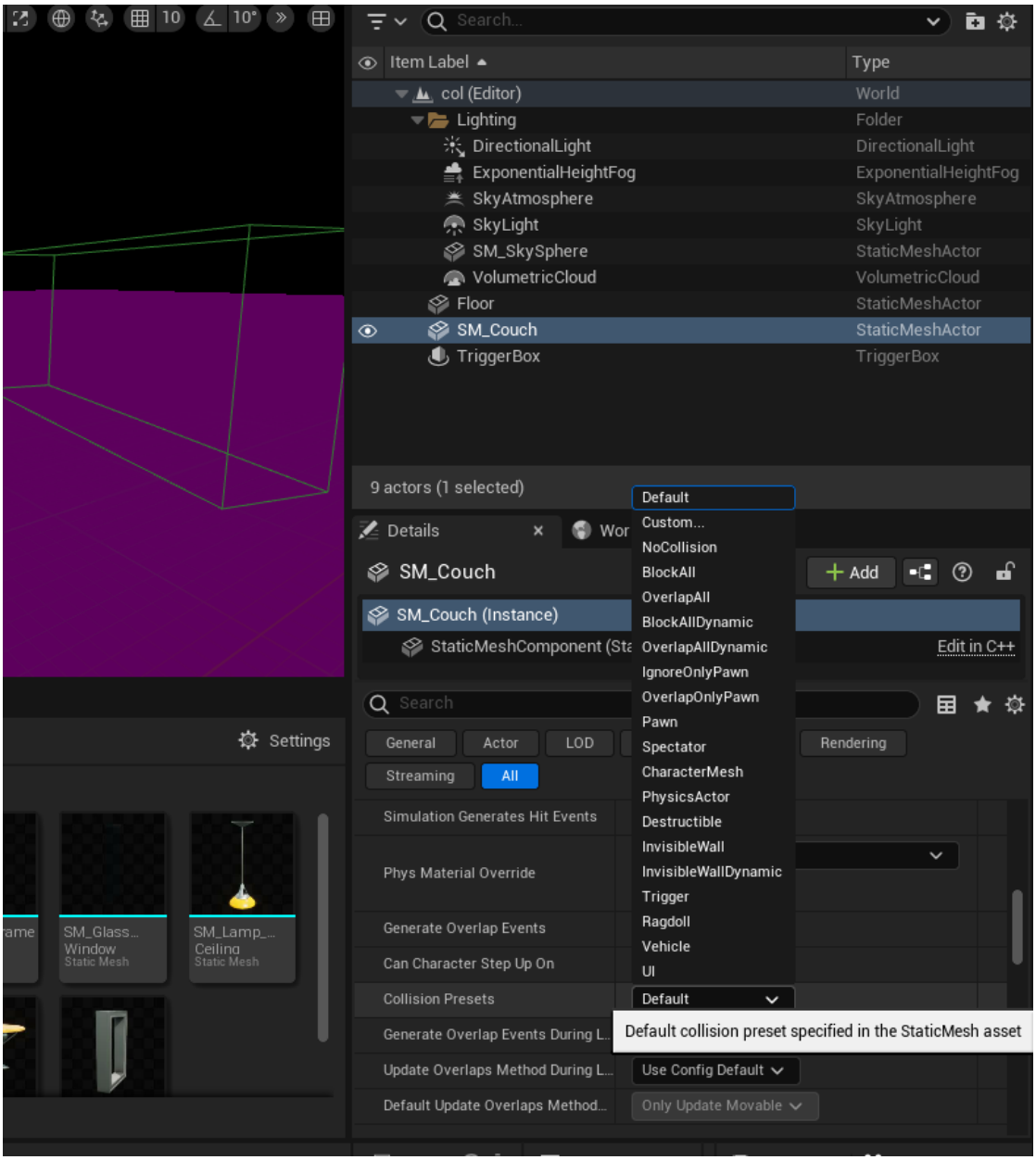

Figure 2.9 Details panel with collision response settings.

You can customize the collision response both within the mesh itself and at a specific object on the stage. The Collision Presets section must be found in the details panel and the Collision Presets section in it.2.9.

2.5 Hardware and Software Specifications

The main development platform is Windows. Although the developers claim that Unreal Engine is compatible with Mac and Linux, these solutions are unstable and very problematic, so in this section I will only describe the recommended technical requirements for Windows and used for research.

- Operating System
  - **Recommended**: Windows 10 64-bit version 1909 revision .1350 or higher, or versions 2004 and 20H2 revision .789 or higher.
  - **Used for research**: Windows 10 64-bit version 1904
- Processor
  - **Recommended**: Quad-core Intel or AMD, 2.5 GHz or faster
  - **Used for research**: 16-core Xeon W-2245 3.9 GHz
- Memory
  - **Recommended**: 8 GB RAM
  - **Used for research**: 64 GB RAM
- RHI Version
  - **Recommended**: DirectX 11/DirectX 12/Vulkan: AMD (21.11.3+) and NVIDIA (496.76+)
  - **Used for research**: DirectX 12
- Video card
  - **Recommended**: Vastly different depending on the features you want to use. For our subject, NVIDIA RTX-2000 series and higher will be more than enough.
  - **Used for research**: NVIDIA GeForce RTX 3090

It is worth realizing that some of the recommended requirements in the context of this research paper are somewhat overstated. It is not even necessary to run a world simulation to study the results of the import outcome. All the work can be done within the framework of an editor like this one.[12]

Chapter 3

Formats

3.1 Understanding File Structures

Since this work will deal with methods of processing different types of data, it is necessary to have a minimal idea of which formats represent what. BIM is an intelligent 3D model-based process in which all stakeholders can coordinate, collaborate, and share information. BIM files are usually organized in a hierarchical structure where at the top level are project files that contain data related to the entire building project, and below that are discipline-specific files that contain information related to each individual discipline, such as: architecture, landscape architecture, construction, structural, mechanical, and lighting/electrical engineering. Each of these disciplines implies a separate layer of information, represented by a separate unique set of dates, which is only partially related to the other disciplines or not related at all. Finally, individual component families contain information specific to each building component. Understanding these file structures is key to efficient and effective use of BIM technology.

3.1.1 BIM File

The term BIM is an abbreviation for Building Information Model - Building Information Model or Building Information Modeling, if we are talking about process or technology. Russian legislation has a similar term - information modeling technology or BIM. When they talk about BIM technology, they most often mean the creation in specialized programs 3D-geometry of the building with associated attributive information (for example: material, article or price). The construction of a BIM model is based on the principle of object design, i.e. the assembly of a complex model from elements, each of which belongs to its own class - windows, walls, floors, fittings, etc. The class of an element defines a set of properties and behavior of the element. Thus, BIM-model, can be called a copy of the building in a virtual computer environment. With the spread of BIM-technologies, the use of information modeling is already spreading to other stages of the life cycle of a construction project. A BIM file architecture system relies on a database that keeps relevant information about building design, construction and maintenance. [13]

Considering appropriate parameters for the evaluation helps to identify unique characteristics between different BIM formats files since each possesses its own structure and data repository defining how project information or individual components/attributes are arranged. Structural organization determines how information is accessed in each format which may have little influence on the reading or importing activities.

3.1.2 IFC File

The Industry Foundation Classes (IFC) Format is widely used in the architecture, engineering, and construction (AEC) industry to exchange information between different building software and systems [14]. It was developed by the International Alliance for Interoperability (IAI) as an open standard to facilitate communication and collaboration across disciplines and organizations. IFC files contain data about the building geometry, properties, relationships, and model views, and they can be imported and exported by many software applications.

High interoperability has come to be used as a standard in the production domain, where in each iteration of the working pipelines the final product of production must be an IFC file.[15]

This interoperability enables stakeholders to work together more efficiently, reduce errors and duplication, and enhance the overall quality and sustainability of the built environment. Critics argue that IFC is too complex, verbose, and outdated, and that it hinders innovation and customization. However, proponents counter that IFC is constantly evolving and improving, and that it provides a common language that transcends hardware, software, and culture.

In addition to IFC, there are two derivative formats of IFC:

IFC-XML is a derived format from the standard IFC that represents the same data structure as standard IFC files, but in XML (*eXtensible Markup Language*[1]) format. XML is a markup language used for storing and transmitting structured data in text format. The IFC-XML format provides a more flexible and human-readable way of representing data than the binary format of standard IFC files.[17]

IFC-ZIP is a way to package IFC files and related data into a single ZIP archive for easy transfer and storage. This format is typically used to group multiple IFC files and associated resources (e.g., texture files, images, etc.) into a single file for ease of data sharing and portability. Typically, an IFC-ZIP file contains both the structured IFC-formatted data and the associated materials needed to fully represent a building or infrastructure model.[17]

[1] eXtensible Markup Language (XML) is a markup language used to organize and store structured data in text format. It is a set of rules for creating custom tags and attributes that can be used to describe data.[16]

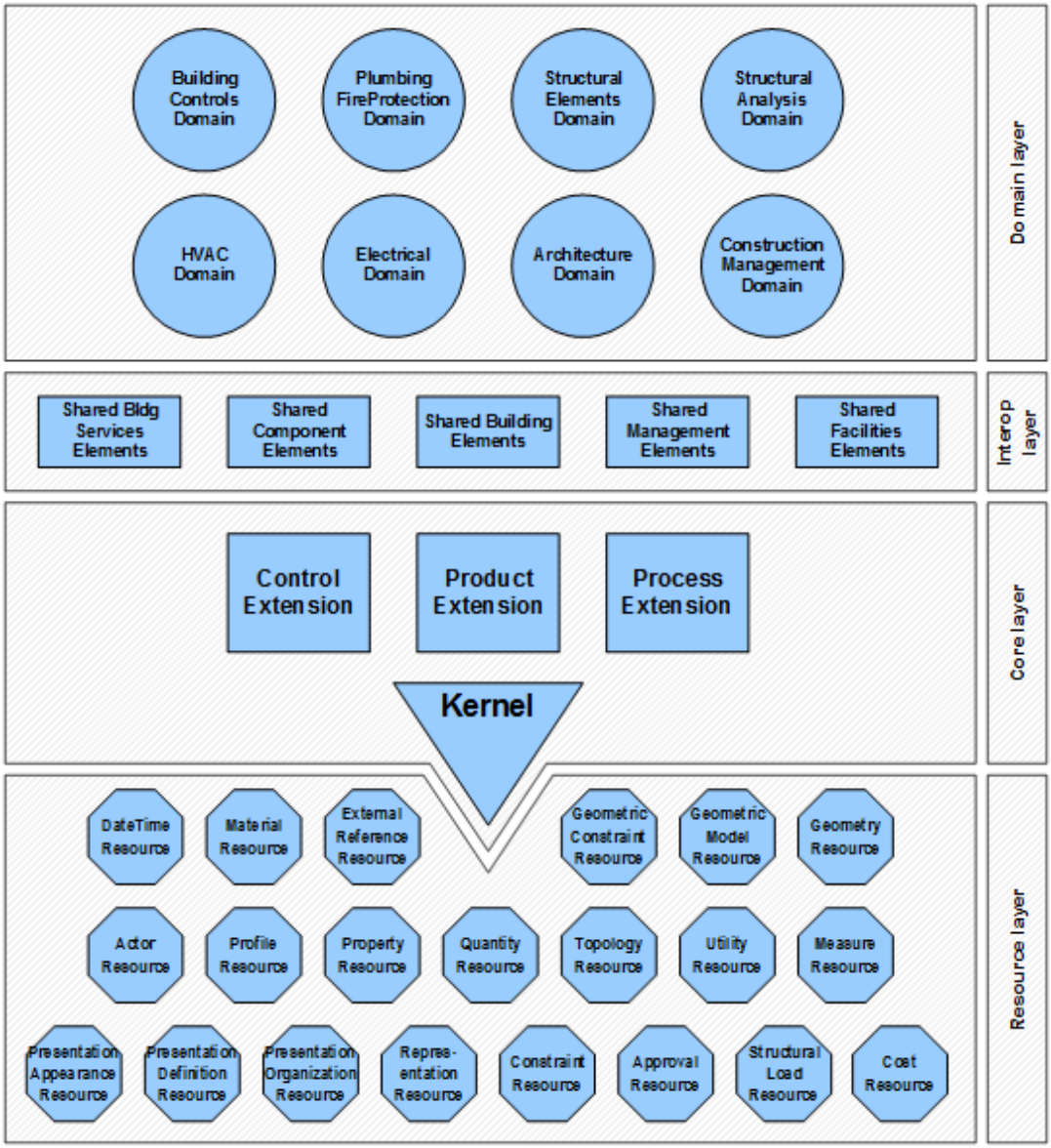

Figure 3.1 IFC domain architecture and structure.[1]

IFC file as a representation of the BIM format group is divided into several domains including:

Resource layer layer containing the basic objects and entities represented in the IFC standard, such as walls, windows, doors, floors and other building or infrastructure elements. This layer defines the basic structural elements that are used to create models of buildings and other objects.

Core layer defines the basic abstract concepts and data models that underpin the IFC standard. Includes objects, relationships, attributes and other elements that are used to describe and model buildings and their components.

Interoperability layer defines mechanisms and standards for information exchange and interoperability between different software and systems using the IFC standard. Includes specifi-

cations for file formats, exchange protocols, and technologies that allow different systems to exchange data and work together.

Domain layer defines the specific application areas and knowledge domains in which the IFC standard applies. Includes various industry domains and disciplines such as architecture, engineering systems, construction, real estate management and others. Each domain has its own characteristics and requirements for modeling and data exchange, which are taken into account at the domain level.

The specific specifications of the domains are configured within a specialized softwire to work with IFC files. Due to these specifications data volumes and data types located within domains can be varied. All information describing IFC file domains was found in the official developer documentation.[1]

3.1.3 DWG File

Engineering design and computer-aided design heavily rely on the widely used DWG (from drawing) file format which is a standard format. It has the ability to store various data types and boasts of a rich structure.

Storing graphic elements, like lines, circles, arcs, polylines, and text, is the DWG file's expertise. Every element possesses its distinct coordinates, defining where it belongs in either 3D or 2D space.

Organizing elements into different layers is made possible by the format, granting you the ability to control their settings and visibility. Drawing components can be reused by utilizing blocks, which are a set of graphical elements. For calculation and analysis purposes, files are equipped with information on both coordinates and points. DWG grants the power to store various display choices like line styles, strokes, text, and customizations. 3D models are supported by geometry, both in 2D and 3D formats.

3.1.4 UASSET File

Containing data and metadata about game assets, uasset is a binary file format utilized within the Unreal Engine game engine. This format serves as a managing system for a diverse range of game assets such as 3D models, textures, animations, sounds, and more. Assets within the uasset structure are organized at multiple levels, integrating seamlessly into projects developed within the Unreal Engine. Handling assets and resources with ease, uasset format is an essential component of the infrastructure that supports game development using the Unreal Engine.

Binary data optimized for the Unreal Engine game engine is what the uasset format stores. The format's purpose is to hold a range of assets, like sounds, animations, textures, and 3D models. Every uasset file is dedicated to a singular asset, and its structure is specific. Binary codes and data structures are utilized within the file to describe the data, allowing for rapid access to the data during gameplay, and efficient storage of information.

The arrangement of assets in a hierarchical order is made simpler thanks to the uasset structure. The uasset carries metadata that plays a role in managing the asset, making certain that it's utilized properly in the project. Managing a large number of assets in complicated game projects is made easier through this structure.

3.1.5 FBX file

Developed by Autodesk, FBX (Filmbox) is a widely used file format in 3D modeling, animation, and visualization, providing a transfer method for 3D data across various platforms and software. In an FBX file, there are geometry, materials, textures, animations, lights, cameras, custom

properties, object hierarchies, and metadata. In the 3D graphics and animation industry, this format is paramount for interoperability, allowing effortlessly exchange data across programs and projects.

3.1.6 Compatability of formats

The standard UE format has always been FBX. Depending on the version of UE it accepts all versions of FBX released earlier, however, if you need to import the newest version of FBX file into UE, there is an option to import this file into Blender or other 3D editor to export as an older version of FBX. The undertaking of incorporating BIM data formats, such as IFC, into Unreal Engine's uasset format poses a potentially valuable, albeit challenging opportunity. It is essential to establish efficient import techniques, attribute mapping, and careful consideration of data structure in order to bridge the gap between the gaming and architectural data standards. The solution to this problem deserves a separate research paper, so I will use a ready-made solution provided by the UE developers (Datasmith).

Datasmith was developed to import and process BIM files. Using a suite of tools and plugins called Datasmith, users may import sophisticated assets and complete pre-built scenes from a range of industry-standard design programs into Unreal Engine.

3.2 Processing and translation

The process of converting data from BIM to uasset format is a complex task, given the differences between these formats and their purpose. A significant impact on the final result can have both the original file and its settings and subsequent manipulations with it.

In work, dealing with the transfer of BIM models to Unreal Engine, little attention has been paid to performance aspects, and even when this is taken into account, the most efficient model transfer techniques are not used in this case, so computing power is used inefficiently. The most efficient model transfer techniques are not used, resulting in inefficient use of computing power. There are many solutions to automatically build interactive visualizations of BIM models. They can be divided into two groups: solutions that fully automate model visualization and solutions that automatically export geometries to the most popular 3D engines.

The advantages of automated model visualization solutions is that they do not require the user to perform any action related to visualization creation. On the other hand, the quality of the visualization is usually not the highest and there are no options for fine-tuning the visualization (available in modern graphics engines) and implementing your own interaction logic.

Automatically exported solution allows you to avoid intermediate editing of the model in the 3D editor and minimize its further adjustment in the engine (Adjustment of materials, placement of light sources, etc.). Clearly, this automation significantly reduces the time required to create interactive visualizations. However, there are currently no solutions that can efficiently export complex BIM models. For these reasons, BIM models need to be manually intermediated before they are transferred to the engine in order to create interactive visualizations that use the available computing power as efficiently as possible.

3.2.1 Datasmith

The content of this chapter is an extract of information on the topic of work from official documentation [18].

A set of utilities and add-ons called Datasmith facilitates the import of material into Unreal Engine 5. The goal of Datasmith is to address the unique problems encountered by non-gaming businesses wishing to leverage Unreal Engine for real-time rendering and visualizations, such as

manufacturing, engineering, building, live training, and architecture. However, game developers that have comparable issues with their asset pipelines could also find it interesting.

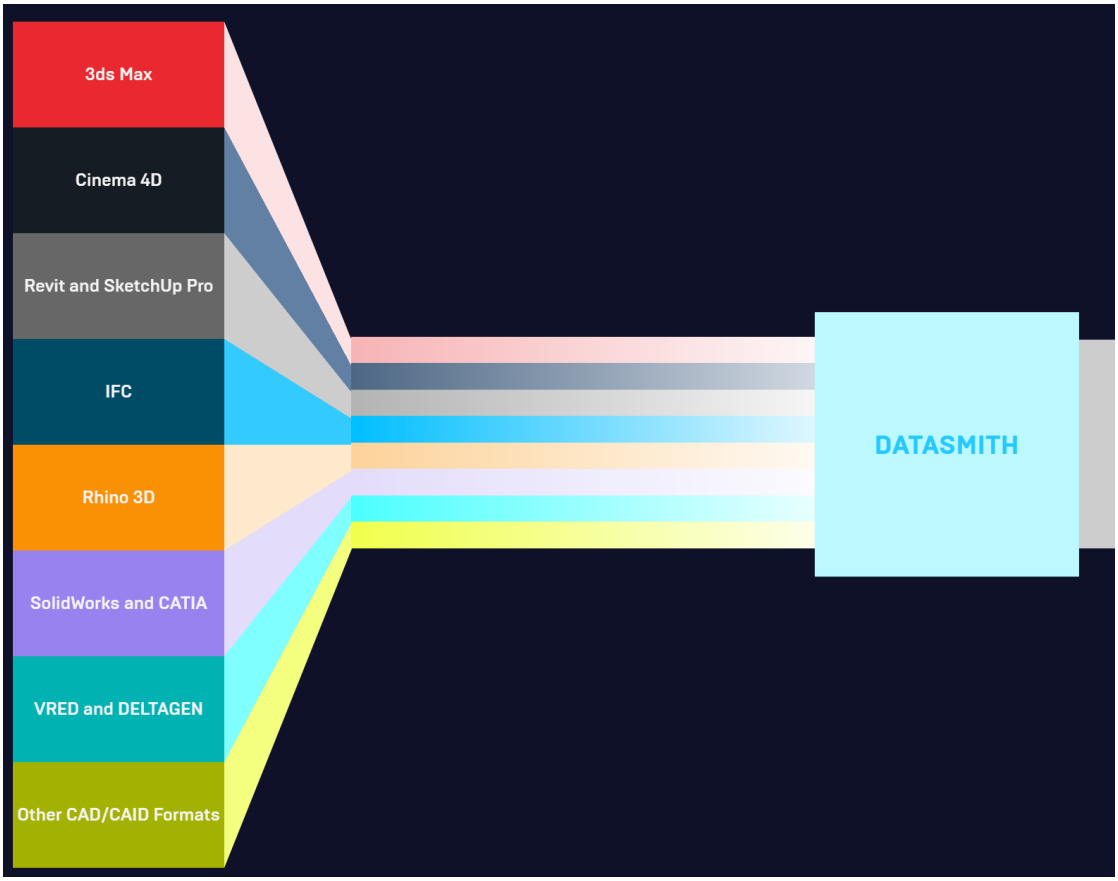

Figure 3.2 Datasmith compatability.[2]

Datasmith has set lofty objectives, which involve effortlessly incorporating entire pre-built scenes and intricate structures into Unreal Engine, regardless of their size or intricacy. It eliminates the necessity of dividing scenes into separate fragments for import and effectively maximizes the usage of existing resources and layouts from other design instruments.

With each release, Datasmith aims to ensure compatibility with a broad range of 3D design applications and file formats like Autodesk 3ds Max, Trimble Sketchup, Dassault Systèmes SolidWorks, among others. The platform also tackles the obstacle of integrating modifications made to the original content without necessitating a total overhaul of brought-in resources in Unreal.

At present, Datasmith is primarily dedicated to the task of converting design material into formats that can seamlessly work with Unreal Engine in order to facilitate real-time rendering. Looking ahead, their aspirations are aimed at improving the process of intelligent data preparation, ensuring that imported content is optimized for efficient performance during execution. Additionally, they aim to integrate more advanced and intelligent behaviors during runtime operations.

3.2.2 Alternative methods

Alternative methods of importing IFC files into Unreal Engine do not necessarily require additional softwares or plugins. For example, it is possible to translate a file into FBX format, which

is the standard for Unreal Engine accepted meshes, but this implies that additional problems may arise as part of the intermediate iteration. There are also custom solutions to solve specific problems [19].

Chapter 4

Data processing and requirements

There are a fair number of methods for accepting Unreal Engine BIM files:

1. Transit format – some devices support translation from IFC to FBX which is directly accepted by Unreal Engine and is the engine's primary format.
2. Connecting third-party plugins, as for example, 3drepo or Modumate, which have quite a lot of functionality, but it does not exceeds the native solution and implies partial use of Unreal Engine command lines and use of the internal API.
3. Others – imply additional qualification on the part of the user.

   Taking into account all of the above, it can be argued that Datasmith is the most suitable solution.

4.1 Software prerequisites

To study the import method I will use Unreal Engine version 5.2.1, although it is worth noting that Datasmith is compatible with almost all versions starting from 4.0.2. Among additional third-party software I use Electronic Nodes, which simplifies the work with Blueprints a bit, but it is entirely cosmetic and is absolutely not important or necessary for importing BIM files. More detailed steps will be described within the following chapters.

4.1.1 Setting up project and Datasmith

If you use Unreal Engine 4, you will be necessary to manually connect the plugin to the project. This is done by going to the plugins section in dropbox of the Edit tab located in the upper left corner of the Unreal Engine window.

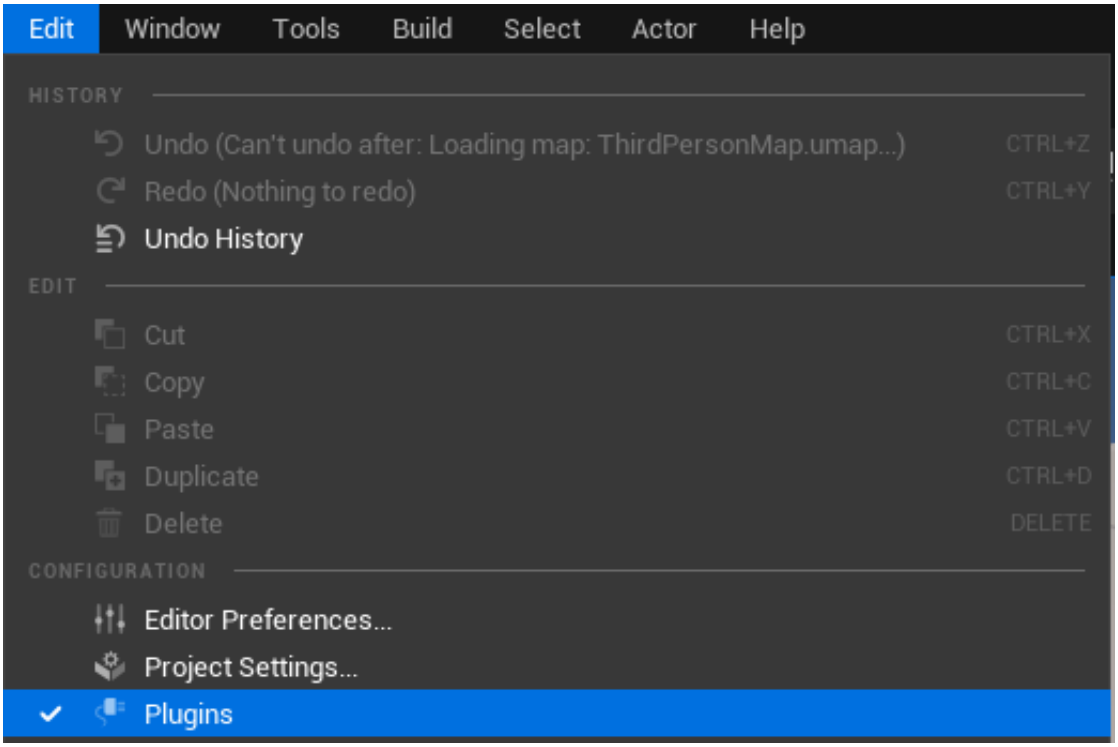

Figure 4.1 Plugins button location

Find the Datasmith plugin or the modification and check the checkbox. In case of use Unreal Engine 5, no additional steps are required as Datasmith is a component of it.

4.1.2 Electronic nodes

Electronic nodes is a custom plugin that modifies the standard Unreal Engine user interface and allows to customize it according to liking. A characteristic feature of this plugin is that it uses angular linear curves instead of traditional curves typical for Blueprint scripting.

4.2 Input/Output

This chapter will describe the characteristic features of import, the difference between the characteristics of the object in its original format and after import according to the official documentation [3] and the results of own experience.

4.2.1 Hierarchy

Datasmith creates an Actor for each object presented in your IFC scene. The name of each Actor is prefixed with the IFC type of the corresponding object. These Actors are then organized in a parent-child hierarchy of the Datasmith scene that exactly follows the layout of your IFC objects, so it is important to keep the hierarchy correctly in the original file to avoid problems before and after import.

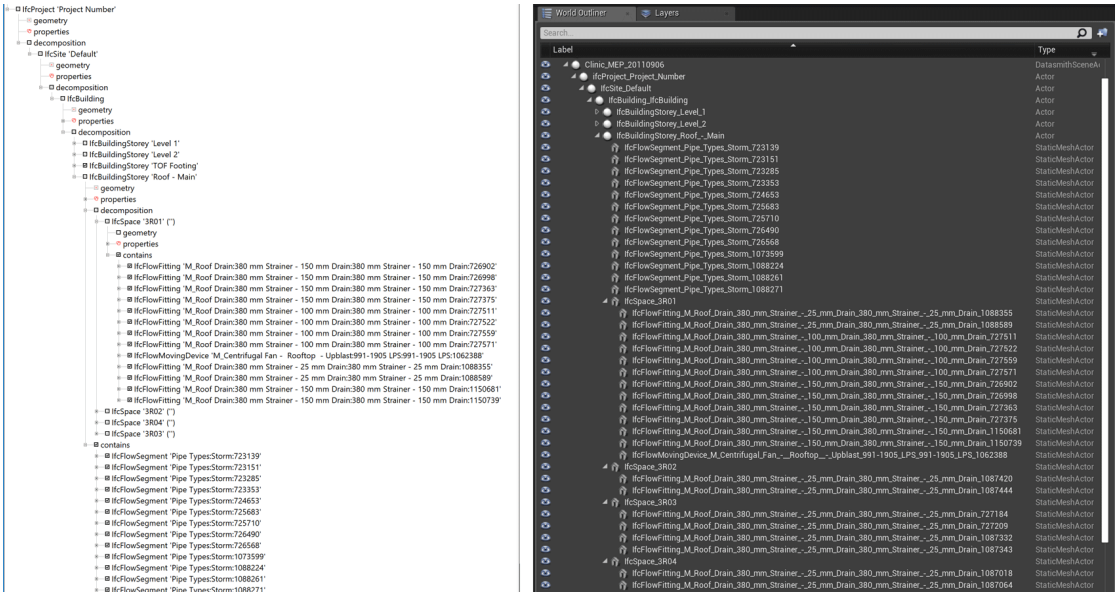

Figure 4.2 Hierarchy of IFC file on the left side and hierarchy of scene in Unreal Engine on the right side.[3]

In the Unreal Editor World Outliner, the actors at each level of the hierarchy are always ordered alphabetically. This can lead to visual differences in the ordering of siblings between Unreal Engine and other IFC viewing and editing applications, but the relationship between parents and children remains the same.

While Unreal Engine requires unique names, IFC allows multiple objects with the same name. For this reason, during import, Unreal will pre-generate a numeric suffix to avoid collisions.

4.2.2 Metrics

The standard unit of measurement in Unreal Engine has always been the centimeter. Some of the applications that work with BIM files have different standards, for example inches. Datasmith uses metadata to understand what units of measurement the original objects were in and when importing, recalculates them in Unreal Engine units so that the original transformation of the imported object will be standard, while rescaling operations will be performed on the object.

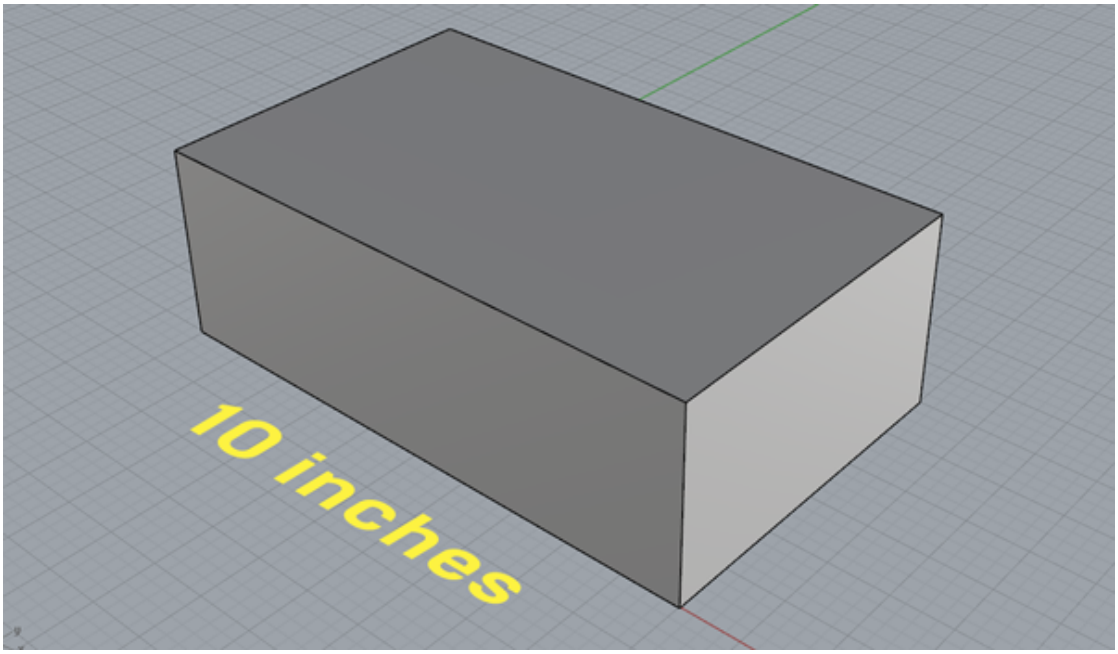

Figure 4.3 Initial dimensions of the object within the original software.[3]

Thus inches as a unit of measurement in the source application will be converted from an object with a length of 10 units in the source scene to a length of 25.4 world units in Unreal Engine.

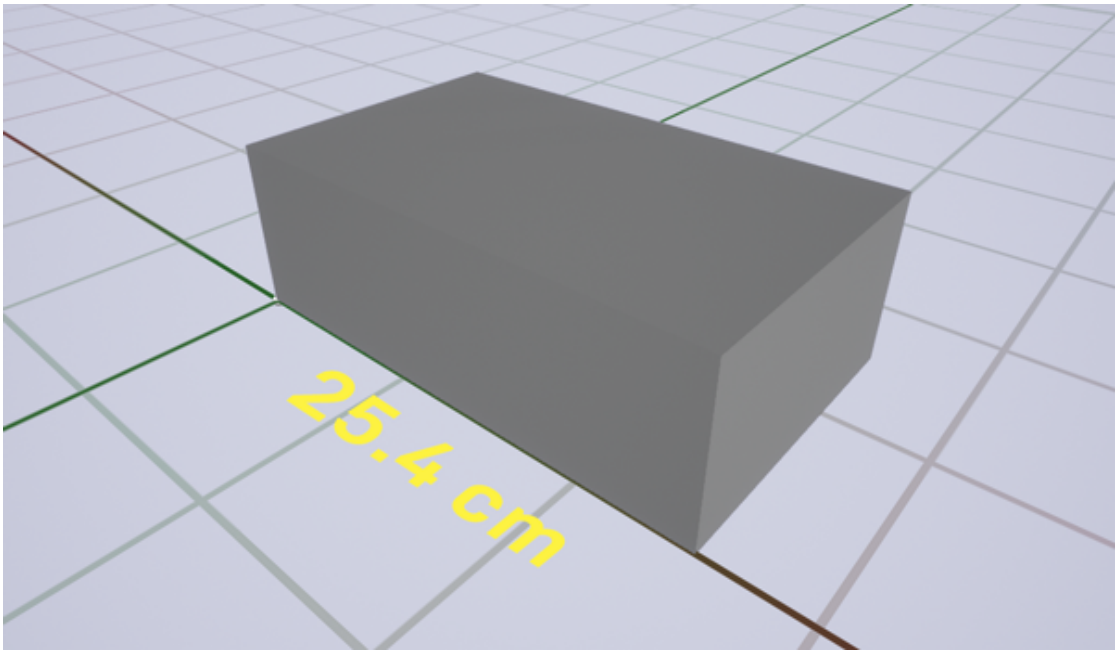

Figure 4.4 Object dimensions after rescaling as a result of importing.[3]

With complex transformation values for scene objects and objects that are in the lower layers of the hierarchy that are compensated at the level of individual objects, the complexity of

coordinate system transformations can cause the results in Unreal Engine to be inconsistent with the original scene. To fix this problem, you must go back to the original scene in your softwire and simplify the transformations within the scene hierarchy

4.2.3 Materials

Datasmith locates a Material Asset in Unreal with the same name for each surface material it discovers in your IFC scene and adds it to the Materials folder adjacent to your Datasmith Scene Asset. To shade the surfaces of the Static Mesh Assets it develops, Datasmith allocates these Material Assets to them.

Every Material in the Materials folder is a Material Instance, exposing the color values, transparency values, specular colors, and other attributes defined in the IFC file.

Depending on your source application, Datasmith provides distinct Parent Materials to your Material Instances based on the principles described below.

Generally, Datasmith assigns a pre-existing Parent Material that comes with the Datasmith plugin to each Material Instance. The material authoring parameters that are available in your source application are typically comparable to the pre-defined settings that are provided by these Parent Materials. Within each Project, Datasmith generates Material Instances, each of which has exposed parameters that are available for modification. Every Parent Material that Datasmith provides has a unique collection of attributes.

Datasmith additionally generates new Parent Materials in your project, located inside the Materials/Master folder, if you're importing content from 3ds Max or Rhino. Compared to most other source software, 3ds Max provides a significantly more robust Material creation method that is essentially comparable to Unreal Engine Material graphs. Consequently, instead of needing to reuse pre-set Parent Materials with a pre-set graph and a pre-set list of exposed options, Datasmith is usually able to construct new Master Materials that are extremely similar to the custom Materials you have in 3ds Max.

In this case, Datasmith typically still creates Material Instances for those custom Parent Materials. To be more specific, it creates Master Material and then Material Instances to apply them to Static Mesh Assets and to the Static Mesh Actors located at the scene. Modification of each Material may be implemented via attributes of Material Instance. For some types of 3ds Max Materials, Datasmith may skip creating the Material Instances. It is solved via applying same Parent Material to Static Mesh Actors and Assets.

4.2.4 Collision

By default, all data imported with Datasmith is collision-free. The only exception is 3Ds Max where it is possible to manually create collisions that will be recognised by Datasmith during import [20]. This is due to the fact that most software supporting IFC decide to break the scene into other layers such as: a layer of electrical wiring, plumbing layer, etc., and collision layer is not presented, while 3Ds Max is focused on design-projecting and animation and in some cases can use collision objects to simulate collisions in animation [21].

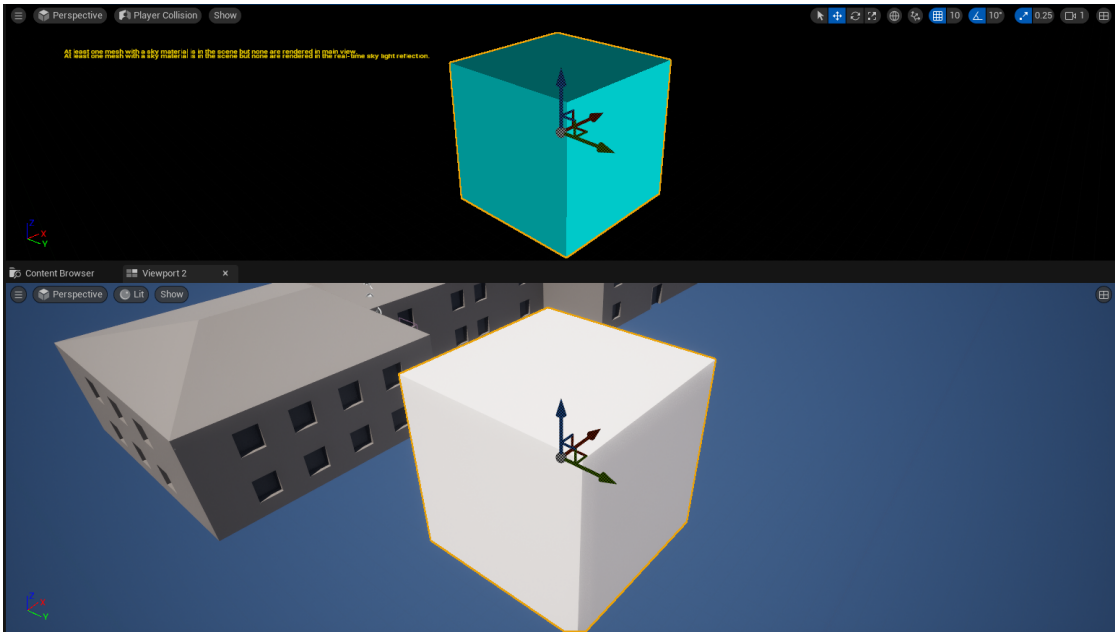

Figure 4.5 Comparing Player collision and Lit layers after import.

However, this does not mean that the imported geometry cannot be given the appropriate collision. After import, the objects become standard uasset and have all their functionality, which means that you can create their geometry using third-party editors such as Blender and others or using the built-in mesh editor in Unreal Engine, which provides the functionality to create complex or simple collision. Due to the fact that each object of the original file is imported as a separate unique mesh (so 20 identical chairs will be imported as 20 separate meshes) this method will definitely not be optimal.

Special attention should be paid to the fact that there is a Datasmith Runtime feature that is at the beta testing stage. It provides access to Blueprint scripting of data imported into Unreal Engine via Datasmith throughout the process and gives the ability to edit them. Also there is a separate functionality for creating and generating collision by means of standard tools of the mesh editor, where Unreal Engine will try to create an optimal collision for imported objects by means of parameters set by user [22].

4.2.5 Metadata

Datasmith imports geometric object metadata that you define in your source application for certain types of source file formats [20]. Python scripts and Blueprint may be used in the Unreal Editor to get this metadata.

Technical metadata values regarding specific scene objects, such as their unique IDs, object classes, or other application-specific information, can be accessed through some third-party applications and file formats [18]. This type of technical data is imported by Datasmith into Component Tags, which are then assigned to the Static Mesh Component, which in the Level reflects the geometry of each item.

For example, the provided test data only had Datasmith_UniqueId (It is the Datasmith-generated ID that is used for identification and naming during the import process), whereas if the object class methods were available, it would be possible to replace duplicate meshes with instances of a single mesh using the instancing technique.

Every property that Datasmith imports from the IFC file is noted, and the values of these

properties are saved as Datasmith Metadata on the Actors that Datasmith constructs to represent those objects in the Unreal Engine. This metadata is accessible during runtime in the Unreal Engine or in the Unreal Editor.

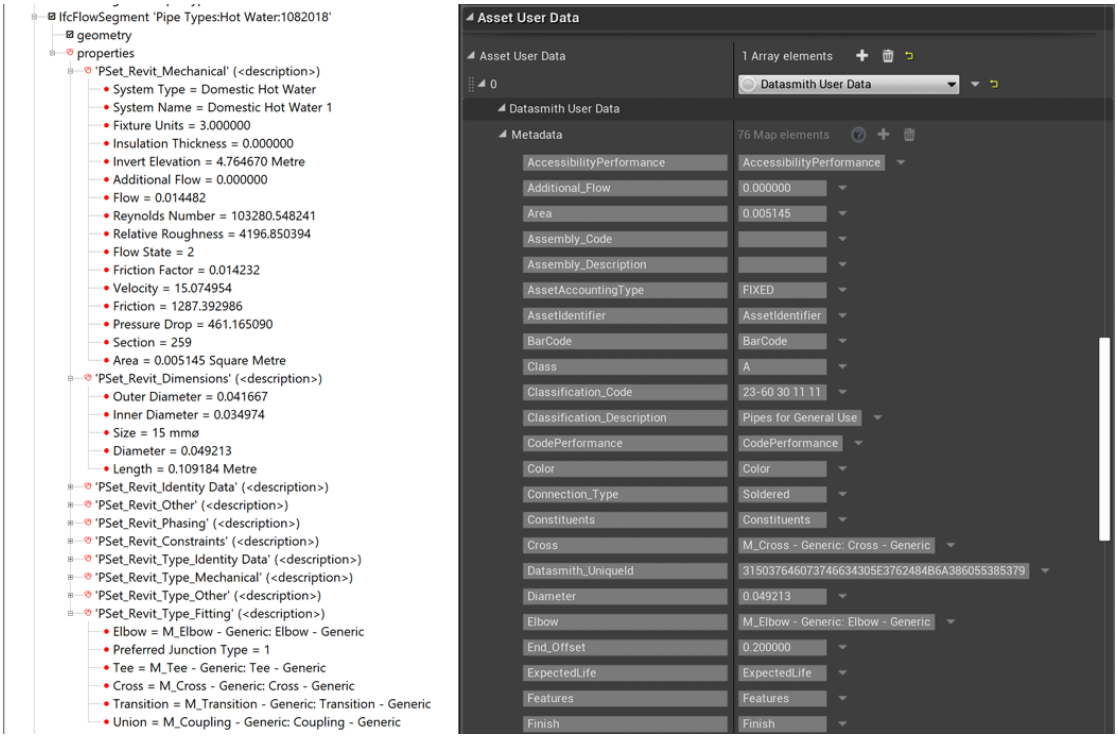

Figure 4.6 Left: an IFC file's attributes. Right: Datasmith Metadata was constructed using those attributes.[3]

Grouping properties is possible with IFC. A few groups, such as PSetRevitMechanical, PSetRevitDimensions, PSetRevitIdentity Data, and so on, are displayed in the image above. But Datasmith in Unreal Engine, metadata is always just a flat list of values and keys. As demonstrated above, if any groups are present in your IFC properties, Datasmith flattens the hierarchy by compiling all of the metadata keys from each group into a single flat list. The collective names are eliminated.

In the metadata key names, Datasmith only permits alphanumeric characters, hyphens, and underscores. Any additional characters in the name of user data will be automatically changed to underscores. For instance, the Fixture Units field in the Datasmith Metadata is changed to FixtureUnits in the image above.

Chapter 5
Result project and features

This chapter is devoted directly to the created project and describes the features implemented by me. The chapter is divided into several sections and represents the whole stage of project implementation, adding test data to it and analyzing the obtained results.

5.1 Provided functionality

The test data provided to me after importing through Datasmith to Unreal Engine and then placing it on the scene contained a huge number of problems such as: missing materials, missing light sources, missing collisions and others. It can be clearly stated that these problems can be solved by means of corresponding scripts, as for example it was already mentioned in the collisions section or replacing the lack of lighting by giving emissive materials to light sources, however, due to time constraints only some of the solutions were implemented.

5.1.1 Material manager

At the time of importing the experimental data provided to me, the main problem was the inability to inspect the data due to the lack of material on the data. In such cases for standard Unreal Engine meshes Static Mesh Actor is used, which refers to one specific parent mesh and due to this all have the same material. After importing, as it was said earlier, each mesh was a unique object, so this method of solving the problem was simply impossible.

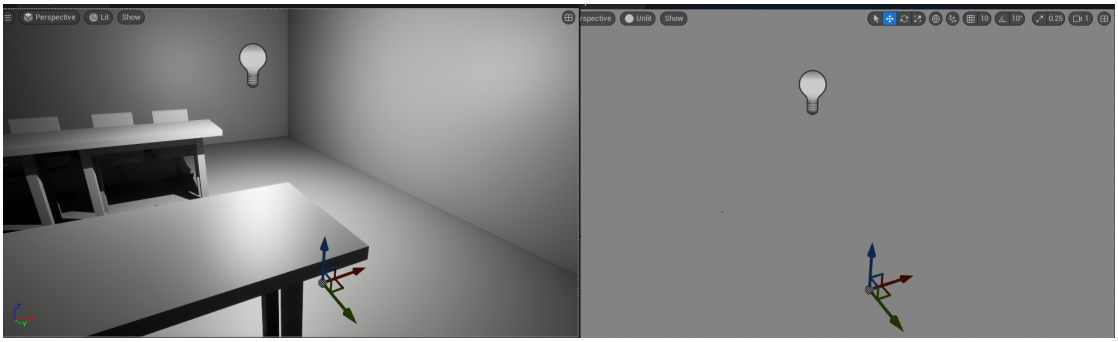

Figure 5.1 Material after import. Light implemented via added point light. Lit on the left side and unlit on the right.

Before describing the solution I have provided, it is necessary to point out the fact that each of the objects located on the stage has a unique name. Thus even several identical objects added to the scene will have a generated unique postfix. Based on this data, I decided to analyze the actors on the stage and provide them with material based on the string of the name of one of them, ignoring the postfix.

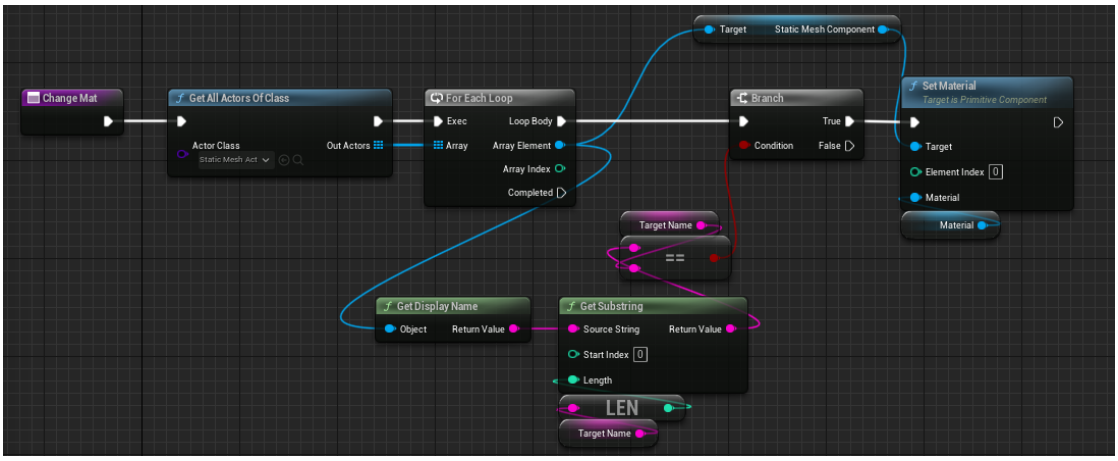

Figure 5.2 A script that changes the material for each actor on the stage with a corresponding name.

Brief description of the script sequence5.2:

1. Retrieves from the current scene all objects whose class is equal to Static Mesh Actor as an array.
2. With the help of a for loop, it checks each received array object for the presence of a substring of the string specified as the ”TargetName” argument. If it does, it continues, otherwise it moves on to the next element of the obtained array.
3. For an element that fulfills the previous condition, changes the material according to the argument specified as ”Material”.

The solution I implemented is to go through all the actors on the stage and compare their name with the string passed as one of the function arguments. The second argument is the material that will be set by all actors whose name matches the string passed as the first argument without taking into account the ignored postfix.

Thanks to the various materials, orientation on the level is noticeably simplified and the emergent nature of the scene becomes noticeably clearer.

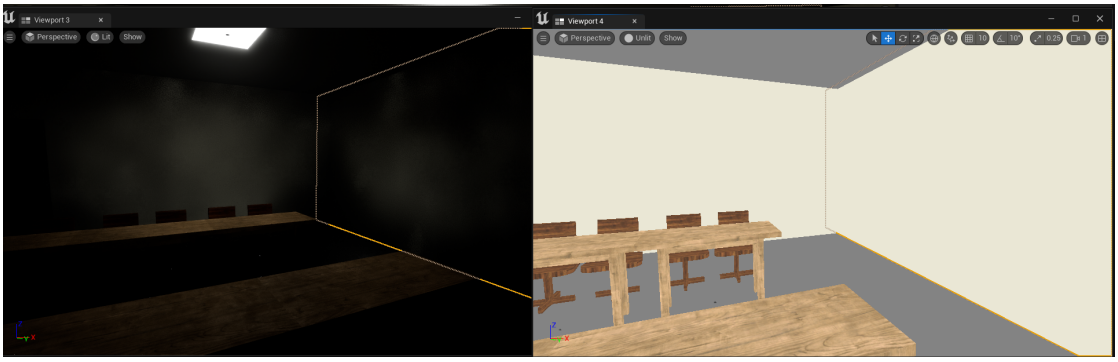

Figure 5.3 Changed materials. Light implemented via emissive materials. Lit on the left side and unlit on the right.

This solution is executed within the framework of the editor and even taking into account that it searches all the actors located on the stage, which is not a big load, it is executed outside the running of the final version of the project that does not reduce the cost of production of the final product to zero.

5.1.2 How to use Material Manager

This actor is located within the project where it can be found using the content browser. It can be used by moving an actor called BP_MaterialManager to the scene and then interacting with its interface.

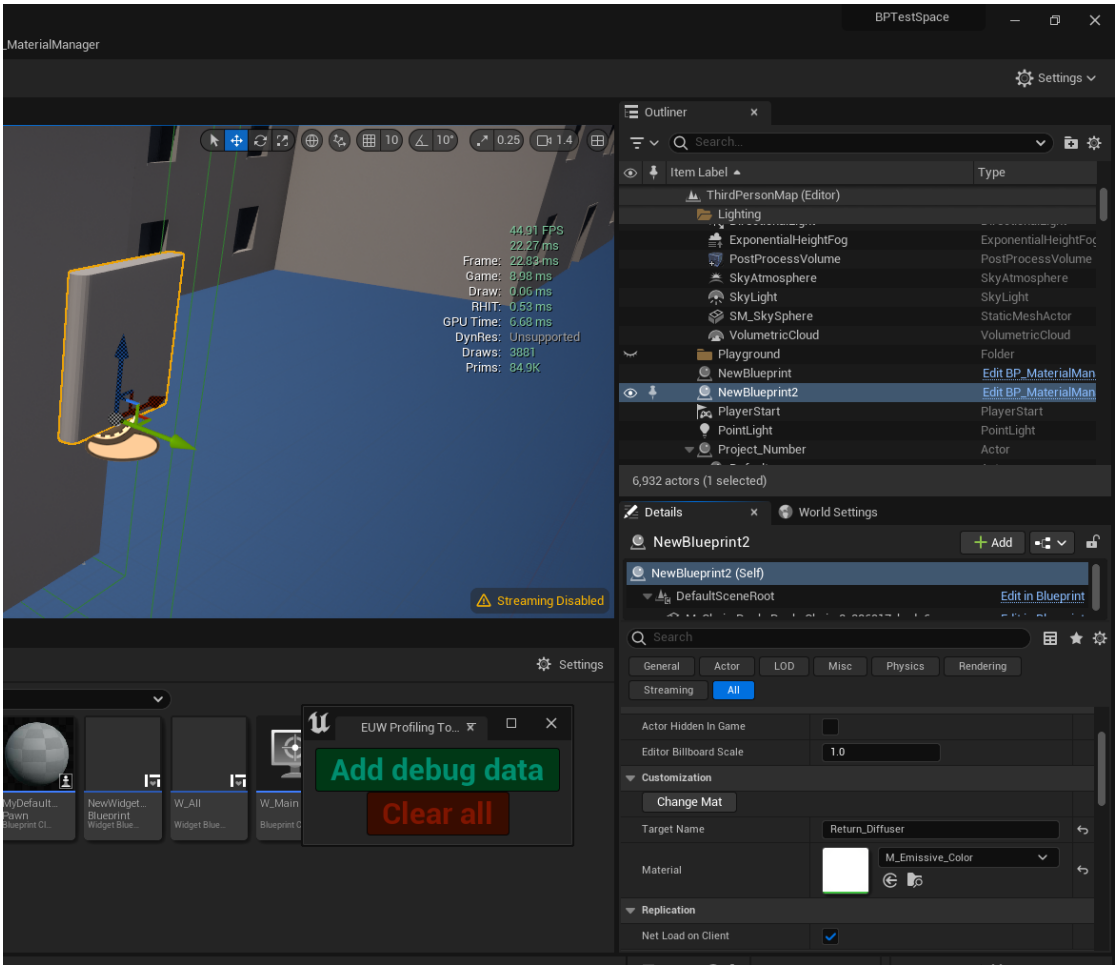

Figure 5.4 Default interface is located in the details panel of actor that is placed on level.

In the bottom right window is the details panel where you can find the customization section. Within this section you need to perform the following sequence of actions.

1. You need to define a common subsequence of object names in which you want to change the material. For example, if you want to add glass material to all windows, you need to find one of the mesh instances, such as Fixed_window_1_477324_body1_57, and take Fixed_window out of it.
2. Enter this string in the details pane at the attribute value TargetName.
3. Select from the Material attribute in the same section one of the existing project materials to be assigned to all target objects.
4. Click on the button above these attributes called ChangeMat.

5.2 Profiling widget

As part of development, you often had the need to switch between different types of statistical information output. In order to make this device the most convenient for subsequent users, a

mechanism was implemented to display actual information about the game performance relative to the current frame using a widget that contains two buttons.

The first button causes the **stat fps** and **stat unit** commands to be executed, while the second button causes the **stat none** command to be executed. More details:[23]

stat fps : The stat fps command is used to display frames per second (FPS) information on the screen, namely the number of frames per second and the average processing time per frame. These values are of fundamental value for any runtime application.

stat unit : The stat unit command in the Unreal Engine game engine displays information about the time spent on various stages of processing each frame of the game. This helps developers optimize game performance by identifying bottlenecks and areas where performance needs to be improved.

Typically, the stat unit output consists of three parts, representing the time (in milliseconds) spent on different stages of frame processing:

CPU time : Time spent on the processor (CPU) to process game logic, calculations, and other CPU-related operations.

GPU time : Time spent on the graphics processing unit (GPU) for rendering a scene, executing shaders, working with graphics resources, and other graphics-related operations.

Draw time : Time spent rendering a frame to the screen, including rendering graphics, texture processing, and other steps involved in displaying the image on the screen.

These three parameters allow developers to determine which parts of the game engine take the most time to process each frame, and help in identifying bottlenecks and possible places to optimize game performance.

stat none : disables the display of statistics on the screen. This is useful when you don't need to monitor game performance or when displaying statistics interferes with gameplay.[24]

All of the above commands can be manually entered by the user at the cmd prompt. This widget simulates automated injection for ease of use by a third-party user.

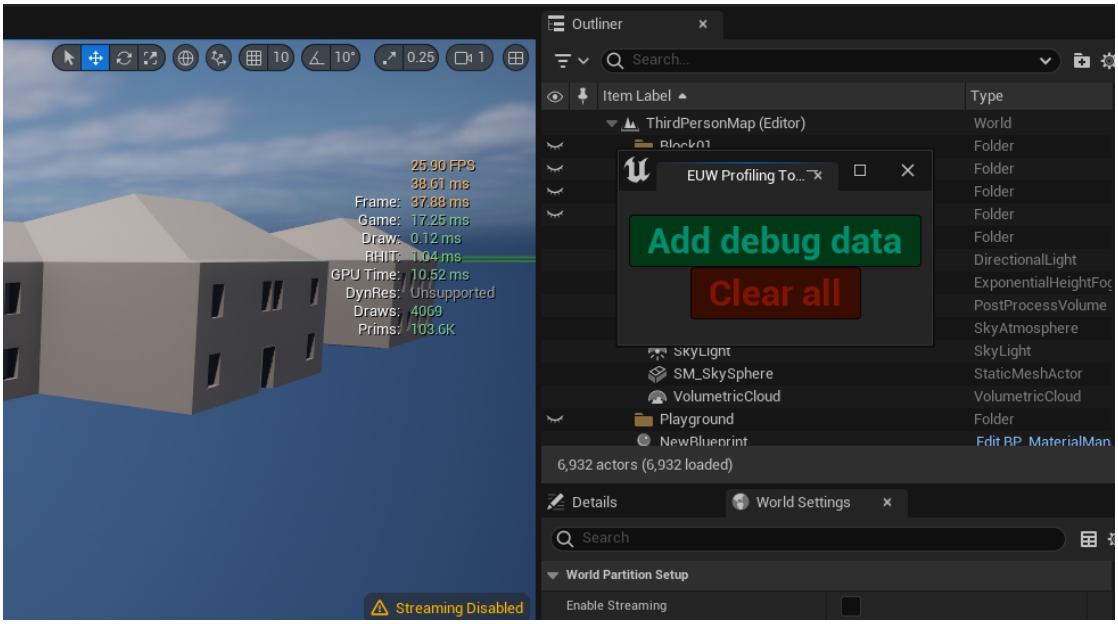

Figure 5.5 Widget and denug data.

The widget has two buttons, one of which displays stat unit and stat fps, and the second one calls stat none, which allows the user to easily add and remove all necessary statistics.

This widget is invoked by right-clicking on the widget's Blueprint, which opens a context menu where the top tab called Run Editor Utility Widget opens it in the editor.

5.3 Analysis

Geometry : The geometry was completely copied from the original scene as it originally appeared. Excessive polygonalization can be observed among the problems. Standard objects produced within the BIM format are rarely used in the playback of rantime applications and as a consequence are not optimized to an optimal state. It is also necessary to pay attention to the fact that each object on the scene is unique. None of the instantiation techniques have been applied. All this increases the total number of Drow calls by many times. However, even so such a scene as a school can be reproduced realtime on not the most modern hardware.

Materials and textures : All materials and textures were lost in the import process. This is easily adjusted using a script I created. However, if you redefine the meshes correctly (for example, all chairs use the same meshes), it will be enough to assign the material to one parent mesh and thus assign it to all of them.

Illumination : Because Unreal Engine uses its own model and lighting techniques, light sources are not imported. This can be easily fixed with a script. For example, in this work we assign emissive materials to all lamps via the same script5.2. It can also be done with a similar script by positioning point or directional light.

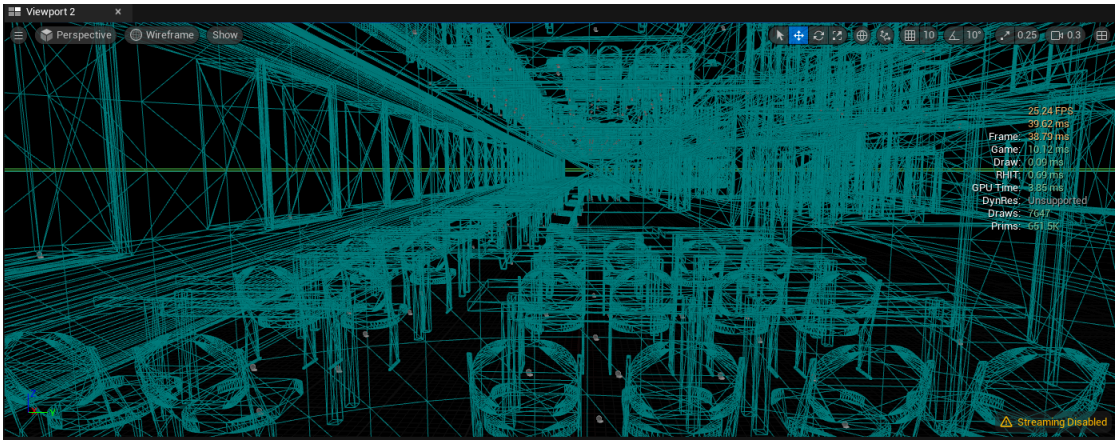

Figure 5.6 Recorded data example.

Performance : Scenes imported into Unreal Engine using Datasmith do not have the best performance, but even in this form they can work in realtime. On the example of test data provided to me in some angles, the number of Drow calls reached up to 8 thousand and primitives up to 700 thousand as may be seen on the image5.6, which are values that are above the accepted standards. In general practice there are different solutions for this problem, but in this case due to limited space it would be sufficient to use level streaming.

Chapter 6
Conclusion

The conclusion contains the result of the research, an attempt to implement one of the solutions, the results obtained and a conclusion based on the data obtained.

The project looked at several existing methods for importing BIM files into Unreal Engine. Based on the collected data, we implemented one of the possible solutions for importing BIM files into Unreal Engine via Datasmith. Various import aspects including geometry, materials, lighting, animation, collisions and metadata were analyzed and appropriate techniques and solutions were applied to improve the performance and visual quality of the project.

Differences between raw data and post-import data were analyzed, identifying key issues and potential improvements for future work. As part of the study of the import results obtained, problems and shortcomings were identified. Some of them were eliminated or optimized, solutions for individual problems were implemented.

Limitations in handling raw data, such as differences in formats or incomplete compatibility with some applications, also remain relevant and may require additional research and development.

Among the directions of further development of the project we can consider additional research and testing with different types of source data and formats to improve the import process and optimize performance, as well as the development and implementation of additional scripts and tools to automate and improve the process of data processing after import. Special attention should be paid to the Datasmith Runtime, which allows direct access to data during the import process and its modification.

The project allowed for a deeper understanding and exploration of the process of importing data into Unreal Engine using Datasmith, as well as identifying problems and potential improvements in this area. The results and experience gained will be useful for further work on projects related to visualization and virtual environments, as well as for the development and improvement of tools and methods for working with data in Unreal Engine.

We should pay attention to the fact that Epic Games are working very actively on Datasmith. For example, near the end of this paper there is Datasmith Runtime, which is able to solve many of the problems listed in this paper. It is highly recommended to familiarize yourself with the current Datasmith articles and documentation at the time of further actions within this paper.

Bibliography

[1] BuildingSmart official documentation. online, [cit. 2024-05-11]. Available from: https://standards.buildingsmart.org/IFC/

[2] Datasmith plugin main page. online, [cit. 2024-05-09]. Available from: https://www.unrealengine.com/en-US/datasmith

[3] Using Datasmith with IFC Files. online, [cit. 2024-01-27]. Available from: https://docs.unrealengine.com/4.27/en-US/WorkingWithContent/Importing/Datasmith/SoftwareInteropGuides/IFC/

[4] Institute, S. P. Unreal Engine basics. online, [cit. 2024-03-16]. Available from: http://web.spt42.ru/index.php/chto-takoe-unreal-engine

[5] Class structure. online, [cit. 2024-05-03]. Available from: https://dev.epicgames.com/community/learning/tutorials/7xWm/unreal-engine-basic-class-structure

[6] Unreal Engine basics. online, [cit. 2024-03-16]. Available from: https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/ProgrammingWithCPP/

[7] Unreal Engine AActor. online, [cit. 2024-03-16]. Available from: https://docs.unrealengine.com/5.3/en-US/actors-in-unreal-engine/

[8] Unreal Engine UActorComponent. online, [cit. 2024-03-16]. Available from: https://unrealcommunity.wiki/component-uxhizejm

[9] Third Person Template. online, [cit. 2024-04-25]. Available from: https://dev.epicgames.com/documentation/en-us/unreal-engine/third-person-template-in-unreal-engine

[10] Designing Visuals, Rendering, and Graphics documentation. online, [cit. 2024-05-11]. Available from: https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/Materials/

[11] Collision Overview documentation. online, [cit. 2024-05-12]. Available from: https://dev.epicgames.com/documentation/en-us/unreal-engine/collision-in-unreal-engine---overview?application_version=5.0

[12] Hardware and Software Specifications. online, [cit. 2024-03-16]. Available from: https://docs.unrealengine.com/5.0/en-US/hardware-and-software-specifications-for-unreal-engine/

[13] BIM files: the main BIM formats for design. online, [cit. 2023-12-11]. Available from: https://biblus.accasoftware.com/en/bim-files-the-main-bim-formats-for-design/

[14] About the IFC File Format. online, [cit. 2024-01-01]. Available from: https://help.autodesk.com/view/RVT/2024/ENU/?guid=GUID-0D546BEA-6F88-4D4E-BDC1-26274C4E98AC

[15] BIM and IFC – What are IFC models, and how do BIM and IFC relate? online, [cit. 2024-05-11]. Available from: https://plannerly.com/bim-and-ifc-models/

[16] Extensible Markup Language documentation (XML). online, [cit. 2024-05-09]. Available from: https://www.w3.org/XML/

[17] FileFormat documentation. online, [cit. 2024-05-11]. Available from: https://docs.fileformat.com/cad/ifc/

[18] Datasmith Overview. online, [cit. 2024-01-27]. Available from: https://docs.unrealengine.com/5.3/en-US/datasmith-plugins-overview/

[19] IFC to Unreal. online, [cit. 2024-01-27]. Available from: https://dasutt.github.io/stories/ifc-to-unreal/

[20] About the Datasmith Import Process. online, [cit. 2024-01-27]. Available from: https://docs.unrealengine.com/4.27/en-US/WorkingWithContent/Importing/Datasmith/Overview/ImportProcess/

[21] 3Ds Max. online, [cit. 2024-01-27]. Available from: https://media.contented.ru/glossary/3ds-max/

[22] Using Datasmith at Runtime. online, [cit. 2024-01-27]. Available from: https://docs.unrealengine.com/4.27/en-US/WorkingWithContent/Importing/Datasmith/Overview/UsingDatasmithAtRuntime/

[23] Advanced analysis. online, [cit. 2024-05-03]. Available from: https://dev.epicgames.com/community/learning/tutorials/dXl5/advanced-debugging-in-unreal-engine

[24] Stat Commands. online, [cit. 2024-05-03]. Available from: https://docs.unrealengine.com/4.27/en-US/TestingAndOptimization/PerformanceAndProfiling/StatCommands/

Chapter 7

Project folder

- `.uproject` ..... exe of UE project
- `thesis.zip` ..... Thesis data to recreate it in overleaf
- `thesis.pdf` ..... Thesis text in PDF format