MSc thesis in Geomatics

LuminaCity: a Real-Time Daylight Analysis Tool for Architectural and Urban Development Using Unreal Engine

Siebren Meines

2023

TU Delft

MSc thesis in Geomatics

LuminaCity: a Real-Time Daylight Analysis Tool for Architectural and Urban Development Using Unreal Engine

Siebren Meines

July 2023

A thesis submitted to the Delft University of Technology in partial fulfilment of the requirements for the degree of Master of Science in Geomatics

The work in this thesis was carried out in the:

Geo-Database Management Centre
Chair of GIS Technology
Section Design Informatics
Architectural Engineering and Technology Department
Faculty of Architecture & the Built Environment
Delft University of Technology

Supervisors: Dr. Azarakhsh Rafiee & Dr. Eleonora Brembilla
Co-reader: Ken Arroyo Ohori

Abstract

This thesis presents the development of a real-time daylight analysis tool for architectural and urban development using the Unreal Game Engine. The tool offers architects and urban planners a fast and precise way of analysing outdoor daylight conditions in their designs. The Unreal Game Engine provides real-time visualisation and analysis of daylight conditions, which makes it an effective tool for real-time decision-making during the design process. The study compares the light values and the process of extracting these values in Unreal with the validated light model Radiance used in Honeybee & Ladybug in the visual scripting program Grasshopper for Rhino. The thesis compares the values of Unreal & Honeybee/ladybug based on outdoor illuminance values, calculation time and ease of use. The comparison demonstrates the potential of Unreal as a valuable daylight analysis tool, with measurements showing a Mean Absolute Error of 9.78% between Honeybee and Unreal. In terms of computational time, the Unreal application requires only 0.6ms to execute and recalculate the daylight analysis, whereas the complete Honeybee script took an average of 93 minutes to calculate daylight values, and a new angle calculation took approximately 645 seconds. Furthermore, unlike Honeybee, which is sensitive to the complexity of urban or complex geometries and requires challenging adjustments, the Unreal application effortlessly accommodates complex geometry without the need for extensive modification.

The thesis concludes that the tool provides a robust and efficient method for analysing daylight conditions in architectural and urban design. The tool's ease of use and real-time visualisation capabilities make it an essential addition to the design workflow. Finally, the thesis presents the tool as a proof of concept for a geospatial urban development platform with built-in geospatial analysis. The research has demonstrated the potential of real-time simulation and analysis using the Unreal Game Engine as a powerful tool for architects and urban planners.

Acknowledgements

I would like to extend my heartfelt appreciation to all those who have contributed to this work, whether directly or indirectly. I would like to express deep gratitude to my supervisors Azarakhsh Rafiee & Eleonora Brembilla, whose guidance and support throughout the project were instrumental in shaping this research. Their expertise in daylight and game engines proved instrumental in shaping this research. However, their most significant contribution lied in their ability to help me prioritize and maintain focus on the main objectives throughout the entirety of this research. Thank you as well to the other members of my committee Bastiaan van Loenen & Ken Arroyo Ohori, who provided valuable feedback and insights that helped to refine my ideas and improve upon my methodologies.

I would also like to express my gratitude to Rui de Klerk, Paul de Ruiter & Hans Hoogenboom, who were of great help and incredibly helpful to spar with and see things from a different perspective.

Finally, thank you to my family and friends, who supported me through long hours spent writing code and analysing results - their encouragement kept me motivated and focused during challenging times. This thesis would not exist without the contributions from each of them.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

GPU	Graphics Processing Unit	32
CRS	Coordinate Reference System	34
UI	User Interface	43

1. Introduction

With increasing weather extremities due to climate change, and the densification and expansion of our urban environments, the need to consider environmental factors such as daylight are becoming increasingly important. Daylight plays a critical role in human health and well-being, but also significantly impacts the energy efficiency and sustainability of buildings as well as its impact on the visual appeal and liveability of a city.

To address these challenges, urban developers must carefully consider the trade-offs between incorporating daylight in a way that maximises its benefits while minimising potential drawbacks. As an answer to these challenges, there are a range of strategies that cities can use to effectively incorporate daylight into urban development. One key approach is the use of passive solar development, which involves orienting buildings and windows in such a way as to maximise the amount of daylight that enters the building (Morrissey et al., 2011). This can be achieved through the consideration of the orientation and placement of buildings, as well as the use of light-reflecting or absorbing materials, the use of strategic window placement, skylights, and other development elements that allow daylight to enter the building while minimising energy loss.

Honeybee, a commonly used tool based on the Radiance model, serves this purpose but comes with limitations. Its specific geometry requirements, limited handling of larger-scale areas, and exponential increase in simulation time for complex and large geometries limit its usability in real-world urban environments. Consequently, there is a pressing need for a more time-efficient and user-friendly approach to daylight analysis in urban design.

To tackle these challenges, this research explores the capabilities of the Unreal Engine as a platform for physically accurate daylight simulation. The Unreal Engine offers real-time rendering and interactive 3D visualisation, which solve the limitations of Honeybee. The primary focus of this thesis is to investigate the suitability of the Unreal Engine for scaling up physically accurate daylight simulation tools. By leveraging the advanced features of the Unreal Engine, such as its interactive platform, the ability to manipulate and add geometry, and the integration of Cesium Tiles, the aim is to develop an integrated solution that overcomes the limitations of existing tools and provides urban developers with an intuitive tool for analysing daylight in their designs. Furthermore, this research aims to contribute to the advancement of architectural and urban design practices by providing a more efficient and effective solution for incorporating daylight analysis into the design process.

1.1. Motivation

With the densification of urban areas, climate change and housing shortages, the architecture & urban development sector drastically needs to evolve. To tackle these challenges, it is becoming increasingly important to incorporate advanced technology and scientific methodologies into the design process. Real-time daylight analysis is a critical aspect of architectural and urban design, as it directly impacts the wellbeing and functionality of the built environment. Traditional daylight analysis tools are often cumbersome and difficult to use, making it challenging for designers to evaluate their designs accurately and efficiently. Coming from a Bachelor of Architecture and the Built Environment, I have experienced first-hand the frustration and limitations of existing tools. This is why I am motivated to develop an alternative tool using the Unreal Game engine. I believe that this tool has the potential to revolutionise the way architects and urban developers evaluate their designs by providing a user-friendly and accurate analysis & design platform. In addition, I hope this tool will help to bridge the gap between the architectural industry and scientific methodologies. My overarching goal is to establish a new approach in architectural and urban development by creating a comprehensive platform that integrates geospatial analysis with real-time visual feedback. With this thesis, I hope to prove the concept of a daylight analysis and design application using the Unreal Engine.

1.2. Research questions

This section states the research question and the sub-questions that will be used to answer that question. It will also elaborate on how these questions will be addressed.

Main Research Question:
To what extent is the Unreal Engine suitable to scale-up physically accurate daylight simulation tools?

The main question will be answered using the sub-questions. The research will result into a methodology to import 3D city models into an Unreal application. Additionally, a study will be conducted to test different methods of extracting light intensity from Unreal. Furthermore, a comparison and benchmark will be performed to evaluate the light intensity in Unreal compared to physically accurate and validated Honeybee. Lastly, an interactive light analysis application will be developed, integrating design tools for architectural and urban developers.

Sub questions:

1. How can 3D city models be effectively integrated into an Unreal application?

This question will involve exploring different methods and formats for integrating 3D city models into the Unreal Engine. Various integration techniques will be compared, and a methodology will be developed based on different scenarios. This methodology will then be implemented within the Unreal application.

2. How can the light values extracted from Unreal be compared and validated against industry-standard light models such as Radiance?

This will be addressed through a multi-step approach. First, light information will be extracted from Unreal using various methods. A Honeybee script will be developed to analyse and represent light values. Finally, a comparison will be made between the light values obtained from Unreal and the benchmark values from Honeybee, enabling the assessment of the realism of lighting in Unreal.

3. To what extent do the light values simulated with Unreal Engine accurately represent real-world lighting conditions?

To investigate the realism of lighting values in Unreal, an inverse square law experiment will be conducted. This experiment will analyse the decay of light intensity with distance, comparing the results with the expected theoretical values. Additionally, the obtained lighting values will be compared against a benchmark provided by Honeybee to further evaluate their accuracy.

4. In what ways can urban or architectural developers use an Unreal application to test and develop different design scenarios?

This question focuses on exploring the possibilities of the Unreal Engine for urban and architectural development. The research will investigate the incorporation of in-game functionality that enables developers to test and iterate different design scenarios within the application.

2. Theoretical background & Related work

The related work chapter provides an overview of the relevant literature and current state of research in the areas of 3D city models, raytracing, daylight analysis, and Unreal Engine. By addressing the benefits, challenges, and gaps in the existing literature, theis thesis aims to contribute to the advancement of knowledge and the development of improved daylight analysis tools within the context of architectural and urban development.

2.1 3D city models

3D city models represent the physical characteristics of a city in 3D, including buildings, roads, bridges, trees, and other infrastructure. They can be used for a variety of applications, including emergency response, traffic management, and urban development.

The main benefit of 3D city models is their ability to provide a more accurate and comprehensive representation of a city compared to traditional 2D maps. By adding a third dimension, these models can more accurately capture the shape, height, and geometry of buildings, as well as the location and orientation of streets and other infrastructure. This level of detail is particularly crucial for analyses dependent on building geometry, such as noise, solar/light, and volumetric analysis (Biljecki et al., 2016). 3D city models can be created at different levels of detail (LOD) based on the acquisition technique and intended usage. Figure 2.1 illustrates the varying LODs of 3D city models.

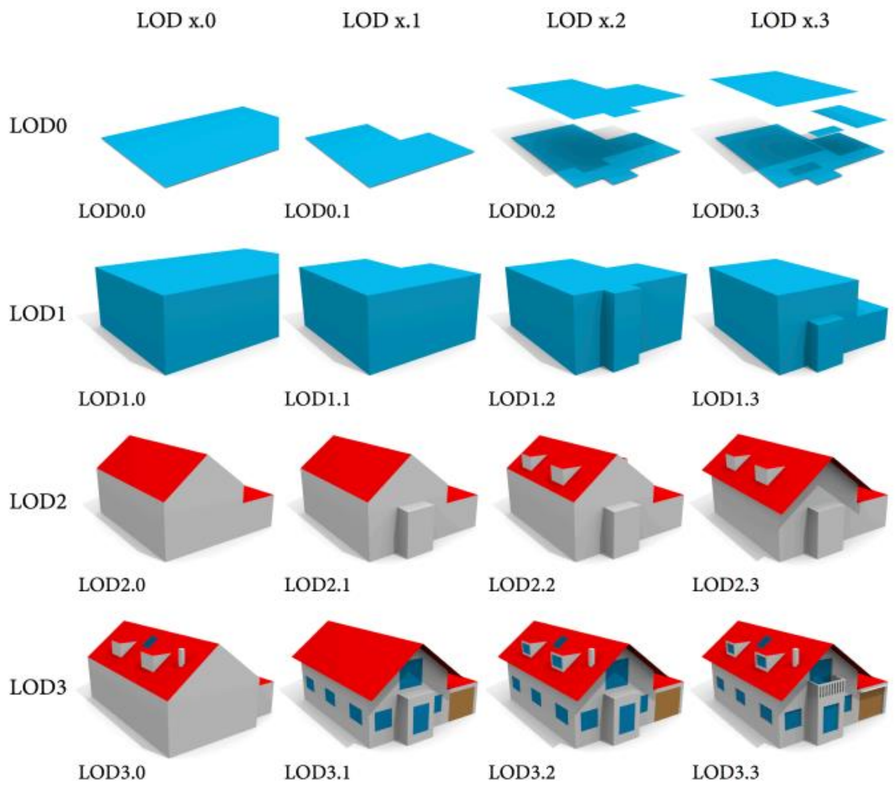


Figure 2.1 Different LOD of 3D city models (Biljecki et al., 2016).

Despite their benefits, there are also challenges associated with the creation and maintenance of 3D city models. These challenges include:

1. Standardisation. 3D models are often generated independently, using varying base data and reconstruction software. Furthermore, there is a lack of standardisation in terms of geometry and semantics (*State of the Art in 3D City Modelling*, 2022).
2. Data interoperability. Due to the lack of standardisation, it is also difficult to convert data models into different formats (*State of the Art in 3D City Modelling*, 2022).
3. Data quality. Many openly available 3D city models exhibit geometric and topological errors, such as missing surfaces, self-intersecting volumes, and duplicate vertices. (Biljecki et al., 2016).
4. Data governance: Ensuring proper governance and management of 3D city model data pose challenges.
5. Cost. Current approaches to generating and maintaining 3D city models are often costly and labour-intensive (Steinhage et al., 2010).
6. Urban analyses. There is a need for standardisation or toolkits for simulation, analyses, and management (Billen et al. 2015).

CityJSON, a standardised data exchange format based on JSON (JavaScript Object Notation) (CityJSON, n.d.), is the official standard of the Open Geospatial Consortium (*The Home of Location Technology Innovation and Collaboration* | *OGC*, n.d.). JSON is a widely used data interchange format that is both human-readable and machine-readable. Even though there is some standardization in 3D city models, they are being used in varying formats. For instance, in the Netherlands, the Dutch Kadaster has published the 3D Basisvoorziening (Blankert, 2021), a 3D dataset representing the Netherlands in CityJSON format. Additionally, TU Delft has created 3DBAG (Ledoux et al., 2019), providing detailed 3D representations of most buildings in the Netherlands in CityJSON format. However, despite the existence of unified national 3D city models like 3DBAG and 3D Basisvoorziening, individual municipalities maintain their own open 3D datasets. For instance, Rotterdam has a CityGML dataset, Den Haag has a CityGML dataset, and Amsterdam offers a Cesium 3D Tiles dataset (TU Delft, n.d.). Consequently, the importing of various 3D city models into Unreal requires addressing the diversity of formats and regional variations.

2.2 Raytracing

Raytracing is a technique used in computer graphics and simulation to create realistic and exact renderings of light distribution (Akenine-Möller et al., 2019). The technique is based on tracing the path of individual light rays as they interact with objects in a virtual scene, allowing for the calculation of light reflections, refractions, shadows, and other optical effects (see Figure 2.2).

The fundamental principle behind raytracing is that each point on an object can cast multiple rays into the scene, which can be blocked, reflected, transmitted, absorbed, or diffused when interacting with new surfaces (Glassner, 1989).

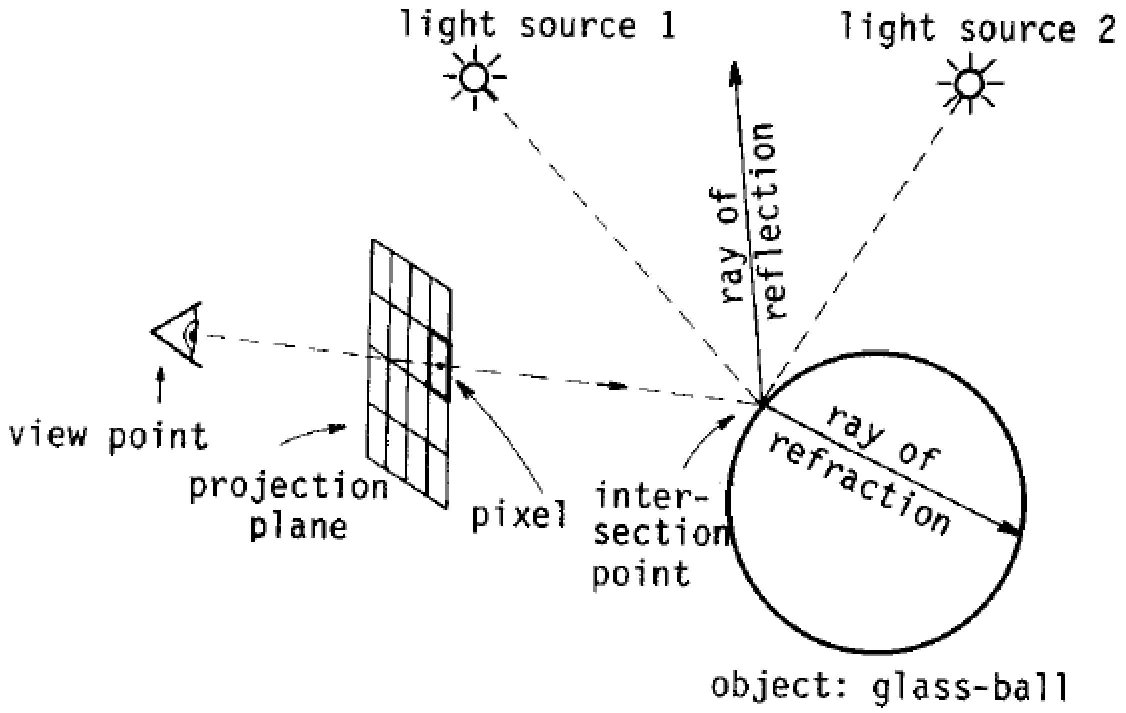


Figure 2.2 Principle of Raytracing (Schmitt et al., 1988).

Accurate rendering of reflective and transparent objects relies on the precise calculation of light reflections and refractions, according to well-established laws of physics. Snell's Law describes the propagation of light when it meets a surface, leading to refracted rays being either deflected away from or attracted toward the normal direction of the interface (Akenine-Möller, 2019). Rays experiencing total internal reflection never cross over to neighbouring mediums; however, they still influence energy flow within the scene. Raytracing excels in accurately simulating shadows and global illumination effects. By tracing secondary rays from the intersection points towards light sources, it is possible to determine if an object is in shadow or receiving direct illumination. Additionally, global illumination techniques, such as ambient occlusion, radiosity, and path tracing, allow for realistic indirect lighting and the simulation of light bouncing between surfaces. Another essential part of efficient raytracing is the implementation of acceleration structures, like bounding volume hierarchies (BVH), KD trees, or octree hierarchies, which enable faster intersection testing, resulting in reduced rendering times without compromising accuracy (Hapala & Havran, 2011).

Traditionally, raytracing has been computationally expensive, limiting its use to offline rendering. However, advancements in hardware and algorithms have made real-time raytracing possible (Singh & Narayanan, 2009). GPUs and specialized raytracing hardware have accelerated the rendering process, enabling interactive and immersive ray-traced experiences in video games, virtual reality, and architectural visualization.

2.3 Daylight analysis

Daylight analysis plays a crucial role in urban environments as it directly impacts the quality of life and well-being of individuals residing in these areas. While some researchers argue that optimal light simulation requires a physical laboratory setup, this approach is often expensive (Chen et al., 2019). Alternatively, computer models can be employed for light simulation, offering a cost-effective solution. Many light simulation models rely on stochastic methods. Chen et al. (2019) suggest that VR lighting environments provide an effective means to represent physical light environments, accurately capturing various lighting attributes such as diffuse/glaring, bright/dim, open/close, noisy/quiet, in coherence with physical environments.

2.3.1 Luminance and Illuminance

Luminance and illuminance values play a crucial role in understanding the spatial distribution and and visual perception of light in architectural and urban environments. Comprehending luminance and illuminance is essential for optimizing lighting conditions, ensuring appropriate levels of brightness, and meeting lighting standards in various architectural and urban contexts. These measurements contribute to creating visually appealing, functional, and comfortable environments for occupants.

Luminance refers to the amount of visible light emitted, reflected, or transmitted by a surface in a specific direction. Measured in candelas per square meter (cd/m^2), luminance represents the perceived brightness of a surface, as perceived by the human eye (Luminous Intensity & Photometry | auersignal.com, n.d.).

On the other hand, illuminance quantifies the amount of light falling on a surface and is typically expressed in lux (lx). It indicates the quantity of light reaching a specific area. Lux provides information about illuminance, serving as a measure of the brightness with which an area is illuminated. It measures how much luminous flux (lumen) of a light source arrives per unit area on a surface. The calculation of lux (see Figure 2.3.1) can be done in the following ways:

1. Lux [lx] = luminous flux [lm] / area [m2].
2. Lux [lx] = luminous intensity [cd] / radius or distance squared

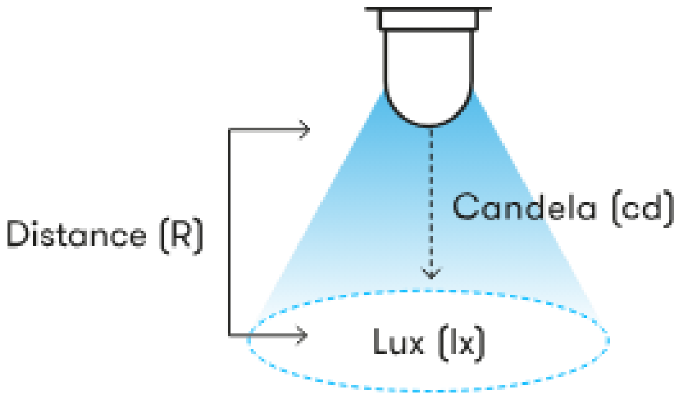


Figure 2.3.1 Formula to calculate lux (Luminous Intensity & Photometry | auersignal.com, n.d.)

For example, if a luminous flux of 100 lumens falls uniformly on an area of 1 m^2, the illuminance or would be 100 lux. The illuminance decreases as the distance from the light source increases. Lux values can be used to determine if certain areas are satisfactorily illuminated. To measure illuminance, a luxmeter, or light intensity meter, is used. It provides a reading of the illuminance in lux at the measurement point. A luxmeter consists of a photo sensor, typically containing photo diodes, which detect the light, and a display that shows the measured lux value. These concepts are visualized in Figure 2.3.2, which illustrates the relationship between luminous flux, luminous intensity, luminance, and illuminance.

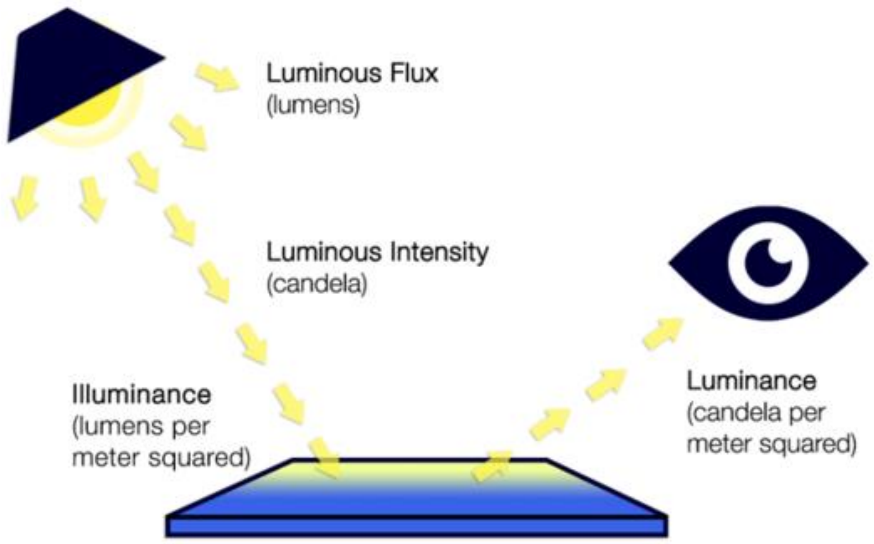


Figure 2.3.2 Luminance and Illuminance (AnneCorning & Systems, 2022).

2.3.2 Daylight Modelling and Simulation

Daylight simulation models usually exist of a model that represents a sky and a model that mimics the way light propagates. Sky models are mathematical representations of the sky (Antonanzas-Torres et al., 2019). Common sky models include the CIE Standard Sky, Perez Sky Model, and Climate-Based Sky Models (CBSM). These models consider factors like the sun's position, sky type, and atmospheric conditions to generate exact representations of the sky's luminance.
Sky models consists of multiple components. The most important two are (direct) sunlight and (indirect) skylight (Mardaljevic, 1999). Sunlight is the direct light from the sun that reaches an area. The position of the sun in the sky, influenced by factors like latitude, time of day, and time of year, affects the intensity and direction of sunlight. Skylight refers to the diffuse light from the sky that reaches an area through the atmosphere. It provides ambient illumination and can significantly influence the overall lighting conditions. Skylight is affected by factors such as cloud cover, atmospheric conditions, and the surrounding built environment.

In terms of light propagation, daylight models are mainly based on two main concepts: raytracing and Monte Carlo simulation. Raytracing, a computer simulation method based on geometric optics, models the path of light through a scene. It is widely utilised to visualise object appearances in virtual environments and analyse light distribution in spaces (Glassner, 1989). Raytracing finds extensive use in lighting design and urban development to predict illuminance levels and luminance distributions in both outdoor and indoor spaces. Notably, its ability to accurately model light interactions with surfaces and materials allows designers to anticipate space appearance under different lighting conditions.

Monte Carlo simulation, on the other hand, is a computer simulation method that utilises random sampling to calculate the probability of different outcomes (Mooney, 1997). In the context of daylight analysis, Monte Carlo simulations analyse light distribution in urban environments by simulating the path of individual light photons as they interact with various surfaces. This method accurately predicts light distribution in complex environments, such as urban canyons, incorporating reflections, refractions, and scatterings (Wang, 2014).

2.3.3 Radiance

Radiance, a widely-used and powerful tool, for daylight simulations, combines deterministic raytracing and Monte Carlo simulation to simulate lighting and daylighting in architectural and urban design (Radiance — Radsite, n.d.). With its software tools, users can generate accurate and detailed simulations of lighting conditions, including direct sunlight, skylight, and reflections.

Radiance uses the Perez sky model as a fundamental component of its daylight simulation capabilities. The Perez sky model, also known as the Perez all-weather model, is a mathematical representation of the sky's luminance distribution based on measured sky radiance data (Perez, 1993). It accurately describes the sky conditions throughout the year under various weather conditions. The Perez model considers parameters such as solar altitude, solar azimuth, turbidity, and other atmospheric properties to calculate the sky's luminance distribution accurately. By simulating the behaviour of light rays interacting with various surfaces, Radiance can calculate illuminance values, daylight factors, and other metrics that help assess the quality and performance of natural lighting within architectural spaces.

2.3.4 Honeybee

In the context of architectural and urban design, Radiance is often used within Grasshopper through the integration of Honeybee plugins. Grasshopper provides a graphical no-code interface for creating and manipulation of 3D models. Honeybee acts as a bridge between the Grasshopper environment and Radiance, allowing users to create and analyse complex daylighting scenarios within their architectural designs. Honeybee offers tools for simulating lighting and daylighting conditions using Radiance. Together, these plugins enable users to create accurate and detailed simulations of lighting conditions and evaluate the performance of their design.

Honeybee offers various components for creating geometry, defining materials, specifying sky conditions, and configuring analysis parameters. Users can create detailed 3D models of buildings, assign materials with optical properties, and set up sky conditions using the Perez sky model or other sky models available in Radiance.

Honeybee communicates with Radiance to perform the actual raytracing calculations. The interaction between Honeybee, Radiance and other software is visualized in Figure 2.3.4. Radiance traces light rays through the virtual environment, considering interactions with surfaces, reflections, and refractions. This allows Honeybee to compute accurate metrics such as illuminance, annual sunlight exposure, and other daylight performance indicators. Honeybee provides visualizations and analysis outputs based on the simulation results. These outputs can include color-coded maps of illuminance levels, daylight factor distributions, annual daylight metrics, and more.

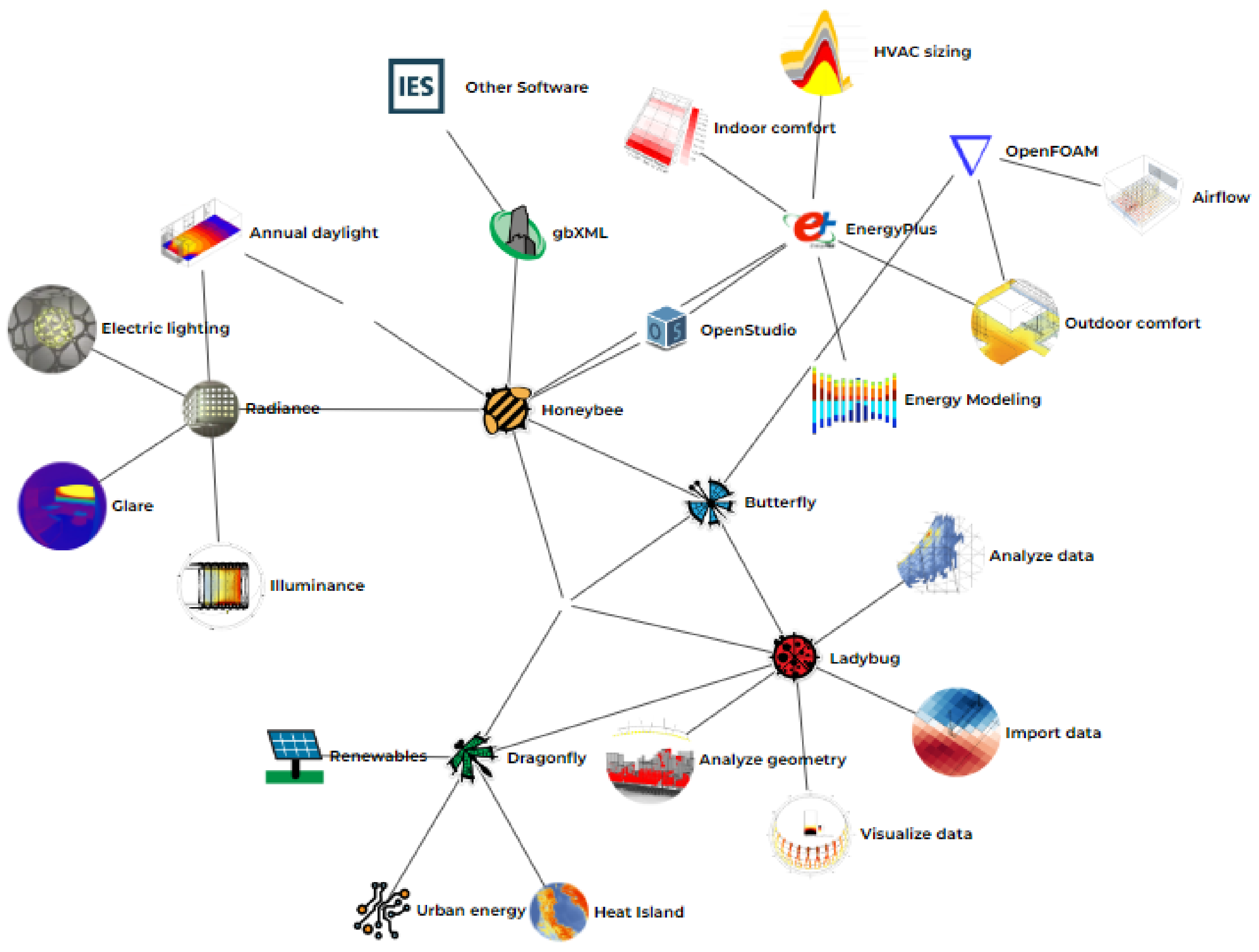


Figure 2.3.4 Connection between Honeybee, Radiance, and other related software (Ladybug Tools | Home Page, n.d.)

2.4 Unreal Engine

Game engines, like Unreal, are software frameworks designed to develop interactive 3D environments and gameplay mechanics, primarily for the creation of video games. However, their potential extends beyond gaming and holds potential for geospatial applications, such as urban development (Keil et al., 2021). Game engines provide a wide range of tools and features that enable designers to build immersive 3D environments and characters quickly. Leveraging geospatial data in game engines allows for the rapid creation of 3D environments and prototypes, making Unreal useful for architectural and urban development.

Examples of the implementation of game engines for geospatial related topics include:

1. The Generation of 3D terrains or DEMs (Mat et al., 2014).
2. The recreation of historical towns in VR (Kersten et al. 2018).
3. Solar potential assessment (Buyuksaliha et al., 2017).
4. Quick prototyping of spatial layouts and building design (Edler et al., 2020).

5. Safety training in dangerous environments such as underground hazard safety training (Liang et al., 2019).
6. Application of physical models for environmental simulation (Edler et al., 2019).

Among the game engines is the open-source Unreal Engine. Epic Games developed the Unreal Engine in the late 1990s as the backbone of the video game Unreal. Unreal is widely adopted in the gaming industry and has also gained traction in creating virtual reality and augmented reality experiences. Initially developed for video games, the Unreal Engine has evolved to encompass real-time visualisation and simulation applications. The Unreal Engine is known for its powerful real-time rendering capabilities, making it an ideal tool for creating high-quality 3D graphics and animations. It offers a range of features, including blueprint visual scripting, C++ coding, real-time rendering, animation, and physics. In recent years, the Unreal Engine has gained popularity in the field of architecture and urban development. Its real-time rendering capabilities make it an ideal tool for creating immersive and interactive visualisations of buildings and urban areas.

2.4.1 Key features and techniques.

The Unreal Engine is composed of three main parts (Lee, 2016), each with its own relationship and components, as depicted in Figure 2.4.1. These parts include the Sound Engine, responsible for handling music and sound in the game, the Physics Engine, which performs calculations for lifelike physical interactions, and the Graphics Engine, which handles rendering. The integration of these different engines within Unreal enables seamless communication between them.

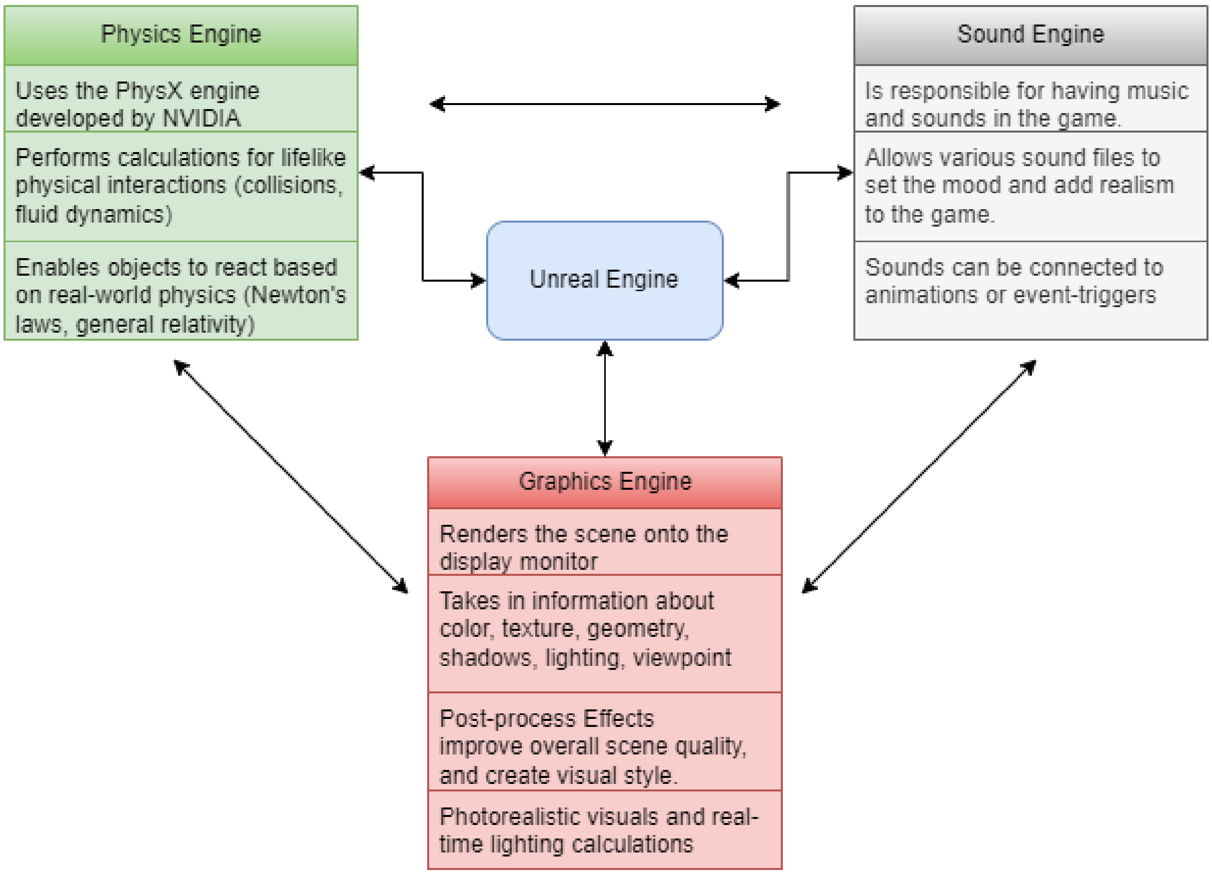

Physics Engine
Uses the PhysX engine developed by NVIDIA
Performs calculations for lifelike physical interactions (collisions, fluid dynamics)
Enables objects to react based on real-world physics (Newton's laws, general relativity)
Sound Engine
Is responsible for having music and sounds in the game.
Allows various sound files to set the mood and add realism to the game.
Sounds can be connected to animations or event-triggers
Unreal Engine
Graphics Engine
Renders the scene onto the display monitor
Takes in information about color, texture, geometry, shadows, lighting, viewpoint
Post-process Effects improve overall scene quality, and create visual style.
Photorealistic visuals and real-time lighting calculations

Figure 2.4.1 Different components of Unreal Engine

The Unreal Engine's exceptional speed and efficiency are achieved through the implementation of various technical aspects. Several key features and techniques contribute to its impressive performance:

1. **Multithreading**
   Unreal Engine takes advantage of multi-core Central Processing Units (CPUs) by using multithreading techniques, to improve performance and responsiveness (*Threaded Rendering*, n.d.). This technique distributes tasks across multiple threads, allowing for parallel processing of different aspects of the game engine.
2. **GPU Acceleration**
   Unreal Engine uses GPU acceleration to offload computationally intensive tasks from the CPU (GPUOpen by AMD, 2022). The GPU is designed for high-performance graphics rendering. By applying this technique, Unreal Engine can realize real-time rendering and complex visual effects.
3. **Culling and Occlusion Techniques**
   Unreal employs culling and occlusion techniques to avoid rendering objects that are not visible or blocked from view (GPUOpen by AMD, 2022). By not rendering non-visible objects, Unreal reduces unnecessary computations and improves performance.
4. **Precompiled Shaders**
   Unreal precompiles shaders during the development process, which lead to optimized rendering at runtime (*Shader Development*, n.d.). By precompiling shaders, it eliminates the need to compile shaders at runtime, improving startup times and improving render-related performance.

2.4.2 Raytracing in Unreal

Unreal uses raytracing to model the behaviour of light. It simulates the path of individual light rays as they interact with objects and surfaces. Unreal Engine uses real-time raytracing, which combines raytracing with rasterization techniques (*What Is Real-time Ray Tracing?*, n.d.). This hybrid approach allows for real-time light interactions. Unlike traditional rasterization methods, which approximate lighting effects using simplified algorithms, raytracing in Unreal Engine calculates the path of light rays as they bounce, reflect, and refract through a scene. This leads to more realistic physical lighting, with realistic shadows, reflections, and global illumination (see Figure 2.4.2).

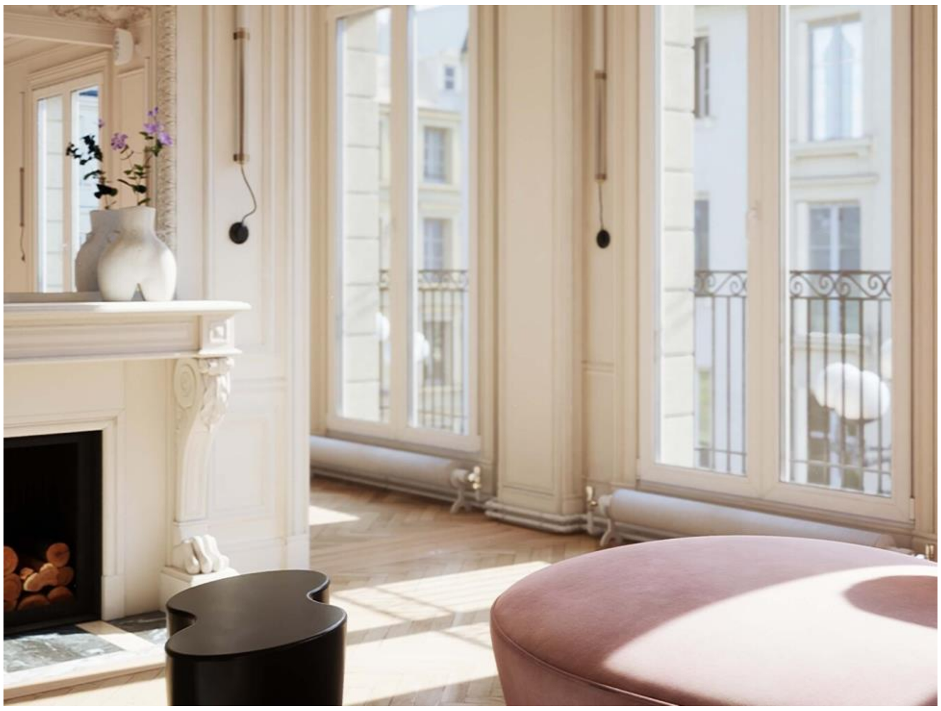


Figure 2.4.2 Realistic physical lighting in Unreal (*What Is Real-time Ray Tracing?*, n.d.)

One of the advantages of raytracing in Unreal Engine is its speed and efficiency. Unreal Engine uses a variety of optimization techniques, described in 2.4.1, to achieve real-time or near-real-time rendering performance. This means that complex scenes with dynamic lighting can be rendered smoothly and interactivity.
The speed and efficiency of real-time raytracing in Unreal Engine make it an appealing possibility to explore as an alternative to traditional daylight analysis tools like Honeybee. Honeybee, which uses Radiance for accurate daylight simulations, can be computationally intensive and time-consuming, especially in dealing with complex geometry.

2.4.3 Development vs. Runtime

Unreal Engine operates in two modes: development and runtime. The development mode facilitates real-time editing, debugging, and experimentation, allowing developers to iterate rapidly on their ideas. The runtime mode focuses on delivering a smooth, optimized experience to end-users without access to development tools or editing capabilities.
Development mode is primarily used during the process of creating and editing a game or application in Unreal Engine. In development mode, developers have access to various tools, features, and debugging capabilities to use in the development process. It allows for real-time editing of the game's content, including levels, assets, materials, and blueprints. Developers can make changes and instantly see the results without needing to restart the game.

Runtime mode refers to the execution of the game or application that has been built and packaged for distribution or testing. Users experience application without any access to development tools or editing capabilities. In runtime mode, the game or application runs independently of the Unreal Editor. It operates as a standalone program. Runtime mode may have different configuration settings compared to development mode, such as different graphical quality settings or performance optimizations specific to the target platform.

2.4.4 Blueprints vs. C++

Unreal Engine has two main methods for development: Blueprints and C++ coding. Blueprints in Unreal Engine is a visual scripting system that allows developers, to create gameplay logic

and functionality using a no-code, node-based interface. It offers a more accessible and intuitive approach. On the other hand, C++ coding provides greater flexibility and control to create complex functionality.
With Blueprints, developers can design and define object behaviour, character interactions, and environmental elements by connecting nodes representing different actions, events, variables, and conditions. It offers an accessible and intuitive way to quickly prototype gameplay mechanics and ideas without traditional coding. The visual nature of Blueprints facilitates collaboration between designers, artists, and programmers, as it allows for better understanding and visualization of logic flow. Blueprints support various functionalities, such as character movement, animation, AI behaviour, physics simulation, and user interfaces. Unreal Engine provides a library of pre-built Blueprint nodes and functions, and developers can also create custom Blueprint nodes using C++ to extend functionality.

Unreal Engine is built on top of the C++ programming language, which allows for low-level access, performance optimization, and complex systems development.
By writing C++ code, developers can create and customize game systems, implement advanced algorithms, optimize performance-critical sections, and integrate with external libraries and systems. Unreal Engine has many build-in classes, functions, and frameworks that developers amongst others can use to create game mechanics, implement game rules, create custom shaders. C++ coding is often preferred when working on large-scale projects, performance-critical features, or when extensive customization and control are required.

Unreal Engine allows developers to seamlessly combine Blueprints and C++ code in the same project (*C++ And Blueprints*, n.d.). This enables a hybrid approach where complex systems and critical components can be written in C++, while other aspects can be implemented using Blueprints.

2.5 Cesium ion & OGC 3D Tiles

To expand the geospatial capabilities of Unreal, Epic Games started a collaboration with Cesium (Cozzi, 2021b). Cesium ion was chosen for creating and hosting 3D geospatial data, publishing tilesets (see Figure 2.5), integration with other geospatial data sources, and performing advanced geospatial analysis. Cesium ion offers a range of geospatial analysis tools, including terrain analysis, 3D measurements, and visibility analysis (Cozzi, 2021b).

A key advantage of Cesium ion are Cesium 3D Tiles. Cesium 3D Tiles are the Open Geospatial Consortium (OGC) community standard for tiling massive and heterogeneous 3D content (Getz, 2021). Cesium OGC 3D tiling revolutionizes the visualization and distribution of complex geospatial data. Cesium not only enables the viewing of 3D tiles but also facilitates their creation, web-based hosting, and REST API-based serving through Cesium ion (Getz, 2021). Notably, Cesium has collaborated with Safe FME to integrate Cesium plugins into FME, providing the capability to generate batched tile models (Getz, 2021). The CRS used for the 3D tiles in Cesium aligns with the EPSG:4978 ellipsoid, which is equivalent to the WGS84 ellipsoid commonly used for representing the Earth (Getz, 2021).

Cesium ion tilesets empower the creation and hosting of 3D geospatial data, offering support for various formats such as OBJ, CityGML, photogrammetry, and LiDAR. These tilesets provide a scalable solution for visualizing large and complex datasets, with efficient streaming and rendering in real-time 3D environments. By integrating Cesium ion tilesets with Unreal through Cesium for Unreal, developers gain the ability to incorporate tiled/streamed 3D geospatial data seamlessly into their Unreal projects (Getz, 2021).

To showcase the geospatial capabilities of Unreal and Cesium, a collaborative effort resulted in the development of Project Anywhere. This open-source initiative harnesses the power of Cesium ion for Unreal to create interactive, real-time 3D visualizations of geospatial data. Project Anywhere demonstrates the potential for leveraging Cesium ion's capabilities in creating captivating and informative geospatial experiences (Getz, 2021).

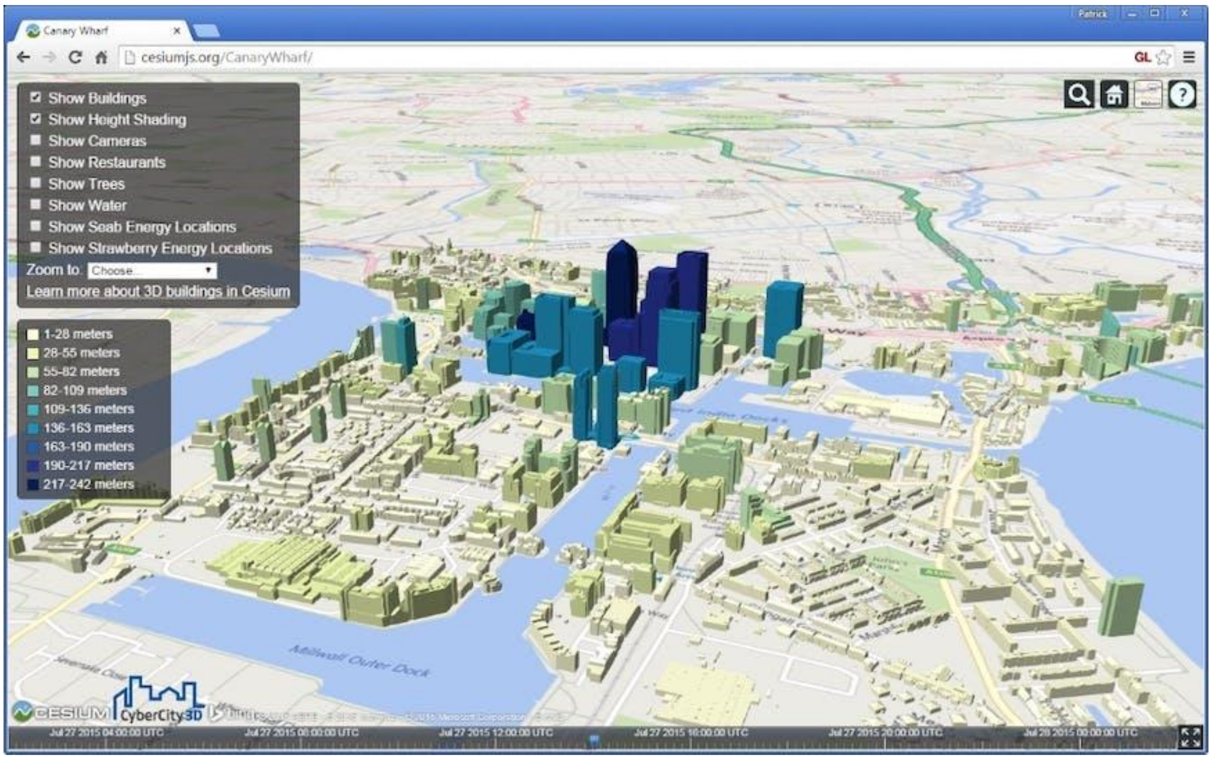


Figure 2.5 Height-dependent building coloured 3D Tile using Cesium (Cozzi, 2021a).

2.6 Daylight Analysis in Unreal

Even though the Unreal Engine is known for its qualitative looking light distribution, quantitative daylight analysis within the Unreal Engine, is not something that is widely used in the industry. This section aims to discuss and summarize some of the research conducted on daylight analysis in Unreal.

1. Visualizing user perception of daylighting: a comparison between VR and reality (Hegazy et al., 2020).

This article compares the perception of visual comfort in virtual reality with real-life scenarios. Participants in the study experienced both real rooms and reconstructed virtual spaces using Unreal Engine and Oculus Rift HMD. During the study participants were questioned about acceptability ratings and preferences. The results reveal significant differences in perception of real-life compared to reconstructed virtual spaces. Participants perceived greater warmth and rich colours under real sunlit settings compared to artificial sources or computer-generated versions. However, Unreal-rendered scenes received better scores than purely synthetic ones in terms of glare control and overall appearance quality. These results imply certain limitations exist in the perception of daylight simulation in Unreal, while also highlighting possibilities for optimizing render techniques in Unreal to reproduce more realistic daylighting characteristics.

2. State-of-the-art review of solar design tools and methods for assessing daylighting and solar potential for building-integrated photovoltaics (Jakica, 2018).

This paper mentions Unreal Engine as a tool that is increasingly being explored as a solar design tool for daylight analysis and design. It states that the advantage of using Unreal Engine for daylight analysis and design is its ability to generate highly detailed, interactive 3D models of buildings that accurately represent how sunlight will interact in architectural environments. These models can then be used to analyze solar performance and optimize PV panel placement. Another benefit of using Unreal Engine for daylight design is its compatibility with other modelling software commonly used in the industry, such as Revit or Rhino. This enables seamless data exchange and streamlined workflows. The paper mentions that as Unreal Engine's focus is on gaming rather than scientific physical analysis, its solar simulation capabilities may need additional validation testing before they can be widely adopted. Nonetheless, the authors note that Unreal Engine has enormous potential as a daylight design tool given its ease of use, low entry barrier, and strong graphical capabilities.

3. Validating Game Engines as a Quantitative Daylighting Simulation Tool (Hegazy et al., 2021).

The study aimed to evaluate the accuracy of Unreal Engine for indoor quantitative daylighting simulations. Traditionally, specialized tools like Radiance have been used for these types of simulations, but recent advancements in technology have led researchers to explore whether game engines could be a viable alternative.

The study includes two case studies with different complexity levels and spatiotemporal settings. The first case study focuses on a simplified shoebox model to evaluate the accuracy of game engine rendering in a basic scenario. The second case study involves a more complicated office model with various functions and a large-scale space to assess the potential of game engines for simulating daylighting in realistic environments.
In both case studies, illuminance levels at selected points were measured in Unreal Engine and compared to those calculated using Radiance and an illuminance meter for real-life measurements. The traditional rendering technique in the game engine showed a high variance in illuminance levels compared to the references.
Contrary, real-time ray tracing demonstrated better accuracy. In the simplified model, real-time ray tracing showed the lowest average error compared to the validated simulation results. In the complicated model, the average error of real-time ray tracing was close to that of the validated simulation, compared to the actual illuminance measurements.

The study highlights the benefits of using real-time ray tracing in game engines for offering an immersive and interactive experience of virtual daylit spaces without sacrificing the quantitative accuracy of the simulated luminous environments. It suggests that game engines have the potential to be a valuable tool for daylighting simulation and should be further validated and adopted in daylighting research. Although this study focusses on the illuminance values in indoor environments, the conclusions suggest that Unreal can realistically propagate daylight, which is a good basis for the further research.

3. Methodology

The methodology chapter provides a detailed description of the methodology used to develop the real-time daylight analysis tool for architectural and urban development using the Unreal Game engine. The purpose of this chapter is to provide a clear and concise description of the methods employed in the development of the tool. Figure 3 shows a flowchart of that methodology that will lead to the interactive daylight analysis application in Unreal.

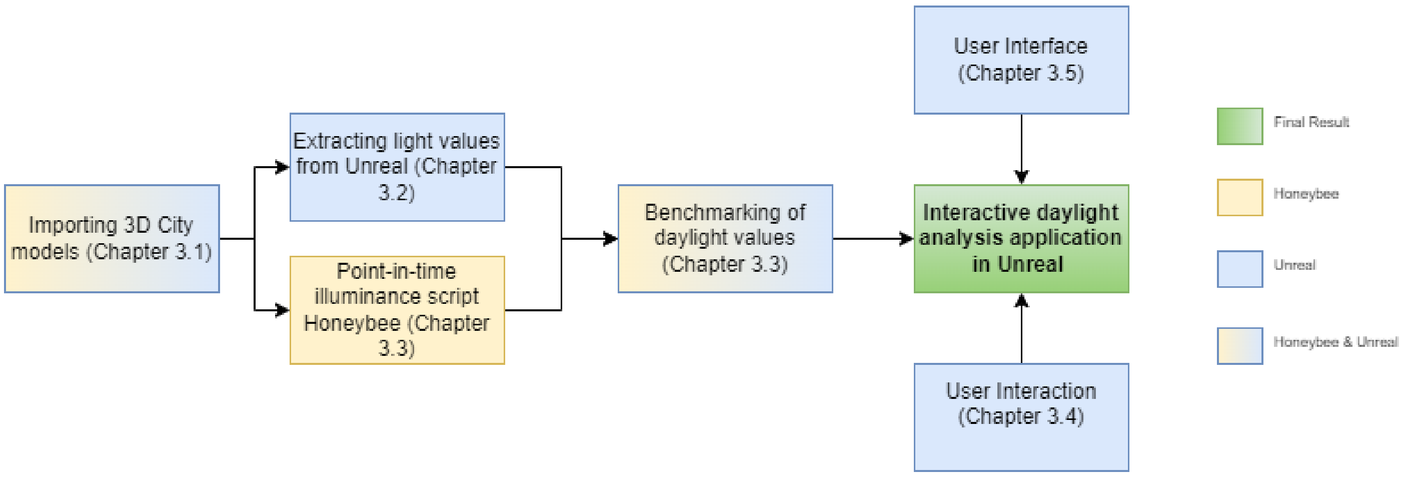

Importing 3D City models (Chapter 3.1)
Extracting light values from Unreal (Chapter 3.2)
Point-in-time illuminance script Honeybee (Chapter 3.3)
Benchmarking of daylight values (Chapter 3.3)
User Interface (Chapter 3.5)
Interactive daylight analysis application in Unreal
User Interaction (Chapter 3.4)
Final Result
Honeybee
Unreal
Honeybee & Unreal

Figure 3 Flowchart of methodology

3.1 Loading 3D city models in Unreal

3D city models come in various formats such as CityJSON and CityGML, which can vary depending on the country, province, or municipality. For instance, in the Netherlands, the Dutch Kadaster has published the 3D Basisvoorziening (Blankert, 2021), a 3D dataset representing the Netherlands in CityJSON format. Additionally, TU Delft has created 3DBAG (Ledoux et al., 2019), providing detailed 3D representations of most buildings in the Netherlands in CityJSON format. However, despite the existence of unified national 3D city models like 3DBAG and 3D Basisvoorziening, individual municipalities maintain their own open 3D datasets. For instance, Rotterdam has a CityGML dataset, Den Haag has a CityGML dataset, and Amsterdam offers a Cesium 3D Tiles dataset (TU Delft, n.d.). Consequently, the importing of various 3D city models into Unreal requires addressing the diversity of formats and regional variations.

Importing a City Model directly into Unreal presents several challenges. Firstly, Unreal does not support formats such as CityJSON and CityGML. One possible approach is to convert the model into an OBJ format and then import it into Unreal. However, this method leads to performance issues and prolonged import times due to the large number of buildings in a 3D City Model file. Additionally, maintaining consistent georeferencing becomes more difficult with this approach. While Unreal offers georeferencing capabilities, they are project-based rather than file or object-based. Since the OBJ format uses a local CRS with offsets (Wikipedia contributors, 2023), imported OBJs are not automatically placed in the correct location but would require manual adjustment. As a result, ensuring the integrity of the CRS and the accurate positioning of buildings becomes exponentially challenging when dealing with different files and CRS. Therefore, both direct importing and importing in the form of OBJ prove impractical, requiring alternative approaches.

To address the variations in 3D model formats, such as CityJSON and CityGML, and their incompatibility with direct placement into Unreal, different methodologies and scenarios will be tested to assess the usability of different formats of 3D City models in Unreal. For this purpose, Cesium ion, a plugin for Unreal, was used. Cesium ion brings tiling, georeferencing, standardization and other geospatial capabilities into the Unreal Game Engine. Cesium 3D Tiles are Open Geospatial Consortium (OGC) community standard for tiling of massive heterogenous 3D content (Getz, 2021).
While CityGML is an accepted format by Cesium ion, CityJSON is not supported. Besides CityGML, and CityJSON, the Google Maps API & Cesium OSM tileset will also be tested.
The Cesium OSM tile from Cesium ion enables the seamless import of OSM 3D tilesets into Unreal Engine, providing access to immersive and realistic 3D environments. These tilesets consist of 3D models and terrain data generated using OSM, bringing access to realistic 3D environments in Unreal. The tilesets generated by Cesium ion are optimized for performance and streaming, enabling efficient visualization of large-scale 3D city model datasets.
On May 10th, 2023, Cesium made an exciting announcement regarding their partnership with Google Maps (Cozzi, 2023). As part of this collaboration, Google launched an exciting experimental release of Photorealistic 3D Tiles via their Map Tiles API. These 3D tiles are created based on Cesium's OGC standard. With the integration of the Google Maps API into Unreal Engine, developers can seamlessly incorporate these photorealistic tiles directly into their projects. This integration allows developers to incorporate the detailed and lifelike representations of real-world environments provided by Google Maps into their Unreal Engine projects.

To test the usability of loading 3D city models in Unreal, the following scenarios will be tested and compared to each other:

1) Convert CityJSON to Cesium tile using FME
2) Load CityGML directly into Cesium ion.
2) Convert CityJSON to CityGML using FME
3) Convert CityJSON to CityGML using citygml-tools
4) Cesium OSM Tileset
5) Google Maps API.

3.2 Extracting light values from Unreal

Because Unreal was originally designed for game development, lighting and its values are primarily focused on the visual aspects. However, Unreal does utilise physical units in its lighting calculations.

Both Radiance & Unreal calculate illuminance values based on raytracing & monte Carlo simulation. However, there is a difference in the way Unreal & Radiance calculate Luminance. Radiance uses the relative luminance * 179. Relative luminance in Radiance is defined as (0.265*Red) + (0.670 * Green) + (0.0065 * Blue). Radiance multiplies by 179, because of the white light efficacy factor of 179 lm/W (gendaylit.pdf — Radsite, n.d.). Whereas Unreal uses the relative luminance values normalised to a range between 0 & 1 multiplied by the intensity of the light source (Physical Lighting Units, n.d.). Relative luminance in Unreal is defined as (0.299 *Red) + (0.587 * Green) + (0.114 * Blue). Unreal does not offer any explanation as to why this formula is chosen. it is important to note that the choice of formulas for calculating luminance in Radiance and Unreal can have implications for the accuracy and visual representation of light in their respective simulations. The multiplication by 179 in Radiance considers the specific white light efficacy factor, which may contribute to a more realistic representation of luminance values. In contrast, Unreal's normalization and multiplication by the intensity of the light source may offer a different perspective on luminance, potentially affecting the perceived brightness and visual appearance of scenes. These differences will be considered when implementing the functionality to calculate luminance in Unreal, considering both approaches.

This thesis will focus on multiple ways to extract light values (Luminance & Illuminance) from Unreal:

1. **Accessing light information directly from build in functionality of Engine**

Unreal provides built-in functionality to calculate the amount of light in a specific area of the scene. This research will investigate accessing this functionality directly, incorporating it into a game mode at runtime, and experimenting with adjusting or modifying the light information functionality.

2. **Calculating Luminance and Illuminance values using a custom C++ actor.**

Another method that will be explored for calculating luminance and illuminance values is through the development of a custom C++ actor. The actor will capture the screen space coordinates and RGB values of the pixels. Once the RGB values are obtained, they will be converted into luminance & illuminance.

3. **HDR eye adaptation.**

The HDR eye adaptation tool in Unreal is a built-in functionality that enables the game engine to measure illuminance values in real-time. This tool mimics the way that the human eye adapts to changing light conditions. It works by constantly measuring the brightness illuminance of the scene and adjusting the exposure settings of the camera accordingly. In this research the tool will be used to measure luminance & illuminance values.

4. **False Colour Post-Processing Material.**

Light perception is a visual phenomenon. Radiance uses a False-colour rendering technique to visualise the amount of light in different colours. This allows users to quickly identify and understand the distribution of light in a certain model, without looking at numerical data. In Unreal, this false-colour rendering will be implemented as a post-processing material. Post-processing materials in Unreal are special types of materials that can modify the final image output of a scene. (Post Process Materials, n.d.). They can use information from lights in the scene to adjust the colour and brightness of materials and apply effects such as bloom, lens flares, and colour grading. This provides greater control over the final appearance of a scene. Additionally, the feasibility of accessing the numerical values of the light and assigning colours based on the numerical values of light. E.g., colouring every object with 5000 lux orange.

3.3 Comparison of different light analysis methods vs. Benchmark

To validate the accuracy of daylight values in Unreal and its ability to realistically propagate daylight, a twofold approach is implemented. Firstly, an inverse square law experiment is conducted to investigate the behaviour of light values in Unreal and their adherence to the theoretical values predicted by the inverse square law (see Figure 3.3). The outcome of this experiment will be used as the basis of the further benchmarking. It is important to note that the inverse square law primarily accounts for the spread of light and does not consider other factors such as diffraction or refraction caused by surrounding buildings. The objective of this experiment is to gain an initial understanding of Unreal's ability to accurately simulate light propagation.

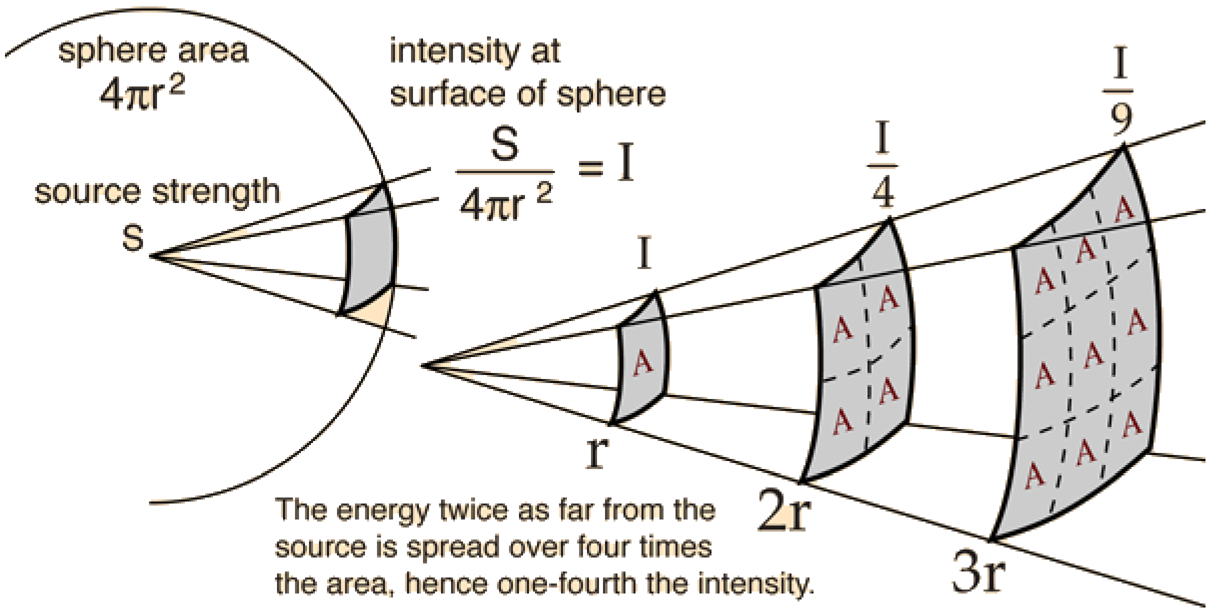

sphere area
$4\pi r^2$
intensity at surface of sphere
$\frac{S}{4\pi r^2} = I$
source strength
S
I
$\frac{I}{4}$
$\frac{I}{9}$
A
r
2r
3r
The energy twice as far from the source is spread over four times the area, hence one-fourth the intensity.

Figure 3.3 Inverse square law (Inverse Square Law, n.d.).

While the inverse square law experiment provides an initial insight into the spread of light, it is acknowledged that additional factors influence the daylight in urban environments. In the further benchmarking in an urban environment, other factors, including measurement errors, sun position, skylight intensity, and sun intensity, could influence the accuracy of the daylight measurements. By conducting the inverse square law experiment a baseline is established for further benchmarking and strive to understand the extent to which Unreal accounts for different behaviours of light.

To further validate the realisticity of daylight propagation in urban environments, the values from Unreal are compared against the established light analysis method Radiance. A 3D city model tileset will be imported, and light values will be extracted from Unreal and compared to those extracted using Honeybee. A comparison between Honeybee and Unreal will be made in terms of their ability to handle complex 3D city models, the computational time required for calculations, and the accuracy of the results.

To calculate daylight values in Honeybee, a Grasshopper script will be developed to calculate point-in-time illuminance values within the tileset. The same 3D city model used in Honeybee will be imported into Unreal for benchmarking purposes. The illuminance values obtained from Honeybee will be compared to those from Unreal.

To ensure the reliability and accuracy of the measurements, precautions will be taken to minimize potential errors.

1. Each measurement point will be measured three times, mitigating the impact of random errors and measurement inaccuracies by averaging the results.
2. To prevent information bias, the illuminance values from the Honeybee model will intentionally be hidden during measurements in Unreal. This approach aims to maintain objectivity and eliminate any unconscious bias that may arise from observing the values in the other application.
3. Comprehensive coverage of a wide range of daylight scenarios will be achieved by strategically placing measurement points on different surfaces, considering areas exposed to direct sunlight as well as shaded regions. Moreover, various environmental complexities will be considered, such as obstruction caused by surrounding buildings, as well as the presence of diffraction and refraction effects. By incorporating these diverse scenarios, the study aims to validate the ability of Unreal to accurately simulate and calculate daylight propagation in urban environments.
4. Finally, both the Unreal SunSky model and the Cesium SunSky model will be compared against the sun position in Honeybee and the resulting values.

By following these methods and conducting a thorough evaluation, the study aims to provide valuable insights into the accuracy and performance of the Unreal Engine in simulating and calculating daylight values in urban environments.

3.4 Interactive User Functionality

User interaction plays a vital role in facilitating the architectural and urban design process. The Unreal Game Engine provides various tools for enabling user interaction in the application. The application should allow users to interactively remodel the urban design and recalculate daylight on the spot. To achieve this, the application will support scale, rotate, and translate functionalities, enabling users to adjust the design to their desired specifications.

In addition to manipulating existing geometry, the application will also support the addition of new geometry to the urban environment. This will be facilitated through the inclusion of building blocks, offering a library of pre-defined elements that enable users to construct their designs quickly and efficiently. Furthermore, the application will also support the importing Cesium Tiles at runtime, enabling users to import new designs created in other applications or 3D city models.

To enhance flexibility and customization, the application will provide users with the ability to switch between different render modes and modify rendering settings at runtime. This feature will enable users to visualise and analyse the urban design in various perspectives and lighting conditions, providing a deeper understanding of the impact of their design choices on the lighting environment.

The effectiveness and capabilities of user interaction within the application will be evaluated based on several criteria. The design capabilities will be compared against industry standard tools for architectural and urban daylight design, such as Rhino. This analysis will help identify the strengths and weaknesses of the application in terms of usability, efficiency, and overall user experience. Furthermore, the application will be evaluated based on its in-game functionality versus the development mode functionality.

The creation and evaluation of user interaction and design capabilities aim to provide potential users with workflow recommendations and insights into the pros and cons of the application. These insights can guide future development, addressing any identified areas for improvement and improving the application's overall performance and user experience.

3.5 User Interface

Creating an intuitive UI is crucial for ensuring a positive user experience in the VR web application. The UI is designed to allow users to easily navigate between different features and functions of the application.

The UI will be developed using the Unreal Engine's built-in UMG (Unreal Motion Graphics) system. This system enables the creation of user interfaces that can be used in VR.

The UI will include a navigation menu that allows users to access the different functionalities from chapter 3.4, such as selecting different 3D models and changing the lighting settings.

4 Implementation

This chapter describes the implementation of this research, building upon the methodology outlined in Chapter 3. This chapter aims to give a clear understanding of how the research methodology was put into action and the different parts involved. The sections in this chapter will provide practical insights and technical details on how the research objectives were achieved. The application is available at: https://github.com/siebren014/LuminaCity. Additionally, a demo video showcasing the application's functionality is available at: https://youtu.be/qc9LLf2PT40.

4.1 Tools, datasets and hardware used

Table 4.1 provides an overview of the various datasets and software used in the research project. Each entry includes the name of the dataset or software, the source from where it was obtained, and its specific usage in the Thesis.
The hardware that was used for the development of this thesis is a laptop with an Intel Core i5-9300H processor with a speed of 2.40 GHz and a quad-core. The graphics controller is an NVIDIA GeForce GTX 1650. The laptop has 16GB of physical memory and 27GB of virtual memory.

Dataset or Software	**Source**	**Usage**
3DBAG Tile Rotterdam	(3D BAG Viewer, n.d.)	Dataset containing detailed 3D models of Rotterdam buildings. Used to evaluate the performance of importing and analysing different 3D models in the Unreal application, as well as used as benchmark dataset for comparison of light values.
Unreal Engine	(Unreal Engine | the Most Powerful Real-time 3D Creation Tool, n.d.)	Development platform utilised for creating the real-time daylight analysis tool.
Rhino	(Robert McNeel & Associates, n.d.)	Modelling software employed for benchmarking purposes. Allows for the creation and manipulation of 3D geometry, providing access to Grasshopper for additional analysis capabilities.
Grasshopper	(Grasshopper, n.d.)	Visual scripting software used in conjunction with Rhino for light analysis
Ladybug/Honeybee	(Ladybug Tools | Home Page, n.d.)	Daylight analysis plugins for Grasshopper. Used to build Point-in-time illuminance script.

FME	(Wij Zijn Tech | FME, n.d.)	Data integration and transformation tool. Used to convert 3DBAG to Cesium tile & CityGML.
Cesium ion	(Cesium Ion, n.d.)	Geospatial data hosting and visualisation platform. Used to host 3D city models.
Cesium for Unreal	(Cesium for Unreal, n.d.)	Integration of Cesium geospatial data in Unreal Engine and CesiumSunSky.
GitHub	(GitHub, n.d.)	Version control and hosting of applications.
VS Code	(Visual Studio 2022, 2023)	Integrated development environment for coding in Unreal. Used to add functionality in Unreal using C++
CityGML file Den Haag	(3D Stadsmodel Den Haag 2022 CityGML - Dataplatform, n.d.)	CityGML dataset of Den Haag (The Hague). Used to evaluate the performance of importing and analysing different 3D models in the Unreal application
citygml-tools	(Citygml4j, n.d.)	Used to convert CityJSON to CityGML
Google Maps API	(Google Maps Platform | Google for Developers, n.d.)	To generate a Google Maps API key to implement the google maps tileset

Table 4.1. Overview of tools and datasets used.

4.2 Loading 3D city models in Unreal

To test the different scenarios mentioned in section 3.2, the following steps will be taken:

1) Convert CityJSON to Cesium tile using FME
Since CityJSON is not an accepted format by Unreal nor Cesium ion, the CityJSON (3DBAG) dataset first needs to be converted to the Cesium Tileset format using FME (see Figure 4.2.1). Afterwards the Cesium Tileset can uploaded to Cesium ion. With the data uploaded to Cesium ion, a connection is made between Unreal and the Cesium ion account. From the Cesium ion assets tab in Unreal, the desired tile is selected and loaded into Unreal. Optionally, the z-coordinate can be adjusted to ensure proper alignment with the Cesium world terrain. It is important to consider that using FME for conversion adds additional steps to the workflow. However, this method allows you to import the CityJSON datasets into Unreal, providing enhanced

visualization and interaction within the Unreal environment.

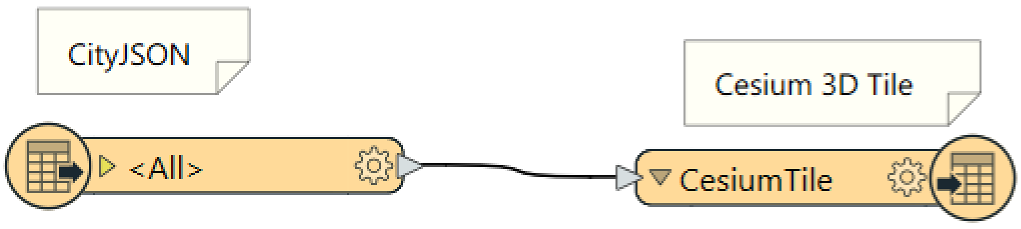


Figure 4.2.1. Converting CityJSON to Cesium 3D Tile using FME.

2) Load CityGML directly into Cesium ion.
CityGML is accepted by Cesium ion, so can be directly uploaded without conversion. When uploading the CityGML file to Cesium ion, there is the possibility to clamp the buildings to “Cesium World terrain” or “Mean Sea level”. This eliminates the need to manually adjust the z-coordinate of the buildings.

3) Convert CityJSON to CityGML using FME
Instead of converting a CityJSON file, into the Cesium tiles format, it can also be converted into CityGML format using FME (see Figure 4.2.2). The resulting CityGML file is then uploaded to Cesium ion, following the previously mentioned steps. This approach offers the advantage of enabling compatibility with the CityGML format for further processing and taking away the need to manually adjust the z-coordinate of CityJSON files converted to Cesium tiles. However, it requires using FME for conversion and involves additional steps for uploading to Cesium ion.

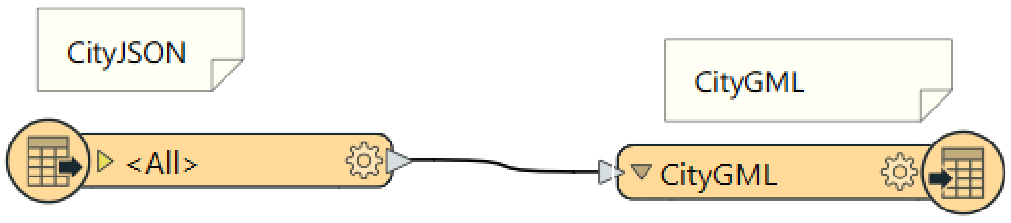


Figure 4.2.2. Converting CityJSON to CityGML using FME.

4) Convert CityJSON to CityGML using citygml-tools (Citygml4j, n.d.)
Alternatively, to using FME to convert CityJSON to CityGML format, citygml-tools can be used. Citygml-tools is the recommended tool to convert CityJSON to CityGML by the creators of the CityJSON format (*Converting to/From CityGML Files*, n.d.). To import the 3DBAG dataset into Unreal in CityGML format, utilize citygml-tools and the "from-cityjson" command in the command line to convert the dataset. Next, upload the resulting CityGML file to Cesium ion. In the Unreal Project, connect the Cesium ion account and load the desired tile from the Cesium ion assets. This method involves using command-line tools and the citygml-tools application, which adds additional steps for uploading to Cesium ion, which can add complexity to the workflow.

5) Cesium OSM Tileset
The Cesium OSM Tileset is a built-in feature of Cesium ion, which can easily be added to the Unreal project from the Cesium ion assets. This approach has the advantage of eliminating the need of uploading and importing datasets. However, the quality, or level of detail of the tileset might not meet the requirements of the user.

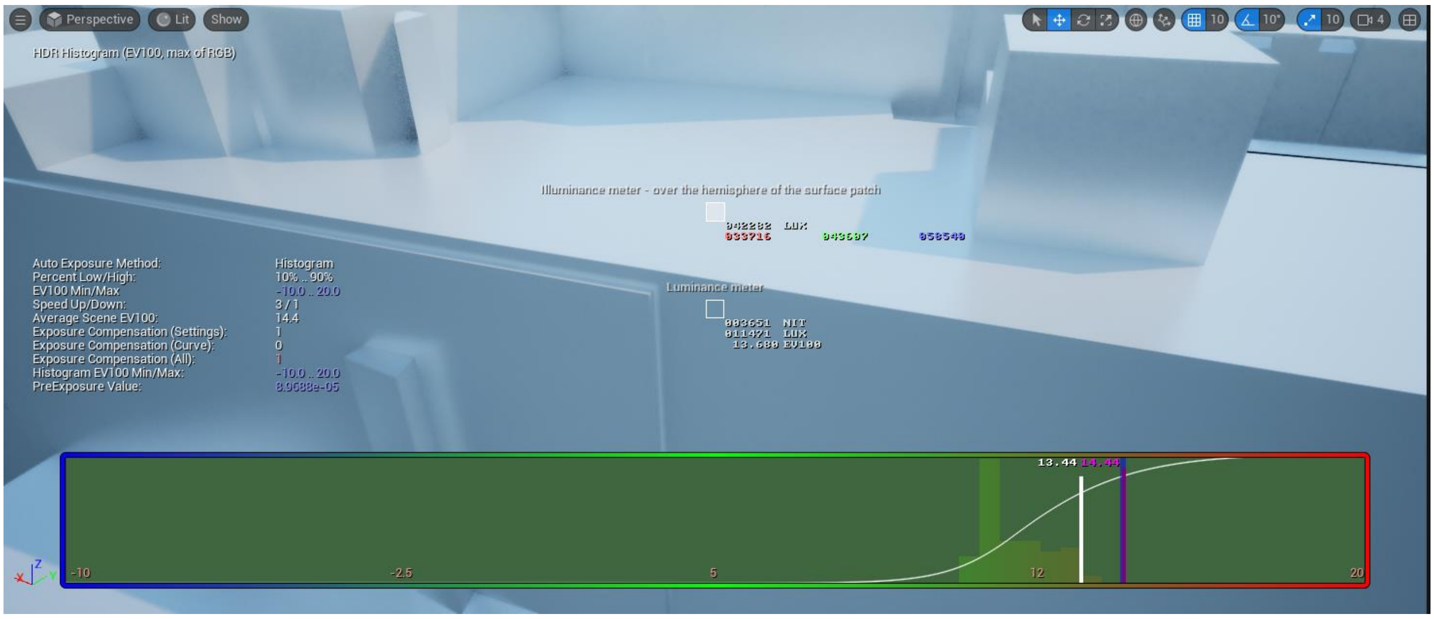
6) Google Maps API.
To integrate Google Maps data into your Unreal project, you will need to obtain a Google Maps API key. By pasting this API key into a "Blank 3D Tiles" tileset in Unreal, you can add the Google Maps API dataset of buildings in Unreal. This approach provides the advantage of seamlessly incorporating Google Maps data into your project. However, it is important to note that it requires a Google Maps API key and may introduce a dependency on external services.

4.3 Extracting light values from Unreal

To extract light values from Unreal, several methods were explored:

1) **HDR eye adaptation.**

To set up a reliable baseline for measuring luminance and illuminance values in the scene, the decision was made to use the built-in HDR eye adaptation tools (see Figure 4.3.1). The main consideration for this decision was the inherent availability and integration of these tools within the engine, which ensured their accuracy and reliability. By using the built-in nature of these tools, they served as a solid reference point for comparing light values with Honeybee and evaluating results obtained from alternative methods. This approach guaranteed that in case other methods did not perform as desired, the benchmarking process would still be supported by exact and dependable measurements from the built-in tools.
However, it is important to note that the built-in HDR eye adaptation tools have certain limitations. They do not work at runtime, require manual reading of light values, and lack the capability to export or save the measured data. Therefore, to address these limitations and ensure a comprehensive analysis of daylight values, it was imperative to explore more possibilities for extracting and calculating the desired information.


Perspective
Lit
Show
HDR Histogram (EV100, max of RGB)
Illuminance meter - over the hemisphere of the surface patch
Luminance meter
13.44 14.44
-10
-2.5
5
12
20

```
042282  LUX
033716        043607        058540

Auto Exposure Method:                 Histogram
Percent Low/High:                     10% .. 90%
EV100 Min/Max:                        -10.0 .. 20.0
Speed Up/Down:                        3 / 1
Average Scene EV100:                  14.4
Exposure Compensation (Settings):     1
Exposure Compensation (Curve):        0
Exposure Compensation (All):          1
Histogram EV100 Min/Max:              -10.0 .. 20.0
PreExposure Value:                    8.9688e-05

003651  NIT
011471  LUX
 13.680 EV100
```

Figure 4.3.1 Measuring light values with HDR eye adaptation tool

2) Accessing light information directly from build in functionality of Engine.
To harness the functionality behind the HDR eye adaptation tool at runtime, and export the measured daylight values, efforts were undertaken to directly access the functions responsible for calculating lighting by accessing them in a C++ script. Functionality from the file "PostProcessVisualizeHDR.cpp", containing HDR Eye Adaptation tool that measures luminance and illuminance values in development mode, and functionality from "IlluminanceMeter.cpp" were made "BlueprintCallable", which should allow accessibility through blueprints, making them available at runtime.

By making these functions available at runtime, the research aimed to use the power of the HDR eye adaptation tool and extract valuable lighting data in dynamic environments. This approach would supply the flexibility to capture and analyze luminance and illuminance values on the fly, facilitating real-time adjustments and enhancing the overall accuracy of our lighting analysis.

3) Calculating Luminance and Illuminance values using a custom C++ actor. As an alternative to utilizing the built-in functionality for accessing daylight information, a custom C++ actor was developed to calculate the luminance values at runtime. The actor passes the screen texture below the cursor as an input and calculates the luminance value based on the amount of light emitted from the surface. The implementation details of this custom actor are provided in pseudo-code in Algorithm 1.

Algorithm 1: Calculating Luminance and Illuminance values using a custom C++ actor.

```
input: Left mouse button L, Scene
if L == pressed then
    Get current viewport;
    x = Viewport.GetMouseX();
    y = Viewport.GetMouseY();

    Create array to Hold pixeldata.;
    Create rectangle to read pixel info from;
    Viewport.ReadPixels( PixelData, FReadSurfaceDataFlags(), rect );

    Red = PixelColor.R;
    Green = PixelColor.G;
    Blue = PixelColor.B;

    Luminance = ((0.265 * Red) + (0.670 * Green) + (0.065 * Blue));

    return Luminance;
```

To invoke the actor, a Blueprint (Appendix G) was created. The resulting luminance values were then normalized within a range of 0 to 256 and multiplied by the intensity of the light. Alternatively, they can be multiplied by 179, to follow the method of Radiance to calculate luminance. In the user interface, the resulting luminance values are shown in the top left corner of the screen (Figure 4.3.2).

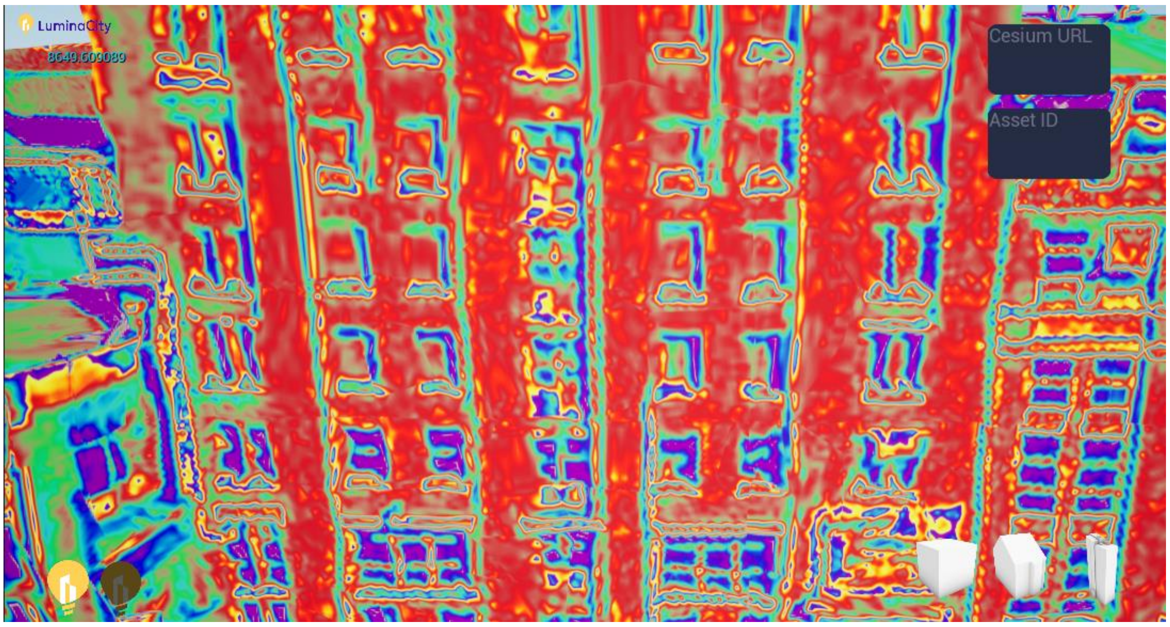


Figure 4.3.2 Custom C++ actor to calculate luminance (value at top left part of the screen)

4) **False Colour Post-Processing Material.**

As an alternative to the previous methods, that numerically represent the amount of light on a surface, a false colour Post-Processing material was developed to visually represent the light distribution in a scene. By utilising the PostProcessInput0 parameter and a false colour texture, objects in the scene are assigned colours based on their Illuminance values, providing a visual representation of the light distribution. The implementation code is provided in Algorithm 2 and the resulting rendering style can be observed in Figure 4.3.3.

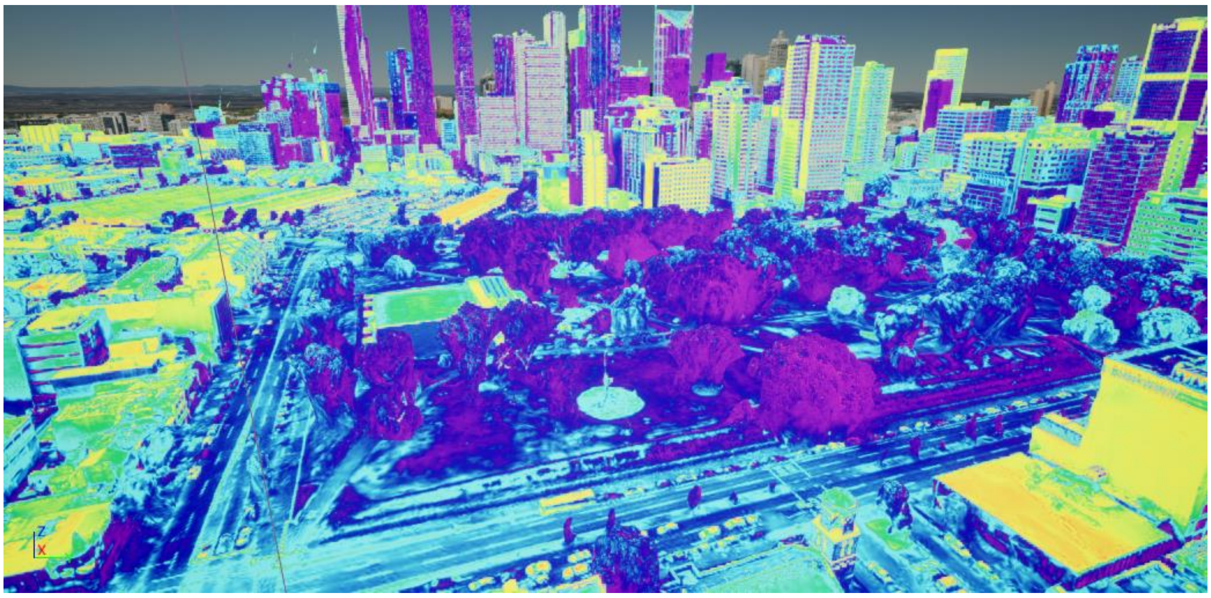


Figure 4.3.3 False Colour Post-Processing Material

To ensure an informative visual representation without distracting the user, the false colour material is only applied to the buildings. In Unreal Engine, the sky is rendered at a considerable distance from the scene, with a deep scene depth. By leveraging the "depth" information, objects with a scene depth smaller than 10,000 are exclusively rendered, effectively excluding the sky from the false colour representation. Similarly, the terrain can be excluded from the rendering by adjusting its "custom depth" value.

In addition to illuminance values, the same logic and a custom material node with C++ code were employed to represent the amount of luminance in false colour. The corresponding blueprints can be found in Appendix H.

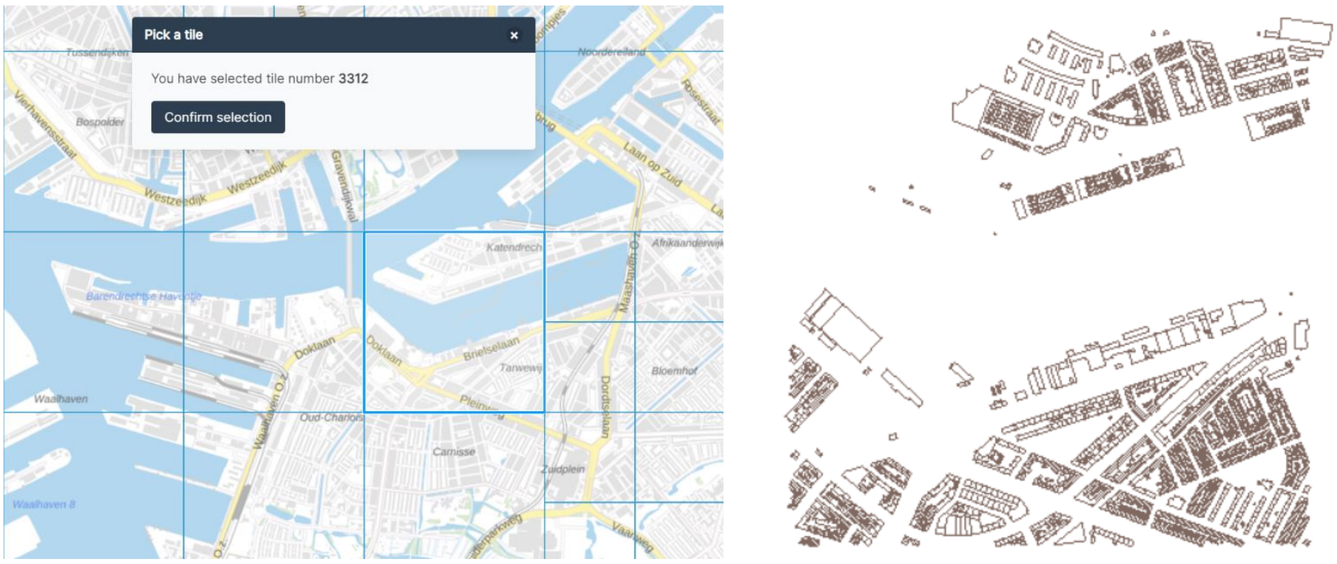
This post-processing material allows for the visual representation of both luminance and illuminance values in false colour. The colours assigned to specific values can be customized using the created custom material node, providing flexibility for representing the data in relative or absolute scales.

Algorithm 2: False Colour Material

```
Input: Scene, SceneTexture

Get Scene;
SceneTexture.PostProcessInput0 = light information;
SceneTexture.Basecolor = base color of the object;

for Every Scene tick do
    Mask R component of PostProcessInput;
    Texture sample = R component * False color texture object;
    Color = Texture sample * Basecolor;

    Non-sky objects: if SceneDepth is smaller or equal to 100000 then
        SceneColor = Color
    end

    Sky objects: else if SceneDepth is bigger than 100000 then
        SceneColor = PostProcessInput0
    end

    return SceneColor;
end
```

4.4 Comparison of different light analysis methods vs. Benchmark

The inverse square law experiment was implemented in Unreal by creating a point light of 100 lux and measuring the Illuminance values on a white surface at r distance from the source. Besides that, there were no other objects or lights active in the scene, to have the results as accurate as possible. To benchmarking the daylight propagation of Unreal in urban environments and facilitate a comparison with values from Honeybee, a 3D city model was used. The dataset (see Figure 4.4.1) that was used is tile 3312 from 3DBAG. This dataset comprises 3010 buildings located in Rotterdam, the Netherlands, with coordinates 51.896438174141096, 4.477799465615449, covering an approximate area of 1.75 m². The initial file size of the dataset before importing into the system was 19716 kB.


Pick a tile

You have selected tile number **3312**

Confirm selection

Figure 4.4.1 Benchmarking dataset with 3010 buildings in Rotterdam

4.4.1 3D city model & Benchmarking in Honeybee

The implementation of the benchmarking phase encountered several challenges that required adjustments to the initial plan. The intention was to utilise the 3DBAG CityJSON tile for comparison; however, it posed compatibility issues with Rhino, as CityJSON format is not supported. Consequently, the OBJ version of the 3DBAG tile was used instead, which could be loaded in Rhino. However, it was not compatible with the Honeybee model due to specific geometry requirements.

Attempts were made to fix the geometry of the mesh using methods such as mesh repair and mesh Boolean union. Unfortunately, these measures proved ineffective as the 3DBAG OBJ consists of multiple meshes, making it incompatible with Honeybee. The presence of height differences on the ground level further complicated the situation. It became evident that Honeybee required individual buildings, with adjacent rooms and closed volumes.

To address these issues, the benchmarking dataset was simplified to two variants. Firstly, a manually constructed simplified version of a section of the original dataset was created, ensuring adjacent closed volumes (see Figure 4.4.2). Secondly, a single building was selected from the original dataset to generate a Honeybee model, while the remaining buildings served as context to cast shadows on the model without calculating their individual light values (see Figure 4.4.3).

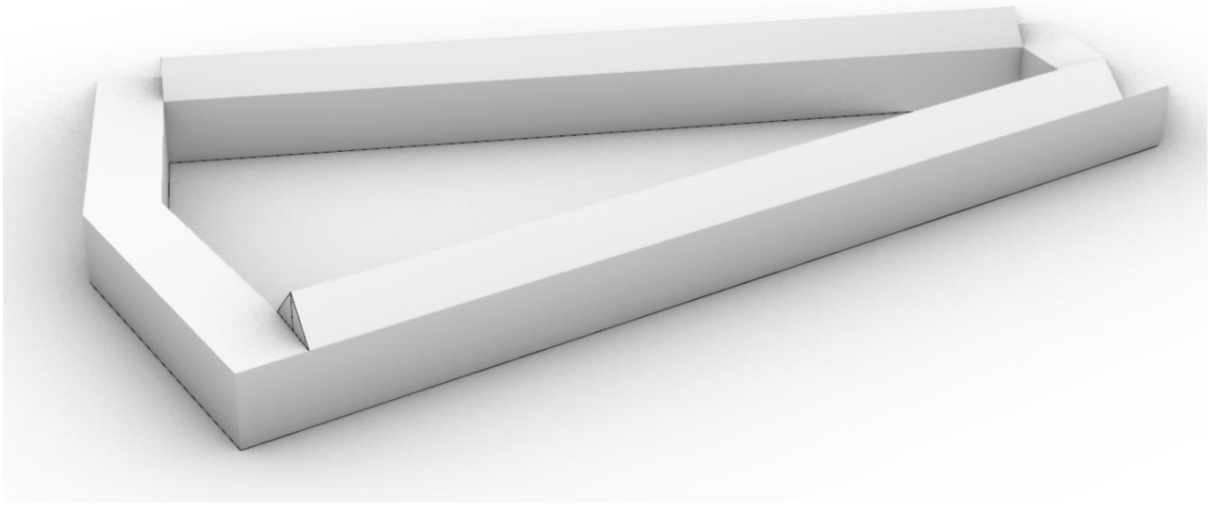


Figure 4.4.2 simplified version of a section of the original dataset with adjacent closed volumes

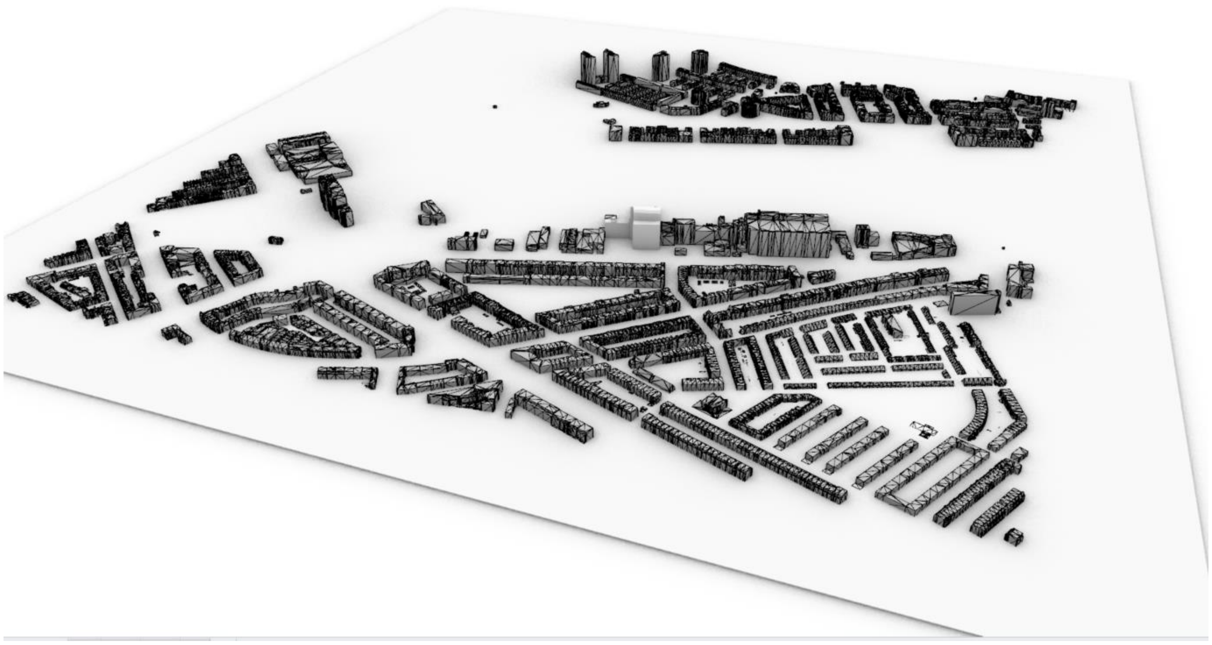


Figure 4.4.3 Complete Benchmarking dataset in Rotterdam

These two simplified versions were then used as input for the Grasshopper script that calculates the point-in-time luminance & Illuminance values. The script is implemented in the following way:

- An adjacent, closed volume is loaded as Building input.
- The other geometry is loaded in the Context geometry input.
- These 2 inputs are then combined to for the Honeybee Model
- Weather data is loaded from a weather file from https://www.ladybug.tools/epwmap/. In this research the weather file which is closest to the site of the benchmarking location was chosen, which was at Schiphol Airport Amsterdam.
- The Honeybee Model is created with a resolution of 5 cm using the "grid_size" parameter.
- To optimize calculation times, the Radiance Parameters, also known as "RadPar", are set to detail level 0 (low).
- The Weather data and Honeybee Model are used as input for a point-in-time image that calculates the brightness values given a particular point in time. In the case of this research point-in-time is 21st of June at 12:00
- These values are then used as input data for 2 HDR images. The first one is a FalseColor, which is mainly used for visual appearance. The AdjustHDR is used to extract the brightness values.

A simplified diagram of the script can be seen in Figure 4.4.4. The full script is placed in Appendix C.

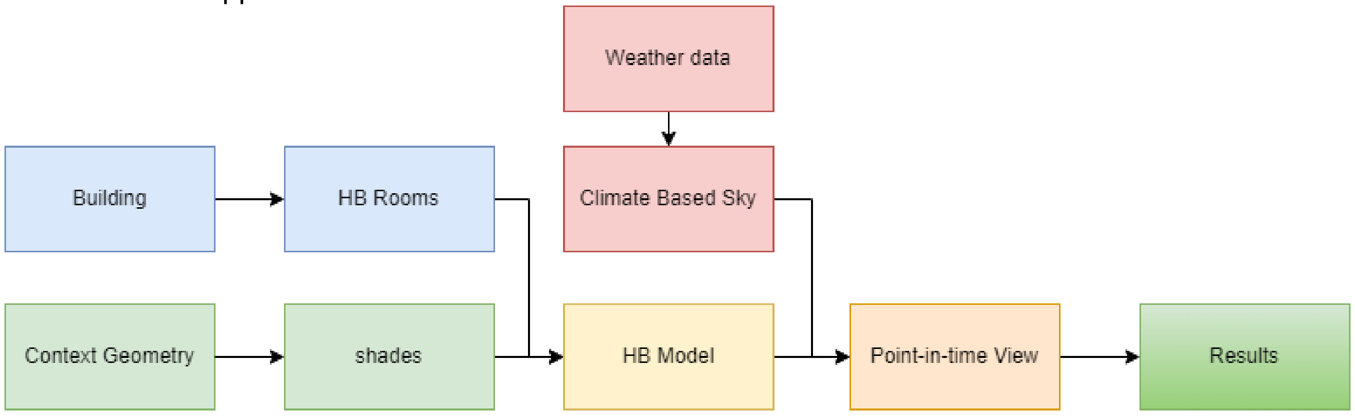


Figure 4.4.4 Simplified diagram of Grasshopper script

4.4.2 Measurement Points

To perform a benchmarking of the daylight values in Unreal and Honeybee, the same model of buildings that were used in Honeybee, was imported in Unreal. Once the Unreal model was set up, illuminance values were extracted from both the Unreal and Honeybee models. The values in Unreal were extracted using the build-in HDR Eye Adaptation tool at runtime (see Figure 4.3.1). In Honeybee the values were extracted by creating a HDR image and clicking on the location of the points to extract the illuminance values (see Figure 4.4.5). To ensure the reliability and accuracy of the measurements, several precautions were taken to minimize potential errors and information bias.

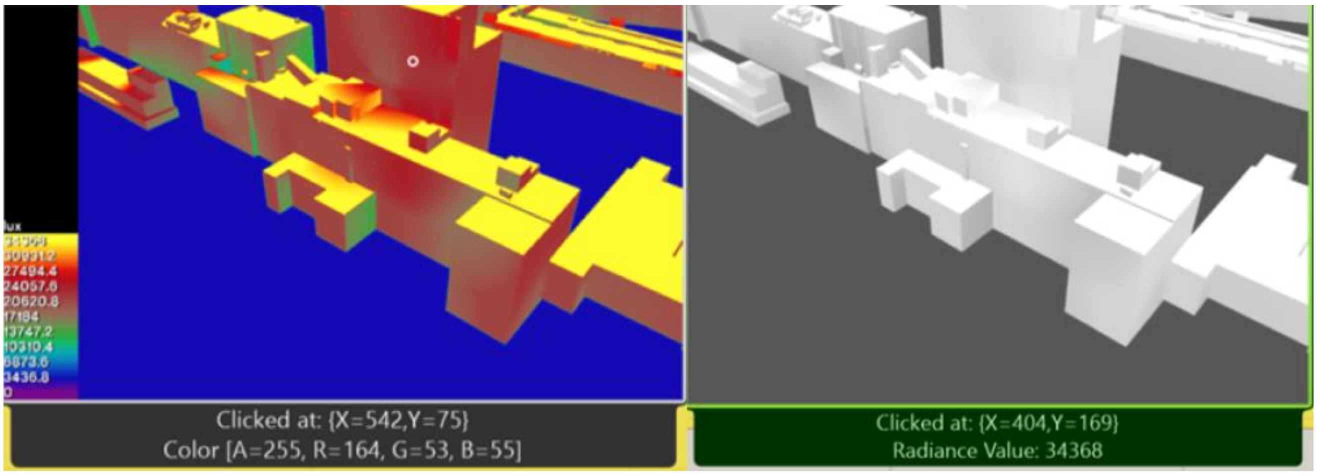


Figure 4.4.5 HDR image in Honeybee used to measure illuminance

To address measurement errors, each measurement point was measured three times, taking the average of these measurements. This approach aimed to enhance the overall precision and reliability of the collected data, by reducing the impact of random errors and measurement errors. To prevent information bias, the illuminance values from the Honeybee model were intentionally not visible during the measurements. This approach was implemented to eliminate any unconscious bias that might arise from seeing the values in the other application. By concealing the Honeybee values, the measurement process remained objective and unbiased.

To ensure comprehensive coverage of various scenarios of daylight propagation, the measurement points were strategically placed on different surfaces, taking into consideration areas exposed to direct sunlight as well as shaded regions. The specific placement of the points can be seen in Figure 4.4.6, 4.4.7, 4.4.8 and 4.4.9. This approach allowed for an examination of how light propagated in both illuminated and non-illuminated areas within the models. Furthermore, the choice of measurement points aimed to represent a balanced representation of varying environmental complexities. Points were chosen with various levels of obstruction caused by surrounding buildings, as well as varying degrees of diffraction and refraction effects. By incorporating these different scenarios, the study sought to display the overall ability of Unreal to accurately simulate and calculate daylight propagation in diverse urban environments. The hypothesis for the benchmarking is that in complex scenarios, the disparities in daylight values between Unreal and Honeybee will be greater compared to simpler scenarios.

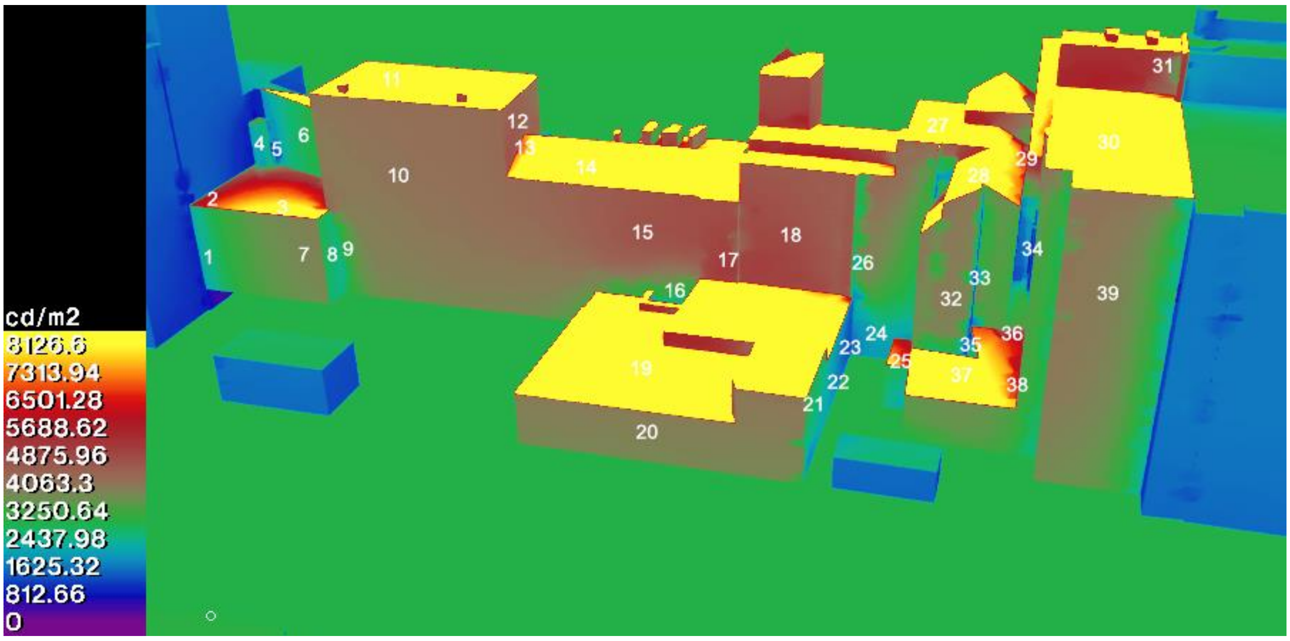


Figure 4.4.6 Measurement points 1 to 39

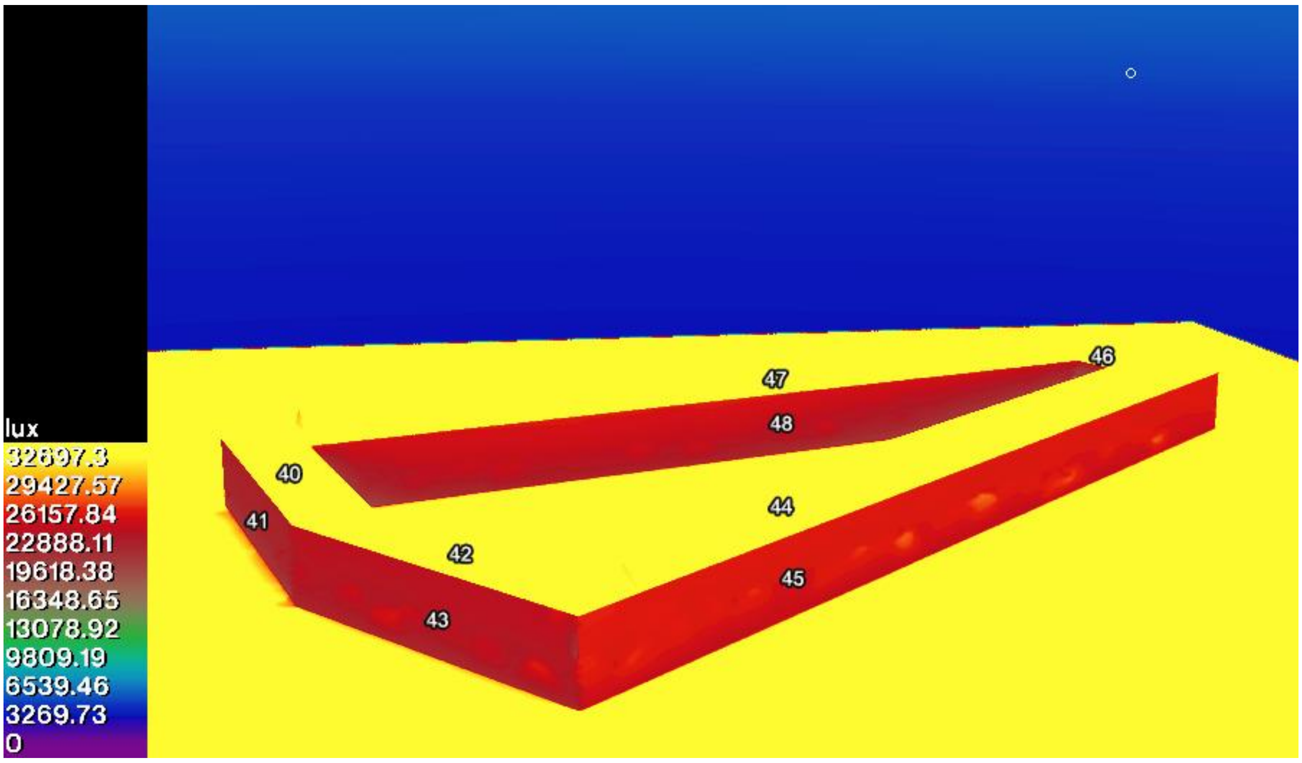


Figure 4.4.7 Measurement points 40 to 48

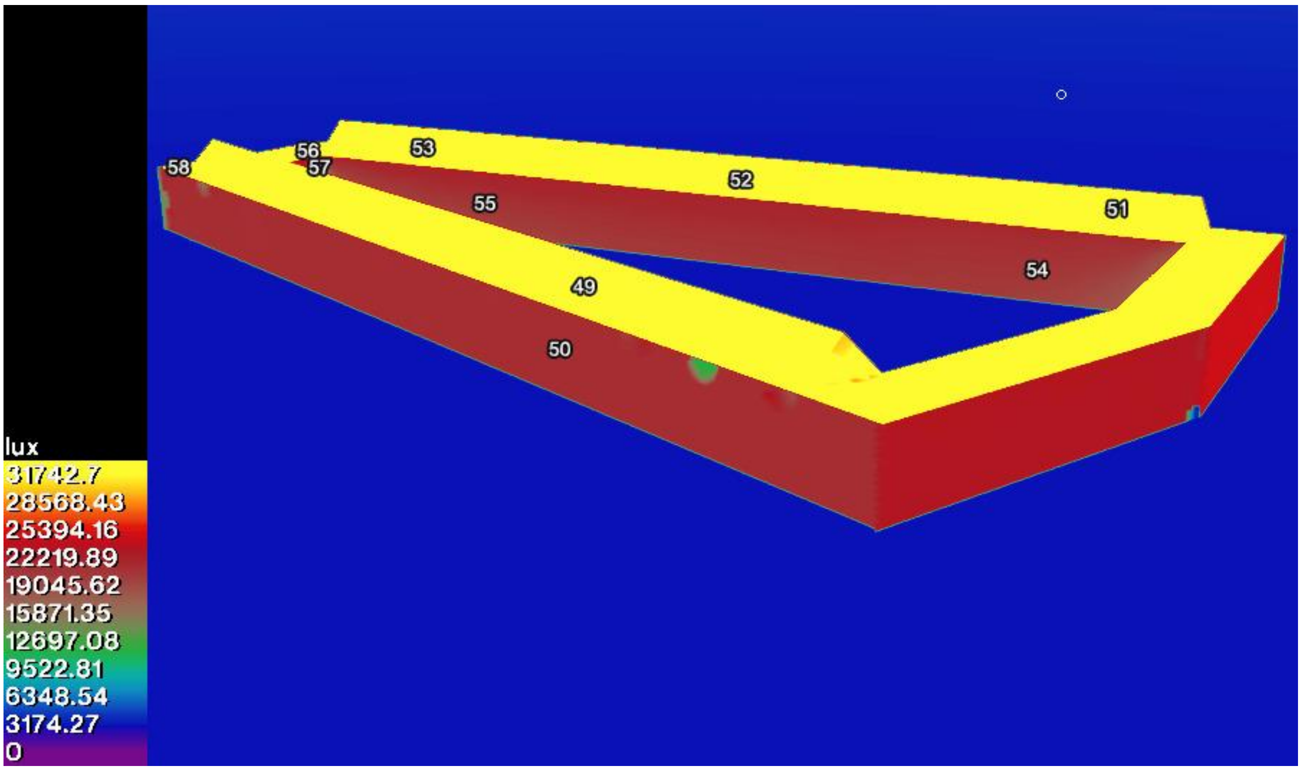


Figure 4.4.8 Measurement points 48 to 58

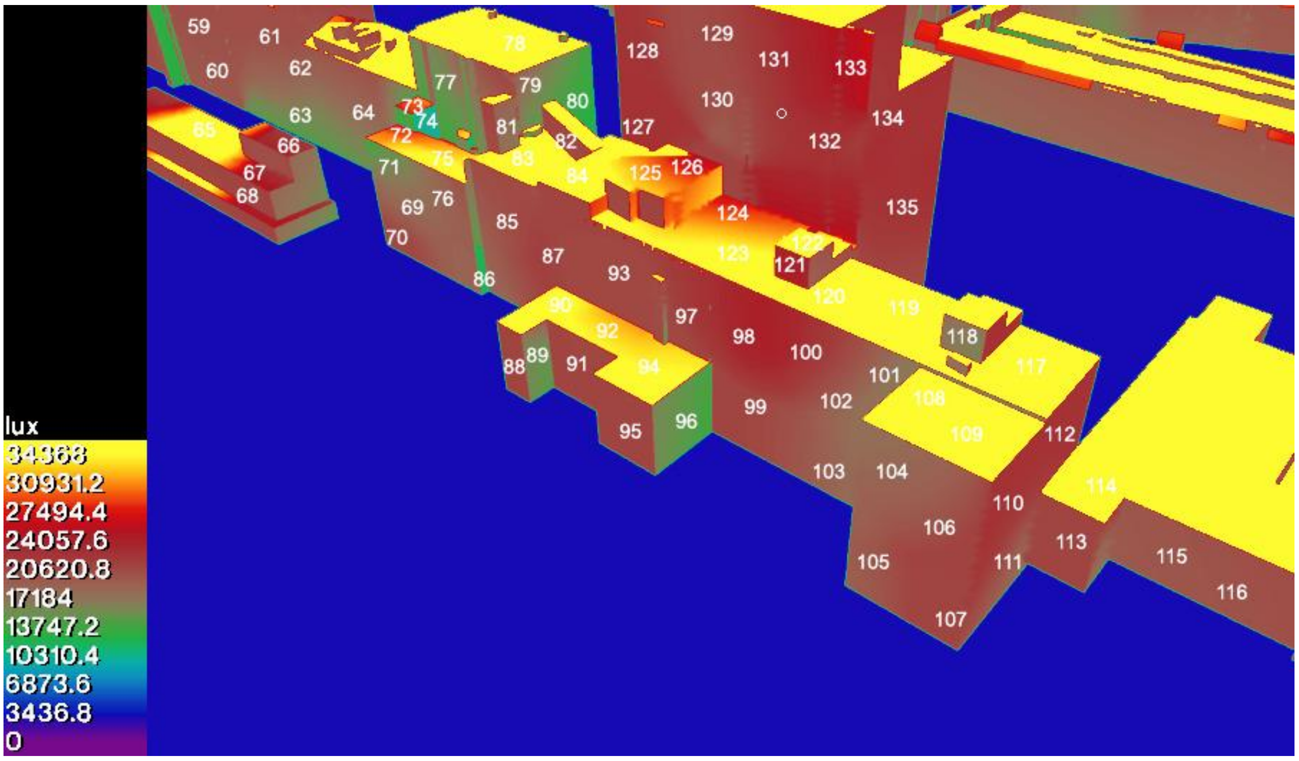


Figure 4.4.9 Measurement points 59 to 135

4.4.3 Optimizing sun & sky model in Unreal

Unreal offers 2 different sun & sky models, that of Unreal, and that of Cesium ion. During the benchmarking process, the choice of sun models was an important consideration. To ensure accuracy and alignment with the Honeybee model, both options were compared against the position of the sun in Honeybee and the resulting values.

Upon first observations, it became evident that the standard light values of both the Unreal SunSky model, and the Cesium SunSky model, were both too high, and not automatically adjusted based on date, time, and position. This meant that the values of direct sunlight and sky light had to be adjusted manually. The direct sunlight component has the physical unit of lux, but the skylight models do not have a physical unit, but rather a relative one.

Initial experiments with the direct sunlight, showed that values of lit areas in Unreal generally aligned with those from Honeybee. However, areas without direct sunlight showed significantly lower values. To address this discrepancy, adjustments were made to the Skylight intensity and Directional light values.

To achieve a more realistic representation, the Skylight intensity was increased. This adjustment enhanced the contribution of indirect lighting from the sky, compensating for the lower values in areas without direct sunlight. Additionally, the Directional light intensity was decreased.

Finding the right balance between values in simpler scenarios with minimal reflection and those with extensive reflection, as well as scenarios where there was no direct light and those with direct light, was crucial to have come to a realistic propagation of daylight under various scenarios in Unreal. This involved an iterative process of fine-tuning the Skylight and Directional light to accurately capture the propagation of daylight and reflection in different environmental conditions.

Finally, after evaluating various scenarios and light sources, it was found that the optimal balance was achieved using the Unreal SunSky with the skylight set to 15 and the sunlight to 25000 lux. Through these adjustments and considerations, the Unreal model achieved a closer alignment with the Honeybee model, ensuring a more exact representation of lighting conditions and enhancing the overall realism of the simulation. The fact that the skylight in Unreal does not have a unit, makes it difficult to compare the values of the light settings from Unreal with Honeybee. In Honeybee the light values of direct sunlight were 44800 lux and indirect light 1200 lux.

4.5 Interactive User Functionality

To make the platform interactive and usable for design rather than solely light analysis, functionality was added that allows users to explore the urban design, move around freely, and interact with objects and elements within the scene.

Building Blocks: To enhance the ease and efficiency of design iterations, the platform offers a library of predefined building blocks. These standardised components could be readily integrated into the design, enabling rapid prototyping and exploration of various architectural configurations. Users can select various objects, including a house, cube & appartement building, which they can drag into world space, and place on the terrain (see Figure 4.5.1). The implementation can be seen in appendix A.

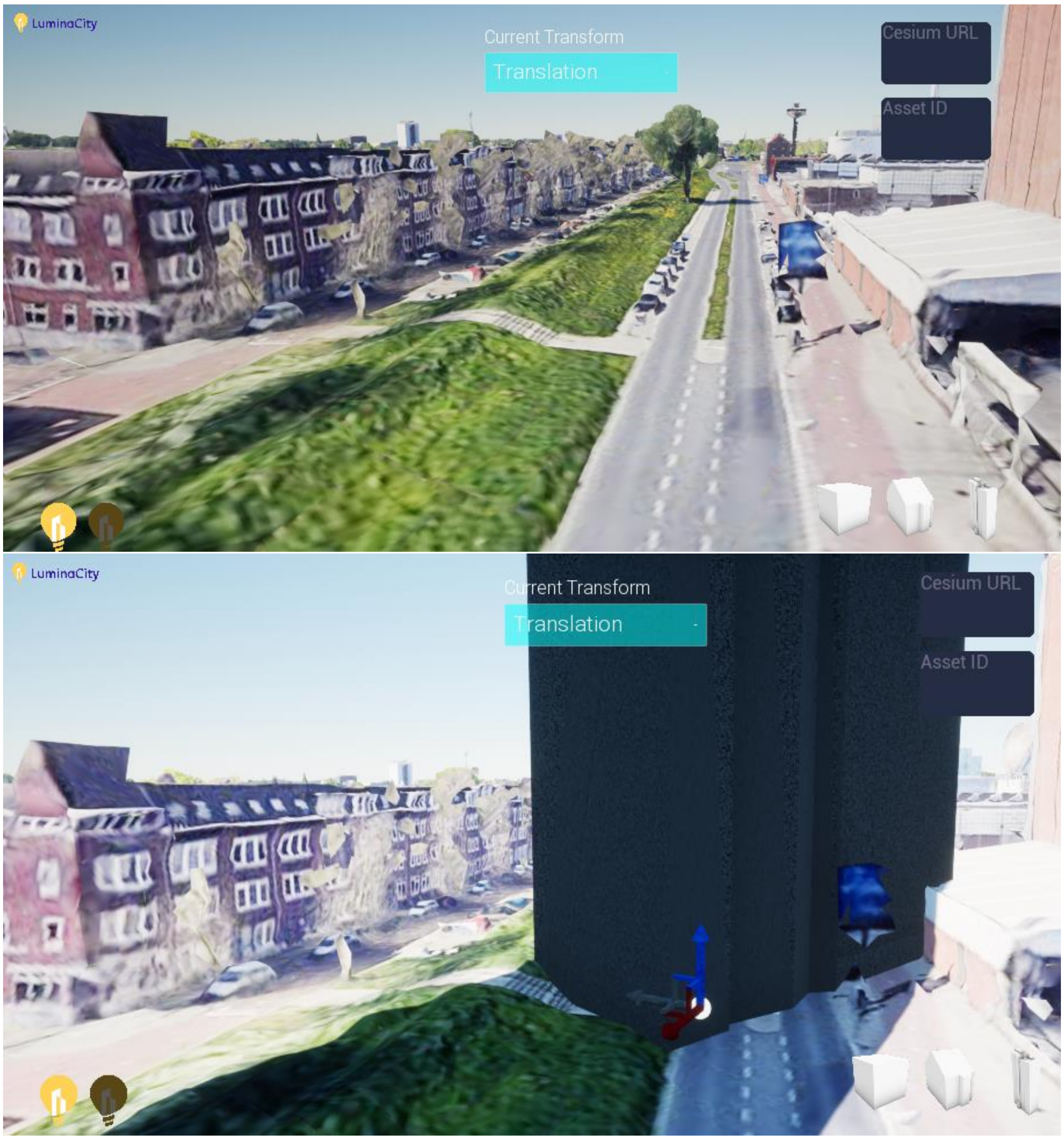


Figure 4.5.1 Building Blocks

Scale, Rotate, and Translate: To facilitate design modifications, the platform incorporated intuitive controls for scaling, rotating, and translating objects (see Figure 4.5.2). This was done by implementing C++ code based on the RuntimeTransformer GitHub repository (Xyahh, n.d.).
The application is available as a plugin that works for UE4, but not for UE5. As this thesis is implemented in UE5, the code was slightly adjusted and implemented as normal code, instead of a plugin. This decision was made as it was observed that plugins often become incompatible with newer versions of the Unreal application, whereas C++ code is easily transferred from a project in one version of the Engine to another version.

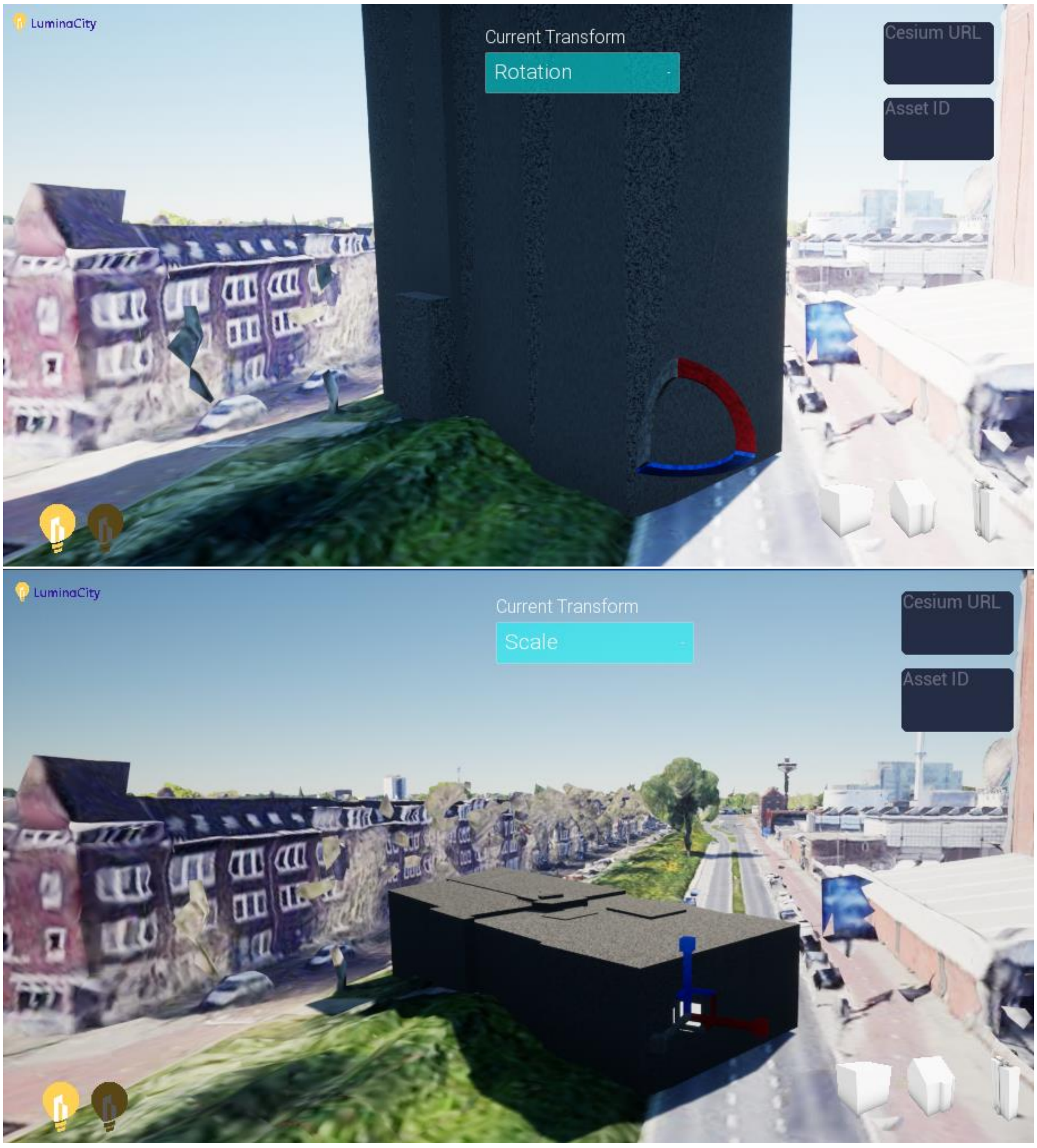


Figure 4.5.2 Scale, Rotate, and Translate geometry at runtime

Cesium Tiles at runtime: Leveraging the capabilities of Cesium ion, the application integrated dynamic placement of Cesium Tiles at runtime, enabling users to import geometry created in other applications (see Figure 4.5.3). To initiate the import process, the user simply provides the URL of a Cesium ion Tile, which is then loaded into the application. This allows users to import designs and tilesets from external sources. The implementation can be seen in appendix E.

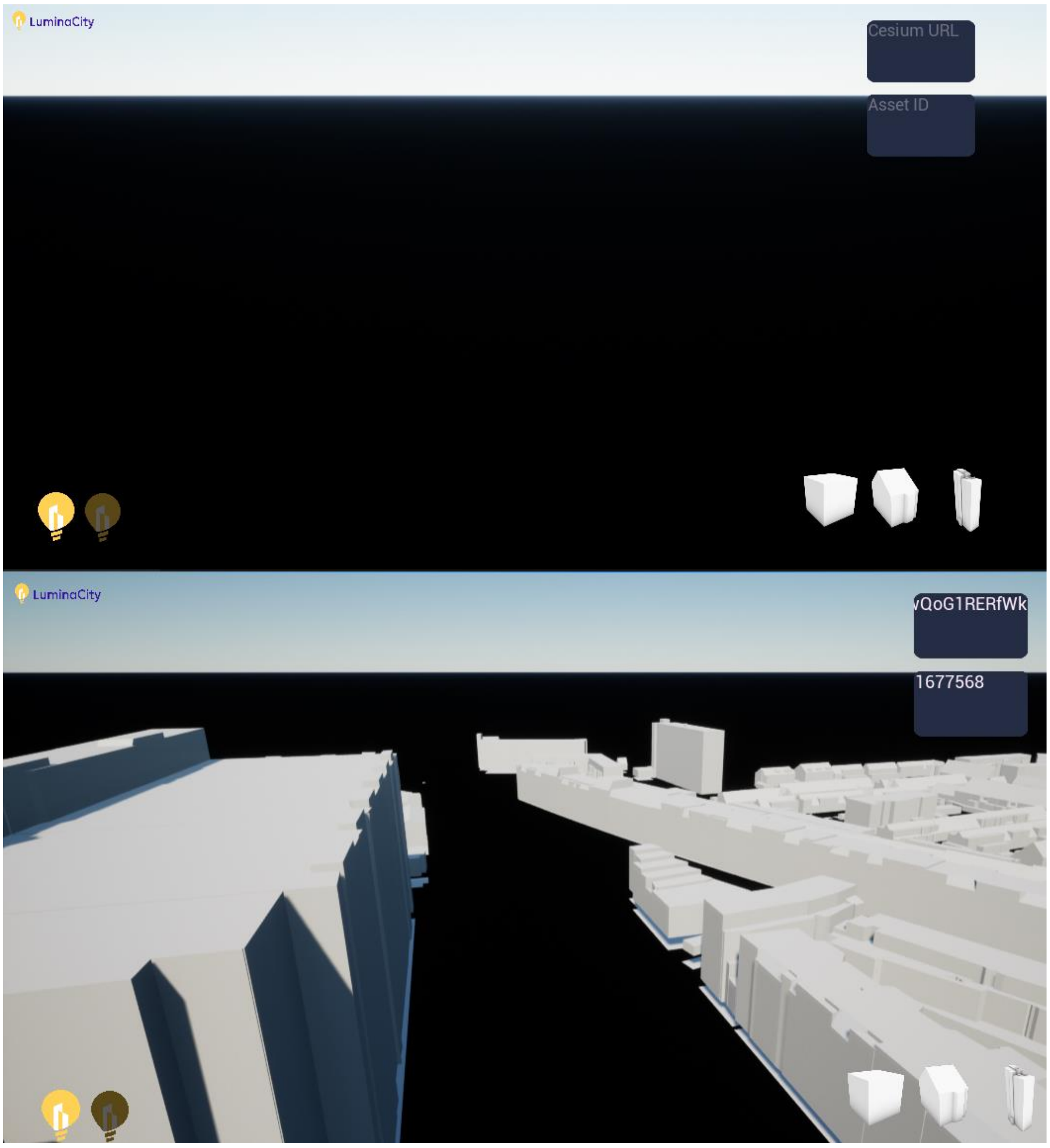


Figure 4.5.3 Import Cesium ion tiles at runtime

Change Render at Runtime: The platform offers the flexibility to change the rendering settings at runtime. Users can dynamically adjust lighting conditions, shadows, and other rendering parameters to simulate different times of day or environmental conditions. This is done by when a user presses the button lit or dark logo on the bottom left of the screen (see Figure 4.5.4), the Post-Processing material is either switched on, or off. The blueprint implementation can be seen in appendix B.

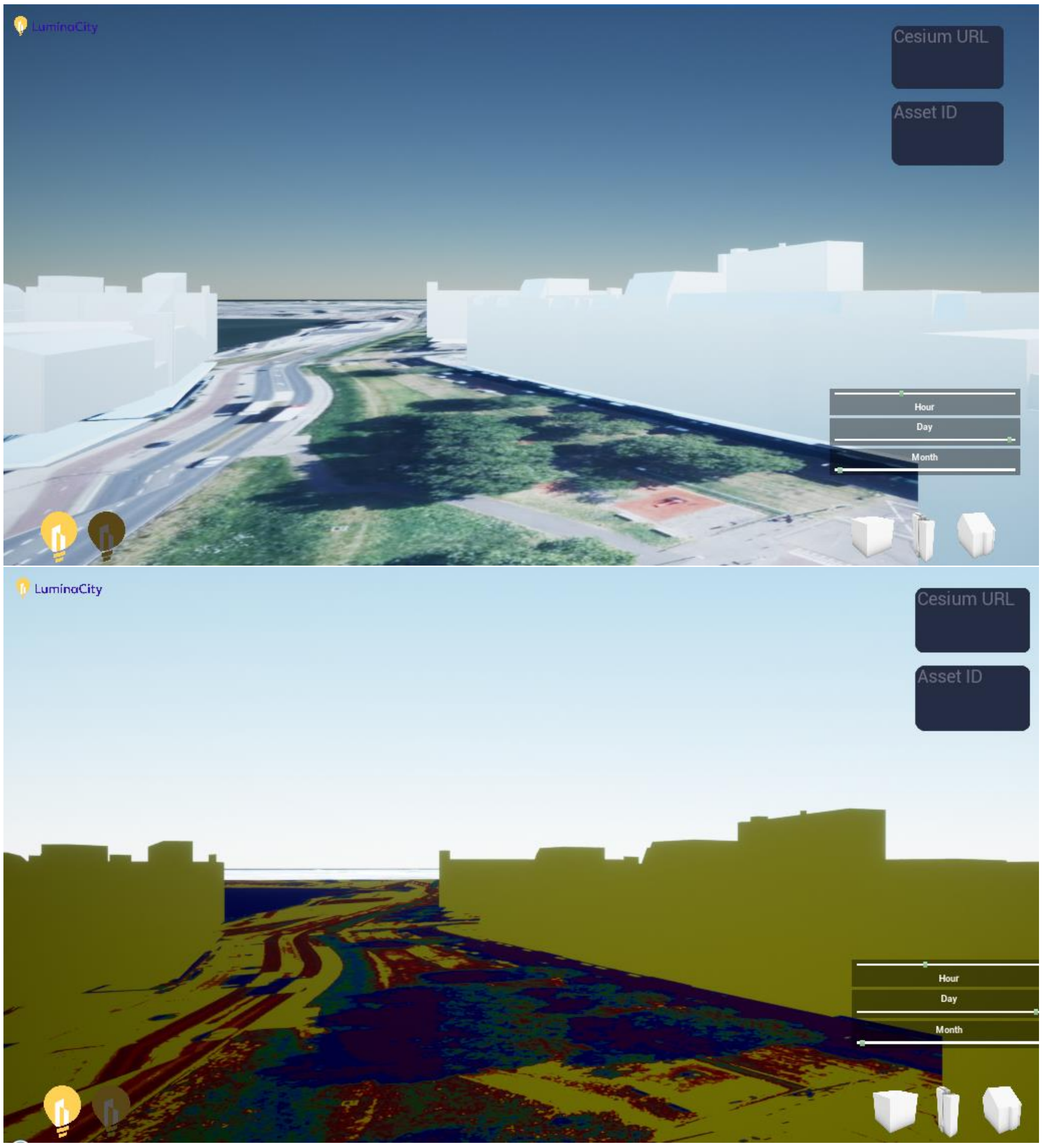


Figure 4.5.4 Changing render at runtime

Change sun position: In addition to the user interaction features, the implemented platform includes the capability to update the sun position, date, and time. This functionality allowed users to simulate different lighting conditions and evaluate the impact of varying solar positions throughout the day and across different seasons. The users can drag the different sliders to adjust the date and time (see Figure 4.5.5). The implementation can be seen in appendix F.

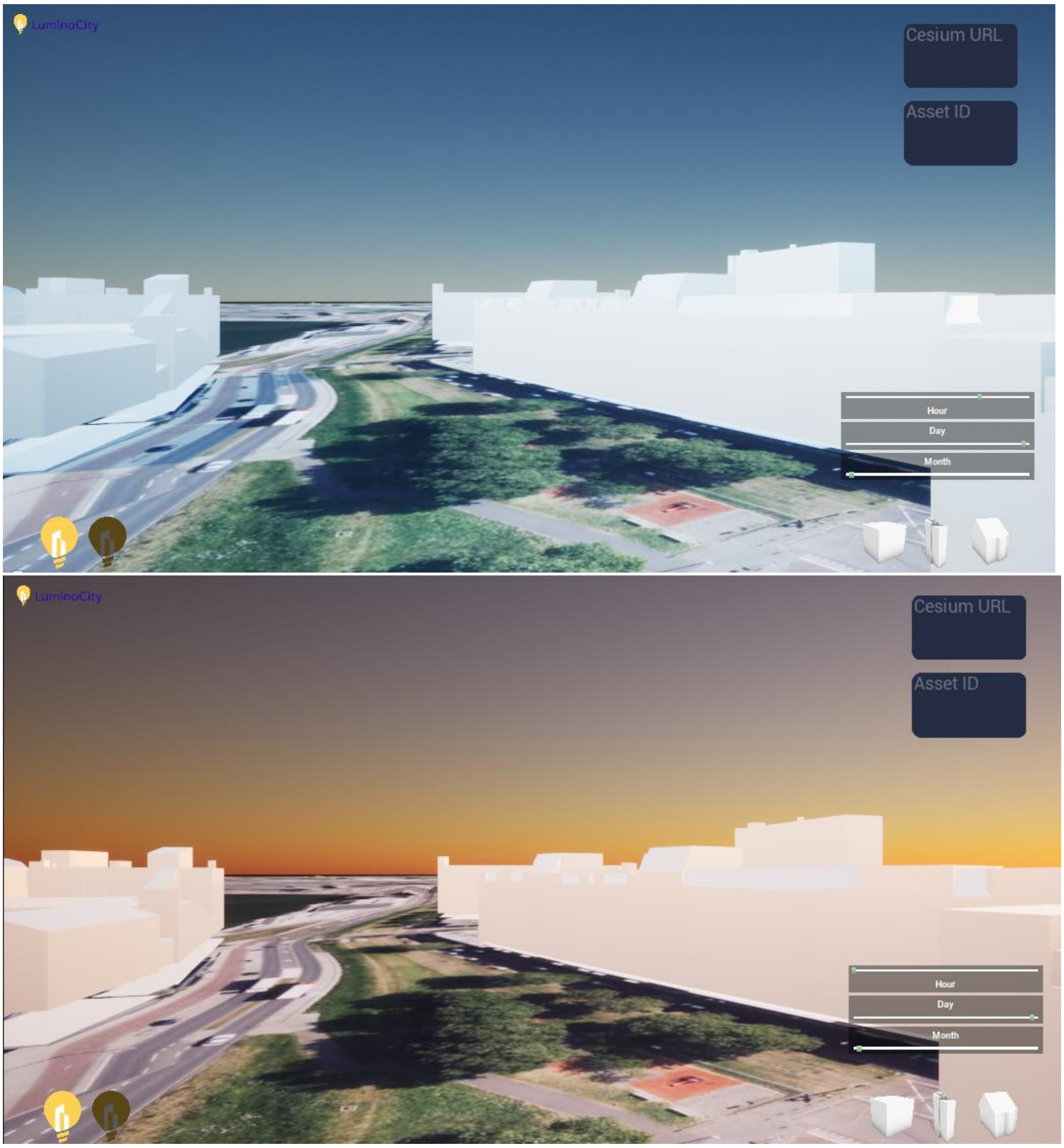


Figure 4.5.5 Changing date & time at runtime

4.6 User Interface

The UI is developed using the Unreal Engine's built-in UMG (Unreal Motion Graphics) system. This system enables the creation of user interfaces that can be used in different systems including Windows, Mac, Linux and VR.

The implemented UI, as can be seen in Figure 4.6 and depicted in Blueprints in Appendix I, includes various components. The menu located on the right side of the screen displays various buttons, including a dynamic text field. This text field facilitates the loading of designs through Cesium ion. Users can import designs from Cesium ion by pasting a link into a text field. Other sliders are available to adjust the day, time, and month. Additionally, icons of a cube, row house, and apartment building, for instance, are situated in the lower-right corner, allowing users to drag and drop these objects into the virtual "world". Furthermore, it enables users to add, scale, rotate, and move buildings or geometry within the environment, and change the render settings.

The UI aims to provide users with a seamless and intuitive experience within the application, enabling efficient navigation and interaction with its diverse functionalities.

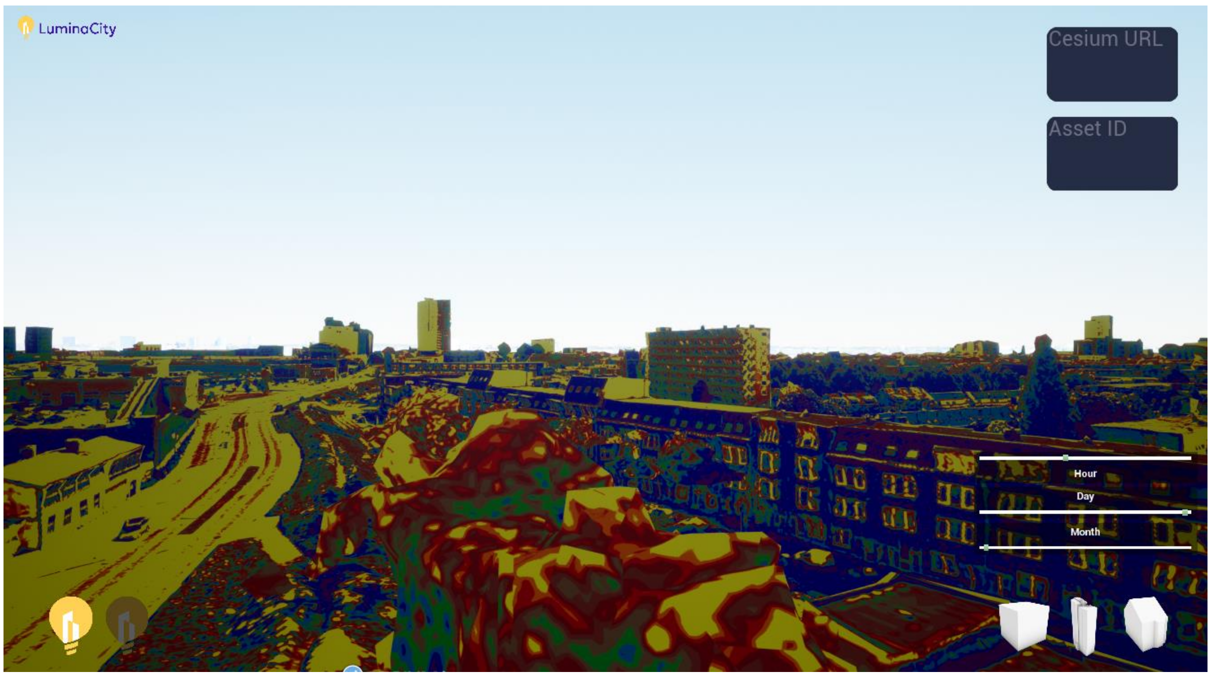


Figure 4.6 UI at runtime

5 Results & Discussion

The results section of this thesis presents the findings and outcomes of the research conducted to explore the potential of using the Unreal Engine for real-time daylight analysis. The purpose of this section is to provide a comprehensive overview and analysis of the data collected during the study.

5.1 Inverse square law

The inverse square law experiment conducted in Unreal aimed to examine if the light propagation in Unreal is realistic. The theoretical values were calculated based on the expected behaviour of light intensity according to the inverse square law. These were then compared with the results from measurements in the Unreal application. The obtained measurements are in Table 5.1. The obtained results from Unreal closely aligned with the expected trend, demonstrating a decrease in light intensity as the distance from the source increased. The measured values in Unreal were in line with the theoretical values, indicating the accuracy of the simulation in replicating the physical properties of light. Minor deviations between the Unreal values and the theoretical values can be attributed to various factors, including the inherent limitations of the simulation and slight measurement inaccuracies.

These results demonstrate promising findings for the further benchmarking of daylight analysis. The close alignment between the measured values in Unreal and the theoretical values indicates the engine's capability to accurately replicate real-world lighting behaviours. It is important to reiterate that this experiment was solely done to get an initial understanding of Unreal's ability to accurately simulate light propagation. Although promising for further benchmarking, these results do not imply Unreal's ability to realistically propagate light in (complex) urban environments.

	Source	1m	2m	3m	4m
Theoretical Value	100	100	25	11,11	6,25
Unreal Values	100	99,021	24,762	10,394	5,894
% difference	n/a	0,98	0,99	6,44	5,70

Table 5.1 Inverse square law experiment

5.2 Importing 3D city models

As previously mentioned in Chapter 3.1, Cesium 3D Tiles are the OGC community standard for streaming 3D tiles (Getz, 2021). However, the process of uploading 3D city models to the Unreal application using Cesium ion revealed several notable findings. Out of the 5 methods to import or add 3D city models to the Unreal application, only the 3rd method of converting CityJSON to CityGML using citygml-tools (Citygml4j, n.d.), failed. The citygml-tools format for converting CityJSON to CityGML was not compatible with Cesium ion.

During the process of importing different formats of 3D city models into the Unreal application with Cesium ion, notable discrepancies were observed in the z-coordinate values of the placed buildings across different formats. Cesium ion accepts CityGML format for importing 3D city models, providing a viable option for integrating geospatial data into the Unreal environment. It is crucial to note that Cesium World terrain, being a custom Geodic terrain offered by Cesium, can potentially affect the placement and alignment of the imported models.
Various models and ways of importing yielded various results related to z-coordinate of the buildings. CityGML buildings, when imported, can be clamped to either the Cesium World terrain, or the ellipsoid. However, CityJSON files that are converted to CityGML using FME, although clamped to the same terrain as files that originally came in the CityGML format, are displayed at a different z-coordinate, usually above the terrain. Cesium ion does not provide any specifics of the format of CityGML that is accepted or recommended, but not every format of CityGML is accepted or clamped to the terrain correctly.
When uploading a Cesium tileset to Cesium ion, it does not provide the option to clamp it to a terrain, instead its CRS is used, which clamps it to the ellipsoid. This causes CityJSON files that are converted to Cesium tiles using FME, to be displayed below the Cesium World terrain.
To overcome these limitations, an alternative approach was explored, which involved uploading the city models as Cesium tiles using FME and adjusting their z-coordinate manually. This method allowed for the georeferencing of models on the Ellipsoid, providing greater flexibility in positioning and alignment. However, it should be noted that if the desired outcome is to incorporate Cesium World Terrain, manual adjustment of the model's z-coordinate is necessary to ensure proper alignment.

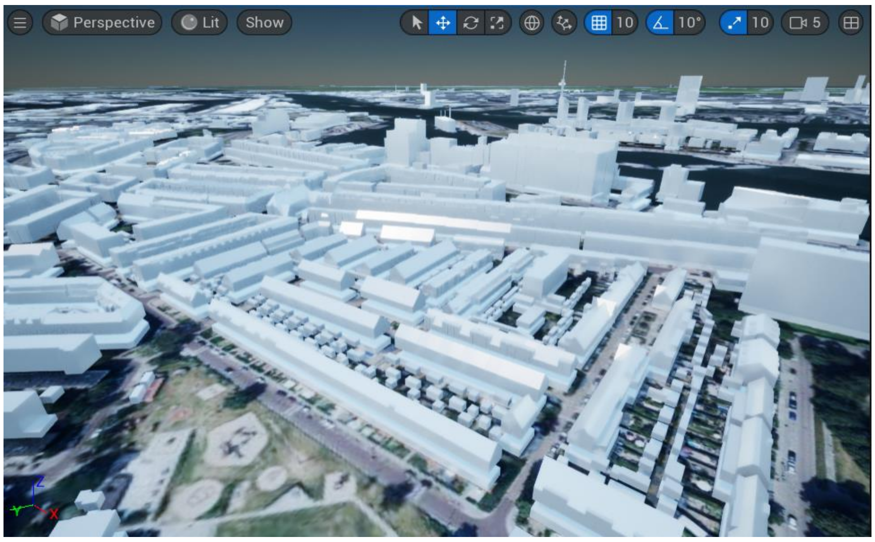


Figure 5.2.1 Imported CityJSON model with adjusted height in Unreal

In addition to importing 3D city models, Unreal also provides the option of utilising a readily available Cesium OSM building dataset. This dataset is based on OpenStreetMap and allows for the visualisation of OpenStreetMap buildings in Unreal in 3D. The buildings in this tileset are clamped to the Cesium World terrain. While this eliminates the need to upload data onto Cesium ion, it should be acknowledged that, as can be seen comparing Figure 5.2.1 & 5.2.2, the level of detail in the buildings is significantly lower compared to the CityJSON or CityGML datasets that were imported.

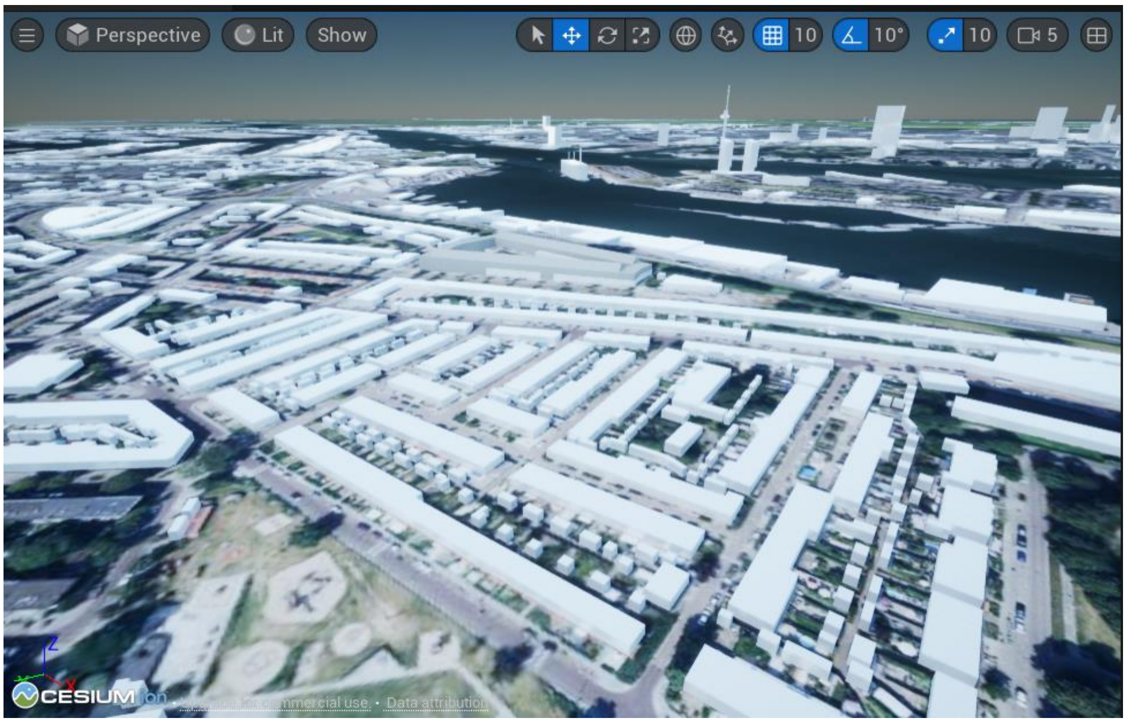


Figure 5.2.2 Cesium OSM tileset in Unreal

Another alternative for importing and integrating 3D city models into Unreal is to utilise the Google Maps API tileset. This tileset by far offers the highest level of detail and realisticity (see Figure 5.2.3). However, it is important to consider the impact on application performance. Loading the tileset requires approximately 30 seconds, which can be a noticeable delay compared to other 3D City models. Additionally, the movement within the application may become slightly less smooth due to the increased processing and rendering requirements of the highly detailed tileset. It however does produce extremely realistic looking results that also considers trees and street furniture (see Figure 5.2.4).

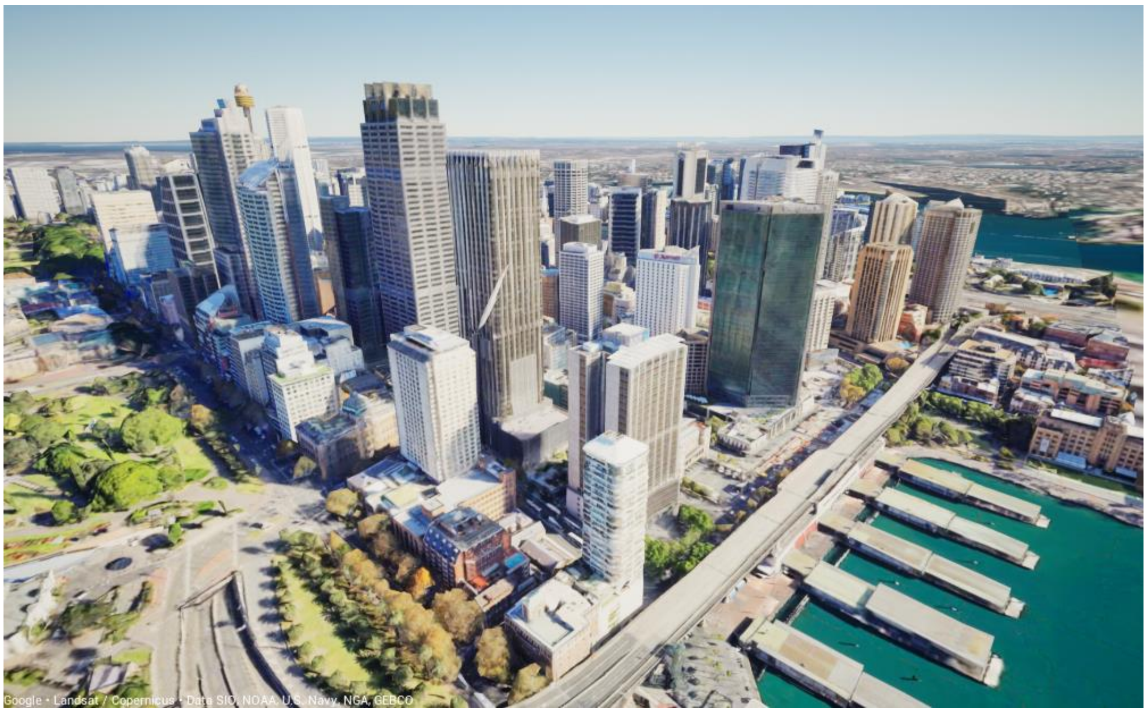

Google • Landsat / Copernicus • Data SIO, NOAA, U.S. Navy, NGA, GEBCO

Figure 5.2.3 Google Maps API tile in Unreal

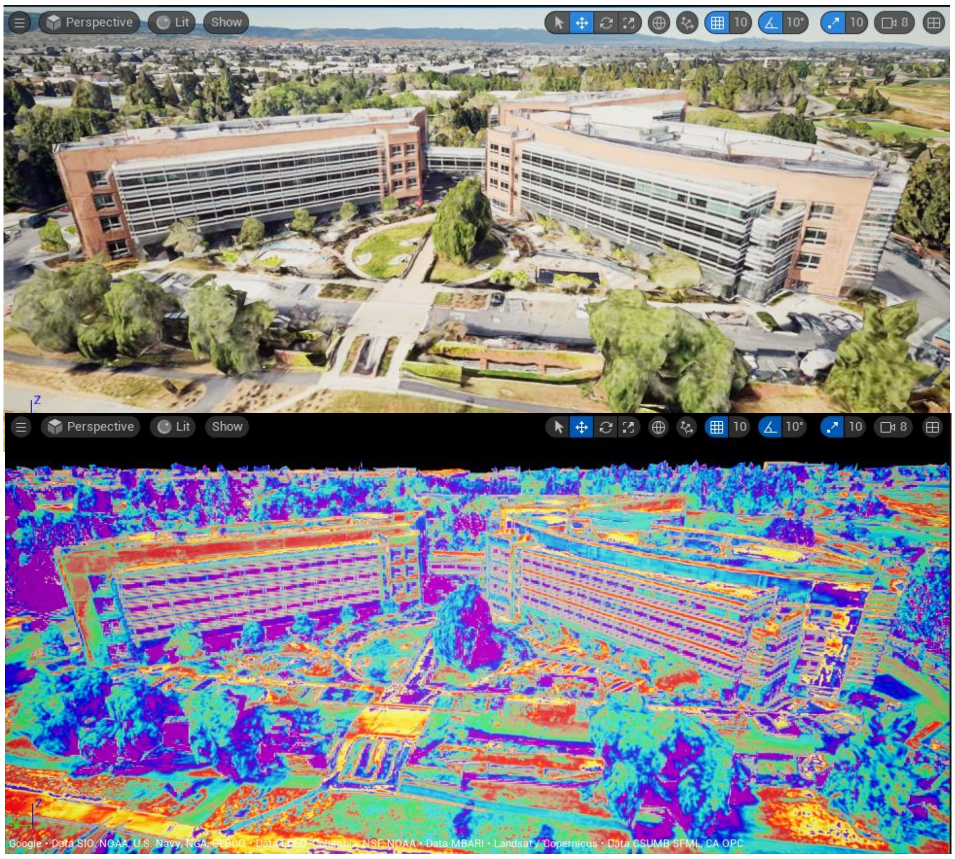

Perspective
Lit
Show
10
10°
10
8
Perspective
Lit
Show
10
10°
10
8
Google • Data SIO, NOAA, U.S. Navy, NGA, GEBCO • Data MBARI • Landsat / Copernicus • Data CSUMB SFML, CA OPC

Figure 5.2.4 Google Maps API in Unreal with false colour rendering

While Cesium ion is widely recognized as an Open Geospatial Consortium (OGC) community standard for tiling massive heterogeneous 3D content (Getz, 2021), the findings of this study shed light on the unanticipated complexity and interoperability challenges associated with importing 3D city models into the Unreal application. These findings highlight the importance of considering factors such as format compatibility, terrain clamping, and manual adjustments of z-coordinates to achieve precise positioning of the imported models. By gaining a comprehensive understanding of the strengths and limitations of different import methods, users can make well-informed decisions when integrating real-world city data into the Unreal environment.

5.3 Extracting light values from Unreal

Accessing and retrieving light values from Unreal Engine presented a mixed outcome during this research. Although the retrieval of light values was accomplished, the implementation of an export or save functionality was not included in this study. Initially, accessing light information directly from the built-in functionality of the Engine proved to be more challenging than anticipated. While Unreal Engine is an open-source game engine, not all functions are readily accessible (*Build Configurations Reference*, n.d.). The engine's design limits the ease of retrieving certain aspects of light information and intensity, as they are not explicitly made accessible. For instance, the HDR Eye Adaptation tool in Unreal Engine cannot be recreated or invoked within a project directly. To utilise such functionality, one must develop a custom engine and define the desired features. Several attempts were made to implement a custom engine with this functionality, but they were ultimately unsuccessful and prone to crashing after extended build times.

However, alternative methods for retrieving light values yielded more promising results. Unreal Engine offered multiple approaches to extract and represent light values within the application. The HDR Eye Adaptation tool, despite its inability to function at runtime, provided valuable luminance and illuminance values. On the other hand, calculating luminance values using a custom C++ actor proved to be effective, while retrieving illuminance values through this method proved to be more challenging due to its complex calculations and many parameters.

In addition, modifying the rendering of the 3D city model using a custom false-colour post-processing material proved successful. This approach allowed for the visualisation of light values and their relative distribution. By creating a custom material node, it was possible to specify the colour range based on luminance or illuminance values, resulting in visually informative renderings.

While the custom engine approach faced obstacles, the exploration of other possibilities for extracting light values proved fruitful. Nonetheless it is important to note the potential of creating a custom engine for achieving complete customizability, although further investigation and experimentation with alternative methods are recommended before pursuing this path.

Overall, the combination of different techniques showcased the potential of Unreal Engine as a platform for accessing and analysing light values, with the HDR Eye adaptation, custom C++ actor and post-processing material offering viable solutions for calculating and visualising luminance and illuminance values. Ultimately the HDR eye-adaptation tool was used to extract lighting values for benchmarking, as it was easier to navigate through the environment in development mode, than using the custom C++ actor at runtime. The post-processing material was primarily designed for visual representation rather than precise extraction of light values, making it less suitable for benchmarking.

5.4 Benchmarking results

This section presents the benchmarking results of Honeybee and Unreal, for daylight analysis with a 3D city model. The objective was to evaluate the performance and effectiveness of these tools in handling complex geometry and large-scale analysis tasks.
Honeybee encountered challenges with the complex geometries of 3D city models. Both CityJSON and CityGML formats are not supported by Rhino. As a workaround, the OBJ version of the 3DBAG tile was used, which could be loaded into Rhino. However, compatibility issues arose when trying to integrate this geometry with Honeybee due to specific requirements regarding closed and adjacent volumes.

Honeybee proved to be incredibly sensitive to complex geometries. Honeybee Rooms, which form the Honeybee Model, require geometry to be adjacent and form closed volumes. 3D city models, however, consist of numerous non-closed and non-adjacent volumes. Additionally, the presence of height differences between the ground floor and surrounding structures posed difficulties in creating a Honeybee model.
Moreover, the computational time required by Honeybee scripts was substantial. To run the full script with a 3D City model took on average 93 minutes. Adding new geometry to the calculation, also took 93 minutes, as it required the entire script to run from the start. This prolonged computation time resulted in delays and inefficiencies during the analysis process, particularly when dealing with large and complex geometries.
Considering these limitations and complexities, it became evident that Honeybee may not be the most suitable tool for handling complex geometric scenarios and large-scale daylight analysis tasks. The challenges related to geometry requirements, slow computational speed, and limitations in handling complex geometries hindered the workflow productivity and efficiency.

In contrast to Honeybee, the Unreal tool demonstrated efficiency in handling complex geometry and conducting real-time daylight analysis. The Unreal application exhibited no significant issues related to computational time or geometry complexity. The tool seamlessly handled the complex geometries present in the 3D city model, regardless of their complexity or scale. The computational time for various operations, including runtime for new angle calculations, complete script and adding new geometry, was consistently low, with an average of 0.6 milliseconds. This real-time processing allowed for efficient analysis iterations and real-time adjustments to the model.

The benchmarking results between Unreal and the Radiance Honeybee method revealed valuable insights into the accuracy and reliability of the Unreal tool for daylight analysis (see Table 5.4). Comparing the illuminance values obtained from both methods, the Unreal application demonstrated a Mean Absolute Error of 9.78% when compared to Radiance Honeybee. This suggests the reliability and accuracy of Unreal for daylight analysis tasks.

The benchmarking study highlighted the potential of Unreal as a daylight analysis tool. Unreal's real-time capabilities, efficient computation, and seamless handling of complex geometries make it highly effective and efficient for such scenarios. As depicted in Figure 5.4.2, 5.4.3 and 5.4.4, the average differences between the values of Unreal Engine and Honeybee are relatively similar. Figure 5.4.4 shows the normalised difference between the measurements. The Root Mean Square Error (RMSE) absolute value was found to be 3576.63, indicating the overall magnitude of the differences between Unreal and Honeybee measurements. Furthermore, the relative RMSE was determined to be 13.23%, standing for the percentage of the RMSE relative to the average measured values.

The Mean Bias Error (MBE) absolute value was calculated to be -5.24 lux, showing a slight bias towards lower values in Unreal compared to Honeybee. The relative MBE was determined to be -0.02%, suggesting a negligible relative bias between the two methods.

It is worth noting that all these values fall below the prediction error of commonly used daylight simulation models when validated against real-life measurements, as depicted in Figure 5.4.1. These findings further reinforce the accuracy and reliability of Unreal Engine simulating daylight information.

As Honeybee, and the models in Figure 5.4.1, are models and not measurements done in the physical world, it is impossible to determine which model is closer to reality. For a more in-depth comparison, conducting real-life measurements and comparing those values with Unreal and Honeybee outputs would provide deeper insights into their performance and alignment with actual daylight conditions.

	Runtime new angle	Runtime Complete script	Runtime adding new geometry	Illuminance values
Unreal	0.6 msec	0.6 msec	0.6 msec	9.78% difference with Radiance
Radiance Honeybee	645sec	93 min	93 min	-

Table 5.4 Benchmarking results

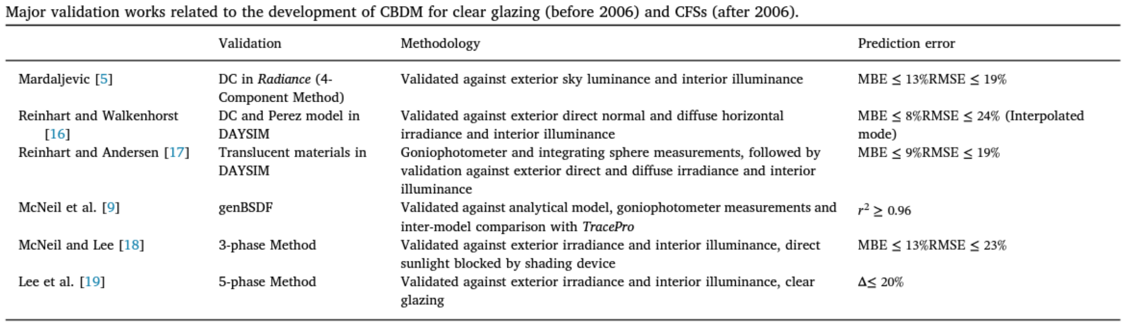


Table 1
Major validation works related to the development of CBDM for clear glazing (before 2006) and CFSs (after 2006).

	Validation	Methodology	Prediction error
Mardaljevic [5]	DC in *Radiance* (4-Component Method)	Validated against exterior sky luminance and interior illuminance	MBE ≤ 13%RMSE ≤ 19%
Reinhart and Walkenhorst [16]	DC and Perez model in DAYSIM	Validated against exterior direct normal and diffuse horizontal irradiance and interior illuminance	MBE ≤ 8%RMSE ≤ 24% (Interpolated mode)
Reinhart and Andersen [17]	Translucent materials in DAYSIM	Goniophotometer and integrating sphere measurements, followed by validation against exterior direct and diffuse irradiance and interior illuminance	MBE ≤ 9%RMSE ≤ 19%
McNeil et al. [9]	genBSDF	Validated against analytical model, goniophotometer measurements and inter-model comparison with *TracePro*	$r^2 \geq 0.96$
McNeil and Lee [18]	3-phase Method	Validated against exterior irradiance and interior illuminance, direct sunlight blocked by shading device	MBE ≤ 13%RMSE ≤ 23%
Lee et al. [19]	5-phase Method	Validated against exterior irradiance and interior illuminance, clear glazing	Δ≤ 20%

Figure 5.4.1 Major validation works related to the development of CBDM (Brembilla & Mardaljevic, 2019).

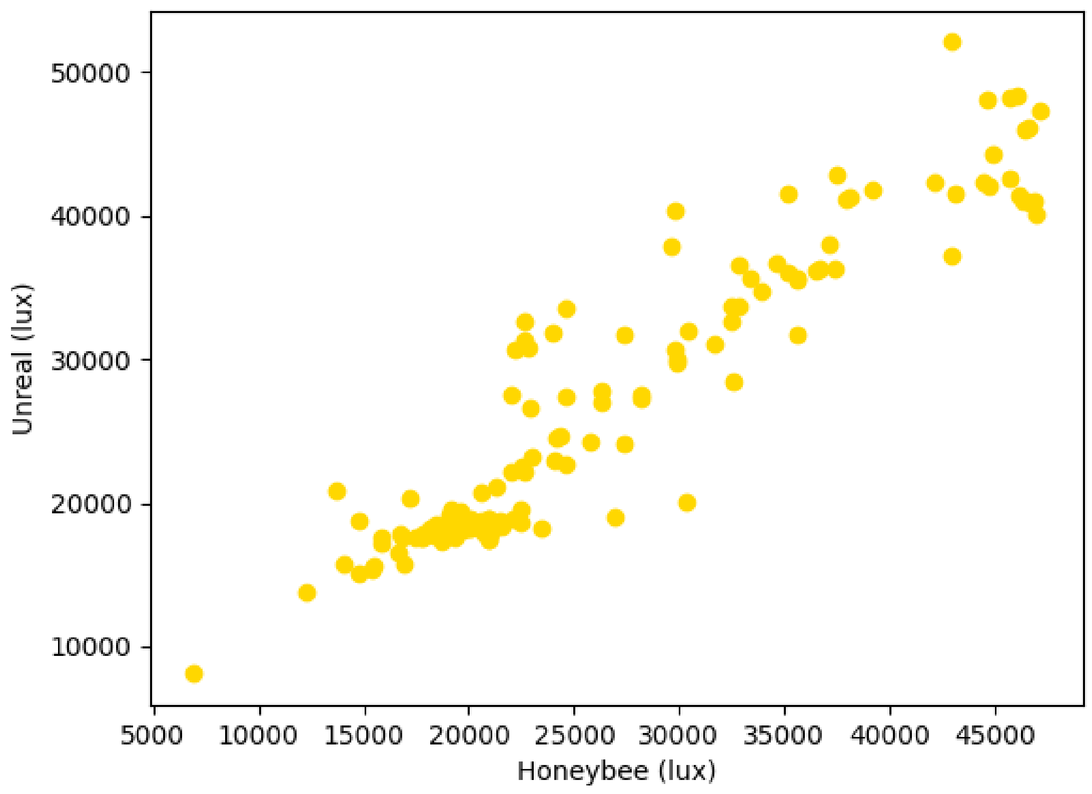


Figure 5.4.2 Measured Values in Honeybee and Unreal

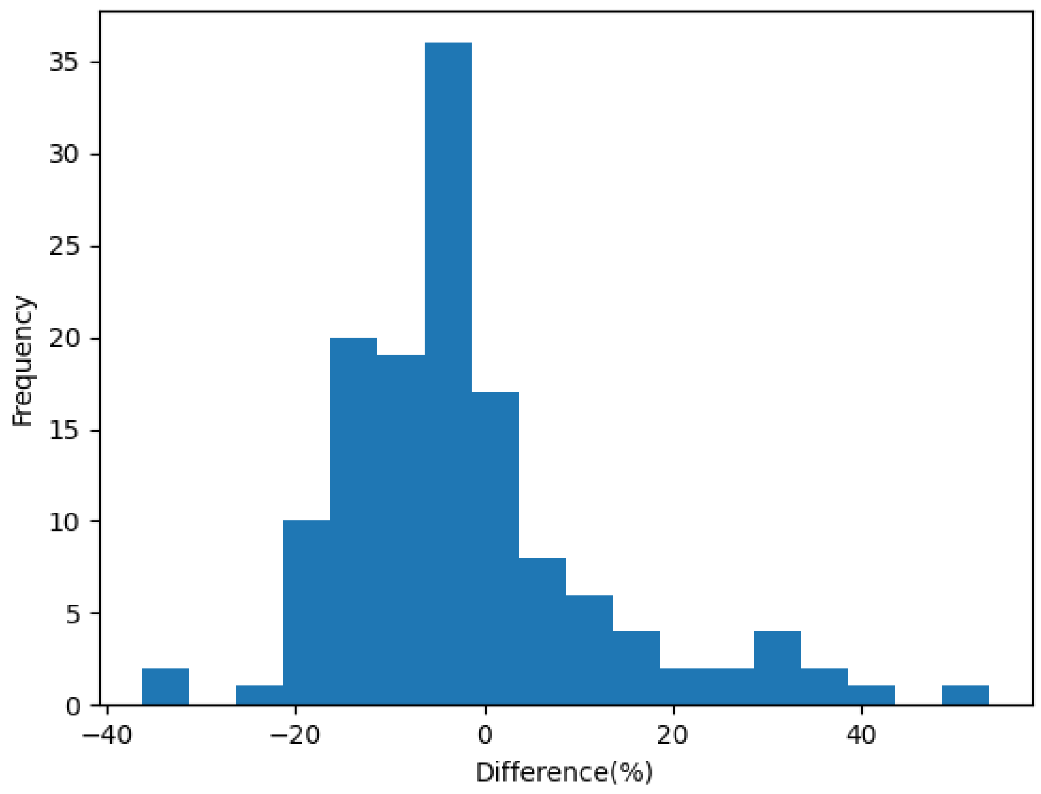


Figure 5.4.3 Histogram of differences between Honeybee & Unreal

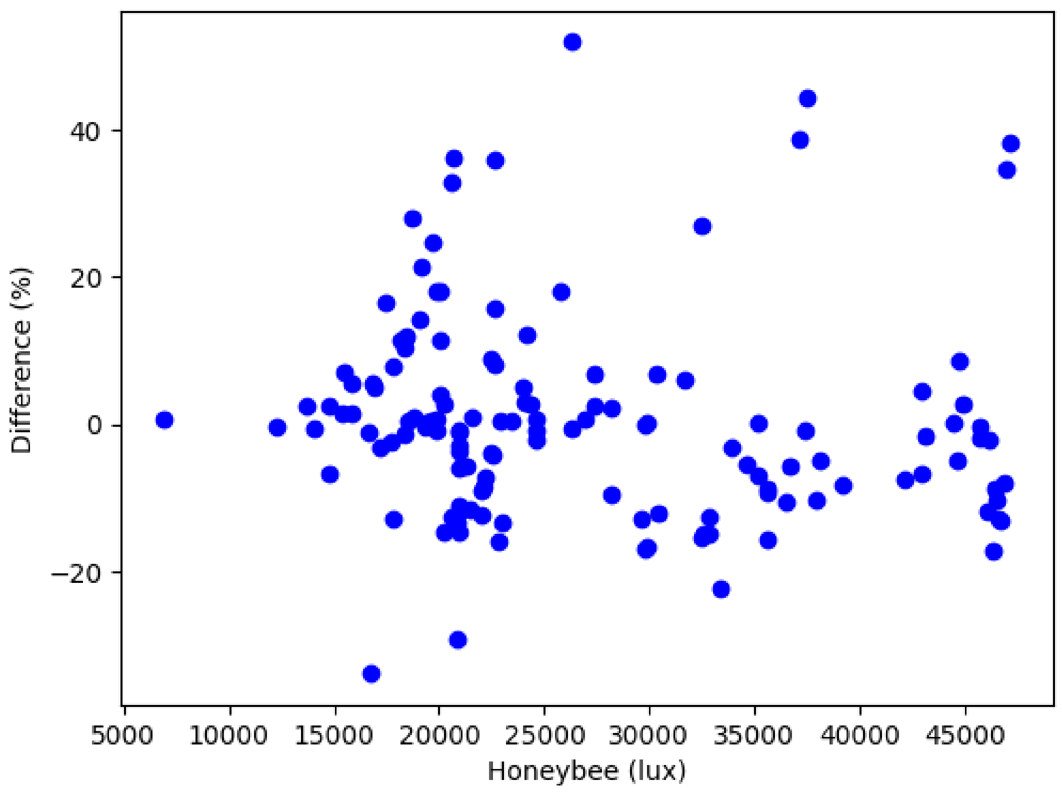


Figure 5.4.4 Relative differences between Honeybee & Unreal

To analyze the distribution and variability of the measurements, a box plot representation (Figure 5.4.5) was used. The majority of the measurements fell within the expected range, as indicated by the box representing the interquartile range (IQR). To identify potential outliers, the 1.5IQR rule was applied, classifying data points below Q1 - 1.5 * IQR or above Q3 + 1.5 * IQR as outliers. Based on this rule, points 12, 25, 41, 43, 66, 80, 96, 110, 112, 134, and 135 were identified as outliers. These outliers significantly deviated from the other data points, suggesting possible irregularities or measurement errors.

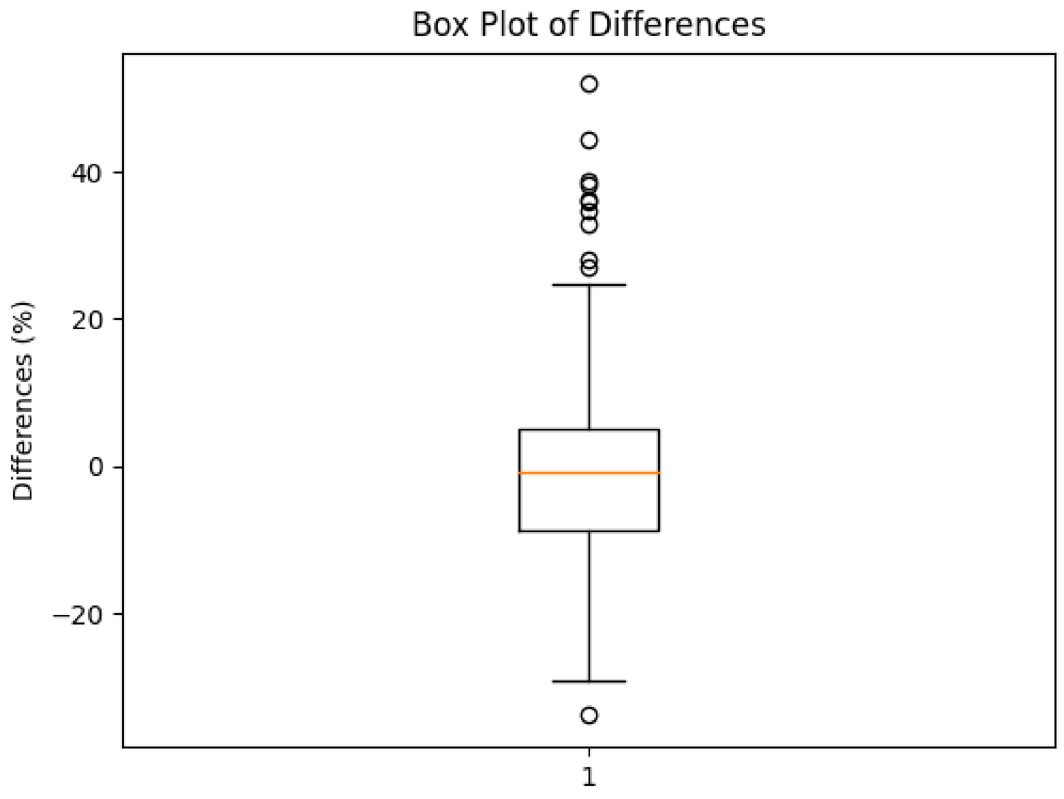


Figure 5.4.5 Boxplot of differences between Honeybee & Unreal

Further analysis of the benchmarking results reveals the following notable observations and conclusions:

1) Honeybee exhibited more consistent values, with fewer variations among nearby measurements. On the other hand, Unreal displayed more granular and subtle differences, likely due to its higher resolution for daylight calculations.
2) Unlit surfaces in Unreal generally had lower illuminance values, indicating that the indirect light component was slightly too low.
3) In contrast, lit surfaces in Unreal showed significant positive differences compared to Honeybee, suggesting variations likely caused by slightly different sun positions in Unreal and Honeybee.
4) Point 25, which had complex surrounding geometries, was expected to be an outlier due to potential refraction issues. The fact that this point is an outlier is not surprising.
5) However, the presence of outliers in other points, such as 12, 41, 43, 66, 80, 96, 110, 112, 134, and 135, representing simple scenarios with minimal interference or refraction, was highly unexpected. During the benchmarking stage, it was observed that the built-in Unreal SunSky and Cesium SunSky displayed different sun positions with the same input settings of date and time. This discrepancy in sun position between Unreal and Honeybee may have caused these outliers in the data. The most reasonable explanation for these outliers is that the sun position in Unreal and Honeybee is not exactly

the same, despite using the same date, time, and hour. Figure 5.4.6 showcases some of these outlier points, where surfaces that appeared shaded in Honeybee were fully exposed to the sun in Unreal, with unobstructed sunlight from surrounding buildings. This discrepancy in sun position between the two simulations could account for the disparities observed in the measurements.

6) The optimal balance in Unreal was achieved by using the Unreal SunSky with the skylight set to 15 and the sunlight to 25000 lux. In Honeybee, the light values for direct sunlight were 44800 lux and for indirect light were 1200 lux. Comparing the light settings between Unreal and Honeybee is challenging because the skylight in Unreal does not have a unit, making it difficult to directly compare values. Further research is necessary to establish a relationship between the daylight units in Unreal and Honeybee.

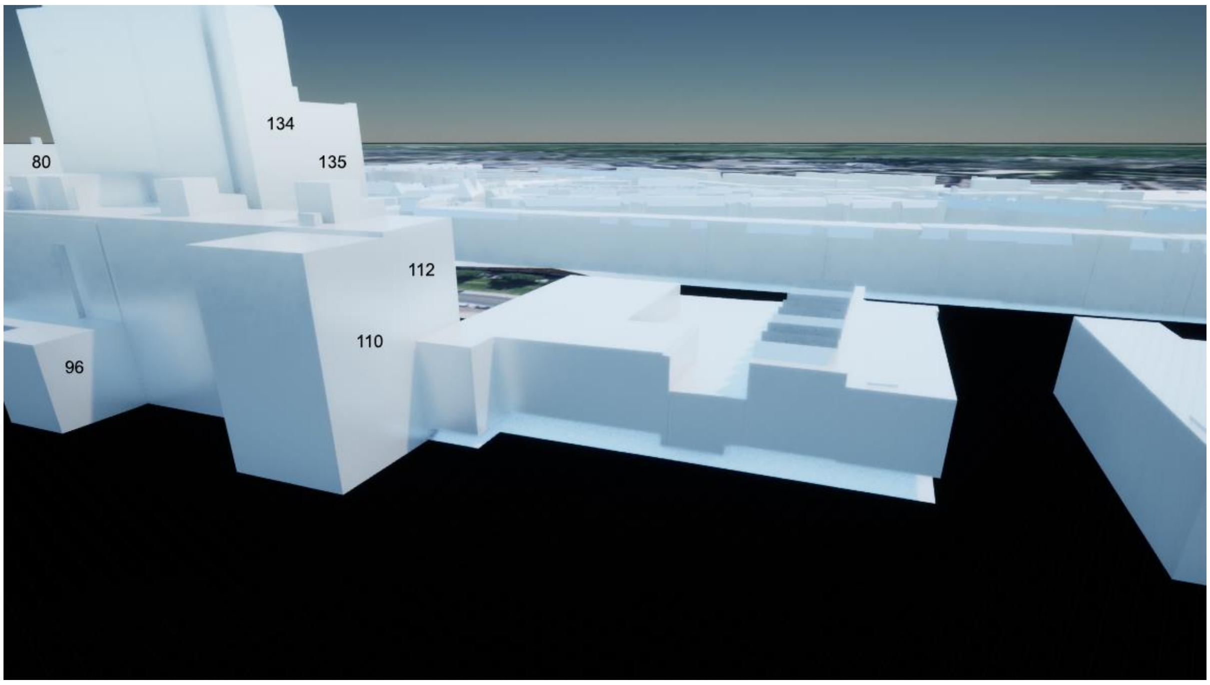


Figure 5.4.6 Outlier points of lit areas in Unreal that are shadowed in Honeybee

These outliers emphasize the need for further investigation and understanding of the underlying factors that influenced these deviating measurements. logical initial step would be to explore the apparent disparities in sun position between Unreal and Honeybee, as well as establish a relationship between the daylight units in Unreal and Honeybee. Examining these aspects, will give valuable insights into the underlying causes of the variations and refine and optimize the daylight simulation process in Unreal.

5.5 Interactive User Functionality & Recommended workflow

Providing users with design capabilities is essential for the practical usability of the application to assess daylight conditions during different design stages. The focus of this thesis is to assess whether the Unreal environment offers a suitable platform for daylight analysis in architectural and urban development. This underlines the importance of user interaction and design possibilities within the platform.

User interaction in the application encompasses various functionalities and considerations. One key aspect is the ability to measure illuminance and luminance and visualize the distribution of light in real-time. Users can import Cesium ion tilesets, add building blocks, edit geometry, and adjust the sun position dynamically at runtime.

While building blocks and geometry editing are suitable for initial massing studies, they may lack the desired granularity in later design stages that require more refined and detailed designs. Comparatively, the Honeybee environment in Rhino offers enhanced design capabilities and ease of use for creating refined designs. Consequently, architects and urban developers are recommended to design using their preferred design tools such as Rhino or Revit and import the designs into Unreal using Cesium ion. Alternatively, users familiar with Unreal or seeking additional functionality may choose to use the application in development mode. The built-in functionality available in development mode provides more extensive options for measuring light values and designing compared to the implemented functionality at runtime.

The application is currently not recommended for detailed daylight analysis requiring the extraction and export of light values for further processing. Adding this functionality proved to be too complex within the timeframe of this thesis. Currently, for such use cases, it is recommended to use Honeybee or alternative tools that support the required functionality.
In collaborative urban development or architectural teams, it might be advantageous to involve a GIS specialist who can prepare the desired area of interest by selecting the appropriate 3D city model and potentially creating pre-determined building blocks for various types of structures, such as family houses, apartment blocks, and office buildings. The application can then be used by architects and urban developers to explore and test different design scenarios in collaborative design sessions within this predefined environment.

Additionally, it is important to note that the application did not include user research and feedback, which would have supplied valuable insights for validating and further developing the user interaction and design capabilities of the Unreal environment. User research is essential for understanding the needs, preferences, and challenges faced by architects and urban developers when using the application. By incorporating user feedback, it would be possible to refine the functionalities, address usability issues, and enhance the overall user experience.

Overall, the user interaction and recommended workflow within the Unreal environment for daylight analysis in architectural and urban development play a crucial role in its practical applicability. By providing real-time measurement and visualization of illuminance and luminance, along with the flexibility to import designs from preferred tools and collaborate within a predefined environment, the Unreal application offers valuable capabilities for early design stages and collaborative sessions. The real-time visualization of the light is arguably the most valuable aspect of the application during the design phase of urban and architectural development. However, for more refined and detailed designs, architects and urban developers are advised to leverage the enhanced design capabilities of tools like Rhino or Revit in combination with Cesium ion for importing designs into Unreal. Additionally, while the application may not currently support detailed daylight analysis with export functionality, alternative tools such as Honeybee can fulfil those requirements. By understanding the strengths and limitations of the Unreal environment and integrating it into the design workflow appropriately, users can effectively use the application for informed decision-making in architectural and urban development projects.

6 Conclusions and future research

In this final section, the research provides a concise summary of the conclusions, limitations, future work, and reflection. It highlights the potential of the Unreal Engine as a real-time daylight analysis tool in architectural and urban development, while acknowledging the encountered limitations.

6.1 Conclusions

This research aimed to explore the potential of the Unreal Engine as a platform for real-time daylight analysis in architectural and urban development. By investigating the effective integration of 3D city models into an Unreal application, the study has demonstrated the viability and versatility of utilising the Unreal Engine for daylight analysis for architectural and urban development. The conclusions section will begin by answering the sub-questions before answering the main research question.

1. How can 3D city models be effectively integrated into an Unreal application
The results discussed in Chapter 5.2 has shown that integrating 3D city models into the Unreal application is possible through various methods. Importing CityGML or CityJSON datasets using Cesium ion proved to be a viable option, although challenges related to differences in z-coordinates and terrain clamping need to be addressed. The use of Cesium OSM buildings and Google Maps API tilesets offers alternatives with varying levels of detail and performance considerations. Ultimately the user will have to decide the 3D city model based on their needs, but the application offers a variety of viable options.

2. How can the light values extracted from Unreal be compared and validated against industry-standard light models such as Radiance?

The process of accessing and retrieving light values from Unreal Engine posed challenges (see Chapter 5.3), but various methods were employed to overcome them. Utilizing the HDR Eye Adaptation tool, custom C++ actors, and post-processing materials enabled the extraction and visualization of light values. However, the implementation of an export or save functionality was realized in this thesis. To ensure accuracy and reliability, further refinement and experimentation are recommended. Despite the obstacles encountered, the combination of these techniques demonstrated the potential of Unreal Engine for accessing and analyzing daylight values. The HDR Eye Adaptation tool proved useful in extracting daylight values, while the post-processing materials served as a powerful visualization tool to depict the distribution of light.

3. **To what extent do the light values simulated with Unreal Engine accurately represent real-world lighting conditions?**

The research findings indicate that the lighting values extracted from Unreal align closely with the expected trends based on the inverse square law (Chapter 5.1). The comparison and validation of daylight propagation in urban environments extracted from Unreal compared against industry-standard light model Radiance, have highlighted both the strengths and limitations of the Unreal Engine in achieving accurate daylight simulations in urban environments. In the benchmarking experiments conducted in urban environments, an average deviation of 9.78% was observed (see Chapter 5.4). However, the results did have some more extreme outliers. While the extracted values have shown promise, further refinement and validation are necessary to ensure the realism and reliability of lighting representations in Unreal, under different circumstances and in different locations. The benchmarking demonstrates the engine's capability to replicate real-world lighting behaviours, although minor deviations and limitations exist.

4. In what ways can urban or architectural developers use an Unreal application to test and develop different design scenarios?

Using Unreal to test and develop different design scenarios and the resulting impact on daylight, offers designers both advantages and disadvantages. The real-time nature of the engine enables instant feedback and visualisation of design choices, allowing for iterative exploration and informed decision-making. This iterative approach enhances the design process by facilitating rapid prototyping, spatial analysis, and visualisation of daylight conditions, thereby empowering designers to create more sustainable and efficient built environments. However, it is important to recognize the limitations of the application, particularly regarding the level of detail required for refined and detailed designs. Chapter 5.5 makes recommendations for architects and urban developers. architects and urban developers are recommended to use their preferred design tools, such as Rhino or Revit, and import their designs into Unreal using Cesium ion for enhanced design capabilities. For detailed daylight analysis with export functionality, it is recommended to use Honeybee or alternatives, as this functionality is not integrated in the application yet. By understanding the strengths and limitations of the Unreal environment and integrating it strategically into the design workflow, users can effectively incorporate the application in their workflow for informed decision-making in architectural and urban development projects.

To what extent is the Unreal Engine suitable to scale-up physically accurate daylight simulation tools?

Regarding the main question of the extent to which the Unreal Engine is suitable for scaling up physically accurate daylight simulation tools, it can be concluded that while the engine showcases significant potential, further advancements are necessary. The Unreal Engine provides a robust foundation for real-time visualisation and analysis, but additional research and development are needed to refine its accuracy, expand its capabilities, and address the limitations identified throughout this study. This includes improving the comparability and validation of lighting values, enhancing the realism of lighting simulations and refining user functionality. The Unreal Engine offers a promising platform for real-time daylight analysis in architectural and urban development. Most notably, the real-time aspect of Unreal compared to the slow and cumbersome processes in Honeybee, along with the ease of importing large datasets, highlights the potential of the Unreal Engine in architectural and urban development. While challenges and opportunities for improvement exist, the findings of this study contribute to the growing body of knowledge surrounding real-time daylight simulations and pave the way for further advancements in this field.

6.2 Limitations

The implementation of the research methodology encountered several limitations, which are essential to acknowledge for a comprehensive understanding of the study's scope and potential implications. Firstly, the comparison of light values between Honeybee and Unreal posed challenges due to the difficulty in extracting and aligning the values. This necessitated manual selection of points for comparison, which limited the amount of sample points for the benchmarking and could introduce information bias and potential inaccuracies.

Furthermore, the functionality offered by the Unreal application in comparison to the custom-built application may not fully cater to the diverse needs of all users. The disparity in available features and tools raises the possibility that certain users may require additional functionalities beyond what the current implementation offers.

Another limitation is the absence of user testing. The application's usability and user experience have not been systematically evaluated through user feedback and testing sessions. This leaves room for uncertainties regarding the user-friendliness and effectiveness of the tool in real-world scenarios.

Moreover, the validation of illuminance values was limited to a specific time and location, disregarding the potential variations in solar settings across different times and locations. The reliance on a single instance for validation restricts the generalizability of the findings and highlights the need for further validation across various scenarios.

Additionally, it is important to note that Honeybee provides a broader range of functionalities beyond point-in-time illuminance analysis, such as sun-hour analysis and annual radiance simulations. While the current implementation shows promise, these extended capabilities have not been fully explored or tested in this study. Although this research showed promising results in daylight measurements, and the other functionality of Honeybee in essence is also based on raytracing, in which Unreal showed great efficiency, they will have to be implemented or built, so pose risks. Nonetheless, by recognizing these considerations and proactively addressing them, it is possible to enhance the overall usability and effectiveness of the Unreal application for a wider range of user requirements.

By acknowledging these limitations, it becomes evident that further research and refinement are necessary to address these constraints and enhance the overall robustness and applicability of the developed methodology and tool.

6.3 Future work

The findings and limitations of this study open several avenues for future research and development in the field of real-time daylight analysis using the Unreal Engine. Building upon the insights gained from this research, the following areas hold potential for further investigation:

1. **Enhanced comparison and extraction methods:** Future work should focus on developing more robust and automated methods for comparing light values between Honeybee and Unreal. This could involve exploring advanced algorithms or data processing techniques to streamline the extraction process and reduce manual intervention.
2. **User testing and feedback:** Conducting thorough user testing sessions is crucial to validate the usability and effectiveness of the developed application. User feedback and observations can provide valuable insights for improving the user interface, functionality, and overall user experience. Iterative testing and refinement cycles should be employed to ensure that the tool meets the needs of its intended users.
3. **Validation across diverse scenarios:** To enhance the reliability and generalizability of the findings, further validation of illuminance values should be performed across different times and locations. Incorporating a broader range of solar settings and considering varying geographical conditions would contribute to a more comprehensive assessment of the tool's accuracy and reliability.
4. **Extended functionality and integration:** Expanding the functionality of the Unreal application to include additional daylight analysis features, such as sun-hour analysis and annual radiance simulations, would provide users with a more comprehensive toolkit for evaluating daylight conditions. Moreover, exploring integration possibilities with other software platforms, such as GIS or BIM tools, could further enhance the application's capabilities and its integration into existing design workflows.
5. **Connection to weather data in Unreal:** Integrating weather data into Unreal to adjust the SunSky settings accordingly based on date and time would provide a more realistic representation of outdoor lighting conditions. This integration would further enhance the accuracy and automation of the daylight analysis within the application.
6. **Standardized conversion for 3DBAG:** There are potential areas for future development and improvement of import 3DBAG. The focus area is addressing the challenges associated with differences in z-coordinates and terrain clamping when importing CityJSON datasets using Cesium ion. To streamline the integration process, it is recommended to explore the development of standardized conversion methods or improve the compatibility between the formats.
7. **Sun position:** Given the observed disparities in sun position between Unreal and Honeybee, it is recommended to conduct further investigations to accurately align the sun positions. Addressing this misalignment will contribute to more reliable and accurate daylight analysis in the platforms, enabling more accurate comparisons and facilitating better decision-making in architectural and urban design processes.
8. **Integrating functionality of existing geospatial urban development tools:** Integrating the functionality of tools such as ArcGIS Urban (Urban Planning & Design-Smart City Planning | ArcGIS Urban, n.d.) and 3D Cityplanner (Strategis Groep bv, n.d.), or incorporating the real-time daylight analysis capabilities of LuminaCity into these applications, presents a valuable opportunity for expanding functionality. ArcGIS Urban and 3D Cityplanner offer a wide range of features, including 3D modeling, spatial analysis, and visualization. By merging their strengths, we can harness their data management and analysis capabilities while overcoming limitations in daylight analysis. This integration allows users to conduct urban development analysis, considering factors such as land use, transportation, infrastructure, and real-time

daylight conditions. Ultimately, aiding in informed decision-making and supports scientifical, sustainable urban planning.

6.4 Reflection

Throughout the course of this thesis, I have learned a great deal about daylight analysis and the Unreal Engine. Although I had some experience in daylight analysis during my Bachelor of Architecture and the Built Environment, as well as experience in C++ during my Master of Geomatics, I quickly found myself on a steep learning curve. This required a lot of dedication and perseverance, as well as strong project and time management skills to stay on track and ensure I was meeting my goals. The educational foundation provided by the Geomatics field equipped me with the necessary knowledge and abilities to tackle these challenges effectively, both in terms of planning and programming.

One of the biggest challenges I faced during this project was exporting the light values from the Unreal Engine. This complex and time-consuming process required manual comparison of light values rather than automated techniques, resulting in lower accuracy and reproducibility. Despite the difficulties, however, I remained committed to the project and focused on finding innovative solutions to overcome any obstacles that arose.

Despite the challenges, I found the experience of developing a real-time daylight analysis tool using the Unreal Engine to be incredibly exciting and rewarding. The ability to create highly realistic virtual environments that can be explored in real time opens new possibilities for the architectural and urban development industries. By providing a tool that is both easy to use and highly accurate, it can help professionals in these fields make more informed decisions about their designs and improve the overall quality of the built environment.

Furthermore, I believe that the bigger goal of this thesis, which was to prove the concept of a geospatial urban development platform with built-in geospatial analysis, has tremendous potential to transform the way we approach architecture and urban development. By incorporating geospatial analysis into the design process, we can ensure that new developments are aligned with the natural features of the environment and consider factors such as sun exposure and shadow analysis. This can ultimately lead to more sustainable and environmentally friendly development practices, which are critical for our planet's future.

Looking back on this experience, I can see how much I have grown both personally and professionally. I have developed a deeper understanding of the importance of time management, project management, and the value of developing a minimum viable product (MVP). In previous projects, I frequently found myself getting sidetracked by the multitude of intriguing elements that captured my attention, diverting my focus from the primary objective. If I were to undertake a similar project in the future, I would immediately refine my scope and focus on specific features, as well as allocate more time to learning the Unreal Engine and other tools that would be required for the project.

In conclusion, this thesis has been an incredible learning experience for me, and I am incredibly grateful for the opportunity to have worked on such an exciting and innovative project with my supervisors. I believe that the real-time daylight analysis tool that I have developed has the potential to revolutionise the way we approach architectural and urban development, and I am excited to see where this technology will take us in the future.

Bibliography

3D BAG Viewer. (n.d.). https://3dbag.nl/en/download?tid=3312

3D Stadsmodel Den Haag 2022 CityGML - Dataplatform. (n.d.). https://ckan.dataplatform.nl/dataset/3d-stadsmodel-den-haag-2021-citygml

Akenine-Möller, T., Haines, E., & Hoffman, N. (2019). *Real-time rendering*. Crc Press.

AnneCorning, & Systems, R. V. (2022, October 14). The Language of Light | Radiant Vision Systems. *Radiant Vision Systems*. https://www.radiantvisionsystems.com/blog/language-light

Antonanzas-Torres, F., Urraca, R., Polo, J., Perpiñán-Lamigueiro, O., & Escobar, R. (2019). Clear sky solar irradiance models: A review of seventy models. *Renewable and Sustainable Energy Reviews*, *107*, 374-387.

Biljecki, F., Ledoux, H., Du, X., Stoter, J., Soon, K. H., & Khoo, V. H. S. (2016). THE MOST COMMON GEOMETRIC AND SEMANTIC ERRORS IN CITYGML DATASETS. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, *4*.

Billen, R., Cutting-Decelle, A. F., Métral, C., Falquet, G., Zlatanova, S., & Marina, O. (2015). Challenges of semantic 3D city models: a contribution of the COST research action TU0801. *International Journal of 3-D Information Modelling (IJ3DIM)*, *4*(2), 68-76.

Blankert, A. (2021). All Dutch 3D buildings in 1.9 GB. *Geodan Research*. https://research.geodan.nl/all-dutch-3d-buildings-in-1-9gb/

Brembilla, E., & Mardaljevic, J. (2019). Climate-Based Daylight Modelling for compliance verification: Benchmarking multiple state-of-the-art methods. *Building and Environment*, *158*, 151-164.

Build Configurations Reference. (n.d.). https://docs.unrealengine.com/5.2/en-US/build-configurations-reference-for-unreal-engine/

Buyuksaliha, G., Bayburta, S., Baskaracaa, A. P., Karimb, H., & Rahmanb, A. A. (2017). Calculating solar energy potential of buildings and visualisation within unity 3D game engine. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *42*(4/W5).

C++ and Blueprints. (n.d.). https://docs.unrealengine.com/5.1/en-US/cpp-and-blueprints-example/

Cesium for Unreal. (2015, June 30). Cesium. https://cesium.com/platform/cesium-for-unreal/

Cesium ion. (n.d.). Cesium. https://cesium.com/platform/cesium-ion/

Chen, Y., Cui, Z., & Hao, L. (2019). Virtual reality in lighting research: Comparing physical and virtual lighting environments. *Lighting Research & Technology*, *51*(6), 820-837.

Citgml4j. (n.d.). *GitHub - citygml4j/citygml-tools: Collection of tools for processing CityGML files*. GitHub. https://github.com/citygml4j/citygml-tools

CityJSON. (n.d.). CityJSON. https://www.cityjson.org/

Converting to/from CityGML files. (n.d.). CityJSON. https://www.cityjson.org/tutorials/conversion/

Cozzi, P. (2021a, February 28). *Introducing 3D Tiles*. Cesium. https://cesium.com/blog/2015/08/10/introducing-3d-tiles/

Cozzi, P. (2021b, March 16). *Announcing Our Collaboration with Epic Games to Create Cesium for Unreal Engine*. Cesium. https://cesium.com/blog/2020/06/04/cesium-for-unreal-engine/

Cozzi, P. (2023, May 15). *Cesium Partners with Google Maps Platform to Render Its New Photorealistic 3D Tiles*. Cesium. https://cesium.com/blog/2023/05/10/cesium-partners-with-google-maps-platform/

Edler, D., Keil, J., & Dickmann, F. (2020). From Na Pali to Earth—An 'unreal'engine for modern geodata?. In *Modern approaches to the visualisation of landscapes* (pp. 279-291). Springer VS, Wiesbaden.

Edler, D., Keil, J., Wiedenlübbert, T., Sossna, M., Kühne, O., & Dickmann, F. (2019). Immersive VR experience of redeveloped post-industrial sites: The example of "Zeche Holland" in Bochum-Wattenscheid. *KN J. Cartogr. Geogr. Inf*, *69*, 267-284.

gendaylit.pdf — Radsite. (n.d.). https://www.radiance-online.org/learning/documentation/manual-pages/pdfs/gendaylit.pdf/view

Getz, G. (2021, March 15). *3D Tiles Now an OGC Community Standard for Streaming Massive 3D Geospatial Content*. Cesium. https://cesium.com/blog/2019/02/05/3d-tiles-ogc-community-standard/

Glassner, A. S. (Ed.). (1989). *An introduction to ray tracing*. Morgan Kaufmann.

GPUOpen by AMD. (2022, November 17). *Unreal Engine Performance Guide - AMD GPUOpen*. AMD GPUOpen. https://gpuopen.com/unreal-engine-performance-guide/

Hapala, M., & Havran, V. (2011, March). Kd-tree traversal algorithms for ray tracing. In *Computer Graphics Forum* (Vol. 30, No. 1, pp. 199-213). Oxford, UK: Blackwell Publishing Ltd.

Hegazy, M., Ichiriyama, K., Yasufuku, K., & Abe, H. (2020). Visualizing user perception of daylighting: a comparison between VR and reality. *BuildSIM-Nordic 2020*, *13*.

Hegazy, M., Ichiriyama, K., Yasufuku, K., & Abe, H. (2021). Validating Game Engines as a Quantitative Daylighting Simulation Tool. 10.52842/conf.caadria.2021.2.285.

Inverse Square Law. (n.d.). http://hyperphysics.phy-astr.gsu.edu/hbase/Forces/isq.html

Jakica, N. (2018). State-of-the-art review of solar design tools and methods for assessing daylighting and solar potential for building-integrated photovoltaics. *Renewable and Sustainable Energy Reviews*, *81*, 1296-1328.

Keil, J., Edler, D., Schmitt, T., & Dickmann, F. (2021). Creating immersive virtual environments based on open geospatial data and game engines. *KN-Journal of Cartography and Geographic Information*, *71*(1), 53-65.

Kersten, T. P., Deggim, S., Tschirschwitz, F., Lindstaedt, M., & Hinrichsen, N. (2018). Segeberg 1600—Eine Stadtrekonstruktion in virtual reality. *KN-Journal of Cartography and Geographic Information*, *68*(4), 183-191.

Ladybug Tools | Home Page. (n.d.). https://www.ladybug.tools/

Ledoux, H., Arroyo Ohori, K., Kumar, K., Dukai, B., Labetski, A., Vitalis, S., & Stoter, J. (2019). CityJSON: A compact and easy-to-use encoding of the CityGML data model. *Open Geospatial Data, Software and Standards, 4(1)*, 4. https://doi.org/10.1186/s40965-019-0066-x

Lee, Joanna. *Learning unreal engine game development*. Packt Publishing Ltd, 2016.

Liang, Z., Zhou, K., & Gao, K. (2019). Development of virtual reality serious game for underground rock-related hazards safety training. *IEEE access*, *7*, 118639-118649.

Luminous intensity & Photometry | auersignal.com. (n.d.). https://www.auersignal.com/en/technical-information/visual-signalling-equipment/luminous-intensity/#What_is_the_density_of_luminance_(cd/m%C2%B2)

Mardaljevic, J. (1999). *Daylight simulation: validation, sky models and daylight coefficients* (Doctoral dissertation, De Montfort University).

Mat, R. C., Shariff, A. R. M., Zulkifli, A. N., Rahim, M. S. M., & Mahayudin, M. H. (2014, June). Using game engine for 3D terrain visualisation of GIS data: A review. In *IOP Conference Series: Earth and Environmental Science* (Vol. 20, No. 1, p. 012037). IOP Publishing.

Mooney, C. Z. (1997). *Monte carlo simulation* (No. 116). Sage.

Morrissey, J., Moore, T., & Horne, R. E. (2011). Affordable passive solar design in a temperate climate: An experiment in residential building orientation. *Renewable Energy*, 36(2), 568-577.

Perez, R., Seals, R., & Michalsky, J. (1993). All-weather model for sky luminance distribution—preliminary configuration and validation. *Solar energy*, *50*(3), 235-245.

Physical Lighting Units. (n.d.). https://docs.unrealengine.com/5.1/en-US/using-physical-lighting-units-in-unreal-engine/

Post Process Materials. (n.d.). Unreal Engine Documentation. https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/PostProcessEffects/PostProcessMaterials/

Radiance — Radsite. (n.d.). https://www.radiance-online.org/

Robert McNeel & Associates. (n.d.). *Rhinoceros 3D*. www.rhino3d.com. https://www.rhino3d.com/

Schmitt, A., Müller, H., & Leister, W. (1988). Ray tracing algorithms—theory and practice. In Theoretical Foundations of Computer Graphics and CAD (pp. 997-1030). Springer Berlin Heidelberg.

Shader Development. (n.d.). https://docs.unrealengine.com/5.0/en-US/shader-development-in-unreal-engine/

Singh, J. M., & Narayanan, P. J. (2009). Real-time ray tracing of implicit surfaces on the GPU. *IEEE transactions on visualization and computer graphics*, *16*(2), 261-272.

State of the Art in 3D City Modelling. (2022, July 13). GIM International. https://www.gim-international.com/content/article/state-of-the-art-in-3d-city-modelling-2

Steinhage, V., Behley, J., Meisel, S., & Cremers, A. B. (2010, March). Automated updating and maintenance of 3D city models. In *ISPRS joint workshop on" Core Spatial Databases-Updating, Maintenance and Services-from Theory to Practice* (pp. 1-6).

Strategis Groep bv. (n.d.). *3D Cityplanner, the future of urban development*. https://3dcityplanner.com/en/

The Home of Location Technology Innovation and Collaboration | OGC. (n.d.). https://www.ogc.org/

Threaded Rendering. (n.d.). https://docs.unrealengine.com/5.0/en-US/threaded-rendering-in-unreal-engine/

TU Delft. (n.d.). Open datasets. https://3d.bk.tudelft.nl/opendata/

Unreal Engine | The most powerful real-time 3D creation tool. (n.d.). Unreal Engine. https://www.unrealengine.com/en-US/

Urban Planning & Design-Smart City Planning | ArcGIS Urban. (n.d.). Esri. https://www.esri.com/en-us/arcgis/products/arcgis-urban/overview

Visual Studio 2022 | Download for free. (2023, May 23). Visual Studio. https://visualstudio.microsoft.com/vs/

Wang, Z. H. (2014). Monte Carlo simulations of radiative heat exchange in a street canyon with trees. *Solar Energy*, *110*, 704-713.

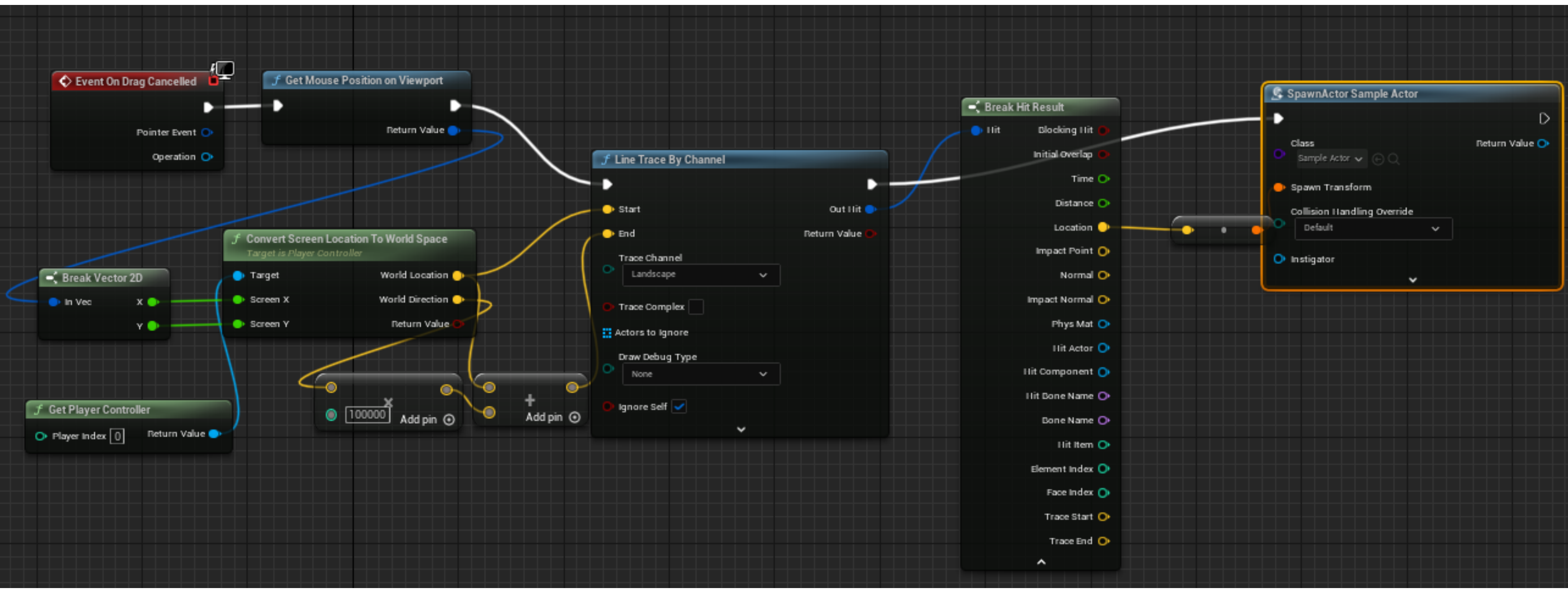
What is real-time ray tracing? (n.d.). Unreal Engine. https://www.unrealengine.com/en-US/explainers/ray-tracing/what-is-real-time-ray-tracing

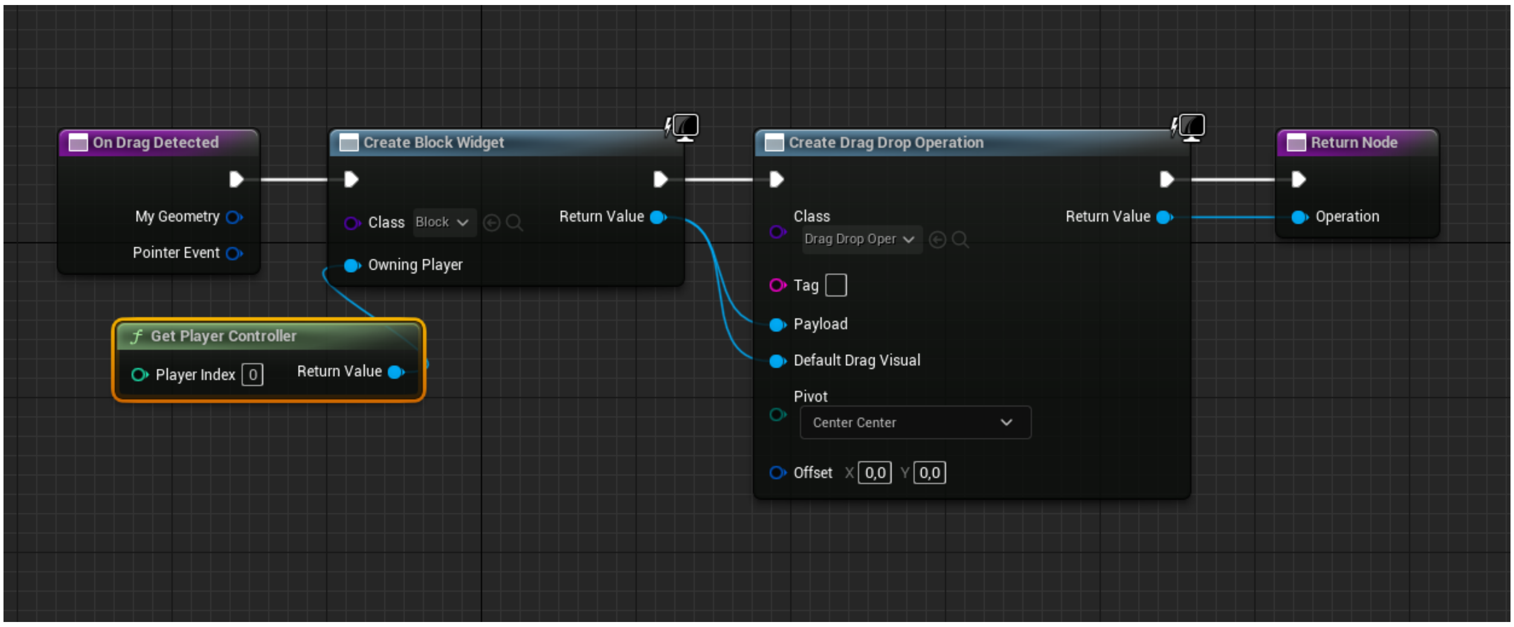
Wij zijn tech | FME. (n.d.). FME. https://www.fme.nl/

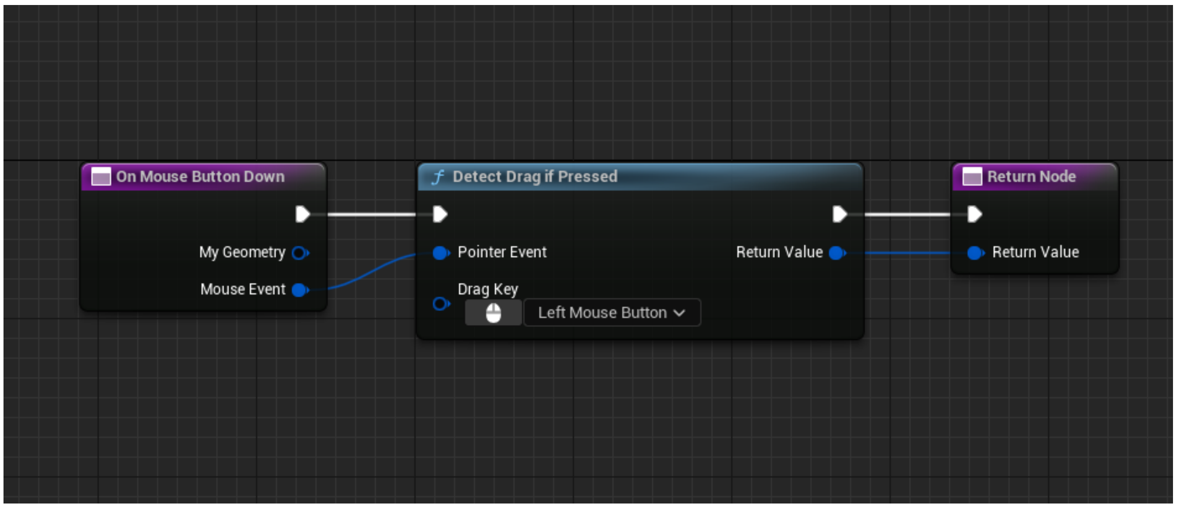
Wikipedia contributors. (2023). Wavefront .obj file. *Wikipedia*. https://en.wikipedia.org/wiki/Wavefront_.obj_file

Xyahh. (n.d.). *GitHub - xyahh/UE4RuntimeTransformer: A Runtime Gizmo Transformer tool helps you translate/rotate/scale objects in runtime!* Easily provide editing tools to your final product! GitHub. https://github.com/xyahh/UE4RuntimeTransformer

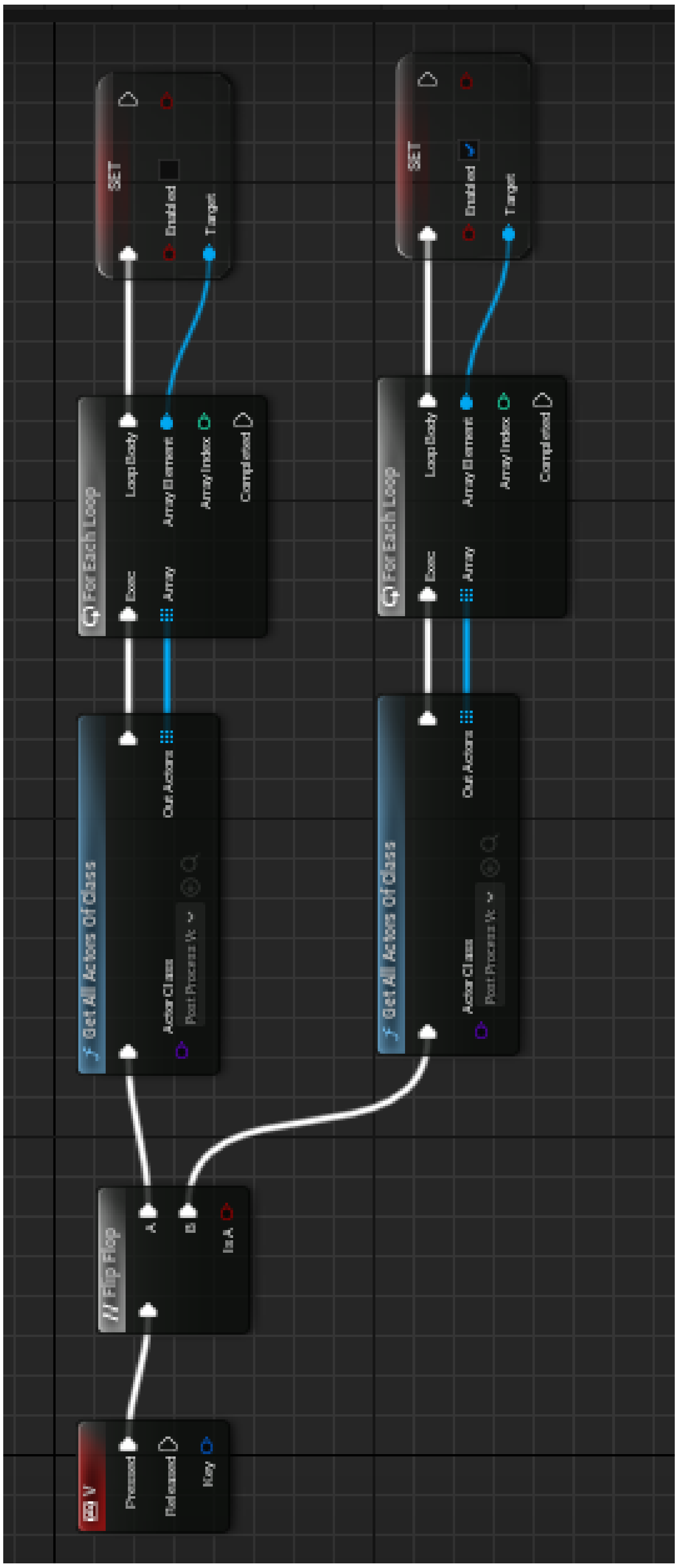
Appendix A Adding Geometry



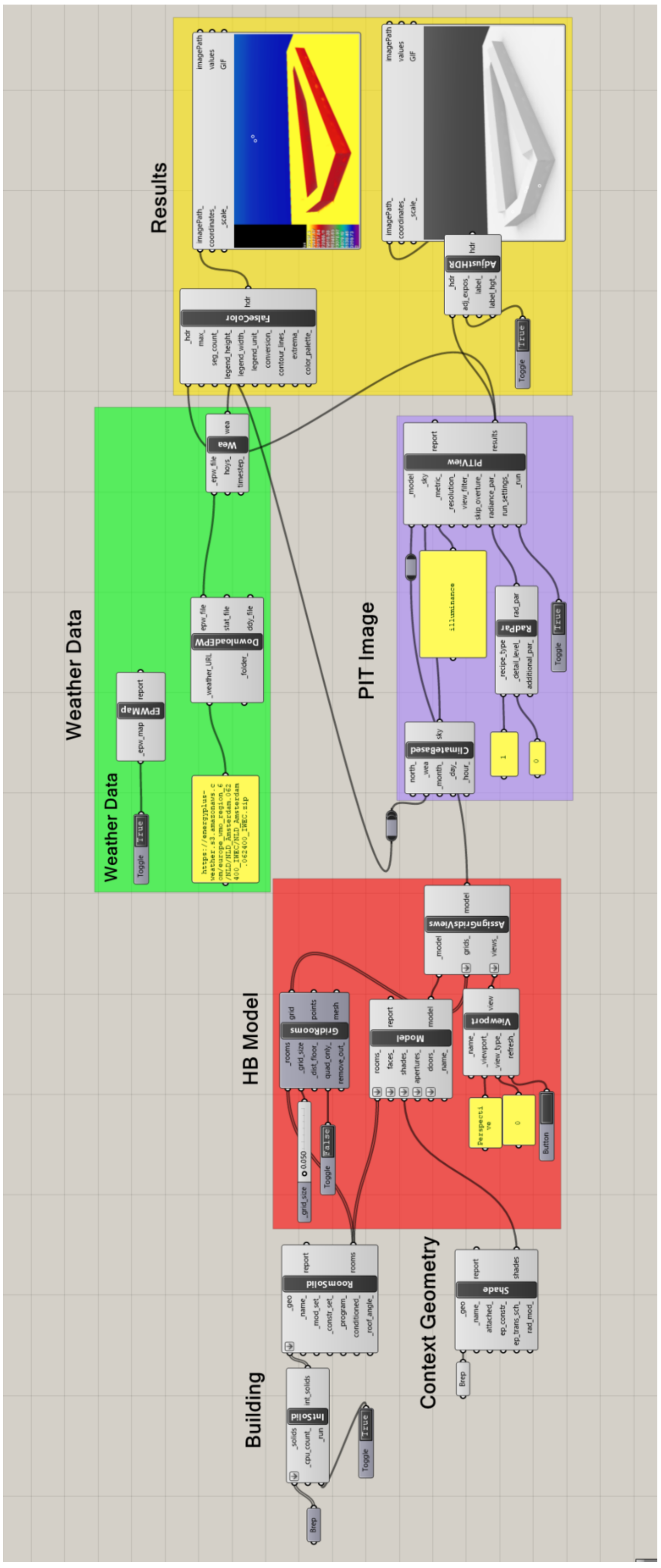



On Mouse Button Down
My Geometry
Mouse Event
Detect Drag if Pressed
Pointer Event
Drag Key
Left Mouse Button
Return Value
Return Node
Return Value

Appendix B change Render at Runtime



Appendix C Grasshopper Script



Appendix D Benchmark Results

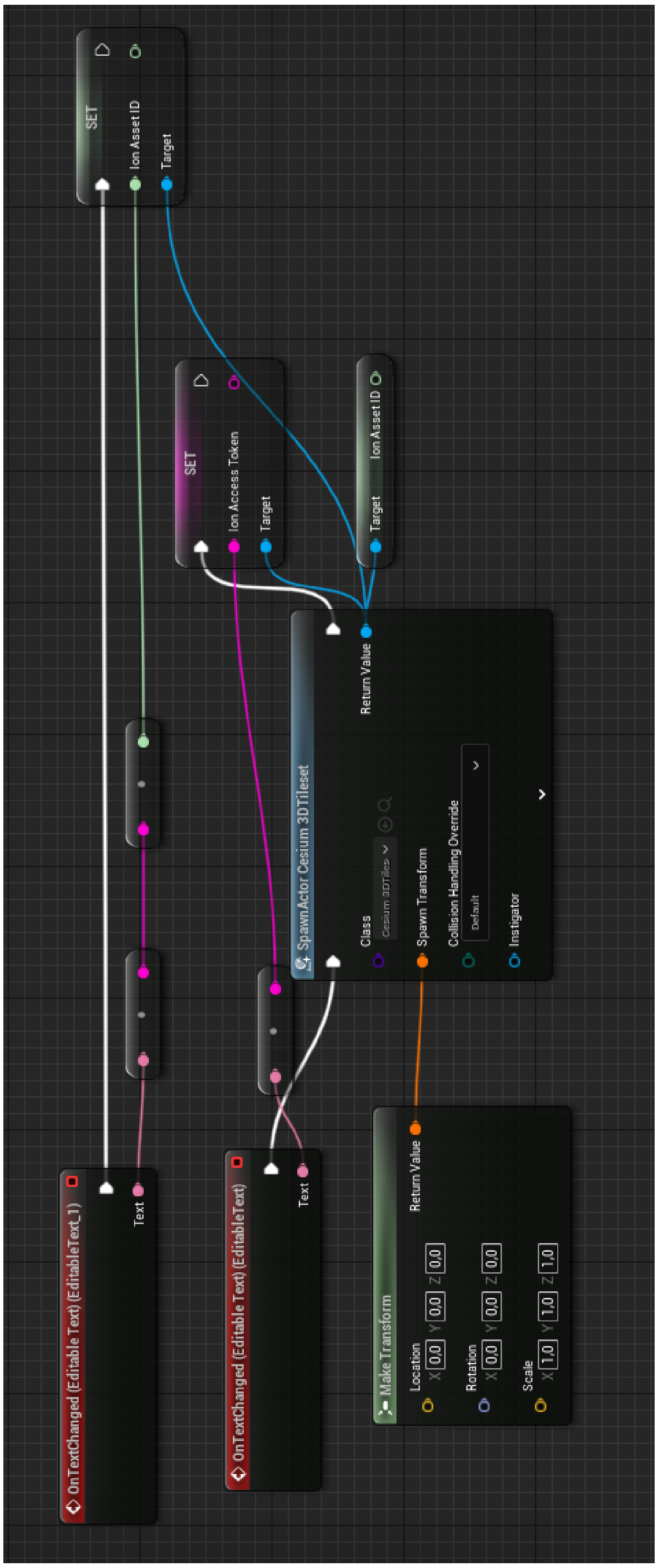
Point	Honeybee	Unreal	Difference
1	19027	19164	0,72
2	22589	22521	-0,3
3	32838	33639	2,44
4	16641	16556	-0,51
5	16922	15786	-6,71
6	19135	19605	2,46
7	24344	24677	1,37
8	16711	17900	7,12
9	24165	24509	1,42
10	26296	27750	5,53
11	46611	46142	-1,01
12	30286	20041	-33,83
13	45752	48251	5,46
14	46038	48393	5,12
15	28156	27308	-3,01
16	22876	26674	16,6
17	28156	27481	-2,4
18	32578	28398	-12,83
19	44606	48099	7,83
20	24612	27432	11,46
21	17842	17618	-1,26
22	15861	17535	10,55
23	12293	13764	11,97
24	22017	22145	0,58
25	29588	37851	27,93
26	23019	23262	1,06
27	37482	42839	14,29
28	42906	52108	21,45
29	35567	35472	-0,27
30	47184	47308	0,26
31	29857	29996	0,47
32	22017	27478	24,8
33	21301	21129	-0,81
34	32434	32696	0,81
35	6859	8107	18,2
36	32434	33726	3,98
37	35137	41512	18,14
38	32846	36580	11,37
39	26295	27007	2,71
40	47000	40092	-14,7
41	24003	31898	32,89
42	46900	41002	-12,58
43	24648	33543	36,09

44	42900	37234	-13,21
45	26921	19058	-29,21
46	46500	41012	-11,8
47	43100	41512	-3,68
48	25758	24259	-5,82
49	35638	31711	-11,02
50	21157	18052	-14,68
51	36498	36219	-0,76
52	36677	36272	-1,1
53	37357	36301	-2,83
54	19869	18723	-5,77
55	20048	18935	-5,55
56	46325	41037	-11,42
57	17422	17590	0,96
58	46665	40933	-12,28
59	20083	18290	-8,93
60	19707	18049	-8,41
61	20209	18743	-7,25
62	18830	18106	-3,84
63	15832	17240	8,89
64	18562	17817	-4,01
65	38144	41275	8,21
66	29768	40433	35,83
67	27387	31708	15,78
68	20960	17658	-15,75
69	18436	18508	0,39
70	20710	17939	-13,38
71	17814	17896	0,46
72	30412	31956	5,08
73	29767	30635	2,92
74	14069	15797	12,28
75	33868	34763	2,64
76	18311	17917	-2,15
77	15454	15578	0,8
78	46450	46019	-0,93
79	17185	20293	18,09
80	13691	20800	51,92
81	17688	17587	-0,57
82	20585	20751	0,81
83	37142	38046	2,43
84	39147	41782	6,73
85	20083	18197	-9,39
86	14762	15078	2,14
87	20960	18296	-12,71
88	20960	17402	-16,98
89	15390	15370	-0,13

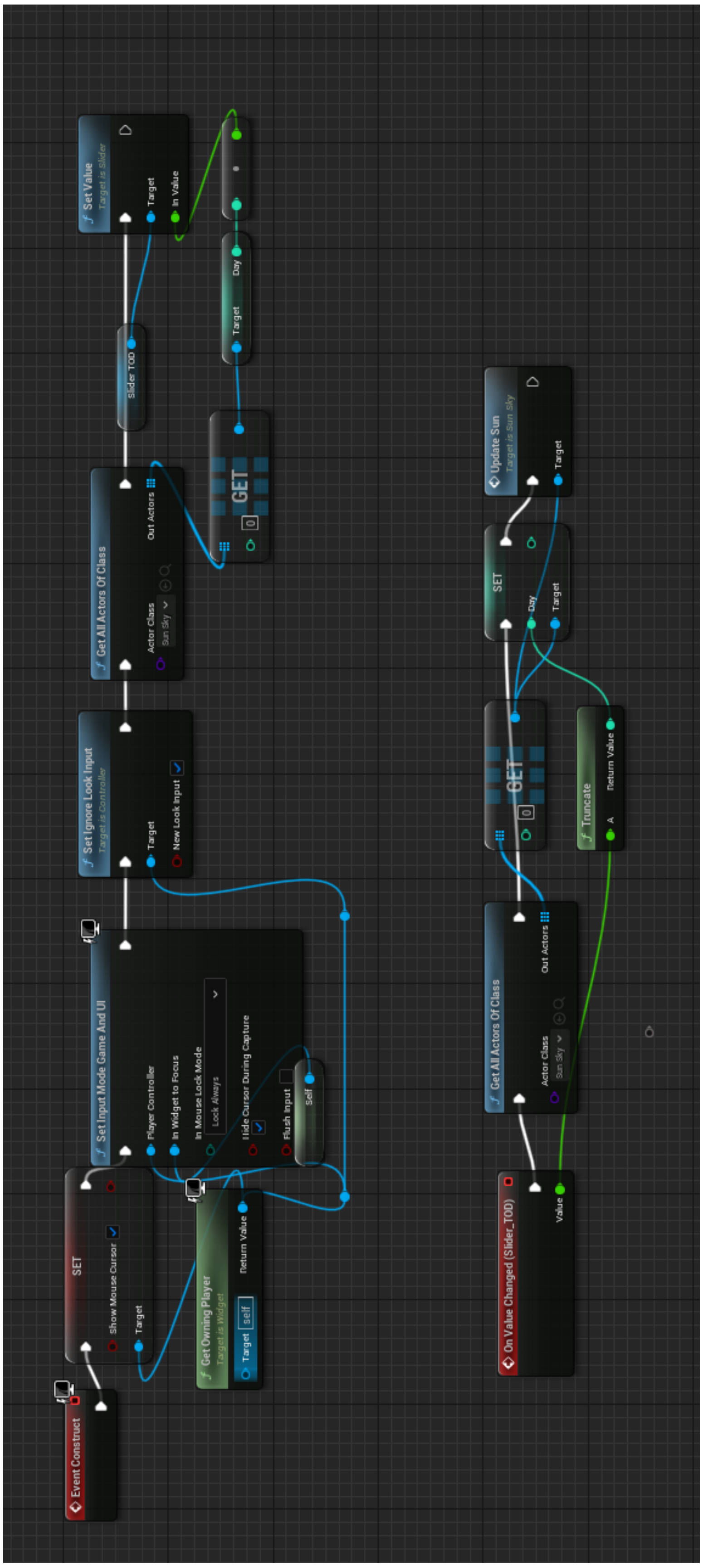
90	35621	35653	0,09
91	20960	17482	-16,59
92	33365	35688	6,96
93	20835	18328	-12,03
94	34618	36720	6,07
95	20960	17765	-15,24
96	14762	18744	26,97
97	21587	18399	-14,77
98	22139	18840	-14,9
99	20835	18239	-12,46
100	23484	18286	-22,13
101	18311	17734	-3,15
102	19332	18274	-5,47
103	18687	17387	-6,96
104	18186	18246	0,33
105	19332	17649	-8,71
106	20209	18357	-9,16
107	20960	17708	-15,52
108	46199	41363	-10,47
109	44696	42122	-5,76
110	22607	31381	38,81
111	19582	19438	-0,74
112	22607	32631	44,34
113	20960	18837	-10,13
114	44445	42327	-4,77
115	19833	18213	-8,17
116	19833	18352	-7,47
117	45698	42593	-6,79
118	16808	17583	4,61
119	44946	44256	-1,54
120	42172	42292	0,28
121	24111	22936	-4,87
122	37894	41181	8,67
123	35119	36064	2,69
124	29893	29813	-0,27
125	31665	31053	-1,93
126	27387	24144	-11,84
127	22607	22146	-2,04
128	22464	18602	-17,19
129	20585	18789	-8,72
130	20960	18836	-10,13
131	21462	18741	-12,68
132	22464	19520	-13,11
133	24612	22645	-7,99
134	22858	30779	34,65
135	22213	30693	38,18

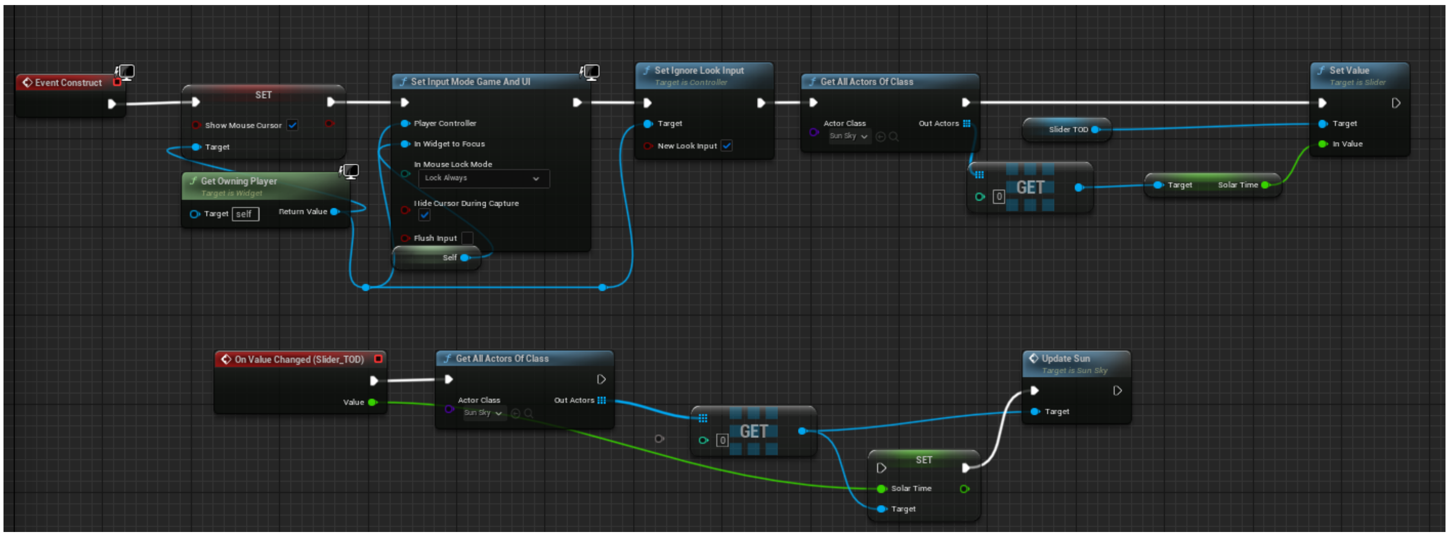
		Average	9,789407407

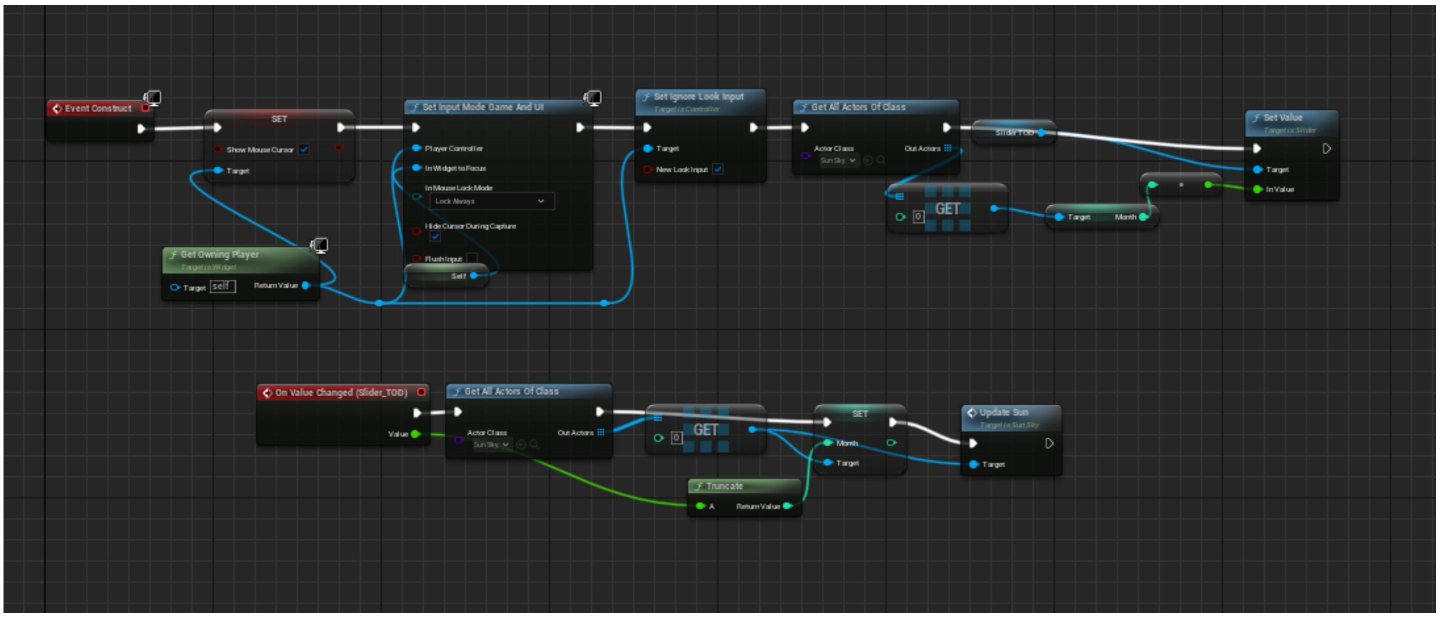
Appendix E Loading Cesium ion tiles at Runtime


SET
Ion Asset ID
Target
SET
Ion Access Token
Target
Ion Asset ID
Target
SpawnActor Cesium 3DTileset
Return Value
Class
Cesium 3DTiles
Spawn Transform
Collision Handling Override
Default
Instigator
Make Transform
Return Value
Location X 0,0 Y 0,0 Z 0,0
Rotation X 0,0 Y 0,0 Z 0,0
Scale X 1,0 Y 1,0 Z 1,0
OnTextChanged (Editable Text) (EditableText_1)
Text
OnTextChanged (Editable Text) (EditableText)
Text

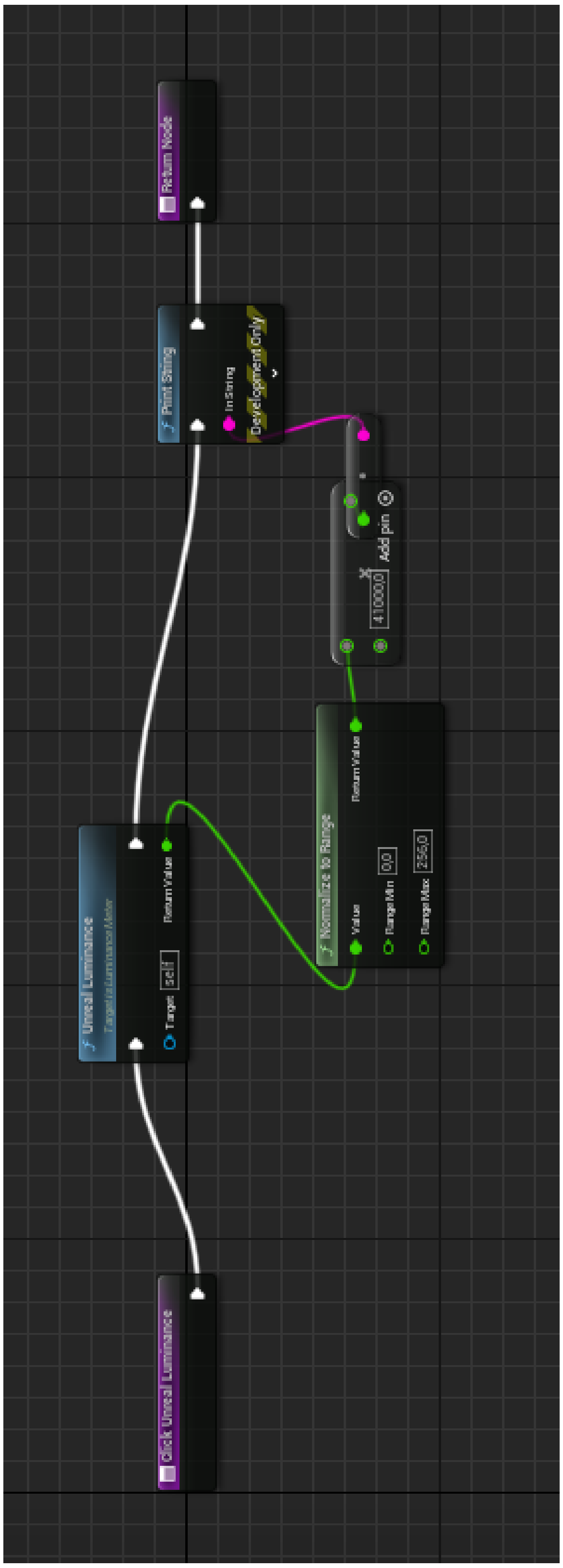
Appendix F Change Sun position, date & time



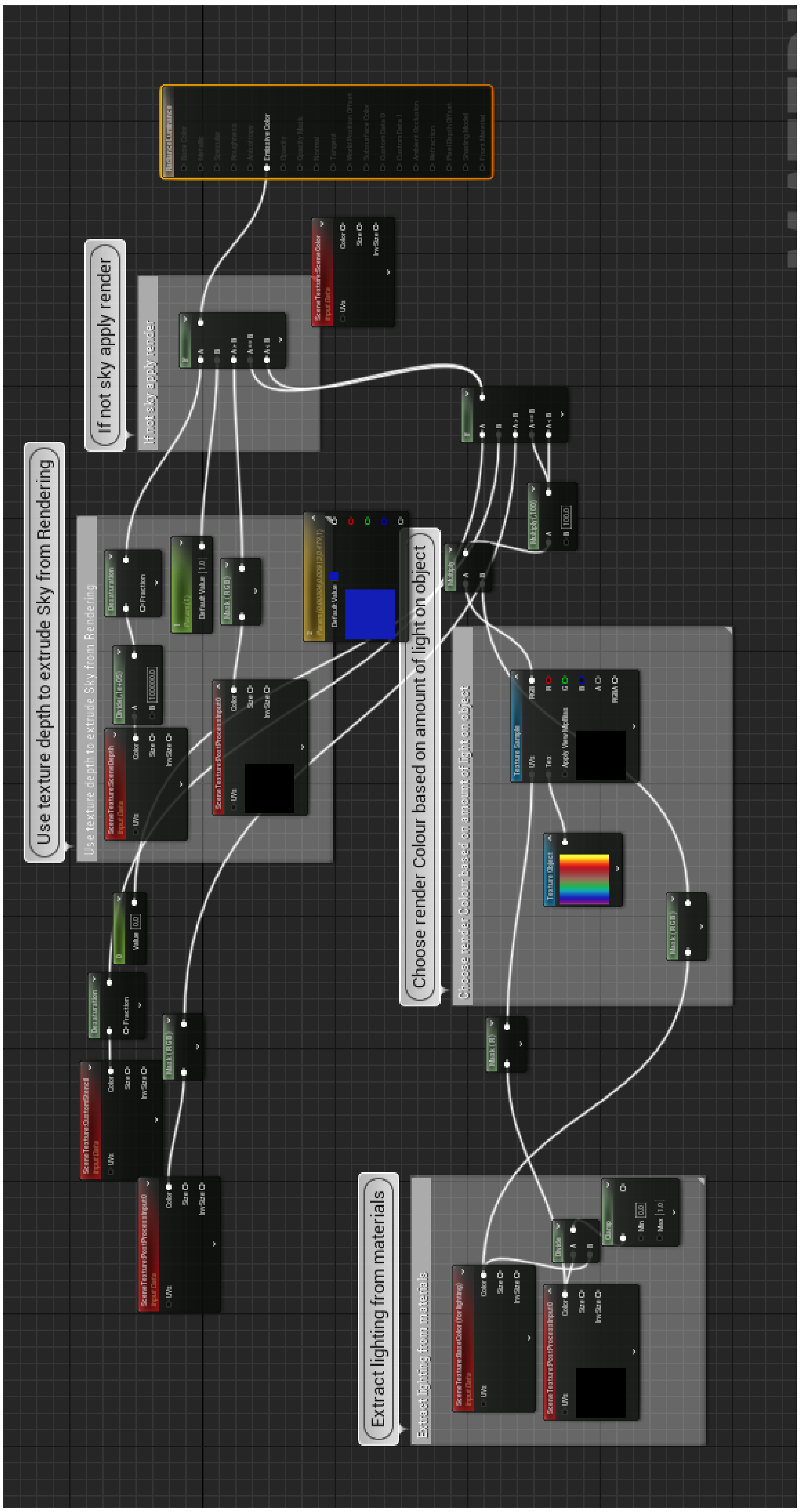

Event Construct
SET
Show Mouse Cursor
Target
Get Owning Player
Target is Widget
Target self
Return Value
Set Input Mode Game And UI
Player Controller
In Widget to Focus
In Mouse Lock Mode
Lock Always
Hide Cursor During Capture
Flush Input
Self
Set Ignore Look Input
Target is Controller
Target
New Look Input
Get All Actors Of Class
Actor Class
Sun Sky
Out Actors
GET
Slider TOD
Target
Solar Time
Set Value
Target is Slider
Target
In Value
On Value Changed (Slider_TOD)
Value
Get All Actors Of Class
Actor Class
Sun Sky
Out Actors
GET
SET
Solar Time
Target
Update Sun
Target is Sun Sky
Target


Event Construct
SET
Show Mouse Cursor
Target
Get Owning Player
Target is Widget
Target self
Return Value
Set Input Mode Game And UI
Player Controller
In Widget to Focus
In Mouse Lock Mode
Lock Always
Hide Cursor During Capture
Flush Input
Self
Set Ignore Look Input
Target is Controller
Target
New Look Input
Get All Actors Of Class
Actor Class
Sun Sky
Out Actors
Slider TOD
GET
Target
Month
Set Value
Target is Slider
Target
In Value
On Value Changed (Slider_TOD)
Value
Get All Actors Of Class
Actor Class
Sun Sky
Out Actors
GET
SET
Month
Target
Truncate
A
Return Value
Update Sun
Target is Sun Sky
Target

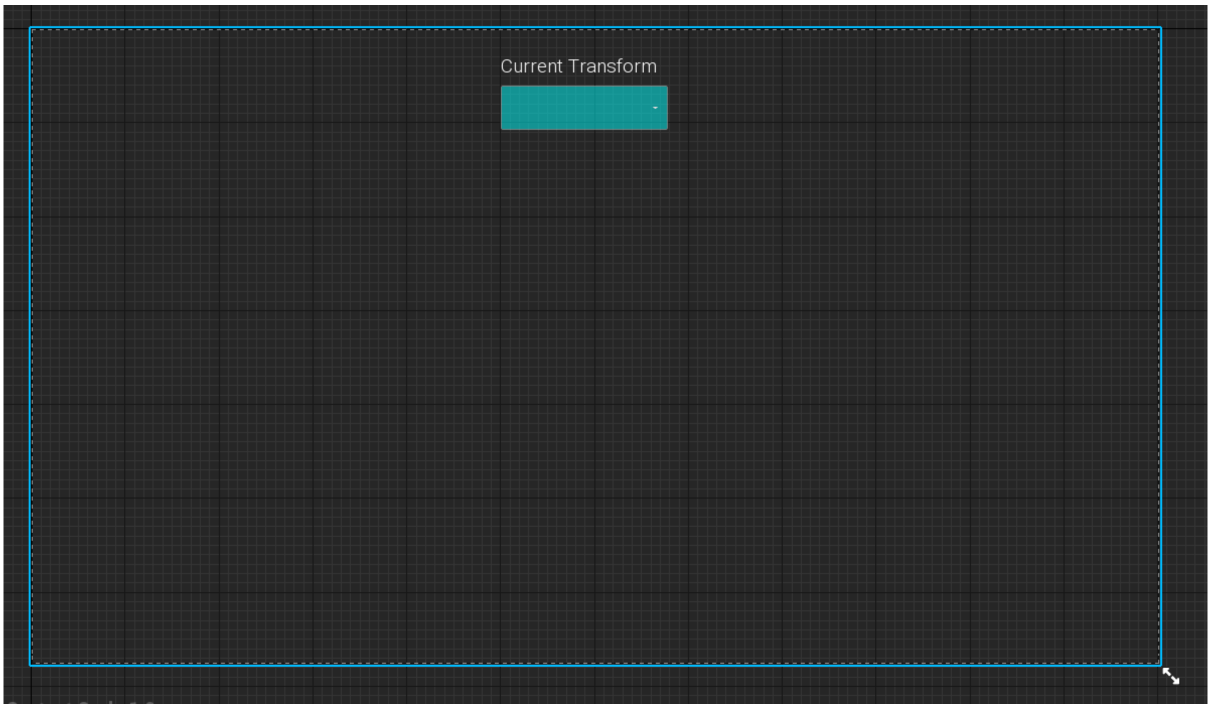
Appendix G Reading Luminance value using custom C++ actor

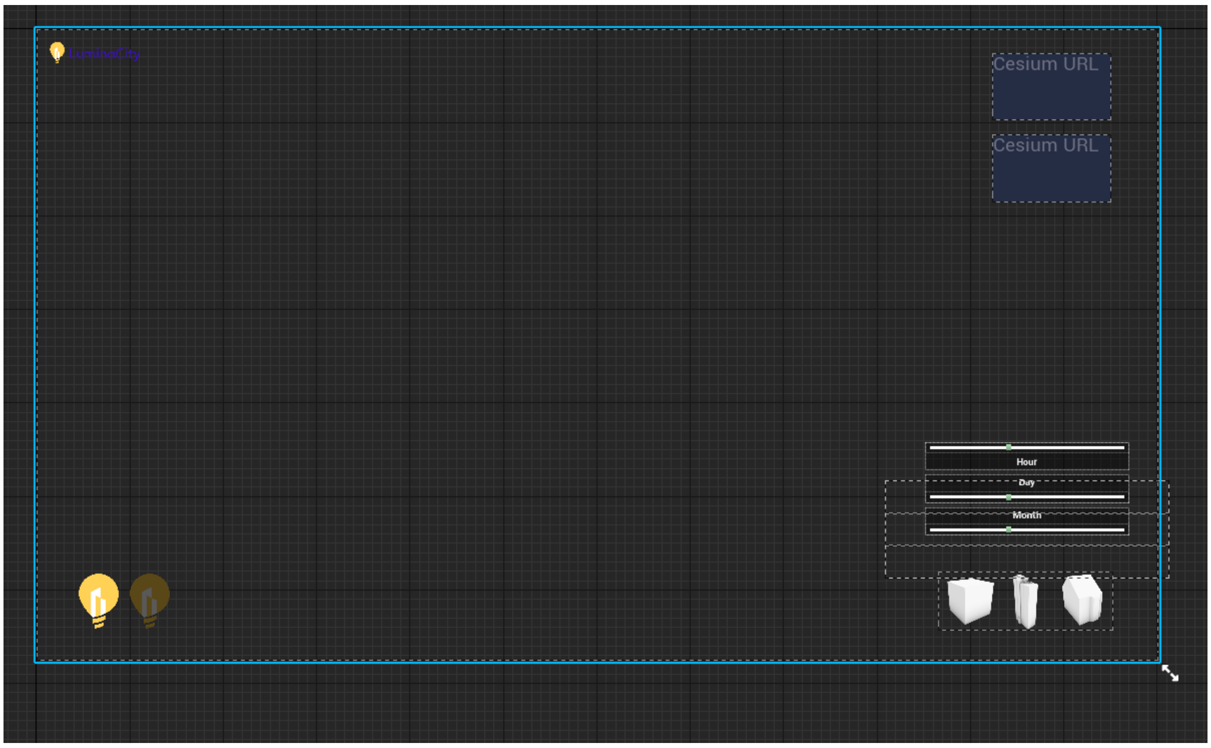


Appendix H False Colour post processing


If not sky apply render
Use texture depth to extrude Sky from Rendering
Choose render Colour based on amount of light on object
Extract lighting from materials

Appendix I User Interface


Current Transform


LuminaCity
Cesium URL
Cesium URL
Hour
Day
Month