applied sciences

MDPI

Article

A Framework for Learning and Rapid Implementation of Real-Time Global Illumination Methods

Cristian Lambru *, Florica Moldoveanu, Anca Morar, Victor Asavei and Alin Moldoveanu

Faculty of Automatic Control and Computers, University Politehnica of Bucharest, 060042 Bucharest, Romania; florica.moldoveanu@cs.pub.ro (F.M.); anca.morar@cs.pub.ro (A.M.); victor.asavei@cs.pub.ro (V.A.); alin.moldoveanu@cs.pub.ro (A.M.)

* Correspondence: andrei.lambru@upb.ro

Abstract: Photorealism has become a growing requirement for real-time graphics applications. Among the components needed to obtain photorealism, one of the most important is the simulation of the light transport, the result of which is called global illumination. With the growing interest in the research in this field, the need to develop a learning approach for the methods that produce real-time global illumination and a technological basis have become important topics. In this paper, we present a framework for the rapid development and testing of such methods. The framework itself does not contain their implementations, but contains a set of components that underlie them. It has been developed with a multi-pass architecture that allows the reuse of pass implementations in several technique pipelines. Moreover, this framework can be used in the process of learning real-time global illumination methods thanks to an editor with graphical user interface (GUI) that offers a rich amount of information and a set of such methods, created in the framework for demonstration purposes. We evaluated the proposed framework in two ways. First, through a comparative analysis with other frameworks, which have been created or at least can be used for educational purposes and the rapid implementation of real-time global illumination techniques. The second evaluation was made for the applicability of the framework in learning by using it as software support in the process of testing global illumination knowledge on eight participants with background in computer graphics.

Keywords: educational framework; computer graphics; real-time global illumination

Citation: Lambru, C.; Moldoveanu, F.; Morar, A.; Asavei, V.; Moldoveanu, A. A Framework for Learning and Rapid Implementation of Real-Time Global Illumination Methods. *Appl. Sci.* **2022**, *12*, 5654. https://doi.org/10.3390/app12115654

Academic Editors: Yutaka Ishibashi and Jan Egger

Received: 6 May 2022
Accepted: 30 May 2022
Published: 2 June 2022

1. Introduction

Photorealism in real-time graphics applications has become an important research topic, with the increasing capabilities of graphics processing units (GPUs). One of the most important components needed to achieve this result is the correct physical simulation of the light transport. Simulating the light transport from a light source to the observer after the reflection on a single surface of the scene or after reflection on several surfaces, the interaction of light with translucent or transparent surfaces and the interaction of light with air particles represent only a part of the phenomena necessary to obtain a correct physical result. This result is called global illumination [1]. Several methods [3**?**] have been proposed to produce the result in real time or have been adapted to achieve real-time performance as the GPU capabilities have grown.

As the interest in this topic grew, so did the need to facilitate the learning of real-time global illumination methods. However, the subject of learning these methods has not been investigated extensively. Furthermore, the real-time global illumination approaches differ quite a bit, so the implementation and management of several techniques in the same software can be difficult. Thus, the need for a framework that supports a large set of such methods has become important for the learning and research processes in the field.

In this paper, we present a support framework for the development of a wide range of real-time global illumination techniques. The source code of the framework is available in

the Supplementary Materials. The framework itself does not contain their implementations, but contains a set of basic components needed to facilitate the rapid implementation of such methods. This framework has a multi-pass architecture that allows the reuse of code for several pipelines. In addition, it contains an editor with GUI that provides debugging information that is important in the implementation process. Thanks to this editor and the set of real-time global illumination methods implemented for demonstration purposes, this framework can be used in the process of learning these methods.

We also performed an evaluation of the proposed framework. First, we have conducted a comparative analysis with several other frameworks that have applicability in the learning and rapid implementation of real-time global illumination techniques. The criteria chosen for the comparison was whether or not the frameworks contain implementations of such methods, because they represent a good educational support and the capacity of the frameworks to allow the implementation of new techniques. The second evaluation was made for the applicability to learn real-time global illumination methods. We used the proposed framework as software support to test real-time global illumination knowledge on eight participants with a background in computer graphics.

This paper is structured as follows. Section 2 presents the related works on frameworks intended for real-time global illumination approaches and which were designed for educational purposes and have applicability in research. Section 3 describes the proposed framework with technical details and software architecture and Section 4 presents a set of real-time global illumination methods implemented in the framework for demonstration purposes. Section 5 analyzes the framework applicability in the learning process, with respect to the rapid implementation of new methods. Section 6 shows the results represented by a comparative analysis between the proposed framework and those analyzed in Section 2 in terms of their use both for educational purposes and for rapid implementation of techniques. Additionally, an evaluation from the point of view of using the proposed framework for learning real-time global illumination methods is presented in this section. Section 7 presents our conclusions.

2. Related Work

2.1. Real-Time Global Illumination Methods

The software applications that contain implementations of global illumination methods are complex. They can be considered to contain three major stages:

1. Management of geometric data and illumination properties in the scene;
2. Light transport simulation;
3. Result display, which may require post-processing effects and adaptation to display hardware requirements.

Usually, the implementation of global illumination methods is done in the light transport simulation stage and remains independent of the other two stages, which are in the management of the software application. However, due to the nature of the processes required for such methods, this stage is the most complex.

To achieve the light transport simulation in real time, several pre-processing steps are used, which obtain the light transport information for the scene geometry, possibly followed by several steps of processing the information obtained in the acquisition stages. The last step of the light transport simulation is the lighting computation for the geometry visible by the observer by extracting the information produced in the previous stages.

Among the real-time global illumination methods, there are several approaches that use certain data structures to store the light transport information, as presented by Ritschel et al. [**?**] and more recently by Lambru et al. [3]. This research field is complex and goes beyond the scope of this paper, so for a comprehensive analysis, the work cited above can be consulted. Here, we give a brief overview of the real-time global illumination methods in terms of the requirements of a software application to contain the implementations of a wide range of approaches. We analyze the nature of the processes and data structures required for such applications to provide the necessary flexibility.

Among the real-time global illumination methods, we further analyze those based on images and voxel volumes. The image-based methods extract information from the scene by using a set of light fields, obtained by rendering the scene from certain positions. A light field contains 2D information along the projection lines of the camera, but can be used to extract the light transport information. In this class, we note the methods that use the information on the screen, obtained by rendering the scene from the position of the observer to produce diffuse indirect illumination [4] and glossy reflections [5]. From the class of image-based methods, there are also those that use the information from the light space, obtained by rendering the scene from the position of the light. Dachsbacher and Stamminger [6] introduced the concept of the reflective shadow map (RSM), which is a set of buffers that store the light transport information for the geometry visible from the light position. Initially, this approach was proposed to compute the diffuse indirect illumination. Another large class of real-time global illumination methods is represented by those that produce a voxels volume [7,8] in which the information about light transport is stored.

There are several approaches [3**?**] that process these data structures after the acquisition stage. Moreover, there is a wide range of techniques that sample them. Additionally, some methods use several such data structures, some simultaneously at the step of extracting lighting information or sequentially to obtain different data structures.

2.2. Learning Real-Time Global Illumination Methods

The subject of learning real-time global illumination methods has not been touched upon much in the literature. An analysis [9] of the topics addressed by several universities in introductory computer graphics courses shows that these methods are not part of the core curriculum. However, the subject of global illumination has been addressed within most of these courses, but in the context of offline rendering through techniques such as ray tracing. The next topics that were addressed within many courses were those related to real-time rendering, but without touching the subject of real-time photorealism.

The subject of real-time global illumination is also little addressed in the literature of technologies and tools designed for learning computer graphics [10]. Several tools have been proposed for learning offline global illumination methods, such as ray tracing, but without touching on the subject of real-time photorealism. Moreover, the possibility of using commercial products, such as Unity [11] game engine, to learn computer graphics has been investigated [12]. Although this engine contains the implementation of some real-time global illumination methods and allows their modification and extension, the possibility of learning global illumination has not been addressed.

Recently, Vitsas et al. [13] proposed an online software application that allows the implementation and visualization of ray tracing-based methods for educational purposes. They have successfully used the application for educational purposes by using the application to teach techniques based on ray tracing. However, they did not analyze other classes of real-time global illumination and the application cannot be extended by users for other such classes of techniques.

2.3. Rapid Implementation of Real-Time Global Illumination Methods

As presented in Section 2.1, the software applications that contain the implementation of global illumination methods are complex and contain several processing stages. For this reason, as Slusallek and Seidel [14] observe, over time, research groups have used their own development frameworks and the use of code between two such frameworks has been deficient. Moreover, the transfer of technology from research to industry has been difficult.

The development of a flexible software kernel for rapid and easy implementation of global illumination methods has been a topic of research interest in order to unify the various approaches in this field. Several frameworks have been described, but most are old and have been proposed in the context of offline global illumination methods, such as ray tracing. Kirk and Arvo [15] proposed such a kernel for the particular method of ray tracing and its variations. This kernel was later extended by Shirley et al. [16].

A framework that has allowed the implementation of several global illumination methods was developed by Cornell University [17]. Another flexible framework, which allowed the implementation and extension of a larger set of global illumination methods was proposed by Slusallek and Seidel [14]. An interesting approach was provided by Debattista et al. [18], who proposed a theoretical system that provides control over the rendering of the scene in terms of global illumination. The approach is independent of the lighting method. Subsequently, Dutre et al. [19] described in the "Advanced Global Illumination" book several methods, in parallel with the software architecture required to implement them.

Over time, several frameworks have been used for research in the field. However, these frameworks have not focused on the flexibility of use for different approaches, but on the ease of use and quality of results for certain classes of techniques and certain particular purposes. The suite of applications known as Radiance [20] is an example. Another system is known as Physically Based Rendering Toolkit [21], which offers similar capabilities. A rendering system that offers more flexibility and introduces a scripting language is Mitsuba 2 [22].

Several approaches to real-time application architectures have been proposed. Dollner and Hinrichs [23] presented a software architecture of real-time rendering techniques, but without the focus on global illumination methods. The book *Real-Time Rendering* by Akenine-Moller et al. [24] presents the state of the art for real-time rendering methods, together with software details for the implementation of such approaches. However, particularly for global illumination, the techniques are mentioned from a theoretical point of view, without emphasis on the software component.

Subsequently, several real-time global illumination methods have been proposed [3**?**]. Research in the field of development frameworks for these methods has been directed towards proprietary and commercial game engines. However, most of these engines contain a monolithic global illumination system that implements a fixed set of methods.

Unreal Engine 5 [25] offers a monolithic illumination system, known as Lightmass [26], that offers implementations for offline methods that cache indirect illumination at a pre-processing step. It also contains another monolithic system, known as Lumen [27], that offers the implementation of real-time global illumination with the ray-tracing hardware. The users can implement their own methods insofar as they modify the code [28] which is currently open source with a subscription to the EpicGames community on GitHub. However, the code base of the engine is large and difficult to comprehend due to the lack of documentation. Therefore, the implementation of new rendering pipelines is difficult.

CryEngine [29] provides real-time global illumination through voxel-based methods in a monolithic system. The user can create their own methods with the modification of the source code [30]. However, similar to the Unreal Engine mentioned above, the code base is large and the code is difficult to modify.

Unity [11] game engine offers several approaches through which different rendering pipelines can be implemented. First, it contains a monolithic system, known as Built-in Render Pipeline (BRP) [31], which has a set of lighting techniques that require offline pre-processing steps. To obtain greater flexibility, the developers have created what they call the Scriptable Render Pipeline (SRP) [32], which allows the user to create rendering pipelines with a scripting language. This approach uses a multi-pass architecture. They created two frameworks on top of it. The first, called Universal Render Pipeline (URP) [33] allows the user some degree of flexibility. However, it is intended to replace the BRP, so it has the same monolithic illumination system. The second framework is called High Definition Render Pipeline (HDRP) [34] and contains implementations of several methods that meet modern requirements of real-time rendering. It also contains the implementation of a suite of illumination effects obtained with ray tracing hardware. In addition, it uses a higher application programming interface (API) level of a multi-pass architecture, compared to the SRP, that offers a certain level of flexibility and allows the user to implement their own rendering passes. The pipeline offers two types of passes [35], scene render pass and

post-process render pass, that allow the implementation of a satisfactory large number of approaches necessary for modern requirements.

In addition to the commercial game engines, there are various open-source game engines that offer different lighting implementations, but most have monolithic systems. Godot [36] game engine has a monolithic system that uses its own implementation of signed distance field global illumination that allows partial real-time indirect illumination. In addition to some post-processing passes that produce certain real-time illumination effects, the rest of the approaches require offline pre-processing steps. Open 3D Engine [37] also has a monolithic system that uses dynamic diffuse global illumination, which applies ray tracing for real-time global illumination. For the situation where the hardware does not support ray tracing, it uses the same technique for offline pre-processing. Similar to Godot, it has different post-processing steps to produce several real-time illumination effects.

Global illumination methods have not been limited to game engines; model editor applications also offer the implementation of such methods. However, they are based on offline rendering approaches. An example is Blender [38], which provides several global illumination systems. Even though this editor offers an experimental game engine, global illumination implementations are intended for offline rendering and the real-time rendering is done by methods that do not contain indirect illumination.

NVIDIA company offers a suite of frameworks designed for offline rendering with the ray tracing method, such as NVIDIA Iray [39], which uses the NVIDIA OptiX Ray Tracing Engine [40]. Due to technological advances of the GPUs, these frameworks can be used in applications for real-time rendering, but their use is experimental and is intended only for ray tracing-based methods. Moreover, in the NVIDIA Gameworks library there are several open-source demonstration frameworks for different approaches to real-time global illumination. RTX Global Illumination (RTXGI) [41] is a framework that uses ray tracing for the hardware that supports this implementation, to obtain real-time illumination. NVIDIA VXGI [42] is a framework that uses the rasterization pipeline to obtain real-time global illumination by methods that use the voxel representation of the scene. NVIDIA Falcor [43] is a more general framework, developed for the implementation of real-time rendering techniques. Although it contains a set of methods that produce different real-time illumination effects, this framework does not contain a comprehensive set of demonstration methods and does not focus on real-time global illumination techniques. This framework optionally incorporates the others presented above. It uses a multi-pass architecture for its real-time rendering pipelines.

Furthermore, the development of the frameworks is not limited to NVIDIA company. AMD company has developed the Radeon Cauldron framework [44], which, similar to NVIDIA Falcor, provides a set of open-source implementations to produce several real-time illumination effects, but does not offer a comprehensive set of real-time global illumination methods. This framework acts as an external library that can be used to implement rendering pipelines and has a multi-pass architecture.

3. Proposed Framework

3.1. Framework Architecture

As presented in Section 2.1, software applications that contain real-time global illumination methods require a high degree of flexibility to support a wide range of approaches and processing steps. Furthermore, some of these processing steps are used by several different methods.

Due to these requirements, it comes naturally to use the concept of rendering pass as an atomic unit in a rendering pipeline. Such a rendering pass needs to be unitary and used in a sequence of passes. One such approach, known as multi-pass architecture, is the one used in the framework described in this paper, tailored to the needs of real-time global illumination methods.

The framework architecture contains three major subsystems: Engine, Render and Editor, as can be seen in Figure 1.

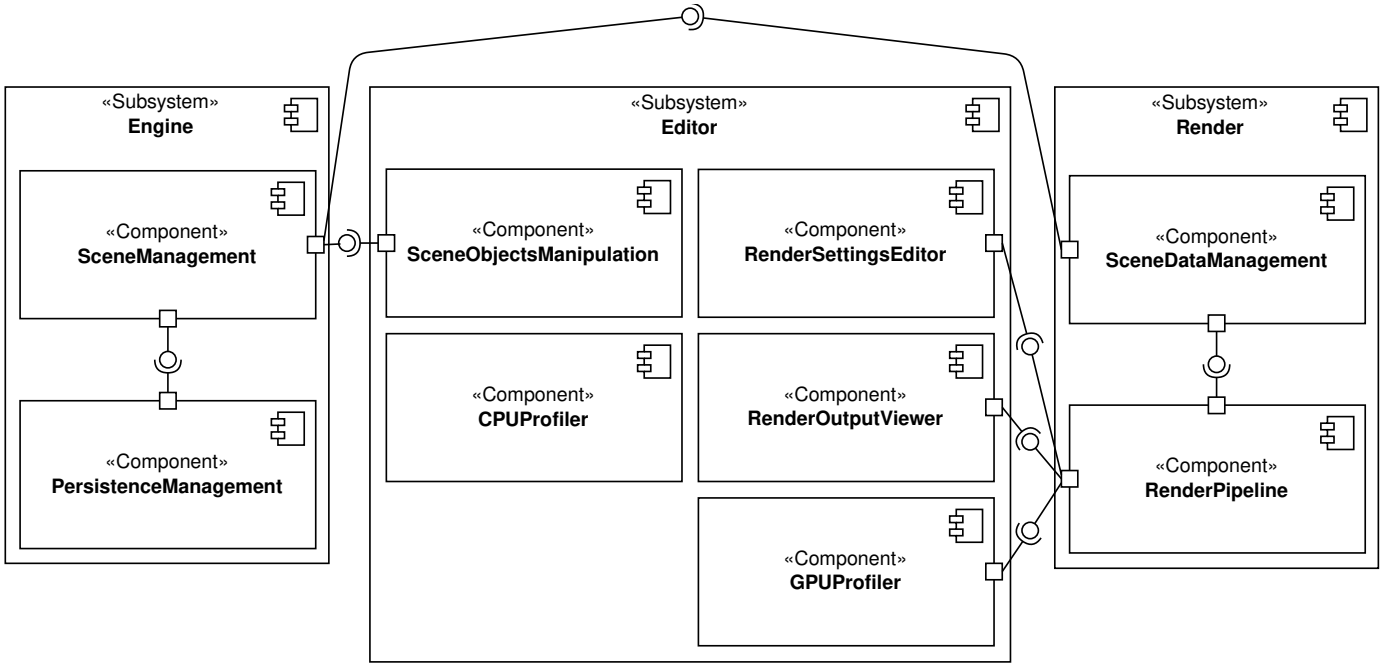

Figure 1. The components diagram of the proposed framework.

3.1.1. Engine Subsystem

The Engine Subsystem manages the flow of the framework and the data. The persistence of the data is managed by the PersistenceManagement Component and is ensured on the disk in specialized files for scene information, such as geometry, lighting information and material properties of the surfaces. This data is loaded by the SceneManagement Component and is stored as a hierarchy of objects for the entire scene.

3.1.2. Editor Subsystem

The Editor Subsystem has multiple responsibilities. It deals with the display of the scene rendering result and the debugging information produced by the other two subsystems. It also allows real-time modifications of the objects hierarchy in the scene and the parameters of the rendering pipeline. To enable these user interactions, a GUI has been created with the Dear ImGui [45] library. This GUI can be seen in Figure 2.

The RenderOutputViewer Component is responsible for managing some of the rendering settings, such as framebuffer resolution, view camera position and direction and projection properties. It is also the one that deals with the display of the visual result produced by the rendering pipeline inside the editor GUI.

The SceneObjectsManipulation Component has the responsibility to allow the user to manipulate objects in the scene. In the window where the result of the rendering pipeline is displayed, visible in Panel (b) of Figure 2, is an interactive gizmo that allows changing the position, scale and rotation of the selected object. This change can be made by the user through the interaction of the gizmo with the mouse. The result of all changes is viewed in real time. In addition, the hierarchy of the scene objects can be changed in the window visible in the Panel (a) of Figure 2. Any change in the scene information can be made permanently by choosing the save option from the menu.

The RenderSettingsEditor Component allows the viewing and modifying the parameters of the rendering pipeline. It also allows a switch between different pipelines. A set of such information can be seen in Panel (c) of Figure 2. In the same panel, the user can view debug information of the pipeline, such as intermediate framebuffers.

The CPUProfiler and GPUProfiler Components, through editor GUI, provide real-time information about the performance of the rendering pipeline. In the window visible in Panel (e) of Figure 2 is a profiler for the central processing unit (CPU) and GPU. This

window allows a switch between viewing the performance information on the CPU and on the GPU.

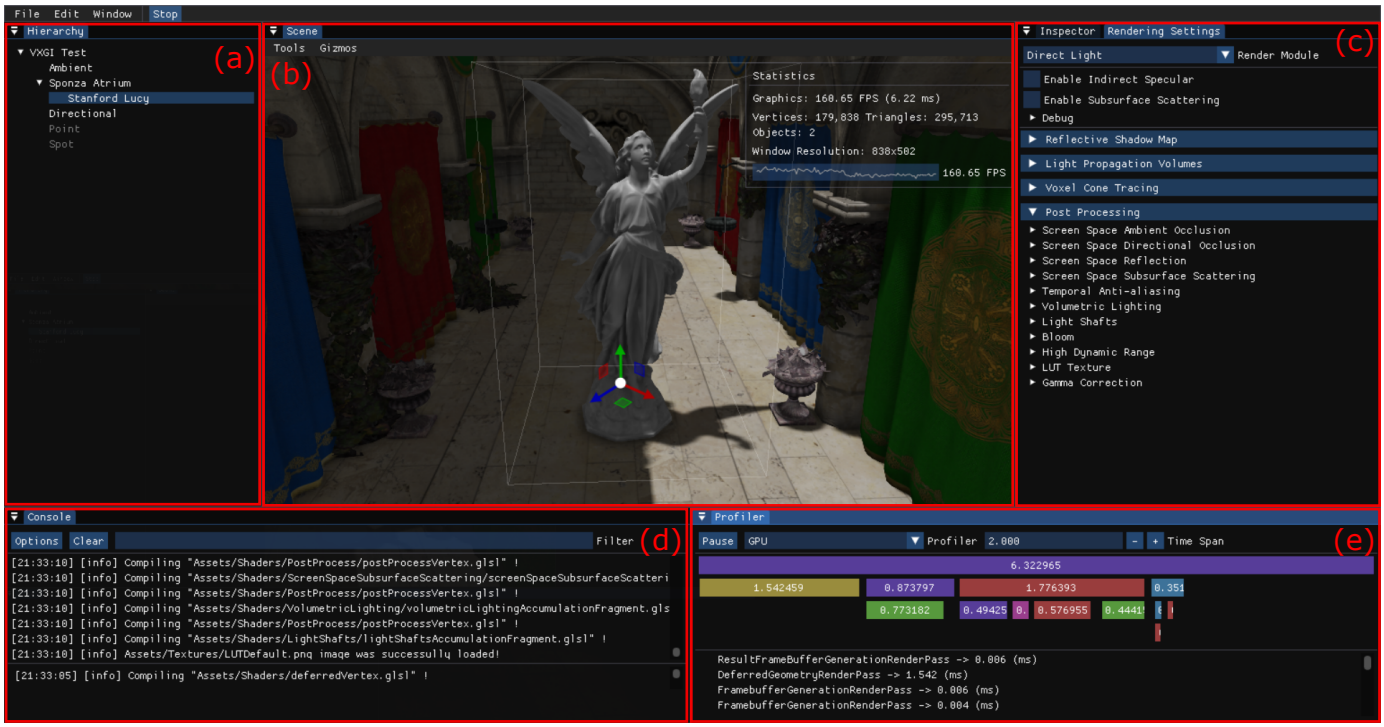

Figure 2. Editor GUI designed for the framework. Panel (**a**)—the window that displays the objects hierarchy of the scene; Panel (**b**)—the window that displays the result of the rendering pipeline; Panel (**c**)—rendering pipeline parameter management window; Panel (**d**)—debug information display window; Panel (**e**)—CPU and GPU profiler information display window.

3.1.3. Render Subsystem

Scene Data Management

The SceneDataManagement Component is responsible for managing the geometry and illumination properties of the scene and to provide access to this information.

Scene data is loaded from files on disk and stored in a hierarchy of objects that contain properties. The geometry of the scene objects is transformed into the format required by the OpenGL 4.6 graphics API and is loaded into the GPU memory when the framework starts. Any modification of the scene objects, such as creating or deleting an object or loading an entire other scene, changes the information loaded in the GPU memory.

For better management of drawable objects and lighting information in the scene, such as the set of directional, point and spot light sources, we stored the scene information in parallel with the management of the Engine Subsystem. We stored a special scene for the Render Subsystem, similar to the approach of physics engines such as Bullet. This scene is stored in `RenderScene` class, visible in the class hierarchy in Figure 3.

The information in the rendered scene is stored optimized for the processes required for the rendering pipeline. One of these processes is the specific iteration through certain types of objects, such as drawable objects or directional, point or spot light sources. Moreover, objects in the rendered scene can be easily queried for information needed for the rendering, such as the position or rendering state. In addition, the scene allows the interrogation of specific information, such as ambient light information.

The scene information in the Engine Subsystem must be synchronized with that in the scene in the Render Subsystem. Thus, for any modification of the objects, such as the creation, deletion or change of the rendering state, the information from the render scene must be updated.

Rendering Pipeline

The RenderPipeline Component is responsible for supporting the processing stages specific to the real-time global illumination methods.

The framework contains several implementations of rendering pipelines for different real-time global illumination methods for demonstration purposes. However, the quality of the framework is represented by the Rendering Subsystem that allows the rapid and easy development of such methods. This subsystem has been designed with a multi-pass architecture.

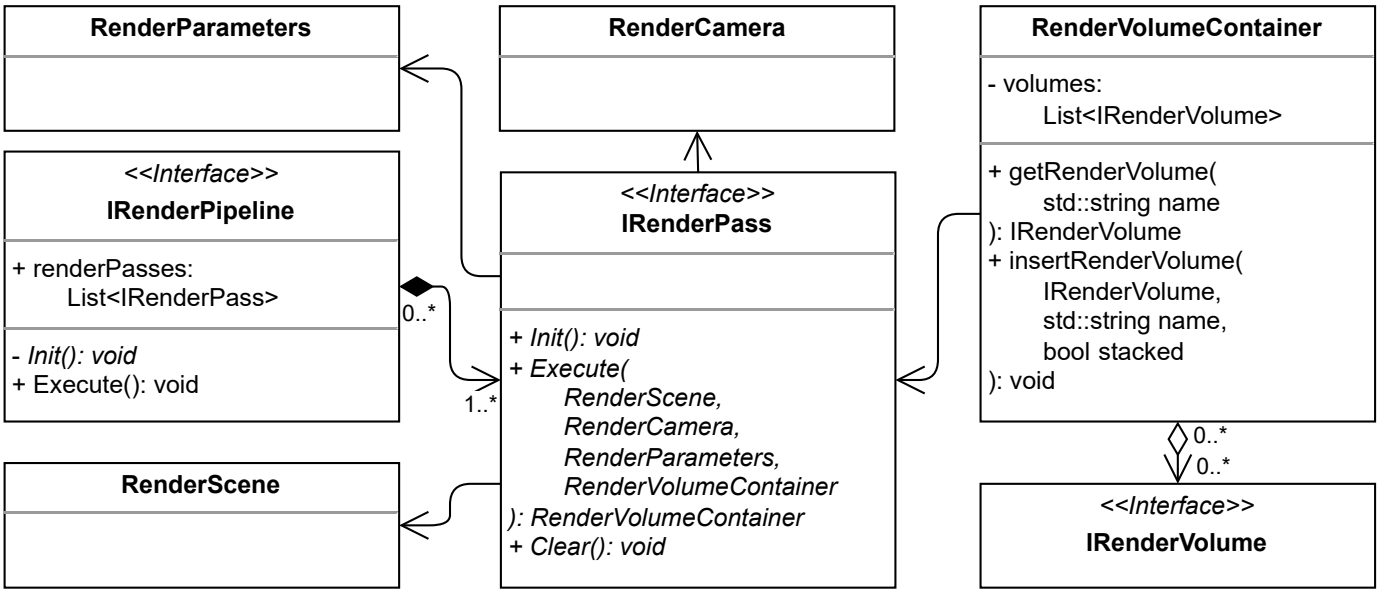

Figure 3. The class hierarchy of the rendering subsystem.

A rendering pass represents a working unit of the pipeline. A sequence of rendering passes represents a rendering pipeline. This can be seen in Figure 3, through the relationship between the `IRenderPipeline` and `IRenderPass` interfaces. To implement a real-time global illumination method, a rendering pipeline and a set of rendering passes must be specialized. The passes can be reused for several pipelines. Concretely, the pipeline specialization must implement the `Init()` method in which to insert in the list of passes instances of the pass specializations. For example:

```
void SpecializedRenderPipeline::Init ()
{
renderPasses.push_back (new SpecializedRenderPass1 ());
renderPasses.push_back (new SpecializedRenderPass2 ());
renderPasses.push_back (new SpecializedRenderPass3 ());
}
```

The order of inserting the rendering passes also represents the order of execution. The `Execute()` method of the `IRenderPipeline` interface is called every frame and calls the `Execute(...)` method for every rendering pass.

A specialization of the `IRenderPass` interface must implement the methods shown in Figure 3. The `Execute(...)` method is called every frame, so that it can perform data processing on the CPU or GPU at the frame level. The purpose of this method specialization is to send to the output a render volume obtained as a result of the pass processing. The rendering volume can contain any information, such as the framebuffer resulting from rendering. The class in which this rendering volume is implemented must be a specialization of the `IRenderVolume` interface. For simplicity, this volume is stored inside a volume container. In the `Execute(...)` method, the rendering pass receives as parameters

the rendered scene (which was detailed above), the rendering camera, the rendering parameters and the container of rendering volumes that were produced at the output by the other rendering passes processed previous to the current one.

The rendering parameters are stored in a structure which contains the information about the rendering pipeline. This information is persistent on disk in a special file and is modifiable in the editor GUI in the window visible in Panel (c) of Figure 2.

Sets Iteration

Some real-time global illumination method pipelines require the ability to process a set of rendering passes for every element of a set, such as point or spot light sources. To facilitate such a process, we have specialized the `IRenderPass` interface for a container of rendering passes that stores and processes a set of subpasses. This approach can be on several levels, a container can store other containers inside it. An iterable set interface has also been introduced that must be specialized for a set, such as `PointLightsIterator` visible in Figure 4. The construction of a container is as follows:

```
void SpecializedRenderPipeline::Init ()
{
renderPasses.push_back (
new RenderPassContainer ()
->setRenderIterator (new PointLightsIterator ())
->insertRenderPass (new SpecializedRenderPass1 ())
->insertRenderPass (new SpecializedRenderPass2 ())
);
}
```

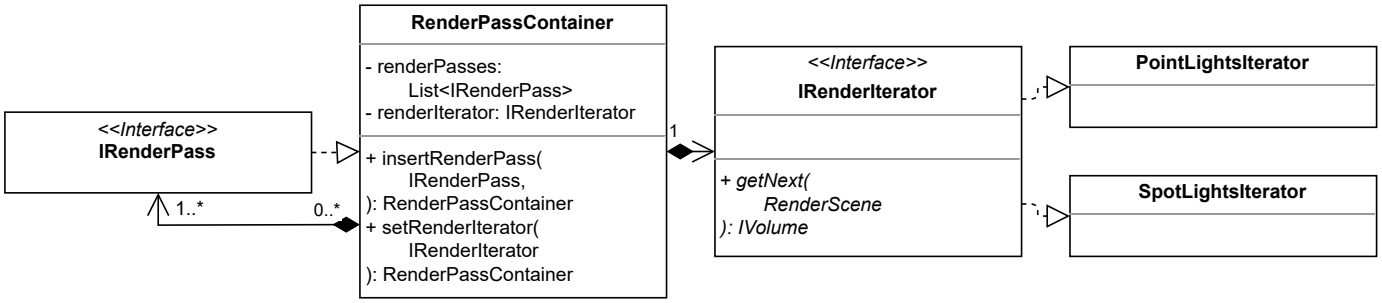

Figure 4. The class hierarchy used for sets iteration in the rendering subsystem.

A container processes the rendering passes in every frame in the order in which they were inserted.

Data Generation

Some real-time global illumination methods require data sets that are independent of scene information, such as sampling patterns. Our approach can support such a scenario without further modification by using a rendering pass to generate such data. Data sets can be generated at every frame or generated only once in the `Init()` method of `IRenderPass` specialization and only sent to the rendering pass process in every frame.

3.2. Technical Details

The framework was developed in the C++17 programming language with the OpenGL 4.6 graphics API. The source code of the framework is available in the Supplementary Materials. It was developed to be used for both Windows and Linux operating systems. Regarding the build suite, we used Microsoft Visual C++ compiler for Windows and the gcc compiler for Linux.

The framework can be built and operated for all versions of Windows 7 and up. For Linux, several distributions have been tested, such as different versions of Debian and Arch.

The framework has been used on several machines that contain NVIDIA GPUs. The type of GPU with the minimum specifications that has been tested is GTX 950 m. In terms of CPU, it has been used on Intel machines.

4. Demonstrative Methods of Real-Time Global Illumination

The framework contains a set of real-time global illumination techniques for demonstration purposes. We have implemented a set of illumination effects for four classes of methods that produce real-time global illumination. More precisely, we chose as lighting effects:

- Diffuse indirect illumination;
- Glossy reflections;
- Subsurface scattering and translucency;
- Direct illumination of area light sources.

These effects have been implemented for global illumination classes that contain:

- Techniques based on RSMs;
- Techniques based on discrete ordinate methods (DOMs);
- Techniques based on voxel volume;
- Techniques based on screen space information.

The field of real-time global illumination is complex and the description of the implemented techniques goes beyond the scope of this article. Below, we only mention the techniques used to implement the real-time illumination effects.

From the class of the RSM-based methods, we implemented diffuse indirect illumination according to the sampling technique proposed by Dachsbacher and Stamminger [6] and both glossy reflections and refraction according to the approach that uses 2D ray casts in RSM proposed by Lambru et al. [46]. For the 2D ray cast process on images, we used the approach proposed by McGuire and Mara [47]. This class is the only one that cannot produce direct illumination for area light sources.

From the class of DOM-based methods, we implemented all the illumination effects based on the method proposed by Kaplanyan and Dachsbacher [7] that uses a light propagation volume (LPV). Moreover, from the class of voxel-based techniques, we implemented all the effects based on a single approach, more precisely the one proposed by Crassin et al. [**?**].

From the class of techniques that use the information on the screen, we implemented diffuse indirect illumination, illumination, following the method proposed by Ritschel et al. [4] and the glossy reflections together with the translucency, based on the one proposed by Sousa et al. [5]. Direct illumination of area light sources is obtained with these methods.

The visual results of the four lighting effects in the same scene can be seen in Figure 5. As previously mentioned, the RSM-based methods cannot produce direct illumination for area light sources, so the light source has been excluded. The visual results are not identical due to the significant differences between the implemented approaches.

In addition, the framework provides a set of parameters that can be changed interactively by the user through the GUI and for which the visual results are modified in real time. This set of parameters are presented in Table 1. The DOM- and voxel-based techniques have a set of generic parameters for the properties of the 3D volumes they use.

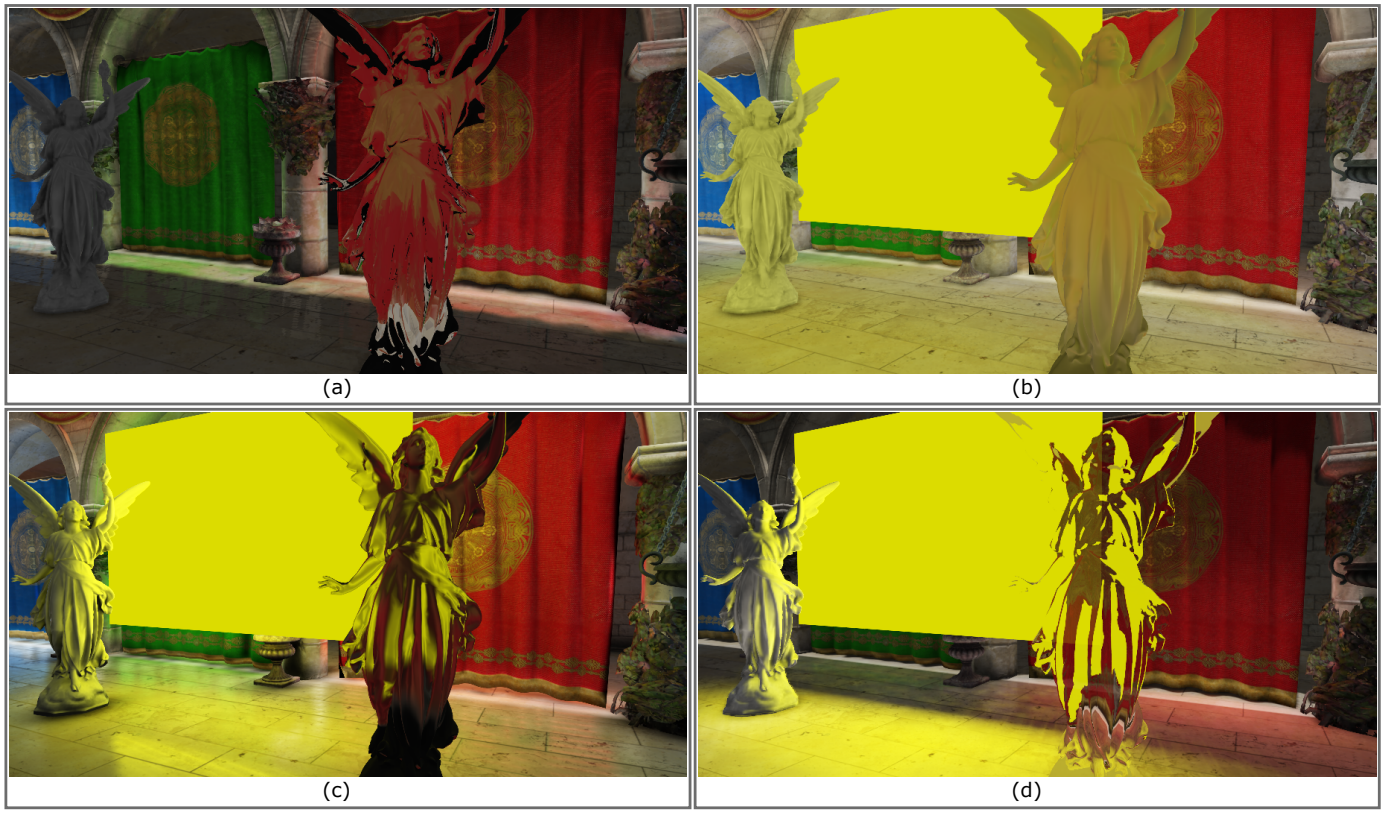

Figure 5. Diffuse indirect illumination, glossy reflections, subsurface scattering, translucency and direct illumination for area light sources implemented for different classes of methods for real-time global illumination. Panel (**a**)—RSM-based methods; Panel (**b**)—DOM-based methods; Panel (**c**)—voxel-based methods; Panel (**d**)—screen space methods.

Table 1. Parameters that can be modified in real time provided by the framework.

Real-Time Global Illumination	Generic	Diffuse Indirect Illumination	Glossy Reflections	Subsurface Scattering & Translucency	Direct Illumination for Area Lights
RSM-based methods	–	intensity, number of samples, sampling radius, screen interpolation parameters	intensity, number of iterations, sample thickness	intensity, number of iterations, sample thickness	N/A
DOM-based methods	LPV resolution, number of propagation iterations, use of geometry occlusion	intensity	intensity, number of iterations	intensity, number of iterations	number of samples per triangle
Voxel-based methods	voxel volume resolution, use of continuous voxelization, voxel volume mipmap levels	intensity, cone distance in voxel volume space	intensity, cone radius/height ratio, cone distance in voxel volume space	intensity, cone radius/height ratio, cone distance in voxel volume space	–
Screen space methods	–	intensity, number of samples, sampling radius, screen interpolation parameters	intensity, number of iterations, sample thickness	intensity, number of iterations, sample thickness	–

5. Use Cases for the Proposed Framework

5.1. Using the Framework for Learning

The educational use of the proposed framework has as main purpose its use for learning real-time global illumination methods. For this purpose, one can take advantage of the fact that a set of such techniques is already implemented within the framework.

The framework can be used to investigate final and intermediate visual results obtained by several real-time global illumination methods. In the video attached in the Supplementary Materials, the final and intermediate results for diffuse indirect illumination and glossy reflections for all classes of real-time global illumination methods implemented in the framework are visible.

Furthermore, for every method there are several rendering parameters that can be modified from the editor, as shown in Table 1. As mentioned in Section 3.1.2, the change of rendering parameters is done through user interactions and parameters are modified in real time. For this reason, the rendering pipeline and implicitly its visual result are updated in real time.

The framework allows the investigation of the real-time global illumination pipeline implementations. Due to the use of a multi-pass architecture, such an implementation can be investigated at several levels, such as an overview of the sequence of rendering passes and the code of a single pass independently of the rest of the pipeline.

Due to the implementation of a rendering pipeline through a sequence of passes, there is the possibility to modify the code without affecting or interacting with the rest of the pipeline. This approach allows the creation of new rendering passes and the modification of the code of some techniques or shaders used inside the passes without affecting a part of the other rendering passes.

A suitable scenario to use the framework for education purposes would be to allow the student to implement particular illumination effects in a rendering pipeline. Implementations can be available for all other rendering passes. The ones aimed for implementation by the students should contain only a skeleton in which they can complete important steps in a rendering pipeline. Such code locations could be the creation of the output data structure of a rendering pass and the indirect lighting computation for different effects, after a description of the theory beforehand.

5.2. Using the Framework for Rapid Implementation of Real-Time Global Illumination Methods

The software applications that contain implementations of global illumination methods are complex. Thus, the implementation requires the use of a large set of approaches. However, the use of a multi-pass architecture together with the real-time global illumination techniques that are already implemented in the framework, as presented in the previous section, offers some advantages.

One of the most important advantages of such an architecture is the ability to use the implementation of the same rendering pass in several different pipelines. Thus, the creation of new rendering pipelines can take advantage of the reuse of rendering pass implementations that are already in the framework. This is done, as specified in Section 3.1.3, by creating an instance of the same rendering pass for every pipeline in which the pass is used. This situation in which a rendering pass is used in several pipelines is quite necessary for the real-time rendering methods and also applies to real-time global illumination ones. This can be seen in our previous work [3], where the generation of G-buffers [49] and RSMs [6] is required for several techniques, so the rendering passes that produce these data structures are reused in several pipelines. Furthermore, the post-processing effects, implemented in rendering passes, can be reused in several rendering pipelines.

The reuse of the rendering passes that already exist in the framework facilitates the rapid implementation of pipelines. However, the process of creating a new rendering pass also takes advantage of existing implementations in the framework, as it contains a set of helpful interfaces that facilitate the implementation of certain types of passes, such as

post-processing passes. Several such interfaces are implemented in the framework, but they are missing from the presentation of the architecture visible in Figures 3 and 4 for simplicity.

Furthermore, the proposed framework has an editor with many debugging capabilities, as presented in Section 3.1.2. The facilitation of debugging through the editor GUI with all the capabilities presented in Section 3.1.2 are important in the implementation process.

5.3. Other Use Cases

With little effort, the framework can be used for other use cases. Due to photorealistic visual results, it can be used to visualize virtual worlds for various purposes, such as virtual reality applications, visualization of robotics or car simulations and designs made in computer-aided design software.

Another example is the use for data extraction, such as lighting information for virtual scenes or temporal information, such as video streams, from actions performed in those scenes. This data can be used directly by other frameworks for their own purposes or for comparative analysis between different results. Moreover, the information can be used for the training process in techniques that use machine learning.

6. Results

6.1. Comparison with Other Frameworks

We decided to compare the proposed framework with the ones described in Section 5.2. We only chose the frameworks for real-time global illumination. The comparison criteria are the possibility of using the framework as educational software and that of using it for the rapid implementation of real-time global illumination techniques. More precisely, we chose the information that tell us whether the framework contains the implementation of techniques that can be used for demonstration purposes, because it represents a good educational support, correlated with the possibility to allow the implementation of other new methods. The result of our analysis can be observed in Table 2.

Table 2. Comparison between the frameworks and game engines for real-time global illumination implementaion.

Framework/Game Engine	Contains Demonstrative Real-Time Global Illumination Technique Implementations	Allows the Implementation of New Methods
Unreal Engine 5 [25]	Monolithic, Lightmass [26] and Lumen [27]	Only by ground-up implementation
CryEngine [29]	Monolithic, Voxel-Based Global Illumination	Only by ground-up implementation
Unity BRP [31]	Monolithic, requires offline pre-processing	No
Unity SRP [32]	No	Yes, multi-pass architecture
Unity URP [33]	Monolithic, requires offline pre-processing	No
Unity HDRP [34]	Ray tracing methods	Yes, multi-pass architecture
Godot [36]	Monolithic, signed distance field global illumination	Only by ground-up implementation
Open 3D Engine [37]	Monolithic, dynamic diffuse global illumination	Only by ground-up implementation
NVIDIA OptiX [40]	Only methods based on ray tracing	Only methods based on ray tracing
NVIDIA RTXGI [41]	Only methods based on ray tracing	Only methods based on ray tracing
NVIDIA VXGI [42]	Voxel-Based Global Illumination	Only by ground-up implementation
NVIDIA Falcor [43]	Several illumination effect implementations	Yes, multi-pass architecture
Radeon Cauldron [44]	Several illumination effect implementations	Yes, multi-pass architecture

As can be seen, most frameworks are not intended for learning real-time global illumination techniques. Most of them were not created for this purpose and have monolithic system for other purposes, which are difficult to modify and have a lack of varied implementations.

The two frameworks, NVIDIA Falcor [43] and Radeon Cauldron [44], at the time of this paper, contain the implementations of several illumination effects, such as ambient occlusion, reflections and shadows, but not of a comprehensive set of real-time global illumination techniques. This is due to the fact that the purpose of these frameworks is mostly support for research. However, it should be noted that the currently implemented techniques can be investigated and evaluated together with a GUI at a detailed level, so that for certain techniques in particular, these frameworks can be a good source for education in this domain.

The frameworks that contain arranged implementations of some real-time global illumination techniques are the only ones that, among others, also have educational purposes. These are NVIDIA OptiX, NVIDIA RTXGI and NVIDIA VXGI [42]. However, they focus on a certain class of techniques and do not offer a varied set of illumination methods.

Regarding the possibility to implement new methods, the frameworks that contain the implementation of lighting techniques in monolithic systems have the least flexibility. Although access to the source code is possible for all chosen frameworks, except for Unity, it is not feasible to implement a real-time global illumination technique from the ground up, due to the fact that the code base is large and often lacks documentation.

NVIDIA OptiX and NVIDIA RTXGI frameworks offer great flexibility for the implementation of methods based on ray tracing and less on techniques that use the rasterization pipeline, for which the frameworks are similar to the monolithic ones, whose situation was presented above. The same goes for NVIDIA VXGI framework, which focuses on techniques that use the voxel representation of the scene.

The two frameworks, NVIDIA Falcor [43] and Radeon Cauldron [44], are similar to the proposed one. Both were created for the rapid implementation of rendering techniques in general and have a multi-pass architecture. However, unlike the proposed framework, they act as an external library used by applications that implement the techniques and have a lower level. They leave a lot of responsibilities to the user's management, such as the sequence of steps contained in a pipeline. Although possible through access to the source code, they do not have their own management of several pipelines at the same time and the interactive transition between them remains in the management of the user.

The Unity SRP and HDRP systems are the most similar to the proposed framework. Unity SRP has a low level, but Unity HDRP offers high level facilities similar to those proposed by the framework. However, it should be noted that due to lack of open sources, these systems may have limitations, such as the export and use of data to which there is only internal access.

Thus, the proposed framework offers facilities that none of the analyzed frameworks offer on their own. The closest, which offer capabilities for rapid implementation of real-time global illumination techniques and have at some level educational purposes, are NVIDIA Falcor [43] and Radeon Cauldron [44] frameworks. However, they offer a much lower level and the demonstration techniques they contain do not focus on real-time global illumination, but rather on rendering techniques in general.

6.2. Evaluation for Educational Purposes

We have made an evaluation of the possibility to use the proposed framework to learn real-time global illumination methods. More precisely, we used diffuse indirect illumination and glossy reflections, implemented with the class of techniques that use the information on the screen as a case study. As described in Section 5.1, we have implemented an incomplete skeleton of these two methods within the proposed framework, with clear indications on the actions that must be performed for the complete implementation of them. We also made an informative video about the theoretical description of the two methods. The code locations to be covered were diverse and included:

- The generation of a set of samples used to extract the VPLs on the screen;
- Accommodation with the framework architecture, in particular with the use of rendering passes and the management of rendering settings;

- Mathematical theory for the computation of diffuse indirect illumination;
- The ray cast process that uses the information on the screen.

We gave this skeleton to eight participants to implement the two techniques. All participants had a background in computer graphics. The evaluation results are visible in Table 3.

Table 3. The results of the evaluation for educational purposes.

Participant	Implementation Time	Implementation Correctness	Number of Times They Asked for Help
Participant 1	45 min	100%	0
Participant 2	45 min	100%	0
Participant 3	75 min	91%	1
Participant 4	23 min	100%	0
Participant 5	150 min	100%	0
Participant 6	33 min	100%	0
Participant 7	70 min	91%	0
Participant 8	75 min	91%	1

As can be seen, all of the participants managed to implement the two methods in a reasonably short time, with an average of 65 min. The resulted implementations were largely correct. It should be mentioned that all participants were provided with the final visual results of the two techniques, so all of them used these results for self-checking and obtained similar final visual results. For this reason, the resultant mistakes in implementation had subtle or non-existent effects on the final visual result and were related to the use of the parameters for the two techniques.

Most participants did not need help. However, it should be mentioned that the two participants who asked for help encountered technical problems with the build and execution of the framework for different development environments. Once the framework could be started, they no longer needed help.

From all this analysis and the experimental results obtained after evaluation, we conclude that the framework was used successfully in an evaluation close to the learning processes within a faculty. In fact, this was only a test for the use of the framework for educational purposes and we intend to use it in the near future for a larger number of participants.

7. Conclusions

In this paper, we have presented a framework for the development of real-time global illumination methods. This framework can be used for the rapid implementation of such techniques due to the use of a multi-pass architecture and the fact that there is an editor with GUI through which debugging information can be viewed and modified. Due to these aspects and the fact that the framework contains several implementations of real-time global illumination methods for demonstration purposes, this framework can also be used for the learning process of such methods.

We evaluated the proposed framework in two different ways. First, we performed a comparative analysis with a total of 13 other frameworks with similar applicability. From our analysis, it resulted that only two of them are close to the goals proposed by us, namely the use for learning and rapid implementation of real-time global illumination techniques. However, these frameworks were created for rendering methods in general, and none focus on lighting techniques in particular. The second evaluation was performed to test the use of the proposed framework for educational purposes. An incomplete implementation of two methods was created within the framework and was sent for complete implementation to eight participants with a background in computer graphics. They managed to use the framework successfully in order to implement the two methods. Thus, we have demonstrated the applicability of the framework both for teaching and for rapid implementation of real-time global illumination techniques.

This framework facilitates the rapid implementation of new methods, but the process requires a certain level of implementation. An option to further reduce the implementation time can be the development of a visual language to describe the rendering pipelines.

Supplementary Materials: The following supporting information (video) can be downloaded at: https://www.mdpi.com/article/10.3390/app12115654/s1. The source code of the framework can be accessed at: https://doi.org/10.5281/zenodo.6574191, accessed on 5 May 2022.

Author Contributions: Conceptualization, C.L., F.M., A.M. (Anca Morar), V.A. and A.M. (Alin Moldoveanu); methodology, C.L.; software, C.L.; writing—original draft preparation, C.L.; writing—review and editing, C.L., F.M., A.M. (Anca Morar), V.A. and A.M. (Alin Moldoveanu). All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Whitted, T. Origins of Global Illumination. *IEEE Comput. Graph. Appl.* **2020**, *40*, 20–27. [CrossRef]
48. oi: 10.1111/j.1467-8659.2012.02093.x Ritschel, T.; Dachsbacher, C.; Grosch, T.; Kautz, J. The State of the Art in Interactive Global Illumination. *Comput. Graph. Forum* **2012**, *31*, 160–188. [CrossRef]
3. Lambru, C.; Morar, A.; Moldoveanu, F.; Asavei, V.; Moldoveanu, A. Comparative Analysis of Real-Time Global Illumination Techniques in Current Game Engines. *IEEE Access* **2021**, *9*, 125158–125183. [CrossRef]
4. Ritschel, T.; Grosch, T.; Seidel, H.P. Approximating Dynamic Global Illumination in Image Space. In Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games, I3D'09, Boston, MA, USA, 27 February–1 March 2009; Association for Computing Machinery: New York, NY, USA, 2009; pp. 75–82. [CrossRef]
5. Sousa, T.; Kasyan, N.; Schulz, N. Secrets of CryENGINE 3 graphics technology. In Proceedings of the ACM SIGGRAPH 2011 Courses, Advances in Real-Time Rendering in 3D Graphics and Games, Vancouver, BC, Canada, 7–1 August 2011.
6. Dachsbacher, C.; Stamminger, M. Reflective Shadow Maps. In Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games, I3D'05, Washington, DC, USA, 3–6 April 2005; Association for Computing Machinery: New York, NY, USA, 2005; pp. 203–231. [CrossRef]
7. Kaplanyan, A.; Dachsbacher, C. Cascaded Light Propagation Volumes for Real-Time Indirect Illumination. In Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, I3D '10, Washington, DC, USA, 19–21 February 2010; Association for Computing Machinery: New York, NY, USA, 2010; pp. 99–107. [CrossRef]
8. Thiedemann, S.; Henrich, N.; Grosch, T.; Müller, S. Voxel-Based Global Illumination. In Proceedings of the Symposium on Interactive 3D Graphics and Games, I3D '11, San Francisco, CA, USA, 18–20 February 2011; Association for Computing Machinery: New York, NY, USA, 2011; pp. 103–110. [CrossRef]
9. Balreira, D.G.; Walter, M.; Fellner, D.W. *What We Are Teaching in Introduction to Computer Graphics*; EG 2017—Education Papers; Bourdin, J.J.; Shesh, A., Eds.; The Eurographics Association: Geneve, Switzerland, 2017. [CrossRef]
10. Suselo, T.; Wünsche, B.C.; Luxton-Reilly, A. Technologies and Tools to Support Teaching and Learning Computer Graphics: A Literature Review. In Proceedings of the Twenty-First Australasian Computing Education Conference, ACE '19, Sydney, NSW, Australia, 29–31 January 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 96–105. [CrossRef]
11. Unity Technologies. Unity. Available online https://unity.com/ (accessed on 16 May 2022).
12. Smith, G.; Sung, K. *Teaching Computer Graphics Based on a Commercial Product*; Eurographics 2019—Education Papers; Tarini, M.; Galin, E., Eds.; The Eurographics Association: Geneve, Switzerland, 2019. [CrossRef]
13. Vitsas, N.; Gkaravelis, A.; Vasilakis, A.A.; Vardis, K.; Papaioannou, G. *Rayground: An Online Educational Tool for Ray Tracing*; Eurographics 2020—Education Papers; Romero, M.; Sousa Santos, B., Eds.; The Eurographics Association: Geneve, Switzerland, : 2020. [CrossRef]
14. Slusallek, P.; Seidel, H.P. *Towards an Open Rendering Kernel for Image Synthesis*; Rendering Techniques '96; Pueyo, X.; Schröder, P., Eds.; Springer: Vienna, Austria, 1996; pp. 51–60.
15. Kirk, D.; Arvo, J.; Apollo Computer Inc. The Ray Tracing Kernel. In Proceedings of the Ausgraph, Melbourne, VIC, Australia, 4–8 July 1988; pp. 75–82.
16. Shirley, P.; Sung, K.; Brown, W. A ray tracing framework for global illumination systems. In Proceedings of the Graphics Interface, Calgary, AB, Canada, 3–7 June 1991; Volume 91, pp. 117–128.

17. Trumbore, B.; Lytle, W.; Greenberg, D.P. *A Testbed for Image Synthesis*; Eurographics Association, Geneve, Switzerland 1991; Volume 91, pp. 467–480.
18. Debattista, K.; Sundstedt, V.; Santos, L.P.; Chalmers, A. Selective Component-Based Rendering. In Proceedings of the 3rd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia, GRAPHITE '05, Dunedin, New Zealand, 29 November–2 December 2005; Association for Computing Machinery: New York, NY, USA, 2005; pp. 13–22. [CrossRef]
19. Dutre, P.; Bala, K.; Bekaert, P.; Shirley, P. *Advanced Global Illumination*; AK Peters Ltd.: Natick, MA, USA, 2006.
20. Ward, G.J. The RADIANCE Lighting Simulation and Rendering System. In Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '94, Orlando, FL, USA, 24–29 July 1994; Association for Computing Machinery: New York, NY, USA, 1994; pp. 459–472. [CrossRef]
21. Pharr, M.; Humphreys, G. *Physically Based Rendering: From Theory to Implementation*; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 2004.
22. Nimier-David, M.; Vicini, D.; Zeltner, T.; Jakob, W. Mitsuba 2: A Retargetable Forward and Inverse Renderer. *ACM Trans. Graph.* **2019**, *38*, 1–17. [CrossRef]
23. Dollner, J.; Hinrichs, K. A generic rendering system. *IEEE Trans. Vis. Comput. Graph.* **2002**, *8*, 99–118. [CrossRef]
24. Akenine-Moller, T.; Haines, E.; Hoffman, N. *Real-Time Rendering*, 3rd ed.; A. K. Peters, Ltd.: Natick, MA, USA, 2008.
25. Epic Games. Unreal Engine | The Most Powerful Real-Time 3D Creation Tool. Available online: https://www.unrealengine.com/ (accessed on 16 May 2022).
26. Epic Games. Lightmass Basics. Available online: https://docs.unrealengine.com/4.27/en-US/RenderingAndGraphics/Lightmass/Basics/ (accessed on 16 May 2022).
27. Epic Games. Lumen Global Illumination and Reflections. Available online: https://docs.unrealengine.com/5.0/en-US/lumen-global-illumination-and-reflections-in-unreal-engine/ (accessed on 16 May 2022).
28. Epic Games. Github—EpicGames/UnrealEngine: Unreal Engine Source Code. Available online: https://github.com/EpicGames/UnrealEngine (accessed on 16 May 2022).
29. Crytek. CRYENGINE | The Complete Solution for Next Generation Game Development by Crytek. Available online: https://www.cryengine.com/ (accessed on 16 May 2022).
30. Crytek. GitHub—CRYTEK/CRYENGINE: CRYENGINE is a Powerful Real-Time Game Development Platform Created by Crytek. Available online https://github.com/CRYTEK/CRYENGINE (accessed on 16 May 2022).
31. Unity Technologies. Unity—Manual: Using the Built-in Render Pipeline. Available online: https://docs.unity3d.com/Manual/built-in-render-pipeline.html (accessed on 16 May 2022).
32. Unity Technologies. Unity—Manual: Scriptable Render Pipeline Fundamentals. Available online: https://docs.unity3d.com/Manual/ScriptableRenderPipeline.html (accessed on 16 May 2022).
33. Unity Technologies. Universal Render Pipeline Overview | Universal RP | 11.0.0. Available online: https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@11.0/manual/index.html (accessed on 16 May 2022).
34. Unity Technologies. High Definition Render Pipeline Overview | High Definition RP | 14.0.2. Available online: https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@14.0/manual/index.html (accessed on 16 May 2022).
35. Unity Technologies. Render Graph | High Definition RP | 10.2.2. Available online: https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@10.2/manual/render-graph.html (accessed on 16 May 2022).
36. Godot Engine. Free and Open Source 2D and 3D Game Engine. Available online: https://godotengine.org/ (accessed on 16 May 2022).
37. Open 3D Engine. O3DE. Available online: https://www.o3de.org/ (accessed on 16 May 2022).
38. Blender Foundation. blender.org—Home of the Blender Project—Free and Open 3D Creation Software. Available online: https://www.blender.org/ (accessed on 16 May 2022).
39. NVIDIA. NVIDIA Iray Graphics Rendering. Available online: https://www.nvidia.com/en-us/design-visualization/iray/ (accessed on 17 May 2022).
40. NVIDIA. NVIDIA OptiX™ Ray Tracing Engine. Available online: https://developer.nvidia.com/rtx/ray-tracing/optix (accessed on 16 May 2022).
41. NVIDIA. NVIDIA RTX Global Illumination (RTXGI). Available online: https://developer.nvidia.com/rtx/ray-tracing/rtxgi (accessed on 16 May 2022).
42. NVIDIA. NVIDIA VXGI 2.0.1. Available online: https://developer.nvidia.com/vxgi (accessed on 16 May 2022).
43. NVIDIA. Falcor. Available online: https://developer.nvidia.com/falcor (accessed on 16 May 2022).
44. GPUOpen by AMD. Cauldron Framework. Available online: https://gpuopen.com/cauldron-framework/ (accessed on 16 May 2022).
45. Ocornut. GitHub–ocornut/imgui: Dear ImGui: Bloat-Free Graphical User Interface for C++ with Minimal Dependencies. Available online: https://github.com/ocornut/imgui (accessed on 17 May 2022).
46. Lambru, C.; Morar, A.; Moldoveanu, F.; Asavei, V.; Ivascu, S. Hybrid Global Illumination: A Novel Approach Combining Screen and Light Space Information. *Univ. Politeh. Buchar. Sci. Bull. Ser. Electr. Eng. Comput. Sci.* **2021**, *83*, 3–20.
47. McGuire, M.; Mara, M. Efficient GPU Screen-Space Ray Tracing. *J. Comput. Graph. Tech. (JCGT)* **2014**, *3*, 73–85.

48. oi: 10.1111/j.1467-8659.2011.02063.x Crassin, C.; Neyret, F.; Sainz, M.; Green, S.; Eisemann, E. Interactive Indirect Illumination Using Voxel Cone Tracing. *Comput. Graph. Forum* **2011**, *30*, 1921–1930. [CrossRef]
49. Saito, T.; Takahashi, T. Comprehensible Rendering of 3-D Shapes. *SIGGRAPH Comput. Graph.* **1990**, *24*, 197–206. [CrossRef]