Vertex Connection and Merging with Vulkan

Pedro Miguel Silva Rodrigues
pedro.m.rodrigues@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

November 2021

Abstract

Physically-based rendering algorithms are capable of producing high-quality images of virtual environments. Since its introduction, Vertex Connection and Merging (VCM) has proven to be a consistent, robust, and efficient algorithm. VCM takes advantage of two algorithms, Bidirectional Path Tracing (BDPT) and Photon Mapping (PM), to generate an image with the strong points of the two referred algorithms. The problem with these algorithms is the time they take when generating a photo-realistic image. A solution to this problem may be the use of the Graphics Processing Unit (GPU) instead of the Central Processing Unit (CPU). To implement this solution, we will use the Vulkan Ray Tracing extension, a low-level API that was developed to take advantage of the hardware support for ray tracing present in NVIDIA RTX graphic cards. This thesis intends to explore the implementation of Vertex Connection and Merging in GLSL and its conversion from the original implementation in the CPU to the GPU. To have a better understanding of VCM we also implemented separately Vertex Connection and Vertex Merging, so that we could analyse its impact on the final image.
Keywords: VCM, VUlkan RTX, GPU, Real-time Rendering

1. Introduction

Real-time ray tracing is experiencing an extensive and fast growth, with special hardware dedicated to it being developed by many companies in the field, from the manufacturers of game consoles to the companies specialized in GPU as Nvidia and AMD. Even with this technology, full real-time ray tracing is still difficult to achieve. The best algorithms used in ray-tracing trace many rays per pixel or many iterations of the same image to get the best result. This process is very demanding and complex, this way requiring a long runtime. So, the use of denoising algorithms to improve the image quality is becoming an industry standard. Using denoising, we can achieve the same or close results with less rays per pixel and less iterations. With the introduction of the NVIDIA RTX graphic card and the use of denoising algorithms, real-time ray tracing has become a possibility. However, the algorithms developed until now were optimized to run in the CPU, so it is important to port them and optimize them to the GPU and to find the algorithms best suited to real-time ray tracing while still delivering good photo-realistic results.

1.1. Objectives

The main objective of our project is to implement Vertex Connection and Merging and integrate it in the Lift [9] Framework, an educational interactive Stochastic Ray Tracing Framework with AI-Accelerated Denoiser, which uses the Vulkan ray tracing API in order to find out if it can achieve photo-realist results while keeping a real time performance and to compare it to other global illumination algorithms that do not achieve the same quality results but that are less complex. Furthermore, we also want to improve the Framework by implementing a better bidirectional scattering distribution function following the Phong model and another one capable of rendering microfacets.

2. Background

2.1. Ray Tracing

Photo-realistic rendering is the process of generating an image from a 3D scene description that is indistinguishable from a photograph of the same scene, employing techniques that model the interaction between light and matter using physics principles to simulate reality. The best method to achieve realistic results is ray tracing (RT). Turner Whitted introduced Ray tracing in 1980 [15], and he used the behaviour of light in the real world as inspiration. Where a light source emits photons, and when they hit an object, they can be reflected or refracted and may be absorbed by someone's eyes. However, instead of having a light source as origin, it originates from the camera, considering that the probability of a ray that originated on a light source hit the

camera is very low. Therefore, it is more efficient to start from the camera. Rays are cast from the camera to the scene, and when they hit an object, they can be reflected or refracted, originating more rays, and using next-even estimation, a shadow ray is cast in the direction of the light source, adding its contribution.

2.2. The Light Transport Equation

The light transport equation or the rendering equation, proposed by Kajiya [6], is a mathematical formula that describes the distribution of light in an equilibrium state. It determines the radiance leaving a point by calculating the sum of the light emitted in direction ω_o at point x, plus the radiance incident from all directions scattered in direction ω_o.

2.3. Monte Carlo integration

Monte Carlo integration is used to help evaluate the light transport equation. Solving the integral of the light transport equation is usually impossible due to the high dimension and frequent discontinuities. However, Monte Carlo integration solves this by using random sampling to estimate the values of integrals. This is done by independently sampling random points according to some probability density function (pdf) and then computing the estimate, turning the integral into a discrete sum.

$$F_n = \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)} \tag{1}$$

Where N is the number of samples, X_i is the random variable from which the samples are drawn, $f(x)$ is the function to be integrated, $p(x)$ is the probability density function. The major advantages of Monte Carlo integration are that, in order to estimate the value of $\int f(x)dx$, it only needs to be able to evaluate it at an arbitrary point in the domain, making it easy to implement. Additionally, it converges at a rate of $O(N^{-1/2})$ in any dimension, regardless of the continuity of the integral[11].

2.4. Real-time Rendering

Real-time rendering focuses on rapidly making images on the computer. As an image appears on the screen, the viewer acts or reacts, and this feedback affects what generated next. This cycle of rendering and reacting needs to happen at a fast enough rate that the viewer does not see still images. Although rasterization is still the most common and efficient way to achieve real-time rendering, ray tracing achieves the results with more realism. Ray tracing light transport methods, path tracing or derived methods as bidirectional path tracing or vertex connection and merging are usually favoured when implementing a real-time ray tracing renderer due to their scalability and algorithmic complexity. These methods work by solving the light transport equation[6], using Monte Carlo integration, so the more rays they send during the render process, the better the final result, but in real-time rendering there is a limited, finite time to render each image, which will result in noisy images, so a denoiser is usually helpful to improve the final result while keeping it fast and interactive.

2.5. Path Tracing

When Kajiya[6] proposed the rendering equation, he also proposed the Path Tracing method to try and solve it. The Path Tracing algorithm is an extension of Ray Tracing, so like Ray Tracing, it is an unbiased method. In this method, rays are traced originating from the camera to the scene: every time a ray intersects an object, the bidirectional scattering distribution function (BSDF) is taken into account. New rays are generated in a random direction, and trace the scene generating new rays if an object is intersected until they find a light source.

2.6. Light Tracing

Light tracing[2] preserves the same ideas as Path Tracing, with the big difference that the rays, instead of having the camera as their origin, have the light sources. Rays originate at the light sources, are traced to the scene, and when they intersect an object, a new ray is traced to the camera. The colour contribution is added to the pixel corresponding to the intersection between the ray and the image plane. Light tracing has some drawbacks considering that some pixels on the image plane may never be hit at all. Nonetheless, it is a very efficient method to find caustics and indirect illumination, which is difficult or impossible to find using Path Tracing due to the probability of sampling paths that contribute to these effects being proportional to their impact on the final image.

2.7. Multiple Importance Sampling

Multiple Importance Sampling (MIS) was introduced by Veach in 1995 [13] and is crucial for Monte Carlo methods. It allows combining samples from different techniques while minimizing the variance. To achieve this, it calculates the weight of each technique used by taking into account the probability density function (pdf) values of each technique. The estimator used:

$$I = \sum_{i=1}^{n} \frac{1}{n_i} \sum_{j=1}^{n_i} w_i(X_{i,j}) \frac{f(X_{i,j})}{p_i(X_{i,j})} \tag{2}$$

Where n is the number of sampling techniques, n_i the number of samples from technique i and w_i the weighting function. The weights have to sum up to one and the weighting function is usually the power heuristic.

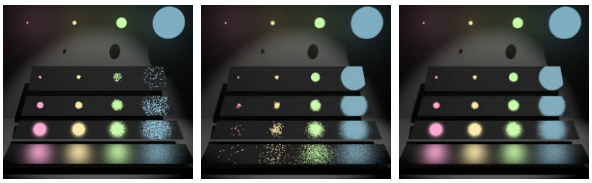


Figure 1: Sampling the light source, Sampling the BRDF and Sampling the MIS (Source:Veach 1995[13])

2.8. Bidirectional Path Tracing

Bidirectional Path Tracing[7] (BDPT) was developed to try and take advantage of the benefits of Path Tracing and Light Tracing without any of the drawbacks. Just as in these two methods, paths are traced originating from both the camera and the light sources, forming the camera path and the light path, respectively. Both paths are initiated simultaneously. At each iteration, the camera path vertices are connected to the vertices of the light path. Then using Multiple Importance Sampling (MIS)[13] each path contribution is added to determine the resulting colour. Even though BDPT has the advantages of both Path Tracing and Light Tracing, there are still paths that present difficulties. More precisely, specular-diffuse-specular (SDS) paths are very challenging to BPT and other methods because they are hard to find but have a high impact on the final image.

2.9. Photon Mapping

Photon Mapping[5] is a biased two-pass method that, contrasting to BDPT, is very effective at capturing SDS paths. In the first pass, photons are shot originating from the light sources and stored on a photon map when they intersect an object. The second pass is identical to Path Tracing. Paths are traced starting at the camera. Every time it intersects an object, the photon map is accessed to compute an estimation of photon density around the intersection point. From this estimation a bias is introduced.

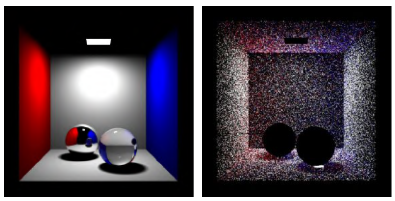


Figure 2: A ray traced image (left). The photons in the corresponding photon map (right). (Source: H. W. Jensen 1996[5])

2.10. Metropolis Light Transport

Metropolis Light Transport[12] is a method of the Markov chain Monte Carlo (MCMC) family, proposed to improve Path Tracing methods by mutating already found paths that carry light. Each new mutated path depends only on the previous path, forming a Markov chain. The mutation can be applied by adding, deleting, or replacing a small set of vertices on the current path. The resulting path still needs to be accepted, so, to improve the efficiency of this method, the mutation strategies need to be carefully designed to guarantee that it not only generates valid paths but that these paths are also significantly different, thus making the implementation of Metropolis Light Transport quite challenging.

Figure 3: MLT (left), BPT (middle) and PPM(right) outputs of the same scene.(Source:Georgiev 2012 [4])

Figure 3 clearly shows that MLT finds more and better light paths than BDPT. However, sometimes it can get stuck in an area leading to brighter zones, as we can see in the front mirror of the car.

3. Vertex Connection and Merging

Vertex Connection and Merging[4] (VCM) is a two-pass algorithm developed to try and take advantage of the benefits of photon mapping and bidirectional path tracing, keeping the high asymptotic performance from BDPT for most light path types while having PM efficiency to reproduce specular-diffuse-specular lightning effects. In the first stage, it traces the light paths, connects them to the camera, and stores the light vertices in a hash grid. In the second stage, it performs three techniques and, using multiple importance sampling, adds their contributions to the pixel resulting colour. It starts by tracing a camera path to each pixel and then, for each node in the camera path it connects them to the light source, it connects them to the light vertices processed in the first stage and it merges them with the light vertices within a predefined merge radius. VCM can be implemented in an iterative manner where for every iteration the merge radius is reduced, decreasing the bias introduced in the final image.

Figure 4: Comparison between VCM, BDPT and PM results of the same scene. (Source:Georgiev [4])

As we have discussed, BDPT suffers with specular-diffuse-specular paths and PM with diffuse illumination. The goal of Vertex Connection and Merging is to use BDPT to deal with diffuse illumination and to use PM to find SDS paths. This combination was thought to be impossible, but as we can see in Figure 4, VCM successfully combined both techniques. This was only possible thanks to a reformulation of the process to calculate the MIS weight of each path by Gerogiev [3].

4. Vertex Connection and Merging

This section consists of a more detailed analysis of Vertex Connection and Merging algorithm and how it calculates the MIS weights for each type of path.

4.1. Algorithm Overview

Listing 1: VCM algorithm.

```
VCMRenderer (){
 lightPaths [numPixels];
 for (int i = 1; i <= numPixels ; i++){
  LightNode lNode = TraceRay(Pixel(i));
  while ( lNode.isValid() ){
   if ( !lNode.Specular ){
    lightPaths.storeLighNode(lNode);
    ConnectToCamera(lNode);
   }
   lNode = SampleScatter();
  }
 }
 BuildRangeStructure(lightPaths);
 Color color;
 for (j = 1; j <= numPixels ; j++){
  CameraNode cNode = TraceRay(i);
  while ( cNode.isValid() ){
   if(emissive material){
    color += cNode.color * GetLight();
    break;
   }
   if(not specular) {
    color += ConnectVertices();
    color += MergeVertices();
   }
   cNode = SampleScatter();
  }
 }
}
```

Every modern global illumination algorithm has the objective of efficiently solving the formulated light transport equation. Veach[11] formulated light transport over a three dimensional scene as a pure integration problem. The integral is solved by a process called Monte Carlo integration. The final colour of each pixel is expressed as different integration problems. The values estimated by algorithms like PT, LT, BDPT or VCM are all approximations of the real value of the integral, and have some bias (the difference between the estimated value and the real value of the integral). The algorithms that have less bias, called unbiased, are usually favoured over the others, since they provide more accurate results.

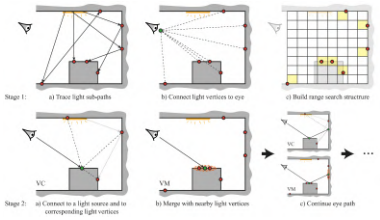


Figure 5: VCM overview. (Source:Georgiev 2012 [4])

Vertex Connection and Merging is the combination of BDPT and PM, so it uses techniques from both algorithms. Like BDPT and PM, its first pass is light tracing. It traces light paths from the light sources to the scene, connects the light nodes to the camera and stores the light nodes in a range structure. The second pass of VCM is where all computations to estimate the pixel final colour are. It starts by shooting a ray from the camera to the scene and every time it intersects an object, it accesses the stored light nodes to calculate the final colour using two different techniques, Vertex Connection and Vertex Merging. Vertex Connection (Stage 2 a) in Figure 5), connects the camera vertex to all visible light nodes, just like in BDPT. This technique allows VCM to find paths that carry light in a fast and efficient way, easily dealing with diffuse lighting. Vertex Merging (Stage 2 b) in Figure 5), merges the camera vertex with all light nodes within merging range, just like PM. With a high number of light nodes it is important to use a range search structure to speed up this process. This technique is capable of finding SDS paths easily. By calculating the photon density around the camera vertex, it increases the weight of these kind of paths that are very hard to find but that have a big impact on the final result.

5. Implementation

We used SmallVCM as base to develop and integrate VCM in in Lift, a framework developed by Gonçalo Soares [9], that uses the Vulkan API and

its ray tracing extension (VKRay) to take advantage of the RTX graphic card family.

5.1. Vulkan

Vulkan is a standard API that provides high-efficiency, cross-platform access to modern GPUs. It is hardware-agnostic, meaning that it does not require any special hardware adaptations nor its extension VKRay. One important feature we used in this thesis is shader programming. In our solution, we used GLSL to implement VC, VM and VCM. In recent years, Vulkan has taken the place of OpenGL as the industry standard for 3D graphics, improving on its predecessor by providing more control to the programmer. Like OpenGL, Vulkan works around the concept of handles, where VulkanObjects are recognized at the API level as unique IDs passed by the drivers. Vulkan is located at a lower-level than OpenGL, which means it can use pointers to the hardware memory buffers being able to access VRAM from the CPU side.

5.1.1 Vulkan Ray-Tracing extension

Vulkan Ray-Tracing pipeline is different from the traditional rasterization pipeline. The rasterization pipeline contains two programmable shaders, the Vertex shader and Fragment shader. The Vertex shader is responsible for identifying the triangle primitive being drawn and pass it to the Fragment shader where the primitive colour will be calculated. While in the ray-tracing pipeline there are five programmable shaders, every shader on the pipeline must be available for execution at any time, and the one executed is chosen at runtime. The five different types of ray-tracing shaders are: Ray Generation, Intersection, Any Hit, Closest Hit, Miss.

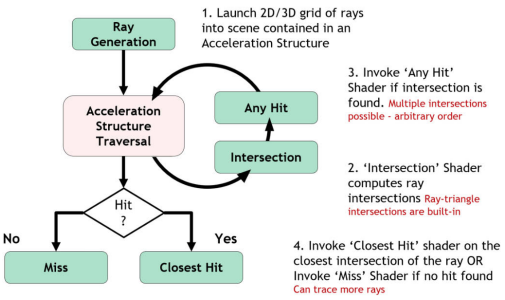


Figure 6: Vulkan Ray tracing Shaders domains and relationships.

Ray-Tracing requires testing each ray against all scene primitives, a slow and expensive process. So Vulkan uses acceleration structures that store the geometric information of the primitives in the scene necessary to render, spatially sorted from the camera location, in order to reject potential intersections in a fast and straightforward way. VKRay uses acceleration structures modelled as a two-level structure composed by the bottom level structures and the top level structures. Bottom level acceleration structures contain geometry data of the objects in the scene, while Top level acceleration structures contain a list of references to bottom level nodes. Bottom level nodes are stored in a tree where its root is a Top level node representing an object, and every leaf is a Bottom level node representing the primitives that constitute that object.

5.2. Lift Framework

The implementation of VCM algorithm was integrated in the Lift framework developed by Gonçalo Soares[9]. Lift was designed around a rendering architecture with progressive refinement that allows the choice of one of the two light transport techniques, PT or BDPT, the selection of the input scene and the enabling of a feature to denoise every frame being generated or just the last one [10].

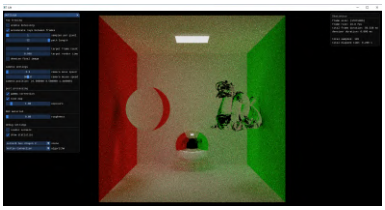


Figure 7: Lift Framework.

Lift, as we can see in Figure 7, gives the user several options: It has the option to activate or deactivate the denoiser and to accumulate rays between frames; It gives the user the option to define a target number of accumulated samples or a desired rendering elapsed time, which can be used to evaluate the different algorithms integrated in the framework. The algorithms integrated are Path Tracing and Bidirectional Path Tracing. We integrated three more algorithms: Vertex Connection, Vertex Connection and Merging and an experimental implementation of Vertex Merging to show its impact in VCM. To compare these global illumination algorithms, Lift provides a different set of statistics: Frame Size, Frame Rate, Total Frame Duration, Denoiser Duration, Total Samples and Total Elapsed Time. Lift also lets the user change the camera settings, the camera move speed and the mouse move speed.It has some post-processing features that can be activated or deactivated, Gamma Correction and Tone Map and lets the user change the exposure level of the scene. We changed the BRDF used in the framework and added Phong shading and GGX materials. To let the users understand the impact of the roughness value in GGX materials, we added the option so that users could change the roughness value of this type of materials. To add value to Lift,

we also added more scenes to the list with different characteristics: "Cornell Box Dragon 2", the scene shown in Figure 7, that shows the impact of diffuse lighting and SDS paths; "Veach Ajar", a scene where the only light source is out of the scene and can only be accessed through a semi-closed door; "Gold Ball", the scene shown in Figure 8 that has as main element a goldball rendered using GGX materials.

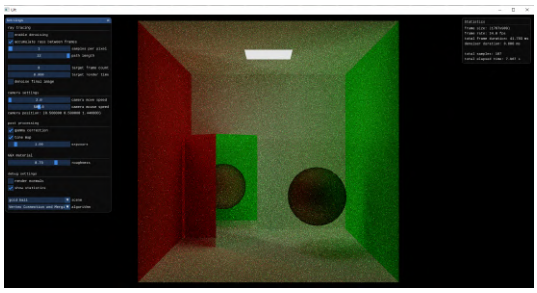


Figure 8: Gold ball with GGX material.

5.3. SmallVCM

Our solution uses SmallVCM, the implementation given by the authors of VCM [4] as a starting point. However their implementation was CPU based so we needed to port it and adapt it to GPU. SmallVCM was developed in C++ and uses OpenMP to concurrently render many frames all with different merging radius.

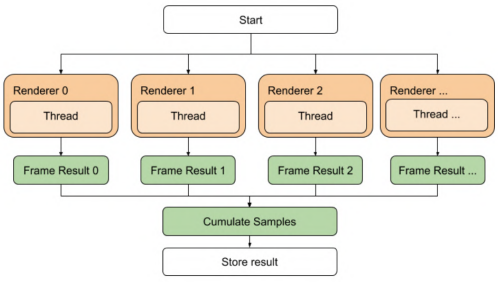
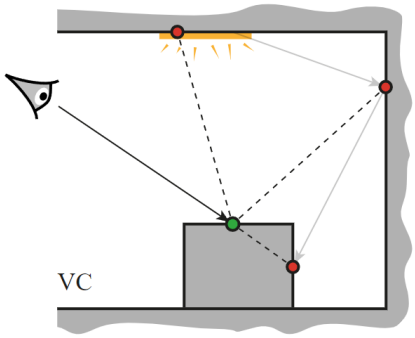


Figure 9: SmallVCM parallel execution model.

Vulkan initiates the ray-tracing rendering by sending a 2D grid of rays from the camera plane to scene, so our shaders must be developed taking in mind the camera path instead of the calculation of the frame. This introduces some complications, namely if and how to reduce the merge radius and how to handle the light path. Since our solution needs to be integrated in Lift framework that has the goal of being able to change the scene being rendered and the algorithm being used without the need to recompile it, we implemented the light path in a non optimized way but that can assure the goal of the framework.

5.4. Vertex Connection

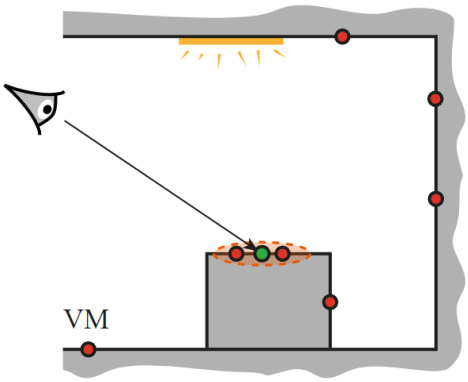
As we have seen, in the Vulkan ray-tracing pipeline, there are five different types of programmable shaders, all important when implementing a global illumination algorithm. In our implementation of Vertex Connection, a variant of bidirectional path tracing[7] implemented in SmallVCM using the recursive formulation developed by Georgiev [3], we only had to develop the Ray Generation shaders, as the other shaders had already been developed in a simple and efficient way that allowed them to be used in different algorithms. The Vertex Connection implementation in SmallVCM starts like Vertex Merging and Vertex Connection and Merging., by shooting rays from the light source to the scene and storing the resulting light vertices. However, Vulkan Ray Generation shader, the starting point of the VKRay pipeline, sends a ray from the camera to a pixel. So, we cannot generate all light paths beforehand. To solve this problem, we start the ray generation shader by shooting a ray from the light source to the scene in a random direction and storing the vertices along its path, which leads to a lower number of light vertices available when connecting light and camera vertices. But the possibility to accumulate rays between frames provided by Lift mitigates this problem.



Stage 2: a) Connect to a light source and to corresponding light vertices

5.5. Vertex Merging

Vertex Merging is a variant of the algorithm Photon Mapping [5] implemented in SmallVCM using the recursive formulation developed by Georgiev [3]. It is used to calculate the photon density of a scene. Our implementation of VM uses a small number of light paths to calculate the final colour of each pixel, because of the way the Vulkan API functions, namely setting different render contexts for each pixel.



b) Merge with nearby light vertices

5.6. Vertex Connection and Merging

Vertex Connection and Merging is the result of combining the previous two algorithms, using multiple importance sampling (MIS) in order to retain the best qualities of each algorithm. This is the reason why implementing VCM and integrating it in Lift [9] is the goal of this thesis, since VCM combines the best of Bidirectional Path Tracing and Photon Mapping, something that until its introduction by Georgiev [4] was thought to be impossible.

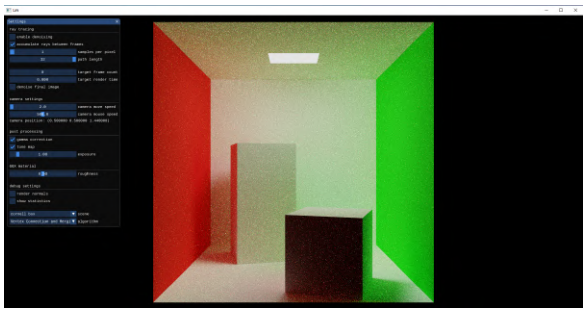


Figure 10: VCM render of Cornell Box.

To implement VCM we had to use the light tracing component of VC. Create the light paths and store its vertices along the way and their sub-vertex data to calculate the MIS weight of each path. The contributions of the techniques VC and VM are added to the pixel colour during the camera tracing loop. At every non-specular intersection we connect the camera vertex to the stored light vertices and merge the camera vertex with the light vertives within range.

5.7. Phong Shading

BSDF can be decomposed into two components, BRDF bidirectional reflectance distribution function when referring only to the reflected component and BTDF bidirectional transmittance distribution function, referring to the scattered transmitted through a material.

5.8. Sampling Multiple Lobes BSDF

Most of the time, when light intercepts a surface, it produces scattering with multiple different characteristics at the same time. When light intersects a glass like object, it showcases reflection and refraction simultaneously with different weights for each component. The way light interacts with surfaces can be divided into different components, which are generally called lobes: Diffuse, Specular, Reflection and Refraction. VCM functions using path tracing and light tracing, meaning that every time a path intercepts an object, it can only create one new path instead of several paths, creating a tree where paths divide themselves at each intersection. To ensure that only one new path is created, it is calculated the probability of each lobe for every material, taking into consideration the contribution of each lobe to the final result.

5.9. Lobes and Shading Models

The Phong BSDF is composed by a Lambertian term for diffuse reflection plus the Phong specular term for glossy reflection:

$$f_r(\omega_i, \omega_o) = \frac{\rho_D}{\pi} + \rho_S \frac{\alpha + 2}{2\pi}(\omega_o \cdot R)^\alpha \qquad (3)$$

To sample the diffuse term the importance distribution used is a cosine-weighted distribution, while the one used to sample the specular term is derived from its expression. It does not use the same distribution because of the spiked shape of the glossy term.

5.10. Cook-Torrance BSDF

Cook-Torrance BSDF [1], just like the BSDF function we have just explained, can be divided into different components. The difference between both BSDF function lies in the Specular component. Cook-Torrance BSDF is capable of rendering microfacet models that describe the roughness of surfaces as a compilation of small microfacets, very small perfect reflectors described by three different functions D, F and G.

$$f_{specular} = \frac{DFG}{4(\omega_o \cdot n)(\omega_i \cdot n)} \qquad (4)$$

D is the distribution function. It statistically describes the amount of microfacets that are aligned with the halfway vector, which is the vector between the surface normal and the light direction. There are many distribution functions that can be used to calculate D and we used the Trowbridge-Reitz GGX [14] distribution function:

$$D(n, h, \alpha) = \frac{\alpha^2}{\pi((n \cdot h)^2(\alpha^2 - 1) + 1)^2} \qquad (5)$$

Where α is the roughness of the surface and $0 \leq \alpha \leq 1$.

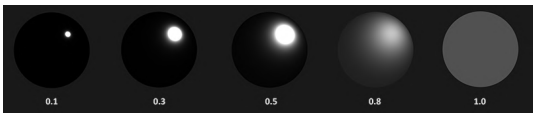


Figure 11: GGX Distribution. Roughness values.

When the surface is smooth, its roughness value us closer to zero, which means that there is a high number of microfacets concentrated in a small radius pointing to the halfway vector. And as the roughness increases, so does the number of microfacets pointing to the halfway vector. However, they are much more dispersed, leading to a greyish result. The geometry function, G, is used to describe how the microfacets shadow each other, leading to an attenuation of the light. It calculates the probability of a given point in a surface being shadowed by the surrounding microfacets. The geometry function we will use is derived from the distribution function:

$$G_{GGX}(n,h,k) = \frac{n \cdot v}{(n \cdot v)(1-k)+k} \quad (6)$$

$$k = \frac{(\alpha+1)^2}{8} \quad (7)$$

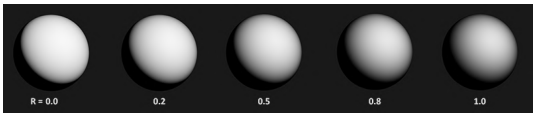


Figure 12: GGX Geometry. Roughness values.

The geometry function returns a value between [0,1] and as we can see, the higher the roughness of the surface, the lower the value returned from the geometry function. This is because the surfaces with higher values of roughness will have more microfacets shadowing each other, leading to darker colours. The fresnel function, F, simulates the way light interacts with a surface at different angles, calculating the percentage of light that gets reflected. Every material has a base level of reflectivity when looked directly upon. But when looking at it from different angles, its reflections become more apparent. Theoretically all surfaces reflect light when seen from a 90 degree angle. This is called the fresnel effect. However, the fresnel equations are very complex and they differ from conductive to dielectrics materials so we will use an approximation introduced by Shlick [8]:

$$F_{Shlick}(h,v,F_0) = F_0 + (1-F_0)(1-(h \cdot v)^5) \quad (8)$$

Where F_0 represents the base reflectivity, and is calculated using the index of refraction (IOR) of the medias present at the intersection point.

6. Evaluation

To evaluate if our implementation of VCM improves on the already developed algorithms, PT and BDPT we needed to evaluate the quality of its output. We also evaluated its performance to confirm if it can keep a real-time performance.

6.1. Evaluation Methodology

To evaluate the algorithms implemented, VC and VCM, and compare them with the algorithms already developed and integrated in Lift, PT and BDPT, we use a set of scenes with different characteristics such as reflected caustics, low light and difficult to reach light sources. All the results were obtained in the same hardware.

6.2. NVIDIA Nsight Graphics

NVIDIA Nsight Graphics is a powerful tool used to debug, profile and export frames from applications that use the Vulkan API, among others. It is a complex tool with many configurations: Frame Debugger, Frame Profiler, Generate C++ Capture, GPU Trace and System trace (Figure 13).

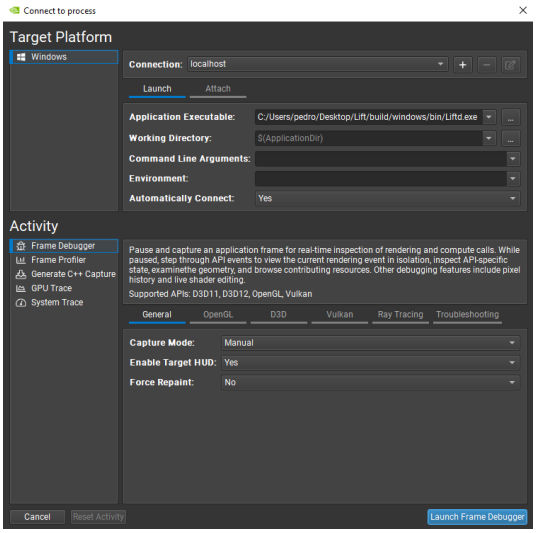


Figure 13: Nsight Graphics launch menu.

We used the options Frame Profiler and GPU Trace to analyze the performance of our implementation and compare it to other algorithms in Lift. Frame Profiler gives the option to analyze the behaviour of our implementation and of the hardware when rendering one frame. As we can see from Figure **??**, Lift is very efficient and after ending the Render Pass, it was rendering the next frame in less than one millisecond. Since we had the opportunity, we also used the option to profile the shaders and evaluate the time distribution between them when rendering one frame. From Figure **??**, we can confirm what we have discussed in Section 5.2. When rendering a frame, 49% of the time is spent in the

Ray Generation shader. This was expected as the complex calculations to find the pixel final colour are all done in this shader. We also did this evaluation with VCM to understand the impact of its higher complexity, and as we can see in VCM, the Ray Generation shader takes even a bigger percentage of the time.

6.2.1 Structural Dissimilarity

Structure Dissimilarity Index Metric (DSSIM) is an image quality metric that compares the differences between two images. It returns a positive value and the closer the images look alike, the closer its value is to 0, and if it returns 0, it means that there are no differences in the two images. DSSIM is calculated using SSIM, $DSSIM = 1/SSIM - 1$.

6.2.2 Peak Signal-To-Noise Ratio

Image quality can be subjective, and so Peak Signal-To-Noise Ratio (PSNR) works as an approximation to the human perception of reconstruction quality. PSNR is used to measure the noise in a distorted image when compared to the reference image. It can be used to evaluate compression or rendering algorithms. The higher the value it returns, the closer the distorted image is to the reference one. For an 8-bit image, values closer to 50db mean that there is almost no noise in the rendered image.

6.2.3 Frames per Second

This metric evaluates the number of frames rendered per second. We will calculate the average and calculate how much time it took to render each frame. This metric is important to evaluate the responsiveness of our implementation and if it can keep a real-time performance perceived as a minimum of 30 frames per second.

6.2.4 Time to Converge

This metric will be used in a static scene to measure how the quality of the output of each algorithm evolves and how long it takes to converge to the reference image. We will also evaluate how the quality of the image evolves per frame rendered. This way we can evaluate the algorithms that need less

7. Test Scenes

To test the performance and the quality of our solution we have chosen scenes with different levels of complexity and different characteristics.

7.0.1 Japanese Classroom

These results confirm how efficiently VCM deals with diffuse lighting. In spite of the fact that BDPT is an algorithm that also deals deals well with diffuse lighting, it is clear that the implementation integrated in Lift does not reach the expected results, achieving worse results than VC. Even though VM adds complexity to VCMin scenes with no caustics and where the low number of light paths leads to a reduced impact by VM,VCM still keeps up with VC, getting the same DSSIM over time.

Figure 14: VCM classroom render after 4096 frames.

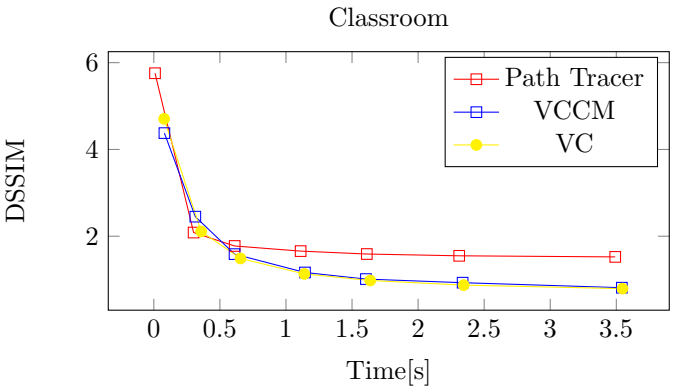


Figure 15: Japanese Classroom DSSIM over time

7.0.2 Dining Room

This scene is composed almost exclusively of diffuse materials and the only light in the scene is coming from a semi closed window. VCM is able to find difficult to reach light carrying paths easier than PT. However, in this scene PT is still able to outperform BDPT and VCM in time and iterations needed to converge to the solution.

Figure 16: PT dining room render after 4096 frames.

Table 1: DSSIM evolution over frames rendered.

Implementation	Frames	DSSIM	PSNR
PT	1	1.477086946	17.34
PT	256	0.669449082	27.43
PT	1024	0.286339079	32.22
PT	4096	0.083306251	38.68
VCM	1	1.341920375	17.77
VCM	256	0.674761347	24.56
VCM	1024	0.366493577	25.9
VCM	4096	0.231982259	26.49

8. Conclusions

With the results obtained during this thesis we can conclude that for more complex scenes Vertex Connection and Merging will achieve better results than Path Tracing and Bidirectional Path Tracing. However, those are the same scenes where it canot keep a real-time performance. It would be interesting to evaluate this implementation with the new NVIDIA Geforce RTX 3080 to see if this implementation is capable of keeping a real-time performance with the best hardware available today. It would be important in the future to try and reformulate the Framework to be able to construct the light paths before starting the rendering process. It would reduce the rendering time of BDPT, VCM and VC. This way, all light vertices would be available during the camera tracing process, which would lead to better final results. However it would also be necessary to use a hash grid, as a range search structure, to identify the light vertices that are within range of the camera vertex. With this improvement, it would be easy to implement a Progressive Photon Mapping [5], which would add even more value to Lift.

Acknowledgements

I would like to thank Instituto Superior Técnico for the high quality education provided to me, and in particular to my thesis supervisor for always being available to help and support me during this process.

References

[1] R. L. Cook and K. E. Torrance. A reflectance model for computer graphics. *ACM Trans. Graph.*, 1:7–24, 1982.

[2] Ph. Dutré, E. P. Lafortune, and Y. D. Willems. Monte carlo light tracing with direct computation of pixel intensities. In *3rd International Conference on Computational Graphics and Visualisation Techniques*, pages 128–137, Alvor, Portugal, December 1993.

[3] I. Georgiev. Implementing vertex connection and merging. Technical report, Saarland University, 2012.

[4] I. Georgiev, J. Křivánek, T. Davidovič, and P. Slusallek. Light transport simulation with vertex connection and merging. *ACM Trans. Graph.*, 31(6), Nov. 2012.

[5] H. W. Jensen. Global illumination using photon maps. In X. Pueyo and P. Schröder, editors, *Rendering Techniques '96*, pages 21–30, Vienna, 1996. Springer Vienna.

[6] J. T. Kajiya. The rendering equation. *ACM Siggraph Computer Graphics*, 20(4):143–150, Aug. 1986. doi:10.1145/15886.15902.

[7] E. P. Lafortune and Y. D. Willems. Bidirectional path tracing. In *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics '93)*, pages 145–153, Alvor, Portugal, December 1993.

[8] C. Schlick. An inexpensive brdf model for physically-based rendering. *Computer Graphics Forum*, 13(3):233–246, 1994.

[9] G. Soares. Interactive physics-based rendering with ai-accelerated denoiser. Master's thesis, Instituto Superior Técnico, 2020.

[10] G. Soares and J. M. Pereira. Lift: An educational interactive stochastic ray tracing framework with ai-accelerated denoiser. *WSCG 2021: full papers proceedings: 29*, pages 325–334, 2021.

[11] E. Veach. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford, CA, USA, 1998. AAI9837162.

[12] E. Veach and L. Guibas. Metropolis light transport. *Computer Graphics (SIGGRAPH '97 Proceedings)*, 31, 02 1970.

[13] E. Veach and L. J. Guibas. Optimally combining sampling techniques for monte carlo rendering. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, page 419–428, New York, NY, USA, 1995. Association for Computing Machinery.

[14] B. Walter, S. R. Marschner, H. Li, and K. E. Torrance. Microfacet models for refraction through rough surfaces. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques*, EGSR'07, page 195–206, Goslar, DEU, 2007. Eurographics Association.

[15] T. Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23(6):343–349, June 1980.