



UNIVERSITAT DE
BARCELONA

Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

**Facultat de Matemàtiques i Informàtica
Universitat de Barcelona**

**TECHNIQUES FOR EFFICIENT GLOBAL
ILLUMINATION THROUGH VIRTUAL POINT
LIGHTS**

Pol Baylina Naval

Director: Dr. Ricardo Marques

Realitzat a: Departament de

Matemàtiques i Informàtica

Barcelona, 13 de juny de 2022

CONTENTS

Resum	3
Resumen	5
Abstract	7
Section 1 - Introduction	9
1.1 - Motivation	10
1.2 - Objectives	10
1.3 - Planification	11
1.4 - Thesis structure	14
Section 2 - State of the Art	15
2.1 - Global illumination	15
Section 3 - Efficient Global Illumination Through Virtual Point Lights	19
3.1 - Virtual Point Lights Algorithm	19
3.1.1 - The importance of Clamping in VPLs-based Algorithms	23
3.3 - VPL-based image synthesis with clustering of visible points	24
3.3.1 - K-means clustering explained in general context	24
3.3.2 - K-means clustering of visible points	26
3.3.3 - Synthesizing an image with visible points clustering	28
3.4 - VPL-based image synthesis with clustering of VPLs	30
3.4.1 - K-means clustering of VPLs	30
Step 4: Recompute the centroids of newly formed clusters	30
3.4.2 - Synthesizing an image with Virtual Point Lights clustering	32
3.5 - VPL-based image synthesis with visible points and VPLs clustering	33
3.5.1 - Synthesizing an image with visible points and VPL clustering	33
Section 4 - Implementation	35
4.1 - Visual Studio	35
4.2 - Base Ray tracer	35
4.3 - Changes done in the base ray tracer	37
Section 5 - Results	39
5.1 - Experimental Set-Up	39
5.2 - Results for the Original VPL Algorithm Implementation	41
5.3 - Results using VPL clusters	44
5.4 - Results using visual point clusters	47
5.5 - Results using VPL clusters and Visible point clusters	51
Section 6 - Discussion	53
6.1 Limitations	53
6.2 Future work	53
6.3 Conclusions	54
Section 7 - Bibliography	56

List of Figures and Tables

Figure 1.3.1 - TFG Gantt chart	12
Table 1.3.2: Gantt chart explanation	13
Figure 2.1: Image demonstrating several global illumination effect.....	15
Figure 2.3.1: Creation and rendering of VPLs.....	18
Figure 2.3.2: Illustration of singularities.....	19
Figure 3.1.0: Difference between amount of VPL.....	21
Figure 3.1.1: How two VPL of 2 bounces each are created.....	22
Figure 3.1.2: Rendering phase.....	22
Figure 3.1.3: Illustration of singularities or artifacts solving.....	23
Figure 3.3.1: Demonstration of standard algorithm.....	24
Figure 3.3.2: Clustering of the visible points.....	26
Figure 3.4.2: Clustering of the visible points or intersection	30
Figure 3.5.3: Clustering VPLs.....	31
Figure 5.1.1: Reference Image.....	35
Figure 5.1.2: Legend.....	36
Figure 5.2.1: Scenes resulting from the tests, using only VPLs.....	38
Figure 5.2.2: Graph of rendering VPLs time.....	39
Figure 5.3.1: Scenes resulting from the tests, using only VPL clusterization.....	41
Figure 5.3.2: Image comparing	42
Figure 5.4.1: Scenes resulting from the tests, using VPL, and visible point clustering	44
Table 5.4.2: Time table.....	45
Figure 5.5.1: Results using VPLs clusters and Visible Point clusters.....	46

Resum

La il·luminació global és la tasca que pretén afegir realisme al modelatge de la llum en escenes 3D, tot això engloba un conjunt d'algorismes que ho fan possible. Aquests algorismes tenen en compte, a més de la llum que prové directament d'una font de llum (il·luminació directa), els raigs de llum que provenen de la mateixa font però han passat per reflexos sobre superfícies de l'escena (reflectants o no) (il·luminació indirecta) . Així doncs, en informàtica és un terme més complex del que sembla a priori, no només es refereix a les fonts de llum, sinó a totes les condicions d'il·luminació de l'escena, és a dir, que hem de tenir en compte tots els objectes de l'escena ja que cadascú influirà d'una manera o altra als altres amb la llum que reflecteix, refracta o absorbeix. Per ser més precisos per calcular l'IG necessitem una descripció de l'escena virtual que inclou la posició, la mida i l'orientació dels objectes geomètrics 3D que componen l'escena, el material associat a cada objecte, la posició i les característiques de les fonts de llum que il·lumina, l'escena, i una càmera virtual que defineix com es veu una escena. Un cop definit això, cal calcular/simular com s'emet la llum de les fonts de llum i rebota a l'escena fins que arriba al pla de la imatge. La llum que arriba a cada píxel de la imatge definirà el color del píxel.

Per calcular tota aquesta il·luminació necessitem una capacitat de còmput molt alta si volem fer-ho en un temps raonable, o uns algorismes que optimitzin aquest càlcul, i en aquesta tesi és del què tractarem .

Començarem parlant de l'ús de l'algorisme Virtual Point Lights, també anomenat per alguns autors Instant Radiosity. Aquest algorisme és capaç de generar imatges de qualitat comparable a la de, per exemple, Path Tracing (que és un algorisme estàndard en renderitzat fotorealista). -Explicarem com funciona l'algorisme VPLs i com és capaç de renderitzar una imatge en menys temps que el de Path Tracing i explicarem com es generen les imatges. A més, també discutirem els artefactes que es generen usant l'algorisme de Virtual Point Lights i com resoldre'ls.

El nucli d'aquest treball és l'acceleració de l'algorisme VPL original. Per això recorrent a tècniques de clustering, que ens permetran reduir dràsticament el nombre de càlculs implicats. En particular, proporcionarem detalls sobre la implementació de l'algorisme K-means (un tipus d'aprenentatge no supervisat) i la seva aplicació al context de la síntesi d'imatges basada en VPL. Mostrarem com es poden fer servir les mitjanes K per agrupar els punts visibles de cada píxel i també per agrupar els punts de llum virtuals.

Proporcionarem resultats detallats utilitzant els quatre algorismes diferents: l'algorisme VPL original, que hem implementat des de zero; VPL amb K-means clustering dels punts visibles; VPL utilitzant K-means clustering de VPLs; i, finalment,

els resultats en utilitzar tant el K-means clustering dels punts visibles com dels VPL. Els resultats mostren millores clares en el temps de síntesi d'imatges.

Finalment, presentarem els resultats obtinguts des de la perspectiva de la qualitat d'imatge final. Identificarem la principal limitació dels mètodes desenvolupats i proposarem millores per a treballs futurs.

Resumen

La iluminación global es la tarea que pretende agregar realismo al modelado de la luz en escenas 3D, todo esto engloba un conjunto de algoritmos que lo hacen posible. Estos algoritmos tienen en cuenta, además de la luz que proviene directamente de una fuente de luz (iluminación directa), los rayos de luz que provienen de la misma fuente pero que han sido reflejados en superficies de la escena (reflexivas o no) (iluminación indirecta). Entonces esto en informática es un término más complejo de lo que parece a priori, se refiere no sólo a las fuentes de luz, sino a todas las condiciones de iluminación de la escena, es decir, que tenemos que tener en cuenta todos los objetos de la escena ya que cada uno va a influir de un modo u otro a los demás con la luz que el mismo refleja, refracta o absorbe. Para ser más precisos para calcular GI necesitamos una descripción de la escena virtual que incluya la posición, tamaño y orientación de los objetos geométricos 3D que componen la escena, el material asociado a cada objeto, la posición y características de las fuentes de luz que iluminan la escena, y una cámara virtual que define cómo se ve una escena. Una vez que se define esto, es necesario calcular/simular cómo se emite la luz desde las fuentes de luz y cómo rebota en la escena hasta que toca el plano de la imagen. La luz que incide en cada píxel de la imagen definirá el color del píxel.

Para calcular toda esta iluminación necesitamos una capacidad de cómputo muy alta si queremos hacerlo en un tiempo razonable, o unos algoritmos que optimicen este cálculo, y de eso trataremos en esta tesis.

Empezaremos hablando del uso del algoritmo Virtual Point Lights, también llamado por algunos autores Instant Radiosity. Este algoritmo es capaz de generar imágenes de calidad comparable a la de, por ejemplo, Path Tracing (que es un algoritmo estándar en renderizado fotorrealista). -Explicaremos cómo funciona el algoritmo VPLs y cómo es capaz de renderizar una imagen en menos tiempo que Path Tracing y explicaremos cómo se generan las imágenes. Además, también discutiremos los artefactos que se generan usando los puntos de luz virtuales originales y cómo resolverlos.

El núcleo de este trabajo es la aceleración del algoritmo VPL original. Para ello recurriremos a técnicas de clustering, que nos permitirán reducir drásticamente el número de cálculos implicados. En particular, proporcionaremos detalles sobre la implementación del algoritmo K-means (un tipo de aprendizaje no supervisado) y su aplicación en el contexto de la síntesis de imágenes basada en VPL. Mostraremos cómo se pueden usar las medias K para agrupar los puntos visibles de cada píxel y también para agrupar los puntos de luz virtuales.

Proporcionamos resultados detallados utilizando los cuatro algoritmos diferentes: el algoritmo VPL original, que hemos implementado desde cero; VPL con K-means

clustering de los puntos visibles; VPL utilizando K-means clustering de VPLs; y, por último, los resultados al utilizar tanto el K-means clustering de los puntos visibles como de los VPL. Los resultados muestran claras mejoras en el tiempo de síntesis de imágenes.

Finalmente presentaremos los resultados obtenidos desde la perspectiva de la calidad de imagen final. Identificamos la principal limitación de los métodos desarrollados y proponemos mejoras para trabajos futuros.

Abstract

Global illumination is the task that aims to add realism to the modeling of light in 3D scenes, all this encompasses a set of algorithms that make it possible. These algorithms take into account, in addition to the light coming directly from a light source (direct illumination), the light rays that come from the same source but have been through reflections on surfaces of the scene (reflective or not) (indirect illumination). So this in computing science is a more complex term than it seems a priori, it refers not only to light sources, but to all the lighting conditions of the scene, that is, we have to take into account all the objects of the scene since each one will influence the others in one way or another with the light that it reflects, refracts or absorbs. To be more precise to calculate GI we need a description of the virtual scene which includes the position, size and orientation of the 3D geometric objects that compose the scene, the material associated with each object, the position and characteristics of the light sources which illuminate the scene, and a virtual camera that defines how a scene is seen. Once this is defined, one needs to compute/simulate how light is emitted from the light sources and bounces on the scene until it hits the image plane. The light that hits each pixel of the image will define the pixel color.

To calculate all this lighting we need a very high computing capacity if we want to do it in a reasonable time, or some algorithms that optimize this calculation, and this is what we will deal with in this thesis.

We will start by talking about the use of the Virtual Point Lights algorithm, also called Instant Radiosity by some authors. This algorithm is able to generate images of comparable quality to that of, for example, the Path Tracing (which is a standard algorithm in photo-realistic rendering). -We will explain how the VPLs algorithm works and how it is able to render an image in less time than Path Tracing and explain how the images are generated. Furthermore, we will also discuss the artifacts that are generated using the original virtual point lights and how to solve them.

The core of this work is the acceleration of the original VPLs algorithm. To this end, we will resort to clustering techniques, which will allow us to drastically reduce the number of computations involved. In particular, we will provide details on the implementation of the K-means algorithm (a type of non-supervised learning) and its application in the context of VPLs-based image synthesis. We will show how K-means can be used to cluster the visible points from each pixel, and also to cluster the virtual light points.

We provide detailed results using the four different algorithms: the original VPLs algorithm, which we have implemented from scratch; VPLs with K-means clustering of the visible points; VPLs using K-means clustering of the VPLsa; and, finally, the

results when using both K-means clustering for the visible points and of the VPLs. The results show clear improvements in image synthesis time.

Finally we will present the results obtained from the perspective of the final image quality . We identify the main limitation of the methods developed, and propose improvements for future work.

Section 1 - Introduction

The methods to get a 3D visualization of a virtual scene that is practically indistinguishable from reality are called Photorealistic Rendering. Many of them have been developed for photorealistic rendering and even more variations of them exist. But before starting to get into the matter, we must understand the basic theory that is behind these methods.

What we are aiming at is to be as close as possible to how light interacts with matter in real life. For that we need to use physics and its laws to describe light transport in a scene. We know that no energy is ever lost, it is merely transformed into another form. Since light is composed of electromagnetic waves, this is basically a flow of energy problem.

All of this leads us to the rendering equation [9]. The rendering equation is an aspect of computer graphics that deals with how light radiates and bounces off surfaces, so that we can synthesize realistic images of three-dimensional (3D) virtual scenes. The problem is that in most cases it is impossible to solve the rendering equation $L_r(x, \omega_o)$ analytically. Therefore, we need to use methods to approximate its value.

Global illumination (GI) is the task that aims to add realism to the modeling of light in 3D scenes. All this encompasses a set of algorithms that make GI possible. The base algorithm that we are going to work on in this Thesis is the Virtual Point Lights (VPL) algorithm, also called Instant radiosity, which was introduced by Alexander Keller in 1997 [1]. Briefly speaking,

The advantage of this method is that it can be quite fast when compared to other (more computationally demanding) techniques such as, for example, path tracing [REF]. We only calculate the VPLs once at the beginning of the algorithm, and then we render the scene. It has some disadvantages like if not enough VPLs are generated then some relevant light transport paths can be missed. This problem is something that we are going to try to solve in this project using a clustering algorithm which will let us use more VPLs without having a negative impact in rendering times.

1.1 - Motivation

3D modeling or graphic design has acquired an important role in recent years in the creation of video games, visual effects, virtual tours for architectural projects, or as a learning and experimentation tool. Every time a greater quality of the images is sought, to the point of being practically impossible to differentiate them from a photograph or a video of the real world. But all this has a problem, and it is the computational cost at the time of rendering such amount of information.

We are talking about millionaire companies that use cutting-edge technology every day to be able to render all their projects in less time. To put what we mean in perspective, the clear example would be the movie "The Jungle Book" (2016) by Disney, which needed to use the Google cloud network, with thousands of computers working in parallel, as reported Robert Legato, responsible for the special effects, since each frame of that film, using a normal computer, had a time of 80 hours, which would be equivalent to about 1500 years to complete the film. That's why getting even a small improvement in rendering time would be a very important improvement in this sector.

Finally, I would like to add the personal and academic interest in this field, which the graphs and data visualization course generated in me last year.

1.2 - Objectives

Here we will see the main objectives of this thesis, which are composed of a main one that we can subdivide into several sub-objectives.

The main goal is to reduce rendering time while maintaining as much as possible the quality of the produced images. For this we have three sub-objectives, which would be: (i) to understand and implement the Virtual Point Lights algorithm efficiently; (ii) use the K-means clustering algorithm for clustering VPLs, hence reducing the number of VPLs that need to be taken into consideration to illuminate the virtual scene; (iii) and finally, use the K-means clustering algorithm to cluster the visible points, hence reducing the number of visible points for which the illumination is explicitly computed.

For this we will use a ray tracer already implemented which will be explained later. This work implies understanding how the raytracer works, how it is organized from the base, and to be able to see later the improvements that have been made to the raytracer.

The Ray tracer has been provided to us by the Doctor and tutor of TFG Ricardo Marques. This ray tracer will serve as the basis to be able to create our project successfully.

1.3 - Planification

In any big project, when we look at the deadline it can seem like we have a lot of time ahead of us and that it will be easy for us to finish it on time. And that's a mistake. We have learned over the years that we have seen ourselves working on a project until the last moment before the deadline, that good planning and guidance of this is very important to succeed in any great project that we get into. So that's why we divided all we have to do into three different phases, and each phase was divided into 5 different tasks.

Global Illumination Through Virtual Point Lights																								
Project Start		24/01/2022		24/01/2022						31/01/2022				07/02/2022										
				24	25	26	27	28	29	30	31	01	02	03	04	05	06	07	08	09	10	11	12	13
TASK	START	DAYS	END	dl.	dt.	dc.	dj.	dv.	ds.	dg.	dl.	dt.	dc.	dj.	dv.	ds.	dg.	dl.	dt.	dc.	dj.	dv.	ds.	dg.
Phase 1																								
Task 1	24/01/2022	14	06/02/2022																					
Task 2	07/02/2022	7	13/02/2022																					
Task 3	14/02/2022	14	27/02/2022																					
Task 4	28/02/2022	3	02/03/2022																					
Task 5	03/03/2022	4	06/03/2022																					

14/02/2022		21/02/2022						28/02/2022												
14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	01	02	03	04	05	06
dl.	dt.	dc.	dj.	dv.	ds.	dg.	dl.	dt.	dc.	dj.	dv.	ds.	dg.	dl.	dt.	dc.	dj.	dv.	ds.	dg.

Project Start		24/01/2022		07/03/2022						14/03/2022						21/03/2022				28/03/2022											
				07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	01	02	03
TASK	START	DAYS	END	dl.	dt.	dc.	dj.	dv.	ds.	dg.	dl.	dt.	dc.	dj.	dv.	ds.	dg.	dl.	dt.	dc.	dj.	dv.	ds.	dg.	dl.	dt.	dc.	dj.	dv.	ds.	dg.
Phase 2																															
Task 1	07/03/2022	7	13/03/2022																												
Task 2	14/03/2022	14	27/03/2022																												
Task 3	28/03/2022	7	03/04/2022																												
Task 4	04/04/2022	14	17/04/2022																												
Task 5	18/04/2022	7	24/04/2022																												

04/04/2022		11/04/2022						18/04/2022												
04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
dl.	dt.	dc.	dj.	dv.	ds.	dg.	dl.	dt.	dc.	dj.	dv.	ds.	dg.	dl.	dt.	dc.	dj.	dv.	ds.	dg.

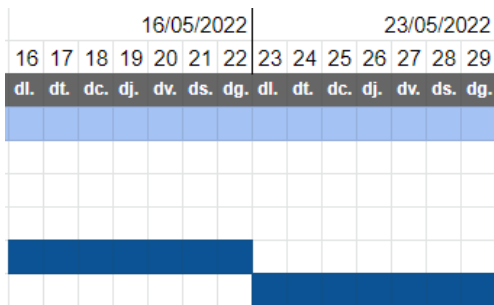
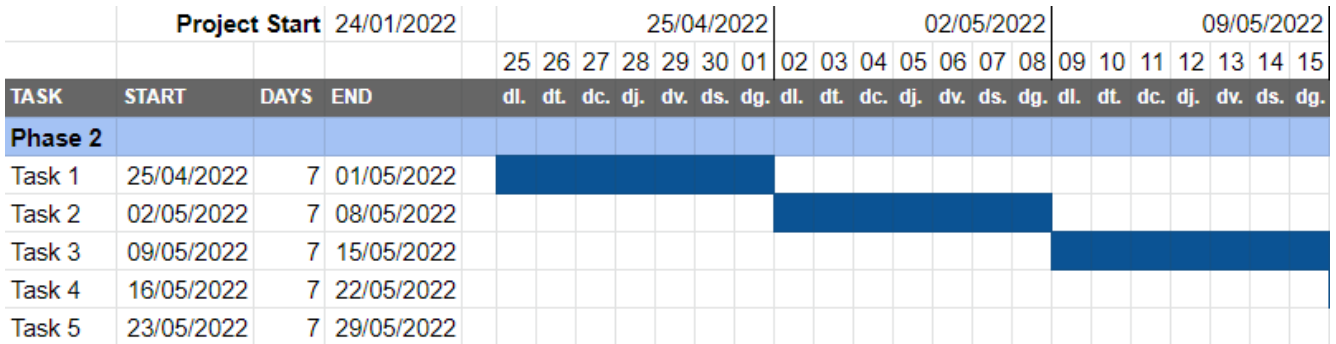


Figure 1.3.1: TFG Gantt chart

As we mentioned before, we had 3 phases of the planification (see Table 1.3.2), the first one called the implementation of the VPLs where we had a long term of learning, first about the ray tracer and then about the VPL base algorithm, that was before to start building a base shader to have something to start implementing the future algorithms.

Once the first one was finished we started learning about the K-means clustering and as soon as we had an idea how it works, the implementation task started. Even in the Figure 1.3.1 all the work seemed very lenear on its evolution, every time we implemented something we had to do some adjustment to previous methods. That's why it took a long time to finish phase one and two and start doing the experiments.

Finally the Experiments, analysis and results phase, as its title says, is the phase where all the code is working fine and we can do the test and know if we achieved our initial objectives.

Phase 1	Implementation VPL
Task 1	Understanding the Ray tracer and implementing a simple shader.
Task 2	Understanding the VPL algorithm
Task 3	Implement VPL algorithm
Task 4	Solving the artifacts with clamping, and other math mistakes in the code
Task 5	Display VPL method and shader
Phase 2	Implementing K-means clustering
Task 1	Understanding the K-means algorithm
Task 2	Implement Visible Points clustering shader

Task 3	Implement shader to display clusters and centroids in different colors
Task 4	Implement Virtual Point Lights clustering shader
Task 5	Method and shader to display VPL clusters
Phase 3	Experiments, Analysis and Results
Task 1	Experiments with VPL shader
Task 2	Experiments with VPL clustering shader
Task 3	Experiments with Visible Point clustering shader
Task 4	Experiments with VPL clustering + Visible Point clustering shader
Task 5	Final Conclusions

Table 1.3.2: Gantt chart explanation: Here we can see a summarization of what each task means.

1.4 - Thesis structure

This Final Degree Thesis deals with one of the methods for rendering photorealistic images: Virtual Point Lights (VPL). Furthermore, it also deals with the algorithm of K-means to accelerate the rendering. Virtual Point Lights is a relatively fast method that can produce realistic-looking results, and K-means is an algorithm used in other fields, more used for Data management. These details will be explained in Section 3.

All of this Final Degree Thesis was implemented using a base ray tracer provided by Dr. Ricardo Marques, tutor of this thesis, and the language we used is C++ and the integrated development environment is Visual Studio. This is detailed in Section 4.

In Section 5, we have the results of all the experiments that we did to complete this thesis, as well as our conclusions. Finally, in section 6, we explain the future work that can be done to improve more of this work that has been done.

Section 2 - State of the Art

2.1 - Global illumination

The behavior of light in the real world it's far from simple, although we can think it is not that complicated, when we start evaluating a scene we realize why it is not simple. All of us know that light can be reflected, refracted, scattered, and absorbed. All of this can happen multiple times from one light source, imagine multiple light sources with different wavelengths to different extents. This means that surfaces that are not directly illuminated, can still be visible (indirect illumination) or a white object can appear in any other color because of the influence of a nearby object, and light is being reflected from it to the wall (color bleeding). All these effects are collectively referred to as global illumination (GI). The term global illumination refers to the fact that we need to take into account lighting conditions in the whole scene, because nearby surfaces (or even surfaces that are quite far away) can influence each other and it is therefore insufficient to just calculate lighting in one place (see Figure 2.1).

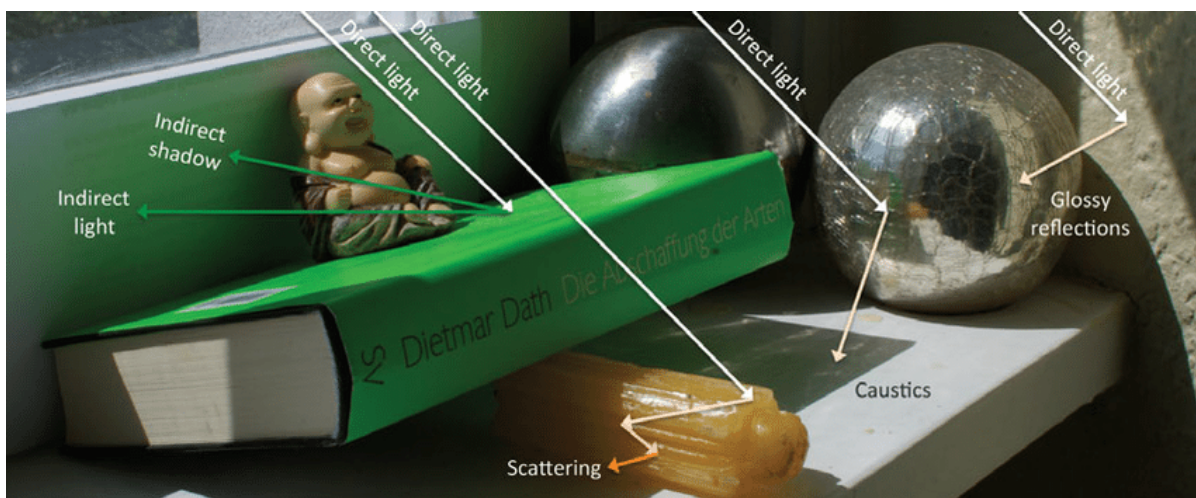


Figure 2.1: Image demonstrating several global illumination effects; multiple diffuse and specular bounces, caustics and scattering.

2.2 - Rendering equation approximation

To model light transport in a scene, we need to express it in mathematical terms. That means that we need to make use of physics and its laws to derive equations, which describe light transport in the scene. One of the fundamental laws of physics is the law of conservation of energy, which states that no energy is ever lost, it is merely transformed into another form. Since light is electromagnetic waves, it is basically a flow of energy. Its amount is expressed as $L(x \leftarrow \omega)$ which means energy arriving at point x from direction ω , or $L(x \rightarrow \omega)$ which means energy emitted from point x in direction ω .

The outgoing radiance $L_o(x, \omega_o)$ from a point x with normal n_x towards a direction ω_o is the sum of the emitted and reflected radiance ($L_e(x, \omega_o)$ and $L_r(x, \omega_o)$ respectively), modeled as:

$$(Eq. 1) \quad L_o(x, \omega_o) = L_e(x, \omega_o) + L_r(x, \omega_o)$$

$$(Eq. 2) \quad L_r(x, \omega_o) = \int_S L_i(x, \omega_i) f_r(x, \omega_i, \omega_o) G(x, s_j) V(x, s_j)$$

As we said before in most cases it is impossible to solve $L_r(x, \omega_o)$ analytically. So an intuitive approximation of Equation (2) would be to model the light reflected at each point $s \in S$ as a virtual point light, with emission $I(s_j, x)$ (Eq. 3) equal to the energy reflected at s towards x , the $f_r(x, \omega_i, \omega_o)$ is the BRDF (Bidirectional Reflectance Distribution Function) the fundamental quantity to describe the spectral and directional properties of reflectivity(the material), $G(x, s_j)$ would be the geometry term and finally $V(x, s_j)$ which is the visibility or the amount of light sources that hit the visible point. Therefore, if we precompute a sampled set of N virtual lights, then $L_r(x, \omega_o)$ can be approximated as:

$$(Eq. 3) \quad L_r(x, \omega_o) \approx \sum_{j=1}^N I(s_j, x) f_r(x, \omega_i, \omega_o) G(x, s_j) V(x, s_j)$$

In this Final Degree Thesis to synthesize an image we calculate the visibility $V(x, s_j)$ just for the cluster center or centroid, and the rest of the equation is calculated for each visible point. We do that because to calculate the visibility we need to throw a lot of shadow rays so it has a high computational cost, and calculating this just for the cluster center or centroid means reducing substantially the cost.

The emission $I(s_j, x)$, material $f_r(x, \omega_i, \omega_o)$ and the geometry term $G(x, s_j)$ are calculated for each visible point because have really low computational time and cost, and it give more smoothness in the images if the $V(x, s_j)$ of the near clusters are similar.

2.3 - Virtual Point Lights Algorithm

Virtual Point Lights (VPL) algorithm, also called Instant radiosity, was introduced by Alexander Keller in 1997 [1]. It is a two phase algorithm (see Figure 2.3.1). The first phase is executed at the beginning (in the preprocess method), where many particles (VPLs) are traced through the scene, to do that we trace a number of light rays bouncing throughout the scene then we record where each ray hits the scene, and create a virtual light at that point. This algorithm has two parameters: Number of VPLs generated on light source (N) and number of bounces each light undergoes until terminated (B), to calculate the final amount of VPLs we have to multiply N by B. Once the light rays are traced randomly from the light source, we store them in an array with or without the light source, depending if we want to render the scene only using VPLs or VPLs plus the light sources.

The second phase is the rendering of the scene which is very simple compared to other algorithms, in Virtual Point Light algorithm we only need to trace primary rays from the camera to the scene and then trace shadow rays to every VPL and determine how much every point is illuminated by taking an sum of the contributions of the VPLs that illuminate it.

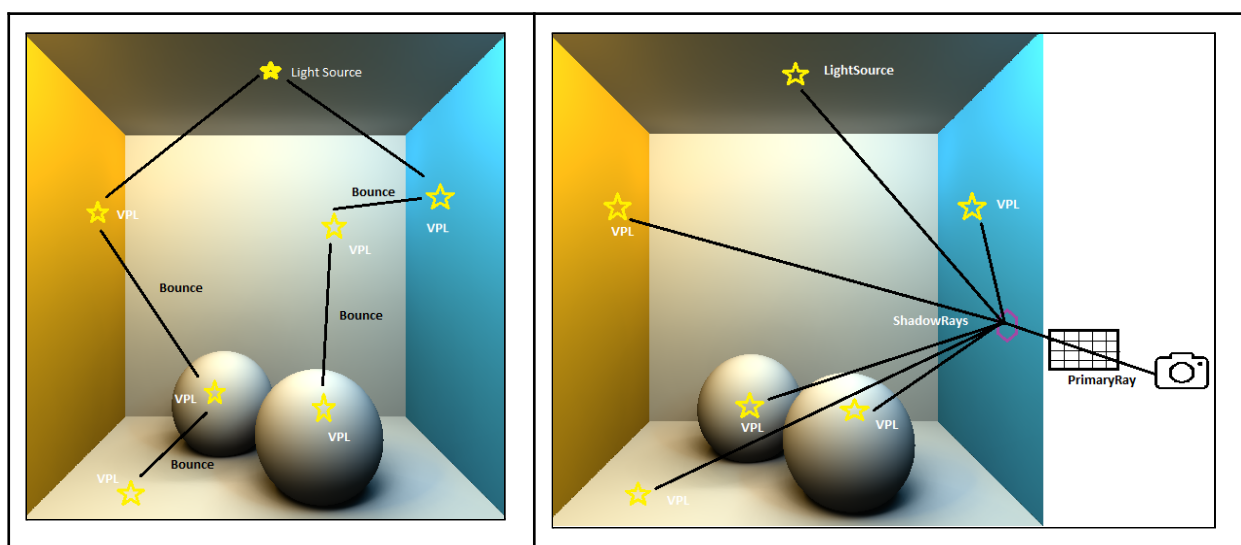


Figure 2.3.1: Creation of the VPLs (left) and rendering of the final image using the resulting VPLs (right).

The advantage of this method is that it can be quite fast. We only calculate the VPLs once at the beginning, as a pre-processing step, and then render the scene using the generated VPLs.

The disadvantages of this method are, that it is biased and that usually not so many VPLs are generated and therefore some light transport paths can be missed, so the final image can not be as perfect as we expected (see Figure 2.3.2). So before rendering a lot of scenes we need to know how many VPLs we will need if we want good results without wasting a lot of time.

Another disadvantage is when we use a glossy surface some artifacts can appear, something that can be solved with clamping.



Figure 2.3.2 Illustration of singularities. From left to right: Unbounded VPLs, Bounded VPLs, unbiased solution (non-VPL) (Source: [6])

Section 3 - Efficient Global Illumination Through Virtual Point Lights

To have an overview of what we are going to explain more precisely in the methodology, we will try to summarize in a few words the algorithms that have been used in this thesis. First of all we are going to explain the base algorithm of virtual point lights, its phases, and we will explain how each VPL of virtual light is formed and how the rays are traced from the camera for later rendering.

We will also explain the importance of clamping in the VPL algorithm since without it you would see a lot of artifacts in the corners of the scene, these artifacts appear as bright points of light.

Finally we will explain the operation of the clustering used in the previous part of the render (preprocess) to improve the rendering times using the K-means algorithm.

We are going to explain this algorithm in a general way, applied to the clustering of visible points, to the clustering of virtual point lights and finally how the images are synthesized in each of these algorithms.

3.1 - Virtual Point Lights Algorithm

Virtual Point Lights (VPL) algorithm, also called Instant radiosity is a two phase algorithm. The first phase is executed at the beginning (in the preprocess method), where many particles (VPLs) are traced through the scene, to do that we trace a number of light rays bouncing throughout the scene then we record where each ray hits the scene, and create a virtual light at that point.

To have a better understanding of the algorithm we can divide the first phase in 5 steps.

Five steps of Virtual Point Lights Algorithm.

1. **Choosing the number of VPL and bounces we will use.** The total number of VPL will be $VPL \times Bounces$ (see Figure 3.1.0).
2. **LightRay:** From the light source we trace a ray to a random direction of the scene as many times as num of VPL we chose.
3. **Position.** The intersection that we will get from the ray we traced before, will be the place where the VPL will be placed.
4. **Light Intensity.** The VPL created will be assigned the light intensity that reaches at that point from the main light source.

5. **Bounces.** The steps 2, 3 and 4 will be repeated but the starting point of the lightRay will be the VPL that we just created instead of the first LightSource. This will be a recursive function that will be repeated for each VPL as many times as bounces we chose (see Figure 3.1.1).

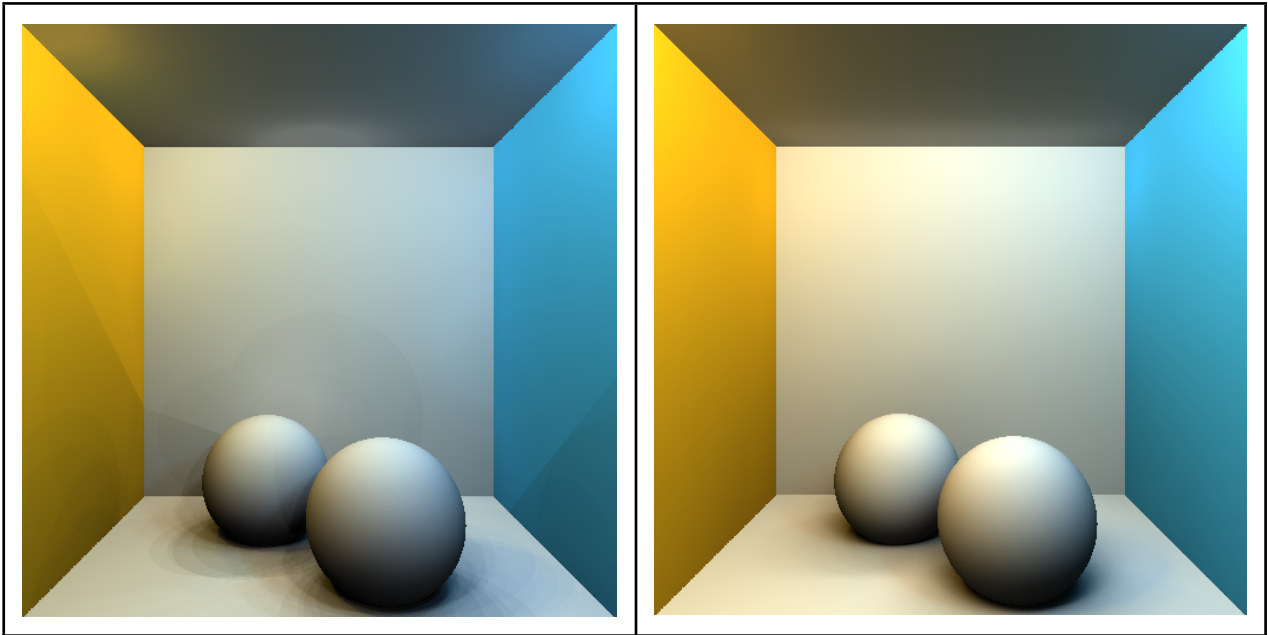


Figure 3.1.0: Two images generated with different numbers of VPLs; The left-most figure was generated using 100 VPLs and 1 bounces, and the second one generated using 2000 VPLs and 2 Bounces.

In the second phase to render the scene using the VPLs that we generated, we need first to trace primary rays from the camera to the surface of the scene and then we trace shadow rays to every VPL generated and determine how much every point is illuminated (see Figure 3.1.2).

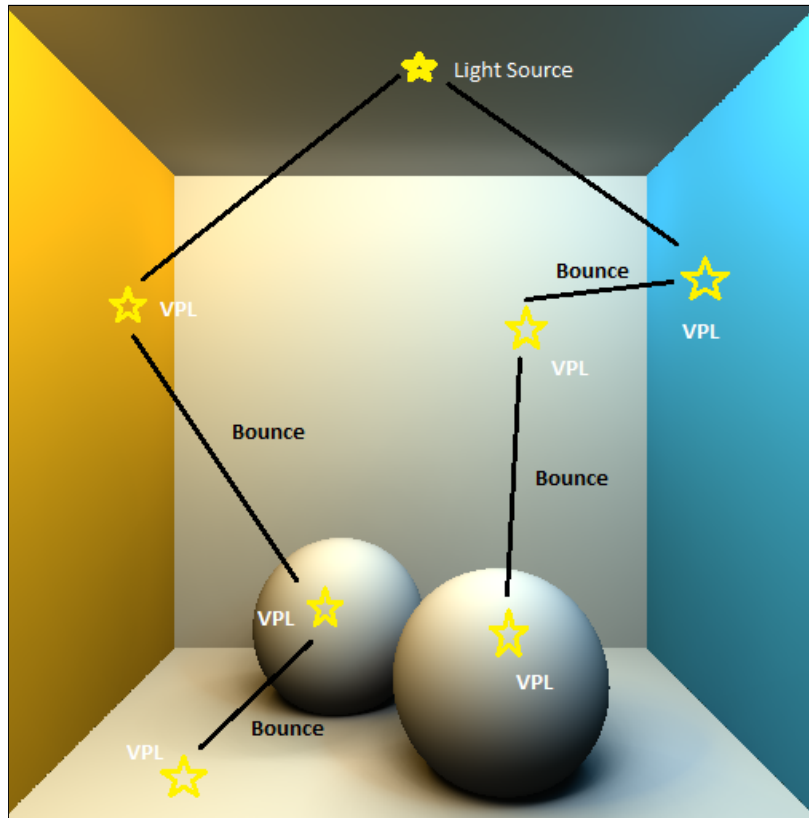


Figure 3.1.1: Representation of how two VPL of 2 bounces each are created. The starting point will be the light source on the top.

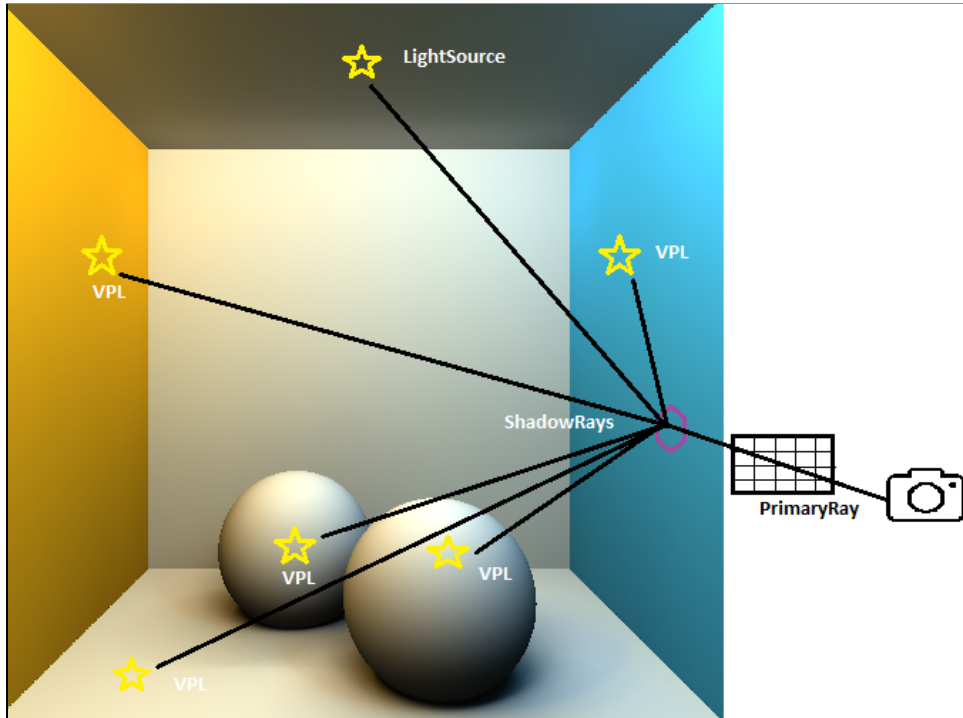


Figure 3.1.2: Rendering phase. In the rendering phase we shoot primary rays to the scene and from the intersection point we trace shadow rays to all the light sources (Point lights and Virtual Point Lights).

3.1.1 - The importance of Clamping in VPLs-based Algorithms

The incident radiance (r) $L_i(x, \omega_i)$ (Eq. 2) at an object surface is calculated by dividing the intensity (I) of the light source for the distance (d) between the object and the light source:

$$r = \frac{I}{d}$$

So when we use VPLs, and those end up in a corner the the distance between the light source (VPL) and the material is nearly 0 which end up in a nearly infinite radiance, and that's why some artifacts or singularities appear in some corners.

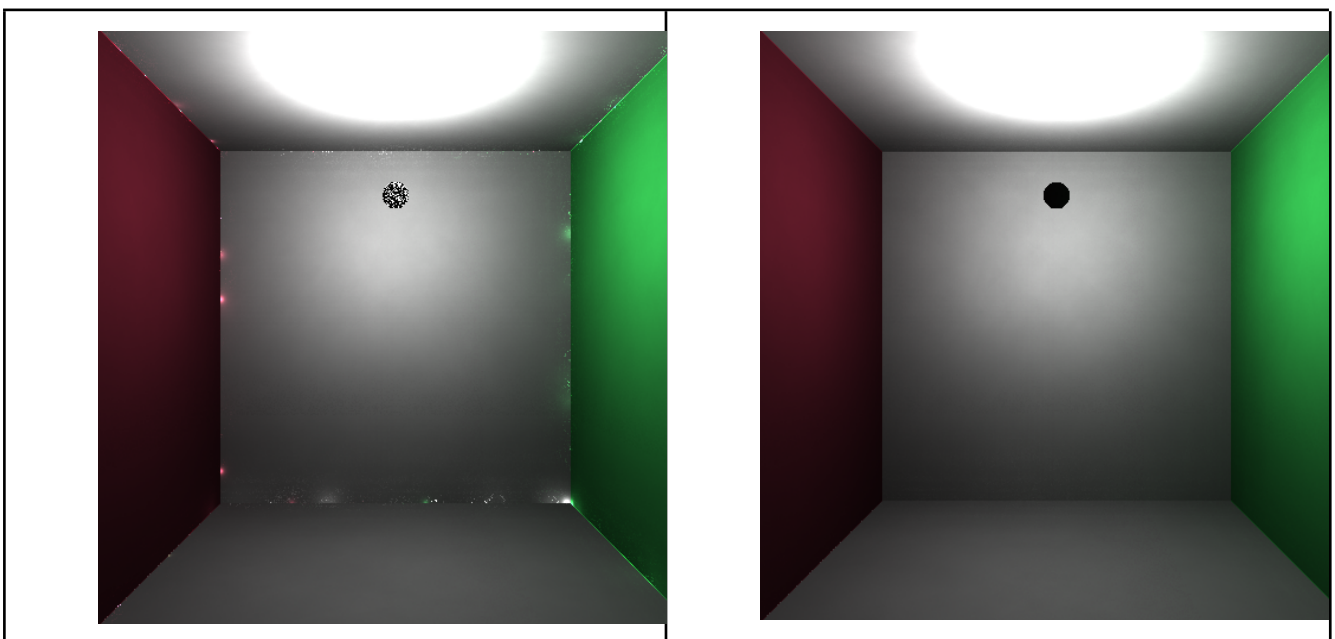


Figure 3.1.3: Illustration of singularities or artifacts solving. A clear improvement in the scene appears when we apply clamping in the VPL radiance.

3.3 - VPL-based image synthesis with clustering of visible points

3.3.1 - K-means clustering explained in general context

K-means clustering algorithm

This algorithm is applied twice in this project, once when we cluster the visible points or intersections (See figure 3.5.2), and when we cluster the VPLs (See Figure 3.5.3).

Clustering is the process of dividing the entire data into groups (also known as clusters) based on the patterns in the data. For that purpose we need to establish how many clusters (k) we need and the characteristics that differentiate one cluster from another. K-means is a distance-based algorithm, where we calculate the distances to assign a point to a cluster.

In K-Means, each cluster is associated with a centroid. The main objective of the K-Means algorithm is to minimize the sum of distances between the points and their respective cluster centroid. All of that and how it works will be explained in more detail below.

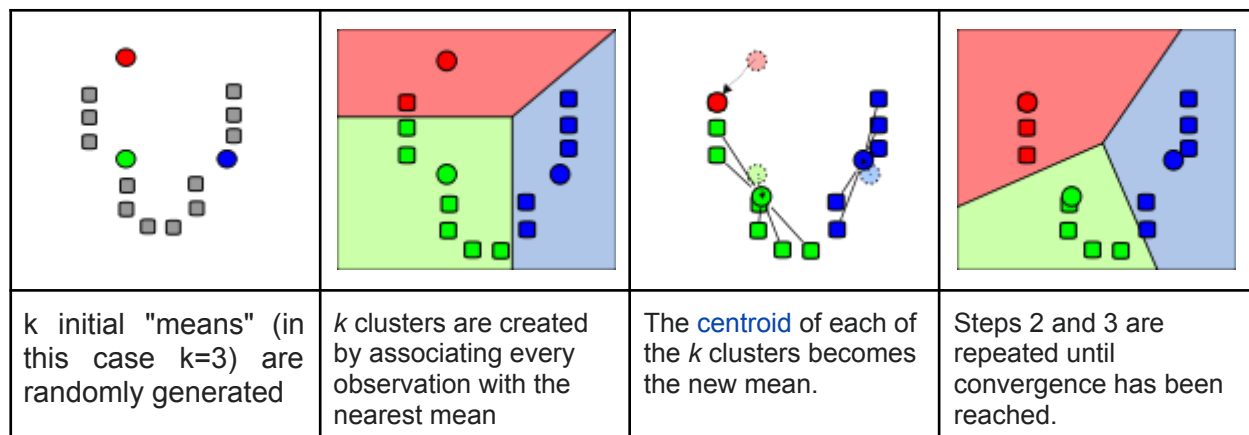


Figure 3.3.1 Demonstration of standard algorithm [11]

The K-means algorithm can be divided in **5 steps**:

Step 1: Choose the number of clusters k

The first step in k-means is to pick the number of clusters, k .

The clusters have two properties:

1. All the data points in a cluster should be similar to each other.
2. The data points from different clusters should be as different as possible.

Step 2: Select k random points from the data as centroids

The **centroid** of a cluster is defined as the equidistant point from the objects belonging to that cluster. (See Figure 3.3)

Step 3: Assign all the points to the closest cluster centroid

Once we have initialized the centroids, we assign each point to the closest cluster centroid. As we said in step 1 we need to focus on the cluster's properties, the data points in a cluster should be similar to each other and the data points from different clusters should be as different as possible.

Step 4: Recompute the centroids of newly formed clusters

Now, once we have assigned all of the points to either cluster, the next step is to compute the centroids of newly formed clusters.

Step 5: Repeat steps 3 and 4

The step of computing the centroid and assigning all the points to the cluster based on their distance from the centroid is a single iteration, and we will have to repeat steps 3 and 4 until one of the stopping criteria that we chose happens.

Possible stopping criteria:

1. Centroids of newly formed clusters do not change
2. Points remain in the same cluster
3. Maximum number of iterations are reached

3.3.2 - K-means clustering of visible points

The goal of using this algorithm applied to visible points is to reduce the time when rendering, to do that we will treat every cluster as a whole visible point, and to understand how this reduces the time of rendering, we will show in numbers. For example working with an image of 512x512, this does in total 262.144 visible points, if we cluster them into clusters of 100 visible points, we will end with 2600 clusters approx, this means the render will have to deal with 2600 “pixels”, obviously is not that simple, and will be explained in more detail just below, but is to show how it reduces the rendering time.

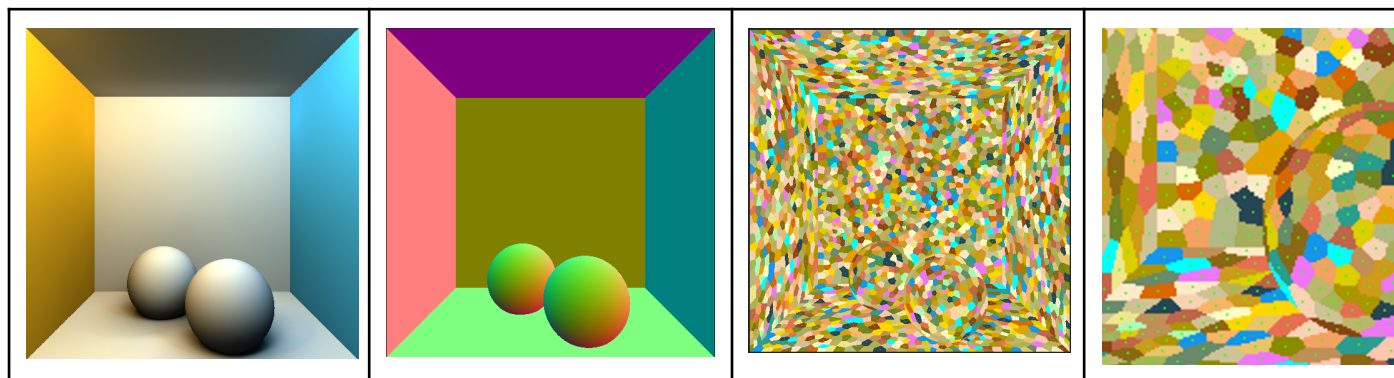


Figure 3.3.2: Clustering of the visible points. From left to right: (left) the original scene on which clustering will be performed; (ii) the result of a normal shader to see which shows the 3D positions and the surface normals ; (iii) the result of clustering the visible points, where each cluster is depicted using a random color; (iv) close-up view of the left bottom corner to appreciate the centroids (green dots) and how the normal distance avoids clusters to spread across differently aligned surfaces.

The steps of the k-means are the same as in Section 3.3.1 but now will be explained for the visible points clustering.

Step 1: Choose the number of clusters k

Choosing the wrong amount of clusters will end in high rendering time or in a poor quality final image, and we want a balanced time and final quality.

Step 2: Select k random points from the data as centroids

The **centroid** of a cluster is defined as the equidistant point from the objects belonging to that cluster. In this algorithm the centroid will be suffering changes in each iteration of the algorithm until it stabilizes.(see Figure 3.5.1)

Step 3: Assign all the points to the closest cluster centroid

For that we will use a three dimensional Euclidean distance and the distance between their Normal all combined.

In a three dimensions plane, for points given by their Cartesian coordinates (p, q) which are three-dimensional vectors, the distance (d_e) is:

$$d_e(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + (p_3 - q_3)^2}$$

To calculate the Normal distance, we use the dot product or scalar product between its Normal vectors (n_p, n_q):

$$n_p \cdot n_q = |n_p| \cdot |n_q| \cos\theta$$

With that we will get the result of the $\cos\theta$ and the outcome of this operation will be between **1** to **-1**. **1** will mean the Normal has the same angle, 0 will mean the vectors are orthogonal, difference of 90 degrees, and -1 will be totally opposite directions.

Our final objective is to get one number as a result of the combination of Euclidean distance and the Normal distance, so we combined them like this:

$$d(p, q) = \left\{ \begin{array}{l} d_e(p, q) \text{ if } \cos\theta < t \\ \infty \text{ otherwise} \end{array} \right\}$$

Step 4: Recompute the centroids of newly formed clusters

In each iteration we will check all distances, explained in step 3, between the new centroids and the intersections.

Step 5: Repeat steps 3 and 4

In the project the stopping criteria is a maximum number of iterations, due to the big changes in the algorithm happen in the first iterations, that's why we will use 3 or 4 iterations maximum, because the changes that can happen after that first 3 iterations will be minimum and in proportion the computational time increment will not be worth.

3.3.3 - Synthesizing an image with visible points clustering

This is the most important part, since it is where we must make the most of the clustering, trying to make it as little noticeable as possible. For this we have used an approximation of the rendering formula, since in most cases it is impossible to solve $L_r(x, \omega_o)$ analytically, and stochastic techniques are usually used to compute it.

The outgoing radiance $L_o(x, \omega_o)$ from a point x with normal n_x towards a direction ω_o is the sum of the emitted and reflected radiance ($L_e(x, \omega_o)$ and $L_r(x, \omega_o)$ respectively).

As we said before in most cases it is impossible to solve $L_r(x, \omega_o)$ analytically. So an intuitive approximation of Equation (Eq. 2 Section 2.2) would be to model the light reflected at each point $s \in S$ as a virtual point light, with emission $I(s_j, x)$ equal to the energy reflected at s towards x , the $f_r(x, \omega_i, \omega_o)$ is the BRDF (Bidirectional Reflectance Distribution Function) the fundamental quantity to describe the spectral and directional properties of reflectivity(the material), $G(x, s_j)$ would be the geometry term and finally $V(x, s_j)$ which is the visibility or the amount of light sources that hit the visible point. Therefore, if we precompute a sampled set of N virtual lights, then $L_r(x, \omega_o)$ can be approximated as:

$$L_r(x, \omega_o) \approx \sum_{j=1}^N I(s_j, x) f_r(x, \omega_i, \omega_o) G(x, s_j) V(x, s_j)$$

To synthesize an image we calculate the visibility $V(x, s_j)$ just for the cluster center or centroid, and the rest of the equation is calculated for each visible point. We do that because to calculate the visibility we need to throw a lot of shadow rays so it has a high computational cost, and calculating this just for the cluster center or centroid means reducing substantially the cost.

The emission $I(s_j, x)$, material $f_r(x, \omega_i, \omega_o)$ and the geometry term $G(x, s_j)$ are calculated for each visible point because have really low computational time and cost, and it give more smoothness in the images if the $V(x, s_j)$ of the near clusters are similar.

3.4 - VPL-based image synthesis with clustering of VPLs

3.4.1 - K-means clustering of VPLs

The goal of using this algorithm applied to Virtual point lights is to reduce the amount of Shadow rays that we have to trace for each visible point, meaning in the end reduce some rendering time.

To simplify what we did with this algorithm is to do an average of all the VPLs intensities and positions.

The steps of the k-means are the same as in the 3.3.1 but now will be explained for the Virtual Point Lights clustering.

Step 1: Choose the number of clusters k

As it happens in the previous clustering algorithm, choosing the wrong amount of clusters will end in higher rendering time or in a poor quality final image, and we want a balanced time and final quality.

Step 2: Select k random points from the data as centroids

The **centroid** of a cluster is defined as the equidistant point from the objects belonging to that cluster. In this algorithm the centroid will be suffering less changes in each iteration of the algorithm until it stabilizes.(See Figure 3.3)

Step 3: Assign all the points to the closest cluster centroid

As we explained previously we will use the combination of Euclidean distance and the Normal distance and we combined them like this:

$$d(p, q) = \left\{ (d_e(p, q) \text{ if } \cos\theta < t) \text{ otherwise } \infty \right\}$$

Step 4: Recompute the centroids of newly formed clusters

In each iteration we will check all distances, explained in step 3, between the new centroids and the intersections.

Step 5: Repeat steps 3 and 4

In the project the stopping criteria is a maximum number of iterations, due to the big changes in the algorithm happen in the first iterations, that's why we will use 2 or 3

iterations maximum, because the changes that can happen after that first 3 iterations will be minimum and in proportion the computational time increment will not be worth.

3.4.2 - Synthesizing an image with Virtual Point Lights clustering

The synthesization of the image using VPL clustering is the same as we don't use this algorithm, but with the algorithm it is faster. From every visual point we trace shadow rays to the cluster center or centroid and we work with that if it was a VPL or Light source, with its own intensity and position. (See Figure 3.5.2)

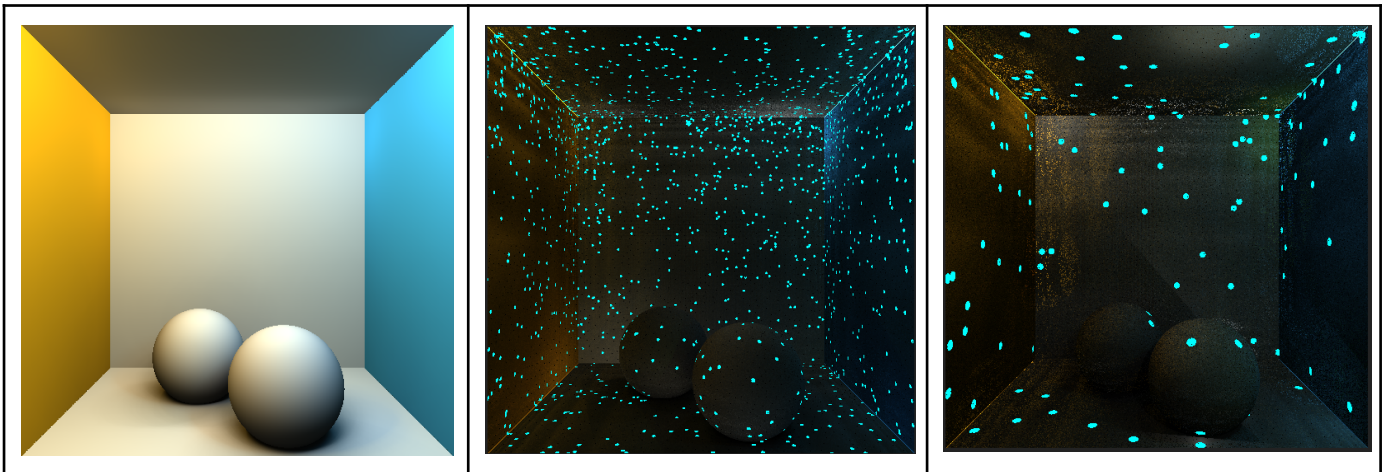


Figure 3.4.2: Clustering VPLs. From left to right, the left-most image shows the scene for which the VPLs are generated; the center image shows the position of all VPLs generated (as small blue dots); and the right-most image shows the centroids (as large blue dots) resulting from clustering the original VPLs .

3.5 - VPL-based image synthesis with visible points and VPLs clustering

When we combine both algorithms, what we have to do to synthesize the final image, is really close of what we explained in the Section 3.3.3, but instead of throwing shadow rays from the centroid of visible points to the VPLs, we trace shadow rays to the VPL clustering centroid or cluster center. Remember that we do that to calculate the amount of Light sources that hits the centroid of visible point cluster center also called visibility $V(x, s_j)$. And then as we explained before we calculate the rest of the rendering equation approximation for each visible point in the cluster. Detailed visual results and timings for this rendering approach are shown in Section 5.

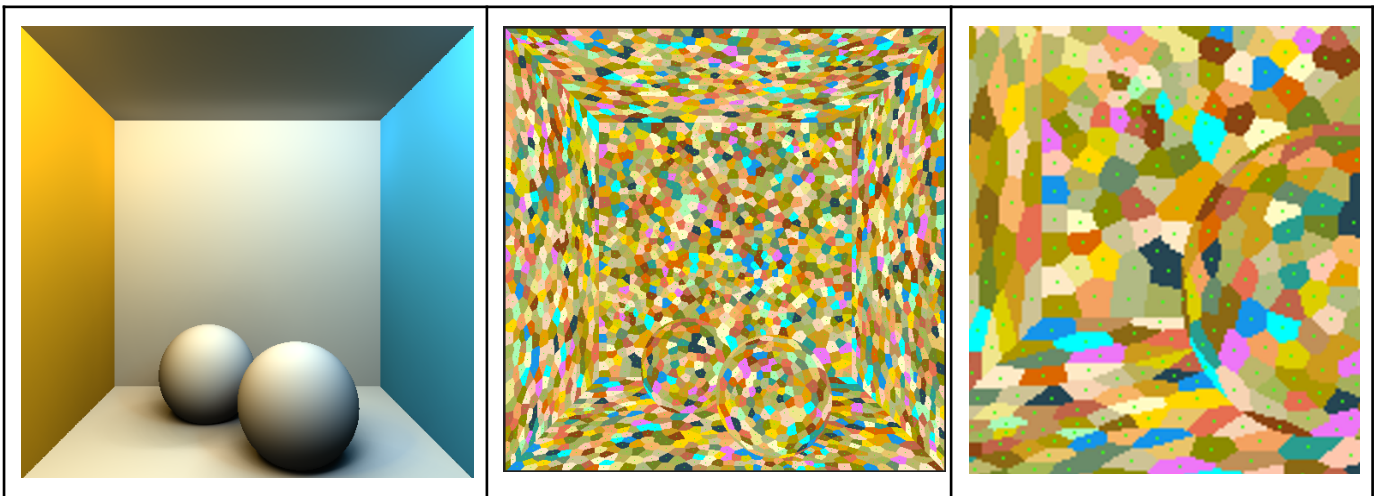


Figure 3.5.1: Clustering of the visible points. From left to right: (left) the original scene on which clustering will be performed; (ii) the result of clustering the visible points, where each cluster is depicted using a random color; (iii) close-up view of the left bottom corner to appreciate the centroids (green dots) and how the normal distance avoids clusters to spread across differently aligned surfaces.

Section 4 - Implementation

4.1 - Visual Studio

Microsoft Visual Studio is an integrated development environment (IDE) for Windows and macOS. It is compatible with multiple programming languages, such as C++, C#, Visual Basic etc... It will be the escollit program since I had previously contacted other subjects, and the Ray tracer Base was created with Visual Studio due to its compatibility with C++.

4.2 - Base Ray tracer

The ray tracer base has been the program that, as the name says, has served as the basis for me to start studying global illumination from VPL and the use of other algorithms. It is capable of rendering scenes using different types of shaders, like path shader, direct shader, depth shader and others, all of which use the programming language of C++, C++ is the most used language for programming graphics related due to its versatility.

To explain the base ray tracer deeply I will divide it into 8 parts or what the visual studio says in 8 filters (see figure 4.1): Cameras, Core, light sources, materials, samplers, shapes, shaders, and The Main.

The cameras are the point of view of the scene, from where the first rays are traced(see Figure 3.2).

The core is where all the necessary objects to work with a ray tracer are needed, like 3D vector, bitmap, intersection, rays, textures etc...

Light sources as its name says is the class that helps us to create light sources(Area lights and point lights) , and in the future will also contain the VPL class.

Materials contain the 3 different materials that we can create(mirror, phong and transmissive), even in this project we will only work with diffuse reflections.

The sampler lets us trace a random ray to any direction of an hemisphere from any point in the scene

Shapes contain two basic shapes to create the scenes, which are spheres and triangles.

Shaders have all the different classes that render images in a different way, here the vast majority of my work will be done.

And finally the main class, is where the preprocess, the render, the post process are executed.

4.3 - Changes done in the base ray tracer

To explain what changes in the base tracer were done we will use the filters used before to explain the base ray tracer to have some order in the explanation.

The filters that had some changes are: Shaders, Core, Samplers, Light Sources and the Main class.

The shaders are where the vast majority of changes were done. Here we added six new shaders, each one render the image in different ways or using different algorithms:

1. Simple Shader
2. VPL shader
3. simple clustering shader
4. VAL shader
5. VPL clustering shader
6. Simple Shader 2

Simple shader, despite the name, is the main shader, where the usage of VPLs, the clustering of them and the clustering of visible points is done, and rendered, so it is everything except simple. In order to do this in this class, we needed to overwrite the render and the preprocess methods. Also we had to implement the distance calculators, one with vectors as parameters and other with VPLs. For the clustering of visible points we needed to modify the **intersection** class in the **Core** filter, we added two labels to set them as a centroid and to assign them to the cluster.

VPL shader only renders images using VPLs, and light sources.

Simple clustering shader is used to render images using VPLs and clustering of visible points.

VAL shader is used to render images using VPLs and the cluster of them.

VPL clustering shader and **Simple Shader 2** both are used to create images where the algorithms can be explained because they can show the scene with the clusters painted with different colors with their centroids, and images where VPLs and VPLs clusters can be shown.(See Figures 3.3 and 3.4)

Continuing with the **samplers**, the necessity to add a **spherical sampler** to trace random rays from point lights was essential.

In **Light Sources** filter we had to add the VPL class which is similar to the lightsource class, but with more functions, to let us set a new position for the VPL (used during the clustering of VPL where we change the position of the centroid) and to set a new intensity due to it changes depending of how many VPL are in the VPL cluster.

Finally in the main class we added a timer to calculate to the millisecond how long each process takes.

Section 5 - Results

5.1 - Experimental Set-Up

As final objectives of this project we sought to maintain a good image quality and lower the execution time in the rendering process, for this, as we have explained previously, the global illumination algorithms have been used through VPL, and the clustering of visible points and VPL through the K-means clustering.

To show the results we will start from a reference image generated only with VPL (Figure 5.1.1), and with it we will see if the application of one, some or all of the algorithms improves the rendering time or not, without substantially losing the quality of the image.

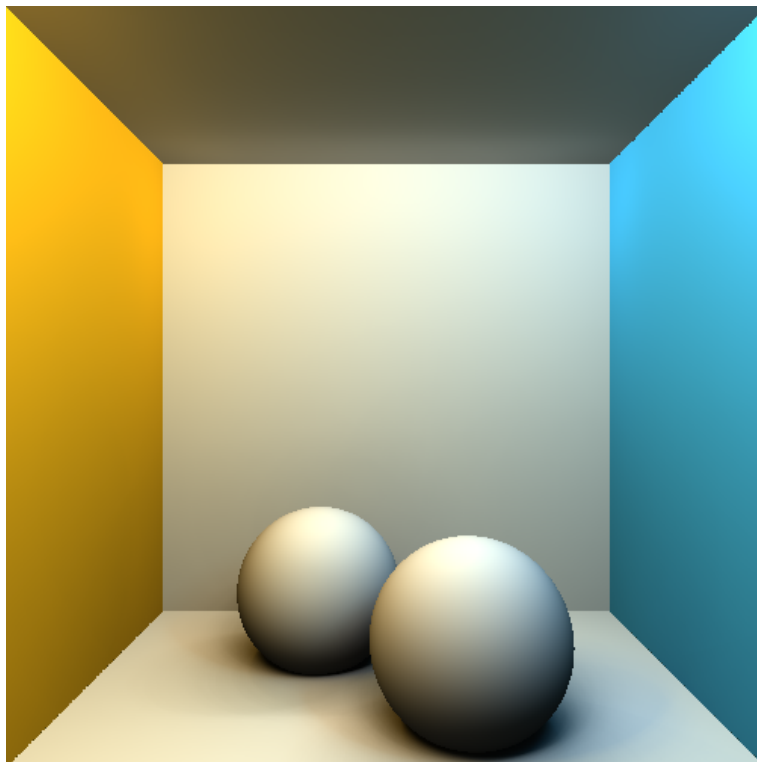


Figure 5.1.1: Reference Image. This image was generated using 2000 VPLs 2 Bounces, 2600 visible point clusters with 4 iterations in the k-means algorithm, 400 VPL clusters and 2 iterations in the k-means clustering.

We are going to do several tests comparing different parameters (Figure 5.1.2):

- Virtual Point Lights
- Bounces
- Visible point clusters
- VPLs clusters
- The number of iterations in the K-means algorithm.

Virtual Point Lights	VPL
Bounces	B
Visible Point Clusters	VpC
VPLs Clusters	$VPLC$
number of iterations in K-means(visible points)	$K-mVp$
number of iterations in K-means(VPLs)	$K-mVPL$
Preprocess Time	pT
Render Time	rT
Total Time	tT

Figure 5.1.2: Table of notation. of the parameters that we will use during all the results.

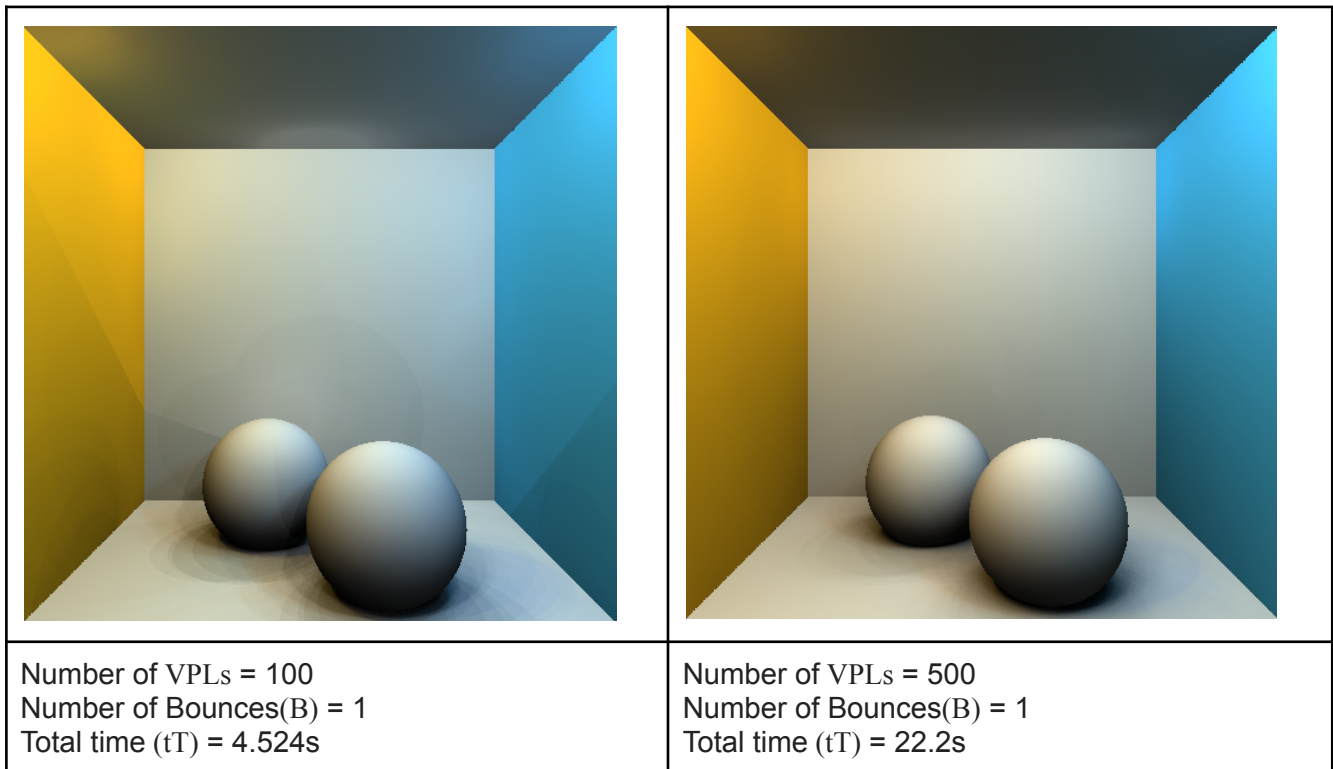
5.2 - Results for the Original VPL Algorithm Implementation

This can be called the base experiment, where we will use the original VPLs algorithm (detailed in Section 3.1) and see how the quality and the time of the images we can get depends on the amount of VPLs used in the implementation.

In the results the preprocessing time will not appear because the higher one was 0.008 seconds, and compared to the render time of 183.06 seconds we can ignore it. So we will only use total time.

The timings obtained for this experiment are linear with the number of VPLs. Every 22 VPLs approximately increases 1 second of rendering time.

About the quality of the images, the important place to spot the biggest differences are in the shadows, as the amount of VPLs increase the shadows are more and more smooth.



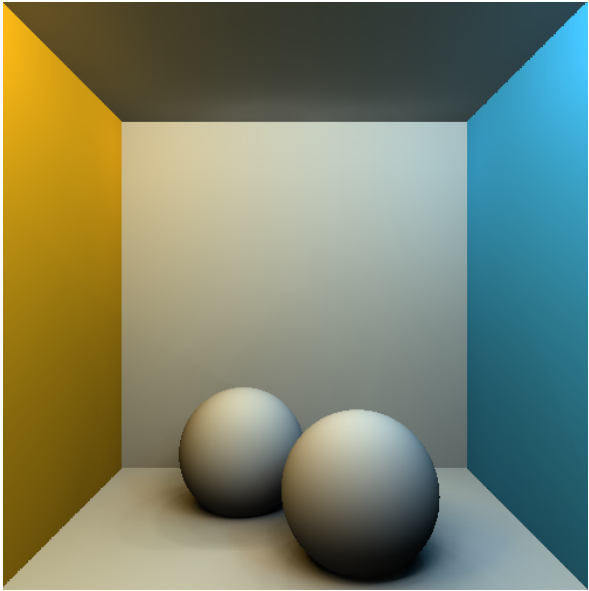
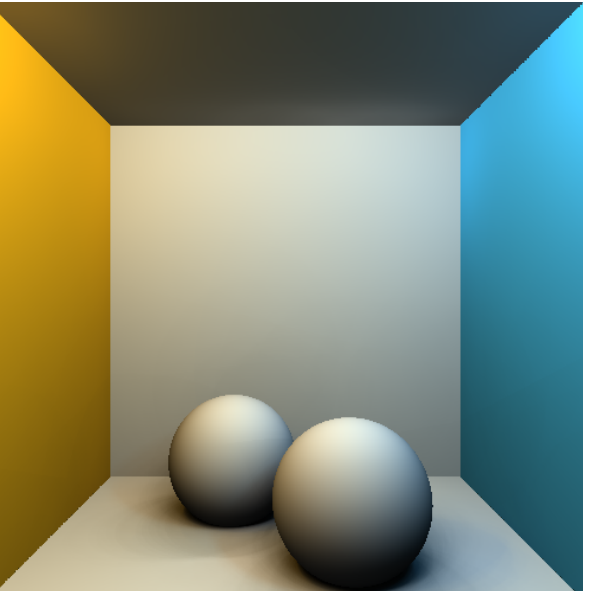
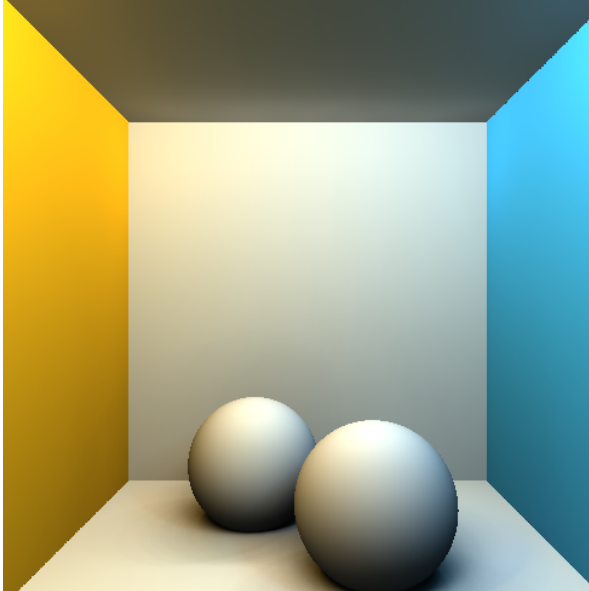
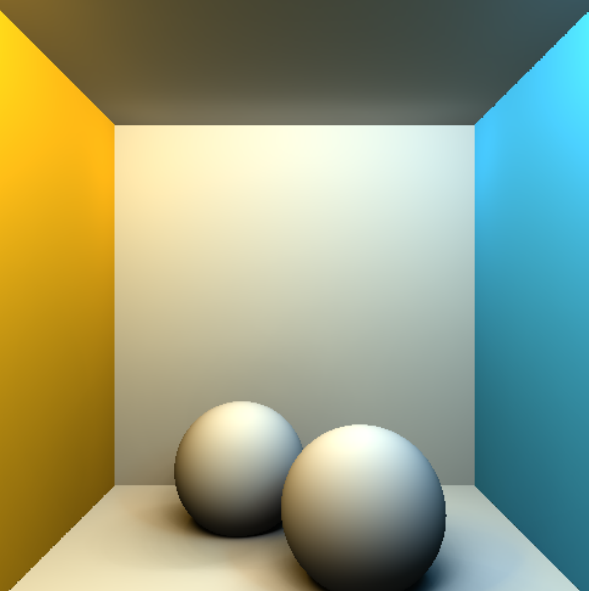
	
<p>Number of VPLs = 500 Number of Bounces(B) = 2 Total time (tT) = 45.7s</p>	<p>Number of VPLs = 1000 Number of Bounces(B) = 1 Total time (tT) = 44.74</p>
	
<p>Number of VPLs = 1000 Number of Bounces(B) = 2 Total time (tT) = 90.61s</p>	<p>Number of VPLs = 2000 Number of Bounces(B) = 2 Total time (tT) = 183.1s</p>

Figure 5.2.1: Scenes resulting from the tests, using only VPLs.

VPL and Time

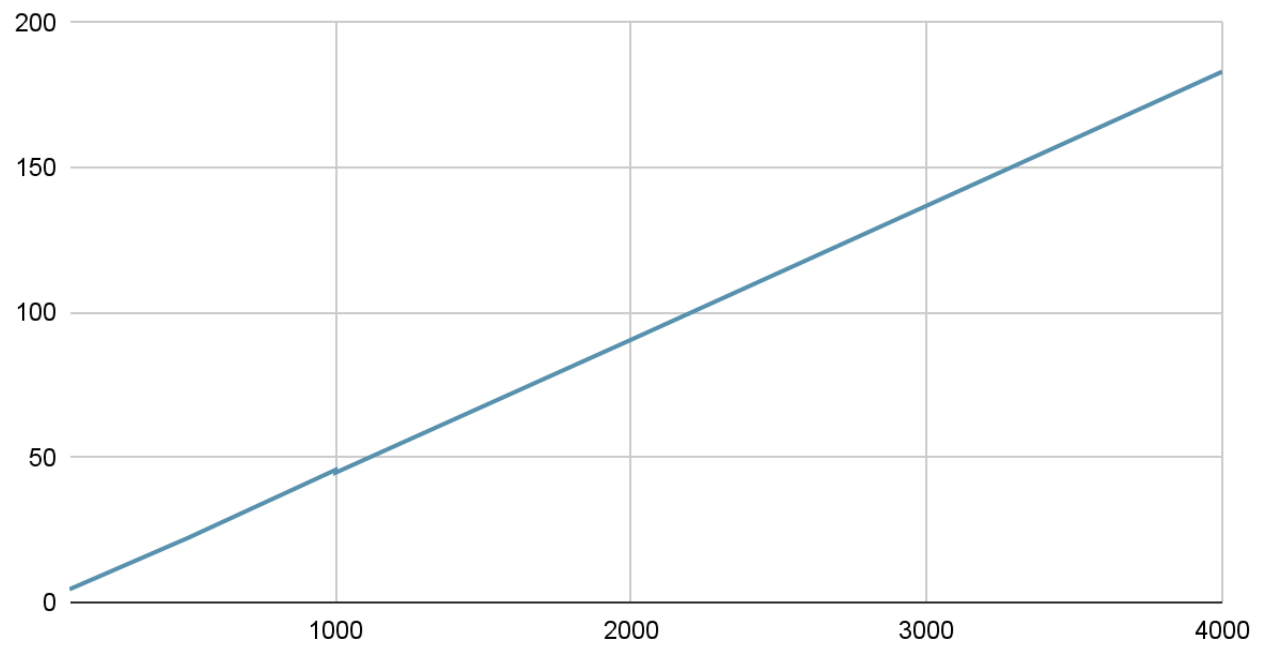
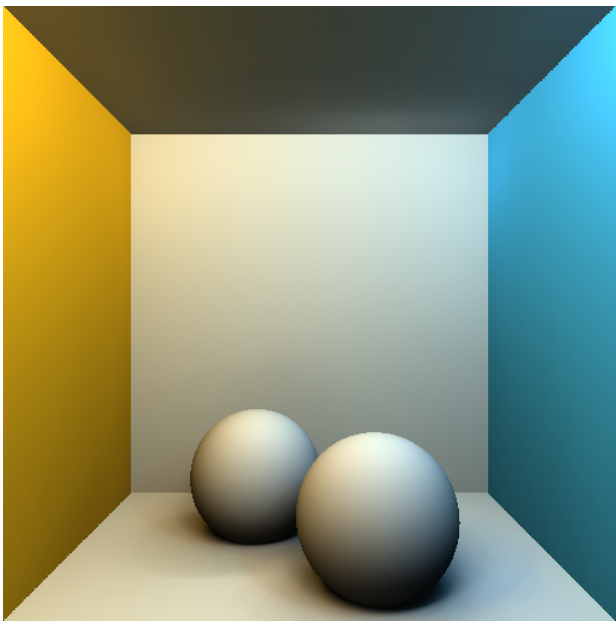
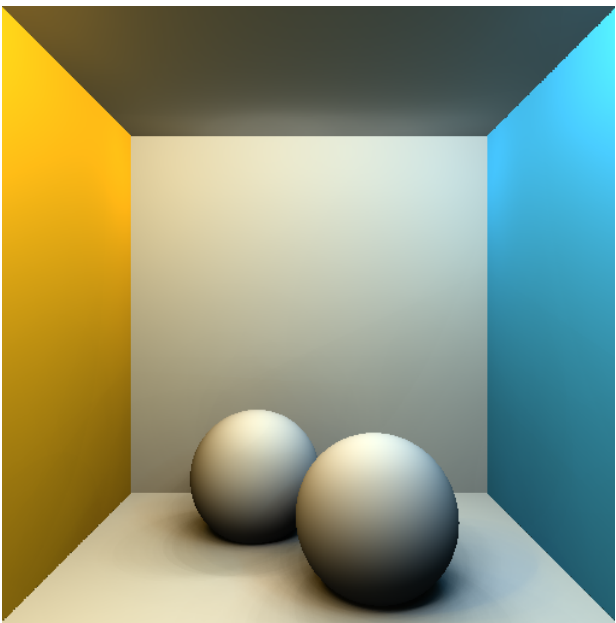


Figure 5.2.2: Graph that shows the linear proportion of the number of VPLs and the rendering time.

5.3 - Results using VPL clusters

The objective of this experiment is to see what is the amount of clusters that we can use to improve the rendering time and still maintain a good quality of the resulting image. As the amount of VPLs that we are going to use are 4000, as a reminder, (the number of total VPLs are the initial number of VPLs multiplied by the number of bounces). The maximum number of clusters that we intend to use is 2000 and the minimum amount is 100. (see Figure 5.3.1)

About the quality of the images, the important place to spot the biggest differences are in the shadows, as the amount of VPLs clusters increase the shadows are more smooth. Also, mention that the first image, which has 2000 VPL clusters, has the half rendering time that the reference image but the same quality.(see Figure 5.3.2)

	
Number of VPLs = 2000 Number of Bounces(B) = 2 Number of VPL Clusters = 2000 Preprocess time (pT) = 0.049s Render time (rT) = 91.649s Total time (tT) = 91.698s	Number of VPLs = 2000 Number of Bounces(B) = 2 Number of VPL Clusters = 1300 Preprocess time (pT) = 0.042s Render time (rT) = 64.931s Total time (tT) = 64.973s

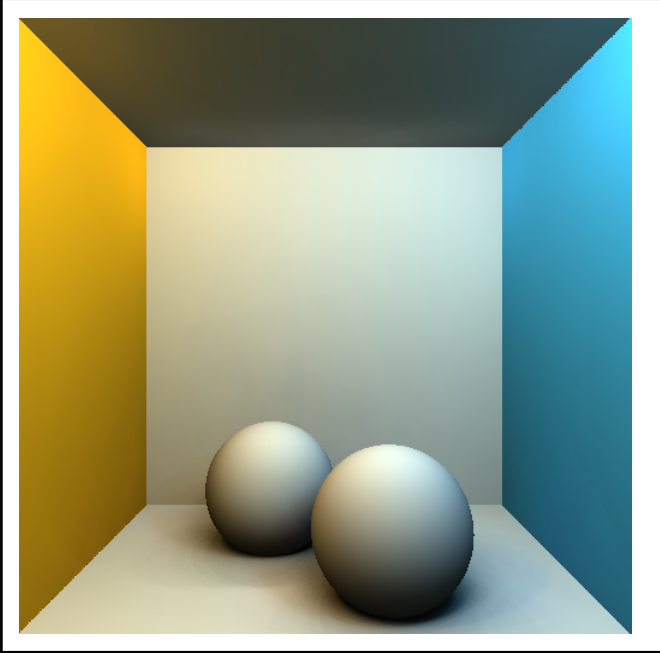
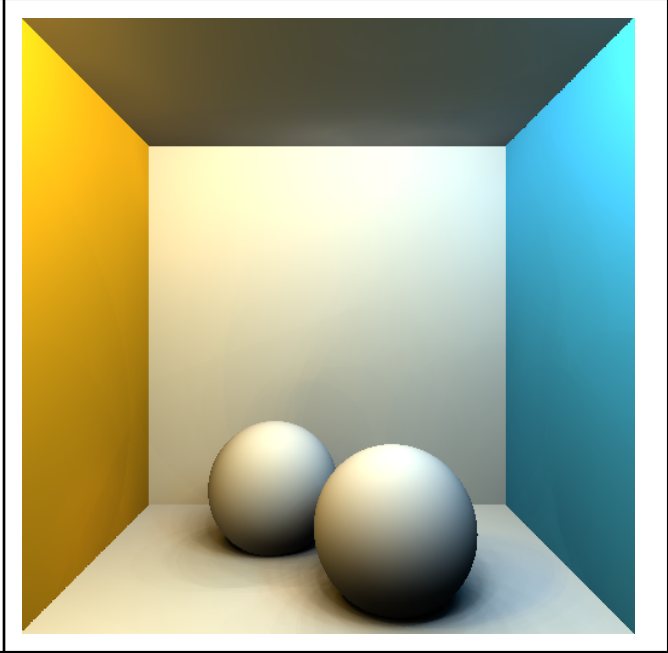
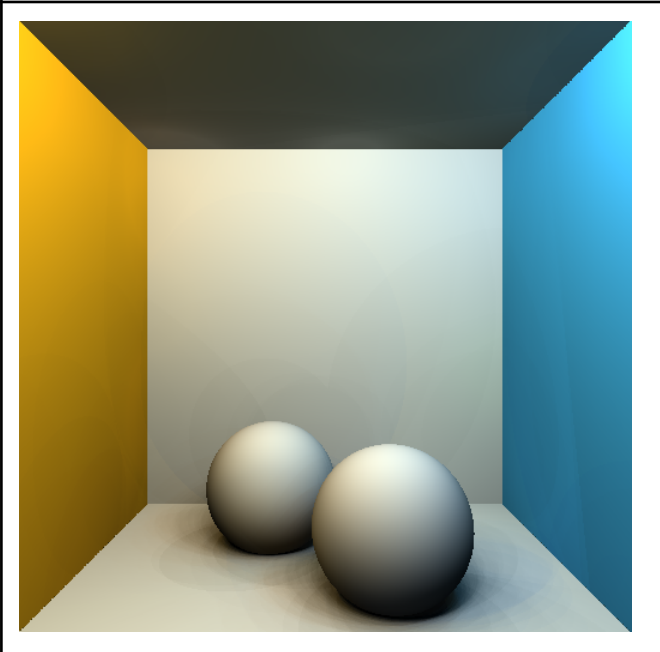
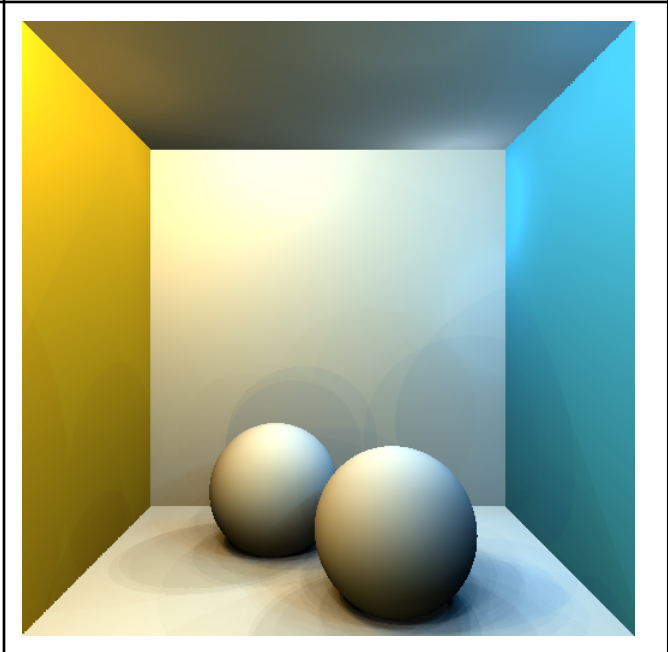
	
<p>Number of VPLs = 2000 Number of Bounces(B) = 2 Number of VPL Clusters = 800 Preprocess time (pT) = 0.024s Render time (rT) = 38.739s Total time (tT) = 38.763s</p>	<p>Number of VPLs = 2000 Number of Bounces(B) = 2 Number of VPL Clusters = 400 Preprocess time (pT) = 0.017s Render time (rT) = 18.688s Total time (tT) = 18.705s</p>
	
<p>Number of VPLs = 2000 Number of Bounces(B) = 2 Number of VPL Clusters = 200 Preprocess time (pT) = 0.012s Render time (rT) = 9.018s Total time (tT) = 9.03s</p>	<p>Number of VPLs = 2000 Number of Bounces(B) = 2 Number of VPL Clusters = 100 Preprocess time (pT) = 0.008s Render time (rT) = 4.521s Total time (tT) = 4.529s</p>

Figure 5.3.1: Scenes resulting from the tests, using only VPL clusterization.

As we can see in the following figure, or rather we cannot see, the image barely changes, almost the same smoothness in the shadows, so in conclusion this algorithm helps to relocate the VPLs with a lot more information not affecting the quality of the image, and reducing the rendering time in a half.

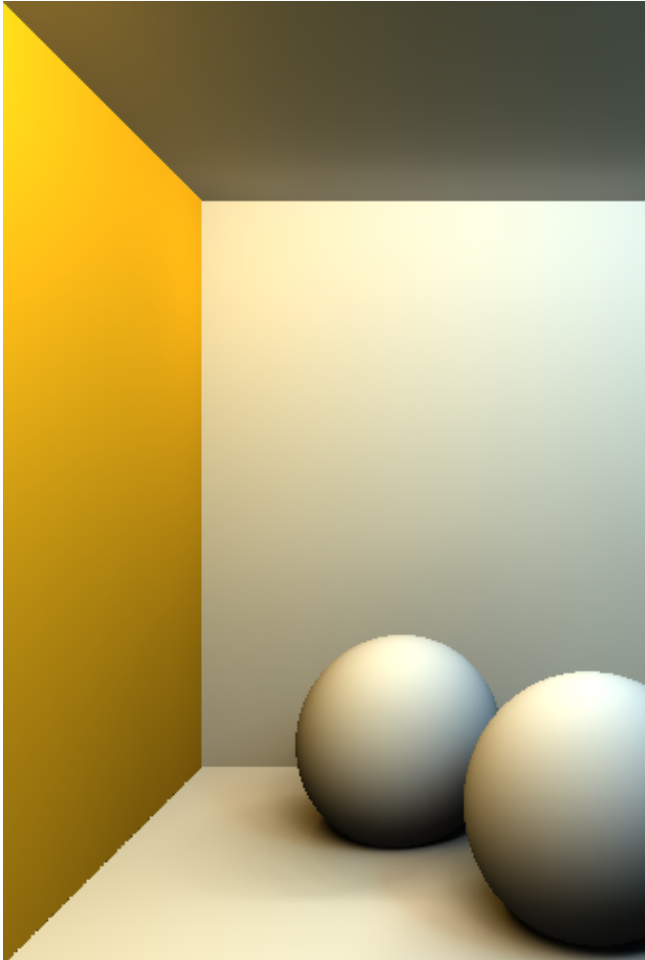
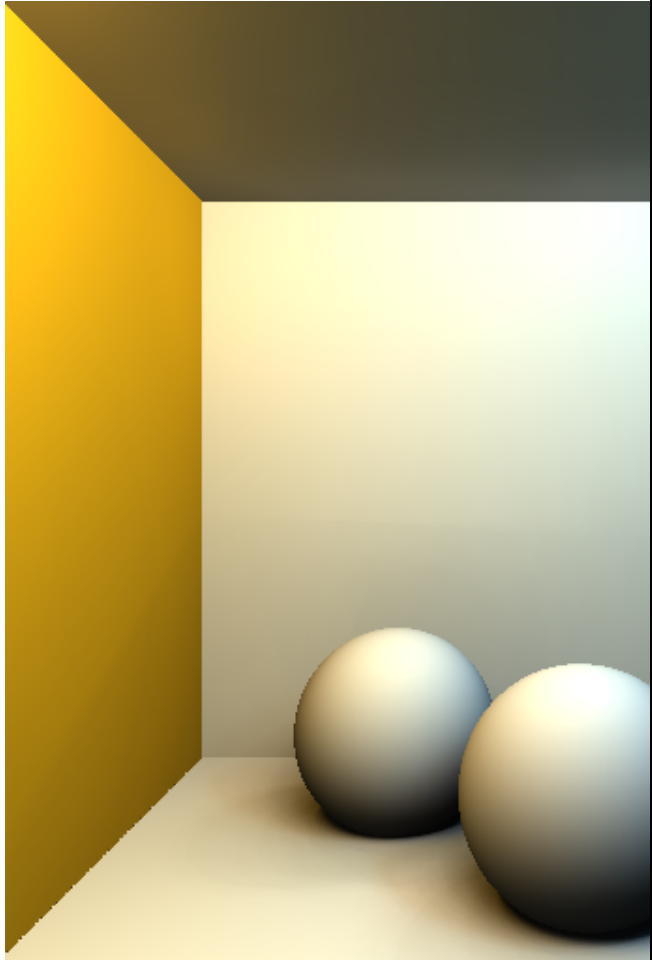
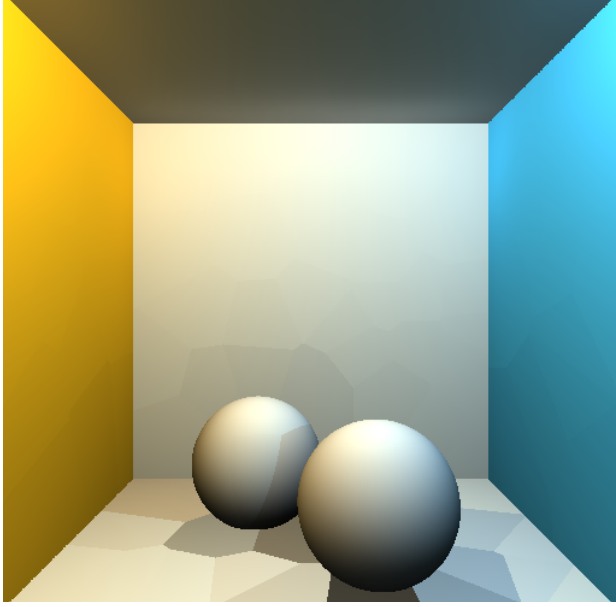
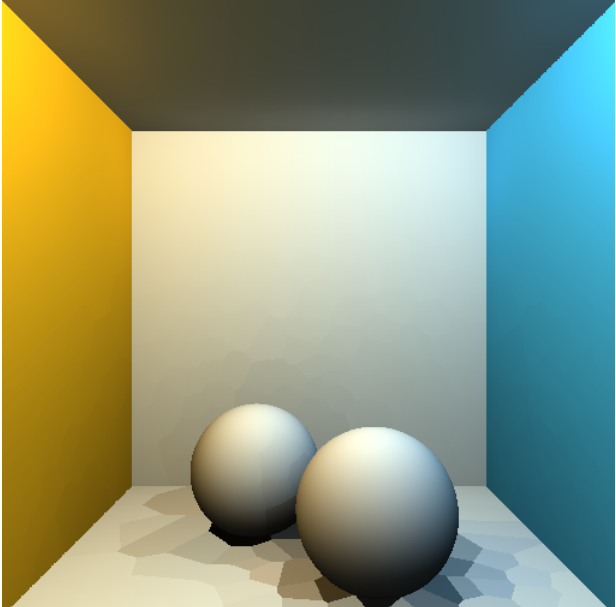
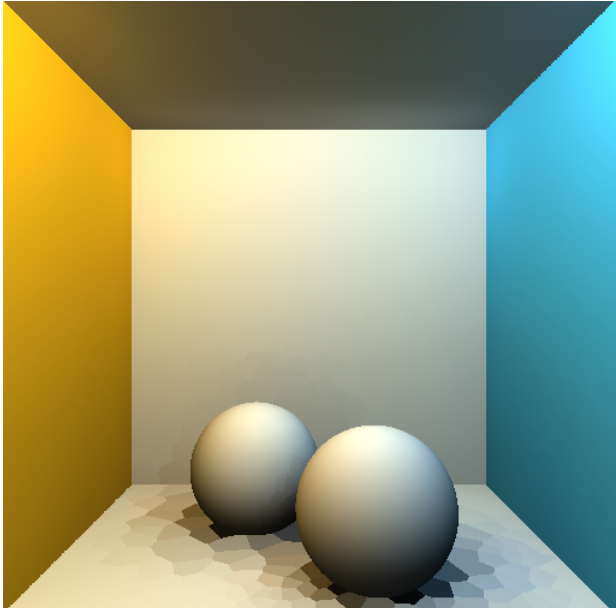
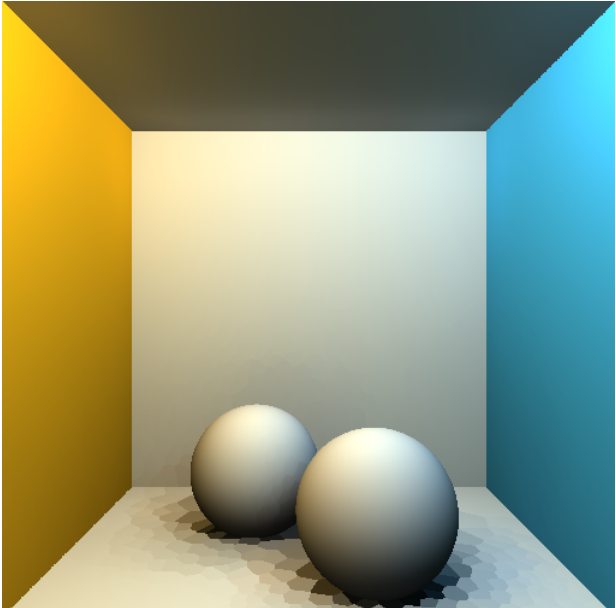
Original VPL Algorithm Implementation	VPL clustering Implementation
	
<p>Number of VPLs = 2000 Number of Bounces(B) = 2 Total time (tT) = 183.1s</p>	<p>Number of VPLs = 2000 Number of Bounces(B) = 2 K-means iterations ($K-mVPL$) = 2 Number of VPL Clusters = 2000 Preprocess time (pT) = 0.049s Render time (rT) = 91.649s Total time (tT) = 91.698s</p>

Figure 5.3.2: Image comparing. Scenes resulting from the tests, using only VPL clustering and half of time reduction.

5.4 - Results using visual point clusters

As our reference image was the one generated with 2000VPLs and 2 Bounces, ending with a total of 4000VPLs, in this experiment we are now looking for the perfect or optimus amount of visible point clusters(VpC). And we will fix the iterations for the clustering in 4 iterations.

	
<p>Number of VPLs = 2000 Number of Bounces(B) = 2 K-means iterations ($K-mVp$) = 4 Number of Visible Points Clusters = 100 Preprocess time (pT) = 1.937s Render time (rT) = 49.481s Total time (tT) = 51.418s</p>	<p>Number of VPLs = 2000 Number of Bounces(B) = 2 K-means iterations ($K-mVp$) = 4 Number of Visible Points Clusters = 500 Preprocess time (pT) = 9.559s Render time (rT) = 51.748s Total time (tT) = 61.307s</p>
	

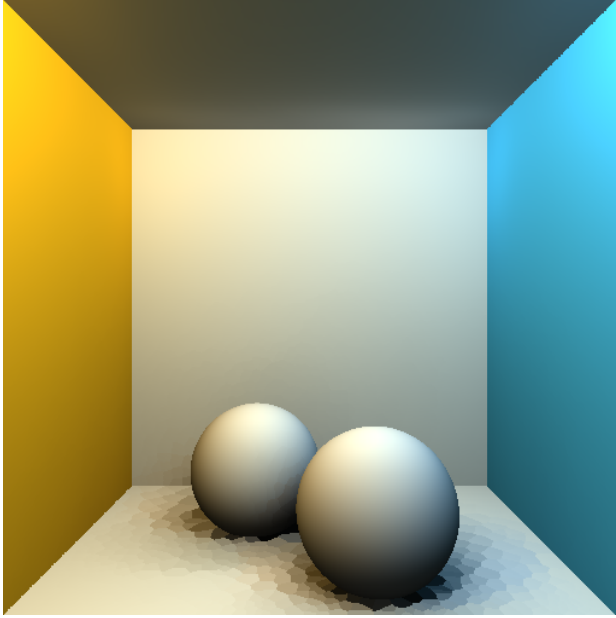
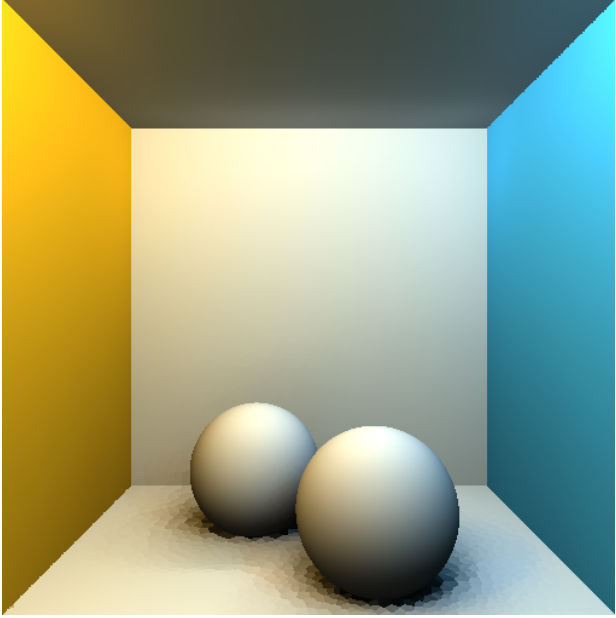
Number of VPLs = 2000 Number of Bounces(B) = 2 K-means iterations ($K-mVp$) = 4 Number of Visible Points Clusters = 1250 Preprocess time (pT) = 23.572s Render time (rT) = 49.542s Total time (tT) = 73.114s	Number of VPLs = 2000 Number of Bounces(B) = 2 K-means iterations ($K-mVp$) = 4 Number of Visible Points Clusters = 2600 Preprocess time (pT) = 49.992s Render time (rT) = 51.701s Total time (tT) = 101.693s
	
Number of VPLs = 2000 Number of Bounces(B) = 2 K-means iterations ($K-mVp$) = 4 Number of Visible Points Clusters = 4000 Preprocess time (pT) = 76.966s Render time (rT) = 53.661s Total time (tT) = 130.627s	Number of VPLs = 2000 Number of Bounces(B) = 2 K-means iterations ($K-mVp$) = 4 Number of Visible Points Clusters = 8000 Preprocess time (pT) = 157.899s Render time (rT) = 53.913s Total time (tT) = 211.812s

Figure 5.4.1: Scenes resulting from the tests, using VPL, and visible point clustering.

As we expected in this experiment we see a linear time increment in the preprocess, where we see an improvement of the quality of the image as we raise the amount of clusters. The final thing that this algorithm needs to compare with an image of only using VPL and no clustering with the image using visible points clusterization, we realize that is totally worth it in rendering time. Forward will be explained how this can be improved. But to have an idea would be using Fuzzy clustering[10], in which each cluster is influenced with its surrounding clusters, this will make a smoother shadow, and maintain the good rendering time.

Surprisingly the render time is almost the same always, there are not big differences, and this is because, at first we thought that the big differences would be when we do the rendering process, and here the big difference is when we do the clustering of visible points.

Even that we did another experiment to see how different is the computational cost of calculating the visibility $V(x, s_j)$ and calculating the emission $I(s_j, x)$, the material $f_r(x, \omega_i, \omega_o)$ and the geometry term $G(x, s_j)$ for each visible point.

The experiment was timing the visibility calculation versus timing the other part of the rendering equation, using 100 clusters versus 8000 clusters.

The result and the conclusions:

	100 clusters	8000 clusters
Visibility $V(x, s_j)$	0.055 seconds	4.559 seconds
Emission $I(s_j, x)$ Material $f_r(x, \omega_i, \omega_o)$ Geometry term $G(x, s_j)$	49.203 seconds	50.323 seconds

Table 5.4.2: Time table. Table that shows that calculation of Visibility is 3 times higher than the 3 other terms.

To put that in perspective we need to know the difference of time of each so we did that:

$$\frac{0,055_s}{100_c} = 0.00055_{s/c}$$

$$\frac{49.203_s}{262144_c} = 0.00018_{s/vp}$$

$$\frac{0,00055_{s/c}}{0,00018_{s/vp}} = 3,05_{vp/c}$$

It means that for each visibility $V(x, s_j)$ we calculate we can calculate 3 **Emission** $I(s_j, x)$, 3 **Material** $f_r(x, \omega_i, \omega_o)$ and 3 **Geometry term** $G(x, s_j)$, so yes there is a difference between them. But the calculations that have to be done to calculate the visibility of 8000 centroids represent less than **3%** of the total sum of calculations that have to be done in total, so the fact that they are three times more expensive does not influence too much in the total time spent.

To add to the information explained above, we have to mention a way to reduce the clustering time, a way not applied in this thesis, which will mean a high total time (tT) reduction. This can be done while the clustering, of visible points, instead of re assign all the visible to one centroid in each iteration, we can do the reassignment one each two visible points, due to the cost of K-means clustering Big O = n^3 , it could be a 6 times reduction of the time.

5.5 - Results using VPL clusters and Visible point clusters

With this experiment we intend to see if the use of the two clustering algorithms at the same time will improve the final time. So far we have been able to verify that the clustering of visible points gives us a very high rendering speed, but lowers the quality in the shadows, and the VPL clustering keeps a high quality of the image, and reduces the rendering time in a half.

So we could think that this would not be the case due to the preprocessing time generated by the clustering algorithm in visible points, in any case we believe that its rendering time together with the improvement in rendering time of the Virtual Point Lights clustering algorithm will achieve a lower total final time than the others.

Now we will check if one algorithm compensates the other. To be able to do it we are going to put the parameters of the reference image at 4000VPLs and 2600 clusters of visible points.

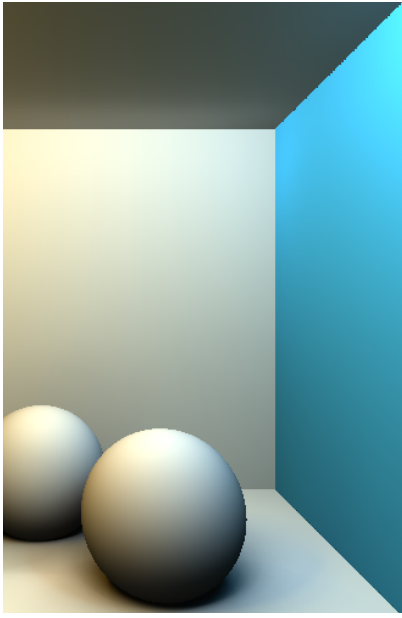
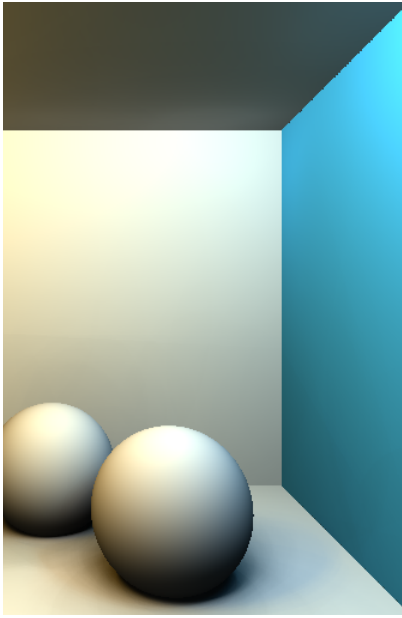
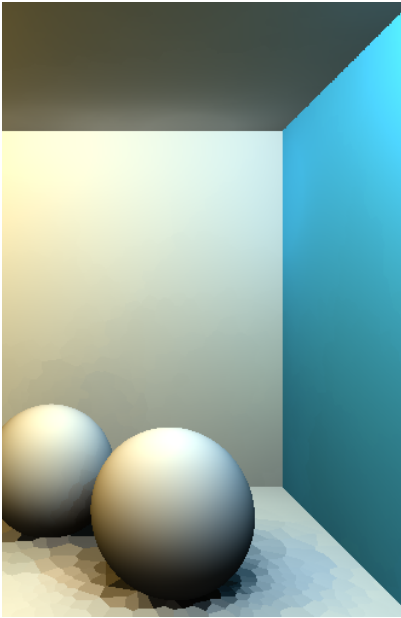
Reference image, Original VPL Algorithm	VPL clustering Implementation	VPL clusters and Visible point clusters
		
Preprocess time (pT) = 0 Render time (rT) = 183.1s Total time (tT) = 183.1s	Preprocess time (pT) = 0.049s Render time (rT) = 91.649s Total time (tT) = 91.698s	Preprocess time (pT) = 48.934 Render time (rT) = 4.915 Total time (tT) = 53.849

Figure 5.5.1: Scenes resulting from the tests, using VPLs clusters and Visible Point clusters.

As we can well see in figure 5.5.1, the use of clustering algorithms have both improved the final image synthesis time, since both improve rendering time, and

although the clustering of visible points has a high time in the preprocess, it is not high enough to negatively affect the final result.

Section 6 - Discussion

The goal of this study or project was to learn about global illumination through the usage of Virtual Point Lights, and try to improve the timings of rendering by applying K-means which is a clustering method, which aims to partition a set of n observations into k groups in which each observation belongs to the group whose mean value is closest.

We started with a base ray tracer, in which we had to familiarize ourselves in the shortest possible time, in this part it took us a little longer than expected.

Secondly, we implemented the original VPL algorithm, which was implemented correctly and we got good results with that implementation. Some artifacts that could be solved lately.

Thirdly we implement the K-means clustering algorithm in different ways, thus creating a total of 3 more classes. To all this add the implementation of extra classes to be able to visualize the results, that is, to be able to see the VPLs, the clusters and their centroids in the scene.

Finally, we performed different tests in order to provide our initial objectives. The tests consisted of executing the same scene using the greatest possible variety of parameters, and thus comparing the execution times and the qualities of the resulting images.

6.1 Limitations

Since we found good results in our tests, it would have been interesting to further test the model with more complex scenes to render, scenes with larger objects, with different textures, shapes and more light sources. Although the Cornell box has been and is very useful for when time is limited and the real need is to do a large number of tests.

Due to the short deadline we had, it was impossible to perform the tests mentioned above, but with the ones already done proving that this algorithm is viable to improve rendering times, the more complex scene test would have been interesting to see more limitations and maybe some artifacts that can appear due to the algorithm.

6.2 Future work

As mentioned during the explanations of the algorithms and in general in different parts of the thesis, there are two main items to improve, both would be in the clustering algorithm, although each one affects it differently.

One item would be the quality of the shadows, which basically show the clusters too much, this is because between cluster and cluster the visibility of the VPLs is very different. Remember that Visibility is only calculated by the centroid and is shared to the visible points of that same cluster. This could be solved by changing from a strict clustering algorithm like the one we have used, to one where each element could belong to more than one cluster, called Fuzzy clustering.

Fuzzy clustering is a class of clustering algorithms where each element has a fuzzy membership degree to the groups. This type of algorithm arises from the need to solve a deficiency of exclusive grouping, which considers that each element can be unequivocally grouped with the elements of its cluster and, therefore, does not resemble the rest of the elements, in our case this translates into highly differentiated clusters in the image.

The last item to solve or improve would be the need to reduce the clustering time, due to the amount of iterations that the clustering have to do to relocate the cluster center, to improve that, instead of going through all the visible points one by one , we can do the reassignment one each two visible points, during all the middle iterations, and in the last one use them all or assign the visible points without cluster to the cluster of the visible point next to it. The best option would be to use them all in the last iteration, because assigning a visible point to the cluster of the neighboring visible point, could cause some artifacts in the image, due to the cost of K-means clustering $f(n) = O(n^3)$ it could be a 6 times reduction of the total time of the clusterization.

Section 7 - Conclusions

The main objective of this thesis was to make a shader based on the Virtual Point Lights algorithm, which in itself is already more efficient than the Path tracer, and improve this shader from a clustering algorithm called K-means, the which has been used to cluster two different elements, the visible points and the virtual point lights.

In the end, the results showed that we achieved objectives we were aiming for. We proved that the Virtual Point Light algorithm is a path to follow to improve the timings of rendering.

Also we proved that a clusterization of different elements in the image helps to reduce the rendering time by quite a lot, in our case was possible with the K-means algorithm, which is an algorithm rather thought for big data management and with that we also proved that we can maintain a good quality of the image even when we are rendering two times faster than without using clusterization.

Therefore, we can say that the objectives of our Final Grade Thesis have been successfully achieved.

Section 7 - Bibliography

- [1] Alexander Keller. Instant radiosity. In Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '97, pages 49–56, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.
- [2] Philip Dutré. Global Illumination Compendium September 29, 2003. Computer Graphics, Department of Computer Science Katholieke Universiteit Leuven
<https://people.cs.kuleuven.be/~philip.dutre/GI/TotalCompendium.pdf>
- [3] Rui Wang, Rui Wang, Kun Zhou, Minghao Pan, Hujun Bao. An Efficient GPU-based Approach for Interactive Global Illumination. State Key Lab of CAD&CG, Zhejiang University University of Massachusetts Amherst
<https://cseweb.ucsd.edu/~ravir/274/15/papers/a91-wang.pdf>
- [4] Jiří Kantor. Global Illumination through Virtual Point Lights. Master's thesis. Masaryk University, Faculty of Computer science. Brno, Spring 2015
<https://is.muni.cz/th/payge/report.pdf>
- [5] Adrian Jarabo Raul Buisan Diego Gutierrez. Bidirectional Clustering for Scalable VPL-based Global Illumination. Universidad de Zaragoza
https://graphics.unizar.es/papers/Jarabo_ceig2015.pdf
- [6] Carsten Dachsbacher, Jaroslav Křivánek, Prague Miloš Hašan, Adam Arbree, Bruce Walter, Jan Novák. Scalable Realistic Rendering with Many-Light Methods. Karlsruhe Institute of Technology, Charles University, UC Berkeley Autodesk, Inc. Cornell University Jan Novák Karlsruhe Institute of Technology
https://cgg.mff.cuni.cz/~jaroslav/papers/2013-mlstar/eg2013star_manylights.pdf
- [7] Peter Shirley. Ray Tracing in One Weekend. Copyright 2018.
<https://www.realtimerendering.com/raytracing/Ray%20Tracing%20in%20a%20Weekend.pdf>
- [8] Juan Diego Polo. 80 Horas para renderizar cada frame del libro de la selva. Artículo. 19 abril, 2016
<https://www.whatsnew.com/2016/04/19/80-horas-para-renderizar-cada-frame-en-la-nueva-pelicula-del-libro-de-la-selva/>
- [9] James T Kajiya. California Institute of Technology. Computer Graphics Volume 20 Issue. 4 Aug. 1986 pp 143–150
<https://dl.acm.org/doi/10.1145/15886.15902>

[10] Yang, M.S.(1993). «A Survey of Fuzzy Clustering». Mathl. Comput. Modeling 18 (11): 1-16.
<https://web.archive.org/web/20131215045842/https://s3-us-west-2.amazonaws.com/mlsurveys/21.pdf>

[11] https://en.wikipedia.org/wiki/K-means_clustering