Pontifícia Universidade Católica
do Rio de Janeiro

André de Souza Moreira

Hybrid Cloud Rendering for Industrial-Plant CAD Models

Tese de Doutorado

Thesis presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Informática.

Advisor: Prof. Waldemar Celes Filho

Rio de Janeiro
March 2020

Pontifícia Universidade Católica
do Rio de Janeiro

André de Souza Moreira

Hybrid Cloud Rendering for Industrial-Plant CAD Models

Thesis presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Informática. Approved by the Examination Committee.

Prof. Waldemar Celes Filho
Advisor
Departamento de Informática – PUC-Rio

Prof. Alberto Barbosa Raposo
Departamento de Informática – PUC-Rio

Prof. Marcelo Gattass
Departamento de Informática – PUC-Rio

Prof. Marcos de Oliveira Lage Ferreira
Instituto de Computação – UFF

Prof. Renato Fontoura de Gusmão Cerqueira
IBM Research – IBM

Rio de Janeiro, March 13th, 2020

André de Souza Moreira

André Moreira received his Bachelor degree in Computer Science from the Federal University of Maranhão (UFMA) in 2019. He also holds a Master degree in Computer Science with emphasis in Computer Graphics from PUC-Rio. Since graduation, he is involved in R&D projects, initially at NCA Institute (UFMA) and more recently at Tecgraf Institute (PUC-Rio). The student's research interests include Visualization, Real-Time Rendering, Medical Imaging, Artificial Intelligence and Data Science.

Bibliographic data

Moreira, André de Souza

Hybrid Cloud Rendering for Industrial-Plant CAD Models / André de Souza Moreira; advisor: Waldemar Celes Filho. – Rio de janeiro: PUC-Rio, Departamento de Informática, 2020.

v., 75 f: il. color. ; 30 cm

Tese (doutorado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui bibliografia

1. Informática – Teses. 2. Renderização na nuvem;. 3. Modelos CAD;. 4. Modelos Massivos;. 5. Plantas industriais.. I. Celes Filho, Waldemar. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Acknowledgments

I would like to express my deepest thanks:
To my advisor, Professor Waldemar Celes, for all contributions and support that turned this work possible and more valuable.
To my family for unconditional support and affection.
To my darling Suellen Motta, for your companionship, opinions, and support.
To all my friends and colleagues that somehow contributed for this work, especially from GEDi/Tecgraf, for sharing knowledge and experience over the years and for the inspiration that created this work.
Thanks to PUC-Rio and Tecgraf, for their financial assistance and support.
This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Abstract

Moreira, André de Souza; Celes Filho, Waldemar (Advisor). **Hybrid Cloud Rendering for Industrial-Plant CAD Models**. Rio de Janeiro, 2020. 75p. Tese de doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Industrial-plant CAD models play an important role in engineering project management. Despite the advances in computing power in past decades, rendering these models remains challenging due to their complexity and large data volume. Different areas of computing have succeeded in adopting cloud services to process massive data. However, when it comes to cloud rendering, there is still a lack of cloud rendering services for CAD models. In this paper, we propose a hybrid cloud rendering architecture for CAD models, dividing the rendering task between client and server. In addition to reducing server overhead, this approach affords greater resilience to the system against variations of network latency. Finally, this work also introduces a metaheuristic-based workload selection algorithm to determine the set of objects to be drawn on the client side. Our results demonstrate that the proposed methodology allows efficient visualization of massive CAD models even under adverse conditions such as clients with limited devices and high connection latency. Lastly, we discuss remaining research opportunities for cloud rendering, opening avenues for future improvements.

Keywords

Cloud Rendering; CAD Models; Massive Models; Industrial Plants.

Resumo

Moreira, André de Souza; Celes Filho, Waldemar. **Renderização Híbrida na Nuvem para Modelos CAD de Plantas Industriais**. Rio de Janeiro, 2020. 75p. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Os modelos CAD de plantas industriais desempenham um papel importante no gerenciamento de projetos de engenharia. Apesar dos avanços do poder computacional nas últimas décadas, a renderização destes modelos continua sendo um desafio devido à sua complexidade e ao grande volume de dados. Diferentes áreas da computação obtiveram êxito ao adotar serviços na nuvem para processar dados massivos. Contudo, quando se trata de rendering na nuvem, ainda há uma deficiência destes serviços para modelos CAD. Neste trabalho, propomos uma arquitetura de rendering híbrido na nuvem para modelos CAD, dividindo a tarefa de renderização entre o cliente e servidor. Além da diminuição da sobrecarga do servidor, esta abordagem garante ao sistema maior resiliência a variações de latência da rede. Neste trabalho também é introduzido um algoritmo de seleção de carga de trabalho baseada em metaheurística para determinar o conjunto de objetos a ser desenhado no lado do cliente. Nossos resultados demonstram que a metodologia proposta permite a visualização eficiente de modelos CAD massivos mesmo em condições adversas, como clientes com dispositivos limitados e latência alta na conexão. Por fim, discutimos as oportunidades de pesquisa restantes para renderização em nuvem, abrindo caminhos para melhorias futuras.

Palavras-chave

Renderização na nuvem; Modelos CAD; Modelos Massivos; Plantas industriais.

Table of contents

List of figures

List of tables

List of abreviations

AEC	–	Achitecture & Construction
ANNs	–	Artificial Neural Networks
BIM	–	Building Information Modeling
CAD	–	Computer-Aided Design
CSR	–	Complete Spatial Randomness
DIBR	–	Depth Image Based Rendering
DMA	–	Direct Memory Access
DNNs	–	Deep Neural Networks
FPS	–	Frames per Second
GPGPU	–	General Purpose Graphics Processing Unit
GUI	–	Graphical User Interface
HUD	–	Heads-up display
IaaS	–	Infrastructure as a service
LAN	–	Local Area Network
LOD	–	Level of Detail
QoE	–	Quality of Experience
QoS	–	Quality of Service
WAN	–	Wide Area Network

It's the possibility of having a dream come true
that makes life interesting.

Paulo Coelho, *The Alchemist.*

1
Introduction

The efficient rendering of massive CAD (Computer-Aided Design) models is a long-time challenging task in computer graphics. These models play a significant role during the whole life cycle of engineering projects, especially in facility management. In order to make their use effective, CAD models must retain all details of the real-world product. This fine-grained representation leads to massive data volume, as shown in Fig. 1.1, requiring high processing power to handle them efficiently.

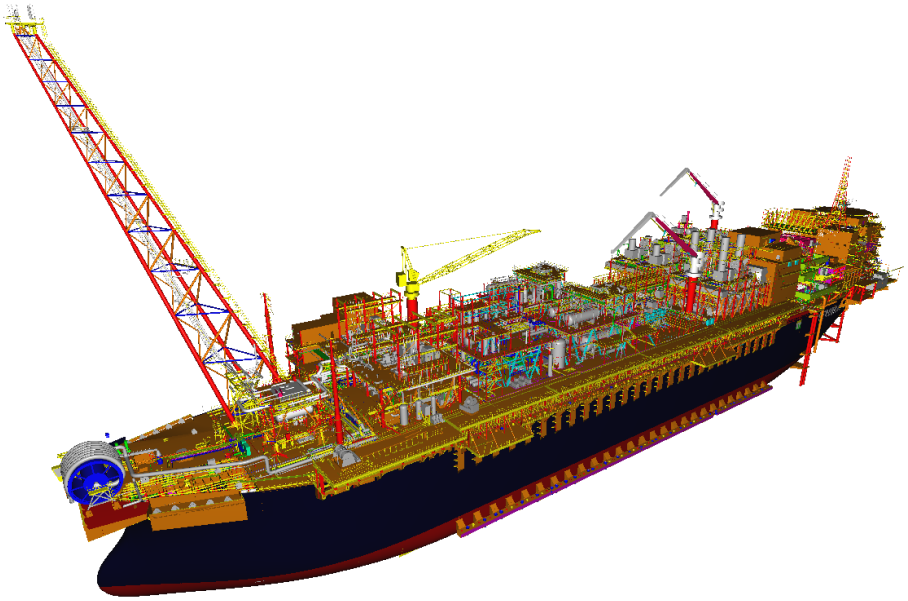


Figure 1.1: Example of CAD model of an oil-platform plant. This model contains nearly 2 millions objects.

As the industrial-plant projects have increasingly become more complex, more risks in terms of costs, deadlines, and safety are involved. The Building Information Modelling (BIM) aims to mitigate the risks by promoting the use of 3D CAD models as a central database to support decision-making, activity planning, and execution (Chen et al., 2005; Dunston & Wang, 2005; Rivard, 2000). The recent popularization of this approach led to CAD models to become even more detailed, massive, and complex. At the same time, we have witnessed both a slowdown in advances of computational power and

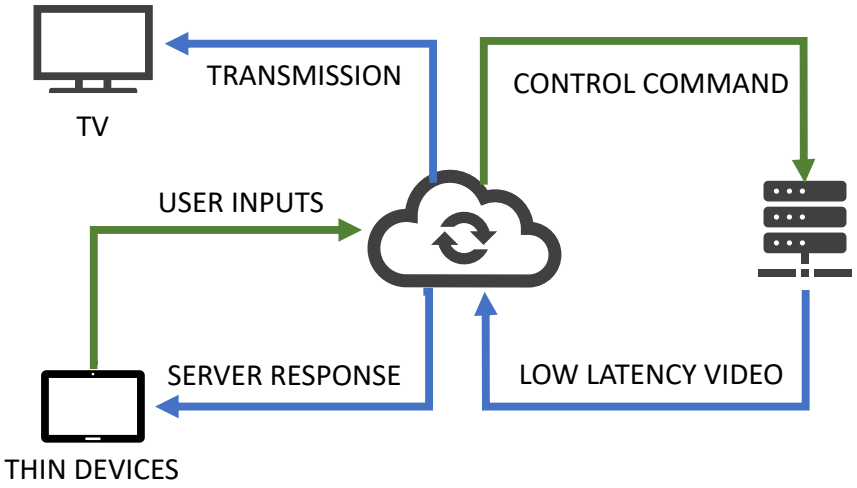


Figure 1.2: Schematic representation of cloud-gaming services. The user inputs are reported to the server, which runs the game logic and produces the game rendering to the client.

the popularization of portable devices, such as tablets. The rendering of such models for interactive applications has become more challenging.

In the past few years, several computer areas have experienced an increase in data volume. As a solution, most of them are shifting the heavyweight tasks to the cloud. When it comes to rendering graphics contents, the cloud gaming industry has achieved a notable progress (Shea et al., 2013). The server is responsible for all graphics processing, including the game execution. The rendered images are encoded by the server and then streamed to the client. The client only acts as a dummy terminal, decoding the received images, listening to user inputs and reporting them to the server. The server can also be used for streaming the game match to an audience. The whole cloud gaming workflow is depicted in Figure 1.2.

When we look beyond the cloud gaming field, though, there is a lack of studies about how to employ cloud rendering in other visualization applications. When it comes to the rendering of industrial-plant CAD models, the existing cloud rendering solutions do not suit the needs since they are mostly general-purpose services. They lack efficient algorithms and optimizations to handle these models properly. In this work, we aim to fill the gap between cloud rendering services and the rendering of industrial-plant CAD models. Our solution relies on a hybrid rendering approach, in which rendering tasks are performed both in client and server sides.

In our method, we split the model into two disjoint sets of objects, one to be rendered by the server and the other to be rendered by the client. While the user is navigating through the scene, each side renders its object set. The server produces a dual depth-augmented image and streams it to the

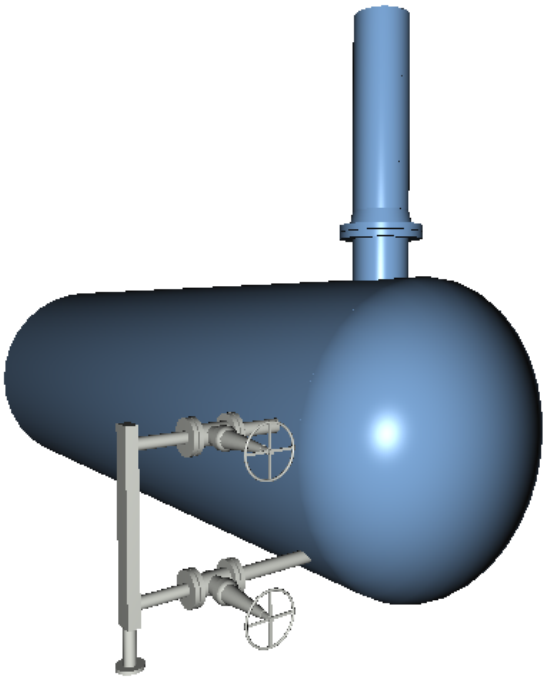


Figure 1.3: Example of valves and equipment in an industrial plant model. The majority of the objects are a composition of simple geometries like spheres, cylinders, parallelograms, and others.

client. The client combines both remote and local images to produce the final image. We decouple the client from the server by using an asynchronous event-based communication model between them. This way, we prevent halting the client operations waiting for a delayed server response. In such delayed cases, the client performs an image-based rendering on the last available remote image to provide a temporary result to the user. Due to this asynchronous communication, the server uses a prediction model for the client camera in order to anticipate future remote frames.

Since our cloud rendering targets a specific domain, we can benefit from prior known particularities of CAD models to achieve some improvements. For example, CAD models of industrial plants are mostly composed of objects with simple geometries, such as spheres, cylinders, and others (Figure 1.3). We describe these objects using only their parameters, which is more efficient for storage, transmission, and rendering. Besides, the high redundancy of these objects allows us to boost the rendering performance using instanced rendering.

We also present a novel metaheuristic-based workload selection to determine the objects to be rendered on the client side. This method attempts to provide spatial awareness to the user. When the server connection is lost or suffers from high latency, the user can still navigate through the scene using only local rendering.

Lastly, the conduction of this work raised some possible research opportunities in different areas of computing. For example, we encourage the use of machine learning techniques, like image completion, to improve the final image quality. Autonomic computing is another important investigation field to mitigate the high resource variations commonly seen in cloud environments. We detail each one of them and discuss how they can be used to provide overall improvements for the current state of this work.

The results show that our system provides a reasonable experience to the users, even on constrained networks. We assessed our method in terms of rendering performance, latency analysis, image quality, and workload division.

As a disclaimer, this work is not concerned about scalability matters. Although this is an important subject for cloud rendering services, we intend to address it in future works. At this moment, the system architecture scalability relies on vertical-scaling (i.e., adding more resources to the existing server).

1.1 Contributions

To the best of our knowledge, this is the first work that addresses the rendering of industrial-plant CAD models on the cloud. This study also resulted in two papers published on premier conference proceedings, as follows:

- A. Moreira, P. Ivson and W. Celes, "Hybrid Cloud Rendering System for Massive CAD Models," 2018 31st SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI), Paraná, 2018, pp. 234-241.
- A. Moreira and W. Celes, "Metaheuristic-Based Workload Selection for Hybrid Cloud Rendering of CAD Models," 2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Sydney, Australia, 2019, pp. 87-94.

Our additional contributions are:

1. **Final image with better quality**. The image produced on the server is compressed before its transmission to the client. This process reduces the image quality. On the other hand, the image produced by the client is not compressed, preserving its high fidelity. Hence, the combination of these two images results in a more pleasant looking image than if it was fully rendered on the server side.

2. **Workload division**. We divide the client object set into two disjoint sets: static and dynamic. The former is determined during the connection establishment phase. This set aims to afford spatial awareness to the

user. The latter set is a view-dependent object set, thus it is constantly updated by the server. Its main goal is to ensure that the objects closer to the client camera will be rendered on the client. Consequently, we prevent wasting the limited client resources with unnecessary work.

In addition, our approach also poses the following advantages:

1. **Efficient data representation**. Instead of using only the traditional triangular meshes for describing the geometry of all objects, we also represent some well-known shapes (spheres, cylinders, torus, and others) using only their parameters. This representation is much more efficient in terms of rendering, transmission, and storage. In our tests, we achieved a compression ratio of up to 87% of the original model size.

2. **Less dependency on network conditions**. Our asynchronous-based communication model, along with the hybrid rendering, ensures that the client application is always responding to the user events. When the server response is delayed or in cases of disconnections, the client application shows partial results to the user. These partial results are obtained by the combination of the local rendering image with the warping on the last available remote image. When the required remote image is finally available on the client side, the partial result is replaced by the complete final image.

3. **Resource savings**: although one of the significant contributions of cloud rendering systems is enabling low-end devices to handle massive models, high-end computers can also make use of cloud rendering solutions. When it comes to CAD models, this can be quite common since these models are stored on the server. In this case, the hybrid approach can fully take advantage of the available processing power by assigning more jobs to be performed on the client side. Consequently, the burden on the server is reduced, allowing it to handle larger scenes and/or more users.

1.2 Document Organization

The remainder of this work is organized as follows. Chapter 2 introduces cloud rendering techniques and discusses them in terms of their classification, advantages, and latency analysis. In addition, that chapter analyzes the existing alternatives and draws a parallel with ours, enlightening the reasons the existing works are unfeasible to suit the needs of rendering industrial-plant CAD models.

We discuss our proposal for a hybrid cloud rendering method for industrial-plant CAD models in Chapter 3. We present our method in terms of data format, client, and server architectures. Then, in Chapter 4, we detail our metaheuristic-based workload selection to choose the objects to be drawn on the client. There, we establish some criteria in order to afford spatial awareness to the user. In Chapter 5, we report the results of our tests in terms of rendering performance, image quality, and workload division.

In Chapter 6, we point out some research opportunities in the field of Cloud Rendering. During our research, we identified some further promising investigations that could enhance our solution. Lastly, in Chapter 7, we present our final remarks and future works.

2
Cloud Rendering Background

2.1
Introduction

A remote or cloud rendering system consists of two network-connected computing devices, where one is responsible for rendering graphical contents and the other for displaying them. The computer that provides the rendering services is named server, while the consuming computer is known as a client. Apart from the generation of remote frames, a cloud rendering system can also involve rendering tasks on the client side. This is the case, for example, of our hybrid cloud rendering discussed in this work.

In interactive cloud rendering systems, the user sends commands to the application using an input device, like keyboard and mouse. The client interprets these events and notifies the server about them. Once notified, the server updates its internal state and produces a new remote image, encodes, and transmits it to the client. Once the remote image is available on the client side, the rendering system decodes and displays it on the screen. This entire workflow is depicted in Figure 2.1.

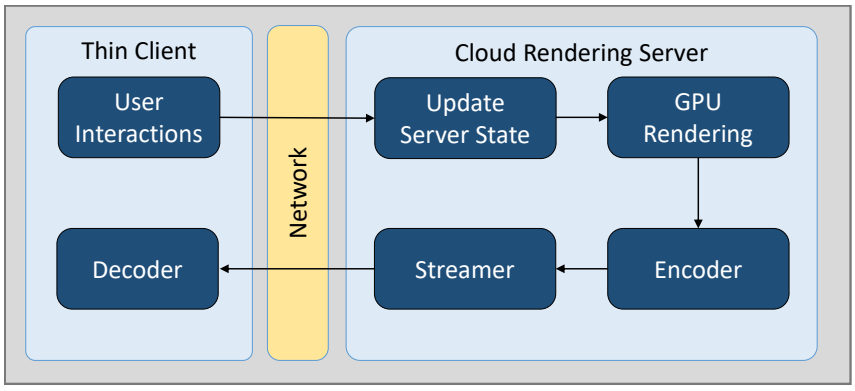


Figure 2.1: Representation of the steps on a regular cloud rendering system.

The strategy of producing graphical content on a remote computer is not new. This idea has its roots in the past century when the costs of the computers were prohibitive, and processing capacity was limited. The solution adopted by large companies was to acquire a few powerful computers, known as mainframes, that could afford processing for dummy terminals.

In 1975, Gordon E. Moore predicted that the processing power would double every two years for at least a decade since smaller transistors could be packed even more tightly, boosting the performance and reducing costs. This prediction is known as Moore's Law (Schaller, 1997). In this scenario, personal computers have become cheaper, and their computing power has hugely increased, whereas the interest in using remote computers for rendering graphical content has been left aside.

Recently, however, the remote rendering area has attracted attention again due to the combination of different factors. In the last years, we have witnessed massive data generation both in volume and detail, whereas the advance of the processing power of the microchips has slowed down. In addition, the usage of portable devices (e.g., smartphones and tablets) has also expanded. These devices have limitations in energy and processing power. This scenario brought back the necessity of using remote services to overcome the limitations in such environments.

2.2 Classification

Depending on the type of data transmitted from the server to the client, the system can be classified into two major categories: model-based or image-based (Shi & Hsu, 2015).

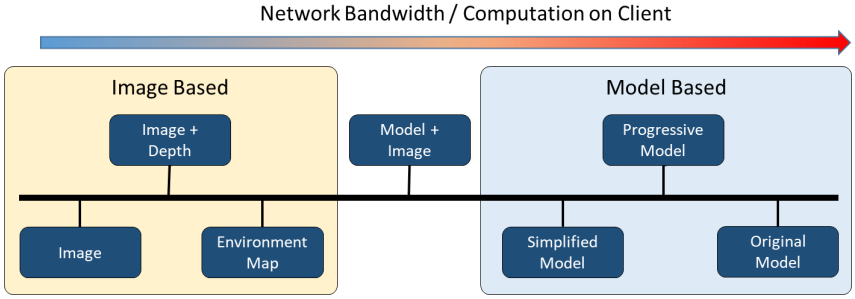


Figure 2.2: Classification of cloud rendering systems by the data type exchanged between client and server.

Model-based systems send geometric information to the client, typically as triangular meshes. The size of transmitted data is proportional to how complex or detailed the underlying representations are. In contrast, image-based systems render the scene entirely on the server side and only display the resulting images on the client. Unlike model-based systems, the size of transmitted data mostly depends on screen resolution, regardless of the scene complexity. This property made image-based systems popular since they are more robust to the variation on the network quality.

Some cloud rendering systems, however, streams both geometries and images to the client, as depicted in Figure 2.2. Compared to the model-based and image-based systems, this approach is usually a happy medium between image quality and system performance. In this approach, the server transmits both the remote images and some parts of the model to the client. The client renders these objects and uses the remote image only to complete the local image. In other similar approaches, the server renders the fine-grained version of the model and transmits it along with a coarse version of the model. When the client needs to produce a new frame, it renders the coarse version locally and uses the remote frame as a texture to reproduce the details of the original model.

Generally, the transmitted geometries are a simplified version of the original model, and the way the image is used varies for each system. For example, some use the image as a texture to encode the details of the original geometry that are missing on its simplified version (Reinert et al., 2016). In others, the client renders the objects assigned to it and uses the remote image only to complete the image produced locally (Bao & Gourlay, 2006).

2.3 Advantages of Cloud Rendering for Industrial-plant models

Cloud services present several widely known advantages like cost savings in acquisition, setup, and maintenance. Nonetheless, the employment of cloud rendering services for CAD models brings forth other relevant advantages, especially for large companies:

- Data Security: CAD models often contain industry secrets and other confidential information. Having these models stored in a central repository on the server, the users do not need to gain direct access to the files, preventing the cases of data misappropriation.
- Team integration: *As built* (Pătrăucean et al., 2015) is an engineering methodology that claims the virtual model should be updated at the same pace as the real facility evolves. As a result, the model is constantly being updated. Working locally, users are very likely to be using different versions of the same model, which may weaken the team integration. On a cloud system, the model only needs to be updated on the central repository, and all users will have access to the same and newest model version.
- Device Expansion: The use of portable devices, such as tablets and smartphones, for visualizing industrial plants is increasingly common.

However, the computing power of these devices is limited, and battery consumption is critical. Offloading rendering tasks to remote computers mitigate these two problems.

– Rendering Improvements: Since a cloud rendering engine runs on a well-known device, the engine can provide state-of-the-art rendering algorithms and optimizations to the users.

2.4 Latency Analysis

The ultimate challenge of any cloud rendering system consists of producing remote frames at the lowest possible *Response Delay* (RD), which is the elapsed time since the user performs some interaction until some response is presented back. The cloud system must ensure low latency rates, preventing the degradation of the Quality of Experience (QoE) (Wang & Dey, 2009). The tolerable delay for first-person shooter games is around 100ms, or approximately ten frames per second (*fps*) (Claypool & Claypool, 2006). The reduction of system latency is achieved by shortening the delay of each step involved in the production of the final image. When considering a conventional remote rendering system, as depicted in Fig. 2.1, the RD is defined as the sum of three components:

$$RD = ND + SD + CD,$$

where:

– **Network Delay** (ND): the time required to exchange data between the client and the server. It is also known as network round-trip time.

– **Server Delay** (SD): the time the server takes since a new event arrives on it until the new frame is produced.

– **Client Delay** (CD): the elapsed time since a new remote frame is available on the client side until this frame is displayed on the screen to the user.

When we break the response delay down into its main components, we observe that the network delay is the most challenging one. Differently from the server delay and client delay, which their results are proportional to the server and client workloads, the network delay highly varies due to several events. Unfortunately, it is very hard to predict these events in advance, resulting in the difficulty in estimating the performance of this component. Consequently, considering the cloud rendering systems which follow the overall steps depicted

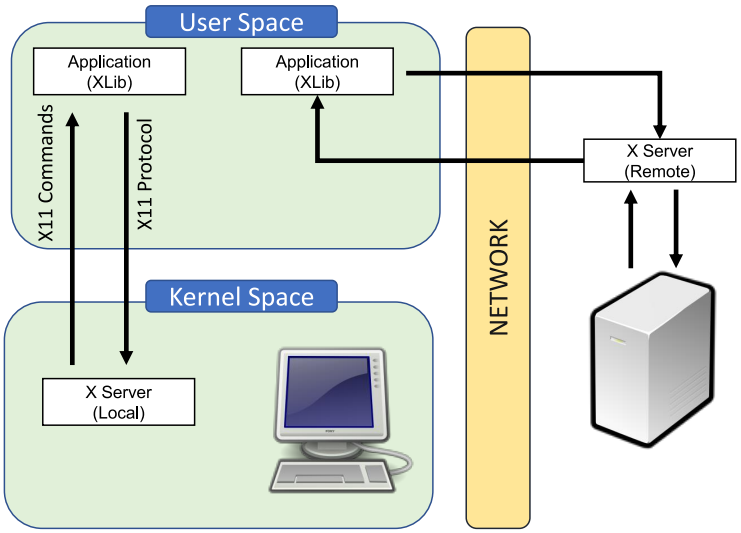


Figure 2.3: X Window server-client model. The X client application can communicate with both a local or remote server.

in Figure 2.1, regardless of how efficient their client and server operations are, their final results highly depend upon the network performance.

2.5 Related Work

Cloud rendering systems have been widely used in different fields, ranging from scientific visualization to the game industry. In this section, we discuss some of the most notable works.

Despite the recent popularization of cloud rendering services, the idea of rendering a graphic content on another machine is not a novelty. In the early computing years, mainframes were responsible for providing computing services to computers with no processing capabilities, known as dummy terminals. One of the services provided by mainframes involved rendering the graphical environment system for these dummy terminals. In 1984, the X Window System (Scheifler & Gettys, 1986) was designed to provide 2D GUI elements using a server-client architecture. The X application, which runs on the client side, only informs the X server about user interactions. The server then interprets these interactions and provides a graphical content to the application. Despite the client/server architecture of the X Window System, it is very common to see applications where both the client and the server are running on the same machine, working like a regular application. Figure 2.3 shows the workflow of an X Window System in which two applications run on the client side, but one is communicating with a remote server, and the other is consuming the local server's services.

Another widely adopted variant for remote rendering is the remote

desktop sharing. One of the most popular examples is the Virtual Network Computing (VNC) protocol. This protocol allows to remotely control another computer by sending input device events and replicating the remote display on the local computer. This way, it is possible to run a complete application on a remote computer, displaying the results on the local machine.

The recent proliferation of high-end cloud services, as well as the massive adoption of the high-bandwidth network, enabled the creation of powerful 3D remote rendering systems. WireGL (Humphreys et al., 2001) is a scalable graphics system that allows users to send graphics commands to graphics servers. Inspired by the WireGL system, the Chromium project (Humphreys et al., 2002) has emerged, providing an interface to control the graphics commands on clusters. This framework leads to a mobile-devices-oriented solution that was used on the server side to manage the rendering commands (Lamberti & Sanna, 2007).

More recently, researches on distributed rendering (Abraham et al., 2004; DeFanti et al., 2011; Renambot et al., 2004) have become popular. The distributed rendering fits well when the rendering computation is complex for a single server or when the system needs to render multiple views, as in the case of CAVE's (Cave Automatic Virtual Environment) (Peternier et al., 2007). The central idea of distributed rendering systems is the division of the final viewport into disjoint spatial regions. The system assigns different computers to each one of these viewport regions. Once all regions are available, the system generates the final image by stitching the tiles.

The Game as a Service (GaaS) (Ahmadi et al., 2015; Al-Rousan et al., 2015; Semsarzadeh et al., 2015) has been pushing the boundaries of cloud rendering in recent years. Different from the traditional video game consoles, with all computation being processed locally, cloud gaming services enable lightweight devices to run the latest games. Despite the similarities to a walkthrough CAD application, the main concern of these services is the game state synchronization. Model massiveness is rarely a concern for these platforms since their models are known previously and the rendering engine can optimize them. In addition, the existing platforms usually comprise a set of inter-connected dependent modules, such as *input*, *rendering* and *game logic*. Thus, if any of these modules is not performing well, the whole system will suffer from slow performance. For these reasons, the existing cloud rendering solutions do not fit well the requirements for rendering massive CAD models.

Remote rendering has also been employed for scientific data (volume rendering) (Tamm & Krüger, 2014). The authors used hybrid rendering to attenuate the network latency. The server keeps several versions of the same

volume data, each one with different level-of-detail (LOD). Depending on the network conditions and the client processing power, the server estimates which version the client should render locally. While the client renders the coarse version of the model, the server renders a more fine-grained version of the model. The goal is to determine a workload schedule that enables synergy between the two sides to provide rendering results to the user as fast as possible.

The hybrid rendering approach has also been used for VR (Virtual Reality) application (Lai et al., 2019). This work splits the VR workload in rendering foreground and background frames. In this works, the background is drawn by the server, while the foreground is rendered on the client. According to the authors, the background frame has a much heavier rendering load due to rich details and complex textures. For such reasons, it is reasonable the pre-rendering the background on the server.

The recent advances in both WebGL and HTML5 motivated some works to study the hybrid rendering using web browsers (Dyken et al., 2012). In that work, both sides render the same geometry, but the client renders a simplified version to reduce network latency. The remote image is transmitted to the client in order to increase the details of the local rendering. Despite the recent advances, web applications still have limited access to computer resources.

The majority of cloud-based solutions rely on virtualized remote computing resources provided by third-party organizations, known as Infrastructure as a Service (IaaS). These services are charged accordingly to the amount and duration of used computing capabilities. In this scenario, some works have emerged aiming at the reduction of the cost involved using these services(Azumah et al., 2018; Lin et al., 2017; Loukopoulos et al., 2018). They tackle this problem by establishing a minimal task set that still produces the desired results but requires minimal computing capabilities as possible. Generally, this problem involves at least two opposite objective functions: minimize the service cost and maximize the system performance, resulting in a multi-objective optimization problem.

2.5.1 Remarks on Existing Works

Our work is closely related to general research on remote rendering of 3D models. However, none of the existing works provide a perfect fit for our requirements. Although some of them share common issues and solutions, they are not concerned about handling a large data volume. The massiveness of CAD models is usually the bottleneck for visualization systems. In order to provide a satisfactory user experience, the model massiveness must be taken

into account, especially for using the model particularities to improve the rendering performance.

At first glance on the existing works, we notice that the communication between the client and the server is tightly coupled, making the client very dependant on the server response. This high dependency is critical for interactive applications. Instead, in our method, we use the client as a complete and independent application that only uses incoming data to improve the user experience.

With regards to use web browsers as the client platform, they are not yet a viable solution. In our tests, although the computer has considerable resources, the browser restricts the use of these resources for security reasons. Since we are dealing with massiveness, we demand full access to the hardware resources.

3
Hybrid Rendering of CAD Models on the Cloud

In this chapter, we present our proposal for rendering industrial-plant CAD models on the cloud. Our goal is to overcome the limitations of the existing cloud rendering works for CAD models, such as handling massive data volume and low response delay. We address the former by establishing a set of nine standard primitives (Table 3.1) in which each one of them can be described by a set of parameters. The key feature of this representation is its compactness, allowing efficient storage, transmission, and rendering. We approach the latter issue by employing a hybrid rendering mechanism, i.e., rendering a small portion of the model on the client and the remaining on the server. Besides, we immediately display a new image to the user as soon as the local rendering is done, instead of waiting for the remote frame. Lastly, the hybrid rendering also allows us to render textual elements on the client side. This way, we ensure their readability since their rendering results are not compressed.

In order to guide our investigation, we first established the following requirements that a hybrid cloud rendering service must meet in order to achieve robustness and efficiency:

1. **Efficient data representation:** network performance is the most critical aspect of cloud services because it is impossible to make early assumptions on the network performance. It is beyond system control. For this reason, we can only ensure that the data exchanged between the two points are as compact as possible. Besides, a compact data representation also reduces the necessity of huge capacity storage devices. In addition, it also important using an efficient representation when rendering massive models since the video memory is usually very constrained on low-end devices.

2. **Efficient rendering:** the primary task of any rendering service is the rendering itself. The massiveness of CAD models turns this task even more important since they demand large graphics memory (VRAM) and high processing power. An efficient rendering engine allows more rendering tasks to be executed on both sides, reducing at the same time

the server's workload and turning the client more independent from the server responses.

3. **Controlled workload:** the workload on the client side must not exceed the client's processing capacity. The remote processing becomes useless if the client is continuously busy processing its tasks and, for this reason, can barely consume the incoming data.

4. **Low response rates:** any interactive application must provide low response rates to guarantee a reasonable user experience. Considering a client/server application, this means it must not rely upon network conditions in order to provide some feedback to the user. Otherwise, the system can suffer from high latency variations. Partial scene rendering results are allowed, as long they convey a spatial awareness to the user.

All these requirements guided our investigation to achieve an efficient cloud rendering system. In the next sections, we discuss the details of our method from the following perspectives: efficient data representation and rendering, communication model, client and server architectures.

3.1 Efficient Data Representation and Rendering

A striking feature of CAD models is the high object redundancy. About 95% of objects from industrial-plant models are a composition of simple objects such as spheres, planes, cones, cylinders and others (Requicha & Voelcker, 1982). Such objects can be defined by their underlying parameters.

Despite the traditional triangle mesh representation being very proper for rendering purposes, it is an inefficient representation in terms of storage and transmission due to its verbosity. For this reason, we establish a set of standard primitives with nine well-known objects that each one of them can be described by the parameters of the underlying shape. All parametric objects and their respective parameters are listed in Table 3.1. This approach plays a central role in satisfying our requirement for efficient data representation.

If we consider a triangle mesh, each triangle is composed of three vertices. Every vertex has two attributes: position and normal, each one represented by three floating-point scalars. A mesh with k non-indexed triangles would require $k \times 72$ bytes[1]. Moreover, at least nine additional floats (36 bytes) may be necessary to instantiate the object in the world: translation, rotation, scale (36 bytes). For the sake of simplicity, we denote these transformations as $\mathbf{M}$:

[1]Considering IEEE 754 single-precision binary floating-point format.

Table 3.1: List of parametric objects.

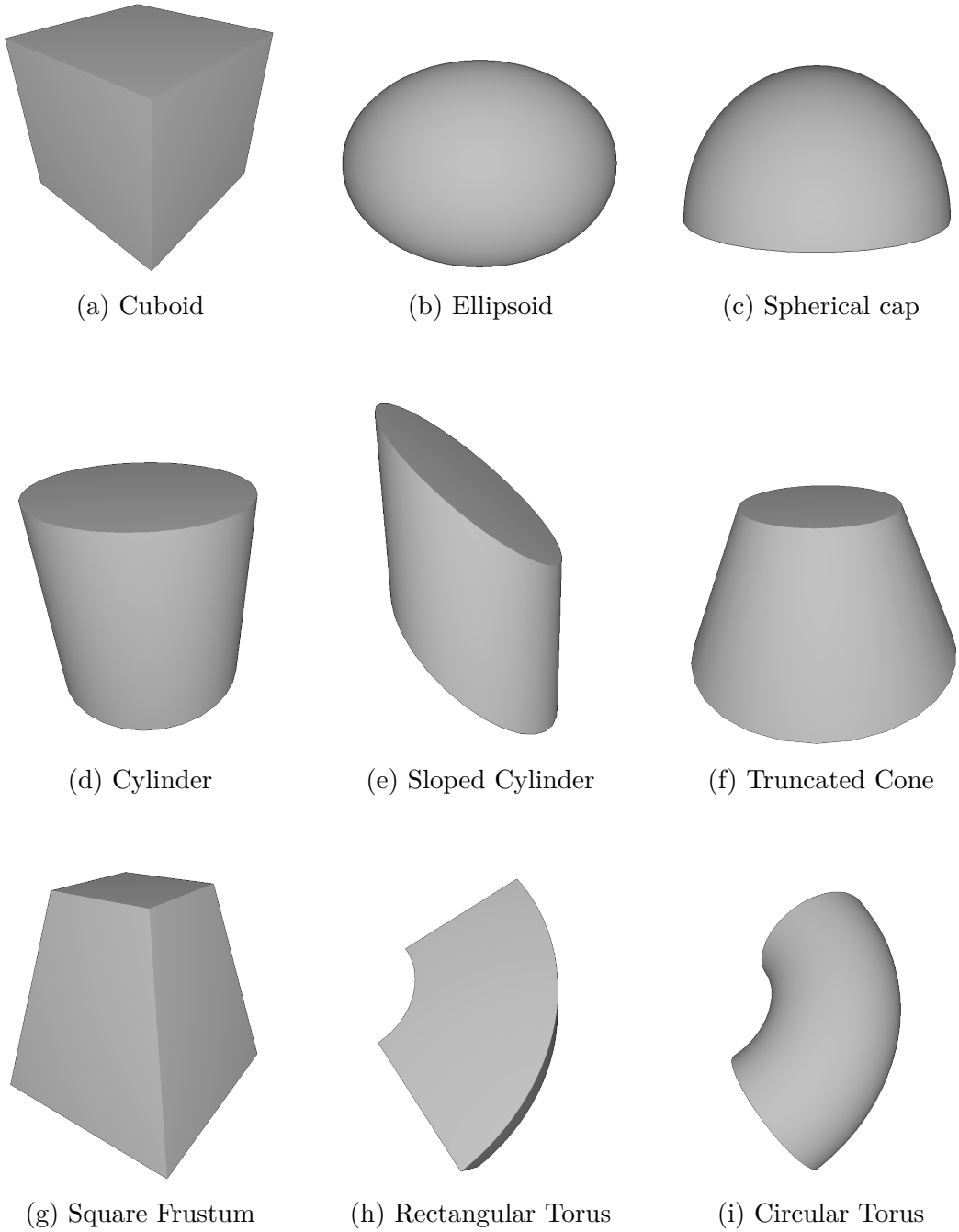
Geometry Type	**Geometry Description**	**Total Size (bytes)**[1]
• Cuboids • Ellipsoids • Spherical Cap	**M**	36
• Cylinders	**M** + 2 offsets + radius + height	52
• Sloped Cylinders	**M** + radius + height + 4 slopes	60
• Truncated Cone	**M** + 2 offsets + 2 radii	52
• Square Frustum	**M** + 2 offsets + 4 side lengths	60
• Rectangular Torus • Circular Torus	**M** + 2 radii + sweep angle	48

a 3x3 matrix containing the values for translating, rotating, and scaling the object in each axis of the coordinate system. Note that this is not the matrix used to represent the affine transformation in the homogeneous space.

In contrast, some of the objects defined by our set of standard primitives can be represented by only the **M** matrix. The information to describe a right circular cylinder is already encoded on the **M** matrix. Both the diameter of the base and its height is already encoded on the scale transformation. The cuboids and spheroids are represented in the same way. For the other class of objects, only a few additional parameters are necessary.

In addition to efficiency in data transmission, the parametric representation also allows thin clients to draw a higher number of objects. The rendering system only needs a single regular grid on the client memory to render all these objects. Therefore, the consumption of VRAM (Video Ram) is constant regardless of the number of objects and their surface resolution. In order to transform the regular grid into the final object shape, each parametric object has a custom vertex shader that properly deforms the grid, as depicted in Figure 3.2.

Finally, we address the high redundancy object of CAD models by taking advantage of hardware-accelerated instanced rendering. This way, the system only issues one draw call for each geometry type. This approach vastly reduces the rendering cost. Previous research has shown that rendering performance can be improved from 6x to 10x by using this approach (Santos & Celes Filho, 2014). Nonetheless, it is important to keep in mind that such a feature may not be available on the client side due to limited graphics capabilities.

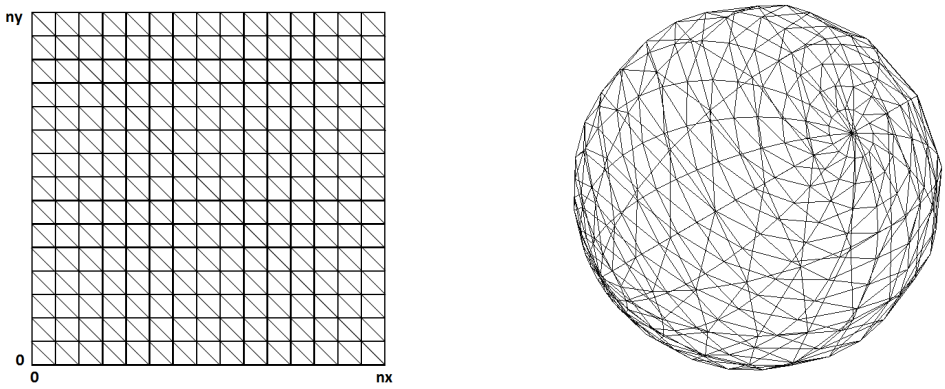


(a) Cuboid (b) Ellipsoid (c) Spherical cap

(d) Cylinder (e) Sloped Cylinder (f) Truncated Cone

(g) Square Frustum (h) Rectangular Torus (i) Circular Torus

Figure 3.1: Categories of objects with parametric representation supported by our rendering system.



(a) (b)

Figure 3.2: Rendering of parametric objects. (a) regular grid used for rendering all objects. (b) resulting sphere from the deformation of the regular grid.

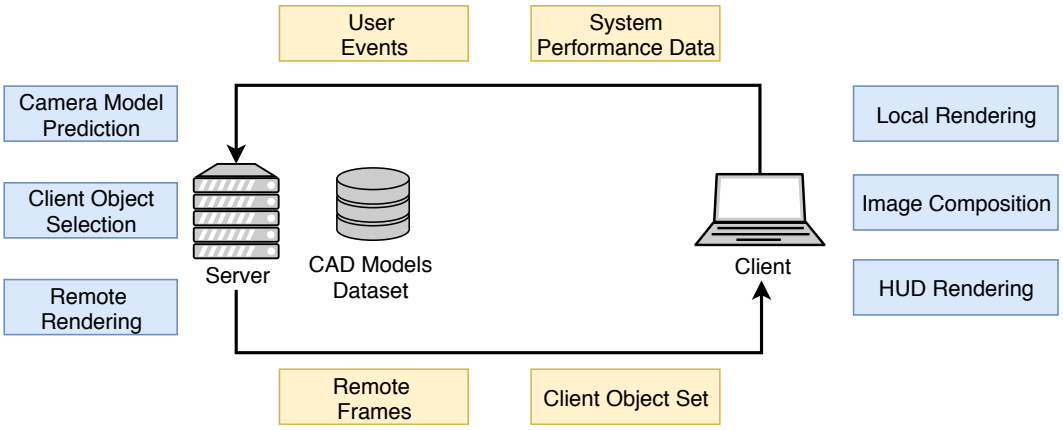


Figure 3.3: Representation of our cloud rendering system architecture. The yellow boxes are the data type exchanged between client and server, while the blue boxes are the tasks each side executes.

3.2 Client-Server Communication Model

In this section, we present our communication model employed to exchange information between the client and the server.

In cloud services, the definition of a communication model is a crucial task once the network performance is beyond developers' control. In this section, we describe the communication flow between the server and the client in a session. The communication model involves two machines playing two roles: client and server. The server provides rendering capabilities through an internet connection or an enterprise network. All CAD models are stored in a repository on the server side. A schematic overview of our system architecture and the data types exchanged between client and server are presented in Figure 3.3.

Communication starts with the client requesting a model to the server. Along with this request, the client also informs about its computing capabilities

(CPU, memory, and graphics card). In the first time the client connects to the server, the server uses this information to determine the proper workload that the client can handle without the loss of efficiency. The server has a predefined list of the most common graphics cards and the ideal workload for each one. The next time this client connects to the same server, its historical performance data may also be taken into account to increase or decrease the client workload.

The server then assigns objects for the client to render. It chooses only parametric objects, due to their compactness, requiring less bandwidth for transmission and memory for rendering. The client object set is subdivided into two disjoint sets: static and dynamic sets. The static object set is defined during the connection establishment phase (Chapter 4), and it remains the same during the session. On the other hand, the server is continuously sending new objects to the client's dynamic object set (Section 3.3). The ratio between the number of objects in each client's object set is defined for each model by the repository maintainer.

During the model transmission, the server also sends the model's metadata, such as creation date, author, and engineering data. Some of them are displayed as HUD (Head-up display) elements (i.e., 2D elements drawn over the final image). All textual elements are always rendered on the client side to avoid harming their readability due to compression. As soon as the first objects arrive on the client side, the client starts rendering, and the first frames are presented to the user.

At first glance, we notice that the existing cloud rendering architectures are composed of several sequential and highly dependent steps, as shown in Figure 2.1. If any of these steps is not performing well, the entire operation of the system will be compromised. Consequently, the user experience is very sensitive to network conditions. We propose to decouple client and server operations using an asynchronous event-based communication model, as depicted in Figure 3.4. The client is always running its task cycle, regardless of whether the remote data is available or not. Therefore, the client application is always responding to user input events, enabling the user to always navigate through the scene.

The client application executes its task loop whenever the user interacts with the application or a remote data arrives, such as new objects or remote frames. In the first case, the client starts rendering the local scene at the same time it notifies the server about the event. The client also sends the world position in which the user event took place. The server uses this information to adjust its camera position, as discussed in Section 3.3.

After rendering the local image, the client completes it using the last

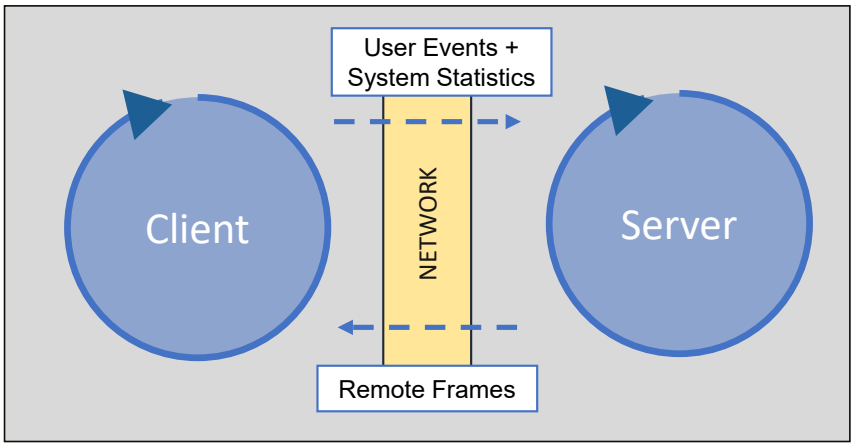


Figure 3.4: Schematic representation of the event-based communication model. The dashed lines represents the weak dependency due to the asynchronous communication between the client and server

available remote image and then displays the result to the user. Due to the asynchronous communication model, we do not halt the client waiting for the remote frame associated with the new local viewpoint. When this remote frame becomes available, the client enters again in the running state and uses it to improve the quality of the displayed image.

From the server perspective, it continuously produces new frames while the user is navigating on the scene. These images are encoded and transmitted to the client. Along with these frames, the server also sends the camera configuration used in their production. This way, the client can use them to combine with its local image. The process of establishing a connection until the first frame is displayed to the user is depicted in Figure 3.5.

Lastly, the client is frequently sending performance statistics to the server, like network latency and local rendering performance. The server uses such information to improve the future workload division for this particular client, and also to predict future frames, as discussed in the next section.

3.3 Server Architecture

For each incoming connection, the server launches a new rendering engine that runs in a separate thread. A shared bus is responsible for the communication between the clients and their associated engine instance.

The server is continuously rendering new frames while its queue of user events is not empty. For each camera viewpoint, the server produces two depth-augmented images: primary and depth-peeled images. The primary image is a regular frame, whereas the second one is produced using the depth peeling technique (Liu et al., 2009). The peeled frame is important to mitigate missing

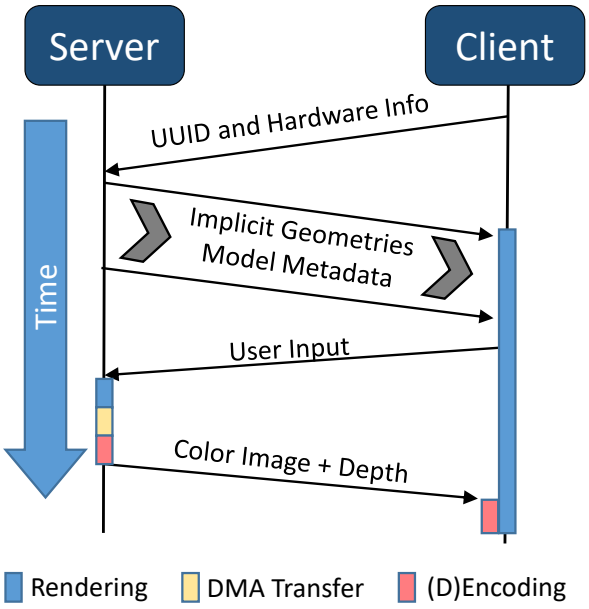


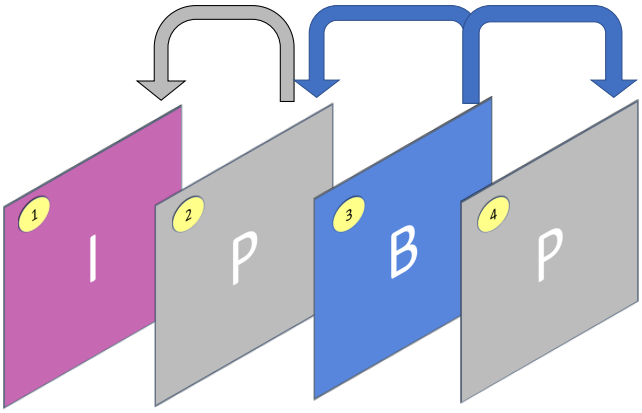
Figure 3.5: Interaction between client and server during connection establishment phase. The first frames are displayed after the first geometries arrive on the client side.

pixels on the client's final image due to disocclusion.

After rendering the primary frame, the server starts a Direct Memory Access (DMA) to download both color and depth frames from VRAM. In the meantime, it starts the rendering of the peeled frame. This approach avoids wasting time synchronizing CPU and GPU. Lastly, the server encodes the depth-augmented images and transmits them to the client along with the camera settings used in their rendering.

In our implementation, we use the H.264 codec (Luthra et al., 2003) to encode the color image, and we use the LZ4 (Bartıik et al., 2015), a lossless real-time compression algorithm, to encode the depth information. The H.264 codec supports three different types of frames: *I-frame* (Intra-coded picture), *B-frame* (Bidirectional predicted picture), *P-frame* (Predicted picture). An *I-frame* is a complete image like any other regular image file. The *P-frame* holds only the changes in the image from the previous frame. This way, to decode a *P-frame* we only need its previous image. Lastly, the *B-frame* encodes the difference between the current image and both the preceding and the following frame.

Despite the *B-frames* being more compact than the others, its use increases the system latency because they need both one preceding and one succeeding frames for decoding them. This way, the client would need buffering frames until the required succeeding frame becomes available. For this reason, we only use *I-frames* and *P-frames* during the encoding. The first is a self-contained frame, and the latter only needs one preceding frame to be decoded, as depicted in Figure 3.6.



Arrival Order	1	2	3	4
Decoding Order	1	2	4	3

Figure 3.6: The three major image types used in the video encoding: *p*, *b* and *i*. The *b-frame* is the one which introduces more latency because it needs one preceding and one succeeding frames.

Due to our asynchronous event-based communication model, the server must anticipate the rendering of future frames. Otherwise, the remote frames would likely be very outdated once the client does not halt their operation, waiting for the server response. The client always uses the last available remote frame to complete its local rendering. The more different the client and server viewpoints are, the more artifacts, such as holes, will appear in the final image. To overcome this issue, the server uses a camera prediction model to estimate future positions of the client camera. This whole prediction model involves two mechanisms: *lag compensation* and *path correction*.

The *lag compensation* estimates the time displacement of the server camera to compensate for the extra overhead in the remote frames production. This overhead involves the encoding/decoding of the color and depth images, the network transmission, and other additional tasks, as explained in Section 3.5.

The camera movement of both sides follows a uniform linear motion: $S \leftarrow S_0 + V\Delta t$, where S and S_0 are, respectively, the final and initial camera positions, V is the velocity and Δt is the time period considered for the next image rendering. The client's time period, Δt_c, is the elapsed time of the last local rendering. The time period on the server camera is $\Delta t_s \leftarrow \Delta t_c + k$, where the k is a constant that grasp the extra overhead of the remote frames. The k value is approximately the response delay of the remote frames (Section 3.5).

Since the computation of the server camera displacement involves several noisy values, the server smooths them using the exponential moving average (EMA) function. Differently from the simple moving average (SMA) function, the EMA function gives more weighting, or importance, to recent data. This

Figure 3.7: Client rendering performance and the resulting smoothing curves using a 10-period simple moving average (SMA) and a 10-period exponential moving average (EMA).

behavior is important in historical data, once recent events are more likely to happen again. Figure 3.7 depicts the behavior of both EMA and SMA on the client rendering performance.

Since the server camera is displaced, whenever the user changes the client camera direction, the server camera is in an invalid path. In this case, the *path correction* fixes the server camera by moving it back to the place where the event took place. Next, the server camera is displaced again toward the new direction.

Figure 3.8 depicts both *lag compensation* and *path correction*: while the client camera is rendering the image from t_0 time point, the server camera is displaced in time, rendering the next predicted position. If in this moment the user changes the camera orientation, the client notifies the server about this event. Nonetheless, the server camera is only notified at t_1 time point due to the network latency. At this moment, the *path correction* moves the server camera back again to the position in which the event took place, and then the camera is displaced again, but now towards the new direction.

Lastly, the server is constantly updating the objects on the client's dynamic object set as the user navigates through the world. The goal of this set is to ensure that the closest objects are drawn on the client side. This is important to improve the image quality since near objects are likely to be visible on the final image. In addition, in case of disconnections, the near objects still be visible. This result is crucial for some engineering activities, such as equipment inspections.

Also, when the user rotates the camera, the last remote image available is often useless because almost no pixels can be reused on the final image once

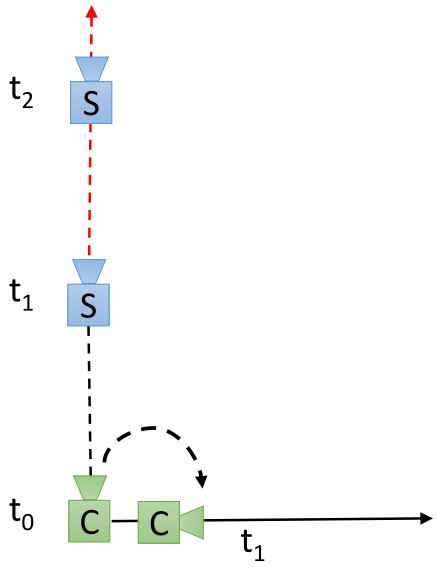


Figure 3.8: Representation of the lag compensation and path correction.

the user is looking in another direction. The dynamic object set also mitigates this issue since the closest objects are surrounding the camera.

The server selects the objects to the dynamic set using the *k-nearest neighbor* algorithm (Bhatia et al., 2010), where the k is the size of the dynamic object set that was defined at the connection establishment. Whenever the server sends a new object set to the dynamic set, it also computes the bounding box of this set. In each rendering loop, the server uses the predicted camera to check if the client leaves the bounding box of the last dynamic object set. If so, the server selects a new set and transmit again to the client.

3.4 Client Architecture

The client architecture comprises two threads: rendering and resource threads. The first one is primarily responsible for rendering the local image and combining it with remote frames. The latter manages the incoming and outgoing data over the network. Besides, it receives remote frames, decompresses, and uploads them to the client VRAM as textures. When the remote frame is ready for rendering, the resource thread notifies the rendering thread.

The client renders its local scene into a texture using offscreen rendering (Oat, 2008). This way, if the user stops navigating and a delayed remote frame arrives, we do not need to render the local scene again to update the final result. We only combine the last result of the local rendering with the new remote frame. To produce the final image, the client combines three images according to their depth: one local and two remotes (the primary and peeled

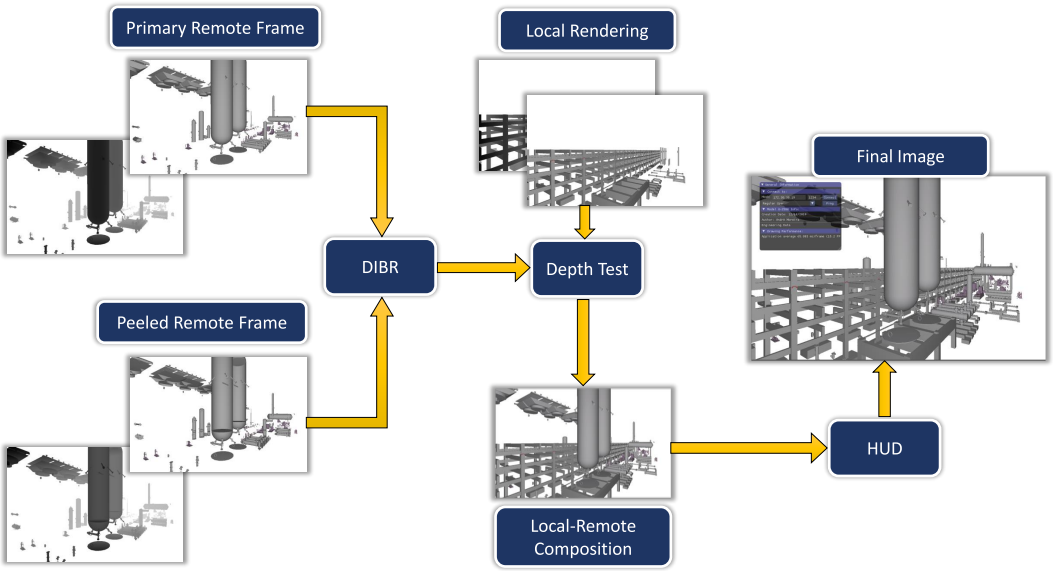


Figure 3.9: Schematic representation of client's operation to produce the final frame.

images (Liu et al., 2009)). Then, the client draws the HUD (Heads-up display) elements over this image (Figure 3.9).

Since the server camera is displaced in time, the viewpoints of the remote and local images are unlikely to match. Thus, the client has to adjust the viewpoint of the remote image to match its local camera viewpoint. The client uses the depth-image based rendering (DIBR) (Fehn, 2004) on the remote image to perform the viewpoint adjustment. The DIBR is a procedure for multi-view image generation. It uses a depth-augmented image as a reference to produce novel images from different viewpoints. To synthesize novel images, we first warp the points from the original image plane to the 3D world coordinates and then back-project these points onto the virtual image plane, which is located at the required viewing position. Figure 3.10 depicts this operation.

DIBR operation often produces holes on the image surface. They come mainly from disocclusion and undersampling. We mitigate the first issue by using the peeled remote frame. If the user is in a region with a high density of objects, when translating the camera, the client will be able to rapidly reconstruct the structures that are now visible.

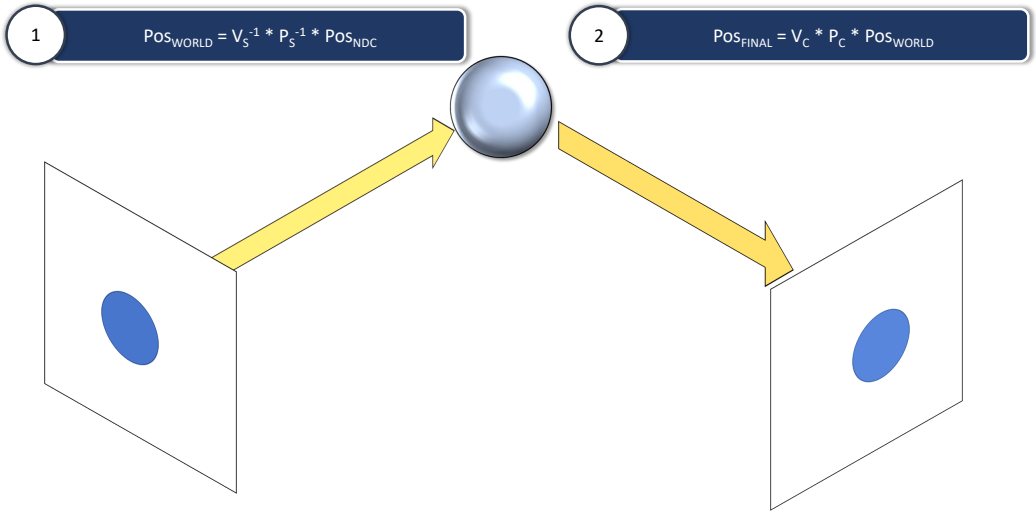


Figure 3.10: Depth-image based rendering operation. The V and P are the view and projection matrices, respectively. The subscript s and c are related to the server and client transformations.

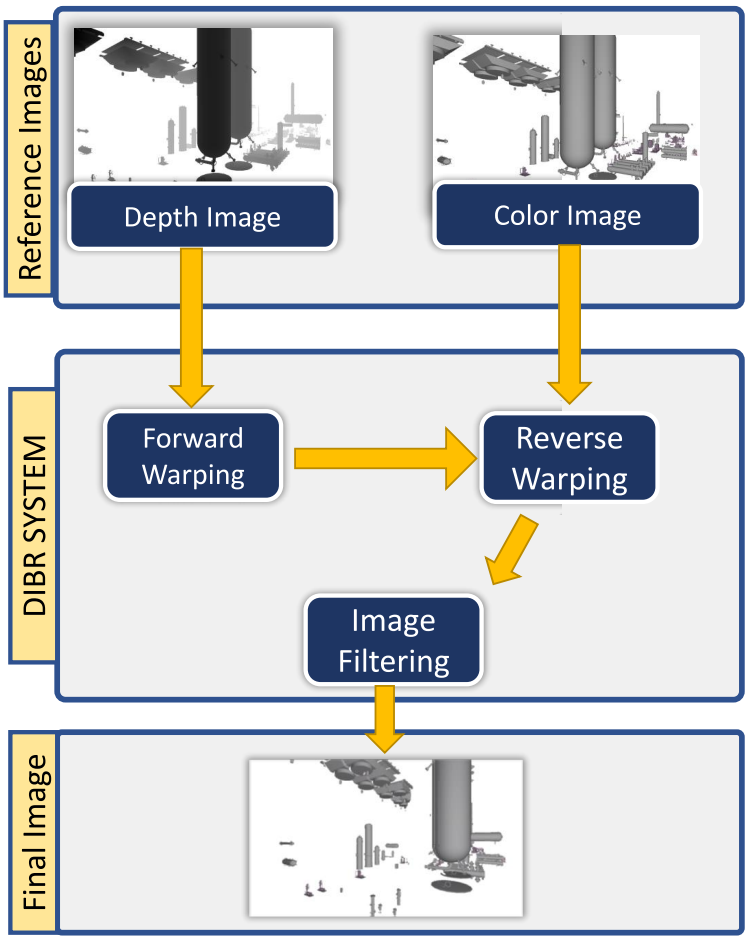
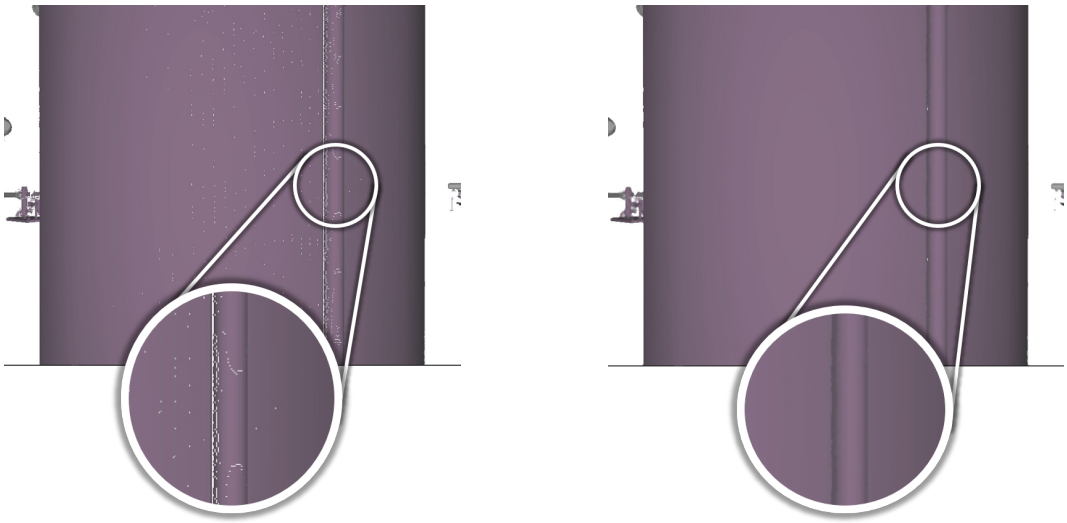


Figure 3.11: Schematic representation of forward-backward depth-image based rendering.



(a) (b)

Figure 3.12: (a) Cracks on remote image surface due to image magnification. (b) Hole-filling using the median filter.

Holes due to undersampling occur when the user suddenly quickly approaches remote objects. In this case, the client has a larger region to cover with remote pixels. While the new remote frame is unavailable, the client will have to use the last remote frame it has, which does not have enough pixels to cover the new region. As a result, some cracks appear on the surface (Figure 3.12). We investigated three different approaches to reduce these cracks: pixel splat (Bao & Gourlay, 2006), dilation of the depth image (Xu et al., 2013), and using the median filter on the color image (McGuire, 2008). In our tests, the last approach provided the best results.

We propose to use forward-backward DIBR, instead of the regular DIBR (Figure 3.11). At the first moment, in the forward step, we only apply the regular DIBR operation on the depth source image. Then, in the backward phase, we retrieve the pixel color by filtering the pixel's source neighborhood using the median filter. The image filtering can successfully fill small cracks at the small cost of slightly reducing the image definition.

Even though these artifacts can harm the user experience, they only appear on temporary frames. After stop navigating, the last remote frames will arrive, and these frames are exactly from the same viewpoint as the local image. Therefore, the client skips the DIBR operation, and the final result is a high-quality image.

Table 3.2: Descriptions of the notations used for latency analysis

Symbol	Description
v_i	Indicates the viewpoint i.
Img^{v_i}	Image from viewpoint v_i.
$rdr(v_i)$	Rendering from viewpoint v_i.
$enc(Img^{v_i})$	Encoding image Img^{v_i}.
$dec(Img^{v_i})$	Decoding image Img^{v_i}.
$W^{v_i \rightarrow v_i'}$	Image Warping from the viewpoint i to the viewpoint i'.
cmb	Combination of remote and local images to produce the final image.
c, s	When subscribed or overwritten it relates to the client and server, respectively.
$down$	Transference from VRAM to RAM.
upl	Transference from RAM to VRAM.

3.5 Latency Analysis on Hybrid Cloud Rendering

As discussed in Section 2.4, the response delay is the elapsed time from a user event occur until the system provides a response to this event. At first glance, we notice that the response delay on the existing remote rendering works is very dependant on the network performance. In our hybrid rendering approach, since the system produces two responses (local and remote frames) for each user event, we break our response delay into two components: the local response delay (LRD) and the remote response delay (RRD). The former is the time to the local frame be available to the user, whereas the second is related to the remote frame. The LRD ensures the system responsiveness, even when the server responses are delayed or unavailable. The RRD affects the quality of the final image since it measures how fast the client has an updated version of the remote image.

Since we decouple the client operation from the server responses, the LRD is similar to a regular on-premise rendering service, i.e., a rendering engine running solely locally. In both cases, the response to the user only depends on the client processing time (CD). Considering the notation defined in Table 3.2, the LRD can be described as:

$$LRD = CD$$
$$LRD = rdr_c(v_c) + W^{v_s \rightarrow v_c} + cmb(Img_c^{v_c}, Img_s^{v_c})$$

The local response delay only depends on the local rendering ($rdr_c(v_c)$) and the composition with the remote image ($cmb(R_c^{v_c}, R_s^{v_c})$). If the viewpoints

from local and remote images are different ($v_s \neq v_c$), the client also performs a DIBR on the remote image to adjust the remote viewpoint to the local viewpoint($W^{v \to v^+}$):

$$Img^{v_c} = W^{v \to v^+}(Img^{v_s})$$

When it comes to the remote response delay (RRD), it comprises the operations from the production of the remote frame until this frame be available on the client side. Generally speaking, it depends on the remote rendering, the encoding on the server side, the network delay for data transmission, and the decoding of the remote frame in the client side:

$$RRD = rdr_s(v_s) + enc(Img_s^{v_s}) + dec(Img_s^{v_s}) + ND$$

These operations can be analyzed as follows: first, the server starts rendering its scene and produces the remote image.

$$Img_s^{v_s} = rdr(v_s)$$

Next, the color and depth images are transferred from GPU to CPU.

$$< Img^{v_s}, D^{v_s} >= down(Img^{v_s})$$

Next, the server encodes the resulting images:

$$Img_{compressed}^{v_s} = enc(< I^{v_s}, D^{v_s} >),$$

When the encoded frame arrives on the client side, the client decodes it:

$$< Img^{v_s}, D^{v_s} >= dec(Img_{compressed}^{v_s})$$

Finally, the client uploads the remote color and depth images to VRAM. Only after this operation, the remote image is ready to be used by the client:

$$Img^{v_s} = upl(< I^{v_s}, D^{v_s} >)$$

4
Client Workload Selection

A key feature in hybrid cloud rendering systems is their high resilience to network conditions. If the communication between the client and server has any issue, the client application can still display partial results to the user. This resilience is only possible due to the local rendering combined with the image-based rendering of the remote image.

For this reason, the workload division plays an important role in hybrid cloud rendering systems. In order to grant a good user experience regardless of the network conditions, we must ensure that the local frame is sufficient to allow the user still navigate through the scene, even when the connection is lost. To achieve this goal, we must ensure that the local rendering can grant spatial awareness to the user. We approach this establishing two criteria for the selection of the client object set: the objects must be as large as possible, and their spatial distribution must be uniform.

Industrial-plant CAD models contain several small objects, such as screws, lamps, spools, nuts, bolts, and others. These objects are almost visually imperceptible when compared to the overall model dimensions. Therefore, selecting the largest scene objects prevents wasting the limited client resource rendering objects that are very likely to contribute to only a few or none pixels on the final image. Besides, this selection approach also improves the final image quality because the local pixels have higher quality than the remote ones. Both the image compression and DIBR rendering reduces the quality of remote pixels.

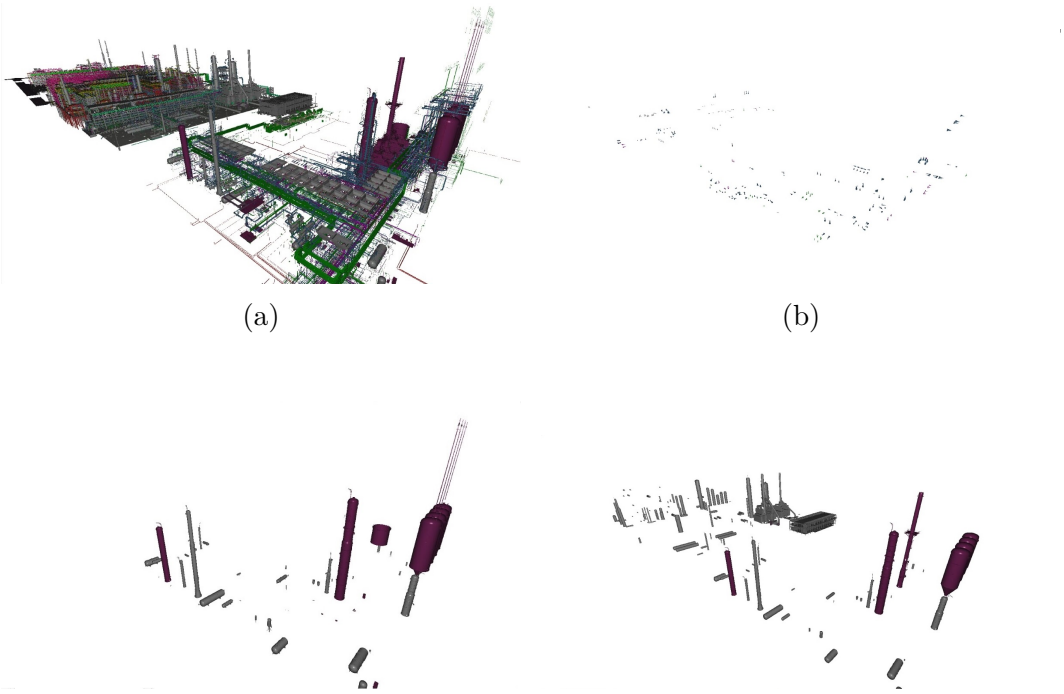


Figure 4.1: Examples of different client workload selection. (a) Complete CAD model. (b) Selection of 49'520 small objects such as valves and screws. (c) Selection of larger objects, but clustered. (d) Desired result, i.e., large objects with uniform spatial distribution.

On the other hand, only selecting the largest objects to the client can also lead to spatial clustering. When the local objects are clustered, there will be a large number of regions that will not be represented in the local frame. Imposing a uniform spatial distribution on the client object set ensures that every region will be covered by some object. As a result, the system grants spatial awareness to the client regardless of the remote image being available. This way, in the cases where the remote image is delayed, the user can still navigate towards the desired region. After some moment, the remote image will be available, and the client can improve the current image.

In short, our workload selection process poses a multi-objective optimization problem: we want to determine the best object set that has the largest objects at the same time these objects are evenly distributed on the scene. The stated problem is an example of a combinatorial problem. Nonetheless, this class of problems is NP-hard, meaning that there is no known polynomial-time algorithm to solve them. On the other hand, we do not need the optimal solution to achieve our goal. We only need a set of objects in which its elements can provide an experience close to the optimal set. For this reason, we use a metaheuristic-based algorithm, Simulated Annealing (Kirkpatrick et al., 1983), to determine which objects to render on the client side. In the next

sections, we discuss how to assess the two discussed criteria. Lastly, we explain the Simulated Annealing and how we use this metaheuristic to solve the stated optimization problem.

4.1 Assessing Spatial Uniformity

There is an increasing number of research topics that need to evaluate how individuals, particles, or components are distributed in a region. In this section, we present a brief overview of these methods, and then we describe how we evaluate the spatial distribution of our client object set.

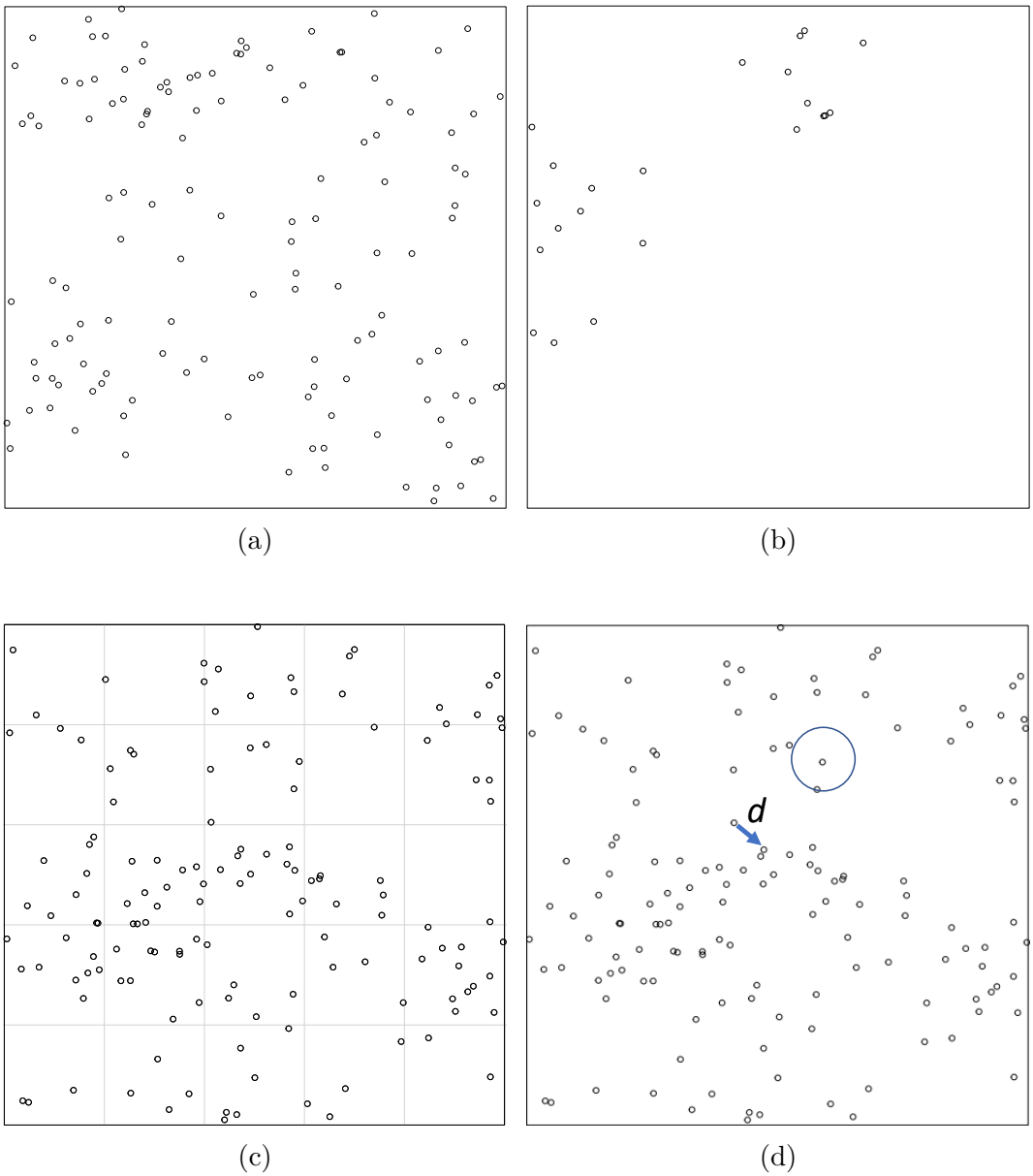
Theoretical Foundations

The *Complete Spatial Randomness* (CSR) (Assuncao, 1994) is the basis for spatial uniformity assessment of point patterns. In the spatial analysis, a common first approach is to compare the investigated data with the complete spatial randomness, which follows a homogeneous Poisson distribution. Several methods have been developed to evaluate uniformity. In the high level, these methods belong to one of the following types: quadrat-based or distance-based (Kam et al., 2013), both methods are depicted in Figure 4.2.

In quadrat-based methods, the region under investigation is partitioned into small-sized regular cells, named quadrats. For each quadrat, there is an associated counter which indicates the number of points contained in this subregion. The CSR hypothesis asserts that the cell-count distribution for each cell must be the same. These methods are very popular due to their simplicity and ease of implementation. In our work, we use the *Index of Dispersion* (Perry & Mead, 1979), a quadrat-based method, to evaluate the degree of uniformity of the client object set.

Nonetheless, there are some drawbacks to quadrat methods. The first is the loss of spatial information from the point pattern since these methods only use the counts. In addition, the process of partitioning the region under investigation has much influence on the final result. In the majority of the cases, there is no natural choice of the partition size, and defining its ideal number is challenging.

As an alternative to quadrat-based methods, other methods assess the degree of uniformity using the observation of the distance between the points and their nearest neighbors. As the name suggests, the nearest neighbor indicates the closest point for a given point, considering all the points in the pattern. Other methods do the same observations, but instead of calculating



(a) (b)

(c) (d)

Figure 4.2: Analysis of spatial distribution of point pattern. (a) Point pattern following the Poisson distribution. (b) Point pattern with clustering. (c) Evaluation of spatial distribution using quadrat-based approach. (d) Two examples of distance-based point pattern analysis: nearest neighbor and fixed-radius near neighbors.

the nearest neighbor from all points that belong to the observation set, they pick random points contained in the investigated region. The average of the distances between the points can be used as a metric that indicates clustering or dispersion.

4.1.1 Index of Dispersion

The index of dispersion (Perry & Mead, 1979) is one of the simplest and most popular indicator for assessing spatial uniformity. It is a quadrat-based method, so it requires the division of the space into regular cells. Since CAD model scenes are naturally static and well-known, we can properly determine the appropriate division resolution for each model.

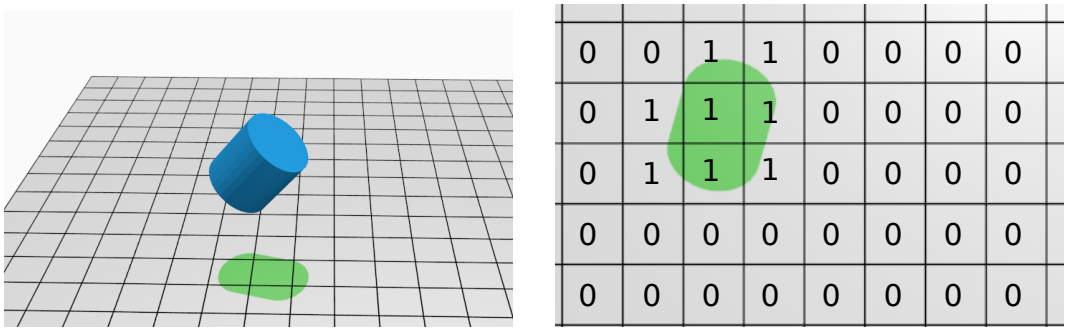
The dimensions of an industrial-plant model along the ground plane are incomparably larger when compared to its height. Therefore, enforcing a uniform distribution considering only this ground plane (i.e., the *XY plane*) is enough to grant spatial awareness to the user. We assess the spatial distribution of the candidate object set as follows:

1. Divide the *XY plane* into q regular quadrats,
2. Associate a counter to each quadrat,
3. Project each object on the *XY plane* (Fig 4.3).
4. Increment the quadrat counter for each object projection that touches its region (green shadow in Fig. 4.3b). The final counter value indicates the number of objects that touch the quadrat.

   At the end of this process, we can evaluate the index of dispersion:

$$ID = (q-1)s^2/\bar{x}$$

where q denotes the number of quadrats, and $\bar{x}$ and s represents the mean and standard deviation of counts. High values indicate non-uniformity or clustering.



(a) (b)

Figure 4.3: Evaluation of spatial distribution using quadrat-based methods. (a) Projection of a cylinder onto *XY plane*. The green shadow represents the resulting projection. (a) Resulting grid of counters after projecting the cylinder.

4.2 Multi-objective Optimization

The general multi-objective optimization problem can be described as:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \boldsymbol{F}(\boldsymbol{x}) = [F_1(\boldsymbol{x}), F_2(\boldsymbol{x}), \ldots, F_k(\boldsymbol{x})] \\ \text{subject to} \quad & g_j(\boldsymbol{x}) \leq 0, \ j = 1, 2, \ldots, m, \\ & h_l(\boldsymbol{x}) = 0, \ l = 1, 2, \ldots, e, \end{aligned}$$

where k is the number of objective functions, m is the number of inequality constraints, and e is the number of equality constraints. $\boldsymbol{x}$ is a vector of *decision variables*, and $\boldsymbol{F}(\boldsymbol{x})$ is a vector of objective functions.

Typically, for a nontrivial multi-objective optimization problem, there is no single global solution that optimizes all objective functions at the same time. This occurs when the objective functions are conflicting. In this case, the improvement in one objective function leads to the worsening of the other. Because of this, there exist several possible Pareto optimal solutions. A Pareto (Marler & Arora, 2004) optimal outcome is one such none of the objective functions can be improved in value without degrading some of the other objective values. All Pareto optimal points lie on the boundary of the feasible criterion space, named Pareto frontier.

The scalarization method is often used for solving multi-objective problems. It is used for obtaining a single-objective optimization problem such that its optimal solutions are also Pareto optimal to the original problem. One of the most common general scalarization methods for multi-objective optimization is the global criterion method in which all objective functions are combined to form a single function. With different parameters for the scalarization, differ-

ent Pareto optimal solutions are produced. The linear scalarization is defined as:

$$\min_{x \in X} \sum_{i=1}^{k} w_i F_i(x),$$

where the weights of the objectives $w_i > 0$ are the parameters of the scalarization.

In our method we use linear scalarization to transform our two objective functions, the volume and spatial distribution, into a single-objective function:

$$\underset{x \in X}{\text{minimize}}\, f'(x) = w' ID(x) - w'' Vol(x)$$

where $ID(x)$ is the index of dispersion, $Vol(x)$ is the volume of the set, $w' > 0$ and $w'' > 0$ are the weights of the scalarization. The volume of the set is obtained by summing the volumes of the oriented bounding box of objects, so we have an approximation that is fast to compute and provides reasonable results.

4.3 Simulated Annealing

Annealing is a technique used in metallurgy, which consists of two steps: heating and cooling. First, the temperature is increased, resulting in the melting of the material. Then the temperature is decreased slowly, cooling the material until it solidifies again. If the material is cooled very quickly, then several separate crystal structures appear, producing an arrangement that is much more disordered.

Simulated Annealing (Kirkpatrick et al., 1983) is a probabilistic method that exploits the annealing process to obtain low-cost solutions for combinatorial optimization problems. It may be modeled as a random walk on a search graph, whose vertices are all possible states, and whose edges are the candidate moves. The ultimate goal is the energy reduction by moving from an arbitrary initial state to another with lower energy.

At each iteration, the system probabilistically decides between stay at same state or moving to neighboring state. The probabilistic approach leads the system to move to states of lower energy. This step is repeated until the system reaches a state which provides the desired result, or until a given stop criterion has been achieved. The system can reach the global optimum if it runs for a long enough amount of time.

Metaheuristics that explore the solution space by seeking better neighboring states can be stuck at local optimum. To prevent this condition, movements to worse states are also allowed, but with less probability.

The Simulated Annealing starts in high temperature, meaning that the system will more easily accept any candidate solutions, even those that are worse than the current one. As time goes by and the temperature decreases, the system becomes more selective. The speed of temperature decay determines how fast the system will provide the solution. Conversely, it determines how close the final solution will be to the optimal one.

The probability $P(e, e', T)$ of moving from an state s to a neighboring state s^* depends on their energy function, $e = E(s)$ and $e' = E(s^*)$, respectively, and time-varying parameter T (the temperature). When T approaches zero, the probability $P(e, e', T)$ must tend to zero if $e' > e$, and to a positive value otherwise. Therefore, for small temperature values, the uphill movements (i.e. replace current solution to a worse one) become less frequent. The acceptance probability from the Simulated Annealing original work (Kirkpatrick et al., 1983) decreases exponentially with how bad this move is, which is the amount energy by which the solution is increased:

$$P(e, e', T) = \begin{cases} 1 & \Delta E < 0 \\ e^{\frac{-\Delta E}{KT}} & \Delta E > 0 \end{cases}$$

where ΔE is the difference between the two states, i.e., $\Delta E = e' - e$ and k is the Boltzmann, a constant that relates the system energy with the temperature values.

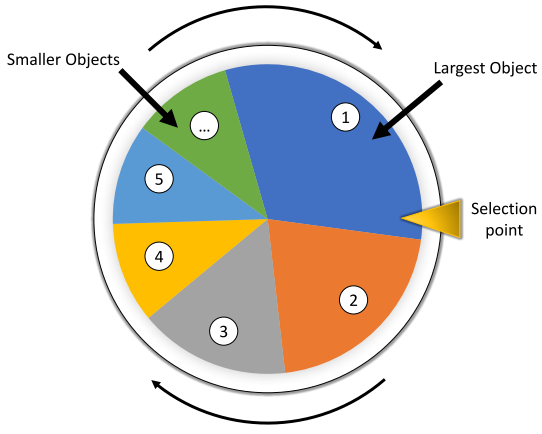


Figure 4.4: Representation of roulette-wheel selection. The fittest individual has the largest share of the roulette. Conversely, the weakest one has the smallest portion. In our work, the object fitness corresponds to the underlying object volume.

The parameters which control the simulated annealing process are the initial temperature, the temperature decay rate (cooling step), and the Boltzmann constant.

We present the pseudocode of our simulated annealing process in Alg. 1. Our system state is the current candidate object set, and the system energy is the result of our fitness function to the current candidate object set. We set the initial state as the n largest objects of the scene, where the n is the number of objects to transmit to the client. This set is a reasonable guess since the largest objects are likely to cover more quadrats when evaluating the index of dispersion. At each iteration, we move to a neighbor state by replacing 5% of the objects from the current candidate set by objects that are not in the current state. We select the neighbor objects using roulette-wheel selection (Lipowski & Lipowska, 2012). The area of each object in the wheel is proportional to its volume (Figure 4.4). This way, we have more chances to select objects that would be in the optimal set.

Algorithm 1 Simulated annealing algorithm

Input:
T: Temperature
K: Boltzmann's constant
α: Reduction Factor
Output:
best: Best Set

1: $best \leftarrow$ **INIT**()
2: **while** Termination criterion is not satisfied **do**
3: $next \leftarrow$ **NEIGHBOUR**($best$)
4: $\Delta E \leftarrow f(next) - f(best)$
5: **if** $\Delta E < 0$ **then**
6: $best \leftarrow next$
7: **else if** $e^{-\Delta E/KT} < random(0,1)$ **then** ▷ uphill movement
8: $best \leftarrow next$
9: $T \leftarrow \alpha \times T$
return *best*

5
Results

In this chapter, we evaluate our proposal in terms of rendering performance, workload selection, and image quality. We developed a prototype using C++17 as the main programming language and OpenGL 3.1+ for computer graphics. We employed the Protocol Buffers library (Google Inc., 2001) for data serialization and Boost Asio (Boost, 2015) for data transmission. We ran the experiments using two computers whose hardware specifications are described in Table 5.1. The image resolutions was set to 720p.

We used two different network setups in our tests. In the first arrangement, we connected both computers on a private local area network (LAN) using the wired connection. In the second, both computers are connected to different networks: the server is still connected in the same private local network, but the client has only access to an internet connection. Since our server is not visible outside its private network, the client connects first with a proxy server. A proxy server is an intermediate computer that sits between the two endpoints. The proxy server makes requests on behalf of the client. Therefore, the client never talks directly to the resource.

In order to use the proposed method in production, an affordable alternative relies on deploying the service to a commercial IaaS (Infrastructure as a service). However, at the moment we write this work, the cloud services that provide rendering services are all located in other continents. Consequently, the latency in this communication is always over 100 ms, which turns their use prohibitive to our purposes. Therefore, in our tests, we only have taken into account the first two scenarios.

Table 5.1: Technical Specification of the computers used in our tests.

	Client	Server
Processor	Intel(R) Core(TM) i7-7700HQ @ 2.80GHz	Intel(R) Core(TM) i7-8700 @ 3.20GHz
Graphics Card	Intel HD Graphics Model 4000	Nvidia GeForce GTX 980
Memory	2x8GB DDR3 1600 MHz	16GB DDR4 2400MHz
Network Interface	Qualcomm Atheros QCA9377 Wireless Network Adapter 866Mbps @ 5GHz connection	Intel Corporation 82578DC Gigabit 1000Mbps
Operating System	Ubuntu 18.04.3 LTS Kernel 5.0.0-23.24	Ubuntu 18.04.2 LTS Kernel 4.18.0-25.26

5.1 Dataset

In order to evaluate the efficiency of our data representation, we used three different real models of oil refineries and platforms. These models differ in the number of objects and scene complexity. Here, we name them as *small*, *medium* and *large* models.

In our first test, we assess how feasible the parametric representation is to reduce the size of industrial-plant CAD models. Looking at the various existing CAD file formats, we notice that the object representation varies among them. Apart from the parametric representation, the majority of CAD file formats also use boundary representation (BREP) and solid revolution to describe the objects. These representations can be directly converted or inferred to the parameters of the shape. However, during our experiments, we observed that some of this information is lost when converted to a neutral file format, such as Wavefront OBJ (McHenry & Bajcsy, 2008). Generally, the objects have their continuous representation converted to the unstructured triangular mesh.

In order to recover the continuous representation of these objects, we preprocess our models with an AI-based reverse engineering method (Moreira, 2015). Given a triangular mesh surface, this method attempts to identify, classify, and recover its continuous representation. In our case, this continuous representation is one of the nine supported standard primitives.

Reverse engineering methods are also useful for reducing object redundancy. Generally, in neutral file formats, when the scene has several instances of the same mesh, the mesh surface is redundantly described for each instance.

Table 5.2: Details of the models used in our experiments.

Model	Total Objects	Mesh Size (MB)	Parametric Size (MB)	Conversion Ratio	Size Reduction
Small	257'296	699.4	166.8	60.5%	76.15%
Medium	487'646	1'842.0	278.4	65.5%	84.88%
Large	659'075	2'555.7	328.6	70.6%	87.14%

Identify such cases is not a naïve task because, in most cases, these meshes have different triangularization or their vertices must be presented in a different order. Therefore, reverse engineering plays an important role in overcoming this object redundancy. As a result, the memory consumption is reduced, and the rendering performance is improved, as we discuss in the next section.

To assess how efficient the parametric representation is for CAD models, we converted as many objects as possible of our three CAD models to the parametric representation. We achieved a compression ratio of up to 87% describing some shapes by their parameters. Table 5.2 describes the obtained results for each one of the models.

In short, the results show the importance of using the parametric representation to define the objects with simple geometries. The compactness of this representation reduces the requirements for storage and transmission. This is particularly important when using a pay-as-you-go service since they use a consumption-based pricing model.

5.2 Rendering Performance

First, we compare two different ways of rendering the scene: issuing one draw call per object and using the instanced rendering approach (Santos & Celes Filho, 2014). In both cases, the rendering performance was measured using the same fixed viewpoint, letting the application continuously rendering the scene. The results are the average of the measurements during a fixed time period. Figure 5.1 shows the obtained results when rendering on the server. We notice that instanced rendering improves up to 9x the rendering performance. The poor performance of the first method comes from the bottleneck on the CPU with API function call overhead and costly memory transfers.

On hybrid cloud rendering systems, the client performance can be described as $x\% + k$, where $x\%$ is the model ratio being rendered locally, and k is a constant factor, due to additional processing tasks like the composition of the final image. The lower the local model ratio, the higher the rendering performance on the client. Conversely, the lower is the final image quality since this image has more chances to have pixels from the remote image. Since the

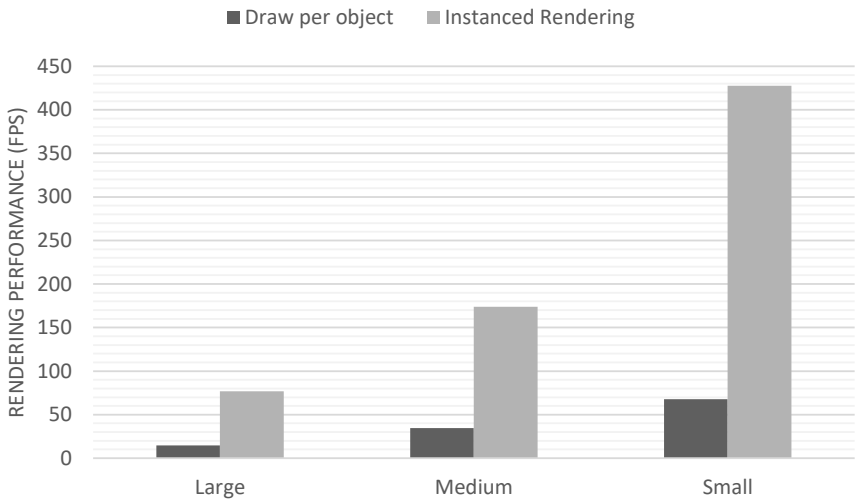


Figure 5.1: Rendering performance comparison for issuing one draw call per object and one draw call for all object instances.

remote image is compressed and it is usually used after a DIBR operation, its quality is lower when compared to the local image.

Figure 5.2 shows the rendering performance for three different workload divisions: rendering the whole model on the server-side (red dot), rendering the entire model on the client-side (orange dot) and, lastly, dividing the rendering tasks for both sides (blue dot).

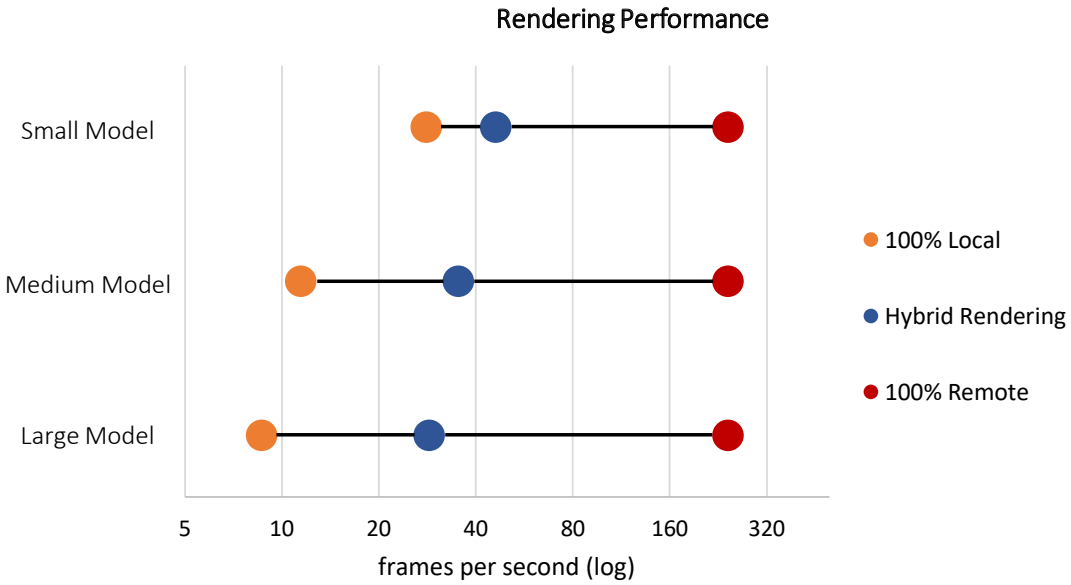


Figure 5.2: Comparison of rendering performance for three different configurations: rendering entire model on the server (higher performance), rendering whole model on the client (higher image quality), and rendering on both sides.

Figure 5.3 shows the response delay of local and remote frames for rendering the large model. Response delay is the elapsed time since a user action occurs until its result is displayed on the screen. The response delay

of the local frame is the same as a regular rendering application since its production does not involve any remote procedure. In our example, this time is around 28 milliseconds, i.e., approximately 35 fps. Once we decouple the client-side operation from the server-side, this is also the overall response delay of our rendering system. When considering only the remote frames, the response delay is higher due to the additional steps: (de)compression, transmission, viewpoint adjustments (DIBR), and others.

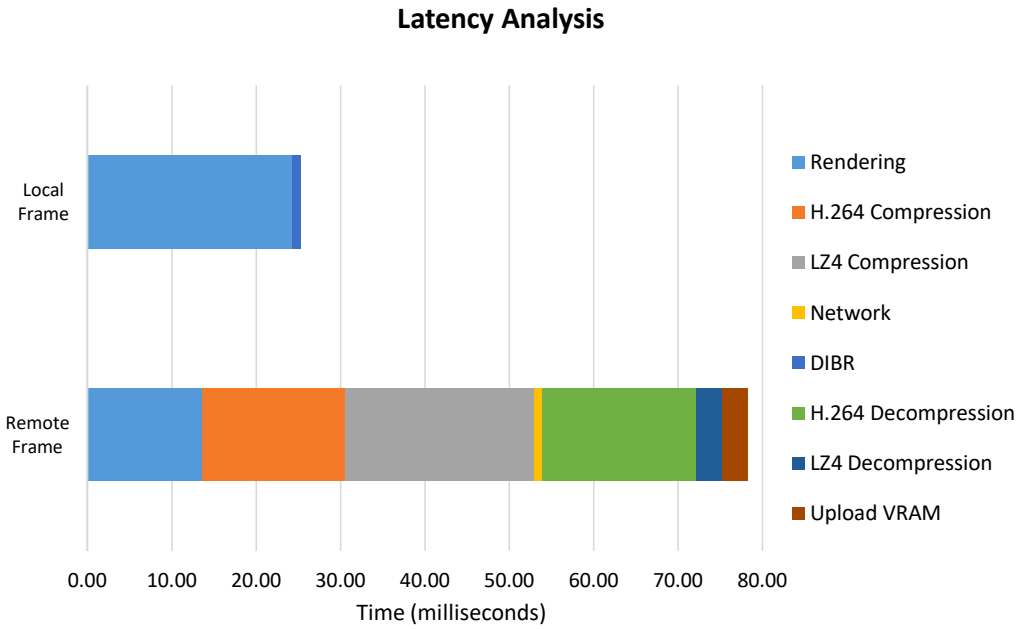


Figure 5.3: The response delay for both local and remote frames be available for rendering on the client-side.

As a future improvement, we will explore alternatives to encode the depth information as part of the image frame. Currently, we are only using a widely adopted real-time compression algorithm, LZ4. Although its impressive runtime performance, it is a general-purpose compression algorithm. Consequently, we are not taking full advantage of the temporal coherence between the produced frames. In our experiments, we used a 16-bits depth map, which compressed is approximately 476 KB on average.

In addition, we intend to leverage the encoding step by shifting this operation to the GPU. The latest graphics cards support fully-accelerated hardware video encoding shipping dedicated chipsets to this goal (Nvidia., 2012). This way, the graphics card run encoding or decoding workloads without slowing the execution of graphics tasks running at the same time. Ultimately, the input frame for encoding is already on VRAM. The transference of the frame from VRAM to RAM is costly, but instead of transferring the raw frame, we would replace it with its compressed version. In our tests, the size of the

raw color frame is approximately 2.63 MB, and when compressed, it can reach up to less than 50 KB.

For each remote image at 720p resolution, we need to transfer around 500 KB of data. Using the dual-view approach (primary and peeled images), and with the server rendering at 13 fps, the required network throughput is around 104 Mb/sec. This is very feasible for local networks, even when using wireless connections. For example, the transfer speed of the 802.11n Wi-Fi standard (Perahia & Stacey, 2013) reaches up to 450 Mb/s. When it comes to a wired Ethernet connection, a network infrastructure using IEEE 802.3an-2006 standard (Gupta et al., 2008) can offer up to 10 Gb/s.

Nonetheless, when we use an internet connection to communicate computers from different networks, this requirement is challenging. In our experiments, when using the internet, we achieved better results when the server produces and transmits only the primary image. This way, we slightly reduce the image quality, but we enhance the user experience by reducing the network throughput requirement. Therefore, we use the dual-view remote images only for inbound connections. Although it is possible to achieve this transfer rate on the internet, especially using optical fiber connection, this suggests the necessity of improving the depth compression.

5.3 Workload Selection

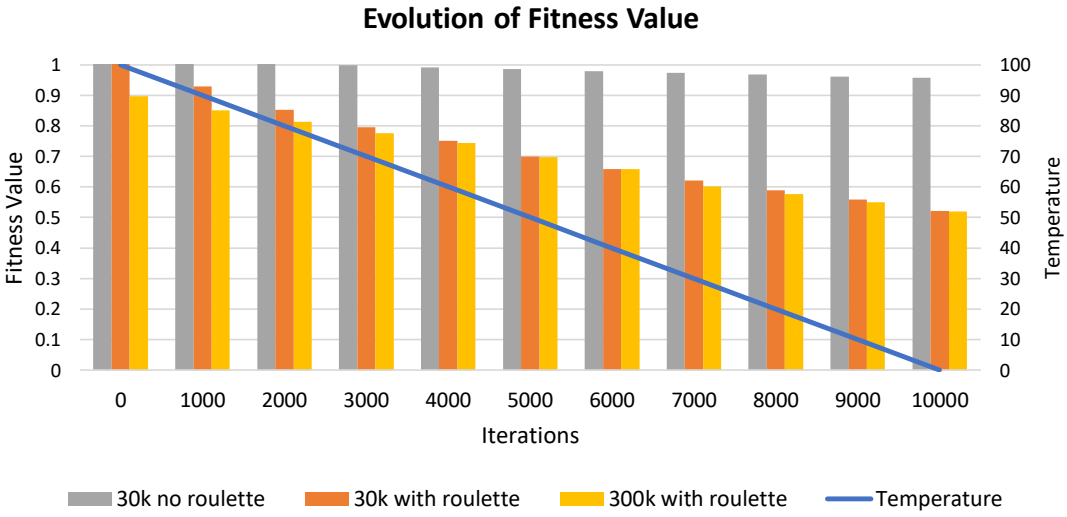


Figure 5.4: The normalized fitness function of our workload optimization method for three different client profiles. We also plotted the temperature values over different iterations.

Regarding our workload selection, we defined three different scenarios to evaluate it. First, we ran our method for selecting 30'000 objects using

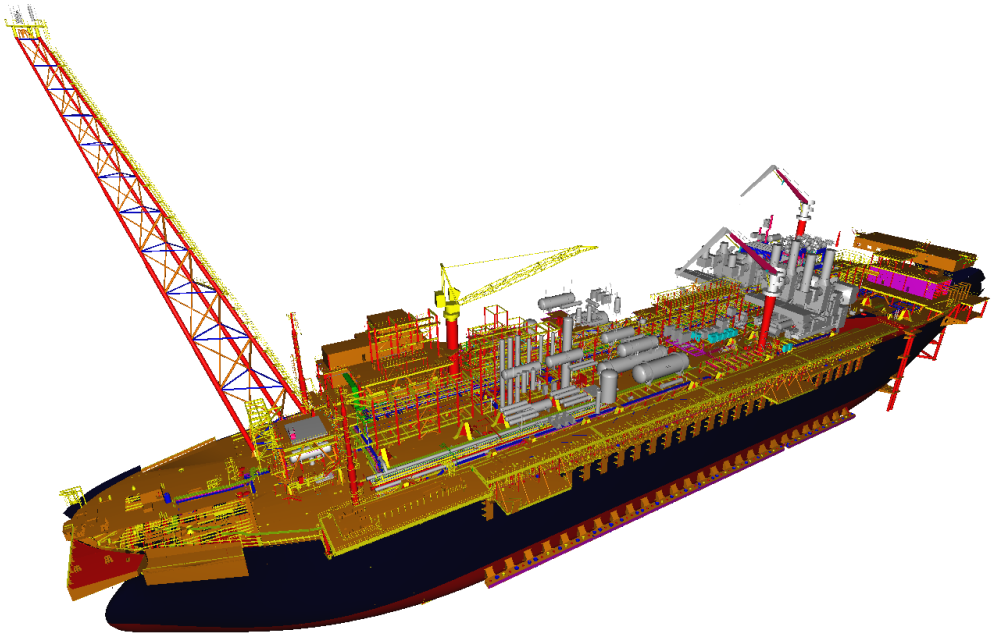


Figure 5.5: The same model presented in Figure1.1, but rendering approximately 27% objects of the original model.

uniform selection distribution. Thus, when moving to a neighboring state in Simulated Annealing, each object has the same probability of being picked up. We also repeated this same setup but using the roulette-wheel instead of uniform distribution for object selection. Finally, we increased the number of objects in the selection.

Figure 5.4 shows the squared normalized results of our fitness function for the three different scenarios. This chart shows how much the roulette-wheel selection method boosts our optimization results. The main reason is that the majority of objects are small (e.g., screws). A uniform random selection is very likely to pick more small objects, and our fitness function is profoundly affected by the size of objects into the set. This is the same cause the value of our fitness function does not improve while we enlarge the size of the client object set. When selecting 30k and 300k objects using roulette-wheel, the values of our fitness has the same pattern. Considering the model used in this test, the selection of 30k objects has already encompassed the largest scene objects. This way, increasing the number of objects barely minimizes the fitness value.

Regarding the quality of workload selection, we noticed that some apparently large objects, which would have a good chance of being selected, were never in the final set, regardless of how many times we ran our method. We identified that some of these "large" objects are actually the junction of several other smaller objects. Figure 5.6 shows a case where the cylinder side was modeled as several rectangles.

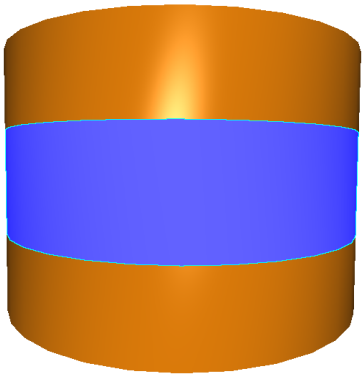


Figure 5.6: Example of surface fragmentation. The entire cylinder is an equipment body, but it is modeled as the junction of three smaller cylinders. The cylinder in the middle is hilighted just for the sake of clarity.

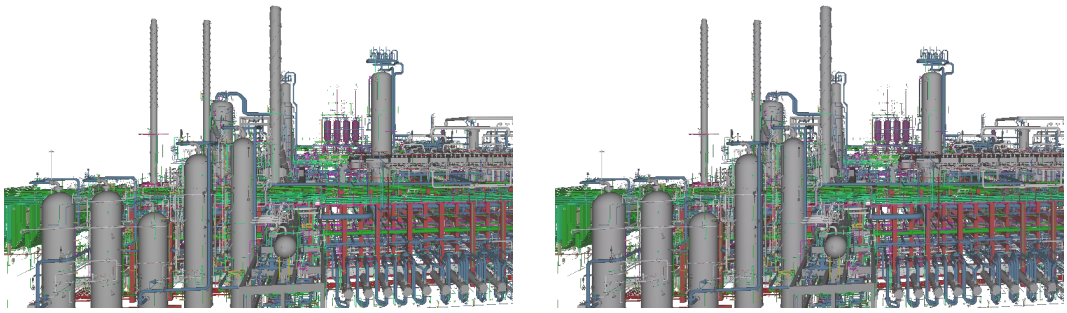
We overcame this using the model's metadata to guide the object selection process. The model's objects are organized hierarchically in logical nodes, which in turn contain some valuable associated information. For example, the equipment has information about its operating specifications. Therefore, instead of randomly choosing model geometries, we select these logical nodes, and then we retrieve the geometries associated with this node.

Lastly, Figure 5.5 shows the result of rendering the same model depicted in Figure 1.1 but rendering only approximately 27% of objects from the original models. The initial temperature was 100, with 0.02 of decay. The majority of the missing objects are screws, valves, and other small objects. Besides, we have a considerable amount of objects spread in the whole scene extents. This visualization is enough to allow the user to navigate through the scene even when the server is not responding.

5.4 Image Quality

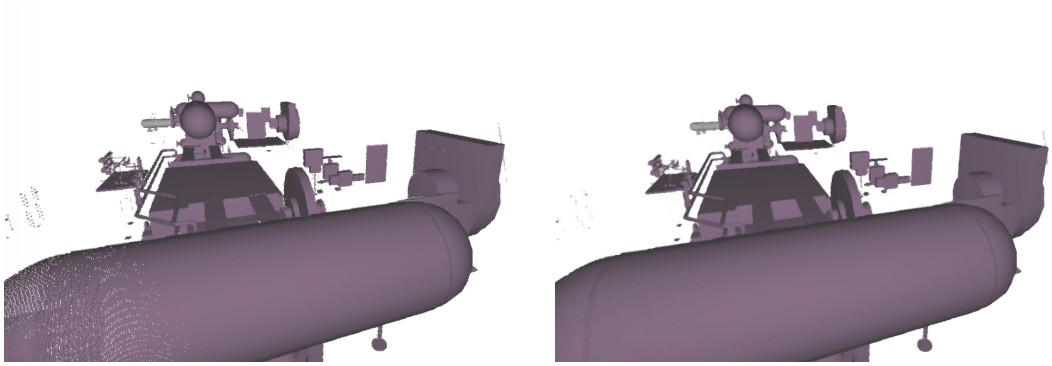
In order to evaluate the image quality, we used the structural similarity (SSIM) (Wang et al., 2004) index, a popular method for measuring the similarity between two images based on human perception. It serves as a quantitative measurement of the quality of one image when compared to the perfect image, i.e., the ground truth.

The image quality can be affected mostly due to three factors: image compression, remote image undersampling, and when a remote object previously occluded or out of the field of vision becomes visible. The image compression is the most controlled one. It is just a matter of deciding between image quality and transmission efficiency. In the cases the network is not a bottleneck, the



(a) (b)

Figure 5.7: Image quality comparisson between the original raw image and the same image compressed with H.264 codec. (a) Ground-truth image, i.e., the raw remote image. (b) The same image from (a) but compressed with H.264 codec (SSIM = 98.16%).



(a) (b)

Figure 5.8: Example of holes on the object's surface due undersampling. (a) is the result of DIBR rendering using an outdated remote image. In (b) we mitigate the holes using the median filter with 5x5 window size.

server can afford a higher image quality. Figure 5.7 shows an image in its raw and compressed version. In this example, the SSIM value between these two images is 98.16%.

The remote undersampling happens when the client rapidly moves closer to a remote object. In this case, the client requests a new frame to the server, and in the meanwhile, the client uses the last available remote image to produce the final image. Nonetheless, this remote image does not have enough pixels to cover the region in the final image, where the remote object is supposed to be. As a consequence, some holes appear on the object's surface. We attenuate these artifacts using the median filter (McGuire, 2008) when fetching pixels from the source image (Figure 5.8).

When the user rotates the camera, there is a high probability that a vast region of the screen is blank. This effect is greater depending how faster the user rotates the camera because this movement highly alters the temporal

coherence of the remote frames. Therefore, while new remote images does not arrive on the client, these missing regions will only contain the objects from local rendering, as shown is Figure 5.9.

We also evaluated the final image quality for different combinations of latency values and compression ratio on the remote images. Figure 5.10 depicts this experiment. We captured these values in a very dense spatial region, where more than 80% of the pixels in the final images belong to the remote image. The bubble size is the obtained quality value (SSIM); the biggest bubble value is 0.78, and the value of the smallest is 0.59. We can easily conclude that the latency affects much more the image quality when compared to image compression. Thus, the system can use high compression ratios, once this vastly saves network bandwidth and reduces the transmission time without compromising image quality. Also, this experiment reinforces the importance of our time camera prediction model on the server-side, as a means of mitigating the network fluctuations by anticipating futures events.

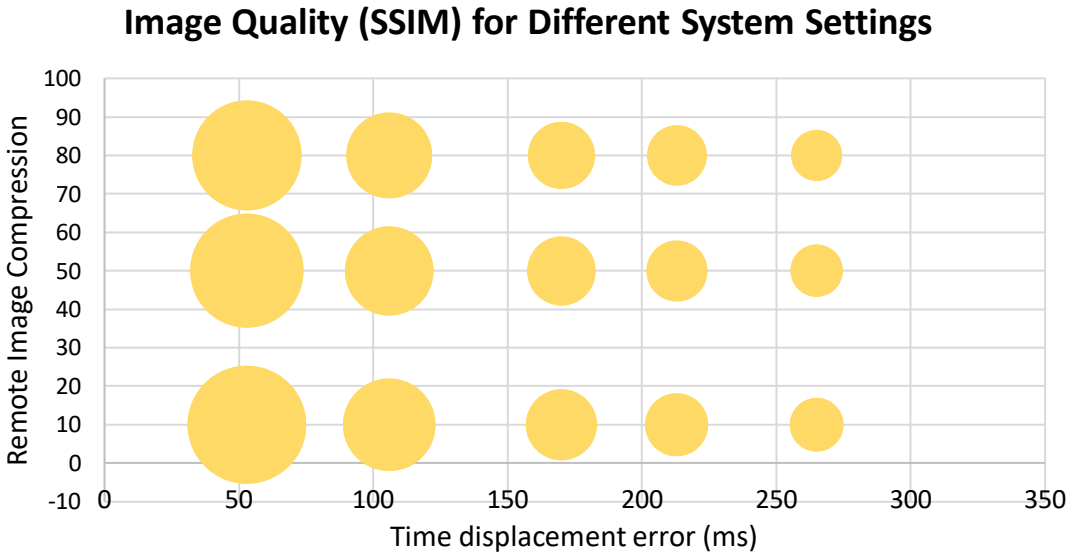
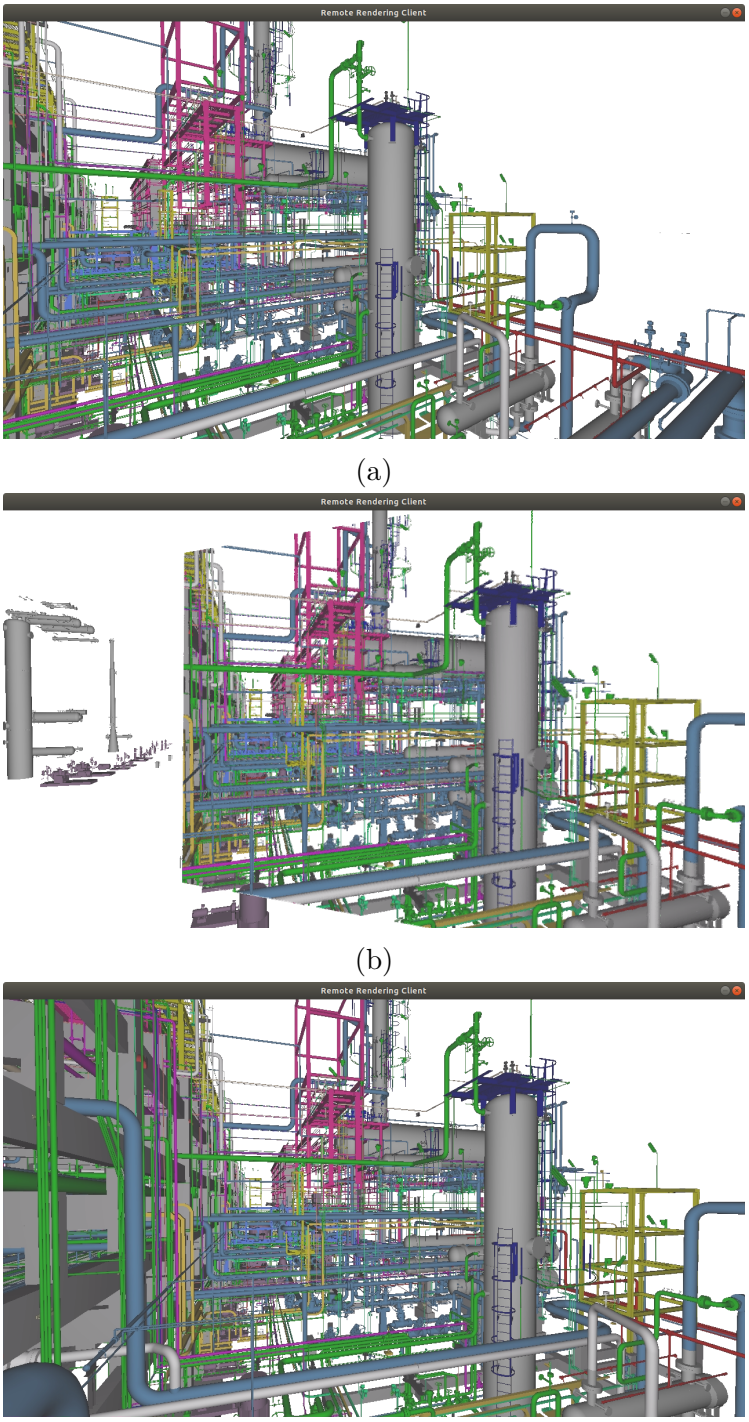


Figure 5.10: The influence of time displacement error and image compression on the image quality. The bubble size is the resulting SSIM value for the given system settings.



(a)

(b)

(c)

Figure 5.9: Example of blank region due to camera rotation movement. (a) shows the initial state, before the rotation movement. (b) was obtained by disconnecting the server on purpose and then moving the client camera to the left. In this image we can see clearly the plane of the last remote image available. The objects that appear on the left are being rendered on the client. Finally, the rotated image is updated when new remote images arrive on the client (c).

6
Future Research Opportunities

While conducting this research, we identified some research opportunities from different fields of computer science that may contribute to the study of rendering CAD models in the cloud. In this chapter, we discuss the main remaining challenges and their possible solutions, thus serving as a guide for future research.

6.1
Artificial Intelligence

In the last decades, artificial intelligence has been experiencing significant advances, pushing the boundaries of several research fields. Among them, the image processing is one of the most notable areas. Recent works have succeeded in using convolutive networks to make improvements to input images. We claim that some of these studies can also be employed on cloud rendering services to improve their results.

6.1.1
Super-Resolution Imaging

High-resolution devices have become increasingly accessible, and it is worth noting the recent popularization of 4k devices. The use of these displays in the cloud rendering scenario presents the challenge of transmitting high definition images from the server to the client without latency and without overloading the network. The super-resolution imaging consists of a set of techniques aimed at the upscale image with no quality degradation. Image super-resolution techniques are not new. The first work in this subject was published in 1984
citeptsai1984multiframe. The traditional approach for super-resolution imaging relies on multiple images of the same scene from slightly different perspectives.

In recent years, we witnessed promising results using deep learning techniques, especially convolutional-based networks, to solve super-resolution problems. The basic difference of these techniques arises from their network architecture

citepkim2016accurate, lim2017enhanced, ahn2018fast, loss function citepsajjadi2017enhancenet, johnson2016perceptual, bulat2018super and how they train the network
citeplim2017enhanced.

We encourage the usage of super-resolution techniques as a mechanism to reduce the burden from the server and the network. When the client is using a high-resolution display, the server could render a low-resolution image, and the client is responsible for image magnification when necessary. Consequently, the server can handle more incoming connections, and the network latency is mitigated.

6.1.2 AI-based Image Completion

Depth-image based rendering is a valuable technique to produce novel images from an arbitrary viewpoint given a reference image and its transformation matrix. Its high efficiency and reasonable results turn these techniques very appealing to provide temporary results in cloud rendering. Nonetheless, the drawback of these techniques is the presence of holes in the synthesized images.

The image completion comprises several techniques that aim filling-in target regions. They are used to remove unwanted objects or to generate occluded regions for image-based 3D reconstruction. In order to succeed, it is important to these techniques the high-level recognition of the scene instead of just filling textured patters.

We believe that such techniques can be successfully employed to fill the holes in the warped images by the depth-image-based rendering. Some works yield impressive results, even for complex images
citepiizuka2017globally. One of the most wanted use cases is the crowd removal from tourist spots. Our images are even simpler than these real-world images in terms of textures, illumination, and coherence. Thus, the use of these techniques to fill the holes of our images would not incur additional challenges.

6.2 Autonomic Computing

Autonomic computing is an emerging philosophy inspired by the autonomic nervous system of the human body. This nervous system controls important vital signals (e.g., breathing rate, heart rate, and body temperature) without any conscious intervention. The increasing complexity of modern systems leads to an enormous complexity in maintaining these systems. Autonomic

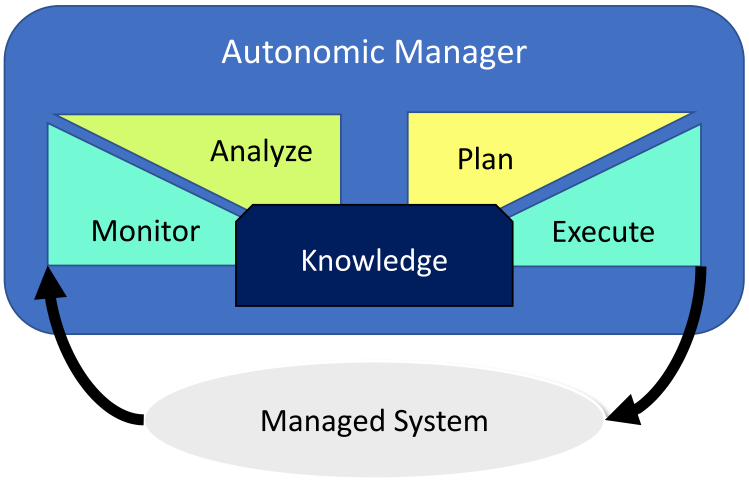


Figure 6.1: Representation of the stages from MAPE-K feedback loop.

systems are supposed to be capable of running themselves, even on different unknown circumstances, and adjusting their internals to achieve the highest possible performance.

The MAPE-K (Monitor, Analyze, Plan, Execute, and Knowledge) citepcomputing2006architectural feedback loop is one of the most famous architecture for autonomic computing. This framework is constantly checking and optimizing its status and automatically adapt itself to changing conditions.

Figure 6.1 shows the general steps of the MAPE-K framework. The monitor stage is responsible for gathering information about the current state of the managed system. This information includes internal and external system metrics, for example, performance and resource availability metrics, respectively. This collected data is used as the input of the next step (*Analyze*). The system performs analysis and reasoning over the raw data to identify possible failures or unwanted conditions. If changes are required, a change request is passed to the planning process. The plan process structures all the actions needed to overcome the identified issues. The actions involved can consist of a single command or a complex workflow. In the executed phase, the system executes the actions recommended by the previous stage. All the historical data of the system management is stored in the knowledge base, and it is very useful to guide future decisions.

The cloud rendering presented in this work contains many parameters that influence the system operations. The ideal values for these parameters are hard to decide in advance due to the unpredictable conditions of the environment. In order to achieve good results, these parameters must be dynamic, as the conditions are. For example, a load balance could change the workload division during the system execution, either because the server

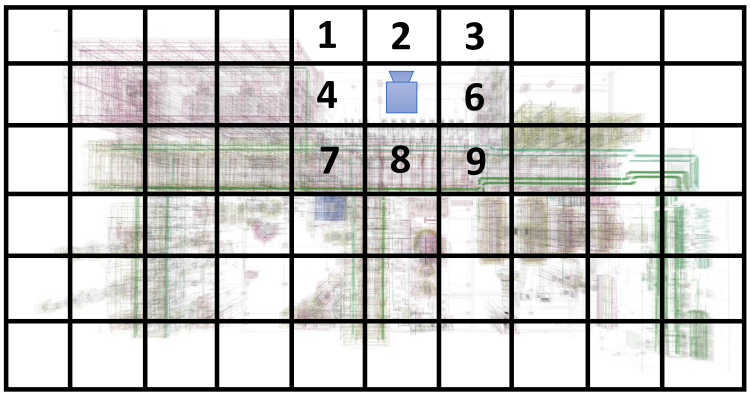
has more clients to serve or because the user started using other compute-intensive tools. Other examples are the video compression rate, the server camera displacement, the division between the client's static and dynamic object set.

6.3 Semantic Optimization

A valuable advantage of using cloud applications is the possibility of gathering information about the users and how each one of them interacts with the application. The system can use this information to identify usage patterns and, consequently, to make some decisions. For example, citepposada2005methodology provided adapted visualizations depending on each user and his intentions.

Considering our proposal, we could use such information to support the object selection for the client's dynamic set. Currently, this process consists only of selecting the nearest neighbor objects. However, in an engineering project, each engineering experts (e.g., structural, mechanical, electrical) has different background and intentions. When selecting the nearest objects, we could use this information to filter the objects that the user is probably more interested in.

Besides, we believe that a statistical motion-aware prediction model could succeed for prefetching objects to the client's dynamic set. Currently, we are using the displaced server camera, which assumes the user will indefinitely walk in the same direction. In this new approach, for each model, the system could make assumptions about future positions considering the historical navigation data for each user group. To achieve this, we suggest the use of Markov Chain citeppvribyl2013motion.



Transition Matrix

States	5
1	0.027534
2	0.173132
3	0.08139
4	0.209113
5	0.026806
6	0.064818
7	0.269674
8	0.081445
9	0.066087

Figure 6.2: First-order Markov Chain representation for 3D model navigation. The transitions matrix shows the probabilities for state changing considering the state 5 as the current state.

Markov Chain is a stochastic model that describes a sequence of possible events. According to this method, the probability of a particular state of the system in the next time interval depends only on the current state and a set of defined transition probabilities. These probabilities can be organized in a matrix form, called stochastic matrix.

A simple approach would be the division of the scene into a regular grid, in which each cell corresponds to a single state, as the schematic example in Figure 6.2. Since we are dealing with the direction of the user's movement, a high order Markov Chain would fit better. In this method, the probability of the state in the next time interval depends on the current and the previous state. For a new model, the transition probabilities would initially be the same for all transitions. Nevertheless, over time, these probabilities would be updated and refined as more engineering experts use the model. As a result, the system could make assumptions even if the user is navigating through the model for its first time.

7
Conclusion

The efficient rendering of massive CAD models has always been a challenging task, especially for interactive visualization. During the past decades, several studies were presented to overcome the massiveness of such models. However, recently, we have witnessed an exponential growth in data generation, both in size and details, while the advances in processing capabilities have been slowing down.

When it comes to computer graphics, cloud rendering has been the solution adopted by many areas of interest. Nonetheless, there is an absence of existing cloud rendering services for industrial-plant CAD models. The main reason is that the existing services are mostly general-purpose rendering services.

All these reasons motivated us to study and propose a novel hybrid cloud rendering approach to support interactive visualization of CAD models. In this work, we attempt to bridge this existing gap between cloud rendering and CAD models. The construction of a specific cloud service for CAD models allows us to make use of some premises to improve the performance of the proposed solution.

In our hybrid rendering approach, the server is responsible for overwhelming rendering tasks, while the client renders a smaller set of objects, proportional to its rendering capabilities. The server provides to the client a dual-view depth-augmented images from the remote scene. This remote image is combined with the local image to produce the final image.

Since our method focus on CAD models, it benefits from some assumptions to improve the system performance. For example, CAD models are mostly composed of simple geometries, which are suitable for the use use of implicit representation. Besides, the objects are very redundant, which allows the use of instanced rendering, resulting in an efficient rendering approach.

Regarding the communication model, we propose a loosely coupled communication mechanism between client and server. As a result, the client application remains responsive even if the server stops responding. The drawback of this approach is the final image degradation when the remote frame is delayed. This happens because the client does not have an updated remote frame. How-

ever, the server anticipates possible future network fluctuations displacing its camera in time.

We also established a multi-objective optimization problem to guide our workload selection. In this problem, we attempt to select an object set to render on the client in such a way the select objects can grant spatial awareness to the user. To solve the optimization problem, we use the simulated annealing metaheuristic since the optimal solution of a combinatorial problem has pseudo-polynomial complexity.

The results show that our technique can achieve high frame rates with satisfactory image quality, even in an adverse environment, such as a high latency network or thin devices. In case the network is not performing well, the client-side is still able to produce interactive frame rates, but slightly reducing the final image quality, as expected. Lastly, the presented method paves the way for other research opportunities. We detail them in Chapter 6.

7.1 Future Work

Considering the current state of the presented work, we point out some suggestions for future work:

- **Dynamic workload ratio**. The workload division between the client and the server is defined during the connection establishment phase. For this, the server takes into account both the computational capability of the client's computer and its historical performance data. However, the performance of both sides varies as the availability of resources also changes. For example, if the user runs another heavy consuming-resources application, the performance of local rendering will be affected. When it comes to portable devices, the battery level is an important aspect to take into consideration. As more workload on client side of portable devices, more energy will be required. Ideally, the workload ratio between the client and the server should vary as the availability of computational resources also changes.
- **Camera Prediction Validation**. As discussed before, the server camera is displaced in time to anticipate future client viewpoints. By now, this displacement is only evaluated using the historical performance data of both network latency and rendering capacity of the client. However, due to the high complexity of the cloud environment, this estimate may not be accurate in some scenarios. Therefore, the client could validate the predicted viewpoints positions and computed error metrics. This metric

could be reported to the server in order to fine-tune its camera displacement.

– **General performance improvements**. Our goal in this work is to investigate a method for rendering massive CAD models using cloud services. Nonetheless, to use it in production, some improvements must be made. The most important is the image encoding on the GPU since this is the most time-consuming step for producing the remote frame. Next, the current algorithm used for depth image compression does not make use of temporal coherence to achieve better results.

Bibliography

ABRAHAM, FREDERICO; CELES, WALDEMAR; CERQUEIRA, RENATO; CAMPOS, JOAO LUIZ. **A load-balancing strategy for sort-first distributed rendering**. Proceedings. 17th Brazilian Symposium on Computer Graphics and Image Processing. IEEE. 2004, pp. 292–299.

AHMADI, HAMED; HASHEMI, MAHMOUD REZA; SHIRMOHAMMADI, SHERVIN. **An Open Source Cloud Gaming Testbed Using DirectShow**. 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom). IEEE. 2015, pp. 606–610.

ASSUNCAO, RENATO. **Testing spatial randomness by means of angles**. Biometrics (1994), pp. 531–537.

AZUMAH, KENNETH KWAME; KOSTA, SOKOL; SØRENSON, LENE TOLSTRUP. **Scheduling in the hybrid cloud constrained by process mining**. 2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom). IEEE. 2018, pp. 308–313.

BAO, PAUL; GOURLAY, DOUGLAS. **A framework for remote rendering of 3-D scenes on limited mobile devices**. IEEE Transactions on Multimedia 8.2 (2006), pp. 382–389.

BARTIIK, MATĚJ; UBIK, SVEN; KUBALIK, PAVEL. **LZ4 compression algorithm on FPGA**. Electronics, Circuits, and Systems (ICECS), 2015 IEEE International Conference on. IEEE. 2015, pp. 179–182.

BHATIA, NITIN et al. **Survey of nearest neighbor techniques**. arXiv preprint arXiv:1007.0085 (2010).

BOOST. **Boost C++ Libraries**. `http://www.boost.org/`. Last accessed 2020-01-30. 2015.

CHEN, PO-HAN; CUI, LU; WAN, CAIYUN; YANG, QIZHEN; TING, SENG KIONG; TIONG, ROBERT LK. **Implementation of IFC-based web server for collaborative building design between architects and structural engineers**. Automation in construction 14.1 (2005), pp. 115–128.

CLAYPOOL, MARK; CLAYPOOL, KAJAL. **Latency and Player Actions in Online Games**. Commun. ACM 49.11 (2006), pp. 40–45. ISSN: 0001-0782. DOI: 10.1145/1167838.1167860. URL: https://doi.org/10.1145/1167838.1167860.

DEFANTI, THOMAS A et al. **The future of the CAVE**. Central European Journal of Engineering 1.1 (2011), pp. 16–37.

DUNSTON, PHILLIP S; WANG, XIANGYU. **Mixed reality-based visualization interfaces for architecture, engineering, and construction industry**. Journal of construction engineering and management 131.12 (2005), pp. 1301–1309.

DYKEN, CHRISTOPHER; LYE, KJETIL OLSEN; SELAND, JOHAN; BJØNNES, ERIK W; HJELMERVIK, JON; NYGAARD, JENS OLAV; HAGEN, TROND RUNAR. **A framework for OpenGL client-server rendering**. 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings. IEEE. 2012, pp. 729–734.

FEHN, CHRISTOPH. **Depth-image-based rendering (DIBR), compression, and transmission for a new approach on 3D-TV**. Stereoscopic Displays and Virtual Reality Systems XI. Vol. 5291. International Society for Optics and Photonics. 2004, pp. 93–104.

GOOGLE INC. **Protocol Buffers**. `https://developers.google.com/protocol-buffers`. Last accessed: 2020-01-31. 2001.

GUPTA, SANDEEP et al. **A 10Gb/s IEEE 802.3 an-compliant ethernet transceiver for 100m UTP cable in 0.13 μm CMOS**. 2008 IEEE International Solid-State Circuits Conference-Digest of Technical Papers. IEEE. 2008, pp. 106–599.

HUMPHREYS, GREG; ELDRIDGE, MATTHEW; BUCK, IAN; STOLL, GORDAN; EVERETT, MATTHEW; HANRAHAN, PAT. **WireGL: a scalable graphics system for clusters**. Proceedings of the 28th annual conference on Computer graphics and interactive techniques. ACM. 2001, pp. 129–140.

HUMPHREYS, GREG; HOUSTON, MIKE; NG, REN; FRANK, RANDALL; AHERN, SEAN; KIRCHNER, PETER D; KLOSOWSKI, JAMES T. **Chromium: a stream-processing framework for interactive rendering on clusters**. ACM transactions on graphics (TOG) 21.3 (2002), pp. 693–702.

KAM, KIN MING; ZENG, LI; ZHOU, QIANG; TRAN, RICHARD; YANG, JIAN. **On assessing spatial uniformity of particle distributions in quality control of manufacturing processes**. Journal of Manufacturing Systems 32.1 (2013), pp. 154–166.

KIRKPATRICK, SCOTT; GELATT, C DANIEL; VECCHI, MARIO P. **Optimization by simulated annealing**. science 220.4598 (1983), pp. 671–680.

LAI, ZEQI; HU, Y CHARLIE; CUI, YONG; SUN, LINHUI; DAI, NINGWEI; LEE, HUNG-SHENG. **Furion: Engineering high-quality immersive virtual reality on today's mobile devices**. IEEE Transactions on Mobile Computing (2019).

LAMBERTI, FABRIZIO; SANNA, ANDREA. **A streaming-based solution for remote visualization of 3D graphics on mobile devices**. IEEE transactions on visualization and computer graphics 13.2 (2007).

LIN, ZIQIAO; WANG, ZEHUA; CAI, WEI; LEUNG, VICTOR CM. **A Novel Game Map Preloading and Resource Provisioning Scheme in Cooperative Cloud Networks**. 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom). IEEE. 2017, pp. 210–217.

LIPOWSKI, ADAM; LIPOWSKA, DOROTA. **Roulette-wheel selection via stochastic acceptance**. Physica A: Statistical Mechanics and its Applications 391.6 (2012), pp. 2193–2196.

LIU, BAOQUAN; WEI, LI-YI; XU, YING-QING; WU, ENHUA. **Multi-layer depth peeling via fragment sort**. 2009 11th IEEE International Conference on Computer-Aided Design and Computer Graphics. IEEE. 2009, pp. 452–456.

LOUKOPOULOS, THANASIS; TZIRITAS, NIKOS; KOZIRI, MARIA; STAMOULIS, GEORGIOS; KHAN, SAMEE. **A Pareto-Efficient Algorithm for Data Stream Processing at Network Edges**. 2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom). IEEE. 2018, pp. 159–162.

LUTHRA, AJAY; SULLIVAN, GARY J; WIEGAND, THOMAS. **Introduction to the special issue on the H. 264/AVC video coding standard**. IEEE Transactions on Circuits and Systems for Video Technology 13.7 (2003), pp. 557–559.

MARLER, R TIMOTHY; ARORA, JASBIR S. **Survey of multi-objective optimization methods for engineering**. Structural and multidisciplinary optimization 26.6 (2004), pp. 369–395.

MCGUIRE, MORGAN. **A Fast, Small-Radius GPU Median Filter**. Published in ShaderX6. ShaderX6. 2008. URL: `https://casual-effects.com/research/McGuire2008Median/index.html`.

MCHENRY, KENTON; BAJCSY, PETER. **An overview of 3d data content, file formats and viewers**. National Center for Supercomputing Applications 1205 (2008), p. 22.

MOREIRA, A. **Engenharia Reversa em Modelos CAD Utilizando Descritores de Forma e Maquina de Vetores de Suporte**. MA thesis. PUC–Rio, 2015.

NVIDIA. **Nvidia NVENC**. `https://developer.nvidia.com/video-encode-decode-gpu-support-matrix`. Last accessed: 2020-02-01. 2012.

OAT, CHRISTOPHER. **Rendering to an off-screen buffer with WGL_ARB_pbuffer**. Technology paper of ATI Inc (2008), pp. 1–13.

PĂTRĂUCEAN, VIORICA; ARMENI, IRO; NAHANGI, MOHAMMAD; YEUNG, JAMIE; BRILAKIS, IOANNIS; HAAS, CARL. **State of research in automatic as-built modelling**. Advanced Engineering Informatics 29.2 (2015), pp. 162–171.

PERAHIA, ELDAD; STACEY, ROBERT. **Next generation wireless LANs: 802.11 n and 802.11 ac**. Cambridge university press, 2013.

PERRY, JN; MEAD, R. **On the power of the index of dispersion test to detect spatial pattern**. Biometrics (1979), pp. 613–622.

PETERNIER, ACHILLE; CARDIN, SYLVAIN; VEXO, FRÉDÉRIC; THALMANN, DANIEL. **Practical design and implementation of a CAVE environment**. Proceedings 2nd International Conference on Computer Graphics Theory. 2007, pp. 129–136.

REINERT, BERNHARD; KOPF, JOHANNES; RITSCHEL, TOBIAS; CUERVO, EDUARDO; CHU, DAVID; SEIDEL, HANS-PETER. **Proxy-guided image-based rendering for mobile devices**. Computer Graphics Forum. Wiley Online Library. 2016, pp. 353–362.

RENAMBOT, LUC et al. **Sage: the scalable adaptive graphics environment**. Proceedings of WACE. Citeseer. 2004, pp. 2004–09.

REQUICHA, ARISTIDES AG; VOELCKER, HERBERT B. **Solid modeling: a historical summary and contemporary assessment**. IEEE computer graphics and applications 1.2 (1982), pp. 9–24.

RIVARD, HUGUES. **A Survey on the impact of information technology in the Canadian architecture, engineering and construction Industry.** ITcon 5 (2000), pp. 37–56.

AL-ROUSAN, NABIL M; CAI, WEI; JI, HONG; LEUNG, VICTOR CM. **DCRA: Decentralized Cognitive Resource Allocation Model for Game as a Service**. 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom). IEEE. 2015, pp. 218–225.

SANTOS, PAULO IVSON NETTO; CELES FILHO, WALDEMAR. **Instanced rendering of massive cad models using shape matching**.

2014 27th SIBGRAPI Conference on Graphics, Patterns and Images. IEEE. 2014, pp. 335–342.

SCHALLER, ROBERT R. **Moore's law: past, present and future**. IEEE spectrum 34.6 (1997), pp. 52–59.

SCHEIFLER, ROBERT W; GETTYS, JIM. **The X window system**. ACM Transactions on Graphics (TOG) 5.2 (1986), pp. 79–109.

SEMSARZADEH, MEHDI; YASSINE, ABDULSALAM; SHIRMOHAMMADI, SHERVIN. **Video encoding acceleration in cloud gaming**. IEEE Transactions on Circuits and Systems for Video Technology 25.12 (2015), pp. 1975–1987.

SHEA, RYAN; LIU, JIANGCHUAN; NGAI, EDITH C-H; CUI, YONG. **Cloud gaming: architecture and performance**. IEEE network 27.4 (2013), pp. 16–21.

SHI, SHU; HSU, CHENG-HSIN. **A survey of interactive remote rendering systems**. ACM Computing Surveys (CSUR) 47.4 (2015), p. 57.

TAMM, GEORG; KRÜGER, JENS. **Hybrid rendering with scheduling under uncertainty**. IEEE transactions on visualization and computer graphics 20.5 (2014), pp. 767–780.

WANG, SHAOXUAN; DEY, SUJIT. **Modeling and characterizing user experience in a cloud server based mobile gaming approach**. Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE. IEEE. 2009, pp. 1–7.

WANG, ZHOU; BOVIK, ALAN C; SHEIKH, HAMID R; SIMONCELLI, EERO P, et al. **Image quality assessment: from error visibility to structural similarity**. IEEE transactions on image processing 13.4 (2004), pp. 600–612.

XU, XUYUAN; PO, LAI-MAN; NG, KA-HO; FENG, LITONG; CHEUNG, KWOK-WAI; CHEUNG, CHUN-HO; TING, CHI-WANG. **Depth map misalignment correction and dilation for DIBR view synthesis**. Signal Processing: Image Communication 28.9 (2013), pp. 1023–1045.