EasyChair Preprint

№ 1700

Metaheuristic-based Workload Selection for Hybrid Cloud Rendering of CAD Models

André Moreira and Waldemar Celes

October 17, 2019

Metaheuristic-based Workload Selection for Hybrid Cloud Rendering of CAD Models

André Moreira, Waldemar Celes
Tecgraf Institute and Computer Science Department
Pontifical Catholic University of Rio de Janeiro - PUC-Rio
Rio de Janeiro, Brazil
{asouza, celes}@tecgraf.puc-rio.br

***Abstract*—Efficient rendering of massive industrial-plant CAD models is a long-time challenge even when using high-end computers. This problem has been aggravated with the popularization of web-based applications and lightweight devices (e.g., tablets and smartphones) in engineering daily basis activities. One promising solution consists in pushing rendering tasks to the cloud. However, the existing cloud rendering platforms are not able to render CAD models efficiently. In this work, we aim to fill the gap between industrial-plant models and cloud rendering by proposing a hybrid cloud rendering method. In our approach, we define two sets of objects: one for rendering on client-side, and the other for rendering on server-side. The client produces the final image combining local and remote images. As a consequence, our cloud rendering system provides better image quality and more robustness to network fluctuations. A key component of such a solution is the choice of objects to be rendered on the client. We propose a metaheuristic-based method to select such objects. Our solution privileges larger objects with a uniform spatial distribution. This way, the user can still coherently navigate even when the server is not responding. Our results show that our approach enables thin devices to interactively visualize massive CAD models with good image quality, even when the networking is not performing well.**

***Index Terms*—Cloud Rendering, CAD Models, Workload Selection, Hybrid Cloud Rendering**

I. Introduction

Computer-Aided Design (CAD) models of industrial plants play important roles in engineering project management, reducing the costs and the project risks. This importance led to the recent popularization of using these models on daily activities of Architecture, Engineering & Construction (AEC) professionals [1]–[3]. Nonetheless, efficiently rendering CAD models is challenging because they are massive in data volume and detail, as shown in Fig. 1. This problem becomes worse with the increasing use of web-based applications and portable devices, such as smartphones and tablets, to support engineering activities.

Recent successful initiatives to handle massive data consist in using cloud services to process overwhelming tasks. Cloud rendering is the process of rendering 3D models on a remote computer and then displaying the resulting frames on the client. The cloud gaming industry has achieved notable progress motivated by high market demand. However, the existing cloud rendering solutions do not suit well for rendering CAD models, since massiveness is not an issue in the game domain; they are mostly concerned about game synchronization.

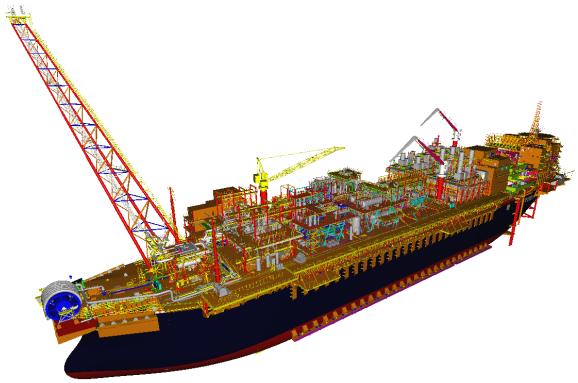


Fig. 1. Example of a CAD model of oil-platform plant. This model contains nearly 2 millions of objects.

In this work, we aim to fill the gap between cloud rendering services and industrial plant CAD models by proposing a hybrid cloud rendering method. In our approach, we split the model into two object groups, one to be rendered on the client-side and the other, generally more extensive, to be rendered on the server-side. The client then produces the final image combining the local image with the remote image, provided by the server. Rendering in client-side improves the final image quality since the local image is not compressed, and also reduces the server overhead once the server renders fewer objects if the client is a high-end device.

Besides the massiveness, CAD models present particularities that invoke for a specialized solution. First, CAD viewers render head-up display (HUD) elements (e.g., texts), which bring essential information. The rendering of such elements does not preserve adequate quality if submitted to image compression. Also, CAD models are generally built on top of several instances of simple objects (e.g., cylinders, spheres, boxes). We take benefit from object instancing to optimize our renderer, enabling the client to draw more objects on its side. Moreover, the simple representation of such objects allows us to use an efficient data representation for transmission, storage, and rendering.

The client-server communication model is loosely coupled by using asynchronous event-based communication. Consequently, we prevent halting the client on waits for server response. The client always uses the latest available remote frame to compose the current local image. Also, we establish a camera prediction model to displace the server camera in time aiming to anticipate future client requisitions.

We also propose a novel procedure to select the object set to the client. We establish a multi-objective optimization problem to select the most significant scene objects, enforcing, at the same time, a uniform spatial distribution. We solve this combinatorial problem by using simulated annealing, a traditional metaheuristic method, to obtain a feasible solution in a reasonable time, since this is a pseudo-polynomial problem.

The main advantage of our selection procedure is to use the client capabilities to render the objects that are significant in the resulting image. At the same time, these objects bring enough spatial information for the user to interact. This way, the user can still navigate throughout the scene when the server response time is high or even when it is not responding anymore.

It is worth to highlight that, although we are selecting a reasonable candidate set to mitigate latency issues, the proposed technique is also useful for reducing costs when using a cloud service with pay-as-you-go billing model [4]. In a walkthrough CAD application, the user does not need all the scene details, so the application can use our optimization to produce a coarse version of the original model.

The obtained results make us very confident that offloading the rendering tasks of massive and complex CAD models to the cloud is the next big thing for Architecture, Engineering & Construction (AEC) professionals.

The rest of this paper is structured as follows: The related works are presented in Section II. In Section III, we introduce our hybrid cloud rendering approach for CAD Models. Next, in Section IV, we detail our metaheuristic-based workload selection algorithm to determine the object set that the client will render locally. We discuss our results in Section V. Lastly, we make our final remarks in Section VI.

II. Related Work

Despite the recent popularization of cloud rendering services, the idea of rendering a graphical 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 this scenario, the X Window System [5] emerged based on a client-server model, enabling clients to request basic drawing commands to the server.

More recently, researches on distributed rendering [6]–[8] 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) [9]. 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) [10]–[12] 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. In addition, they 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.

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 the reduction of the cost involved using these services [13]–[15]. 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. Similarly, in our work, we state a multi-objective workload selection problem to decide the best objects to transmit to the client. Our workload optimization differs from those because it is mostly interested in providing spatial knowledge to the user to improve his experience.

III. Hybrid Cloud Rendering

In this section, we present our hybrid cloud rendering proposal for CAD models. We discuss it through three aspects: system architecture, object representation, and final image composition.

A. System Architecture

Our system architecture comprises two entities: client and server. The server provides rendering services to the client through an internet connection or an enterprise network. In addition, all models are located on a repository on the server-side. A schematic overview of our system architecture and the data types shared between client and server is presented in Fig 2.

The communication starts when the client requests the server one of the available models. With this requisition, the client also informs the server about its computing capabilities (CPU, memory, and graphics card). After the connection establishment phase, the server determines the number of

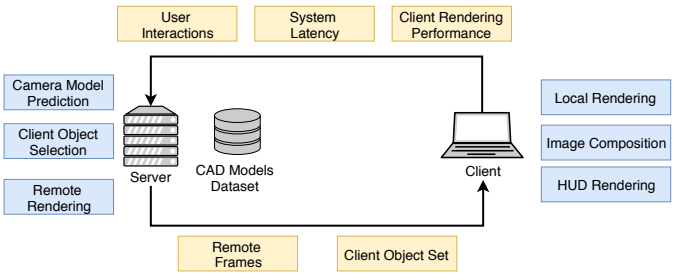


Fig. 2. A schematic representation of our cloud rendering system architecture. The yellow boxes are the type of data exchanged between client and server, while the blue boxes are the tasks performed on each side.

geometries to be rendered in the client. If that is the first time the server attends this client, the number of objects is estimated by the client hardware profile. Otherwise, the server uses the historical performance data of the client. In this moment, the server runs our workload selection method (Sec. IV) to determine which objects the client will render. Since the set of client objects is static, this is the only moment the server uses our heuristic. Besides, the selection process only consider implicit-represented objects (Sec. III-B). All HUD elements are always rendered on the client side because the image compression vastly decreases the quality of textual elements. The remaining objects are rendered on the server side. The user sees the first frames right after the first objects arrive on the client-side.

We decouple client and server operations using an asynchronous event-based communication model. Apart from the local rendering, the client is responsible for notifying the server about every user event. For example, when the user presses any navigation key, the client notifies the server about the camera navigation change. After this event, both client and server start rendering their respective object set. Due to the asynchronous communication model, we do not halt the client activity pipeline waiting for a new remote frame. Instead, the client produces the final image combining the new local frame with the last available remote frame. This whole process is detailed in Sec. III-C. Lastly, the client is frequently informing the server about its rendering performance and the latency of remote frames. The client rendering performance data is used by the server to estimate the number of objects to assign to this client in future connections.

When the server receives a notification about a user event, it starts rendering the remote objects. For each rendering pass, the server produces two images: the primary view and the depth-peeled image [16]. Afterward, the server compresses these two depth-augmented images and transmits them to the client. The server also informs its camera configurations used for producing these frames. The camera data are required to reconstruct the remote image on the client-side (Sec. III-C).

The asynchronous communication between client and server poses a challenge: if the server rendered frames using the same client camera position, the produced frames would likely be outdated when they arrive on the client side. We overcome this issue using two mechanisms: *lag compensation* and *path correction*. The lag compensation consists in using a predictive model to displace the server camera in time, putting it in a future position. The server estimates the displacement using the exponential moving average of latency information reported by the client. On the other hand, walking in a predicted future position can lead to invalid paths. This problem happens when the server camera is going along a predicted direction, but the client changes the navigation course. The path correction fixes the server camera model by moving the server camera back to the position in which the event took place. Then, the server displaces its camera again towards the new predicted direction.

B. Data representation

The massiveness of CAD models poses a challenge to achieve efficient rendering, transmission, and storage. It is mandatory to make use of efficient data representation to overcome this burden. The triangular mesh representation is widely adopted in the majority of rendering systems. Despite this representation being suitable for rendering purpose, its verbosity results in overwhelming data representation.

About 95% of industrial objects can be described by a simple set of geometries, such as planes, spheres, cylinders, and cones [17], [18]. We take advantage of this fact to describe the following objects using the implicit representation: parallelogram, spheroid, ellipsoid, cylinder, sloped cylinder, truncated cone, square frustum, rectangular toroid, and circular toroid. When compared to triangular meshes, the implicit representation is compacter, achieving up to 90% of reduction in memory usage [19]. Besides, the rendering system only needs one regular grid on the video memory to render all implicit-represented surfaces. Therefore, the implicit representation reduces the video memory requirements, and it also enables the use of instanced rendering mechanism, which improves the rendering performance from 6x to 10x [20].

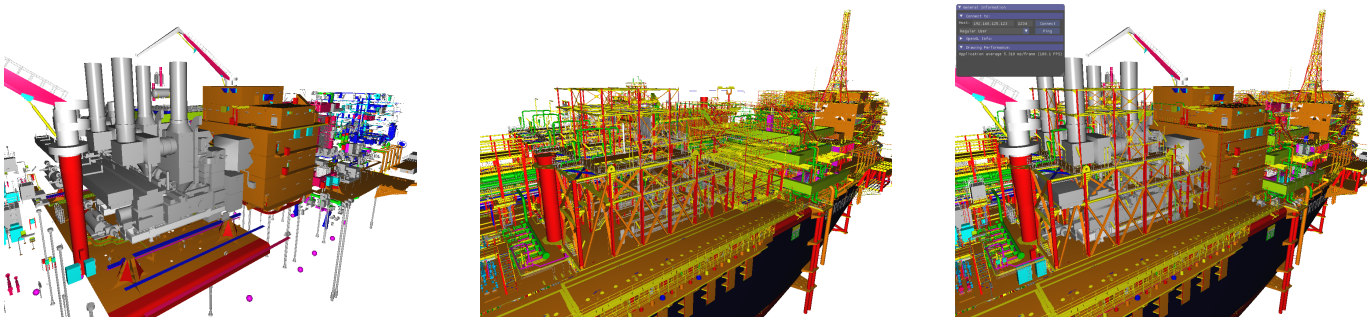
C. Final Image Composition

As mentioned, the client produces the final image by combining the local image with the last available remote frames. However, the local and remote images are unlikely to be from the same viewpoint due to asynchronous communication. Therefore, to combine both local and remote images, the client transforms the remote frame to its current viewpoint using depth-image-based rendering (DIBR) [21]. In the case the local and remote viewpoints match, it skips the DIBR step. The depth-peeled remote image is used to mitigate holes that may appear when performing the DIBR solely on the primary remote image.

The client also renders head-up display (HUD) elements, e.g., texts, over the combinations of local and remote images. This way, we avoid reducing the quality of textual elements due to image compression. The final image composition process is depicted in Fig. 3.

IV. Client Workload Selection

Rendering large models is a long-time challenge in computer graphics. Over the years, the majority of works addressed



(a) Local Image (b) Remote Image (c) Final Image

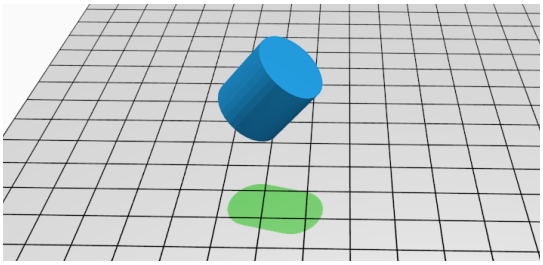
Fig. 3. Composition process of the final image. (a) The resulting image of the client-side rendering. (b) The primary image provided by the server. (c) The final image produced by the client combining (a) and (b). Besides, the client also renders the HUD elements over the final image.

this issue using model simplification. Despite the importance of these approaches, their result model representations remain costly to be rendered in thin devices. Instead, we have opted for adopting a hybrid solution. The client renders the most significant objects in the scene, according to its computing capabilities, while the server renders the rest. In this way, we always have the local image available, to be combined with the remote one. If the server is unable to deliver up-to-date frames, the client uses the most recent available, still providing a reasonable experience to the user. Besides, the most significant objects are always presented in good quality, without suffering from compression and DIBR operations.

A notable benefit from our hybrid rendering is the ability to allow the user to navigate throughout the scene even when the server stops responding. However, to achieve this goal, the client object set must afford a spatial notion to the user. Consequently, in case the server stops responding, the user can continue navigating to the desired location; and, as soon as the server responds again, the client completes its image with updated information. The selection of objects assigned to the client is then guided by an objective function that combines the selection of the most significant objects while enforcing a uniform spatial distribution of them, constrained to the client computing capabilities. As a result, our workload selection poses a multi-objective combinatorial problem: selection of the largest objects at the same time enforcing uniform distribution in the scene domain.

We assess how large the object set is by summing the volume of the oriented bounding box of each object. The spatial distribution can be evaluated using one of the several existing quantitative metrics. These metrics can be roughly categorized into two types: distance-based methods and quadrat-based methods [22]. In this work, we use the *Index of Dispersion* (ID) [23], a quadrat-based method, to evaluate the distribution of objects in the scene.

The dimension of an industrial-plant model along the ground plane is usually much larger when compared to its height. Therefore, enforcing a uniform distribution considering only this ground plane (i.e., the *XY plane*) is sufficient to grant spatial knowledge to the user. We assess the spatial distribution



(a) The projection of a cylinder onto *XY plane*. The green shadow represents the resulting projection.

0	0	0	0	0	0	1	1	0	0	0	0
0	0	0	0	0	1	1	1	0	0	0	0
0	0	0	0	0	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

(b) The resulting grid of counters after projecting the cylinder.

Fig. 4. Schematic representation of the method for assessing spatial uniformity.

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).
4) Increment the quadrat counter for each object projection that touches its region (green shadow in Fig. 4b). 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 (Eq. 1):

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

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. Our goal is to minimize this index.

Once we have set two objective functions, we use the linear scalarization method to produce a single-objective optimization problem. The linear scalarization is described in Eq. 2, where $w' > 0$ and $w'' > 0$ are the weights of the scalarization.

$$\underset{x \in X}{\text{minimize}}\, f'(x) = w' ID(x) - w'' Volume(x) \qquad (2)$$

Our optimization problem is an instance of the traditional knapsack problem: we want to select objects that most maximize the objective function without violating our knapsack capacity (the number of objects that the client can render satisfactorily). The time complexity of the knapsack problem is $\mathcal{O}(nW)$, where n is the number of objects and W is the capacity constraint value. Therefore, the knapsack is pseudo-polynomial time, that is, the algorithm is polynomial in its input value, but it is exponential in its input length. In this case, metaheuristic algorithms can often find reasonable solutions with less computational effort. In this work, we use the simulated annealing metaheuristic to obtain a feasible solution to our optimization problem.

A. Simulated Annealing

Simulated Annealing [24] is a probabilistic method inspired by the annealing process in metallurgy. Annealing is a thermal process that comprises two steps: the heating and cooling steps. The heating step increases the temperature of the heat bath to high values in order to melt solids. Next, the cooling decreases the temperature of the heat bath until the particles arrange themselves to ground state of solid.

The simulated annealing metaheuristic mimics the annealing method to find the global optimum of an objective function that may possess several local optima. The algorithm attempts to minimize the system energy by moving the system from an arbitrary initial state to another one that has the minimum energy as possible. This is achieved by assessing the energy of neighbor states. If the energy of a neighbor state is lower, the current state is replaced by the neighbor one. Nonetheless, the system can still replace its state by a worse neighbor state. This process is known as hill climbing. This way, the system prevents being stuck at local optima. Hill-climb moves are more likely to happen at high temperatures and when the difference between states is low. The parameters which control the simulated annealing process are the initial temperature, the temperature decay rate (cooling step), and the Boltzmann constant. The Boltzmann constant relates the system energy with the temperature values.

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 [25]. The area of each object in the wheel is proportional to its volume. This way, we have more chances to select objects that are likely to be in the optimal set.

Algorithm 1 Simulated Annealing Optimization

Input:
T: Initial temperature
α: Temperature decay rate
k: Boltzmann constant
function SA(T, α, k)
  $best \leftarrow$ **INIT()** ▷ Initial object set
  for #iterations **do**
    $next \leftarrow$ **NEIGHBOR**($best$) ▷ Roulette wheel
    $\Delta E \leftarrow f(next) - f(best)$
    if $\Delta E < 0$ **then**
      $best \leftarrow next$
    else if $e^{-\Delta E/KT} < random(0,1)$ **then**
      $best \leftarrow next$ ▷ Hill-climbing move
    $T \leftarrow \alpha \times T$ ▷ Temperature decay
  return $best$

TABLE I
DETAILS OF THE MODELS USED IN OUR EXPERIMENTS.

Model	Total Objects	Mesh Size (MB)	Implicit 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%

V. EVALUATION

We conducted a set of computational experiments to evaluate our system in terms of rendering performance, workload selection, and image quality. We developed the proposed system using C++ and OpenGL. We ran the experiments in a controlled environment with two computers (server/client) connected to the same enterprise network. We varied our network performance in order to simulate the different conditions of cloud services, such as high latency. The server is a desktop PC with an Intel Core i7-870 2.93GHz quad-core processor, 16GB of RAM and Nvidia GeForce GTX 980 graphics card. The client is a laptop with Intel Core i7 2.6 GHz, 4GB of RAM and Intel HD Graphics 4000.

A. System Performance

First, we executed some tests to assess how feasible is the implicit object as a means to reduce large industrial-plant CAD model representations. We used three different models that differ in the number of objects and scene complexity. Here, we name them as *small*, *medium* and *large* models (Table I). We achieved a compression ratio of up to 87% of the original model size.

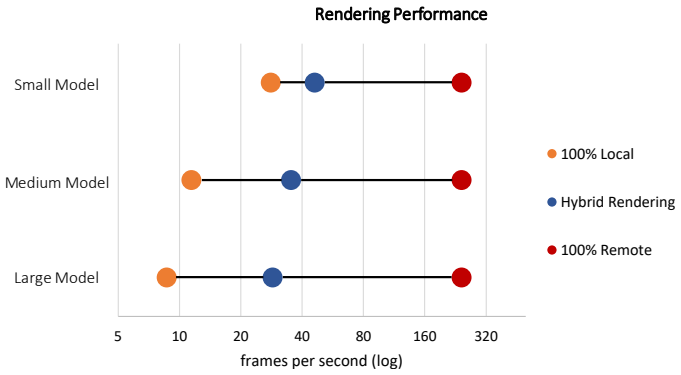


Fig. 5. Comparison of rendering performance for three different configurations: rendering entire model on the server (higher performance), rendering whole model on the client (higher image quality), and rendering on both sides.

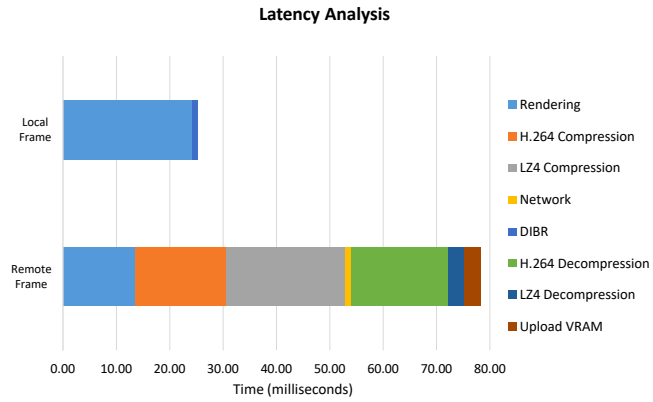


Fig. 6. The response delay for both local and remote frames be available for rendering on the client-side.

In Fig. 5, we observe the rendering performance for three different workload divisions: rendering the whole model on 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). 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 image composition. The lower the local model ratio, the higher the rendering performance, but the lower the image quality. Therefore, the location of the client performance is a trade-off between image quality and performance.

Lastly, in Fig. 6, we show the response delay of local and remote frames. 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 25 milliseconds, i.e., approximately 40 fps. Once we decouple the client-side operation from the server-side, this is also the overall response delay of our rendering system. When it comes to remote frame, the response delay is higher due to the additional steps: (de)compression, transmission, viewpoint adjusts (DIBR), and others.

B. Workload Selection

Fig. 7 shows the squared normalized results of our fitness function for 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.

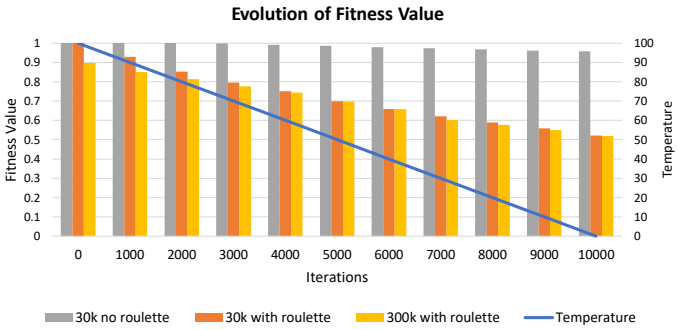


Fig. 7. The normalized fitness function of our workload optimization method for three different client profiles. We also plotted the temperature values over different iterations.

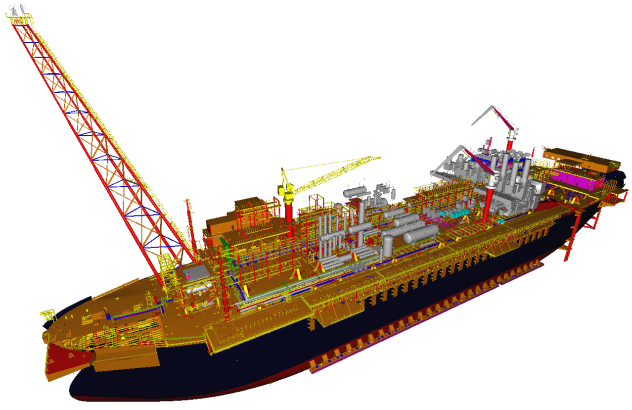


Fig. 8. The same model presented in Fig.1, but rendering approximately 27% objects of the original model.

Lastly, Fig. 8 shows the result of rendering the same model depicted in Fig. 1 but rendering only approximately 27% of objects from the original models. The majority of the missing objects are screws, valves, and other small objects. In addition, 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.

C. Image Quality

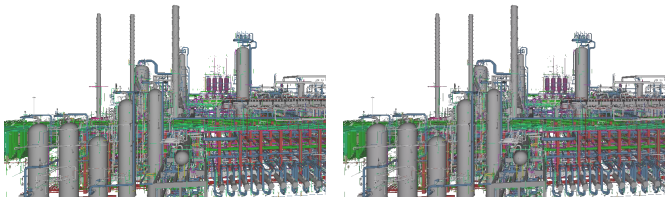
The structural similarity (SSIM) [26] index is a popular method for measuring the similarity between two images based on human perception. It serves as a quantitative measurement of the quality of one image when compared to the perfect image, i.e., the ground truth.

Three situations can degrade the quality of resulting image: image compression, remote image undersampling, and when an object that was previously outside the field of view becomes visible. For image compression, we use the H.264 codec on the server-side since it provides real-time encoding with satisfactory quality and reasonable cost-benefit ratio.

The remote undersampling occurs when the client rapidly moves closer to a remote object. At this moment, the client requests a new frame to the server, but while it is not yet available, the client composes the final image using the last available remote image. In the remote image, the object is smaller than it is from the current camera position. Consequently, the client has less available remote pixels to cover the final image, resulting in some holes in the final image. Nonetheless, we mitigate the undersampling issue by splatting [27] the remote pixels onto the final image. Finally, some holes may appear when an object that was previously out of view becomes visible since its remote image will still be produced.

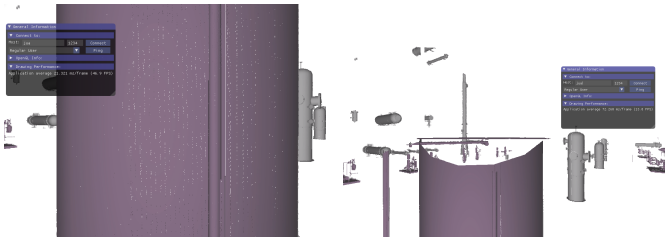
However, it is important to point out that these shortcomings only occur when the camera prediction model running on the server produces wrong results, or when the server stops responding to the client. In the case of prediction error, the server quickly fixes the camera model (path correction), as discussed in Sec. III-A. When the server stops responding, the holes are filled as soon as new remote frames arrive. Fig. 9 shows all discussed types of remote image degradation. We obtained Fig. 9c and Fig. 9d forcing the server disconnection.

We also evaluated the final image quality for different combinations of latency values and compression ratio on the remote images. Fig V-C depicts this experiment. We captured these values in a very dense spatial region, where more than 80% of the pixels in the final images belongs 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 displacement prediction model on the server-side, as a means of mitigating the network fluctuations by anticipating futures events.



(a) Ground-truth image, i.e., the raw remote image.

(b) Image (a) compressed with H.264 codec (SSIM = 98,16%).



(c) Holes due to remote image magnification.

(d) The results after objects that was previously out of vision become visible.

Fig. 9. Different types of image degradation of remote image.

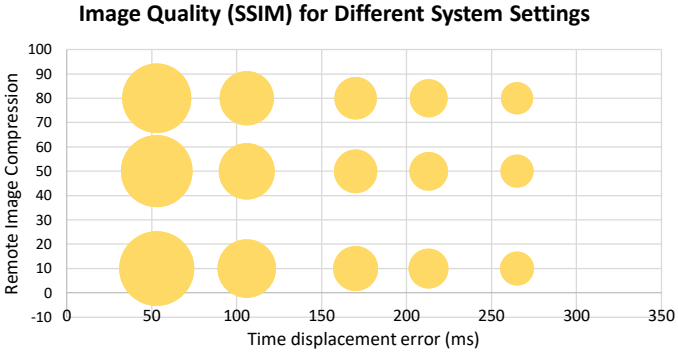


Fig. 10. The influence of time displacement error and image compression on the image quality. The bubble size is the resulting SSIM value for the given system settings.

VI. Conclusion

In this work, we highlight the challenges involved in rendering industrial plant CAD models in lightweight devices, such as tablets, smartphones, and laptops. These portables devices are increasingly used to support engineering activities. Although the cloud gaming industry is pushing the boundaries of cloud rendering, these services lack an efficient rendering mechanism for handling large and detailed models. In this work, we present a hybrid cloud rendering system aiming to fill the gap between cloud rendering services and massive CAD models.

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.

Since our system targets CAD models, it benefits from particularities from high object redundancy and the simple geometry representation to efficiently store, transmit, and render the scene.

The communication model of our client-server architecture is loosely coupled. As a result, the client application remains responsive even if the server stops. The only drawback is the image degradation since the client does not have an updated remote frame. However, the server anticipates possible future network fluctuations displacing the server camera in time.

In the sequence, we established a multi-objective optimization problem to select the objects that will be rendered on the client-side. We use the simulated annealing metaheuristic to obtain a feasible solution since this combinatorial problem has pseudo-polynomial execution time.

Our results demonstrate the viability of our solution in rendering large-scale CAD models in lightweight devices. The rendering system produces high-quality images. In case the network is not performing well, the client-side is still able to produce interactive frames rates, but slightly reducing the final image quality, as expected.

REFERENCES

[1] P. S. Dunston and X. Wang, "Mixed reality-based visualization interfaces for architecture, engineering, and construction industry," *Journal of construction engineering and management*, vol. 131, no. 12, pp. 1301–1309, 2005.

[2] H. Rivard, "A survey on the impact of information technology in the canadian architecture, engineering and construction industry." *ITcon*, vol. 5, pp. 37–56, 2000.

[3] P.-H. Chen, L. Cui, C. Wan, Q. Yang, S. K. Ting, and R. L. Tiong, "Implementation of ifc-based web server for collaborative building design between architects and structural engineers," *Automation in construction*, vol. 14, no. 1, pp. 115–128, 2005.

[4] P. Mell and T. Grance, "The nist definition of cloud computing," National Institute of Standards and Technology (NIST), Gaithersburg, MD, Tech. Rep. 800-145, September 2011. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[5] L. Mui and E. Pearce, *X Window system administrator's guide: for X version 11*. O'Reilly & Associates, Inc., 1992.

[6] F. Abraham, W. Celes, R. Cerqueira, and J. L. Campos, "A load-balancing strategy for sort-first distributed rendering," in *Proceedings. 17th Brazilian Symposium on Computer Graphics and Image Processing*. IEEE, 2004, pp. 292–299.

[7] T. A. DeFanti, D. Acevedo, R. A. Ainsworth, M. D. Brown, S. Cutchin, G. Dawe, K.-U. Doerr, A. Johnson, C. Knox, R. Kooima *et al.*, "The future of the cave," *Central European Journal of Engineering*, vol. 1, no. 1, pp. 16–37, 2011.

[8] L. Renambot, A. Rao, R. Singh, B. Jeong, N. Krishnaprasad, V. Vishwanath, V. Chandrasekhar, N. Schwarz, A. Spale, C. Zhang *et al.*, "Sage: the scalable adaptive graphics environment," in *Proceedings of WACE*, vol. 9, no. 23. Citeseer, 2004, pp. 2004–09.

[9] A. Peternier, S. Cardin, F. Vexo, and D. Thalmann, "Practical design and implementation of a cave environment," in *Proceedings 2nd International Conference on Computer Graphics Theory*, no. CONF, 2007, pp. 129–136.

[10] H. Ahmadi, M. R. Hashemi, and S. Shirmohammadi, "An open source cloud gaming testbed using directshow," in *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2015, pp. 606–610.

[11] N. M. Al-Rousan, W. Cai, H. Ji, and V. C. Leung, "Dcra: Decentralized cognitive resource allocation model for game as a service," in *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2015, pp. 218–225.

[12] M. Semsarzadeh, A. Yassine, and S. Shirmohammadi, "Video encoding acceleration in cloud gaming," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 12, pp. 1975–1987, 2015.

[13] T. Loukopoulos, N. Tziritas, M. Koziri, G. Stamoulis, and S. Khan, "A pareto-efficient algorithm for data stream processing at network edges," in *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2018, pp. 159–162.

[14] K. K. Azumah, S. Kosta, and L. T. Sørenson, "Scheduling in the hybrid cloud constrained by process mining," in *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2018, pp. 308–313.

[15] Z. Lin, Z. Wang, W. Cai, and V. C. Leung, "A novel game map preloading and resource provisioning scheme in cooperative cloud networks," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2017, pp. 210–217.

[16] B. Liu, L.-Y. Wei, Y.-Q. Xu, and E. Wu, "Multi-layer depth peeling via fragment sort," in *2009 11th IEEE International Conference on Computer-Aided Design and Computer Graphics*. IEEE, 2009, pp. 452–456.

[17] H. R. Hakala, D. G. and P. J. Malraison, "Natural quadrics in mechanical design," in *Autofact West*, vol. 1, 1980, pp. 363–378.

[18] A. A. Requicha and H. B. Voelcker, "Solid modeling: a historical summary and contemporary assessment," *IEEE computer graphics and applications*, no. 2, pp. 9–24, 1982.

[19] A. de Souza Moreira, "Engenharia reversa em modelos cad utilizando descritores de forma e maquina de vetores de suporte," Ph.D. dissertation, PUC–Rio, 2015.

[20] P. I. N. Santos and W. Celes Filho, "Instanced rendering of massive cad models using shape matching," in *2014 27th SIBGRAPI Conference on Graphics, Patterns and Images*. IEEE, 2014, pp. 335–342.

[21] C. Fehn, "Depth-image-based rendering (dibr), compression, and transmission for a new approach on 3d-tv," in *Stereoscopic Displays and Virtual Reality Systems XI*, vol. 5291. International Society for Optics and Photonics, 2004, pp. 93–104.

[22] K. M. Kam, L. Zeng, Q. Zhou, R. Tran, and J. Yang, "On assessing spatial uniformity of particle distributions in quality control of manufacturing processes," *Journal of Manufacturing Systems*, vol. 32, no. 1, pp. 154–166, 2013.

[23] J. Perry and R. Mead, "On the power of the index of dispersion test to detect spatial pattern," *Biometrics*, pp. 613–622, 1979.

[24] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[25] A. Lipowski and D. Lipowska, "Roulette-wheel selection via stochastic acceptance," *Physica A: Statistical Mechanics and its Applications*, vol. 391, no. 6, pp. 2193–2196, 2012.

[26] Z. Wang, A. C. Bovik, H. R. Sheikh, E. P. Simoncelli *et al.*, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.

[27] W. R. Mark and G. Bishop, "Post-rendering 3 d image warping: visibility, reconstruction, and performance for depth-image warping," Ph.D. dissertation, University of North Carolina at Chapel Hill, 1999.