Efficient Delivery of 3D Scenes via Layered 3D Gaussian Splats

ABSTRACT

Traditional representations of 3D content consist of dense point clouds that consume large amounts of data and network bandwidth, while newer representations such as neural radiance fields (NeRF) suffer from poor frame rates due to a non-standard volumetric rendering pipeline. 3D Gaussian splats (3DGS) can be seen as a generalization of point clouds and meet the best of both worlds, with high visual quality and efficient rendering for real-time frame rates. However, it is unclear how to deliver 3DGS scenes from a server hosting 3DGS scenes to client devices. The goal of this work is to create an efficient and high-fidelity 3D content delivery framework that enables users to view 3D scenes with 3DGS as the underlying data representation. The main contributions of the paper are: (1) Creating new layered 3DGS models for efficient delivery, (2) Scheduling algorithms to choose what splats to download at what time, and (3) Trace-driven experiments from users wearing virtual reality headsets to evaluate the visual quality and latency. The results demonstrate high visual quality, achieving 16.9% higher average SSIM compared to a sorting baseline inspired by the literature.

ACM Reference Format:

. 2024. Efficient Delivery of 3D Scenes via Layered 3D Gaussian Splats. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Traditional representations of 3D content include meshes and point clouds. Recently, new technologies to model 3D scenes have emerged that outperform traditional representations in terms of realism and modeling capability, such as neural radiance fields (NeRF [19]) and 3D Gaussian splats (3DGS [13]). NeRF requires substantial training of machine learning models to represent the 3D scene and relies on slow volumetric rendering techniques. 3D Gaussian splats, introduced in 2023, can be seen as a generalization of point clouds, where each 3D "splat" has position, volume, and color features, and have gained prominence for their real-time rendering capabilities and excellent visual quality.

Typically, these 3D scenes are created and stored on a server, due to the substantial amount of computation needed to create the 3D scenes. Clients seeking to view the 3D scenes can then download these models and render them locally for viewing. This creates several network delivery challenges for viewers of 3DGS scenes: (1) The 3DGS scenes can be very large (*e.g.,* 1.52 GB for the bicycle scene from Mip-NeRF360 [3]), and downloading the entire scene before viewing will create long delay for the viewer. (2) 3DGS scenes are comprised of a large number of splats (700k to 1M splats in standard datasets), which are of varying importance to the users, so it's not clear which splats should be prioritized for delivery to the client. It's also not scalable to make prioritization decisions for individual splats. (3) 3D scenes, unlike 3D objects, immersively surround the user. Users have a full 6 degrees of freedom to walk around and view different parts of the scene from different angles, making it difficult to determine what parts of the scene to deliver to users.

In this work, we design an efficient and high-fidelity delivery framework for 3D scenes using 3D Gaussians as the underlying data representation. Our framework addresses the above challenges as follows. (1) Leveraging the unique structure of the Gaussian splats, we design a custom training scheme that produces *layered* 3DGS. This representation enables a "base layer" of splats to be displayed first, followed by additional "enhancement layers" on top. This enables progressive download of different parts of the scene based on network bandwidth. (2) To enable scalable selection of the most important splats, we segment the 3D scene into objects, where each object is a set of splats. This segmentation also enables interactive editing of the 3D scene, where the user can move objects around to their liking. (3) We collected traces of users wearing virtual reality headsets (Meta Quest 3) and moving around standard 3DGS scenes. This powers a user prediction module to determine what splats are likely to be important to the user and require priority delivery.

Overall, the contributions of the paper are:

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM

https://doi.org/10.1145/nnnnnnn.nnnnnnn

- Custom training to create layered 3D Gaussian models to represent 3D scenes. Objects are segmented in the scene to provide fine-grained control for downloading and editing.
- Scheduling methods to choose the right sets of splats to download maximize visual quality, based on predictions of the user's viewport and network bandwidth. The scheduler works with other types of splat representations too.
- Experiments to measure visual quality and latency, driven by traces that we collected of users exploring 3DGS scenes using VR headsets.

The paper is organized as follows. Section 2 discusses the background and related work. Section 3 describes the overall system design and the individual modules. Section 4 shows the experimental results and we conclude in Section 5. Due to space constraints, more results are available in an anonymous technical report [1].

2 BACKGROUND AND RELATED WORK

3D representations. Traditional 3D representations such as point clouds and meshes usually represent 3D scenes explicitly. ***Point clouds*** represent 3D scenes using a set of points in 3D space. It can also include other features like colors to better represent the scene. ***3D meshes*** use vertices, edges and faces to represent 3D objects. These vertices can also have feature vectors associated with them. Despite many efforts to efficiently stream such representations [10, 11, 17], they do not leverage the latest advances in 3D representations.

Emerging 3D representations such as NeRF and 3DGS can more accurately represent 3D scenes. ***NeRF*** [19] represents 3D scenes as a continuous volumetric field using a multilayer perceptron (MLP) [2]. ***3DGS*** [13] represents 3D scenes as a set of 3D Gaussian splats. Each splat is represented by a set of attributes including its position, opacity, a covariance matrix containing size and rotation information, and Spherical Harmonic coefficients representing view-dependent color. When rendered, these 3D Gaussian splats will be projected into 2D camera coordinates and a tile-based rasterizer will be used for color computation.

3DGS training. To train a set of splats to represent a 3D scene [13], the splats will be initialized with a sparse point cloud produced by Structure-from-Motion (SfM) methods. The training proceeds in iterations, where in each iteration adjusts the values of the splat attributes (color, position, radius, etc.) to create rendered images that match the ground truth. At the same time, densification and pruning processes are used to improve the overall quality of the 3DGS model and ensure that splats are created in the right places to accurately represent the 3D scene. Specifically, to cover the geometry in under-reconstructed regions (regions with too few splats), the training algorithm will ***clone*** the splats in the region by simply creating a copy of the splats. Also, because larger splats in visually complex regions may not be able to capture all the details, the training algorithm will ***split*** a splat into smaller splats (replace a splat with two new ones). The training algorithm will also ***prune*** unimportant splats (*e.g.,* splats with too low opacity) to keep the scene a reasonable size.

NeRF and 3DGS optimizations for efficient delivery. Although NeRF and 3DGS have high visual quality, they still face problems that impede their efficient delivery. NeRF suffers from large model size and slow rendering [20], while 3DGS faces the challenges of large model size [22, 26]. Compression techniques are needed to decrease model size for efficient network delivery and rendering. For NeRF compression, efforts usually focus on decreasing the size of the MLP [4, 8, 20]. Standard compression techniques such as quantization, pruning and knowledge distillation can also be used for NeRF compression [6] and 3DGS compression [22]. For 3DGS, LightGaussians [9] uses knowledge distillation, pseudo-view augmentation and global significance scores to compress 3D scenes. To eliminate structural redundancies and further compress 3DGS, Scaffold-GS [18] used anchors to cluster 3DGS. HAC [7] further introduced a hash grid to make 3DGS representation more compact. Based on octrees, Octree-GS [24] anchors Gaussians with different level-of-details to improve the rendering performance of 3DGS. Mip-Splatting [33] uses a 3D smoothing filter and a 2D Mip filter to improve rendering quality. These works mainly focus on non-layered representations and do not consider their network delivery. Further, we demonstrate that our framework can work alongside versions of [7, 14, 27] in Section 4.

Multimedia delivery. Layering and viewport prediction are two important techniques that can be used to improve the efficiency of multimedia streaming. Layering can improve streaming efficiency by making the streaming more adaptive. Scalable Video Coding (SVC) [25] encodes video stream in layered structure, where a base layer provides a minimum quality level, and enhancement layers improve the resolution, frame rate, and quality. Our framework is for 3D scenes, not 2D videos. Viewport prediction makes streaming more efficient by only fetching parts of the scene that include what a user is about to view. Regression, machine learning, and video saliency features are commonly used techniques for viewport prediction [16, 23, 29]. In our work, we focus on lightweight 6 DoF viewport prediction where we have to predict not only orientation but position in a 3D scene, particularly for the standard 3DGS scenes for which user traces and viewport prediction has not been well studied.

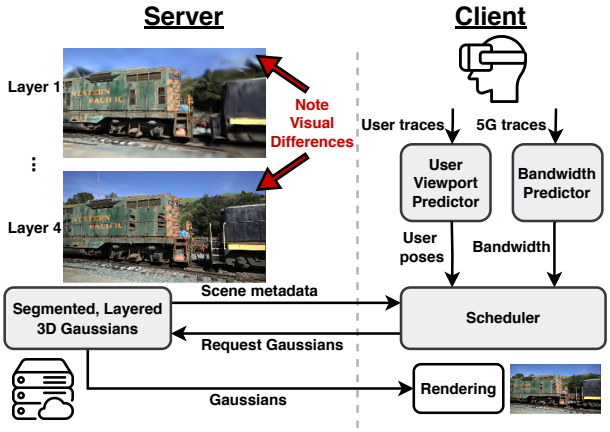


Figure 1: System architecture. Given a set of layered and segmented 3D Gaussian splats, our system retrieves the most useful splats within the user's predicted viewport and network bandwidth.

3 SYSTEM DESIGN

The overview architecture of our system is presented in Figure 1. Given a 3D scene comprised of 3DGS, our system decides what are the best splats to retrieve in order to render the content in the user's viewport while respecting the estimated network bandwidth. To accomplish this, there are four components in the system:

- **Segmented, layered 3D Gaussians (Section 3.1)**. To provide users with a progressive quality-enhancing experience, we create 3D Gaussian scenes with layers, including a base layer and several enhancement layers. Further, we also have layered complex scenes that are automatically segmented into semantically meaningful objects to enable scene editing.
- **Splat download scheduler (Section 3.2).** Given a 3D Gaussian scene, our scheduler determines the retrieval order of each Gaussian splat based on the utility values for each segmented object and each layer, plus the available network bandwidth. The utility values are computed over a time window that accounts for the user's estimated viewport movements, so that splats within and closer to the predicted viewport for a longer period yield a higher utility. We formally define the optimization problems for various cases (layered, non-layered, segmented, non-segmented), prove the NP-hardness of the main cases, and propose pseudo-polynomial time algorithms to solve them.
- **User viewport predictor (Section 3.3).** We collect our own traces of users' 6-DoF movements around standard 3DGS scenes while wearing VR headsets (Meta Quest 3). To predict the user viewport, we use linear regression that uses the history of the user's viewport to make online predictions.
- **Bandwidth predictor (Section 3.4).** We use outdoor 5G users' walking traces [21] dataset to simulate the highly variable 5G network bandwidth. To estimate the available network bandwidth as input to the Scheduler, borrow from existing methods [32].

3.1 Segmented, Layered 3D Gaussians

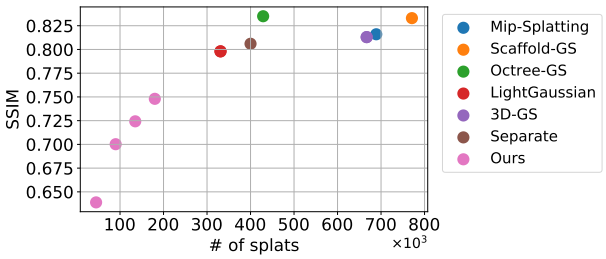


Figure 2: Our approaches ("Ours" and "Separate") can gracefully trade off visual quality for the number of splats. Data from the "Train" scene [15].

A key component of our framework is the layered 3DGS representation. When creating layered splats, we need to balance between the number of splats (which is directly proportional to model size) and the visual quality (in terms of SSIM). Figure 2 presents this tradeoff for existing methods, alongside our proposed methods (to be discussed below). With default pre-trained 3DGS models [13], although they have high visual quality, each scene can contain anywhere from 650K to more than 5M Gaussian splats, which is challenging for network delivery due to the large size (*e.g.,* 1.52 GB for the bicycle scene from Mip-NeRF360 [3]). This is represented by "3D-GS" for a specific 3D scene in Figure 2. Existing 3DGS methods (Mip-Splatting [33], Scaffold-GS [18], Octree-GS [24], LightGaussian [9]) can achieve similarly high visual quality (SSIM $\geq$ 0.8), but still require at least 300k splats. In contrast, our goal is to create splats that achieve a graceful tradeoff between visual quality and number of splats, represented by "Ours" and "Separate" in the figure.

Algorithm 1 The overall pipeline of Layered Model Training

Input: Pretrained 3D-GS: $\mathcal{G}$, Target size $\mathcal{D} = \{d_l\}_{l=1}^{L}$
Output: Layered 3D-GS $\mathcal{LG}_1, ..., \mathcal{LG}_L$
1: $\mathcal{G}_L = \{G_i\}_{i=1}^{d_l} \leftarrow$ PRUNE2TARGETSIZE($\mathcal{G}, d_L$) ▷ Algorithm 2
2: $\mathcal{LG}_1, ..., \mathcal{LG}_L \leftarrow$ PROGRESSIVETRAINING($\mathcal{G}_L, \mathcal{D}$) ▷ Algorithm 3

The overall procedure of training these layered 3D Gaussians splats is summarized in Algorithm 1 and Figure 3. There are two main steps in the example shown, where we want to

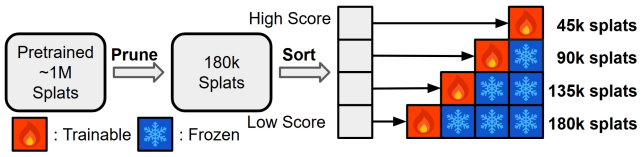


Figure 3: Training framework for layered 3DGS. The layers are iteratively trained, with subsequent layers relying on frozen splats from preceding layers.

create a scene with 180k total splats, split into layers of 45k splats each. First, we create an initial scene with a controlled number of splats by pruning from the default pretrained model down to the total target number, 180k in the example (left hand side of Figure 3). Second, we take the 180k splats and split them into layers, 45k splats each in the example, and train them in a layered fashion (right hand side of Figure 3). Below, we delve into these steps in more detail.

(1) Creating an initial scene with a controlled total number of splats. To create splats that achieve this tradeoff, we need to be able to control the total number of splats in the initial 180k model. In the original 3DGS training pipeline, the number of splats is indirectly influenced by a set of hyperparameters, such as τ_p for cloning and splitting decisions, or the densification interval. However, since these hyperparameters interact with each other in complex ways to produce the final output, the mapping between the set of hyperparameter values and total number of splats is unclear. Furthermore, the mapping could change for different 3D scenes being created.

Therefore, to effectively control the model size, we design Algorithm 2 to obtain a layer of precisely the desired size while achieving good visual quality. We start with the default pre-trained model from [13] (line 1), and prune down to the target number of splats from there. This is a top-down approach; the basic idea is to repeatedly prune the number of splats (which decreases the visual quality), and then grow (which improves the visual quality) multiple times. Note that we also experimented with a bottom-up approach where we grew the number of splats to the target size, but found top-down approach worked better in practice.

Specifically, if the number of splats is much larger than the target d, we PRUNE the splats down to remove the worst r fraction of splats, according to their score [9] (line 6). After pruning, we will run RECOVERY to recover the visual quality by training the splat parameters, according to a regular training iteration that allows cloning, splitting, and pruning (line 7). This might cause the number of splats to increase again, so we repeat the pruning and recovery process multiple times. For the final pruning, to ensure the number of splats in the final model exactly matches the target, we FINETUNE the splat attributes according to a regular training iteration but without allowing cloning, splitting, or pruning (line 11). After the desired number of splats are obtained, we also sort the splats by their score to prepare for the next step, layered model training (line 15).

Algorithm 2 PRUNE2TARGETSIZE

Input: Pretrained 3DGS: $\mathcal{G}$, Target size d, Max pruning ratio r, Training view images $\mathcal{I}$, and their associated camera poses $\mathcal{P}$
Output: pruned and sorted 3DGS $\mathcal{G}_L = \{G_i\}_{i=1}^{d}$

1: Initialize 3D-GS $\mathcal{G}^* \leftarrow \mathcal{G}$ ▷ $\mathcal{G}$ pretrained by [13]
2: n = NUMOFGS($\mathcal{G}^*$) ▷ Compute the number of splats
3: **while** $n > d$ **do**
4: $\mathcal{S} \leftarrow$ CALGS($\mathcal{G}^*, \mathcal{I}, \mathcal{P}$) ▷ Compute global significance score [9]
5: **if** $n * (1 - r) > d$ **then**
6: $\mathcal{G}^* \leftarrow$ PRUNE($\mathcal{G}^*, \mathcal{GS}, n * r$) ▷ Prune r faction of splats with lowest $\mathcal{S}$
7: $\mathcal{G}^* \leftarrow$ RECOVERY($\mathcal{G}^*, \mathcal{I}, \mathcal{P}$, IsRefinementIteration = True) ▷ Training method [13] with prune, split, and clone
8: n = NUMOFGS($\mathcal{G}^*$)
9: **else**
10: $\mathcal{G}^* \leftarrow$ PRUNE($\mathcal{G}^*, \mathcal{GS}, n - d$)
11: $\mathcal{G}^* \leftarrow$ FINETUNE($\mathcal{G}^*, \mathcal{I}, \mathcal{P}$, IsRefinementIteration=False) ▷ Training method [13] without prune, split and clone.
12: **end if**
13: **end while**
14: $\mathcal{S} \leftarrow$ CALGS($\mathcal{G}^*, \mathcal{I}, \mathcal{P}$)
15: $\mathcal{G}_L \leftarrow$ SORT($\mathcal{G}^*, \mathcal{S}$) ▷ Sort splats by their global significance score in descending order.

(2) Layered model training. In order to support efficient 3DGS delivery, we need to create layered splats that can overlay on top of one another to increase visual quality. A naive strategy to create a layered structure is to create a combined loss function and jointly train splats in the L layers simultaneously. For example, if there are $L = 2$ layers, the first model would be trained with loss function ℓ_1 (corresponding to splats in layer 1), and the second model trained simultaneously with loss function $\ell_1 + \ell_2$ (corresponding to splats in layer 1 and layer 2). However, we found with this technique, the splats in layer 1 would effectively get weighted twice in every training iteration compared to layer 2, leading to instability in the layer 1 splats and poor performance.

To overcome this, we had to design a custom training procedure instead of simply modifying the training loss function, given in Algorithm 3. The main idea, as shown in the right hand side of Figure 3, is to incrementally train the models from the lower to higher layers. To ensure the higher layers

can overlay on the preceding lower layers, when training each layer, we freeze the lower layers and only train on the added splats. In the example in Figure 3, we first train layer 1 (45k splats in the figure), freeze layer 1, add on another 45k splats for layer 2 and train them, and so on.

Specifically, we first obtain the splats from Algorithm 2 and split its splats into L layers, denoted as $\Delta\mathcal{LG}_l$ (line 1). We train the smallest layer $\mathcal{LG}_1$ using the FINETUNE function, which doesn't change the number of splats (line 2). For subsequent layers, we freeze the splats from the preceding layers (line 5) and only allow training for the current layer's newly added splats (line 6), training them using FINETUNE, which again doesn't change the number of splats. This continues until all the enhancement layers have been trained. Note that in each training iteration, the loss function is computed based on rendering all splats in a given layer, but only the latest enhancement layer's splats are allowed to be updated to reduce the loss. This enables the layered effect.

Algorithm 3 PROGRESSIVETRAINING

Input: Pruned and sorted 3DGS: $\mathcal{G}_L = \{G_i\}_{i=1}^{d_L}$, Target sizes $\mathcal{D} = \{d_1, d_2, \dots d_L\}$
Output: Layered 3D-GS $\mathcal{LG}_1, ..., \mathcal{LG}_L$
1: $\mathcal{LG}_1, \Delta\mathcal{LG}_2, ..., \Delta\mathcal{LG}_L \leftarrow \{G_i\}_1^{d_{i=1}}, \{G_i\}_{i=d_1+1}^{d_2}, \dots, \{G_i\}_{i=d_{L-1}+1}^{d_L}$ ▷ $\Delta\mathcal{LG}_l$ denotes $\mathcal{LG}_l \setminus \mathcal{LG}_{l-1}$
2: $\mathcal{LG}_1 \leftarrow$ FINETUNE($\mathcal{LG}_1, \mathcal{I}, \mathcal{P}$,IsRefinementIteration=False) ▷ Train smallest layer
3: **for** $l \leftarrow 2$ to L **do**
4: $\quad\mathcal{LG}_l = \{G_i\}_{i=1}^{d_l} \leftarrow$ CONCATENATE($\mathcal{LG}_{l-1}, \Delta\mathcal{LG}_l$)
5: $\quad\{G_i\}_1^{d_{l-1}}$.requires_grad_(false) ▷ Freeze splats from previous layer
6: $\quad\{G_i\}_{d_{l-1}+1}^{d_l}$.requires_grad_(true) ▷ Newly added splats are trainable
7: $\quad\mathcal{LG}_l \leftarrow$ FINETUNE($\mathcal{LG}_l, \mathcal{I}, \mathcal{P}, IsRefinementIteration = False$) ▷ Train
8: **end for**

Segmentation. Segmenting the 3D scene into multiple objects turns out to be useful, in order to cluster splats together into semantically meaningful objects and do more efficient scheduling later in Section 3.2. To implement segmentation in our 3DGS models, we add the object ID as an additional feature to each splat in Algorithm 1, assigning the initial object IDs according to [30]. These object IDs are refined throughout the training process described above. Although the total number of splats in each layer d_l is fixed, we did not constrain the object ID during training, so the number of splats assigned to each object is non-uniform in a given layer. In other words, more or less splats can be automatically allocated to different objects by the training process to achieve the best visual quality.

$B[t]$	predicted bandwidth in time slot t
c_{jl}	For non-layered splats, the cost of object j of version l.
Δc_{jl}	For layered splats, the cost of object j of layer l. $\Delta c_{jl}[t] = c_{jl}[t] - c_{j,l-1}[t]$
d_l	Target number of splats for layer l
$\mathcal{G}, \mathcal{LG} = \{G_i\}$	entire set of splats in scene
hw	length of history time window (s)
pw	length of prediction time window (s)
N	number of layers per scene
M_{jl}	number of splats for object j in layer l
T	duration of time slot
$U_{jl}[t]$	For non-layered splats, the utility of splat j of version l at time t.
$\Delta U_{jl}[t]$	For layered splats, the utility of splat j from layer l at time t. See (2).
$\tilde{U}_{jl}[t]$	utility of splat j from layer l from time t onwards. See (9).
$x_{jl}[t]$	decision variable of whether to download object j from layer l at time t
$y_{jl}[t]$	binary indicator of whether splat j from layer l is stored at time t

Table 1: Table of Notation.

3.2 Splat Download Scheduler

Given the layered splats from Section 3.1, the goal of the scheduler is to determine what splats to download in what order. This section describes the problem setup, including the splat utility definition, problem formulation, and scheduling algorithms. Due to space constraints, proofs and some problem definitions are provided in the Appendix B.

3.2.1 Problem Setup. As discussed in Section 3.1, a scene is comprised of splats encoded into different layers. An enhancement layer of an object is only useful if the base layer and all lower enhancement layers were previously downloaded. Optionally, the scene can be segmented into objects, and multiple splats belong to a single object. Notation-wise, this means each splat has a layer ID l and optionally an object ID j. Time is divided into slots of duration T indexed by t. Given the bandwidth $B[t]$ over time, the goal is to select which splats to download that maximize the total utility while staying under the available bandwidth. We denote the main decision variable $x_{jl}[t]$ as whether to download the splats comprising object j at layer l at time slot t. The scenario of layered splats with the scene segmented into objects is the main scenario we consider, because it flexibly allows for different layer selection for different objects. There are several alternative scenarios, depending on whether the splats at layered (or not) and whether the scene is segmented into objects (or not). Our scheduler is designed to work with

	segmented objects	not segmented
layered splats	**case I** (Section 3.2.2); NP-hard (Theorem 1); Solved by knapsack.	**case III** (Section 3.2.4); NP-hard (Theorem 1); Solved by knapsack.
separate splats	**case II** (Section 3.2.3); NP-hard (Theorem 2); Solved by progressive loading.	**case IV** (Section 3.2.4); NP-hard (Theorem 2); Solved by progressive loading.

Table 2: Taxonomy of problems and algorithms for different 3D Gaussian splat representations.

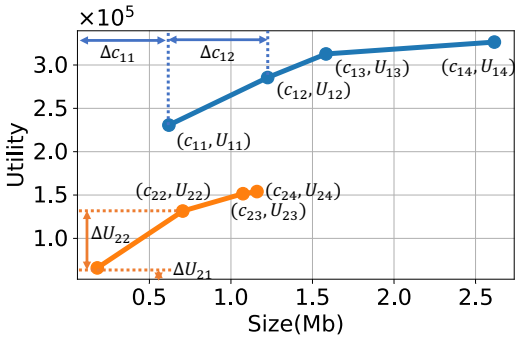


Figure 4: Illustration of the utility function (1) for two objects (egg and pork belly) from the "Ramen" scene [30].

all these scenarios, which are summarized in Table 2. The mathematical notation is summarized in Table 1.

Splat utility definition. We need to assign a numerical utility to each splat to determine its value to the scene and whether its download should be prioritized in the scheduler. However, it is impossible to assign a utility to an individual splat because it is just one component of the image; typical methods of estimating visual quality like PSNR or SSIM require an entire image (similarly, calculating the SSIM of a 3D point in a point cloud is ill-defined). Therefore, we design an estimated utility of each splat for use in the scheduler. The utility function of layer l of object at time t is:

$$U_{jl}[t] = closeness_{jl}[t] \times overlap_{jl}[t] \times opacity_{jl} \quad (1)$$

The "closeness" measures the 3D Euclidean distance between the center of the splat to the center of the user's viewport. The "overlap" is the 2D area of the 3D splat when projected onto the user's 2D viewport. The "opacity" is simply the opacity of the splat, a standard feature stored within the splat data structure. Note that closeness and overlap depend on the user's viewport at time t, while opacity is a static property. In other words, the utility captures the user's movements as time passes with the $overlap_{jl}[t]$ component. When the user remains stationary, this component weighs the same set of splats more than the periphery regions outside the viewport. Note that this utility function is different than the global significance score [9] used when training the layered splats in Algorithm 2; that score is viewport-independent because it is used to general a general set of splats that could work with any viewport trace. We experiment with a version of [9]'s score as a baseline in Section 4.

It's also useful to define the utility difference from the previous layer as:

$$\Delta U_{jl}[t] = U_{jl}[t] - U_{j,l-1}[t] \quad (2)$$

An empirical example of the utility function for a single splat at time t is shown in Figure 4. Note that U_{jl} and ΔU_{jl} are used only by the scheduler to determine the utility of individual splats; the final evaluation in Section 4 is done using overall scene SSIM in accordance with standard practice.

3.2.2 Case I: Layered splats with segmented objects. This is the main scenario we consider in this paper, which utilizes our specially trained layered splats representation (Section 3.1) with semantic segmentation into objects. The scheduling problem is defined as follows.

Problem 1. *Layered splats with time-varying viewport and bandwidth*

$$\max_{x,y} \sum_j \sum_l \sum_t \Delta U_{jl}[t] y_{jl}[t] \quad (3)$$

$$s.t. \sum_j \sum_l \Delta c_{jl} x_{jl}[t] \le B[t] \quad \forall t \quad (4)$$

$$\sum_t x_{jl}[t] \le 1 \quad \forall j, l \quad (5)$$

$$y_{jl} = \sum_{t'<t} x_{jl}[t'] \quad \forall j, t, l \quad (6)$$

$$x_{jl}[t] \le y_{j,l-1}[t] \quad \forall j, l, t \quad (7)$$

$$x_{jl}[t], y_{jl}[t] \in \{0, 1\} \quad \forall j, t \quad (8)$$

The main variable $x_{jl}[t]$ is 1 if layer l of object j is downloaded in time slot t, and 0 otherwise. The objective (3) is to maximize the total utility across all splats in the viewport across all time. Constraint (4) states that the total size of the downloaded splats in each time slot must not exceed the available predicted bandwidth. Constraint (5) states that each layer of each object can be downloaded only once over the entire duration of the user trace. Constraint (6) keeps track of previously downloaded splats in the helper variable $y_{jl}[t]$, which is 1 if layer l of object j has been downloaded before time slot t, and 0 otherwise. In other words, $x_{jl}[t]$ is 1 when the splat is downloaded, and the corresponding $y_{jl}[t]$ is 1 thereafter. Constraint (7) states that layer l can only be

downloaded if the preceding layer $l-1$ has previously been downloaded.

To understand the difficulty of Problem 1, we first transform it into an equivalent Problem 3 according to Lemma 1.

Lemma 1. *Problem 3 is equivalent to Problem 1.*

The transformation is done by defining a new utility cumulative $\Delta\tilde{U}_{jl}[t]$ as

$$\Delta\tilde{U}_{jl}[t] \equiv \sum_{t' \geq t} \Delta U_{jl}[t], \tag{9}$$

summing up the utility of layer l of object j for a user from time t onward. Intuitively, it aggregates the total future utility, from time t onwards, of a user for an object based on the user's movements around the scene. The new Problem 3 (complete description in the technical report [1]) is similar to a generalized assignment problem, except it has the wrinkle of an additional constraint (18) that states that layer l can only be displayed if the previous $l-1, l-2, \ldots, 1$ layers have already been retrieved. This adds a complication to the already NP-hard generalized assignment problem. However, one subtlety is that adding another constraint to the generalized assignment problem does not necessarily mean that it is also NP-hard; constraining the feasible set could potentially make it easier to solve the problem. Therefore, we require Theorem 1 to state that our precedence-constrained generalized assignment Problem 3, and hence the original Problem 1, is NP-hard. The proof relies on a reduction from GAP to Problem 3.

Theorem 1. *Problem 1 is NP-Hard.*

Because Problem 1 is NP-hard, we turn our attention to heuristic solutions. In the simple case where $B[t]$ and $U_{jl}[t]$ are static over time, it turns out that a greedy algorithm (sort the utility functions by their slopes, and pick the best slope each time) is optimal; however, this static bandwidth and user viewport are unlikely to occur in practice. More realistically, bandwidth and user prediction far into the future is difficult. We therefore focus on one individual time slot and optimize within that. This turns out to be a multiple choice knapsack problem, which can be solved in pseudo-polynomial time via dynamic programming. Lemma 2 shows the equivalency of Problem 1 to a knapsack problem within a given time slot.

Lemma 2. *Within a single time slot, Problem 1 is a multiple choice knapsack problem.*

3.2.3 Case II: Separate splats with segmented objects. Next consider Case II, when the splats are still segmented into different objects but not layered. Instead of calling them layers, we call the different qualities of 3D scenes as separate "versions". We need to choose the right version for each object, and subsequent better quality versions replace prior low quality versions. We consider this non-layered scenario because recent work in the computer vision community design new compression schemes for non-layered splats, and we would like our framework to be able to work with them (see Section 4.5). The main difference from Problem 1 is we now consider the *maximum* utility across layers for a given object (rather than summation) in the objective function (10), because subsequent versions replace earlier ones rather than adding onto them as in the layered approach. The other difference is the lack of precedence constraints, as each layer is downloaded independently, so no version of (7) is needed in Problem 2 below.

Problem 2. *Case II: Non-layered splats with time-varying viewport and bandwidth*

$$\max \sum_j \sum_t \max_l U_{jl}[t] y_{jl}[t] \tag{10}$$

$$s.t. \sum_j \sum_l c_{jl} x_{jl}[t] \leq B[t] \quad \forall t \tag{11}$$

$$\sum_t x_{jl}[t] \leq 1 \quad \forall j, l \tag{12}$$

$$y_{jl} = \sum_{t'<t} x_{jl}[t'] \quad \forall j, t, l \tag{13}$$

$$x_{jl}[t] \in \{0, 1\} \quad \forall j, t \tag{14}$$

It turns out that this problem is also NP-hard according to Theorem 2, according to Theorem 2, so in practice we solve it by simply loading the versions in sequence, starting version $0, \ldots, L$, similar to a progressive JPEG image.

Theorem 2. *Problem 2 is NP-hard.*

3.2.4 Cases III and IV. We note that case III, layered splats without object segmentation, has some similarities to Case I and is covered by Problem 1 and hence its knapsack algorithm, as follows. Without segmented objects in the scene to choose from, the decision is what layer of the entire scene to download. In our framework, for case III, the decision variable $x_{jl}[t]$ represents the choice of whether to download layer l of the entire scene, and the index j is moot. Once the layer is chosen, the splats within the layer are downloaded in order according to their score (1).

For Case IV, the 3D scene without object segmentation with separate versions of the scene, can be covered by Problem 2 following similar arguments as the previous paragraph. The representation is similar to using existing splat compression methods [7, 9] and tuning their parameters to create different versions of the same scene. We consider this as a baseline in the Evaluation.

3.3 User Viewport Predictor

Trace collection and generation. We collected real viewport traces from 6 users for 8 scenes. The viewport trajectory data is recorded as a time series using the Meta Quest 3 VR headset's inside-out tracking with six degrees of freedom (6DoF). This 6DoF tracking system can track rotational and positional movements along the x, y, and z axes. The sampling rate is set to 36Hz. Examples of the viewport trajectory data are shown in fig. 5 for 6 users in different colors. Additionally, 6 synthetic traces are generated, including elliptical, circular, and spiral paths, and three other paths derived from the original test sets of the 3DGS datasets.

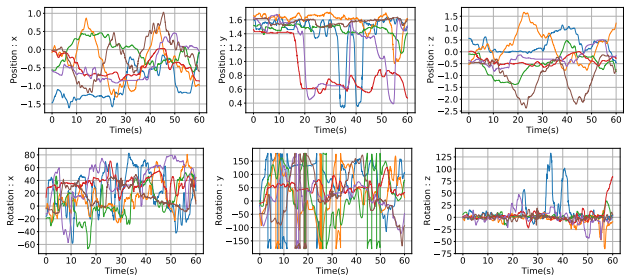


Figure 5: Real viewport trajectory data from the bicycle scene, each color representing a different user.

Prediction model The linear regression model is used in several viewport-adaptive systems due to its simplicity, speed, and reasonable accuracy in short-term forecasting[12, 23, 29]. We employ the linear regression model to make online predictions. Given the recent viewports within a history window (hw), we apply the model to predict the subsequent viewports within a prediction window (pw). During runtime, the model is continuously fitted to the data points from the recent history window (hw) and then used to forecast future positions in the prediction window (pw). The six features of 6DoF are predicted separately. We treat the value of each feature as the dependent variable and its relative order in the sequence, compared to the first point in hw, as the independent variable. A key challenge arises from the cyclical nature of angles, where 0 degrees is equivalent to 360 degrees. To address this, we recenter the data before each prediction to ensure the model accurately adapts to this angular continuity. We set the first point in hw as 0. Using it as the reference, angles increase in the clockwise direction and decrease in the counterclockwise direction, resulting in a final range of [-180, 180]. In practice, we set $hw = 0.5s$ and $pw = 1s$.

3.4 Bandwidth Predictor

Our system makes downloading decisions based on available network bandwidth, utilizing the existing throughput predictor [32]. Specifically, we used harmonic means with a history window (hw) of size 5 seconds to predict the next second (pw). To simulate the outdoor 5G network, we sampled outdoor user walking traces from [21], scaling down the bandwidth to adapt to our use case as has been done in other work [31]. The scaled average network bandwidth is 11.8 Mbps with a standard deviation 2.9.

4 EXPERIMENTAL RESULTS

4.1 Setup

We conduct evaluations on 8 scenes from public datasets, including 3 scenes from Mip-NeRF360 [3], 2 scenes from Tanks&Temples [15], and 3 segmented scenes from Gaussian Grouping [30]. We create each scene with 4 layers, each layer having 45k splats more than the previous layer. To evaluate our delivery framework, we report the visual quality (SSIM) of the user viewport every 1 second within the 60-second user traces, averaging over all scenes, and all user traces. To decrease the influence of the difficult position, we randomly sample 3 different starting points for each trace and show the average SSIM. Since we do not have the ground truth images for all viewports in our real user traces (the standard datasets only provide ground truth images for a handful of viewports), we calculate the SSIM with reference to renderings generated by the official pre-trained 3DGS models.

4.2 Baselines

We evaluate our 3DGS delivery pipeline with our layered representations, marked as "Ours", along with two other baselines "Splats sorting" and "Non-layered models". We also show the ideal performance that could be achieved by preloading our largest separate model (180k splats, generated after Algorithm 2), onto the client in advance instead of downloading it, marked as "Pre-load".

- **Splat sorting.** We sort the Gaussian splats of the officially pre-trained 3DGS models with their global significance score proposed by LightGaussians[9]. Recall that this score is viewport-independent so it does adjust to where the user is currently looking. When downloading, splats are progressively retrieved with higher scores first. We marked it as "Sort" in the following results.
- **Non-layered models.** We use our Algorithm 2 to create independent models with different target number of splats without the layered structure. When downloading, a complete stand-alone version of the scene is downloaded sequentially at increasing quality. Once a lower version is retrieved and displayed, a higher version is subsequently downloaded. This method is annotated as "Separate" in the following results and corresponds to cases II and IV from Table 2.

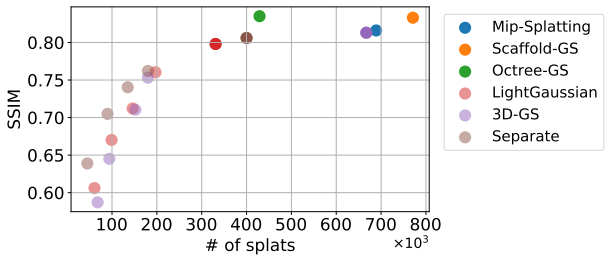
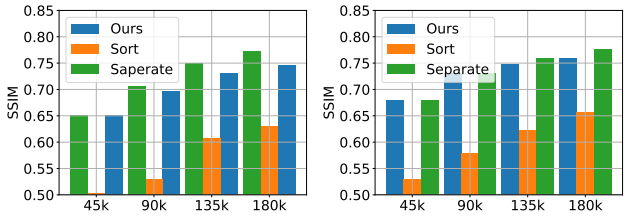


Figure 6: SSIM vs number of splats for the Train scene in the Tanks&Temples Datasets [15]. Our methods provide viable options to deliver 3D scenes with fewer splats.



(a) With segmentation **(b) Without segmentation**

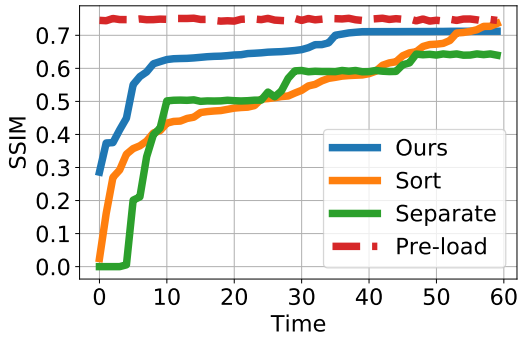
Figure 7: Visual quality (SSIM) of our layered splats (Ours) compared to baselines. The layering achieves high visual quality, even close to the non-layered versions (Separate).

We also compare our model with various 3DGS models [9, 18, 24, 33], particularly [9] and [24] are specifically designed for model compression.

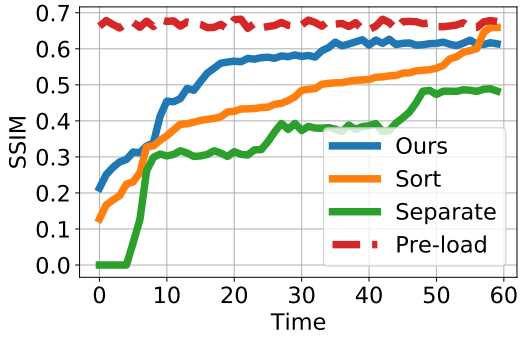
4.3 Visual quality of our layered splats

Our algorithm precisely controls the number of splats while achieving reasonable visual quality compared to models from the literature that contain larger numbers of splats. The results, presented in Figure 6 and Figure 2, utilize ground truth images and positions from the test set to compute the SSIM. The performance values of models from other works (dots without transparency) are directly taken from [9, 24]. Our models ("Ours" and "Separate") significantly reduce size and provide viable options to deliver 3D scenes with fewer splats. We created additional versions of original 3DGS model [13] and LightGaussians [9] (dots with transparency) by optimizing their hyperparameters using grid search to vary the model size. Note that to ensure a fair comparison with other models from the literature, we exclude feature distillation and quantization strategies. Such strategies can be applied to all splat models to reduce file size, although not splat numbers. Overall, our algorithm not only precisely controls the number of splats but also achieves higher SSIM than other compressed variants, tackling the challenges discussed in Section 3.1.

Our algorithm constructs layered models without significantly compromising performance. We also evaluate Algorithm 3 by comparing against the splat-sorting and non-layered model baselines across all 8 scenes with varying model sizes. In Figure 7, we showed that our layered 3DGS ("Ours") outperforms the splat-sorting model significantly. Moreover, although our model builds layered-structured models for cumulative Gaussian splats that could potentially constrain the optimization space and worse visual quality, our algorithm 3 still achieves comparable performance with the non-layered model ("Separate").



(a) SSIM of synthetic traces.



(b) SSIM of real traces.

Figure 8: We evaluate our pipeline with both synthetic and real traces for the segmented and layered 3DGS we developed. The SSIM is significantly higher than the baselines and not far from the optimal.

4.4 Main results of our delivery pipeline

Our delivery pipeline outperforms other baselines because our scheduler can effectively retrieve the splats within the user's viewport. The main performance is illustrated in Figure 8 for both synthetic and real user traces. Within the first 5 seconds, "Ours" significantly outperforms other baselines. For "Sort", not only the visual quality of models with limited number of splats is poor, but their selection

is only determined by a global significance score, which is not view-dependent. Worst of all, "Separate" can not finish downloading even the smallest model, resulting in blank rendered images and an initial SSIM of 0. In contrast, our model efficiently selects the most critical splats for the current viewport, thereby achieving superior performance. Even when the base 45k-splat "Separate" model is fully loaded at around 8 seconds on average, our model achieves better performance by prioritizing important splats based on predicted future viewports, such as those closer to the camera.

With our layered model, performance stabilizes after downloading all 180K splats, which takes approximately 30 seconds. The performance is comparable to the pre-loading 180k separate model, with only minor losses due to the layered structure. However, "Sort" can achieve better performance by loading more splats towards the end of the traces (360k splats at time 60s), because it has access to a larger model. Regarding "Separate", although the model's visual quality is slightly better than ours for the same number of splats, the lack of progressively overlapping splats in different layers necessitates loading the entire larger model and discarding the previous one. This process results in substantial bandwidth wastage, leading to lower SSIM compared to our model under limited bandwidth conditions.

The best SSIM we can achieve is lower than the ideal values shown in Figure 7, particularly for real traces shown in Figure 8a. We find that this is because users may explore some strange positions in the scene, such as walking too close or inside the train, or attempting to walk beyond the boundaries of the scene. Our data collection revealed that users are often curious about the holes and low-quality parts, resulting in out-of-scene viewports. Even the original pre-trained 3DGS model [13] shows low quality in these scenarios due to lack of ground truth images. We provide some example screenshots in Figure 9. The main take-away is that such real user behavior hurts our SSIM, because the SSIM of these strange positions tends to be lower, but this issue affects all methods, not just ours.

4.5 Adapting to other splat representations

3DGS development is rapid, with new versions continuously emerging despite the original version only appearing in 2023. Thus the ability of our pipeline to adapt to such improvements is an important consideration. Two popular strategies are being explored: compressing the feature size of each splat through distillation or quantization, and building hierarchical models for more efficient rendering. We present the performance of our delivery framework using these two alternative 3DGS representations, to demonstrate the flexibility of our framework. The alternative methods are:

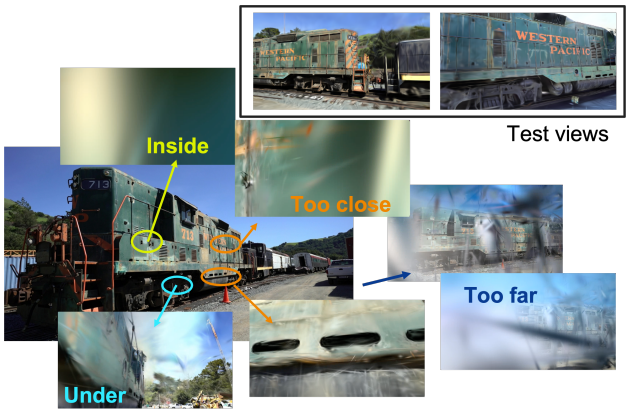


Figure 9: Screenshots capturing various viewpoints. Views from the test set demonstrate higher visual quality. However, some views derived from real-world traces display uncommon characteristics resulting in low visual quality, which affects all methods.

- **Ours + spherical harmonics distillation [9].** We incorporate the spherical harmonics (SH) distillation module from [9] into our layered model to create two versions of splats. A schematic of the model is illustrated at the bottom of Figure 10 as "Ours+Distill". The main idea is that splats are still organized into layers as in our model, but each splat contains both a compact and full version of the SH representation. Meanwhile, other shared features such as opacity and location remain consistent with the original model. When downloading the splats, we add the dimension (compact or full) of the SH as a component to our utility function, and thus the scheduler can choose from an additional dimension of splats with the full or compact SH.
- **Hierarchy [14, 27]** Inspired by the classical idea of Levels of Detail (LoD), some works store the splats in a tree structure. Each node in this tree can be represented in greater detail by several of its child nodes. To showcase the hierarchy model in our framework, we use the non-layered models ("Separate") with different target sizes to form the tree structure, with lower-version splats set as root nodes and the higher-version splats as their child nodes, sharing the same object ID and highest similarity score. An illustration of this method is shown in the top half of Figure 10.

The results with these various 3DGS compression methods are shown in Figure 11. The "Ours + Distill" variant demonstrates that beyond the layered approach of our model, when there are versions of individual splats available through compressing the spherical harmonics, our framework can achieve good performance. Initially, the compact SH version

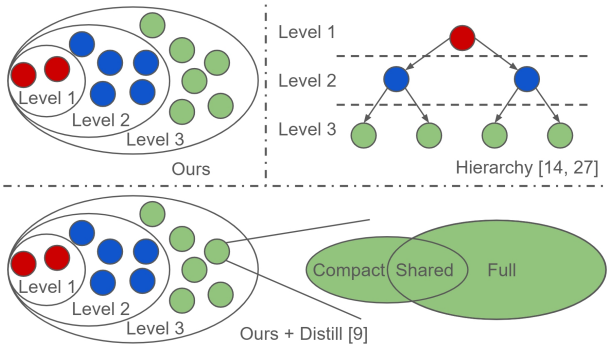


Figure 10: Other representations of 3DGS that our framework can work with. Both "Ours" and "Ours+Distill" utilize layered architectures; however, splats in the "Ours+Distill" model can be represented in two forms: shared features with a compact version or a full version of spherical harmonics (SH) representation. Additionally, the splats can be organized in a hierarchy.

of the splats is loaded, allowing more splats to be downloaded within the same bandwidth and achieving higher SSIM. Over time, as total number of bits transferred increases, the compact SH will be replaced by the full version to enhance visual quality. Sharing other splat features, only the SH feature needs to be upgraded, enabling "Ours + Distill" to quickly reach performance levels comparable to our original model.

The "Hierarchy" variant demonstrates our framework's adaptability to hierarchical splat organization similar to LoD. Same as the original separate model baseline, no splats are shared across levels of the tree, requiring the replacement of multiple splats, leading to bandwidth inefficiencies compared to our layered approach. However, since parent nodes can be replaced by child nodes, "Hierarchy" allows fine-grained progressive splat replacement, achieving a smoother performance enhancement compared to the original separate model that can only do coarse-grained replacement at the per-scene level rather than per-splat.

4.6 3D scenes with and without object segmentation

We compare the performance of the scheduler with and without segmenting the 3D scene into objects. With segmentation, referred to as **Case I** in Table 2, we can decide whether to download splats based on both layer ID and object ID, resulting in a larger selection space. In contrast, in **Case II**, which uses layered splats without object segmentation, for which we only have coarse-grained control over which layer to download. Once the layer is chosen according to the scheduler, the splats within the layer are downloaded in order

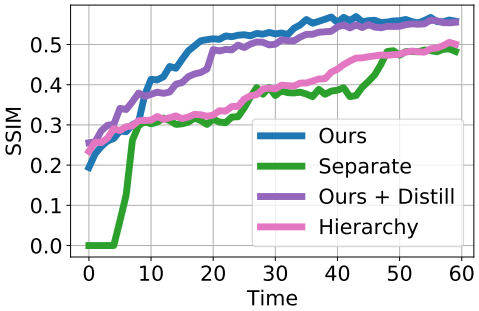


Figure 11: The visual quality evaluation demonstrates the adaptability of our pipeline across various types of 3D Gaussian models.

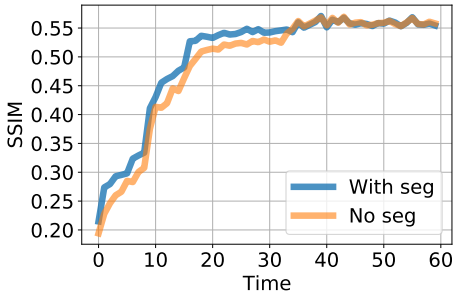


Figure 12: Ours with and without segmentation.

(a) GroundTruth (b) Remove truck (c) Truck Only

Figure 13: With segmentation, we can render objects independently. To visualize the truck clearly (right), the background is set to white.

according to their utility (1). The results, shown in Figure 12, indicate that the scheduler with segmentation demonstrates better performance. This suggests that incorporating object ID information enhances the Scheduler.

With segmentation, we can also selectively render specific objects. For instance, in Figure 13b, we exclude the truck and render only the background. Conversely, in Figure 13c, we only render with the splats labeled as the truck which takes only 25.9% of the total splats. This approach enables more efficient rendering based on user preferences. For example, users may prefer to prioritize the rendering of the main foreground object before the background [5, 16, 28].

4.7 Latency

We also measure the latency shown in Figure 14. To perform the rendering, we first initialize the camera parameters, a

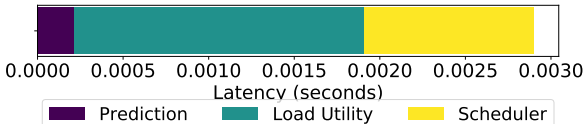


Figure 14: Latency breakdown per time window.

process that takes approximately 1.30 seconds but occurs only once. Following this, the model predicts the viewport and bandwidth in parallel; the maximum latency of these two predictions is measured as "prediction". Once the predictions are completed, we use the predicted viewport for utility calculation and predicted bandwidth to determine which splats to download with the scheduler. Calculating the utility of all splats for a single position takes around 10.29 seconds, which is infeasible for real-time rendering. To address this, we precompute the utility values by uniformly sampling 1,000 positions and 1,728 rotations, then load them in runtime. This precomputation step is measured as "Load Utility". Next, we run the scheduling algorithm to decide which splats to download. In practice, we predict Gaussians every $T = 1s$, so the accumulated latency for Gaussian prediction remains reasonably small. For visualization, as the model size is compressed, the rendering time of our largest model(180k) can be reduced to 0.00369s. The rendering can happen at any time with current downloaded splats. All values are measured with 2vCPU @ 2.2GHz.

4.8 Ablation studies

User viewport prediction is inherently imperfect due to the unpredictable nature of human behavior and attention. Environmental factors and real-time processing limitations further contribute to inaccuracies. Similarly, bandwidth prediction is challenged by fluctuations in network conditions. The average error of our viewport predictor is presented in Figure 15. In Figure 16, we showed an example of the predicted network bandwidth with the ground truth network trace. The performance of our system is reasonable, and strategies such as recenter have been implemented to enhance accuracy. However, imperfections remain. This raises the question of how these inaccuracies affect the overall system. To investigate this, we conducted an ablation study, running the system with perfect prediction of viewport or bandwidth, with results presented in Figure 17. Replacing both predictors with ground truth values improves average quality, as imperfect predictions can lead to suboptimal splat downloads and potential bandwidth wastage; however, the observed gaps are relatively modest, with average SSIM errors of 0.032 for bandwidth and 0.021 for viewport over a 60-second period. This indicates that the predictive system works effectively.

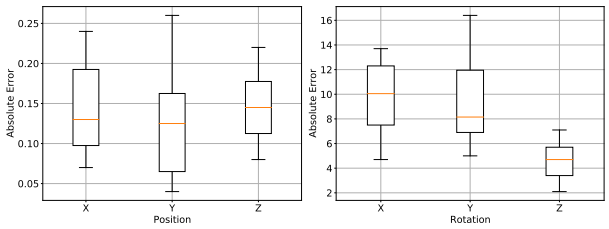


Figure 15: Absolute error of viewport prediction.

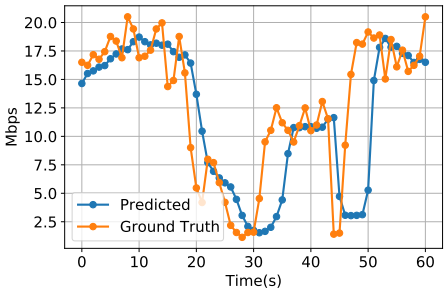


Figure 16: Absolute error of bandwidth prediction. The Mean Absolute error of all traces is 4.293 Mbps with a standard deviation(std) 0.796.

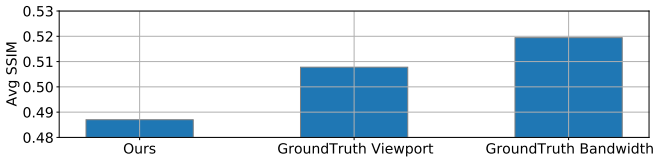


Figure 17: Ablation study with perfect user viewports or network bandwidth prediction.

5 CONCLUSIONS

In this paper, we created an efficient delivery framework for 3D scenes using layered 3D Gaussian splats. We developed a training pipeline to create layered 3DGS models, where the scenes can further be segmented into objects to provide fine-grained control for downloading and editing. By layering the 3DGS models and creating splat scheduling algorithms based on user viewport and network bandwidth, our system can adapt to varying network bandwidths while maintaining high visual quality. In addition, the scheduler also works with various types of 3DGS representations. The experimental results show higher performance compared to the baseline, achieving higher average SSIM scores and low overhead. Future work includes adapting our framework to encompass additional 3DGS representations and further prototyping over networks in the wild.

REFERENCES

[1] 2024. Efficient Delivery of 3D Scenes via Layered 3D Gaussian Splats (technical report). https://anonymous.4open.science/r/3dGS_delivery-AF48/.

[2] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. 2021. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *IEEE/CVF ICCV*.

[3] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. 2022. Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields. *CVPR* (2022).

[4] Bo Chen, Zhisheng Yan, Bo Han, and Klara Nahrstedt. 2024. NeRFHub: A Context-Aware NeRF Serving Framework for Mobile Immersive Applications. In *Proceedings of the 22nd Annual International Conference on Mobile Systems, Applications and Services*. 85–98.

[5] Cuiqun Chen, Mang Ye, Meibin Qi, Jingjing Wu, Yimin Liu, and Jianguo Jiang. 2022. Saliency and Granularity: Discovering Temporal Coherence for Video-Based Person Re-Identification. *IEEE Transactions on Circuits and Systems for Video Technology* 32, 9 (2022), 6100–6112. https://doi.org/10.1109/TCSVT.2022.3157130

[6] Yihang Chen, Qianyi Wu, Mehrtash Harandi, and Jianfei Cai. 2024. How Far Can We Compress Instant-NGP-Based NeRF?. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 20321–20330.

[7] Yihang Chen, Qianyi Wu, Weiyao Lin, Mehrtash Harandi, and Jianfei Cai. 2024. HAC: Hash-grid Assisted Context for 3D Gaussian Splatting Compression. arXiv:2403.14530 [cs.CV] https://arxiv.org/abs/2403.14530

[8] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. 2023. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 16569–16578.

[9] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. 2023. Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps. *arXiv preprint arXiv:2311.17245* (2023).

[10] Yongjie Guan, Xueyu Hou, Nan Wu, Bo Han, and Tao Han. 2023. Metastream: Live volumetric content capture, creation, delivery, and rendering in real time. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*. 1–15.

[11] Bo Han, Yu Liu, and Feng Qian. 2020. ViVo: Visibility-aware mobile volumetric video streaming. In *Proceedings of the 26th annual international conference on mobile computing and networking*. 1–13.

[12] Jian He, Mubashir Adnan Qureshi, Lili Qiu, Jin Li, Feng Li, and Lei Han. 2018. Rubiks: Practical 360-Degree Streaming for Smartphones. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services* (Munich, Germany) *(MobiSys '18)*. Association for Computing Machinery, New York, NY, USA, 482–494. https://doi.org/10.1145/3210240.3210323

[13] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics* 42, 4 (July 2023). https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/

[14] Bernhard Kerbl, Andreas Meuleman, Georgios Kopanas, Michael Wimmer, Alexandre Lanvin, and George Drettakis. 2024. A hierarchical 3d gaussian representation for real-time rendering of very large datasets. *ACM Transactions on Graphics (TOG)* 43, 4 (2024), 1–15.

[15] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. 2017. Tanks and temples: benchmarking large-scale scene reconstruction. *ACM Trans. Graph.* 36, 4, Article 78 (jul 2017), 13 pages. https://doi.org/10.1145/3072959.3073599

[16] Jie Li, Zhixin Li, Zhi Liu, Pengyuan Zhou, Richang Hong, Qiyue Li, and Han Hu. 2023. Viewport Prediction for Volumetric Video Streaming by Exploring Video Saliency and Trajectory Information. *arXiv preprint arXiv:2311.16462* (2023).

[17] Yu Liu, Puqi Zhou, Zejun Zhang, Anlan Zhang, Bo Han, Zhenhua Li, and Feng Qian. 2024. MuV2: Scaling up Multi-user Mobile Volumetric Video Streaming via Content Hybridization and Sharing. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*. 327–341.

[18] Tao Lu, Mulin Yu, Linning Xu, Yuanbo Xiangli, Limin Wang, Dahua Lin, and Bo Dai. 2024. Scaffold-gs: Structured 3d gaussians for view-adaptive rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 20654–20664.

[19] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. 2021. Nerf: Representing scenes as neural radiance fields for view synthesis. *Commun. ACM* 65, 1 (2021), 99–106.

[20] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics* 41, 4 (2022), 1–15.

[21] Arvind Narayanan, Eman Ramadan, Rishabh Mehta, Xinyue Hu, Qingxu Liu, Rostand A. K. Fezeu, Udhaya Kumar Dayalan, Saurabh Verma, Peiqi Ji, Tao Li, Feng Qian, and Zhi-Li Zhang. 2020. Lumos5G: Mapping and Predicting Commercial MmWave 5G Throughput. In *Proceedings of the ACM Internet Measurement Conference* (Virtual Event, USA) *(IMC '20)*. Association for Computing Machinery, New York, NY, USA, 176–193. https://doi.org/10.1145/3419394.3423629

[22] Panagiotis Papantonakis, Georgios Kopanas, Bernhard Kerbl, Alexandre Lanvin, and George Drettakis. 2024. Reducing the Memory Footprint of 3D Gaussian Splatting. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 7, 1 (2024), 1–17.

[23] Feng Qian, Bo Han, Qingyang Xiao, and Vijay Gopalakrishnan. 2018. Flare: Practical Viewport-Adaptive 360-Degree Video Streaming for Mobile Devices. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking* (New Delhi, India) *(MobiCom '18)*. Association for Computing Machinery, New York, NY, USA, 99–114. https://doi.org/10.1145/3241539.3241565

[24] Kerui Ren, Lihan Jiang, Tao Lu, Mulin Yu, Linning Xu, Zhangkai Ni, and Bo Dai. 2024. Octree-gs: Towards consistent real-time rendering with lod-structured 3d gaussians. *arXiv preprint arXiv:2403.17898* (2024).

[25] Heiko Schwarz, Detlev Marpe, and Thomas Wiegand. 2007. Overview of the scalable video coding extension of the H. 264/AVC standard. *IEEE Transactions on circuits and systems for video technology* 17, 9 (2007), 1103–1120.

[26] Yuang Shi, Simone Gasparini, Géraldine Morin, and Wei Tsang Ooi. 2024. LapisGS: Layered Progressive 3D Gaussian Splatting for Adaptive Streaming. arXiv:2408.14823 [cs.CV] https://arxiv.org/abs/2408.14823

[27] Qing Shuai, Haoyu Guo, Zhen Xu, Haotong Lin, Sida Peng, Hujun Bao, and Xiaowei Zhou. 2024. Real-Time View Synthesis for Large Scenes with Millions of Square Meters.

[28] Wenguan Wang, Jianbing Shen, Fang Guo, Ming-Ming Cheng, and Ali Borji. 2018. Revisiting video saliency: A large-scale benchmark and a new model. In *Proceedings of the IEEE Conference on computer vision and pattern recognition*. 4894–4903.

[29] Tan Xu, Bo Han, and Feng Qian. 2019. Analyzing viewport prediction under different VR interactions. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies* (Orlando, Florida) *(CoNEXT '19)*. Association for Computing Machinery, New York, NY, USA, 165–171. https://doi.org/10.1145/3359989.3365413

[30] Mingqiao Ye, Martin Danelljan, Fisher Yu, and Lei Ke. 2024. Gaussian Grouping: Segment and Edit Anything in 3D Scenes.

arXiv:2312.00732 [cs.CV] https://arxiv.org/abs/2312.00732

[31] Wei Ye, Xinyue Hu, Steven Sleder, Anlan Zhang, Udhaya Kumar Dayalan, Ahmad Hassan, Rostand AK Fezeu, Akshay Jajoo, Myungjin Lee, Eman Ramadan, et al. 2024. Dissecting Carrier Aggregation in 5G Networks: Measurement, QoE Implications and Prediction. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 340–357.

[32] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (London, United Kingdom) *(SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 325–338. https://doi.org/10.1145/2785956.2787486

[33] Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. 2024. Mip-splatting: Alias-free 3d gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 19447–19456.

A PROOFS AND PROBLEM DEFINITIONS

Problem 3. *Generalized assignment problem with precedence constraints*

$$\max \sum_j \sum_l \sum_t \Delta\tilde{U}_{jl}[t] x_{jl}[t] \tag{15}$$

$$s.t. \sum_j \sum_l \Delta c_{jl} x_{jl}[t] \leq B[t] \quad \forall t \tag{16}$$

$$\sum_t x_{jl}[t] \leq 1 \quad \forall j, l \tag{17}$$

$$x_{jl}[t] \leq \sum_{t'<t} x_{jl}[t'] \quad \forall j, l, t \tag{18}$$

$$x_{jl}[t] \in \{0, 1\} \quad \forall j, t \tag{19}$$

Lemma 1

Proof. Substituting (6) into the objective function (3), we can re-write the objective as

$$\sum_{j,l} \sum_{t=1}^{T} \Delta U_{jl}[t] y_{jl}[t] = \sum_{j,l} \sum_{t=1}^{T} \Delta U_{jl}[t] \sum_{t'=1}^{t} x_{jl}[t'] \tag{20}$$

$$= \sum_{j,l} \sum_{t'=1}^{T} \sum_{t=t'}^{T} \Delta U_{jl}[t] x_{jl}[t'] \tag{21}$$

$$= \sum_{j,l} \sum_{t'=1}^{T} \Delta\tilde{U}_{jl}[t'] x_{jl}[t'] \tag{22}$$

where (22) is just (15), the objective of Problem 3. The other constraints are equivalent: (4) and (16), (5) and (17), and (7) and (18). □

Theorem 1

Proof. The first step is a reduction from the generalized assignment problem (GAP) to an instance of Problem 3. The GAP can be stated as: given T bins (with capacity c_t) and n items (with profit p_{jt} and weight w_{jt}), find the assignment of items to bins to maximize the profit without exceeding each bin's capacity. Consider the following simplified instance of our Problem 3 where there is only one layer ($L = 1$). Map each item in the GAP to a splat in Problem 3 by setting $\Delta\tilde{U}_j[t] = p_{jt}$, $c_j = w_{jt}$ $\forall t$, and $B[t] = c_t$. Hence the reduction from GAP to Problem 3 is shown, so Problem 3 is NP-hard. Second, we note that Problem 3 is equivalent to Problem 1 by Lemma 1, so Problem 1 is also NP-hard. □

Theorem 2

Proof. The proof involves a reduction from the generalized assignment problem (GAP) to an instance of Problem 2. The GAP can be stated as: given T bins (with capacity c_t) and n items (with profit p_{jt} and weight w_{jt}), find the assignment of items to bins to maximize the profit without exceeding each bin's capacity. Consider the following simplified instance of our Problem 2 where there is only one layer ($L = 1$). Then the objective function (10) can be written as

$$\sum_j \sum_{t=1}^{T} \tilde{U}_j[t] x_j[t] \tag{23}$$

using (9) and following the same derivation as Lemma 1 with $\tilde{U}_j = \sum_{t' \geq t} U_j[t]$, similar to (9). Map each item in the GAP to a splat in Problem 2 by setting $\tilde{U}_j[t] = p_{jt}$, $c_j = w_{jt}$ $\forall t$, and $B[t] = c_t$. Or to be more precise, set $U_j[t] = p_{jt} - p_{j(t+1)}$ (reverse transformation of (9)). Hence the reduction from GAP to Problem 2 is shown. □

Lemma 2

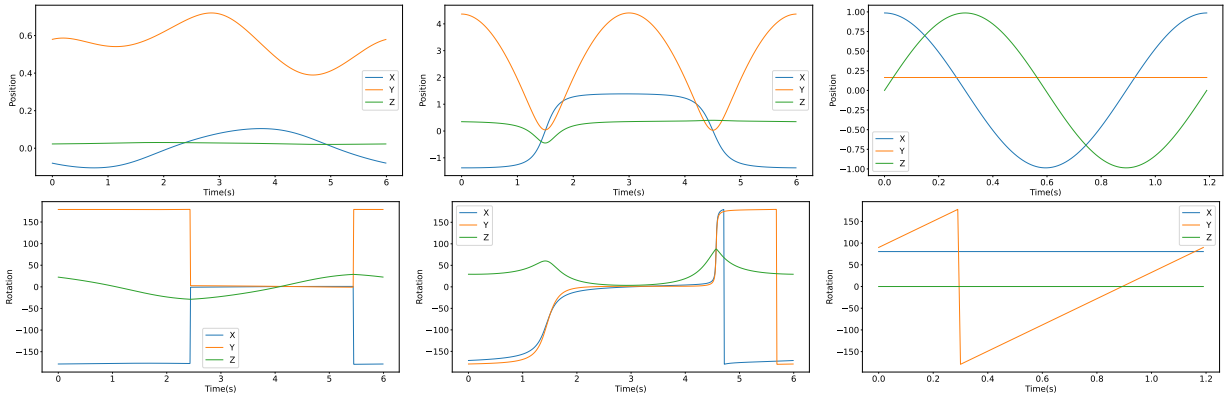


Figure 18

Proof. The multiple choice knapsack problem is defined as:

$$\max \sum_j \sum_l U_{jl}[t] z_{jl}[t] \tag{24}$$

$$\text{s.t.} \sum_j \sum_l c_{jl} x_{jl}[t] \leq B[t] \quad \forall t \tag{25}$$

$$\sum_l z_{jl}[t] \leq 1 \quad \forall j \tag{26}$$

$$z_{jl}[t] \in \{0, 1\} \quad \forall j, l \tag{27}$$

We will show that Problem 3, which is equivalent to Problem 1 by Lemma 1, can be transformed into this knapsack problem.

Let the decision variable from Problem 3 be defined as $x_{jl} \equiv \sum_{l' \geq l} z_{jl'}$. For a given time slot, the objective (15) from Problem 3 can be re-written as:

$$\sum_j \sum_l \Delta U_{jl} x_{jl} = \sum_j \sum_l (U_{jl} - U_{j,l-1}) x_{jl} \tag{28}$$

$$= \sum_j \sum_l U_{jl} (x_{jl} - x_{j,l+1}) \tag{29}$$

$$= \sum_j \sum_l U_{jl} \left(\sum_{l' \geq l} z_{jl'} - \sum_{l' \geq l+1} z_{jl'} \right) \tag{30}$$

$$= \sum_j \sum_l U_{jl} z_{jl} \tag{31}$$

A similar transformation can be done on the cost constraint (16) from Problem 3 to turn it into the knapsack cost constraint (25). Finally, we see that (17) from Problem 3 can be written as $x_{jl} = \sum_{l' \geq l} z_{jl'} \leq 1, \forall j, l$, which implies $\sum_l z_{jl} \leq 1, \forall j$, which is constraint (26) from the knapsack problem. □

B VISUAL QUALITY OF OUR LAYERED SPLATS

We further evaluate the models using additional metrics, as presented in Table 3. Our layered model consistently demonstrates comparable performance to the "Separate", while consistently outperforming the "Sort". However, due to variations in scene size, model performance differs significantly across scenes. For instance, the "Bicycle" scene, which is notably large with an original model size of 1.52 GB, exhibits lower visual quality.

C SYNTHETIC TRACES GENERATION

Since the user's movements are uncontrollable, and as described in the main body, they might focus on some out-of-scene viewports such as holes, which affects the evaluation of our pipeline. Therefore, we generated 6 synthetic traces. 3 traces are generated with the test views and the other 3 are visualized in Figure 18.

Table 3: Evaluation was conducted using visual quality metrics, including PSNR(Peak Signal-to-Noise Ratio), SSIM(Structural Similarity Index), and LPIPS(Learned Perceptual Image Patch Similarity).

	# of splats	psnr			ssim			lpips		
		Separate	Sort	Ours	Separate	Sort	Ours	Separate	Sort	Ours
Bicycle	45000	21.315	14.180	21.315	0.464	0.313	0.464	0.571	0.612	0.571
	90000	21.771	14.663	21.817	0.490	0.342	0.493	0.531	0.572	0.523
	135000	22.082	15.020	21.949	0.510	0.363	0.497	0.499	0.545	0.502
	180000	22.301	15.317	22.028	0.527	0.379	0.516	0.474	0.525	0.490
Bonsai	45000	25.930	18.193	25.930	0.797	0.639	0.797	0.371	0.458	0.371
	90000	27.993	20.279	27.685	0.856	0.707	0.853	0.306	0.399	0.315
	135000	29.159	21.848	28.304	0.887	0.758	0.875	0.269	0.357	0.287
	180000	30.045	23.181	28.600	0.9065	0.799	0.885	0.244	0.3240	0.2731
Counter	45000	24.874	18.214	24.874	0.789	0.643	0.789	0.376	0.446	0.376
	90000	26.412	20.422	26.410	0.831	0.706	0.834	0.319	0.388	0.315
	135000	27.270	21.957	26.847	0.854	0.748	0.850	0.286	0.3497	0.291
	180000	27.799	23.179	27.076	0.870	0.781	0.860	0.263	0.319	0.276
Train	45000	19.603	14.694	19.603	0.638	0.490	0.638	0.427	0.508	0.427
	90000	20.712	16.286	20.610	0.704	0.561	0.700	0.358	0.439	0.364
	135000	21.288	17.399	20.937	0.740	0.615	0.724	0.319	0.390	0.335
	180000	21.608	18.355	21.047	0.762	0.658	0.734	0.293	0.352	0.322
Truck	45000	21.106	13.870	21.106	0.707	0.559	0.707	0.380	0.460	0.380
	90000	22.527	14.872	22.5652	0.767	0.575	0.771	0.315	0.4352	0.310
	135000	23.239	16.104	22.902	0.799	0.625	0.790	0.275	0.384	0.284
	180000	23.676	17.104	23.071	0.819	0.664	0.800	0.248	0.346	0.268
Figurines	45000	21.228	14.075	21.228	0.461	0.439	0.461	0.541	0.603	0.541
	90000	22.281	14.872	22.279	0.493	0.451	0.490	0.507	0.587	0.522
	135000	23.938	20.986	23.610	0.509	0.458	0.498	0.500	0.551	0.516
	180000	24.657	22.985	24.176	0.525	0.573	0.514	0.482	0.535	0.516
Ramen	45000	25.236	21.014	25.236	0.798	0.674	0.798	0.397	0.448	0.397
	90000	27.099	22.154	26.804	0.836	0.678	0.814	0.302	0.416	0.338
	135000	27.499	23.891	27.021	0.859	0.701	0.850	0.264	0.411	0.289
	180000	27.787	23.904	27.283	0.884	0.704	0.862	0.240	0.407	0.253
Teatime	45000	27.588	21.891	27.588	0.746	0.685	0.746	0.395	0.403	0.395
	90000	28.016	23.076	27.819	0.770	0.694	0.762	0.341	0.402	0.348
	135000	28.476	23.098	28.051	0.845	0.700	0.816	0.276	0390	0.300
	180000	28.912	23.371	28.517	0.856	0.703	0.833	0.255	0.387	0.296