CaT-GS: Efficient 3DGS Rendering for Large-Scale Scenes with Inter-frame Caching and Tile Scheduling

Supplementary Material

Algorithm 1 Identify Tile Ranges

Data: Render length L, point list $point\_list\_keys$, ranges array $ranges$, task counter $task\_count$, tile number $tile\_number$

Result: Updated $ranges$ array, render label $label$.

$idx \leftarrow global\ thread\ index$;
$k \leftarrow gaussian\ counts\ threshold$;
$key \leftarrow point\_list\_keys[idx]$;
$currtile \leftarrow key \gg 32$ $prevtile \leftarrow 0$
if $idx = 0$ **then**
  $ranges[currtile].x \leftarrow 0$;
  $task\_count \leftarrow tile\_number$;
end
else
  $prevtile \leftarrow point\_list\_keys[idx-1] \gg 32$
  if $currtile \neq prevtile$ **then**
    $ranges[prevtile].y \leftarrow idx$;
    $ranges[currtile].x \leftarrow idx$;
  end
end
if $idx = L - 1$ **then**
  $ranges[currtile].y \leftarrow L$;
end
Synchronize threads, split the task.
if $currtile \neq prevtile$ ***and*** $idx \neq 0$ **then**
  $start \leftarrow ranges[currtile].x$;
  $end \leftarrow ranges[currtile].y$;
  $task\_size \leftarrow end - start$;
  if $end - start > k$ **then**
    $split\_num \leftarrow \lfloor task\_size/k \rfloor$;
    $ranges[currtile].y \leftarrow start + (task\_size - split\_num * k)$;
    for $i \leftarrow 1$ ***to*** $split\_num$ **do**
      $offset \leftarrow$ atomicAdd$(task_count, 1)$;
      Write additional task range to $ranges[offset]$.
      Write tile information $currtile$ and split information i to $label[offset]$.
    end
  end
end

7.1. Detailed Implementation

7.1.1. Dataset Details

The UAV City Dataset is designed for benchmarking city reconstruction tasks using UAV-captured data. The dataset is constructed by flying a UAV around urban areas and extracting frames from recorded videos. It comprises 16 large-scale **real-world** scenes, each covering one to two city blocks, with a typical area exceeding 20,000 m². For each scene, the processed 3DGS dataset includes approximately 800 images along with the original video clip recorded at 30 FPS.

Since real-world scenes contain more complex geometries and textures—such as trees and grasslands—this dataset offers a more realistic simulation of practical applications compared to previous graphically modeled 3D city datasets like UrbanScene3D. This dataset will be open-sourced in recent months.

7.1.2. Task Splitting Implementation

We provide a detailed explanation of our design in Section 3.4 of the paper. The task splitting strategy is implemented by modifying the Tile-Identify step of 3DGS. As outlined in Algorithm 1, after generating the tile range for rasterization, we check whether the number of Gaussians in each tile exceeds a threshold k. If so, the tile task is split with respect to k, generating new subtasks. To identify the tile and the number of splits during rasterization, additional task information of tile index and partition number is stored in a $label$ array. The total task number is reformed to $task\_count$.

During the rasterization stage, the kernel launches $task\_count$ blocks. Blocks with an index below the original tile count proceed with rasterization as usual. For the additional blocks corresponding to split tasks, each one retrieves its respective parameters from the $label$ array. Based on the tile index and split index provided by the $label$ array, the output is written to the corresponding pixel and slice of the image tensor and radiance map to store the values of R and A as defined in Equation (6) of the paper.

A redundancy introduced by our task-splitting kernel involves the early-termination mechanism of rasterization. In standard 3DGS, this mechanism halts alpha blending once the accumulated radiance becomes sufficiently small, which can be intuitively understood as the light being absorbed by preceding Gaussians along the path. Consequently, in our split rendering approach, the latter portions of a split task may become unnecessary. To mitigate this, we employ two key adjustments: first, the early-termination threshold is set higher for these additional task blocks; second, during task splitting, we allocate a larger size to the latter tasks, the first sub-task is set to a size of $task\_size - split\_num * k$ rather than k. Generally, this approach introduces some additional computational overhead compared to the original method.

Figure 7. The speculative process includes the needed Gaussian in subsequent frames and avoids fault effects.

Nevertheless, the overall acceleration achieved still results in performance improvement.

7.2. Speculative Pre-processing Details

We will further demonstrate the effectiveness and robustness of our speculative multi-frame pre-processing design in Section 3.3. Figure 7 shows an example of the last frame in a cached batch, which illustrates the impact of our approach. The first column under w/o-cache shows the result without inter-frame caching—reflecting the original 3DGS pipeline. The second column presents the result of applying inter-frame caching directly without speculative pre-processing. As shown, stride-like visual artifacts appear due to missing Gaussians in certain tiles caused by viewpoint movement. The third column displays the result of CaT-GS, where the speculative pre-processing stage successfully incorporates all necessary points across the frame batch, as the render result remains consistent with the original pipeline.

Table 6. The impact of speculation window size.

UAV-1	FlashGS	None	W-2	W-4	W-8	W-16
Render Len.$\times 10^3$	10,766	10,766	10,982	11,222	15,482	25,934
Max Latency(ms)	7.7	6.3	6.4	6.6	9.1	14.1
UAV-2	FlashGS	None	W-2	W-4	W-8	W-16
Render Len.$\times 10^3$	12,101	12,101	12,382	12,886	15,926	24,621
Max Latency(ms)	8.6	7.1	7.2	7.5	10.9	13.4

We also conducted an ablation study on the speculation window size to evaluate its impact on performance and robustness. As an example, Table 6 presents the corresponding results. Rows labeled W-2 to W-16 show the outcomes when the speculation window size is set from 2 to 16. It can be observed that the rendering length increases nonlinearly as the window size grows, which is attributed to the quadratic decrease in image tile overlap across frames. Although the rendering quality remains consistent even with a window size of 16, the maximum latency increases significantly if the window size is set too large, and this will impact the rendering smoothness. Therefore, we set the initial window size to 4 and designed a Motion Adaptive Adjustment mechanism to dynamically control the window size. A more fine-grained adaptive window strategy could be developed in the future as a potential improvement to our work.

7.3. Compatibility With Pruning Method

In this section, we discuss the adaptability of render pipeline optimization and model pruning. Although model pruning suffers from certain deficiencies that limit its practicality, as outlined in Section 1, we demonstrate that it can be effectively combined with our rendering optimization method to yield situational advantages. Even in these scenarios, our approach maintains superior performance over previous baselines. To evaluate this, we applied the Compact-3DGS method to retrain and prune models on the UAV dataset. As shown in Table 7, UAV1-3 represents the original large-scale model, while UAV1-3P denotes the pruned versions. The results indicate that our method still retains an advantage over the baselines. Furthermore, the combination of model pruning and our method exhibits promising potential for achieving smooth rendering in larger-scale scenes.

Table 7. Render speed across original models and pruned models.

AvgFPS.	UAV-1	UAV-2	UAV-3	UAV-1P	UAV-2P	UAV-3P
Size$\times 10^3$	6,895	7,189	5,926	2,671	3,269	2,241
3DGS	25.2	23.2	54.3	49.2	44.6	73.2
Flash-GS	129.2	113.1	241.1	296.6	278.2	436.9
Ours	241.5	202.5	378.5	489.4	448.3	746.2

7.4. Visual Effects

Finally, we present visual comparisons between our method and the baseline approaches. The figure below shows rendering examples from random viewpoints. It can be observed that our results are largely consistent with those of Flash-GS, which is expected as our method also employs opacity-aware culling in the pre-processing stage. In contrast, when compared with 3DGS, our approach and FlashGS exhibit minor distortions in detailed image regions, often manifesting as color drift in uniform areas. We attribute these artifacts to the opacity-aware culling strategy and consider this issue a potential direction for future optimization.

3DGS	FlashGS	Ours-key	Ours-sub


3DGS
FlashGS
Ours-key
Ours-sub