The Visual Computer (2024) 40:1825–1838
https://doi.org/10.1007/s00371-023-02888-w

ORIGINAL ARTICLE

Hashed, binned A-buffer for real-time outlier removal and rendering of noisy point clouds

Hendrik Sommerhoff[1] · **Andreas Kolb**[1]

Accepted: 27 April 2023 / Published online: 14 June 2023
© The Author(s) 2023

Abstract
Typical point-based rendering algorithms cannot directly handle large outliers or high amounts of noise without either a costly pre-processing of the point cloud or using multiple render passes. In this paper, we propose an A-buffer-based approach that directly renders unprocessed point clouds and filters out outliers in real time, without introducing additional render passes. The core concept of our approach uses bins along each pixel ray in which intermediate information is accumulated. To improve storage efficiency, we extend the binned A-buffer approach using per-ray bin hashing. Our method significantly improves visual quality when rendering noisy point clouds with varying noise levels and large outliers, while only requiring little performance and memory overhead compared to traditional point rendering methods.

Keywords Point rendering · Splatting · Denoising

1 Introduction

Using points as a rendering primitive [25] has gained much popularity over the last decades. In recent years, powerful laser scanners and consumer-grade depth measuring devices like the Microsoft Kinect make it easier than ever to capture dense point clouds of arbitrary objects and scenes. The lack of topology information makes point clouds very lightweight and enables applications such as 3D reconstruction [18, 24, 40], mobile robotics [33], tele-immersion [11, 29] and the visualization of large-scale terrestrial data [3, 13].

However, besides lacking topological information, captured point clouds exhibit challenges in terms of non-even sampling density, locally varying point size, as well as sensor noise and outliers. The latter mainly occurs as flying pixels at jumping edges between foreground and background objects.

When rendering point clouds "as is", the missing topology information can result in missing pixels if the point cloud is sparse and each point only contributes to a single pixel at a time. One way to alleviate this issue is splatting, i.e., to give points a spatial extend, possibly along a plane defined by a normal that is stored together with each point [35, 42]. A naïve but efficient approach often used in 3D reconstruction pipelines [18, 40] is to render each splat as an opaque disk and determine the radius using the distance to the camera such that the rendered image becomes hole-free. This means that the splats need to have an overlap, which necessarily leads to artifacts in high-curvature regions, since only the splat closest to the camera contributes to each pixel.

More sophisticated approaches use a *elliptically weighted averaging (EWA)* of point attributes (like depth and color) based on the distance of fragments from the splat center [35, 42]. Efficient GPU implementations of this approach in a rasterization pipeline usually employ a three-pass approach [4–6]: (1) compute the splats closest to the camera for each pixel, (2) accumulate point attributes within a depth range ϵ (surface thickness) along each viewing ray, and (3) finally normalize the accumulated values. While the traditional three-pass approach works well for homogeneously sampled scenes without noise and outliers, it cannot handle these kinds of real-world data without either discarding the pixel completely or doing costly additional full rendering passes [11].

The *main restriction* of EWA is that only points in the vicinity of the closest splat to the camera are taken into account in the subsequent accumulation pass. If the closest splat is an outlier, all fragments belonging to the real sur-

✉ Hendrik Sommerhoff
hendrik.sommerhoff@uni-siegen.de

Andreas Kolb
andreas.kolb@uni-siegen.de

[1] Computer Graphics Group, Center for Sensor Systems (ZESS), University of Siegen, 57076 Siegen, Germany

face intersection behind the outlier do not get accumulated. However, the question if a splat is an outlier or not can only be answered after all fragments have been processed, since the order of processed fragments is entirely random. As the reconstruction of the real surface for real-world point clouds is mainly a statistical problem, solving this issue efficiently requires information of all fragments to be retained, not just the fragments closest to the camera.

We propose an enhanced three-pass rendering approach for real-world point clouds that efficiently handles high and varying levels of noise in combination with strong outliers, without requiring any preprocessing. Our method borrows from multifragment rendering [38], which has some conceptual similarity to rendering noisy point clouds with outliers, in that a (partial) list of all fragments is retained for each pixel (see Sect. 2). However, due to memory and runtime requirements, multifragment rendering concepts cannot be directly applied to point clouds, which usually have large amounts of pixel overdraw. This paper introduces the *hashed, binned A-buffer* concept, comprising the following main contributions to close this gap:

- Direct rendering of *raw, noisy point clouds with varying noise levels*, without the need of pre-processing.
- The binning concept for accumulating *weighted contributions* of all surface splats along a viewing-ray in a *single-render pass* without explicit storage or sorting of splat-fragments.
- An efficient implementation of binned A-buffers comprising
  - *per-pixel bin hashing* to improve memory efficiency, and
  - $\mathcal{O}(1)$ *splat insertion and surface retrieval complexity* by tracking the closest valid surface during the accumulation pass without an additional post-processing pass.

Our background motivation is instant rendering of point clouds acquired with hand-held depth cameras, which results in point clouds with varying point density and distance-related noise levels. We, therefore, evaluate our method qualitatively and quantitatively using simulated RGB-D range images.

2 Related work

Point-based rendering: The usage of points as rendering primitives was first proposed by Levoy and Whitted [25]. Since purely point-based representations require a very dense sampling, the idea was extended by Zwicker et al. [42] by using object-aligned elliptical resampling kernels which are accumulated in screen space. This surface splatting approach was later mapped to modern graphics hardware by Botsch et al. [4–6], using a 3-pass-approach in a deferred rendering pipeline. Even though splats are elliptical, sharp features can be rendered by clipping them with pre-computed clipping lines [6, 31], which, however, require pre-processing the point clouds.

Closely related to our work is the method proposed by Diankov et al. [11] which directly renders a noisy point cloud with an iterative surface splatting scheme. In their work, they iterate over the usual 3-pass approach 3-5 times, thereby decreasing performance by the same factor.

Another method was proposed by Uchida et al. [37] in the context of rendering transparent point clouds with noisy points and possibly many outliers. However, the method requires costly pre-processing for noise removal and point sorting, and the resulting images cannot contain opaque surfaces due to design restrictions. Similarly, Seemann et al. [36] propose to segment a point cloud into solid and soft points by thresholding a scalar field, like point uncertainty. By only rendering the solid points, this method can also be used for outlier removal, but the scalar field computation requires a pre-processing step, resulting in an offline outlier removal scheme.

An orthogonal area of research is the out-of-core handling and rendering of very large point clouds, which is often accelerated with structures like kd-trees [15]. Pintus et al. [32] propose a non-splatting out-of-core direct rendering approach for massive raw point clouds using screen-space operators, which does not require special point attributes like normals and radii. They define a point's visibility via the maximal aperture cone in the direction of the camera that does not contain any other point. This approach cannot handle a massive amount of outliers like our method.

Recent works have incorporated deep learning approaches into point cloud rendering. Dai et al. [10] aggregate learnable per-point feature vectors into multiple planes along the camera frustum and subsequently use those feature planes as input to a convolutional neural renderer. While the results are impressive, the per-point features have to be retrained per scene, making it unsuitable for direct point cloud rendering. The Z2P neural rendering approach [30] can handle noisy, raw point clouds of small size, e.g., it requires 3 s for 5k points. For an additional overview on recent advances in photorealistic rendering of non-noisy point clouds we refer to the survey by Kivi et al. [20].

Multifragment rendering: Rendering noisy point clouds with outliers requires retaining information from all surfaces, which is closely related to the area of multifragment rendering, which mainly has applications in order-independent transparency, shadows and global illumination. In multifragment rendering, the final pixel value is generated using

a (partial) *list* of all fragments per pixel ray and a post-processing reduction step to combine the list of fragments. This approach allows for more or less arbitrarily complex blending operations, making them potentially more suitable for real-time graphics. For a more complete overview of multifragment methods we refer to the recent survey by Vasilakis et al. [38]. A simple multi-fragment method is depth peeling [14], which basically renders the scene iteratively starting from the closest fragment layer and successively rendering layers front to back. Advanced approaches, which can peel multiple layers in a single-render pass, exist [2, 26]. However, since it is an iterative approach, the performance constraints are prohibitive for rendering point clouds, which can have very high-depth complexity due to overlapping splats, especially in the case of outliers and high noise levels. A-buffer methods [7] store *all* fragments of a pixel in a single-render pass and are more closely related to our work. Since there is currently no hardware implementation for A-buffers, authors write the fragments to pre-allocated data structures in GPU memory, e.g., linked lists [41], fixed size per-pixel arrays [27] or hash-based methods like the HA-buffer approach [23]. As (noisy) point clouds have high-depth complexity, the memory requirements of storing all fragments are extremely challenging and prohibitive for real-time rendering. A more memory-efficient variant of the A-buffer is the k-buffer, which only stores the *k best*, e.g., closest fragments [1, 17] per pixel. Kim et al. [19] proposed an A-buffer compatible k-buffer variant for fragment merging for order-independent transparency. This approach, however, either stores a complete list of fragments and sorts the fragments for merging, which requires a lot of memory, or maintains a sorted list of fragments and inserts/merges new incoming fragments. Inserting new fragments from an unordered fragment stream into a sorted list has an overall worst case complexity of $\mathcal{O}(n^2)$.

Our approach: In contrast to prior work, our hashed, binned A-buffer approach avoids additional render passes (as needed by, e.g., [6, 11]), does not require any preprocessing to remove outliers from raw point clouds, such as [36], and can handle varying point densities (other than, e.g., a k-buffer algorithm). Our approach only requires the "common" three-render passes and realizes *on-the-fly* noise removal by identifying the closest surface above a minimal density threshold without any sorting of surfaces and/or splats. Our hash-based data structure allows to *simultaneously* render the scene, merge fragments and find the closest surface, and provides insertion of new fragments and reading the closest surface with $\mathcal{O}(1)$ runtime complexity.

3 EWA splatting

Our method is a direct extension of the *elliptically weighted average (EWA)* surface splatting approach by Zwicker et al. [42] to noisy point clouds with varying point density. We, therefore, explain EWA splatting and its shortcomings when rendering point clouds with high noise level and outliers in more detail in this section. We follow the hardware-accelerated approach for EWA splatting by Botsch et al. [4] and use a deferred rendering pipeline with modern OpenGL.

We define a point cloud as a set of points $\{(p_i, n_i, r_i, a_i)\}_i$, where each point has a position $p_i \in \mathbb{R}^3$, a normal $n_i \in \mathbb{R}^3$, a radius $r_i \in \mathbb{R}$, as well as additional attributes $a_i \in \mathbb{R}^k$ (usually color). As naïvely rendering this point cloud as simple, normal aligned disks (splats) leads to artifacts at overlaps of neighboring splats, EWA splatting computes a weighted average of these overlapping splats for each screen-space pixel (u, v):

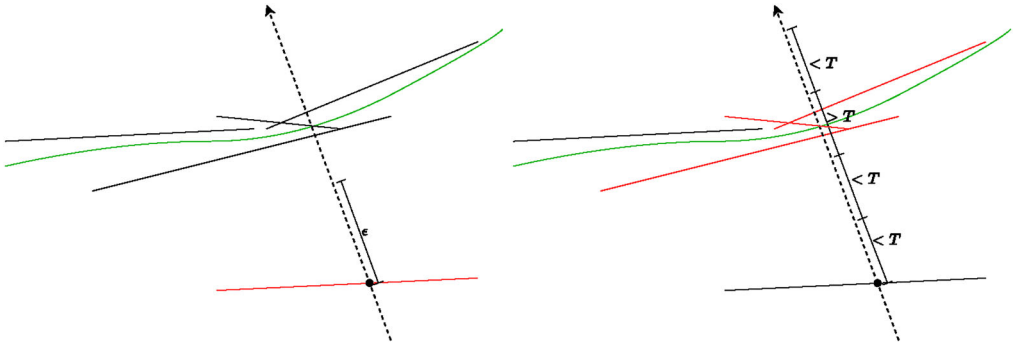
$$a(u,v) = \frac{\sum_i w_i(||q_i(u,v) - p_i||)a_i}{\sum_i w_i(||q_i(u,v) - p_i||)} \tag{1}$$

where $q_i(u, v) \in \mathbb{R}^3$ is the position of the intersection of (u, v)'s viewing ray with the splat with index i and $w_i(d)$ is a radial weighting function that falls to 0 toward the splat boundary. Weighting is commonly done using kernel functions, such as the Wendland function [39]

$$w_i(d) = \begin{cases} (1 - d/r_i)^4(4d/r_i + 1) & d/r_i \leq 1 \\ 0 & \text{otherwise} \end{cases}. \tag{2}$$

To avoid occluded background splats to be included in the weighting computation, a *surface thickness* parameter ϵ is used to discard splats whose intersection with the viewing ray is farther away than ϵ from the intersection closest to the viewer.

Hardware-accelerated implementations usually implement EWA splatting in a 3-pass approach [4] utilizing the rasterization pipeline. In the first pass (*visibility pass*) the splats are moved back by ϵ along the viewing rays and rendered into the depth buffer. The second-pass (*accumulation pass*) writes the weighted attributes $w_i(||q_i(u, v) - p_i||)a_i$ and the weights $w_i(||q_i(u, v) - p_i||)$ from Eq. 2 into two buffers with additive blending and depth test (but not depth write) enabled. The back-shifted depth buffer from the first pass ensures that contributions from points too far away from the closest surface are discarded. Finally, the *normalization pass* computes $a(u, v)$ with a simple pixel-wise division



(a) EWA (b) Ours

Fig. 1 Comparison of the EWA implementation of [4] and our bin-based method. Contributing splats for a pixel along the viewing ray (dotted) are marked in red. The outlier splat is ignored with our method, and only splats close to the real surface (green) are considered for the averaging, since their bin passes the threshold T

of the accumulated weighted attributes $\sum_i w_i(||q_i(u,v) - p_i||)a_i$ by the summed weights $\sum_i w_i(||q_i(u,v) - p_i||)$. Using multiple rendering targets, multiple different attributes can be accumulated in the accumulation pass into additional buffers. Moreover, additional deferred shading passes, e.g., for lighting computation, can be applied, making use of computed shading-relevant attributes like normals or albedo. Both of these aspects are not the focus of this work and will be omitted.

The problem for rendering noisy point clouds with EWA lies in the visibility pass. In general, e.g., assuming white Gaussian noise, it is not possible to set the surface thickness ϵ appropriately, i.e., we would have to set $\epsilon = \infty$ to prevent extreme outliers from suppressing the surface. This is exactly the situation we are confronted with when using real-world sensor data, as the noise level of, for instance, time-of-flight (ToF) cameras depends on various aspects, such as the background light intensity, the surface reflectivity and the scene geometry ("flying pixels") [21] and cannot be captured using a global threshold. Moreover, ϵ represents the surface thickness and should be chosen as small as possible, so that distinct surfaces can be separated and farther surface layers do not interfere with the front surface layer. Figure 1a demonstrates this problem, i.e., the red outlier splat excludes the surface splats for the given ϵ.

A potential avenue to fix this issue is to combine thresholding in the normalization pass to discard pixels whose summed weights are smaller than a certain predefined value [11] and depth peeling [14] to progressively fill in holes with valid surfaces from further behind, as this information has been skipped in the current accumulation pass. This approach is, however, very costly, as it requires re-rendering the scene multiple times.

4 Method

In this section, we introduce our *hashed, binned A-buffer* approach that solves the problems discussed in Sect. 3. Our approach successfully and efficiently removes outliers when rendering noisy point clouds, without adding missing pixels. We will explain the concept of binned A-buffers in Sect. 4.1. Afterward, we will present an efficient implementation of our approach on current GPUs (Sect. 4.2) and the specifics of our per-pixel hashing approach (Sect. 4.3).

4.1 Binned A-buffer

As discussed in Sect. 3, the core of the problem of rendering noisy point clouds with outliers using traditional EWA splatting is the fact that only information about splats within ϵ depth distance to the points closest to the camera is retained after the visibility pass, and discarding outlier pixels requires re-rendering the scene for hole-filling.

We, therefore, propose to generate *binned A-buffers* with equidistant bins that allow for a *parallel accumulation pass* along the viewing ray. Denoting the distance from the camera to the closest splat along the viewing ray at pixel (u, v) as $t_{min}(u, v)$, we compute the bin index of the camera ray intersection with splat i at distance t_i as

$$b(u, v, t_i) = \left\lfloor \frac{t_i - t_{min}(u, v)}{\Delta} \right\rfloor, \tag{3}$$

where $\lfloor \cdot \rfloor$ denotes the floor operation and Δ is the bin size. Using this formulation, the binned A-buffer starts at the closest splat and uniformly extends in viewing ray direction, where we limit the depth extent by fixing a maximum bin count for now.

For each pixel and each bin, we accumulate multiple values: the splat weights $w_i(||q_i(u,v) - p_i||)$ corresponding to the splat's intersection with the ray segment, and the accumulated values $w_i(||q_i(u,v) - p_i||)a_i$ (Eq. 1). That is the buffers used by EWA splatting are 3-dimensional now, depending on the pixel coordinate (u, v) and the bin index b. Denoting

$$\mathcal{I}(u,v,b) = \left\{ i \mid \left\lfloor \frac{t_i - t_{min}(u,v)}{\Delta} \right\rfloor = b \right\} \tag{4}$$

as the set of splat indices whose ray intersections lie in bin b, we can define the accumulated values in each bin as

$$a(u,v,b) = \frac{\sum_{i \in \mathcal{I}(u,v,b)} w_i(||q_i(u,v) - p_i||)a_i}{\sum_{i \in \mathcal{I}(u,v,b)} w_i(||q_i(u,v) - p_i||)} \tag{5}$$

After the accumulation pass we need to find the closest A-buffer bin $\hat{b}$, i.e., the bin with the smallest index, which contains a value larger than a predefined threshold T, i.e.

$$\hat{b}(u,v) = \min\{b \mid \sum_{i \in \mathcal{I}(u,v,b)} w_i(||q_i(u,v) - p_i||) > T\}. \tag{6}$$

This is analogous with the thresholding approach for EWA splatting, except that if the closest bin does not pass this threshold, we simply have to traverse along the bins in depth-order instead of rendering the scene again; Algorithm 1 summarizes this process. Pixels are discarded only, if *none* of the A-buffer bins contain sufficient (weighted) splats to reconstruct a surface. Otherwise, the closest A-buffer bin exceeding the weight threshold is taken as a surface intersection, and its attributes are normalized, yielding the final attribute value (Eq. 2). Figure 1b depicts the concept of the binned A-buffer.

An apparent issue in binning is the impact of the segmentation of the underlying data. Unfavorable locations of the bin borders, i.e., being directly on a surface, could lead to surface misses, because the highest density region is split into two bins. We experimented with an extension to our method which uses an A-buffer with *overlapping* bins that can avoid this situation; however, the improvements were only marginal at the cost of decreased performance and memory efficiency. For a detailed analysis and experimental evaluation of using overlapping bins we refer to the supplementary material.

4.2 GPU implementation of binned A-buffers

We implement our method in a 3-pass approach using modern OpenGL. The first pass is a depth-only pass where we compute $t_{min}(u,v)$ with simple minimum blending, and is equivalent to EWA's visibility pass; see Sect. 3. The second pass contains the bulk of computations. Here we generate the binned A-buffer, and, in parallel, identify the smallest bin that exceeds the threshold in a single pass. In our simple approach, the per pixel bins are implemented as a pre-allocated texture array of size $W \times H \times D$, where D is the number of bins. This means the maximum depth in the scene is limited to $t_{max}(u,v) = t_{min}(u,v) + D\Delta$. As mentioned above we need one texture array for the weights and additional ones for each attribute.

As in GPU-based EWA, the current weight w_{curr} is compute based on the view-space fragment positions in the fragment shader. Using the image load/store functionality of OpenGL, we can directly add these values in the corresponding bin b of the texture array by using the `imageAtomicAdd` function. This function also returns the previous bin value w_{prev}. The minimal bin index currently above the threshold is determined via OpenGL's min-blending. Note that since all weights are positive, the accumulated weight value is monotonically increasing for each bin. Thus, it is guaranteed that the weight inside the bin *after* the `atomicAdd` is *at least* $w_{prev} + w_{curr}$ (line 13 & 14 in Algorithm 3). Accumulating positive weights also ensures that bins cannot fall below the threshold anymore, and min-blending delivers the correct result $\hat{b}$. The essence of the GLSL code of our fragment shader is given in Listing 1.

After computing the binned A-buffer, we do a final normalization pass that reads the weight and attribute textures at the computed minimal bin index. Like in EWA, this pass does not require a re-rendering of the scene. In case none of the bins is above the threshold T (missed pixel), we render the last bin that contains information or apply a common background color, depending on if the captured scene provides some kind of background information or not.

4.3 Hashed, binned A-buffer

The usage of a 3D texture array to represent binned A-buffers introduces two major problems. First, the maximum depth of the scene is limited to $t_{max}(u,v) = t_{min}(u,v) + D\Delta$ for D samples in viewing direction. This is a problem especially for scenes with a large z-extend, but relatively low noise, which induces small values for Δ, and, consequently, too large texture arrays that may exceed the GPU memory. Note that, in the pseudocode in Listing 1 fragments beyond $t_{max}(u,v)$ are rendered into the last bin. The second problem is that most bins are actually unused, since in general a large amount of space in the scene is empty. This leads to a lot of wasted memory, and occupied bins are fragmented across large memory locations, potentially yielding cache coherence problems.

We, therefore, propose to use a hashing scheme that utilizes the available memory more efficiently and conveniently also makes the memory requirement independent of the depth extend of the scene.

```
int   binIdx = int(clamp((t - t_min) / delta, 0, numArrayLayers));
ivec3 coord  = ivec3(gl_FragCoord.xy, binIdx);
float dist   = length(splatCenter - viewRayIntersection);

float currWeight = wendland(dist, splatRadius);
vec3  currColor  = currWeight * splatColor;
float currDepth  = currWeight * splatCenter.z;

float prevWeight = imageAtomicAdd(weightTexture, coords, currWeight);
imageAtomicAdd(colorTexture, coords, currColor);
imageAtomicAdd(depthTexture, coords, currDepth);

if((prevWeight + currWeight) >= threshold){
    minBinAboveTreshold = binIdx; // fragment output; min-blending activated
}
```

Listing 1 GLSL fragment shader snippet for the histogram computation pass. In this example, two attributes are used, color and depth

```
uint hash(uint val) {
    return (val*2654435769) >> (32
   - B);
}
```

Listing 2 Fibonacci hashing in GLSL

We use a hash function that maps any positive integer into the integer range $[0, 2^B - 1]$ to compute the index into our texture arrays. We choose Fibonacci hashing [12] whose 32-Bit GLSL implementation is given in Listing 2. Fibonacci hashing ensures that indices are spread out as evenly as possible. Note that other hash functions which do not require a power of 2 sized array could be chosen. The input of the hash function is the computed bin index b. We resolve collisions with linear probing which can be efficiently implemented with `atomicCompSwap`.

The fragment code for hash table computation including linear probing can be found in Listing 3. The bin index also has to be translated to a hash table index in the normalization pass, with the same collision handling. While in the worst case this could lead to looping over all hash table indices, we found that the overhead compared to the approach without hashing is negligible and the better cache locality due to the smaller arrays even improved performance.

5 Experiments and evaluation

We evaluate the effectiveness of our methods on a variety of scenes and compare it against naïve *opaque* surface splatting and the GPU-based *EWA* splatting approach by Botsch et al. [4], which we will simply refer to as *EWA splatting*, for convenience. In EWA splatting and our method, we accumulate depth and normals, and color, if available. If no color is available, we assume white material and apply simple Phong shading. If not stated differently, we set EWA's surface thickness equal to our bin size to 1 cm, i.e., $\epsilon = \Delta = 0.01$ m,

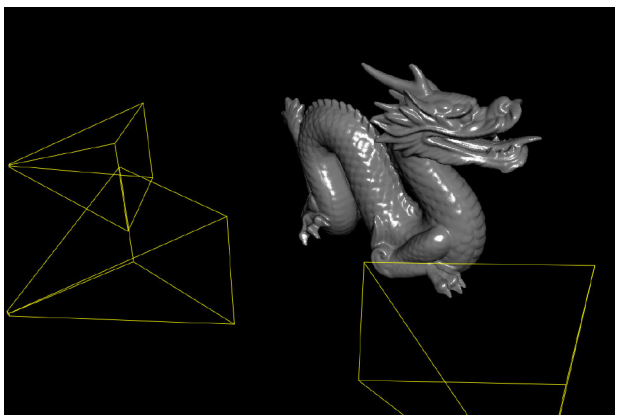


Fig. 2 General setup for our experiments. We assume multiple depth cameras with known positions (yellow) capture depth maps, which we project back into world space to get the point clouds to be rendered

while the threshold T is manually adjusted according to the noise level and point density.

Unfortunately, there is no code base available for Diankov et al. [11]. Moreover, this method also applies hole-filling, i.e., a comparison of the rendering results would be biased, and a runtime comparison gives little insight, as the method applies 3-5 full EWA passes, i.e., 9-15 render passes. Similarly, comparing to Uchida et al. [37] is hardly possible and would not lead to a fair comparison, as they address rendering transparent point clouds and apply offline preprocessing for noise removal and point sorting.

Conceptually, we use a similar experiment setup as Preiner et al. [34] and Diankov et al. [11] and assume that the ground truth surface is captured by multiple RGB-D observations with known camera pose; see Fig. 2). This scenario emulates real-world scene acquisition using hand-held range or RGB-D cameras. The noisy point sets from all camera observations are transformed into world space and concatenated to a single point cloud, which we render from novel camera positions. We do not apply any preprocessing on

```
uint binIdx = int((t - tMin) / delta);
uint hashIdx = hash(binIdx);
uint hashTableSize = pow(2, B);

// ... compute weights and weighted attributes as usual

// linear probing loop
for(int i = 0; i < hashTableSize; i++){
    uint  currIdx = int(hashIdx + i) % hashTableSize;
    ivec3 coords  = ivec3(gl_FragCoord.xy, currIdx);
    uint  prev    = imageAtomicCompSwap(hashTableTexture,
                                        coords, emptyCell, binIdx);
    // if prev is ...
    //  ... emptyCell, then the bin is empty
    //  ... binIdx, then the bin contains values of the same surface
    //  ... anything else, then a collision occurred
    if(prev == emptyCell || prev == binIdx){
        // 1. add weight and attributes to the cell at currIdx
        // 2. write binIdx to frame buffer if weight is over threshold
        return;
    }
    // only reached if collision happened
}
// only reached if all cells are occupied
```

Listing 3 Hashing scheme with collision handling

the point cloud. We compute point normals using a simple cross-product in image space, applied to (auxiliary) bilateral filtered range images. All compared methods receive the exact same splat geometry as their input and render normal-aligned splats.

We use the following dataset for our comparison.

Stanford Dragon [9]: A point cloud with ≈1.5 M points, augmented by randomly sampled 125,000 points in its bounding box; see Fig. 3.

Fireplace Room [28]: This synthetic scene provides ground truth depth. The point cloud comprises 10 depth views rendered from different distances and angles, focusing on the horse statue, and comprises ≈3.5 M points. We add simple Poisson noise approximated as Gaussian noise with standard deviation dependent on the depth z and a noise level parameter k as $\sigma(z) = k(z^2/1000)$; see Fig. 4.

Table : This synthetic scene was generated using the ToF-simulator by Lambers et al. [22] to accurately simulate four RGB-D images and contains ≈0.3 M points; see Fig. 5.

ICL-NUIM: We use the *Livingroom1* and the *Office1* sequence from the augmented dataset [8, 16]. This synthetic dataset comprises 7 RGB-D input images, 10 frames apart from each other in the original stream, summing up to ≈1.4 M points for the *Livingroom1* and ≈1.8 M points for the *Office1* scene. The *ICL-NUIM* dataset results from a simulator emulating a structured light Microsoft Kinect v1 camera. Ground truth poses and color images are taken for other input frames not used in the point cloud; see Fig. 6.

Memorial: This real-world data contains 10 images for ≈0.64 M points from a RGB-D outdoor sequence captured with a ToF-based Kinect v2 under strong ambient lighting conditions, yielding very noisy range data; see Fig. 7.

Depth reconstruction accuracy:

We quantitatively evaluate the depth reconstruction accuracy using the *fireplace room* and the *Table scene* datasets. Moreover, we qualitatively evaluate the depth reconstruction quality using the *Memorial* dataset.

Figure 4 and Table 1 state the quantitative results of the depth reconstruction for the *Fireplace Room* scene; Fig. 5, bottom row gives the quantitative results for the *Table scene* dataset. The error maps display the absolute depth error between the reconstruction results obtained from rendering the point cloud and the ground truth mesh. In general, we observe from the depth error maps that our binned A-buffer approach significantly increases the depth reconstruction quality, both in regions with homogeneous depth and at jumping edges with wrong normal alignment. This gets specifically apparent at the depth discontinuity between the horse statue and the background wall in Fig. 4, leading to a similar effect as achieved by the introduction of clipped splats [6, 31], albeit without the need to pre-compute the clipping lines. Since the distance between the horse and the wall

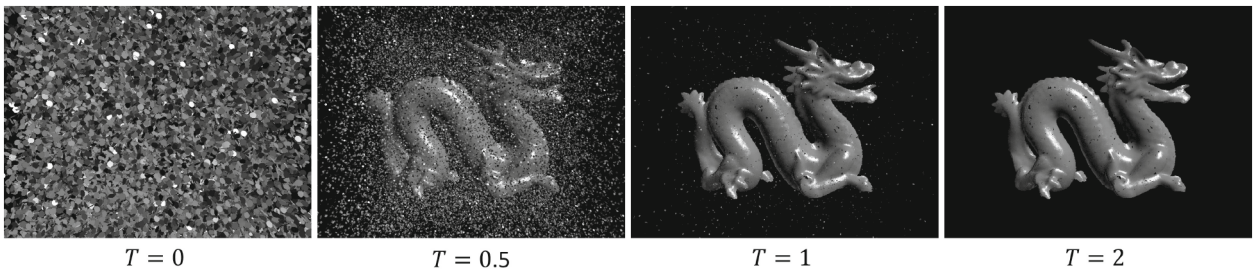


Fig. 3 *Stanford Dragon* with extreme outliers and different thresholds ($\Delta = 0.01m$)

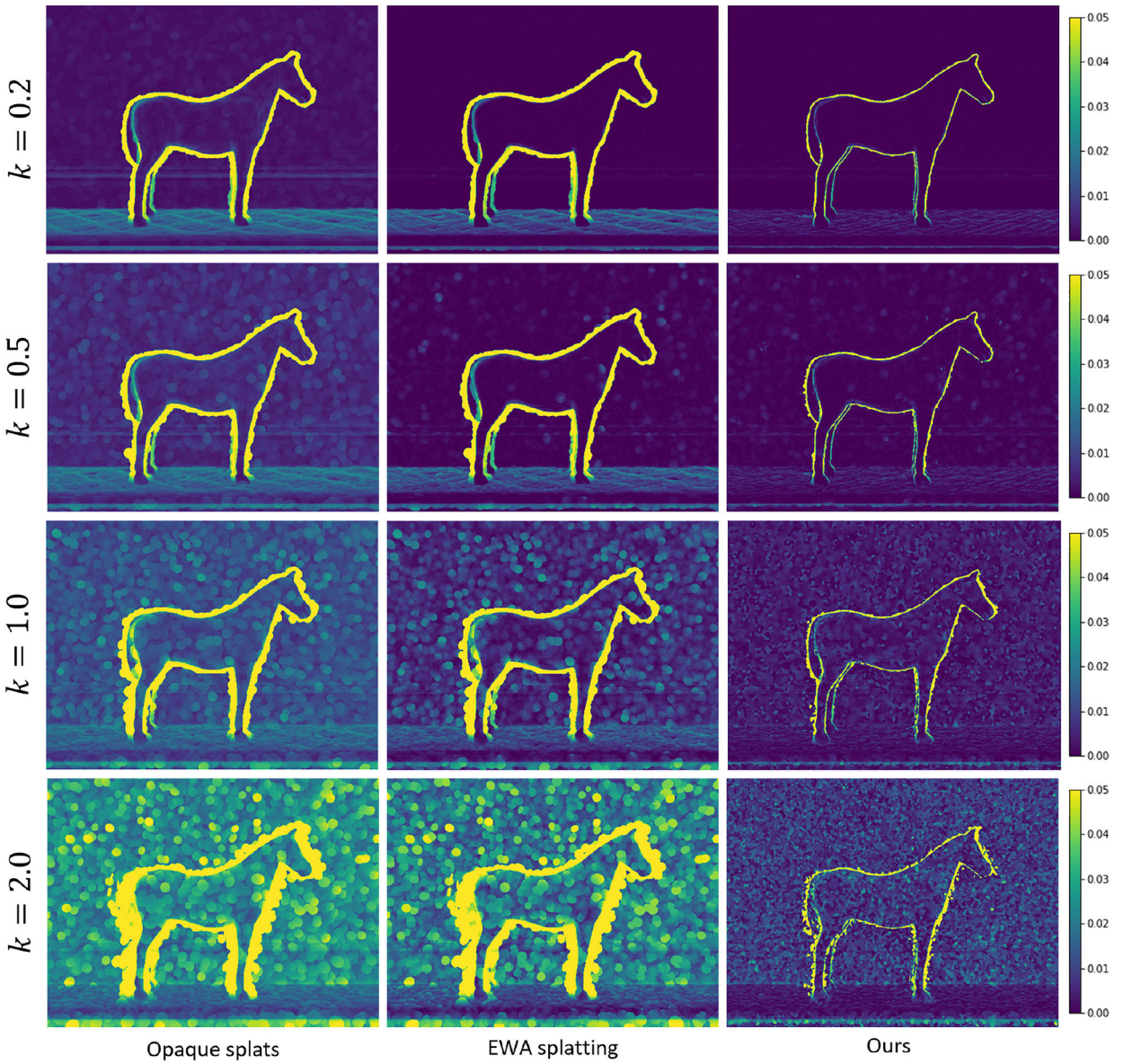


Fig. 4 Difference to ground truth depth for different noise levels in the *Fireplace Room* scene for different noise levels k. Errors outside of the color bar range are clamped for visualization purposes

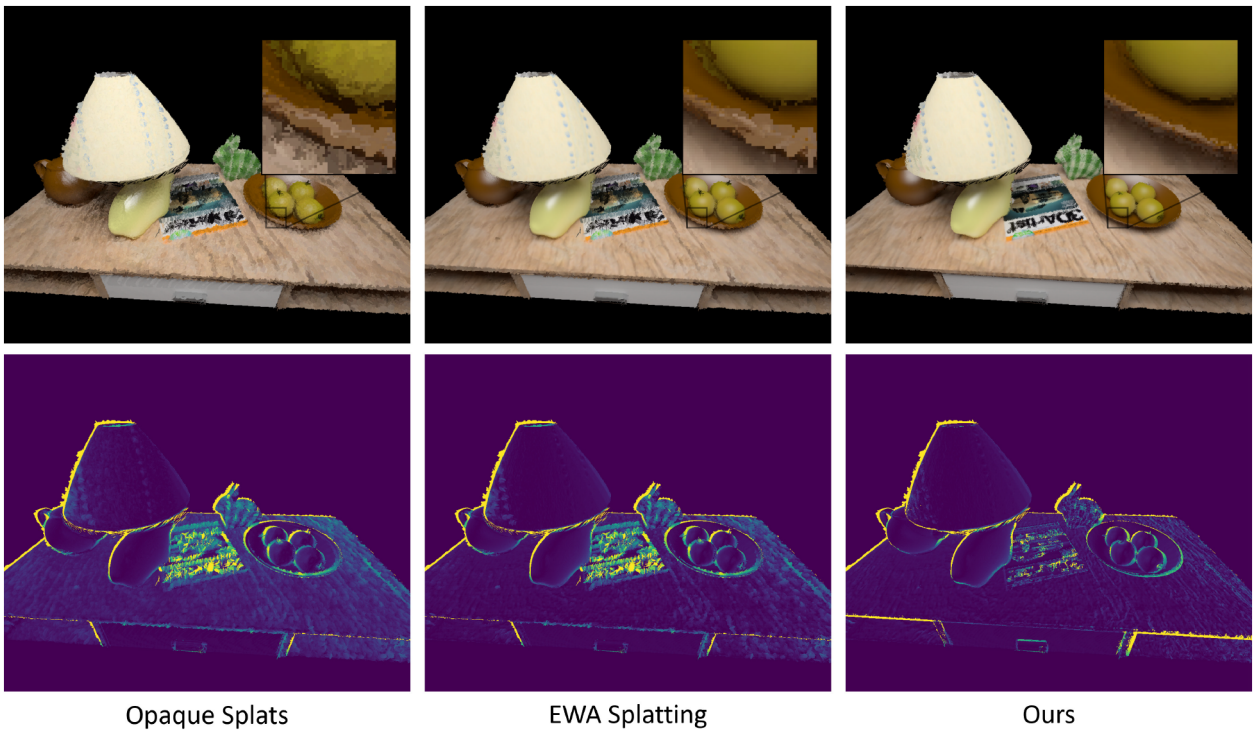


Fig. 5 Color (top row) and depth (bottom row) reconstruction results for the synthetic *Table scene* sequence

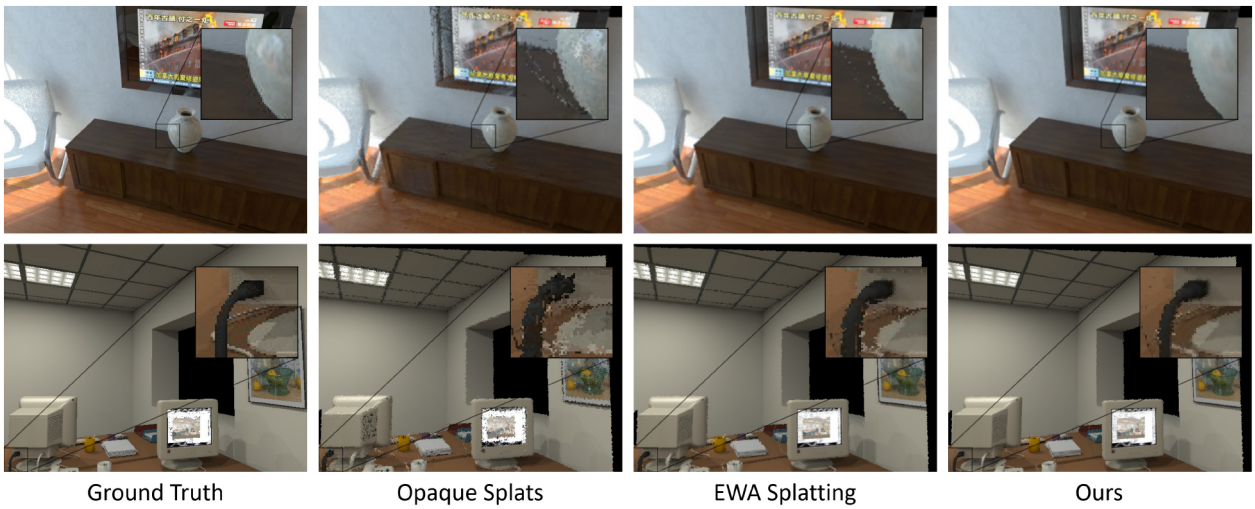


Fig. 6 Results on the augmented *ICL-NUIM* dataset [8]. Top row: *Livingroom1* sequence. Bottom row: *Office1* sequence

is larger than the surface thickness ($\epsilon = \Delta = 0.01m$), EWA splatting renders splat parts that protrude the horse's boundary, while our binned A-buffer approach skips them to a large extend, as the splat density is low in these regions. Increasing noise levels produce more outliers farther away from the surface, which manifests in isolated splats with large errors for traditional EWA splatting, while the binned A-buffer exhibits significantly reduced errors, which is underlined by the gain in the MAE and RMSE figures in Table 1. Similar effects can be seen at the fruit bowl in Fig. 5. Moreover, we see strongly reduced depth errors on the magazine, which is mainly due to the higher noise level on the dark scene region.

However, at the edge of the table in the *Table scene*, we also see an increase of errors. This is mainly due to the usage of a *global* threshold parameter T that generates either holes at the edge of the table or does not sufficiently suppress the outliers. In this particular case our algorithm fails to find a bin at the table edge that is above the threshold and instead renders a bin further from the camera in the background, resulting in a large error.

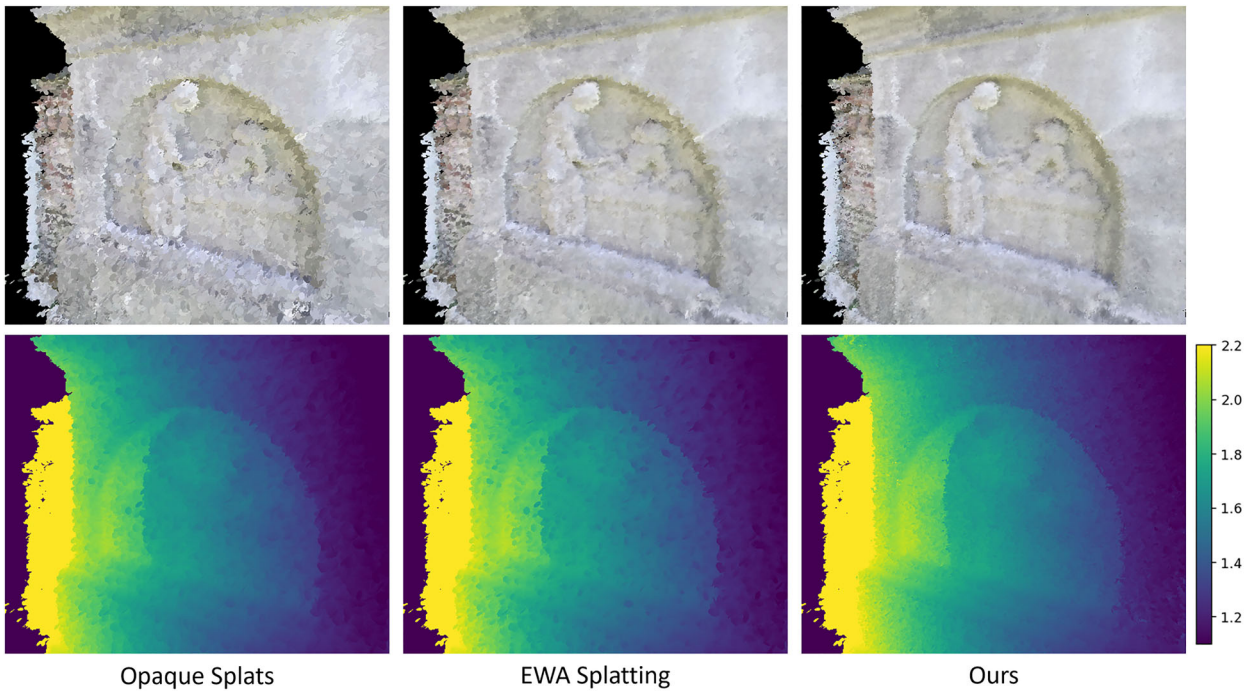


Fig. 7 Reconstruction results for color (top row) and depth (bottom row) on the real world *Memorial* dataset. Due to the high noise we set $\Delta = \epsilon = 0.05m$

Table 1 Quantitative results for the *Fireplace Room* scene for different noise levels

	$k = 0.2$		$k = 0.5$		$k = 1.0$		$k = 2.0$	
	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
Opaque	0.78	1.74	1.13	1.96	1.80	2.55	3.20	4.01
EWA	0.47	1.61	0.59	1.76	1.29	2.31	2.86	3.82
Ours	**0.17**	**0.91**	**0.27**	**1.02**	**0.61**	**1.24**	**1.20**	**1.86**

Bold numbers indicate the best result for a given experiment and error metric. This is standard practice in Computer Science literature
All errors are given in cm

Table 2 Quantitative color reconstruction results

Dataset	PSNR		
	Basic	EWA	Ours
Livingroom1	22.89	26.73	**27.74**
Office1	23.57	27.15	**28.24**
Table	25.05	25.99	**27.73**

Bold numbers indicate the best result for a given experiment and error metric. This is standard practice in Computer Science literature

Considering the real-world *Memorial* dataset in Fig. 7, bottom row, we also observe clearly improved depth quality in homogeneous regions as well as at edges using our binned A-buffer technique.

Color reconstruction quality: We evaluate the color reconstruction accuracy using the *ICL-NUIM*, the *Table scene* and the *Memorial* datasets. As can be seen from the quantitative results in Table 2, our method consistently outperforms the other two approaches.

Figure 6 depicts the visual color reconstructions of the *ICL-NUIM* datasets in comparison with a ground truth pose. While the noise is subtle in this dataset, outliers can be seen at the border of the vase and the armrest of the chair for the *Livingroom1* scene and at the cables in the *Office1* scene, which are nicely removed by our method. In these regions, EWA splatting exhibits strong dilation-like effects that considerably reduces the color reconstruction quality. The *Table* scene contains artifacts commonly seen in ToF depth data, like flying pixels at darker regions (magazine) and at depth continuities (fruit bowl); see Fig. 5. Compared to EWA splatting, our method clearly exceeds EWA's quality in these image regions by removing all flying pixels and picking the proper surface color, thus making the text legible. In the real-world *Memorial* dataset, depicted in Fig. 7, similar effects can be seen also in homogeneous regions. While EWA yields many areas visually similar to opaque splatting, our

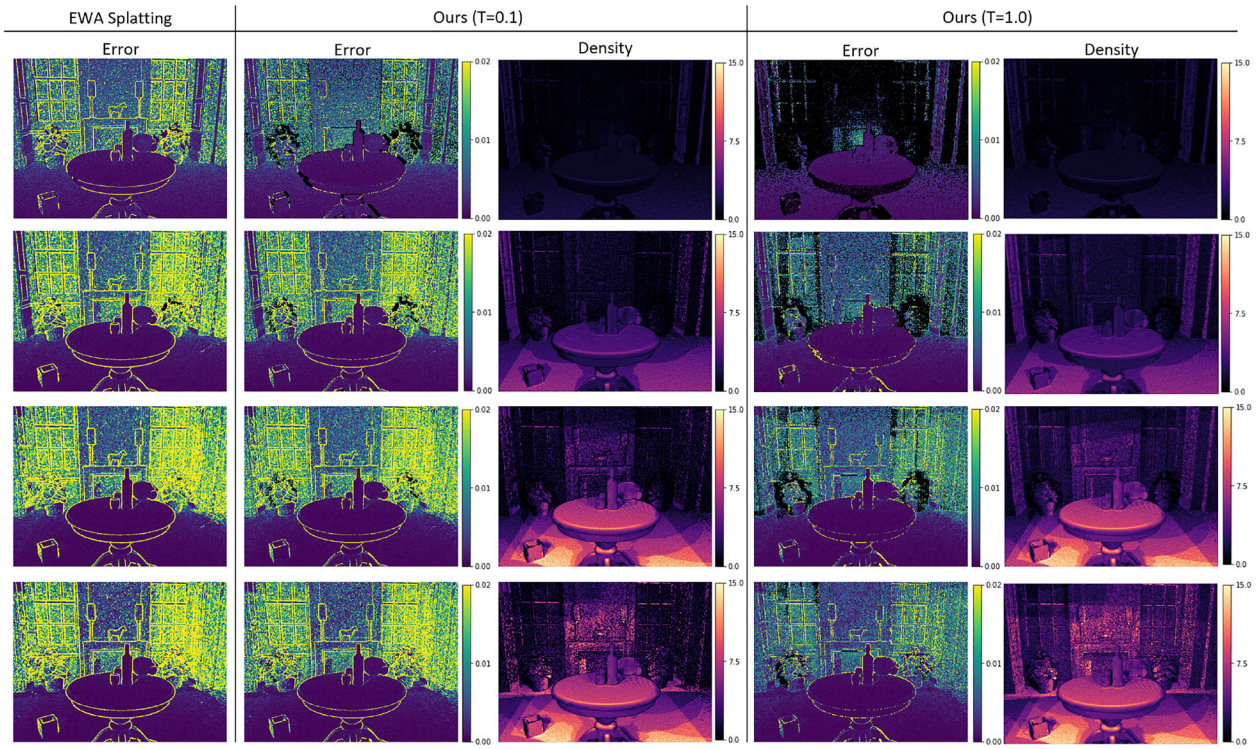


Fig. 8 Evaluation on the *Fireplace Room* scene with different point densities and thresholds. Each row adds points from further capturing positions to the point cloud. The density visualization refers to the weight in the bin selected by our algorithm. Note the increased noise in the background resulting from the distance-dependent noise model. Error images are per-pixel differences to ground truth depth. Missing pixels, i.e., pixels where no bin over the threshold was found by our algorithm, are set to black

binned A-buffer approach further removes these unwanted artifacts.

High noise and outlier levels: We qualitatively evaluate the robustness against high noise levels and strong outliers using the *Dragon* and the *Memorial* datasets. We use the *Dragon* dataset to showcase the impact of the threshold parameter T of our hashed, binned A-buffers method; see also Sect. 4.1. Remember that we set $\Delta = \epsilon = 0.01m$, and that for $T = 0$ out method is equal to traditional EWA splatting. Figure 3 shows the results for the extreme outlier situation in the *Dragon* scene. Our approach can nicely extract the object's surface by adapting the threshold T, i.e., for increasing T the outliers are filtered out and only the denser sampled surface is rendered. Note that all 4 images are the result of a single iteration of our 3-pass approach, we just extract data from different bins of our hashed, binned A-buffer.

Considering the real-world *Memorial* dataset that comprises very high noise levels under daylight conditions in Fig. 7, we can clearly confirm the superiority of the binned A-buffer approach over EWA splatting, both in terms of depth and color reconstruction quality. Note that in contrast to most other experiments, due to the high noise in this scene, we adapt the EWA surface thickness and our bin size to $\Delta = \epsilon = 0.05$ m, while the threshold remains at 1.

Robustness to varying density: Our method assumes that the point cloud has higher density close to the true surface. Our (global) threshold T defines the minimal point density, i.e., any weighted contribution below T is suppressed, which leads to issues on point clouds where the density varies spatially. Moreover, a larger number of splats per bin improves the denoising effect, i.e., detected surfaces for large T are more reliable.

Our experiment setup (see Fig. 2) already induces a spatially varying point density and noise level in all scenes, since the points originate from different, overlapping capturing positions, and the noise model is distance dependent. To further evaluate the robustness of our method, we apply it on a challenging scene with highly varying density and noise distributions (see Fig. 8).

As expected, higher thresholds filter out points in low-density regions. In general, increasing threshold results in only higher-quality bins being retained, which can be seen especially in the first row of Fig. 8, where only pixels with low errors are present for $T = 1$. Higher density (bottom rows) allows the threshold to be set higher, while still retaining a

Table 3 FPS in different scenes

Scene	#Points	EWA	Ours
Fireplace room	3.5 M	92	64
Livingroom1	1.4 M	171	102
Office1	1.8 M	178	118
Table	0.3 M	380	260

The value in parentheses is the FPS when using overlapping bins in our method

Table 4 Measured memory overhead over EWA and FPS of our method on the *Memorial* dataset. EWA renders the scene at 179 FPS

Hash table size	FPS	Memory overhead
2^3	115	+98 MiB
2^4	95	+294 MiB
2^5	56	+700 MiB
2^6	34	+1471 MiB

hole-free image with low error. Moreover, the higher noise level toward the background results in a lower density per bin, which means more points are required here for good results. In any case, even for $T = 0.1$ our approach provides better results than EWA.

Temporal coherence: We conduct an experiment the regarding temporal coherence of our method by rendering a 360-View of the *Dragon* scene, both with and without extreme outliers, and comparing the frames to EWA and using opaque splats. The video can be found in the supplementary material.

Without extreme outliers not much flickering can be seen. This is because the first bin is always anchored at the nearest splat to the camera, i.e., small changes in the camera pose does not significantly change the bin borders for a given a surface position. Flickering is more pronounced when extreme outliers are present, since the first bin's location is most likely determined by an outlier far from of the surface position, yielding strongly varying bin borders and, thus, estimates of the surface position for slightly changed camera poses. Compared to EWA, the amount of flickering is on par or slightly reduced.

Performance: All experiments were performed on a machine with an Intel i7-8600K processor and a GeForce 1080Ti GPU. We achieve real-time framerates for all shown scenes and experiments in this paper (see Table 3). Since we accumulate the contributions to all surfaces along a viewing ray at once, our method has a computational and memory overhead compared to EWA splatting on all scenes. The EWA splatting implementation we compare to also uses hardware blending for accumulation, which in general is faster than `imageAtomicAdd` operations used in our approach. In general, the memory overhead of our method depends on the chosen hash table size, which cannot be too small because it both leads to more hash collisions and could result in overflow if the hash table is smaller than the amount of surfaces in the current viewing direction. However, if the hash table size is too large, the performance of our method also degrades, which we suspect is due to worse cache locality of our texture accesses.

We exemplary compute the running performance as well as memory overhead for different hash table sizes on the *Memorial* dataset, which is quite challenging for our algorithm since the high noise and many outliers result in write accesses to many different bins per pixel. The scene contains 654,083 points which we render at a resolution of 1024×848, which is twice the resolution of the input images. The result can be found in Table 4.

A hash table size under 2^3 results in memory overflow and artifacts occur. Note that sizes other than a power of two would be possible if a different hash function were used.

6 Discussion and conclusions

The main benefits of the binned A-buffer method regarding the quality of the reconstructed color and depth values are explicitly prominent for noisy (real-world) point clouds with strong outliers and comprise (1) filtering strong noise of surface splat and suppressing large amounts of outliers, (2) suppressing the dilation-like effect caused by splats at boundaries of front objects, and thus, (3) reducing the visibility of individual splats that protrude the contour of the front object.

Our hashed, binned A-buffer algorithm implementation is efficient in terms of memory footprint and runtime performance. Still, there is room for improvement, mainly in terms of the selection of the threshold value T. As discussed in Sect. 5, in the paragraph on depth reconstruction accuracy and the ablation on varying pixel densities, in certain constellation there is no single, global threshold that fits all local needs. Therefore, a theoretical investigation of the statistical relations between noise level, outlier density, and surface splat density aiming at an automated adaptation of T is a very worthwhile research objective.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s00371-023-02888-w.

Funding Open Access funding enabled and organized by Projekt DEAL.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Data availability statement The *Memorial* and *Table scene* datasets that were generated for this paper will be made publically available upon acceptance of this manuscript.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Bavoil L., Callahan, S.P., Lefohn, A., Comba., J.L.D., Silva, C.T.: Multi-fragment effects on the gpu using the k-buffer. In Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games, pp. 97–104 (2007)
2. Bavoil, L., Myers, K.: Order Independent Transparency with Dual Depth Peeling. Technical report, NVIDIA (2008)
3. Bienert, A., Scheller, S., Keane, E., Mohan, F., Nugent, C.: Tree detection and diameter estimations by analysis of forest terrestrial laserscanner point clouds. In: ISPRS Workshop on Laser Scanning, vol. 36, pp. 50–55. IAPRS Espoo, Finland (2007)
4. Botsch, M., Hornung, A., Zwicker, M., Kobbelt, L.: High-quality surface splatting on today's gpus. In Proceedings Eurographics/IEEE VGTC Symposium Point-Based Graphics, 2005., pp. 17–141. IEEE (2005)
5. Botsch, M., Kobbelt, L.: High-quality point-based rendering on modern gpus. In 11th Pacific Conference on Computer Graphics and Applications, 2003. Proceedings, pp. 335–343. IEEE (2003)
6. Botsch, M., Spernat, M., Kobbelt, L.: Phong splatting. In Proceedings of the First Eurographics Conference on Point-Based Graphics, pp. 25–32 (2004)
7. Carpenter, L.: The a-buffer, an antialiased hidden surface method. In Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques, pp 103–108 (1984)
8. Choi, S., Zhou, Q.-Y., Koltun, V.: Robust reconstruction of indoor scenes. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015)
9. Curless, B., Levoy, M.: A volumetric method for building complex models from range images. In Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, pp. 303–312 (1996)
10. Dai, P., Zhang, Y., Li Z., Liu, S., Zeng, B.: Neural point cloud rendering via multi-plane projection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 7830–7839 (2020)
11. Diankov, R., Bajcsy, Ruzena: Real-time adaptive point splatting for noisy point clouds. GRAPP (GM/R) **7**, 228–234 (2007)
12. Knuth,D.E.: The Art of Computer Programming, vol. 3. Addison Wesley Longman Publishing Co., Inc., second edition (1998)
13. Dong, Z., Liang, F., Yang, B., Yusheng, X., Zang, Y., Li, J., Wang, Y., Dai, W., Fan, H., Hyyppä, J., et al.: Registration of large-scale terrestrial laser scanner point clouds: a review and benchmark. ISPRS J. Photogram. Remote Sens. **163**, 327–342 (2020)
14. Everitt, C.: Interactive order-independent transparency. Technical report, NVIDIA (2001)
15. Goswami, P., Erol, F., Mukhi, R., Pajarola, R., Gobbetti, E.: An efficient multi-resolution framework for high quality interactive rendering of massive point clouds using multi-way kd-trees. Vis. Comput. **29**(1), 69–83 (2013)
16. Handa, A., Whelan, T., McDonald, Davison, A.J.: A benchmark for rgb-d visual odometry, 3d reconstruction and slam. In: 2014 IEEE International Conference on Robotics and Automation (ICRA), pp 1524–1531. IEEE (2014)
17. Jouppi, N.P., Chang, C.-F.: Z 3: an economical hardware technique for high-quality antialiasing and transparency. In Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware, pp. 85–93 (1999)
18. Keller, M., Lefloch, D., Lambers, M., Izadi, S., Weyrich, T., Kolb.: Real-time 3d reconstruction in dynamic scenes using point-based fusion. In: 2013 International Conference on 3D Vision-3DV 2013, pp. 1–8. IEEE (2013)
19. Kim, Dongjoon, Kye, Heewon: Z-thickness blending: effective fragment merging for multi-fragment rendering. Comput. Graph. Forum **40**, 149–160 (2021). (**Wiley Online Library**)
20. Kivi, P.E.J., Mäkitalo, M.J., Žádník, J, Ikkala, J., Vadakital, V.K.M., Jääskeläinen, P.O.: Real-time rendering of point clouds with photorealistic effects: a survey. IEEE Access **10**, 13151–13173 (2022)
21. Kolb, A., Barth, E., Koch, R., Larsen, R.: Time-of-flight cameras in computer graphics. Comput. Graph. Forum **29**, 141–159 (2010). (**Wiley Online Library**)
22. Lambers, M., Hoberg, S., Kolb, A.: Simulation of time-of-flight sensors for evaluation of chip layout variants. IEEE Sens. J. **15**(7), 4019–4026 (2015)
23. Lefebvre, S., Hornus, S., Lasram, A.: HA-Buffer: coherent Hashing for single-pass A-buffer. PhD thesis, INRIA (2013)
24. Lefloch, D., Kluge, M., Sarbolandi, H., Weyrich, T., Kolb, A.: Comprehensive use of curvature for robust and accurate online surface reconstruction. IEEE Trans Pattern Anal Mach Intell. **39**(12), 2349–2365 (2017)
25. Levoy, M., Whitted, T.: The use of points as a display primitive (1985)
26. Liu, Fang, Huang, Meng-Cheng., Liu, Xue-Hui., En-Hua, Wu.: Efficient depth peeling via bucket sort. Proc. Conf. High Performance Graph. **2009**, 51–57 (2009)
27. Liu, F., Huang, M.-C.., Liu, X.-H., Wu, E.-H.: Freepipe: a programmable parallel rendering architecture for efficient multi-fragment effects. In Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pp. 75–82 (2010)
28. McGuire, M.: Computer graphics archive, July (2017)
29. Mekuria, R., Blom, K., Cesar, P.: Design, implementation, and evaluation of a point cloud codec for tele-immersive video. IEEE Trans. Circuits Sys.t Video Technol. **27**(4), 828–842 (2016)
30. Metzer, G., Hanocka, R., Giryes, R., Mitra, N.J., Cohen-Or, D.: Z2p: Instant visualization of point clouds. In: Computer Graphics Forum, Wiley Online Library **41**, 461–471 (2022)
31. Pauly, M., Keiser, R., Kobbelt, L.P., Gross, M.: Shape modeling with point-sampled geometry. ACM Trans. Graph. (TOG) **22**(3), 641–650 (2003)
32. Pintus, R., Gobbetti, E., Agus, M.: Real-time rendering of massive unstructured raw point clouds using screen-space operators. In: Proceedings of the 12th International Conference on Virtual Reality, Archaeology and Cultural Heritage, pp. 105–112 (2011)
33. Pomerleau, F., Colas, F., Siegwart, R. et al.: A review of point cloud registration algorithms for mobile robotics. In: Foundations and Trends® in Robotics, **4**(1):1–104 (2015)
34. Preiner, R., Jeschke, S., Wimmer, M.: Auto splats: dynamic point cloud visualization on the gpu. In: Childs, H., Kuhlen, T., (eds) Proceedings of Eurographics Symposium on Parallel Graphics and

Visualization, pp. 139–148. Eurographics Association 2012, May (2012)
35. Rusinkiewicz, S., Levoy, M.: Qsplat: a multiresolution point rendering system for large meshes. In Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, pp. 343–352 (2000)
36. Seemann, P., Palma, G., Dellepiane, M., Cignoni, P., Goesele, M.: Soft transparency for point cloud rendering. In: EGSR (EI&I), pp. 95–106 (2018)
37. Uchida, T., Hasegawa, K., Li, L., Adachi, M., Yamaguchi, H., Thufail, F.I., Riyanto, S., Okamoto, A., Tanaka, S.: Noise-robust transparent visualization of large-scale point clouds acquired by laser scanning. ISPRS J. Photogram. Remote Sens. **161**, 124–134 (2020)
38. Vasilakis, A..-A.., Vardis, K.., Papaioannou, G..: A survey of multifragment rendering. Comput. Graph. Forum **39**, 623–642 (2020). (**Wiley Online Library**)
39. Wendland, Holger: Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. Adv. comput. Math. **4**(1), 389–396 (1995)
40. Whelan, T., Leutenegger, S., Salas-Moreno, R., Glocker, B., Davison, A.: Elasticfusion: dense slam without a pose graph. Robot. Sci. Syst. (2015)
41. Yang, J.C., Holger Grün, J.H., Thibieroz, N.: Real-time concurrent linked list construction on the g. Comput. Graph. Forum **29,** 1297–1304 (2010). (**Wiley Online Library**)
42. Zwicker, M., Pfister, H., Van Baar, J., Gross, M.: Surface splatting. In: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, pp 371–378 (2001)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Hendrik Sommerhoff received his master's degree in computer science from the University of Siegen, Germany, in 2019. He is currently a PhD student at the Computer Graphics and Multimedia Systems Group in Siegen. His research interests include point-based rendering, machine learning and computer vision.



Andreas Kolb received the PhD degree from the University of Erlangen, Germany, in 1995. He is currently the Head of the Computer Graphics and Multimedia Systems Group, University of Siegen, Germany. His research interests include computer graphics and computer vision, including particle-based simulation and visualization, light-fields, real-time simulation, processing, accumulation and visualization of sensor data.