High-Fidelity Point-Based Rendering of Large-Scale 3-D Scan Datasets

Patric Schmitz, Timothy Blut, Christian Mattes, and Leif Kobbelt
Visual Computing Institute,
RWTH Aachen University

***Abstract*—Digitalization of three-dimensional (3-D) objects and scenes using modern depth sensors and high-resolution RGB cameras enables the preservation of human cultural artifacts at an unprecedented level of detail. Interactive visualization of these large datasets, however, is challenging without degradation in visual fidelity. A common solution is to fit the dataset into available video memory by downsampling and compression. The achievable reproduction accuracy is thereby limited for interactive scenarios, such as immersive exploration in virtual reality (VR). This degradation in visual realism ultimately hinders the effective communication of human cultural knowledge. This article presents a method to render 3-D scan datasets with minimal loss of visual fidelity. A point-based rendering approach visualizes scan data as a dense splat cloud. For improved surface approximation of thin and sparsely sampled objects, we propose oriented 3-D ellipsoids as rendering primitives. To render massive texture datasets, we present a virtual texturing system that dynamically loads required image data. It is paired with a single-pass page prediction method that minimizes visible texturing artifacts. Our system renders a challenging dataset in the order of 70 million points and a texture size of 1.2 TB consistently at 90 frames per second in stereoscopic VR.**

Digital preservation and restoration of cultural artifacts is an integral component to the recording of human history. Virtual replicas of historical sites can serve different purposes. They enable archiving of geometry and texture, such that faithful reproduction is possible in the case of decay, destruction, or vandalism. Digital models furthermore broaden the availability of cultural property. Travel distance, admission

Digital Object Identifier 10.1109/MCG.2020.2974064
Date of publication 17 February 2020; date of current version 28 April 2020.

Published by the IEEE Computer Society

charges, lack of accessibility, or the necessity for site protection can deny many individuals the opportunity to experience our cultural treasures.

Digital acquisition of high-precision three-dimensional (3-D) models can nowadays be performed in numerous ways. Using specialized equipment such as 3-D laser scanners, structured light or time-of-flight sensors, or by the fusion of multiview image data in photogrammetry, highly detailed digital reconstructions of real-world objects are achievable. The affordability of sensors combined with ever-increasing data capacity and transmission bandwidth enables us to capture, store, and exchange 3-D scanned datasets of enormous size.

Heritage preservation, however, comprises more than just archiving an artifact's appearance. Ideally, we want to enable observers to experience a historic site as realistically as possible, given the sensor resolution at which the object was captured. People should be able to engage in the virtualized environment and perceive their surroundings in the same way as if visiting the real site.

Virtual reality (VR) offers the potential to create this sense of presence in immersive virtual environments (IVE). Yet, while VR systems have recently become affordable, requirements on rendering performance are still demanding. For a sufficient degree of immersion, the system needs to deliver high interactive frame rates at all times. This is challenging, particularly for datasets that do not fit into video memory. Downsampling and compression combined with dynamic level-of-detail (LOD) techniques can alleviate this, but typically result in decreased visual fidelity.

For exploration of 3-D scanned heritage sites in IVEs, we argue that perceivable losses in reproduction quality are not tolerable. The inability to faithfully reproduce an artifact at interactive frame rates ultimately amounts to a loss of cultural heritage. Our main goal is, therefore, to create a rendering technique that yields the highest visual quality, while rendering massive point cloud datasets at interactive frame rates. To this end, we propose a point-based rendering system using 3-D ellipsoids as a novel rendering primitive.

We furthermore observe that users in IVEs have the unique possibility to approach artifacts very closely. While many cultural heritage sites are not fully open to the public, a digital model enables users to explore freely. Even for high-quality photographs, the achievable resolution from a given distance might be insufficient. Therefore, we apply a data-driven super-resolution technique to improve the effective texture resolution by adding plausible detail.

Increasing the effective texture resolution, however, significantly aggravates the problem of insufficient GPU memory. Consequently, we propose a virtual texturing system that performs on-demand loading of image data at full resolution, while minimizing visible artifacts with an efficient prediction algorithm for required virtual texture pages.

In summary, our main contributions are:

- 3-D ellipsoids as improved point rendering primitives for thin and sparsely sampled objects (see the "Ellipsoid Splatting" section);
- virtual texturing for massive image data (see the "Virtual Texturing" section);
- a single-pass page prediction algorithm (see the "Page Prediction" section).

RELATED WORK

Our proposed method draws heavily on prior research in point-based rendering techniques and methods to handle large-scale texture datasets. This section gives an overview of related work in those fields.

Point-Based Rendering

While many applications require a mesh representation with topological surface information, visual reproduction alone does not benefit from it. Points can instead be used directly for rendering, saving manual and computational effort. Levoy and Whitted introduced the idea of using points as rendering primitives.[1] While naive point cloud rendering as colored points or viewer-aligned quads is simple to implement and yields high performance, it can result in visual gaps and aliasing artifacts. Pfister *et al.* proposed *surfels* (surface elements) as improved rendering primitives that approximate the local surface with oriented discs in 3-D space.[2] Zwicker *et al.* created the

Surface Splatting technique that renders surfels in a high-quality manner on the CPU using screen space elliptical weighted average (EWA) texture filtering.[3] Botsch *et al.* further improved the rendering quality with *Phong Splatting* that associates a linearly varying normal field with each primitive.[4] The method was subsequently implemented using capabilities of modern graphics programming units and a cheap but effective approximation of EWA texture filtering.[5] To render sharp features, Zwicker *et al.* proposed the use of *cliplines* that truncate splats along a line in tangent space.[6] Preiner *et al.* developed *Auto Splats* that computes normals and splat radii in screen space during rendering.[7]

Large-Scale Texturing

Much interest in methods to handle the ever-increasing size of texture data has originated in the visualization of large geospatial datasets as well as in the gaming industry.

Texture Streaming Streaming approaches keep only the textures or mipmaps required for rendering in memory at a given point in time. The main challenge is to determine when to load which textures. A simple approach is to subdivide the scene into fixed zones in which the required set of textures is precomputed. By keeping track of neighboring zones, textures can be streamed into memory before they are needed for rendering. Blow presented a system that tracks mipmap levels in a least-recently-used (LRU) cache and predicts required textures using a mip bias and extrapolated camera movement.[8] Dumont *et al.* prioritized textures based on perceptual importance, using factors such as view point, illumination, image contrast and frequency content.[9] Van Waveren proposed a multithreaded streaming method that loads compressed images from disk and recompresses them using a GPU format.[10] Barb presented a method that weights mip levels by importance based on the covered screen space when rendered from a probe's position.[11]

Clipmaps Streaming whole mipmap levels can still require too much memory, which is often encountered when visualizing aerial scanned terrain data. Such datasets are typically rendered with a fixed-perspective viewport that shows a rectangular section of the dataset. Tanner *et al.* propose *clipmaps* that keep only a fixed-size clipping area in memory.[12] While the original implementation relies on specialized hardware, Makarov describes how to implement clipmaps on commodity hardware using array textures.[13] While clipmaps work well for geographic visualization applications, they are not suited for general scenes since only a single contiguous region of data is kept in memory.

Virtual Texturing A generalization of clipmaps and texture streaming is virtual texturing. Similar to virtualized memory in modern operating systems, textures are partitioned into fixed-size pages and stored in a page pool. Accesses are mapped via an indirection table that translates virtual into physical texture coordinates on a per-page basis.

Lefebvre *et al.* propose a virtual texturing system that marks required pages by rendering texture coordinates into a framebuffer that maps each fragment to a page.[14] Van Weveren describes virtual texturing in the game engine *id Tech 5* that determines required pages by rendering the scene itself into a framebuffer and reduces visible LOD popping by fading in newly loaded mipmaps.[15,16] Mittring *et al.* investigate streaming from slow storage devices and efficient page pool updates.[17] Hollemeersch *et al.* propose GPU computing for performance improvements such as flattening the page ID buffer to reduce the transfer time, or device-side updating of the indirection table.[18]

Contemporary GPU architectures support virtualized resources that are not limited to 2-D textures, but support 1-D textures, cubemaps, volume textures, and general buffers. The indirection tables are opaquely implemented in hardware, which makes texture accesses much simpler and removes the need for an additional texture fetch.[19] Our proposed method leverages hardware support for virtual texturing using the Vulkan graphics library.

ELLIPSOID SPLATTING

Planar splats approximate locally flat surfaces well. Sparsely sampled, long and thin

objects, however, are challenging to represent with existing point-based rendering primitives. Discs and ellipses struggle to approximate high local curvature because of their 2-D nature. To render such objects, a large amount of sample points from several viewing angles is required, which is prohibitive considering the small contribution to the overall scene. To address this issue, we propose 3-D ellipsoids as a novel point-based rendering primitive for improved surface approximation. The piecewise quadratic surface elements reproduce high-curvature objects with much fewer sample points and significantly improve their visual quality.

Fitting Ellipsoids to Point Cloud Data

To better approximate surfaces, we fit an ellipsoid to the local neighborhood of each point in a preprocessing step. This involves finding the ellipsoid's center $\mathbf{c}$ and three axes $\mathbf{u}, \mathbf{v}, \mathbf{w}$, which we determine as the axes of maximum variation of a principal component analysis.[20]

Let P be a set of sufficiently dense sample points of a surface S. For each point $\mathbf{p}_i \in P$, the algorithm for computing the axes of an ellipsoid comprises the following steps.

1) Find the neighborhood $N = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$.
2) Compute the mean $\bar{\mathbf{p}} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{p}_i$.
3) Compute the covariance matrix $\mathbf{C}$ of N.
4) Compute the eigenvalues λ_i and the eigenvectors $\mathbf{e}_i$ of $\mathbf{C}$.

Common approaches to define the neighborhood are to use the k-nearest neighbors (k-NN) or all points within a certain radius ε. The advantage of the k-NN method is that exactly k points are in the resulting set. However, outliers can be included in the set if not enough points are in close proximity. We choose to first include all points in the ε-neighborhood for a predefined ε. If fewer than $N_{\min}$ points are in the set, we inspect the k-NN set for $k = N_{\min}$. Points are classified as outliers and removed from the set if their distance to $\mathbf{p}_i$ is larger than $\delta > \varepsilon$. If the resulting number of points n satisfies $N_{outlier} \leq n < N_{\min}$, the k-NN set of $N_{\min}$ points is used, otherwise $\mathbf{p}_i$ is discarded. This procedure yielded robust results with the parameters $\varepsilon = 0.02$, $\delta = 0.06$, $N_{\min} = 6$, $N_{outlier} = 3$, where ε and δ depend on the dataset scale and average sampling density.

The covariance matrix is then computed as

$$\mathbf{C} = \frac{1}{n-1}\sum_{i=1}^{n}(\mathbf{p}_i - \bar{\mathbf{p}})(\mathbf{p}_i - \bar{\mathbf{p}})^{\mathsf{T}}$$

and can be interpreted as a transformation of the unit sphere to an ellipsoid that matches the shape of the data distribution. This symmetric positive definite matrix can be factorized as $\mathbf{C} = \mathbf{U}\mathbf{\Sigma}\mathbf{U}^{\mathsf{T}}$, where $\mathbf{\Sigma} = \mathrm{diag}(\lambda_1, \lambda_2, \lambda_3)$ contains the eigenvalues of $\mathbf{C}$ and $\mathbf{U} = [\,\mathbf{u_1} \quad \mathbf{u_2} \quad \mathbf{u_3}\,]$ is a rotation matrix. The eigenvalues of $\mathbf{\Sigma}$ correspond to the variance σ^2 along each axis. We choose the axes as $\mathbf{u} = 3\sqrt{\lambda_1}\mathbf{u_1}$, $\mathbf{v} = 3\sqrt{\lambda_2}\mathbf{u_2}$, $\mathbf{w} = 3\sqrt{\lambda_3}\mathbf{u_3}$ to contain 99.7% of the points along each axis inside the ellipsoid.

Rendering

Our method for rendering ellipsoids improves the quadrics splatting method presented by Sigg *et al.*[21] by perspectively correct projective texture mapping using photographs taken from 3-D scanner positions. We furthermore improve rendering performance by utilizing geometry shaders for primitive instantiation.

The authors define $\mathbf{T}$ as the *variance matrix* that transforms from parameter space to object space. In parameter space, the ellipsoid becomes the unit sphere, which can be exploited for rendering. The matrix $\mathbf{T}$ is defined in terms of the three basis vectors $\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$ and the ellipsoid center $\mathbf{c}$ as

$$\mathbf{T} = \begin{pmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{c} \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

As with all splatting techniques, geometric primitives need to be rendered to trigger the fragment shader for the respective area of the framebuffer. One way of doing this is to render a quad with UV coordinates (u, v) ranging from $(-1, -1)$ to (1,1). The shader then discards all fragments for which $u^2 + v^2 > 1$ and, thus, lie outside of the ellipsoid's projection. The quad needs to face the camera and be sized such that it acts as a bounding box of the ellipsoid. This can be achieved by first computing the basis vectors $\mathbf{x}$, $\mathbf{y}$ and the offset $\mathbf{z}$ of the camera-facing quad in parameter space as

$$\mathbf{z} = \frac{(\mathbf{VMT})^{-1}[\,0 \quad 0 \quad 0 \quad 1\,]^\mathsf{T}}{\|(\mathbf{VMT})^{-1}[\,0 \quad 0 \quad 0 \quad 1\,]^\mathsf{T}\|}$$
$$\mathbf{x} = \frac{[\,0 \quad 1 \quad 0 \quad 0\,]^\mathsf{T} \times \mathbf{z}}{\|[\,0 \quad 1 \quad 0 \quad 0\,]^\mathsf{T} \times \mathbf{z}\|}$$
$$\mathbf{y} = \mathbf{z} \times \mathbf{x}$$

where $\mathbf{V}$ and $\mathbf{M}$ are the view and model matrix. The quad vertices in parameter space can then be transformed to object space for rendering as

$$\mathbf{v}_i = \mathbf{T}(u_i\mathbf{x} + v_i\mathbf{y} + \mathbf{z}).$$

Applying $\mathbf{z}$ as an offset in parameter space moves the quad toward the camera such that the depth of the fragments of the quad serve as a lower bound for the depth of the ellipsoid. Compared to positioning the quad at the center of the ellipsoid, this reduces depth misordering when ellipsoids of different sizes overlap in screen space.

One of the most performance-critical parts when rendering a large number of splats is vertex processing. To improve the performance, we draw a single triangle instead of a quad, with the following UV coordinates:

$$\mathbf{a} = \left[\,-\sqrt{3} \quad -1\,\right]^\mathsf{T}, \qquad \mathbf{b} = [\,0 \quad 2\,]^\mathsf{T}, \qquad \mathbf{c} = \left[\,\sqrt{3} \quad -1\,\right]^\mathsf{T}.$$

These coordinates form an equilateral triangle with an edge length of $2\sqrt{3}$ that circumscribes the unit circle. This optimization works for disc and ellipse splatting as well. A disadvantage is that about 30% more fragments have to be rasterized. However, most fragments are discarded early and in practice the performance gain from the reduced vertex count outweighs the cost of additional fragments on modern hardware.

In contrast to discs and ellipses, the per-fragment depth and normal cannot be interpolated by the hardware based on vertex attributes. In the special case of ellipsoids, both can be derived from the UV coordinates by projecting onto the unit sphere in parameter space: $\mathbf{p}_{\text{par}} = u\mathbf{x} + v\mathbf{y} + \sqrt{1-u^2-v^2}\mathbf{z}$. The world space normal $\mathbf{n}_{\text{world}}$ and depth d can then be computed as follows:

$$\mathbf{n}_{\text{world}} = (\mathbf{MT})^{-\mathsf{T}}\mathbf{p}_{\text{par}}$$
$$\mathbf{p}_{\text{clip}} = \mathbf{PVMT}\mathbf{p}_{\text{par}} = [\,x_{\text{clip}} \quad y_{\text{clip}} \quad z_{\text{clip}} \quad w_{\text{clip}}\,]^\mathsf{T}$$
$$d = \frac{z_{\text{clip}}}{w_{\text{clip}}}$$

with $\mathbf{P}$, $\mathbf{V}$, and $\mathbf{M}$ being the projection, view, and model matrix, respectively.

Texturing is performed using photographs that were taken at the 3-D scanner positions. As with the per-fragment depth and normal, we cannot rely on bilinear interpolation provided by the hardware. To color each fragment correctly, we project the world position into the image plane of the associated scanner view. The texture coordinates (s, t) are computed as follows:

$$\mathbf{p}_{\text{proj}} = \mathbf{K}[\,\mathbf{R} \quad \mathbf{t}\,]\mathbf{MT}\mathbf{p}_{\text{par}} = [\,x_{\text{proj}} \quad y_{\text{proj}} \quad z_{\text{proj}}\,]^\mathsf{T}$$
$$[\,s \quad t\,]^\mathsf{T} = \left[\,\frac{x_{\text{proj}}}{z_{\text{proj}}w} \quad \frac{y_{\text{proj}}}{z_{\text{proj}}h}\,\right]^\mathsf{T}$$

where $\mathbf{K} \in \mathbb{R}^{3\times3}$, $\mathbf{R} \in \mathbb{R}^{3\times3}$, and $\mathbf{t} \in \mathbb{R}^3$ are the intrinsic camera matrix, rotation, and translation of the camera view, and w and h are the image dimensions.

VIRTUAL TEXTURING

In the following, we detail our virtual texturing system. It renders texture datasets much larger than the available graphics memory with a page-based streaming approach. Perceivable texturing artifacts are minimized by a page prediction heuristic based on the user's motion.

The key differences to previous work[16,18] are:

- rendering and determination of the required virtual texture pages in a single draw pass;
- a page prediction heuristic to prefetch pages;
- hardware support from modern GPUs.

An overview of the system architecture is given in Figure 1. The *residency manager* keeps track of all pages and updates the respective GPU resources every frame. The *renderer* uses the resources to render the scene and to determine required pages. These are marked in the *feedback buffer* that the *feedback resolver* then flattens into a linear array of page IDs to be processed by the residency manager. Pages that are not yet in system memory are requested from the *page loader* that fetches pages from the *pagefile*.

Rendering

Our rendering pipeline adapts the deferred shading approach by Botsch *et al.*[5] and performs texturing in the attribute pass. Image data from

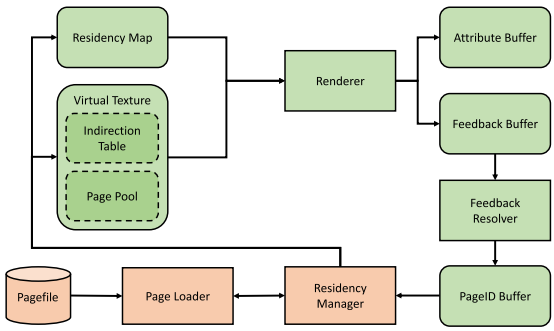

Figure 1. Overview of the virtual texturing system. System components are represented as rectangles and resources as rounded rectangles. CPU-side elements are colored in orange and GPU-side elements in green.

all 3-D scanner photographs is stored in an array texture with one layer per view. Textures are sampled via sparse partially resident images of the Vulkan API. Virtual address translation is performed in hardware, and texture filtering, such as anisotropic and trilinear filtering, is performed transparently.

Texture fetches from nonresident pages, which are not mapped to physical memory, result in undefined behavior. Such fetches have to be avoided and prior knowledge of which pages are resident is required. This is provided by the *residency map* in Figure 1. It is a 2-D array texture with the same amount of layers as the virtual texture and keeps track of the most detailed resident mip level per page. This is used to clamp the minimum mip level during sampling.

We designate a "mip tail" of pages that are always resident as a fallback when no higher quality pages are mapped. This is especially important when the view changes quickly and the system cannot load map pages in time for rendering. Our system keeps all mipmaps smaller than the hardware-defined page size permanently in memory. This guarantees that some low-detail texture information can always be presented, while the memory overhead is negligible at a hardware page-size of 128×128 for current GPU architectures.

We improve rendering performance by sorting all splats by their texture layer. This reduces texture cache misses, since splats close to each other that are mapped to the same layer will likely sample the same virtual texture pages.

Figure 2. Example of *LOD popping*. The marked region on the left appears to be blurred because no high-detail mipmap is resident for that page. In the next frame (on the right), the mipmap is available and the blurred region suddenly disappears. This effect can be very noticeable for users.

Required Page Feedback

There is no obvious correlation between the splat geometry and virtual texture pages. Consequently, the system relies on feedback from the rendering pipeline to determine which pages are required. In the attribute pass, the IDs of required pages are computed and marked in the *feedback buffer*.

A difficulty with blended splat rendering is that multiple pages can be required per fragment. We, therefore, allocate a buffer that holds an integer for each virtual texture page, which contains all information that is used for the prioritization of page uploads. Required pages are marked in the buffer by writing the respective entry nonatomically. While nearby fragments write slightly different values to the feedback buffer, we find that the resulting chance for nonoptimal upload orders is vastly outweighed by the performance gain of avoiding an atomic operation.

Similar to Hollemeersch *et al.*,[18] the feedback resolver performs a compute shader pass that reduces the buffer to a fixed-sized linear array containing all required pages, the *page ID buffer*. The buffer is asynchronously read back by the CPU for further processing by the residency manager.

Page Prediction

Simply loading pages after they were seen causes noticeable artifacts since newly available mipmap levels suddenly pop into view. This so-called "LOD popping" is illustrated in Figure 2 and can be very irritating to users, affecting the immersive experience. Our virtual texturing system minimizes such artifacts by prefetching

pages before they are required for rendering. We employ several heuristics for the prediction of required pages.

One cause of perceivable LOD popping is forward movement. When getting closer to surfaces, increasingly detailed mipmap levels are required. A simple but effective solution is to apply a negative bias to the required mip levels.[8] The system, thus, detects more detailed mipmap levels before they are required for rendering.

In practice, camera rotation and sideways movement often cause more LOD popping than moving forward. Different parts of the scene move into the camera's field of view (FOV), causing completely different textures to be required for rendering. Without prediction, pages will be loaded too late and perceivable popping occurs. Camera motion prediction can be employed to counteract this.[8] Performing true camera motion prediction, however, usually requires rendering a second time with the predicted camera transformation in addition to normal rendering.

We propose to use an *extended frustum* for both rendering and required page prediction in a single pass. The scene is rendered with an enlarged FOV to see pages close to the border before they are needed. Only the part of the viewport that represents the camera's original FOV is then textured, shaded, and presented to the user.

Increasing the size of the frustum evenly on all sides, however, leads to inefficient use of fragments. Generally, the sides of the frustum facing the movement direction are more important, since pages appearing along the camera trajectory are likely to be required for rendering in the next frames. Fragments can be used more efficiently by employing a generalized camera frustum[22] illustrated in Figure 3. We apply a heuristic that maps the angular velocity ω and translational velocity v to the inner viewport's offset $(\Delta x, \Delta y)$ from the extended frustum center as

$$\Delta x = \mathrm{clamp}\left(0.5 \cdot \left(\frac{v_x}{v_{\max}} + \frac{\omega_y}{\omega_{\max}}\right), -1, 1\right) * m_x$$

where $v_{\max}$ and $\omega_{\max}$ are the translational and rotational velocity that cause maximum displacement, m_x denotes the margin size in pixels, and Δy follows analogously.

More elaborate human motion models can be applied at this point. We observe, however, that this simple heuristic already yields satisfying results in our test cases.

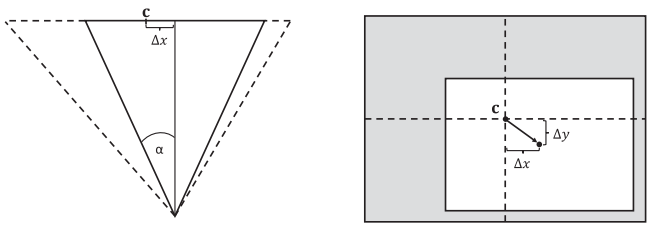

Figure 3. Left: Top view of the extended camera frustum. The dashed lines represent the off-axis generalized frustum that extends the symmetric frustum of the visible viewport. Right: The gray region marks the margin area, while the white region contains the final rendered image.

Physical Page Updates

An overview of the physical page updating process performed by the residency manager is given in Figure 4. Virtual texture pages need to be tracked to know which pages have to be loaded from the pagefile, can be mapped to a physical page, or be evicted from the page pool to free up space for other pages. Not all pages are equally important and, thus, have a priority assigned to them. The system tracks all pages in several priority-sorted lists corresponding to the possible states: *seen*, *loading*, *loaded*, *mapped,* or *evicted*.

After processing the page ID buffer, a fixed number of pages with highest priority are fetched from the pagefile. Fetching pages happens asynchronously to avoid blocking, which usually takes several frames. Next, space in the page pool is allocated for a fixed number of high-priority pages, which also involves deallocating low-priority pages. Afterwards, the virtual to physical page mappings in the GPU's translation table are

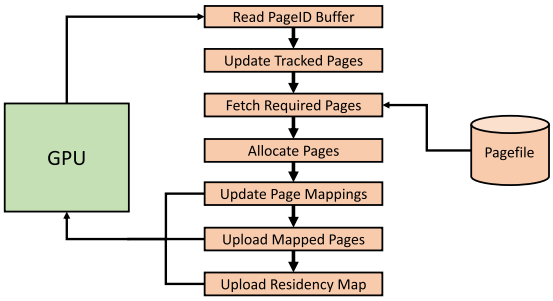

Figure 4. Overview of the physical page update process. CPU-side elements are colored in orange and GPU-side elements in green.

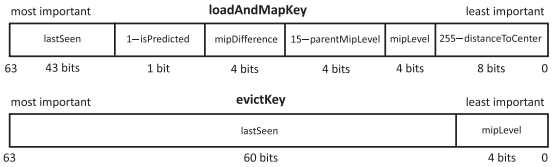

Figure 5. Bit layouts of the sort keys for loading and eviction of virtual texture pages.

updated and the physical page data are uploaded. Lastly, the updated residency map is uploaded to the GPU. All of these steps are performed once per frame.

Page Prioritization We propose different sort keys to compute the priorities of tracked pages depending on their state. Seen and loaded pages are sorted based on how much value they add to the rendered image. Mapped pages are sorted based on their "unimportance" to evict pages from the pool that are least likely to be needed for rendering. The respective sort keys are combined into a 64 bit priority code, illustrated in Figure 5. The seen and loaded pages are sorted using the *loadAndMapKey* in an increasing order, while already mapped pages are sorted using the *evictKey* in a decreasing order.

Recently seen pages are most likely to be required for rendering in the next frames. Similarly, least recently seen pages should be evicted first. On top of this, pages that are only required due to prediction are loaded after pages that are currently required for rendering. Coarse mip levels are loaded and mapped first, while the most-detailed mip levels are evicted first, which ensures filtering across levels works at all times. Pages are further prioritized by the difference between their mip level and the corresponding resident mip level. If the difference is greater, the image is potentially improved more by loading the page. Pages that are closer to the screen center are more likely to be required in the next frames, in contrast to pages near the edge of the screen that are more likely to be out of sight.

Loading From the Pagefile Before virtual pages can be mapped to physical pages, the texture data needs to be loaded from the pagefile. It contains the precomputed mipmap levels for all textures, split into pages and compressed individually, and is prefixed by a table that contains the per-page size and file offset. Pages are compressed using a conventional image format, such as PNG or JPEG, to reduce the memory footprint and required bandwidth for reading during runtime.

Page loading should not stall the pipeline and is performed asynchronously. Performance strongly depends on the underlying storage medium. A fast solid-state drive (SSD) is advisable to achieve high throughput. Load requests are pushed into a thread-safe queue and handled by a pool of worker threads that push decompressed pages into a second queue to be consumed by the residency manager. A considerable bottleneck is the nondeterministic cost of memory allocation. We solve this with a custom allocator that reuses page-sized chunks from a preallocated pool.

RESULTS AND DISCUSSION

In the following, we present the results of our ellipsoid splatting technique and virtual texturing system.

Ellipsoid Splatting

Results of our ellipsoid splatting technique are shown in Figure 6. The first row shows disc-shaped splats that are rendered with a globally uniform splat size and cliplines. Object contours are either overestimated by splats that are larger than geometric features, or the surface appears fragmented due to an insufficient local sampling density. The staircase edge in the foreground of the left image shows that cliplines are well suited to preserve sharp edge features and that the texture quality is improved. This is due to a better fit of the primitives to the planar sides of the staircase, which results in less blended fragments and, consequently, reduced ghosting and blurring.

The second row illustrates 2-D ellipse-shaped splats. While they adapt to the local sampling density and size of geometric features, their planar 2-D appearance becomes apparent. Our 3-D ellipsoid splats in the third row do not exhibit this problem and yield a plausible appearance from any viewing direction. The current lack of a clipping primitive, however, degrades texture quality in planar regions.

Figure 6. Comparison of splatting primitives: Uniform-sized disc splats in the first row illustrate the difficulty to find an adequate global splat size. 2-D ellipses in the second row improve the result, but fail to capture object contours from all viewing angles. Finally, our 3-D ellipsoid splats in the third row achieve a plausible object contour from all viewing angles.

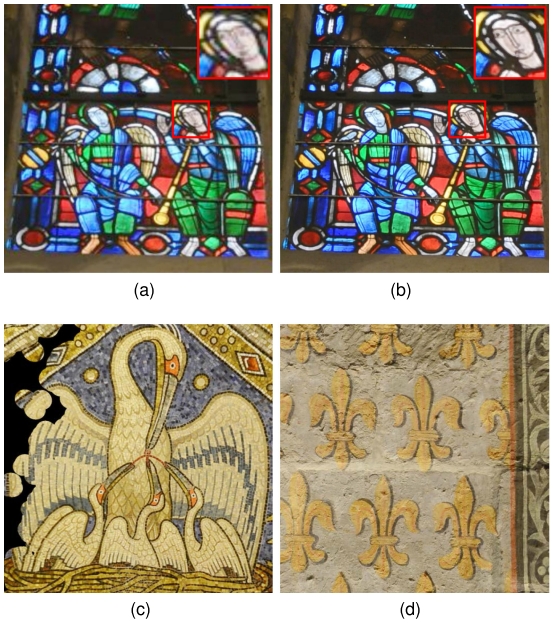

Figure 7. Examples of the achievable visual detail by texture upsampling. (a) Original 8 k textures. (b) Upsampled 32 k textures. (c) Mosaic of 110 cm. (d) Wall patch of 65 cm.

Virtual Texturing

Our virtual texturing system was evaluated regarding both visual quality of the result and rendering performance. We describe the measurement setup and evaluation criteria before we present the results in the respective sections. An impression of the achievable image detail is given in Figure 7.

Evaluation Methods To evaluate different aspects of the system, camera trajectories were recorded from real-world motion with a tracked VR headset. The following four representative paths were recorded:

1) *look around:* head rotation without translational motion or disocclusion;
2) *forward:* constant forward motion;
3) *backward:* constant backward motion;
4) *sideways:* sudden disocclusion, sideways motion past an obstacle revealing large parts of the scene.

We quantify visual quality in terms of virtual texture *page misses*, since perceivable LOD popping can occur whenever a page is required for rendering but not resident in memory. System performance is evaluated in terms of frame timings.

Our test dataset is a high-quality laserscan of the Aachen Cathedral interior. It consists of over 70 million points and 443 photographs each with a resolution of 7360 $\times$ 4912. While this is already a large texture dataset, our goal is to provide maximum visual detail, even if users in an IVE approach objects very closely. Therefore, we applied a state-of-the-art super-resolution technique based on generative adversarial networks to upscale the individual images to a resolution of 29 440 $\times$ 19 648.[23] While the improved image detail should not be mistaken for actual detail, it significantly improves the perceived quality for exploration in IVEs and demonstrates the feasibility of our approach for datasets of much higher resolution than our current data

Table 1. Average number of page misses for each configuration.

	Look Around	Forward	Backwards	Sideways
nothing:	1.00	1.00	1.00	1.00
bias:	0.82	0.33	0.97	1.09
frustum:	0.21	1.06	0.13	0.91
combined:	0.07	0.44	0.15	1.17

acquisition pipeline provides. The corresponding pagefiles contain 79.35 GB and 1269.61 GB of uncompressed texture data, respectively.

In the following, we denote the original dataset as 8 k and the upsampled dataset as 32 k. If not explicitly noted otherwise, performance measurements in this section are given for the much more challenging 32 k dataset. With the proposed prediction methods, the original 8 k dataset could be reproduced with virtually no page misses at all.

The scenes were rendered in stereo with an effective screen resolution of 1440 × 1600 per eye. A prediction margin of 15% and a constant mip bias of −0.5 were used. All tests were performed on a Debian GNU/Linux system with the following specifications: Intel Core i7 4770, 16 GB RAM, Nvidia Geforce GTX 1080 8 GB (driver version 418.74), Samsung 840 EVO 250 GB SSD.

Page Prediction To evaluate the amount of visible artifacts caused by LOD popping, Table 1 lists the average number of page misses for each test condition. Four prediction configurations were compared: *nothing* applies no prediction, *bias* applies a constant negative mip bias, *frustum* uses the extended frustum from the "Page Prediction" section, and *combined* applies both prediction heuristics. All values are given relative to the *nothing* condition as a baseline.

System Performance The rendering cost incurred by our proposed methods is illustrated in Figure 8. The top figure shows the GPU-side overhead for 3-D ellipsoid splatting compared to 2-D disc and ellipse splatting without virtual texturing. The bottom figure shows the additional cost of frustum page prediction for the virtual texturing system.

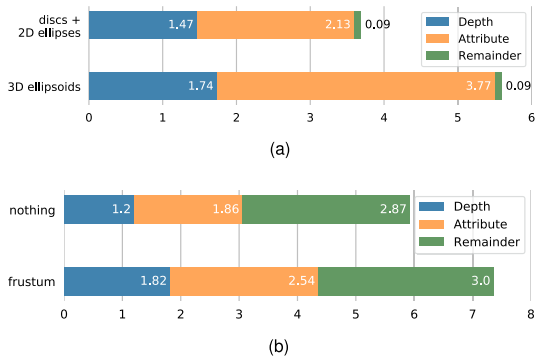

Figure 8. GPU performance overhead of the proposed methods (in ms). For technical details about the different render passes refer to the article by Botsch *et al.*[5] (a) Rendering cost of 3-D ellipsoids versus 2-D discs or ellipses. (b) Rendering cost of frustum prediction with required page feedback and texture page upload in the remainder pass.

To evaluate overall system performance, we present absolute frame timings for the two different texture datasets in Figure 9. The CPU frame times amount to the processing steps illustrated in Figure 4, excluding read operations from disk, which are performed asynchronously. The GPU frame times include rendering and reduction of the feedback buffer in addition to the data transfer times.

Discussion

We discuss our results regarding the achievable visual quality and overall system performance. Finally, we give a brief report of feedback we obtained from visitors in a public museum exhibition.

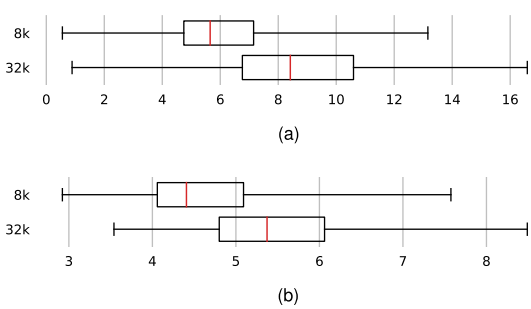

Figure 9. Distribution of frame timings in milliseconds over all configurations. The red marker denotes the median, the box represents quartiles, and the whiskers the extrema of the distribution. (a) CPU frametimes (in ms). (b) GPU frametimes (in ms).

Visual Quality Our ellipsoid fitting and splatting approach works particularly well for sparsely sampled, long and thin objects, as can be seen in Figure 6. For these kinds of objects, the cliplines for disc splatting cannot be reliably estimated. The splat size, thus, commonly exceeds the object contours and produces noticeable texturing artifacts. Due to their 3-D nature, ellipsoids approximate objects such as ropes, candles, and chains much more accurately using a smaller number of primitives.

Even more importantly, a 3-D ellipsoid gives rise to a plausible contour from all viewing angles. This becomes particularly apparent when moving around objects. Discs and ellipses reveal their planar shape, while 3-D ellipsoids appear as consistent volumetric objects, which makes close-up views appear much more realistic. This significantly improves the plausibility and, in our judgement, the illusion of presence in the reconstructed environment.

Objects with sharp but planar features that can be approximated well using cliplines are, however, not reproduced as precisely by the rounded 3-D ellipsoids. This can be observed in the left column of Figure 6. Therefore, we suggest to use a hybrid method that uses the proposed 3-D ellipsoids for thin and long objects, and the well-established disc and ellipse splats with cliplines for planar regions of the dataset. Implementation of a suitable heuristic to choose between the two primitives on a point-by-point basis remains a topic for future research.

Evaluation of the prediction heuristics in Table 1 shows that by using the extended camera frustum in combination with a negative mip bias, the number of texture page misses, and thus visual artifacts in the form of LOD popping, can be significantly reduced for the most common cases.

In the less frequent case where large parts of the scene are disoccluded at once, such as in the sideways test, the prediction can have a slightly negative effect. Too many pages need to be mapped and loaded, such that the overhead of prematurely loading pages that are eventually not needed outweighs the potential benefit. Our implementation can be improved in this regard by carefully flushing the loading queues during all stages. Currently, a loading job that has once started will occupy resources in the pipeline that could potentially be allocated for higher-priority pages.

Performance Our virtual texturing system with both page prediction heuristics enabled can maintain a constant update rate of 90 frames per second required for today's VR headsets. As evident in Figure 9, the GPU frametimes were consistently below 11 ms even for the high-resolution texture set.

Rendering 3-D ellipsoids compared to planar primitives is about 50% more expensive, as shown by the GPU timings in Figure 8(a). Real scenes, however, typically contain large planar areas that can be rendered with disc-shaped splats just as effectively. The additional performance cost to render the typically small subset of thin and elongated objects as 3-D ellipsoids is negligible given the overall improvement in visual reproduction accuracy.

Furthermore, Figure 8(b) shows that the performance cost of page prediction using an extended frustum is very moderate. Considering the substantial reduction in page miss artifacts, which can be observed in Table 1, the improvement in visual fidelity is certainly worth the additional effort.

Overall, the performance of our point-based rendering and virtual texturing system has proven sufficient to render very challenging scenes with massively detailed texture datasets in VR without sacrificing reproduction accuracy.

Audience Reception We had the fortunate opportunity to present our work to the public at the German museum exhibition "Thrill of Deception. From Ancient Art to Virtual Reality" at Ludwig Forum Aachen. It attracted over 30 000 visitors during a four-month period, and we received very positive informal audience feedback. People were excited to see the potential of contemporary VR technology and were thoroughly impressed by the quality and visual detail that can be achieved on consumer-grade hardware.

LIMITATIONS AND FUTURE WORK

In the following, we outline limitations of our proposed method as well as potential future research opportunities.

For an effective hybrid rendering approach, a robust determination of how well a splat neighborhood can be approximated using 2-D ellipses and cliplines rather than 3-D ellipsoids is necessary. Outlier points need to be reliably characterized to enable an approximation with better feature preservation. Furthermore, large features might be approximated with fewer splats using a region growing approach that fits ellipsoids to neighborhoods of increasing size.

Further improvements can be made regarding rendering quality by reconsidering the choice of blending weights for the final shading pass. While the current approach uses only the fragment's distance to the splat center, more sophisticated blending methods could reduce ghosting or smearing artifacts that sometimes become apparent. The depth of the blended fragment along the view ray could be considered, using a heuristic similar to McGuire and Bavoil,[24] or fragments with more reliable image data could be preferred to better preserve sharp texture features.

Emerging VR headsets enable tracking of the user's eye movement. Foveated rendering can be implemented as presented by Guenter *et al.*,[25] and more fine-grained prioritization of virtual texture pages is possible. In combination with methods that estimate the perceived change in visual quality caused by mapping or eviction, this could achieve faithful reproduction of even larger datasets, without requiring additional bandwidth or rendering performance.

CONCLUSION

We presented a method for the immersive visualization of 3-D scanned datasets with massive amounts of texture data. Using the presented techniques, cultural heritage sites can be reproduced in VR with minimal loss of fidelity.

To better approximate the shape of thin and sparsely sampled objects, we presented 3-D ellipsoids as a novel point-based rendering primitive. For certain types of objects, the visual quality we could achieve is superior to prior methods, especially when moving freely around objects in an IVE.

We implemented a virtual texturing system that leverages modern hardware capabilities and was carefully optimized to meet our performance requirements. To further improve the rendering quality while moving through the environment, a virtual texture page prediction heuristic was proposed. With the presented solution, we are able to render terabyte-scale texture datasets at interactive frame rates without compromising visual detail.

ACKNOWLEDGMENTS

This project was supported in part by the European Regional Development Fund, within the Terra Mosana Interreg Euregio project, and in part by the security research program "Anwender-Innovativ: Research for Civil Security II" of the German Federal Ministry of Education and Research, within the "VirtualDisaster" project. The authors would like to thank Scasa[26] for their work on digitally reconstructing the Aachen Cathedral.

REFERENCES

1. M. Levoy and T. Whitted, "The use of points as a display primitive," Comput. Sci. Dept., Univ. North Carolina at Chapel Hill, Chapel Hill, NC, USA, Tech. Rep. 85-022, 1985.
2. H. Pfister, M. Zwicker, J. Van Baar, and M. Gross, "Surfels: Surface elements as rendering primitives," in *Proc. 27th Annu. Conf. Comput. Graph. Interactive Techn.*, 2000, pp. 335–342.
3. M. Zwicker, H. Pfister, J. Van Baar, and M. Gross, "Surface splatting," in *Proc. 28th Annu. Conf. Comput. Graph. Interactive Techn.*, 2001, pp. 371–378.
4. M. Botsch, M. Spernat, and L. Kobbelt, "Phong splatting," in *Proc. 1st Eurograph. Conf. Point-Based Graph.*, 2004, pp. 25–32.
5. M. Botsch, A. Hornung, M. Zwicker, and L. Kobbelt, "High-quality surface splatting on today's GPUs," in *Proc. Eurograph./IEEE VGTC Symp. Point-Based Graph.*, 2005, pp. 17–141.
6. M. Zwicker, J. Räsänen, M. Botsch, C. Dachsbacher, and M. Pauly, "Perspective accurate splatting," in *Proc. Graph. Interface*, 2004, pp. 247–254.
7. R. Preiner, S. Jeschke, and M. Wimmer, "Auto splats: Dynamic point cloud visualization on the GPU," in *Proc. Eurograph. Workshop Parallel Graph. Visualization*, 2012, pp. 139–148.
8. J. Blow, "Implementing a texture caching system," *Game Developers Mag.*, vol. Apr., pp. 46–56, 1998.

9. R. Dumont, F. Pellacini, and J. A. Ferwerda, "A perceptually-based texture caching algorithm for hardware-based rendering," in *Rendering Techn.*, 2001, pp. 249–256.
10. J. Van Waveren, "Real-time texture streaming & decompression," ID Software, Inc., Shreveport, LO, USA, Tech. Rep., 2006.
11. C. Barb, Efficient texture streaming in 'titanfall 2'. 2017. [Online]. Available: https://www.gdcvault.com/play/1024418/Efficient-Texture-Streaming-in-Titanfall
12. C. C. Tanner, C. J. Migdal, and M. T. Jones, "The clipmap: A virtual mipmap," in *Proc. 25th Annu. Conf. Comput. Graph. Interactive Techn.*, 1998, pp. 151–158.
13. E. Makarov, "Clipmaps," Whitepaper WP-03017-001, Nvidia Corporation, Feb. 2007.
14. S. Lefebvre, J. Darbon, and F. Neyret, "Unified texture management for arbitrary meshes," INRIA-Rhone-Alpes, Montbonnot-Saint-Martin, France, Tech. Rep. RR-5210, May 2004.
15. J. Van Waveren, "ID tech 5 challenges-from texture virtualization to massive parallelization," *SIGGRAPH Course: Beyond Programmable Shading*, Aug. 2009.
16. J. van Waveren, "Software virtual textures," Id Software LLC, Tech. Rep., Feb. 2012.
17. M. Mittring *et al.*, "Advanced virtual texture topics," in *Proc. ACM SIGGRAPH Games*, 2008, pp. 23–51.
18. C. Hollemeersch, B. Pieters, P. Lambert, and R. Van de Walle, "Accelerating virtual texturing using cuda," *GPU Pro: Adv. Rendering Techn.*, vol. 1, pp. 623–641, 2010.
19. J. Obert, J. M. P. van Waveren, and G. Sellers, "Virtual texturing in software and hardware," in *Proc. SIGGRAPH Courses*, 2012, pp. 1–29. [Online]. Available: https://doi.org/10.1145/2343483.2343488
20. I. T. Jolliffe, *Principal Component Analysis* (Springer Series in Statistics). New York, NY, USA: Springer-Verlag, 2002.
21. C. Sigg, T. Weyrich, M. Botsch, and M. H. Gross, "GPU-based ray-casting of quadratic surfaces," in *Proc. SPBG*, 2006, pp. 59–65.
22. R. Kooima, "Generalized perspective projection," *J. Sch. Electron. Eng. Comput. Sci.*, 2009.
23. W. Yifan, F. Perazzi, B. McWilliams, A. Sorkine-Hornung, O. Sorkine-Hornung, and C. Schroers, "A fully progressive approach to single-image super-resolution," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops*, 2018, pp. 977–97709.
24. M. McGuire and L. Bavoil, "Weighted blended order-independent transparency," *J. Comput. Graph. Techn.*, vol. 2, no. 2, pp. 122–141, 2013.
25. B. Guenter, M. Finch, S. Drucker, D. Tan, and J. Snyder, "Foveated 3D graphics," *ACM Trans. Graph.*, vol. 31, no. 6, pp. 1–10, 2012.
26. Scasa GmbH. 2019. [Online]. Available: https://scasa.eu/

Patric Schmitz is a Researcher with the Visual Computing Institute of RWTH Aachen University. After having received the M.Sc. degree in computer science in 2015, he worked at the Institute for Research and Coordination in Acoustics/Music (IRCAM) at Centre Pompidou in Paris. Back at RWTH Aachen University, he pursues the Ph.D. degree with a major research focus on locomotion in immersive virtual environments, 3-D reconstruction, and realistic realtime rendering. Contact him at patric.schmitz@cs.rwth-aachen.de.

Timothy Blut received the M.Sc. degree in computer science from RWTH Aachen University, Aachen, Germany, in 2019. During his studies, he put a strong focus on computer graphics and related fields. His major research interests include realtime and photorealistic rendering. Contact him at timothy.blut@rwth-aachen.de.

Christian Mattes is currently a Researcher with the Visual Computing Institute, RWTH Aachen University, Aachen, Germany, where he previously received the M.Sc. degree in computer science in 2016. His major research interests include realtime rendering, user interaction in virtual reality, and 3-D reconstruction and visualization. Contact him at mattes@cs.rwth-aachen.de.

Leif Kobbelt is currently a Distinguished Professor of Computer Science with RWTH Aachen University, Aachen, Germany. Since 2001, he is the head of the Institute for Computergraphics and Multimedia. After having received the Ph.D. degree from the Karlsruhe Institute of Technology, he worked at the University of Wisconsin in Madison, the University of Erlangen-Nuremberg, and the Max Planck Institute of Computer Science. His major research interests include 3-D reconstruction, efficient geometry processing, and realistic realtime rendering. Contact him at kobbelt@cs.rwth-aachen.de.