DOI: 10.1111/cgf.14474
EUROGRAPHICS 2022 / R. Chaine and M. H. Kim
(Guest Editors)
*Volume 41 (2022), Number 2*

# Meshlets and How to Shade Them: A Study on Texture-Space Shading

T. Neff[1], J. H. Mueller[1], M. Steinberger[1] and D. Schmalstieg[1]

[1]Graz University of Technology, Institute of Computer Graphics and Vision, Austria

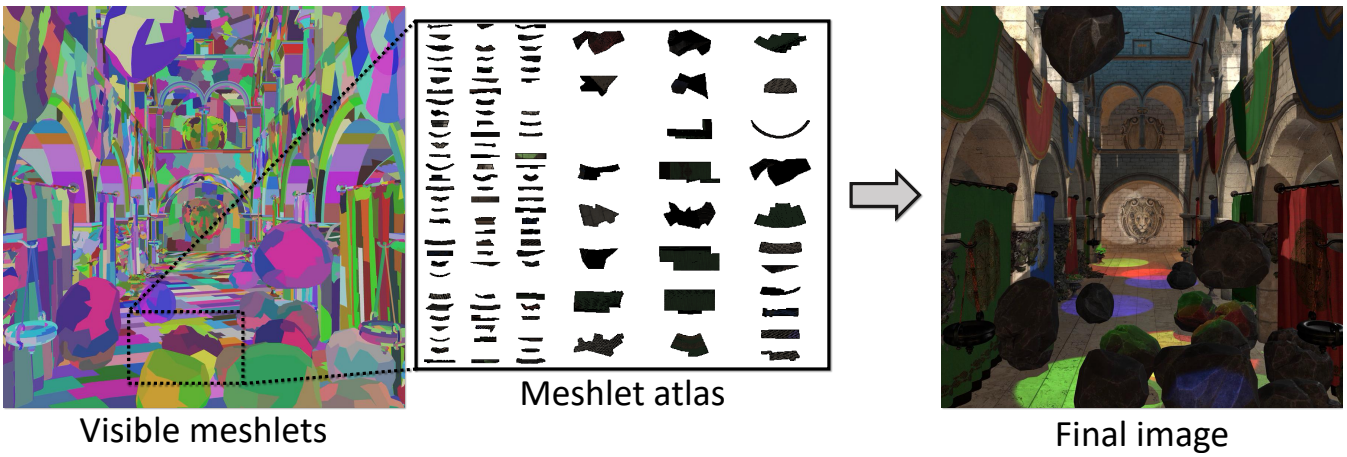



**Figure 1:** *The meshlet shading atlas operates on large groups of primitives by clustering the scene into meshlets (left). For each meshlet, a suitable mapping is used to transform it into texture space and shade it (center). By grouping as many primitives as possible, we can reduce the impact of overshading and better distribute samples in texture space, which results in higher image quality per sample (right).*

**Abstract**

*Commonly used image-space layouts of shading points, such as used in deferred shading, are strictly view-dependent, which restricts efficient caching and temporal amortization. In contrast, texture-space layouts can represent shading on all surface points and can be tailored to the needs of a particular application. However, the best grouping of shading points—which we call a shading unit—in texture space remains unclear. Choices of shading unit granularity (how many primitives or pixels per unit) and in shading unit parametrization (how to assign texture coordinates to shading points) lead to different outcomes in terms of final image quality, overshading cost, and memory consumption. Among the possible choices, shading units consisting of larger groups of scene primitives, so-called meshlets, remain unexplored as of yet. In this paper, we introduce a taxonomy for analyzing existing texture-space shading methods based on the group size and parametrization of shading units. Furthermore, we introduce a novel texture-space layout strategy that operates on large shading units: the meshlet shading atlas. We experimentally demonstrate that the meshlet shading atlas outperforms previous approaches in terms of image quality, run-time performance and temporal upsampling for a given number of fragment shader invocations. The meshlet shading atlas lends itself to work together with popular cluster-based rendering of meshes with high geometric detail.*

**CCS Concepts**

• ***Computing methodologies*** → ***Rendering; Texturing;***

## 1. Introduction

In real-time rendering, the runtime tends to be dominated by the cost of pixel shader invocations. Consequently, many computer graphics techniques aim to reduce shading cost, either by reducing overshading (i.e., avoiding shader invocations that do not contribute to the final image) or by facilitating spatio-temporal reuse of cached shading information. Popular approaches such as temporal anti-aliasing [YLS20] or learned supersampling [Liu20; NVI18] rely on the organization of shading information in image space to reuse shading. However, a shading cache in image space only has two

dimensions, and its sample distribution is view-dependent, making it a poor match for techniques that rely on coherence in object space. For example, global illumination (e.g., Lumen [Epi21a]), stochastic sampling [RLC*11] or disocclusion-free image warping [PZ17] require identifying a 3D neighborhood of a shading point in object space. Besides better support for this class of algorithm, caching in object space enables much longer periods of shading reuse compared to image-space caching [MNV*21].

For these and other reasons, recent research has investigated object-space shading representations. Unlike image space, object space allows for a wide variety of layouts, each with its own set of advantages and disadvantages. Some object-space shading methods index into the cache directly from the 3D coordinates of shading points—for instance, using 3D hashing [RLC*11; LD12]. Alas, this form of indexing precludes the use of hardware features such as texture interpolation, which makes a high-performance implementation difficult. Therefore, most object-space shading methods rely on a *parametrization* of shading point coordinates that maps into a suitably chosen texture space.

All of these *texture-space shading* (TSS) techniques require answering two key questions: (1) How should shading points be grouped into what Mueller et al. [MNV*21] call a *shading unit* (SU), which contains shading points that are processed and stored together? (2) How should the *parametrization* that maps an SU into a given texture space be determined?

The choice of SU grouping and SU parametrization determines the two essential performance characteristics of a TSS technique, namely, *image quality* and *shader invocations*. These characteristics form a trade-off: Image quality depends on the ability to reconstruct a target image from the shading cache. Ideally, the shading cache is filled with exactly the samples required to reconstruct the target image at reference quality (which is usually derived from a super-sampled forward rendering of the same scene).

In general, the ideal sample selection is not known in advance, so oversampling, undersampling, or distortion occurs. *Oversampling* means generating samples at a higher than necessary resolution, generating samples which do not appear in the target image, or generating duplicate samples. Although oversampling can enhance the image quality, it also increases the number of shader invocations and, thus, the cost. Oversampling may also be amortized when more than one final image is reconstructed from a given shading cache. Applications of such multi-view rendering include stereo displays, framerate upsampling with image warping, or game streaming for multiple players. Like oversampling, *undersampling* is also generally undesirable. It reduces the number of shader invocations, but leads to a deterioration of image quality. *Distortion* refers to sample patterns that do not match the final reconstruction sample distribution. Especially angular distortion is bothersome, as it leads to oversampling and undersampling at the same time, even when the number of samples in the shading cache matches the number of samples in the target.

Besides image quality and shader invocations, an additional concern is the *memory footprint*. A mandatory requirement of TSS is that each shading point must have a unique mapping to a texel. If large scenes containing more surface points than would fit into GPU memory are desired, this unique mapping must reside sparsely in texture space, i.e., only the currently visible portion of the scene is mapped to valid cache addresses. This requirement influences the choice of SU parametrization. In particular, sparse textures cannot solely rely on a statically determined UV-parametrization (also known as *pre-charting*). To support sparse residency, pre-charting must always be combined with an online parametrization that determines the final size and placement of an SU in texture space. In this paper, we focus on methods that support sparse residency and the trade-off between image quality and shader invocations, but not on memory efficiency. We assume that the resident set of a scene does not exceed the GPU memory capacity.

The choice of SU grouping is also influenced by the recent trend in GPU programming towards favoring large(r) clusters of primitives [NVI18; Epi21b]. This trend is not considered in previous TSS techniques, which operate on small pixel blocks or on few $(1-3)$ triangles. Small SU group sizes have a poor ratio of interior area to border length, leading to substantial overshading, and require more effort for border handling. A larger SU group size may overcome these difficulties.

Consequently, we introduce a novel TSS method, called meshlet shading atlas (MSA), that operates on clusters of primitives which match the preferred primitive granularity of the GPU hardware. MSA operates on groups consisting of a varying number of spatially coherent triangles, called *meshlets*, and makes use of recent NVIDIA extensions to transform and shade such meshlets. MSA combines the memory management for sparse residency [MVD*18] with an SU configuration that can be customized by choosing the best SU group size (i.e., triangle counts per meshlet) and SU parametrization (pre-charted or derived from the current view). By comparing MSA to previous TSS methods, we make the following contributions:

- We show how MSA extends existing work by classifying it within our texture-space shading taxonomy.
- We address the question how an SU parametrization based on pre-charting compares to an SU parametrization determined based on online perspective projection to the current view.
- We investigate how image quality is influenced by the number of primitives in an SU.
- We show that using large meshlets is advantageous for temporal accumulation, temporal upsampling and run-time performance.
- We demonstrate that MSA outperforms existing methods in terms of quality per shaded pixel.

## 2. Background

In recent years, TSS techniques mainly focused on efficient spatio-temporal caching and reuse of shading information in order to enhance applications such as streaming rendering, virtual reality, or real-time global illumination. New TSS methods have clearly brought advances in sampling efficiency, memory requirements and image quality. Yet, it remains difficult to compare methods and to reason about their specific benefits and drawbacks.

All texture-space rendering systems can be roughly divided into four stages: *visibility* (a.k.a. geometry pre-pass), *memory management*, *shading* and *final image* rendering. Among those, the visibility stage is independent of the other stages, as it only serves to

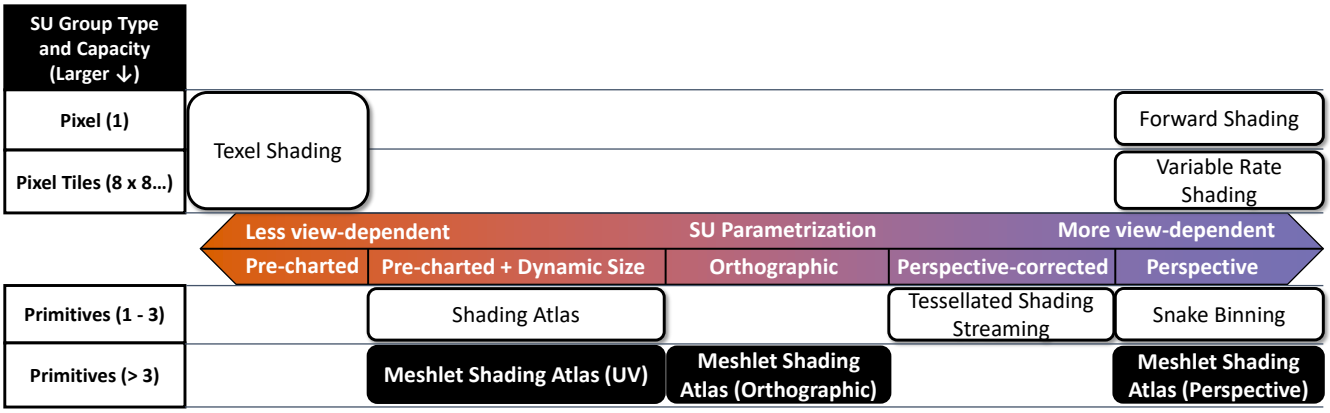



**Figure 2:** *Texture-space methods can be characterized along two dimensions: The SU parametrization can be less or more view dependent, and the SU group size can be larger or smaller. Because we must also distinguish by SU group type (pixel or primitive), the group size is ambiguous – one primitive can be smaller than one pixel, although this is not usually the case. This observation explains why competing TSS methods may or may not agree on performance comparisons, since results are dependent on the choice of primitives in the scene. Our new family of methods, MSA, occupies the previously neglected space with large primitive counts and view-dependence in size or projection.*

determine the relevant portion of the scene. In its simplest form, it can just be an exact visible set of primitives or shading points, but more sophisticated approaches can make use of visibility data structures [HSS19b] or sample a potentially visible set [MVD*18]. Final rendering always samples the shading cache, but can otherwise use arbitrary image synthesis approaches, be it forward rendering, deferred rendering or ray-tracing. Hence, the two remaining stages, *memory management* and *shading*, form the core of TSS techniques, and we will discuss only these stages further here.

We have already identified SU grouping and SU parametrization as the two key parameters that characterize a TSS technique. We use these two parameters as the dimensions of a taxonomy that lets us classify existing techniques and identify gaps in the design space of possible TSS techniques. This taxonomy is illustrated in Figure 2, and we will refer to it in the following discussion.

The SU grouping, arranged along the vertical axis in the figure, can either be done on shading points or on primitives. Both types of SU need to be given a contiguous array of pixels in the shading cache. Hence, we require a suitable SU parametrization from shading points to cache entries. The SU parametrization, arranged along the horizontal axis in the figure, can be roughly subdivided into methods that use pre-charting and methods that use a view-dependent projection. Pre-charting can be completely view-independent, but it can also enforce a pixel budget in the shading cache based on a view-dependent size estimate of the SU. Perspective projection methods are view-dependent by definition, but we can further distinguish methods that use a pure perspective projection and methods that try to strike a compromise between view-independence and view-dependence to facilitate better reuse (labeled "perspective-corrected" and "orthographic" in the figure).

Techniques using *forward rendering* occupy the top right corner, corresponding to pure perspective projections of single pixels in the image space of the final view. This choice has the lowest overhead and no overshading. It is also the configuration used in multi-sample anti-aliasing [Ake93] and temporal anti-aliasing [YLS20]. Multi-sample anti-aliasing can be interpreted as a TSS technique if we count sub-pixel coverage samples as an SU grouping. Along similar lines, temporal anti-aliasing can be interpreted as a TSS technique if we count reverse reprojection [NSL*07] as an SU parametrization. Variable-rate shading (VRS) [HGF14] is another technique relying on pure perspective projection, applied on an SU group size slightly larger than a single pixel (e.g., $16 \times 16$ pixels). It adapts the number of shader invocations across image space according to a user-specified value per screen tile. VRS is useful in applications such as lens-distortion compensation for virtual reality headsets or content-aware blurring.

*Pre-charting* is favorable if the scene has been authored using globally non-overlapping texture coordinates. In our taxonomy, pre-charting occupies the spot on the top left. The most prominent TSS technique in this space is texel shading (TS) [HY16], which operates on tiles (i.e., SU groupings) of $8 \times 8$ pixels in the pre-charted UV space. A first geometry pass determines which shading points are visible and marks the corresponding SU and mip level. A second pass determines the shading for the marked SU set, stored at the right mip level. A third pass computes a final image by sampling the shading cache with trilinear interpolation. To handle trilinear interpolation correctly, the shade queuing pass needs to account for borders in UV islands by using conservative rasterization, and it needs to mark all tiles that will be required by the trilinear filter during display. The pre-charting of TS consumes a large amount of memory, especially if the texture-space needs to be unwrapped onto multiple layers for overlapping UV maps. However, hardware support for sparse textures could potentially be utilized to reduce memory pressure. Although there is considerable reuse between samples, the main cause of overshading in TS is that each sample with trilinear interpolation can potentially touch eight tiles. This results in more overshading, even if only those texels would be shaded that are absolutely necessary to generate the final image.

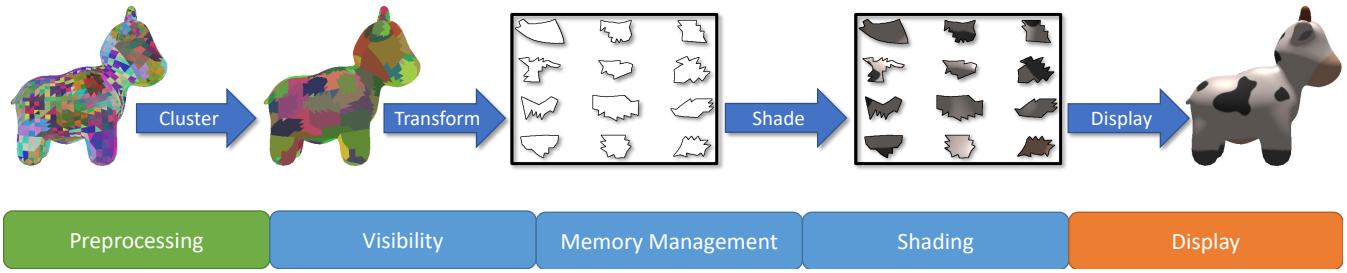



**Figure 3:** *The MSA pipeline consists of* 5 *stages. First,* preprocessing *clusters large coherent groups of primitives into meshlets. During run-time,* visibility *is determined per meshlet. Then,* memory management *is performed by considering space inside the texture atlas based on the screen-space size of each meshlet. Afterwards, meshlets are* shaded *using mesh shaders and the previously determined parametrization. Finally, the* display *stage can sample independently from the atlas. Working with meshlets, we make use of efficient mesh shaders throughout the pipeline.*

*Small primitive sets* as SU grouping are favorable for streaming and for supporting extended periods of temporal amortization. Among these TSS techniques, which can be found in the bottom half of the table, are the *shading atlas (SA)* [MVD*18], *tessellated shading streaming* [HSS19a], and *snake binning* [HSS21]. Rather than pre-charting the whole scene with unique coordinates, the preprocessing used in these techniques just breaks down the scene into groups consisting of a few primitives. At runtime, a visibility prepass determines the visible SU set and a size for each SU. From the visible set, a memory management pass determines each SU parametrization to generate a densely packed chart. The shading pass uses this SU parametrization to fill the shading cache. The rationale of this type of approach is that spatially coherent primitives represent a natural grouping of pixels, making groups of primitives more memory efficient than groups of pixels. The disadvantage is that grouping triangles of different sizes or shapes may cause various sampling problems.

The shading atlas (SA) [MVD*18] is a TSS technique that specifically targets compactness and temporal coherence. Its groups consist of $1-3$ triangles. For each visible SU, a rectangular area of suitable size in the shading cache is determined, which is entirely filled with shading points of the SU primitives. For final rendering, a half-pixel offset within each block is used to enable bilinear sampling. The simple and efficient SU parametrization comes at the price of a disregard for perspective effects. If primitives within an SU require different sample distributions or sizes, SA tends to produce over- or undersampling artifacts. The major benefit of SA is that the SU parametrization must only be updated after a significant change in image-space size, an optimal behavior for temporal coherence and video encoding.

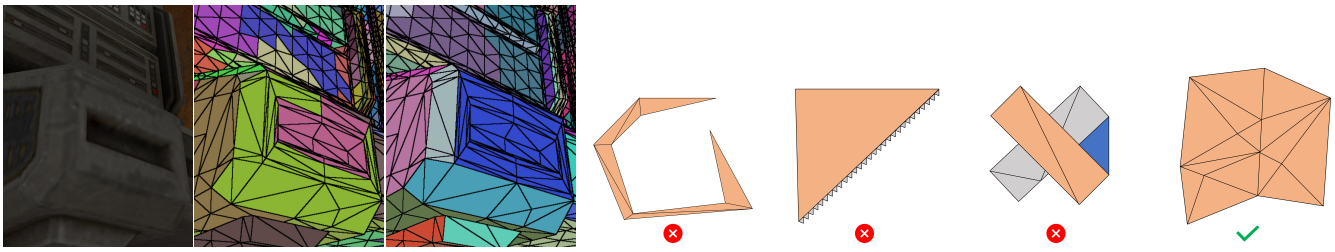
Tessellated shading streaming [HSS19a] uses a single primitive as its SU grouping. Inspired by non-standard texture-space representations such as Ptex [BL08] and Mesh Color Textures [Yuk17], it exploits the hardware tessellation stages to improve sampling and packing efficiency by uniformly sampling near-equilateral triangles. Samples are adjusted based on a perspective correction to prevent sampling artifacts for non-uniform triangles. For slanted triangles, the authors propose an oversampling approach, splitting these triangles into two right-angled triangles and conservatively oversampling them by 25% to 35%. Compared to SA, tessellated shading streaming incorporates perspective information more strongly into its SU parametrization. Most of its overshading comes from increasing the border of individual triangles to support bilinear sampling and from the handling of slanted triangles.

Snake binning (SB) [HSS21] takes the ideas of tessellated shading streaming further towards a fully perspective SU parametrization. To obtain a tightly packed shading cache, triangles are binned in to two dimensions: the image-space height and the angle adjoining the longest edge. This binning enables dense packing in texture space after a simple rotation of the triangles. A hysteresis is applied to the binning to ensure a sufficient amount of temporal coherence across frames. The main downside of SB is that temporal coherence breaks down if its bins overflow, making it necessary to reserve a substantial amount of empty space. Its sampling errors mostly come from the discrete binning, which introduces minor distortions if the geometry does not fully match the bin size. Furthermore, large bins are very sparsely filled. Finally, bilinear sampling of isolated triangles requires conservative rasterization with significant overshading.

Figure 2 reveals that the bottom area, i.e., TSS with SU groupings of more than a handful of primitives, has not been explored so far. In the next section, we will present a method that relies on groups of up to 84 triangles, and can leverage special hardware support in the form of mesh shaders for additional acceleration.

## 3. Meshlet shading atlas

Approaches using small shading units either trade compactness and temporal coherence for sampling distortion, such as the shading atlas (SA), or they favor low amounts of distortion at the cost of overshading many individual triangles, such as snake binning (SB). A remedy is to use larger groups, which reduces overshading while keeping distortion low. To this end, we propose a novel texture-space shading approach, the meshlet shading atlas (MSA). It is optimized to take advantage of modern GPU hardware by grouping many primitives into a meshlet. Compared to individual primitives, meshlets have a much better ratio of border to interior area, leading to a substantial reduction of overshading owed to the need for redundant



**(a)** *Forward shading* **(b)** *Naively clustered* **(c)** *Extended Garland* **(d)** *Concave* **(e)** *Non-uniform size* **(f)** *Overlapping* **(g)** *Uniform size*

**Figure 4:** *We use hierarchical clustering of primitives into meshlets. (b) A naive clustering may greedily group meshlets and produce suboptimal layouts after mapping them into texture space. (c) Our extended Garland [GWH01] approach favors uniform meshlets that do not vary significantly in their surface normals. (d, e, f) A naive approach cannot account for special cases that lead to meshlets that can exhibit sampling issues or wasted memory under various parametrizations. (g) Ideally, uniformly clustered meshlets (as targeted by our clustering approach) utilize the space inside the atlas efficiently and are well behaved for arbitrary parametrizations.*

pixels at the border of an SU. At the same time, MSA preserves the benefits of temporal coherence, caching and upsampling that we have observed for methods based on small primitive sets. As the final SU parametrization is determined at runtime, we have the opportunity to combine view-dependent projection and pre-charting to keep distortion low. The SU packing of MSA extends the memory management of rectangular power-of-two-sized blocks proposed by SA, but replaces its SU grouping ($1-3$ triangles in a rectangular block) with SU handling at meshlet granularity. This decision lets us speed up all relevant stages of the pipeline by relying on mesh shaders [NVI18] for memory management, shading and display.

The full MSA pipeline is shown in Figure 3. In a *preprocessing* pass, performed once per scene, it gathers spatially coherent *meshlets* of up to 84 triangles as recommended for current GPU hardware. In the *visibility* pass, it renders a visibility buffer [BH13] at meshlet granularity. For framerate upsampling and streaming, it predicts one or multiple future camera poses and combines the visibility results.

The *memory management* operates on rectangular *blocks* with power-of-two sizes that are allocated and deallocated in parallel. These blocks are managed within power-of-two wide columns inside square *superblocks*. Each superblock can contain blocks of a specific *width* and variable *height*. The memory management commences in three steps. First, all previously visible meshlets that are now invisible are deallocated, freeing up their slots within the superblock and potentially deallocating empty superblocks. Second, for each visible meshlet, we determine its texture-space *block size* by modifying the *level selection* pass of the original SA [MVD*18] to account for meshlets. We describe in Section 3.2 how the block sizes and SU parametrization for meshlets are determined. The resulting target block size is then slightly increased to account for bilinear sampling and can be biased to meet a specific shading budget. Finally, we allocate one rectangular block for each visible meshlet based on the previously determined size.

The *shading* pass iterates over the shading cache and invokes the fragment shader for every texel of each visible meshlet. Shader image footprints [NVI18] could be used at this point to further reduce the overshading, but only if temporal accumulation, frame-to-frame caching or novel view generation are not required. The *display* pass renders all visible meshlets to the current view with bilinear sampling from the shading cache, using the SU parametrization that was determined during the memory management stage. This provides a flexible, effective solution to combine efficient temporal caching and upsampling, while avoiding sampling errors and overshading caused by an SU that is too small.

### 3.1. Meshlet generation

A good meshlet generation ensures that the resulting meshlets can be mapped into the atlas with as little distortion as possible, while being agnostic to the actual parametrization. Ideal meshlets are flat to prevent overshading of backfaces; their borders should be as convex as possible to avoid wasting space in the atlas, and they should not overlap themselves after perspective projection.

An ideal geometric pipeline for the MSA would include levels of detail (LOD) that contain optimized meshlets for specific viewing distances.However, for simplicity, we first preprocess the scene geometry globally without LOD (using a maximum edge length of 0.1 units in world space), ensuring that the geometric resolution is fine enough such that grouping triangles does not produce meshlets that exceed the maximum block size in texture space. This establishes a baseline that follows the trend of modern micropolygon real-time rendering content.

Second, we generate our meshlets by applying a modified version of the hierarchical clustering proposed by Garland et al. [GWH01]. This clustering method first constructs a dual graph, where each node corresponds to a face cluster, and edge contractions within this graph correspond to merging two clusters. During clustering, a greedy optimization scheme assigns each potential merge a cost depending on the geometric properties of the merged cluster. We modify this cost function to prioritize more important properties for our meshlets, and our final cost function is defined as follows:

$$C = \alpha \cdot E_{fit} + \beta \cdot E_{dir} + \gamma \cdot E_{shape} + \delta \cdot E_{count}. \quad (1)$$

$E_{fit}$, $E_{dir}$ and $E_{shape}$ are the original error measures introduced by Garland et al. that describe the flatness, orientation and compactness of the resulting cluster, respectively [GWH01]. $E_{count}$ is an additional term that encourages cluster merges to target the maximum meshlet sizes, which will bias otherwise equal merges towards

larger meshlet sizes. From initial experiments, $\alpha = 1$, $\beta = 8$ and $\gamma = 0.5$ produced the most reliable results in our scenes. With these parameters, the optimizer will primarily prefer merging clusters that do not fold back onto themselves, with flatness and meshlet size being secondary criteria. However, the cost function alone does not preclude merging of clusters with strongly different orientations if the clustering runs out of alternatives. Therefore, we impose an additional hard constraint that clusters with large difference in average normal vectors must not be merged, which we tuned experimentally on our scenes. Large groups (more than 50 triangles) are not merged if the cluster normals deviate by more than 60 degrees. For smaller merges, this criterion is less strict, and merges with fewer than 10 triangles are permitted a deviation up to 90 degrees in their cluster normals. An example result of our clustering process and examples for the decision criteria of the clustering algorithm are shown in Figure 4. This clustering procedure only needs to be done once per scene, and the resulting hierarchy can be used to generate meshlets of various target sizes.

### 3.2. Meshlet parametrization

No single SU parametrization can lead to an optimal sample distribution for all possible views. This observation is independent of the SU type, be it groups of shading points or groups of primitives. Neither the mip-mapped $8 \times 8$ tiles of TS nor the combination of $1-3$ triangles into power-of-two-sized rectangles used in SA is distortion-free from arbitrary viewpoints. This raises the question on what SU parametrization works best for the meshlets of MSA. We investigate three different ways of determining the SU parametrization: perspective projection, pre-charting and orthographic projection.

**Perspective projection** A straight-forward choice for determining texture coordinates is to use a standard perspective projection from the current point of view. In terms of image quality, the result is optimal for the given viewpoint, and deteriorates in a manner proportional to the displacement of a new viewpoint from the current one. While such a proportional response is generally desirable, the sampling density is also proportional to the angle between the normal at a shading point and the view vector. Near the silhouette in the current view, undersampling may deteriorate the image quality even for small viewpoint displacements, and disocclusion artifacts may occur if the silhouette crosses the interior of the meshlet. We determine the texture-space block size for each meshlet based on its clipped image-space bounding box to ensure that the sample placement is as close as possible to the image-space sample placement. Unfortunately, a small performance penalty must be expected from the need to perform clipping inside the shader, since we cannot use hardware clipping when our render target is the shading cache and not a framebuffer corresponding to the current view.

**Pre-charting** Instead of defining a perspective parametrization on the fly, we can rely on a pre-charted parametrization per meshlet. Essentially, this combines MSA with a texel shading (TS) parametrization to achieve a more view-independenent result. Among the vast amount of surface parameterization methods [FH05], we chose least squares conformal maps (LSCM) [LPRM02], since it comes closest to our idea of a view-independent, yet largely distortion-free parametrization. We first determine an initial texture-space block size via screen-space edge lengths [MVD*18]. To calculate the scale of the final transformation that is used to allocate the meshlet, we combine this initial block size with the bounding box and area of the pre-computed UV chart. This optimizes the allocated memory individually also for UV charts that do not perfectly utilize their individual UV space.

**Orthographic projection** To mitigate the undersampling near the silhouette, we would prefer a "weaker perspective" projection which is less sensitive to the surface orientation. One can imagine how the image of a meshlet changes if the camera is not close to the meshlet, but rather a zoom lens with a large optical power is used. We simulate this type of parametrization using an orthographic projection, with the camera looking towards the meshlet center. We compute the texture-space block size based on the maximum subtended angle between the meshlet center and all meshlet vertices. This parametrization attempts to strike a compromise between view-independent pre-charting and view-dependent perspective parametrization. However, it obviously introduces some distortion with respect to either of the previous two parametrizations. In ideal cases, where meshlets are completely flat and perspective effects are minor, this could potentially be a useful view-agnostic parametrization.

All three types of parametrizations operate in a common framework: First, each transformed meshlet is rotated according to its oriented bounding box (using the gift wrapping algorithm [Jar73]) to get an axis-aligned bounding box of minimal size in landscape orientation. This rotation can directly be baked into any precomputed mapping parametrization. If necessary, the scale component of the SU parametrization can be biased by the available memory in the texture atlas or to limit the number of fragment shader invocations. Moreover, we use a dilation filter to add a one pixel border necessary for bilinear filtering. Trilinear filtering is not expected to enhance quality if we can choose the size of each meshlet in the shading cache individually, which reduces overshading if meshlets are neither too large nor too small in image space. The final SU parametrization is used to determine a power-of-two texture-space block size in the shading cache. Once this parametrization is determined, it can be kept temporally coherent for multiple frames as required.

## 4. Evaluation

We compare the different versions of MSA—MSA with perspective projection (MSA-P), MSA with pre-charting (MSA-UV) and MSA with orthographic projection (MSA-O)—against a selection of texture-space shading methods (see Section 2) as well as against standard forward rendering with a depth prepass.

**Texel shading** Our TS implementation uses $8 \times 8$ tiles in the pre-charted UV space with a total of five mip levels and trilinear filtering. To support overlapping UV spaces, we use layered rendering to make the UV space unique on a per-triangle basis without modifying the original UV layout. As TS with $8 \times 8$ tiles is prone to overshading, we also evaluate against a variant that only shades individual pixels (TS-1).

**Shading atlas** We configure SA to use a generous atlas size of 128 megapixels, allowing for large superblocks of $2048 \times 2048$ pixels to prevent SA from being limited by its maximum block size. These conditions let us focus specifically on the distortion introduced by its blockwise mapping into the atlas.

**Meshlet shading atlas** For MSA, we use a similar atlas size of 128 megapixels and a superblock size of $2048 \times 2048$ pixels. We evaluate the three parametrizations discussed in Section 3: MSA-P, MSA-UV and MSA-O.

### 4.1. Evaluation setup

**Test scenes** We test on three scenes containing physically-based materials [Sch94] and animated lighting and models to evaluate both SA and MSA. *Robot Lab* uses the stock materials, lights and animations provided in the Unity sample scene, including a light shining through a rotating fan and multiple moving robots. *Sponza* (Crytek) was extended by animated, colored spotlights as well as animated falling boulders that provide a challenging scenario in terms of shading and visibility. *Space* is a custom scene built from assets that are freely available on the Unity Asset Store, and showcases a platform in space, with flying asteroids and space ships, as well as challenging animated lighting and high-frequency metallic textures. To properly evaluate TS, we use slightly modified versions of two scenes, *Robot Lab*-texel and *Space*-texel. Both scenes were altered so that their TS footprint fits into 24 GB of video memory. We also evaluate all other approaches on these modified scenes. In all scenes, large triangles are subdivided to provide optimal conditions for both SA and MSA, see Figure 1 for a visualization of the geometric resolution when grouped into meshlets. For each scene, we evaluate four different camera setups, including two dynamic camera paths and two static camera positions for a total of 900 frames at 60 Hz.

**Comparison setup** We compare all shading methods against a forward renderer with $4\times$ supersampling in terms of quality, shader invocations and overshading (OS). We compute overshading individually for each texture-space method by counting the amount of accessed texels during display, versus the amount of shaded texels—this enables us to judge the sample efficiency given frame-by-frame shading. We roughly bias all methods towards a shading budget of $1920 \times 1080$ samples, either by decreasing the bounding box or block size of the SU (SA, MSA) or by adding a positive mip level bias (TS). This bias is computed by measuring the average shader invocations without bias in a calibration run through the scenes. We measure the quality of each approach by computing FLIP [ANA*20] and LPIPS [ZIE*18] metrics against the supersampled forward renderer.

We also evaluate the temporal amortization capabilities of each renderer. We run each scene with a constant temporal upsampling rate, where only every $n$-th frame contains updated shading and animations, and intermediate frames are generated by sampling from the texture-space representation. In this *temporal amortization* experiment, we simulate a server-client streaming setup with perfect visibility prediction for the next $n$ frames. We evaluate temporal upsampling rates of $n \in \{4, 8, 16\}$ on all dynamic camera paths, and bias all renderers towards an increased shading budget of $2 \times 1920 \times 1080$ samples. Thus, we can determine the temporal amortization capabilities of each approach, while still allowing for a higher shading budget to make up for the temporal reuse and resampling of shading information.

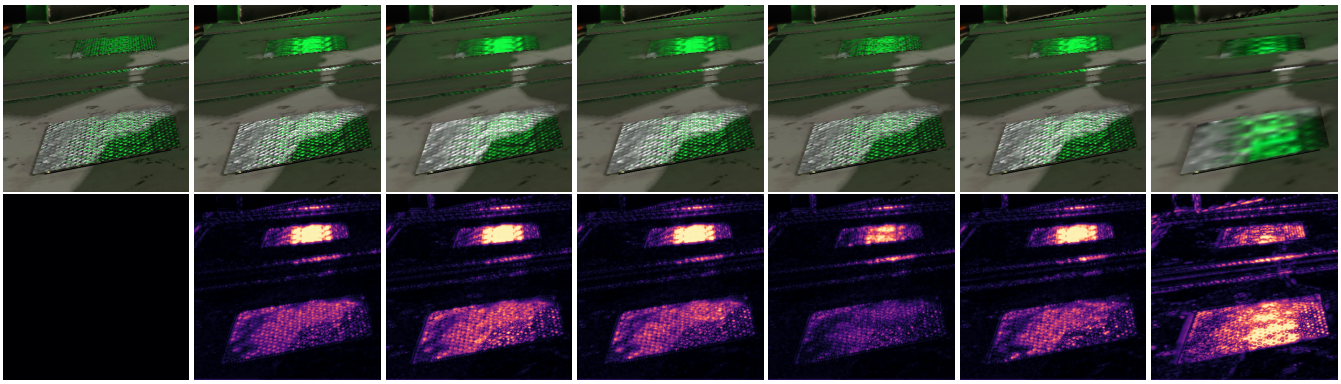
**System setup** All tests were run on a workstation with an NVIDIA RTX 3090 GPU with 24 GB of video memory and an Intel Core i7-8700K CPU with 32 GB of system memory.

### 4.2. Optimal primitive count in meshlets

To determine the optimal number of triangles per meshlet, we first perform a small ablation experiment on some selected scenes. For each scene described in Section 4, we choose an animated camera path and evaluate the maximum number of triangles per meshlet in the range of $[1, 4, 16, 40, 84]$. We bias the shader invocations of each method to $1920 \times 1080$ pixels, and measure FLIP [ANA*20] and LPIPS [ZIE*18] for MSA using perspective (MSA-P), precharted (MSA-UV) and orthographic (MSA-O) parametrizations. Furthermore, to show the effect of meshlet sizes on run-time performance, we also compare timings of all stages of MSA-UV against the original SA.

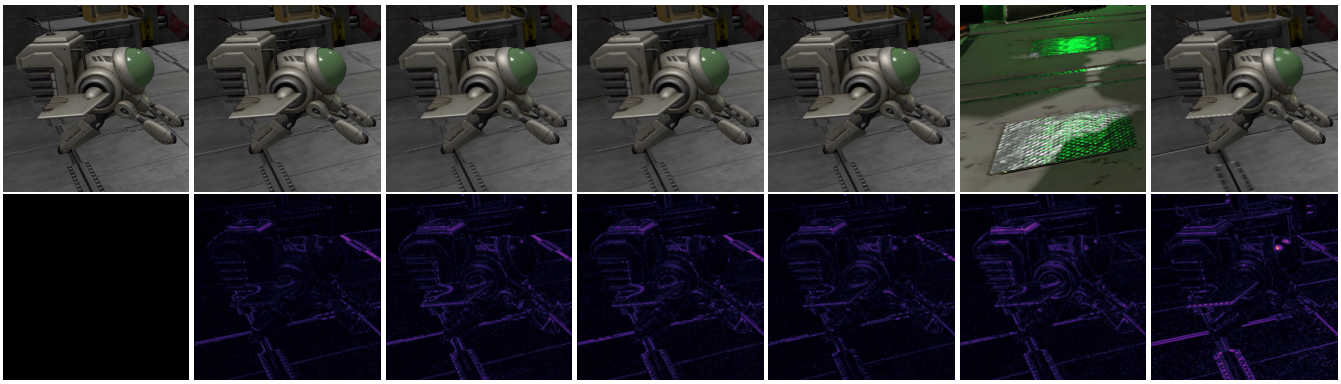
The quality results of the meshlet-size ablation can be seen in Table 1. Both FLIP and LPIPS clearly show that larger meshlet sizes lead to improved quality at a fixed shading budget. This matches the expectation that smaller groups of meshlets require more samples at the borders to enable bilinear sampling, which results in overall worse quality when the shading budget is limited. MSA-UV benefits the most from larger meshlet sizes, while MSA-O benefits the least. For MSA-O, this can be explained by triangles that overlap unfavorably after projection, and thus smaller meshlets achieve similar quality even though the overshading due to additional boundary pixels is higher for small meshlets. As it showed the best overall image quality, we only consider the maximum meshlet size of 84 throughout the rest of our evaluation.

In terms of run-time performance, Table 2 clearly shows that small meshlet sizes are suboptimal, especially for texture-space memory management. Compared to SA, MSA-UV at a maximum meshlet size of 84 shows a $4.72\times$ speedup when allocating and deallocating blocks in texture space, as well as a $2.18\times$ speedup when determining the optimal block sizes. This speedup is optimal across our tested meshlet sizes—both larger and smaller meshlet sizes reduce the achieved speedup. Very small meshlet sizes (such as 1, 4 or 16) show up to $3-7\times$ worse performance across all stages, with the shading stage being impacted the most. Small groups of primitives are more suited towards traditional geometry pipelines and need specific tuning to achieve optimal performance. Furthermore, our implementation also does not benefit from even larger meshlet sizes. During shading, MSA-UV shows slight overheads compared to the original SA, which are mostly due to the usage of conservative rasterization. Finally, in the display stage, MSA-UV outperforms SA with as few as 16 triangles per meshlet. Overall, our prototype implementation of MSA-UV outperforms SA at meshlet sizes larger than 84 both in terms of quality and run-time performance.



**(a)** *Reference (4× SSAA)* **(b)** *Forward* **(c)** *MSA-O* **(d)** *MSA-P* **(e)** *MSA-UV* **(f)** *SA* **(g)** *TS*

**Figure 5:** *Qualitative comparison of all shading methods in the* Space*-texel scene. Among the texture-space methods, MSA-UV achieves the closest result compared to the reference. All other methods blur the fine high-frequency texture and shading details in this scene.*



**(a)** *Reference (4× SSAA)* **(b)** *Forward* **(c)** *MSA-O* **(d)** *MSA-P* **(e)** *MSA-UV* **(f)** *SA* **(g)** *TS*

**Figure 6:** *Qualitative comparison of all shading methods in the* Robot Lab*-texel scene. All methods perform very well, as this scene contains mostly low-frequency texture detail and shading. MSA-UV, MSA-P and SA manage to achieve slightly sharper textures, with MSA-O and TS being slightly blurrier in comparison.*

**Table 1:** *FLIP and LPIPS results averaged across three representative paths in* Robot Lab*,* Space *and* Sponza *to evaluate differing maximum triangle counts per meshlet. At a limited shading budget, larger meshlet sizes show improved quality, suggesting that the sampling error and overshading is reduced for larger meshlets.*

| | MSA-O | | MSA-P | | MSA-UV | |
|---|---|---|---|---|---|---|
| Meshlet Size | FLIP | LPIPS | FLIP | LPIPS | FLIP | LPIPS |
| 1 | .061 | .241 | .059 | .235 | .061 | .255 |
| 4 | .054 | **.192** | .053 | .189 | .049 | .176 |
| 16 | .054 | .195 | .052 | .181 | .049 | .176 |
| 40 | .054 | .193 | .051 | .177 | .048 | **.167** |
| 84 | **.053** | .193 | **.049** | **.170** | **.047** | .170 |

**Table 2:** *Timings in milliseconds for all stages of MSA-UV and SA. MSA-UV-N denote the timings for a variety of meshlet sizes N. Overall, MSA-UV outperforms the original SA at meshlet sizes larger than* 84 *in terms of memory management, while incurring slight overheads during shading due to conservative rasterization.*

| Renderer | Block Size | Block (De-)Alloc. | Shading | Display | Total |
|---|---|---|---|---|---|
| MSA-UV-1 | .151 | .355 | 7.488 | .275 | 8.269 |
| MSA-UV-4 | .075 | .156 | 6.408 | .141 | 6.781 |
| MSA-UV-16 | .044 | .081 | 5.852 | .071 | 6.047 |
| MSA-UV-40 | .030 | .055 | 5.852 | **.058** | 5.997 |
| MSA-UV-84 | **.022** | **.036** | 5.713 | **.058** | **5.829** |
| MSA-UV-126 | .023 | .039 | 5.740 | **.058** | 5.859 |
| SA | .048 | .170 | **5.623** | .092 | 5.932 |
| MSA-UV-84 vs. SA (Speedup) | 2.180 | 4.716 | .984 | 1.583 | 1.018 |

**Table 3:** *We evaluate all methods by comparing them in terms of quality (FLIP and LPIPS) and overshading (OS) at a limited shading budget of* $1920 \times 1080$ *pixels. Across all our scenes, the MSA methods are the best performing texture-space methods, performing almost as well as the forward renderer. Although MSA-UV exhibits slightly more overshading compared to MSA-P and SA, it still achieves the best quality, suggesting that its quality per shaded pixel is higher due to a more suitable mapping into texture-space. Both TS and TS-1 fall behind the competing texture-space methods, as these approaches either show significant overshading (TS) or require explicit mip mapping (TS-1), which results in lower image quality at a fixed shading budget.*

| | *Robot Lab* | | | *Space* | | | *Sponza* | | | *Robot Lab*-texel | | | *Space*-texel | | | Average | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Renderer | FLIP | LPIPS | OS | FLIP | LPIPS | OS | FLIP | LPIPS | OS | FLIP | LPIPS | OS | FLIP | LPIPS | OS | FLIP | LPIPS | OS |
| Forward | .028 | .107 | - | .062 | .078 | - | .038 | .078 | - | .023 | .054 | - | .066 | .085 | - | .043 | .074 | - |
| MSA-O | .035 | .185 | 1.150 | .075 | .172 | 1.357 | .050 | .203 | 1.232 | .030 | .116 | 1.103 | .073 | .127 | 1.183 | .052 | .143 | 1.205 |
| MSA-P | .035 | .178 | 1.078 | .070 | **.140** | **1.032** | **.047** | **.178** | **1.041** | .029 | .105 | 1.047 | .071 | .118 | 1.072 | .050 | .128 | 1.054 |
| MSA-UV | **.032** | **.157** | 1.175 | **.068** | .155 | 1.348 | .048 | .191 | 1.250 | **.026** | **.086** | 1.115 | **.063** | **.101** | 1.170 | **.047** | **.123** | 1.212 |
| SA | .037 | .193 | **1.022** | .074 | .166 | 1.068 | .048 | .188 | 1.054 | .029 | .107 | **1.015** | .071 | .127 | **1.043** | .052 | .139 | **1.041** |
| TS | - | - | - | - | - | - | - | - | - | .051 | .249 | 3.031 | .117 | .192 | 3.884 | .084 | .175 | 3.457 |
| TS-1 | - | - | - | - | - | - | - | - | - | .032 | .119 | 1.000 | .087 | .125 | 1.000 | .059 | .101 | 1.000 |

### 4.3. Image quality

The results in Table 3 show that MSA comes closest towards matching the quality of forward rendering at a fixed shading budget. Among the tested texture-space methods, MSA-UV achieves the highest quality, followed closely by MSA-P. MSA-O suffers from the fact that an orthographic projection differs too much from the required sample distribution during display, even for well-behaved meshlets. SA performs surprisingly well, given that it tends to produce blurring artifacts if triangles of different sizes are matched within blocks [MVD*18; HSS21]. We attribute this finding to the near-optimal geometric resolution in our scenes—triangles are sufficiently subdivided such that they can be mapped into rectangular blocks very efficiently. Finally, TS achieves the worst quality throughout all scenes. In contrast to its competitors, which directly choose a size per SU, TS needs to explicitly shade all the required texels for trilinear filtering in two mip levels, which significantly increases the total shading load, resulting in much lower quality when biased towards a fixed shading budget. The same result can also be seen when shading individual texels instead of tiles (TS-1), where the requirement for explicit shading of mip levels still results in overall worse quality at a limited shading budget.

Figures 5 and 6 showcase cropped images and FLIP results from the *Space*-texel and *Robot Lab*-texel scenes. *Space*-texel (Figure 5) shows noticeable shading aliasing, which can vary significantly depending on the sample distribution in texture space. All tested methods (including the standard resolution forward renderer) blur the final result too much, with TS being completely blurry due to the limited shading budget. MSA-UV better replicates the high-frequency texture detail compared to all the other methods, as it allocates slightly more space based on its pre-charted layout. In *Robot Lab*-texel (Figure 6), all methods fare very well due to the low-frequency nature of the scene. Most textures are low-frequency, and there is no high-frequency lighting or animation present in this scene. Still, we observe that MSA-UV shows the most faithful representation compared to the reference, with SA and MSA-P following closely. TS also handles this scene well, despite the significant bias to fit the shading budget.

### 4.4. Overshading

For each TSS method, we compute the overshading (OS) as the ratio between shaded pixels and pixels that are used to generate the final image. Ignoring TS-1 (which only shades the exact number of pixels necessary for final display), the overshading results in Table 3 show that SA exhibits the least amount of overshading. We attribute this to the well-behaved geometric resolution of our scenes—if triangles can be packed efficiently into blocks at almost correct resolution, SA operates at optimal conditions. Furthermore, since SA only uses $1-3$ triangles per SU, it only shades a small portion of occluded or out-of-view samples. The MSA variants occupy the second place, with MSA-P exhibiting less overshading than MSA-UV, followed by MSA-O. Its use of a perspective parametrization suggests that MSA-P is efficient in terms of overshading, as it clips away a majority of invisible texels that would lie outside the view frustum, and maps the meshlets with the same parametrization that is used in the current view. In contrast, the pre-charted layouts of MSA-UV can generate more overshading for partially invisible geometry, as full meshlets are shaded as a unit, even if only a single triangle is visible. The mesh shaders could be modified to discard fully invisible triangles within meshlets, further increasing quality for single frame rendering as it lowers the bias. However, this would diminish the support for temporal accumulation and novel view generation, where shading primitives that are slightly out of view must be rendered. MSA-O suffers from the same problem, as it shades all triangles of each partially visible meshlet. Since it is not as optimized with respect to the texture layout as MSA-UV, it produces worse image quality at similar amounts of overshading. Overall, even though MSA-UV exhibits more overshading compared to MSA-P, it is more robust to resampling its texture-space layout to fit into lower shading budgets, resulting in MSA-UV achieving better quality compared to MSA-P on average. Finally, TS shows the most amount of overshading, with most of it attributed to the $8 \times 8$ tiles that need to be shaded for each texel that is required by the final display pass. With trilinear filtering, this can result in a potential 8 tiles that need to be shaded for a single output pixel if viewing angles are very oblique or only a small part of the texture is visible on screen. At worst, this can require $512 (= 8 \cdot 8 \cdot 8)$ shaded texels for a single output pixel, though this extreme case is unlikely in practice.

### 4.5. Temporal amortization

When keeping shading information over multiple frames and sampling from different novel views, view agnostic parametrizations have a clear advantage over perspective parametrizations that are tailored towards a specific view. This can be seen in Table 4—at all upsampling rates, MSA-UV achieves the best quality. Compared to the results without temporal upsampling, MSA-O even outperforms MSA-P on all upsampling rates, suggesting that the more view-independent orthographic projection is slightly more suitable compared to a reference perspective projection if shading results are reused over multiple frames. The SA performs almost as well as MSA-O in FLIP, but fares slightly worse in LPIPS—suggesting that it is also well suited for temporal upsampling, but still suffers from its inherent sampling issues due to the fixed rectangular layout, reducing image quality. Finally, even with an increased shading budget, TS shows the worst quality in the temporal upsampling use case. Even though its texture-space mapping is completely view-agnostic, its inherent overshading still results in quality degradation at a fixed shading budget.

**Table 4:** *Across various temporal upsampling factors n, the MSA variants achieve the best image quality. MSA-UV benefits the most from its view-agnostic mapping, which lends itself well to temporal upsampling. Furthermore, orthographic projection (MSA-O) enables higher quality upsampling compared to perspective projection (MSA-P), as it is not as affected by sample distortion over time. Even with an increased shading budget, the inherent overshading of TS reduces its effectiveness for temporal upsampling.*

| | $n = 4$ | | $n = 8$ | | $n = 16$ | |
|---|---|---|---|---|---|---|
| Renderer | FLIP | LPIPS | FLIP | LPIPS | FLIP | LPIPS |
| MSA-O | .045 | .098 | .049 | .102 | .056 | .113 |
| MSA-P | .047 | .105 | .051 | .111 | .061 | .126 |
| MSA-UV | **.041** | **.080** | **.045** | **.084** | **.052** | **.094** |
| SA | .048 | .117 | .051 | .122 | .058 | .130 |
| TS | .061 | .099 | .065 | .105 | .072 | .112 |

## 5. Limitations, conclusion and future work

Based on existing work, we showed that texture-space methods can be primarily identified by the *grouping* and *parametrization* of *shading units*. We derived a taxonomy of texture-space shading, which is a useful tool to analyze the benefits and drawbacks of texture-space shading methods. By observing a gap in this taxonomy, we motivated a texture-space shading approach that is able to use arbitrary *parametrizations* on large groups of primitives: the meshlet shading atlas (MSA). We showed that it is beneficial for the final image quality to group as many primitives as possible into meshlets, and when compared to existing texture-space shading methods, the MSA achieves better quality at the same shading budget. Furthermore, when reusing shading information over multiple frames, we demonstrated that using pre-charted LSCM maps enables a high-quality view-agnostic mapping into texture space. Even without temporal reuse, pre-charted meshlets result in high quality, likely because charting meshlets separately results in less distortion compared to charting a whole mesh at once. Finally, we show that even a simple perspective parametrization can be sufficient for high-quality texture-space shading, without requiring precomputation of charts. Overall, the MSA provides an elegant solution for various texture-space shading problems, while being simple to integrate into modern rendering systems. The main limitation of MSA is that its effectiveness can depend on the geometric resolution and clustering of the scene, especially for the orthographic and perspective variants. These variants can show disocclusion artifacts within projected meshlets if the camera view changes. Furthermore, large-scale meshlets might exceed the maximum block size in texture-space, and can also result in more overshading if only a few of their primitives are visible in the current view, as visibility is always handled on a per-meshlet granularity. Although the amount of overshading for MSA is comparable to existing work, if novel view generation is not desired, invisible triangles can be culled in the mesh shader to further reduce the amount of overshading. Additionally, these considerations could be addressed by optimized levels of detail, where the meshlet clustering pipeline produces meshlets that are sized appropriately for any given viewing distance.

A second limitation of our initial prototype is the packing density for streaming scenarios. MSA-UV achieves a packing density of 55% when scaling the meshlets to the allocated block size, mostly due to the empty space of non-rectangular meshlets. However, for streaming, due to the inherent temporal coherence of the MSA, it would be possible to cut and pack the shading cache into MPEG-block-sized pieces to achieve a more optimal packing density. The MPEG-encoder could then efficiently encode the remaining empty space, resulting in bitrates and encoding speeds that are similar to image sequences only containing shaded pixels. Furthermore, we could provide an empty-space mask to a custom encoder, such that all non-shaded pixels could be completely ignored.

Thirdly, as the MSA directly inherits its block-based memory management from the SA, it inherits some of its shortcomings. In particular, applying a large bias towards the block sizes in texture space can lead to visible popping artifacts if block sizes change between neighboring powers of two. This is not noticable when meshlets are shaded at their optimal size, and could also be remedied by lifting the power-of-two limitation on block allocation or modifying the meshlet parametrization to smoothly scale when transitioning between block sizes under heavy bias.

Finally, MSA allows fine-grained control over the desired amount of shader invocations. Individual triangles can be culled when not needed; we can even use exact shader footprint determination to shade only the required pixels for the final display pass. These measures can be mixed and matched for individual pixels, individual triangles or full meshlets.

To our knowledge, MSA is the first texture-space shading system that transforms and shades large groups of primitives in texture space. We believe that this approach will be an essential part of modern rendering systems, and that the MSA provides an attractive solution to the memory management and parametrization parts of this pipeline.

**Acknowledgment** This work was supported by the Christian Doppler Laboratory for Semantic 3D Computer Vision, funded in part by Qualcomm Inc.

## References


[Ake93] AKELEY, KURT. "Reality Engine Graphics". *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '93. Anaheim, CA: Association for Computing Machinery, 1993, 109–116. ISBN: 0897916018. DOI: 10.1145/166117.166131. URL: https://doi.org/10.1145/166117.166131 3.

[ANA*20] ANDERSSON, PONTUS, NILSSON, JIM, AKENINE-MÖLLER, TOMAS, et al. "FLIP: A Difference Evaluator for Alternating Images". *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 3.2 (2020), 15:1–15:23 7.

[BH13] BURNS, CHRISTOPHER A. and HUNT, WARREN A. "The Visibility Buffer: A Cache-Friendly Approach to Deferred Shading". *Journal of Computer Graphics Techniques (JCGT)* 2.2 (Aug. 2013), 55–69. ISSN: 2331-7418. URL: http://jcgt.org/published/0002/02/04/ 5.

[BL08] BURLEY, BRENT and LACEWELL, DYLAN. "Ptex: Per-Face Texture Mapping for Production Rendering". *Eurographics Symposium on Rendering 2008*. 2008, 1155–1164 4.

[Epi21a] EPIC GAMES. *Lumen Technical Details*. Visited on September 01, 2021. 2021. URL: https://docs.unrealengine.com/5.0/en-US/RenderingFeatures/Lumen/TechOverview/ 2.

[Epi21b] EPIC GAMES. *Nanite Virtualized Geometry*. Visited on September 01, 2021. 2021. URL: https://docs.unrealengine.com/5.0/en-US/RenderingFeatures/Nanite/ 2.

[FH05] FLOATER, MICHAEL S. and HORMANN, KAI. "Surface Parameterization: a Tutorial and Survey". *Advances in multiresolution for geometric modelling*. Ed. by DODGSON, N. A., FLOATER, M. S., and SABIN, M. A. Springer Verlag, 2005, 157–186. URL: http://vcg.isti.cnr.it/Publications/2005/FH05 6.

[GWH01] GARLAND, MICHAEL, WILLMOTT, ANDREW, and HECKBERT, PAUL S. "Hierarchical Face Clustering on Polygonal Surfaces". ACM Press, 2001. ISBN: 1581132921 5.

[HGF14] HE, YONG, GU, YAN, and FATAHALIAN, KAYVON. "Extending the Graphics Pipeline with Adaptive, Multi-Rate Shading". *ACM Trans. Graph.* 33.4 (July 2014). ISSN: 0730-0301. DOI: 10.1145/2601097.2601105. URL: https://doi.org/10.1145/2601097.2601105 3.

[HSS19a] HLADKY, JOZEF, SEIDEL, HANS-PETER, and STEINBERGER, MARKUS. "Tessellated Shading Streaming". *Computer Graphics Forum (Proc. Eurographics Symposium on Rendering 2019)* (2019). ISSN: 1467-8659. DOI: 10.1111/cgf.13780 4.

[HSS19b] HLADKY, JOZEF, SEIDEL, HANS-PETER, and STEINBERGER, MARKUS. "The Camera Offset Space: Real-time Potentially Visible Set Computations for Streaming Rendering". *ACM Transactions on Graphics (Proc. SIGGRAPH Asia 2019)* 38.6 (2019). DOI: 10.1145/3355089.3356530 3.

[HSS21] HLADKY, J., SEIDEL, H.P., and STEINBERGER, M. "SnakeBinning: Efficient Temporally Coherent Triangle Packing for Shading Streaming". *Computer Graphics Forum* 40.2 (2021), 475–488. DOI: https://doi.org/10.1111/cgf.142648. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.142648. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.142648 4, 9.

[HY16] HILLESLAND, KARL E and YANG, JC. "Texel Shading". *Proceedings of the 37th Annual Conference of the European Association for Computer Graphics: Short Papers*. EG '16. Goslar Germany, Germany: Eurographics Association, 2016, 73–76. DOI: 10.2312/egsh.20161018 3.

[Jar73] JARVIS, R.A. "On the identification of the convex hull of a finite set of points in the plane". *Information Processing Letters* 2.1 (1973), 18–21. ISSN: 0020-0190. DOI: https://doi.org/10.1016/0020-0190(73)90020-3. URL: https://www.sciencedirect.com/science/article/pii/0020019073900203 6.

[LD12] LIKTOR, GÁBOR and DACHSBACHER, CARSTEN. "Decoupled Deferred Shading for Hardware Rasterization". *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D '12. Costa Mesa, California: Association for Computing Machinery, 2012, 143–150. ISBN: 9781450311946. DOI: 10.1145/2159616.2159640. URL: https://doi.org/10.1145/2159616.2159640 2.

[Liu20] LIU, EDWARD. *DLSS 2.0 - Image Reconstruction for Real-time Rendering with Deep Learning*. 2020. URL: https://developer.nvidia.com/gtc/2020/video/s22698-vid 1.

[LPRM02] LÉVY, BRUNO, PETITJEAN, SYLVAIN, RAY, NICOLAS, and MAILLOT, JÉROME. "Least Squares Conformal Maps for Automatic Texture Atlas Generation". *ACM SIGGRAPH conference proceedings*. Ed. by ACM. July 2002. URL: http://www.loria.fr/publications/2002/A02-R-065/A02-R-065.ps 6.

[MNV*21] MUELLER, JOERG H., NEFF, THOMAS, VOGLREITER, PHILIP, et al. "Temporally Adaptive Shading Reuse for Real-Time Rendering and Virtual Reality". *ACM Trans. Graph.* 40.2 (Apr. 2021). ISSN: 0730-0301. DOI: 10.1145/3446790. URL: https://doi.org/10.1145/3446790 2.

[MVD*18] MUELLER, JOERG H, VOGLREITER, PHILIP, DOKTER, MARK, et al. "Shading Atlas Streaming". *ACM Transactions on Graphics* 37.6 (Nov. 2018). DOI: 10.1145/3272127.3275087 2–6, 9.

[NSL*07] NEHAB, DIEGO, SANDER, PEDRO V., LAWRENCE, JASON, et al. "Accelerating Real-Time Shading with Reverse Reprojection Caching". *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*. GH '07. San Diego, California: Eurographics Association, 2007, 25–35. ISBN: 9781595936257 3.

[NVI18] NVIDIA. *NVIDIA Turing GPU Architecture Whitepaper*. Visited on September 01, 2021. 2018. URL: https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf 1, 2, 5.

[PZ17] PENNER, ERIC and ZHANG, LI. "Soft 3D reconstruction for view synthesis". Vol. 36. Association for Computing Machinery, Nov. 2017. DOI: 10.1145/3130800.3130855 2.

[RLC*11] RAGAN-KELLEY, JONATHAN, LEHTINEN, JAAKKO, CHEN, JIAWEN, et al. "Decoupled Sampling for Graphics Pipelines". *ACM Trans. Graph.* 30.3 (May 2011). ISSN: 0730-0301. DOI: 10.1145/1966394.1966396. URL: https://doi.org/10.1145/1966394.1966396 2.

[Sch94] SCHLICK, CHRISTOPHE. "An Inexpensive BRDF Model for Physically-based Rendering". *Computer Graphics Forum* 13.3 (Aug. 1994), 233–246. DOI: 10.1111/1467-8659.1330233 7.

[YLS20] YANG, LEI, LIU, SHIQIU, and SALVI, MARCO. "A Survey of Temporal Antialiasing Techniques". *Computer Graphics Forum* 39.2 (2020), 607–621. DOI: https://doi.org/10.1111/cgf.14018. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14018. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14018 1, 3.

[Yuk17] YUKSEL, CEM. "Mesh Color Textures". *High-Performance Graphics (HPG 2017)*. Los Angeles, CA: ACM, 2017. ISBN: 978-1-4503-5101-0/17/07. DOI: 10.1145/3105762.3105780. URL: http://doi.acm.org/10.1145/3105762.3105780 4.

[ZIE*18] ZHANG, RICHARD, ISOLA, PHILLIP, EFROS, ALEXEI A, et al. "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric". *CVPR*. 2018 7.