Tile Pair-based Adaptive Multi-Rate Stereo Shading

Yazhen Yuan, Rui Wang and Hujun Bao

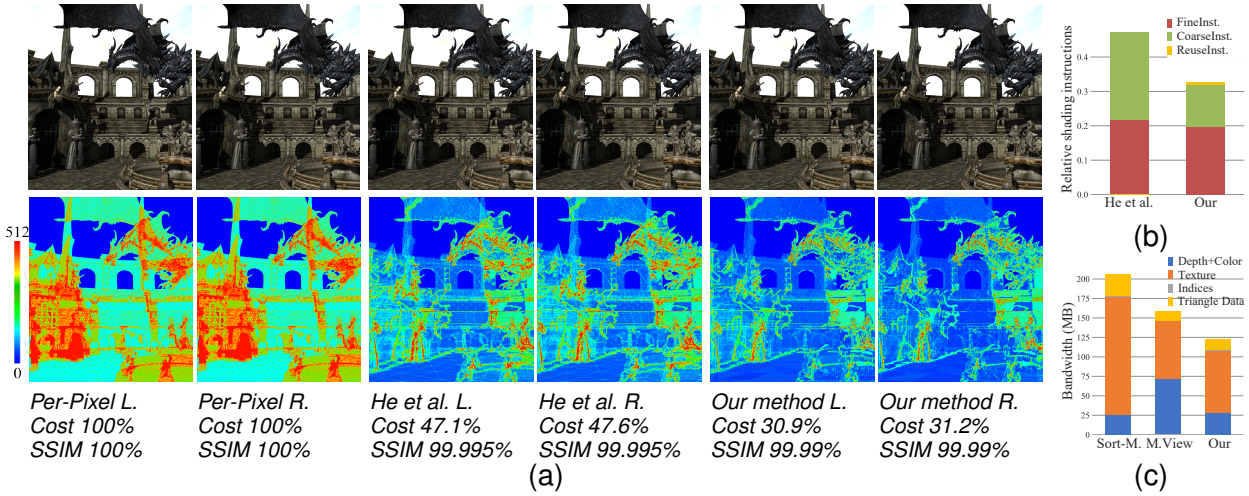


Fig. 1: *Stereo shading results from the traditional per-pixel shading, the adaptive multi-rate shading proposed by He et al. [1], and our tile pair-based adaptive multi-rate shading on the Arena scene: (a) one frame of stereo images (top row) and per-pixel shading instruction counts (bottom row); (b) the reduced shading instructions; and (c) the memory bandwidth used to render this frame. The results demonstrate that our newly proposed method outperforms the traditional sort-middle shading and the state-of-art rendering framework by achieving much lower shading costs and memory bandwidth. When shading this one frame, our method effectively reduces the shading rate to only 30% compared to 47% of multi-rate shading, and still maintains very good visual quality (over 99.99% visual similarity). Our proposed scheduling method consumes a considerably lower memory bandwidth, only 123 MB compared to 205 MB of the traditional sort-middle method (Sort-M.), and less than the previous multi-view rasterization approach [2](M.View).*

Abstract—The work proposes a new stereo shading architecture that enables adaptive shading rates and automatic shading reuse among triangles and between two views. The proposed pipeline presents several novel features. First, the present sort-middle/bin shading is extended to tile pair-based shading to rasterize and shade pixels at two views simultaneously. A new rasterization algorithm utilizing epipolar geometry is then proposed to schedule tile pairs and perform rasterization at stereo views efficiently. Second, this work presents an adaptive multi-rate shading framework to compute shading on pixels at different rates. A novel tile-based screen space cache and a new cache reuse shader are proposed to perform such multi-rate shading across triangles and views. The results show that the newly proposed method outperforms the standard sort-middle shading and the state-of-the-art multi-rate shading by achieving considerably lower shading costs and memory bandwidths.

Index Terms—stereo rendering, tile-pair based rendering, multi-rate shading

1 Introduction

Recent developments in virtual reality (VR) have led to a boom of real-time VR applications, but the immersion VR experience demands high visual realism and low latency at synthesizing stereo images. Low latency arises as a fundamental and necessary requirement, because high latency may lead to a detached VR experience, motion sickness, or dizziness. According to a previous study, "20 milliseconds or less will provide the minimum level of latency deemed acceptable" [3]. However, even after several decades of development, the synthesis of realistic stereo images at 50 or higher FPS is still one of the largest challenges in computer graphics. Stereo rendering deserves enhanced software and hardware solutions.

Compared with conventional rendering, stereo rendering generates two views at once instead of one, resulting in higher computational overload. However, under the stereo configuration, the geometry and shading of the two views are usually the same or largely similar. Such correlations between the two views have already inspired different rendering algorithms to accelerate the rendering process [2], [4], [5], [6], [7], [8], [9]. However, these approaches mainly

• *Authors were in State key laboratory of CAD&CG, Zhejiang University, China. E-mail: rwang@cad.zju.edu.cn*

focused on adapting two views generations in one rendering pass or only accelerating the rasterization step.

Moreover, in current graphics processors and real-time applications, a large portion of the computational resource is spent on executing per pixel shading. The sharing of pixels, especially the reuse of shading between two views, is becoming increasingly important and sometimes necessary to accelerate stereo rendering. Reducing the cost of shading by reusing samples within one pixel or sharing samples across pixels has been studied for a while, and such methods are also known as multi-rate (frequency) shading [1], [10], [11], [12], [13], [14]. Although some of these approaches have the potential to be extended to stereo views, at present, a specific multi-rate shading solution for stereo rendering has not been well explored.

In this paper, a new rendering pipeline is proposed that enables adaptive multi-rate shading at stereo views. The proposed technique presents several novel features. First, the present tile-based shading was extended to tile pair-based shading. Specifically, tiles at two views are first paired by corresponding geometry. Each tile pair is then scheduled to rasterize triangles and shade pixels at two views simultaneously. To better schedule the shading order of tile pairs, inspired by previous work [2], [4], [5], the underlying geometry relations of stereo views were used to guide the shading order of tile pairs along corresponding horizontal epipolar lines at two views. As a result, triangles are efficiently rasterized with horizontal overlapping tiles to avoid being reloaded multiple times in tile-based shading (which is also known as the bin spread problem in tile-based shading [15]). More importantly, shading pixels at two views simultaneously leads to more coherent texture accesses during shading calculation, thereby significantly reduces memory bandwidth used for shading. Figure 1(c) shows such a saving for shading one frame, where the I/O of triangle data and texture is effectively reduced.

The second novel feature of the new pipeline is an adaptive multi-rate shading framework for stereo rendering. Once one tile pair was rasterized and shaded, following previous work [1], [11], the execution of the post-rasterization screen space shading was partitioned into fine and coarse fragment operations. A screen space based cache and a cache reuse shader were then newly designed to store coarse fragment inputs and outputs to allow the reuse of shading across views and triangles. The results show that the proposed rendering pipeline can efficiently reuse and reduce shading at stereo views. Fig. 1(a) shows an example of the substantial savings possible.

The main contributions of the proposed architecture are as follows:

- We propose an adaptive multi-rate stereo shading architecture that enables flexible control of shading rates and automatic shading reuse between two stereo views.
- We present a tile pair-based shading scheme for stereo rendering that rasterizes triangles and shades pixels at two views simultaneously;
- We design a scheduling algorithm to better explore geometry correspondences and consistencies at shading tile-pairs via epipolar geometry.
- We describe a new multi-rate shading framework to cache shading at view tiles and reuse them across triangles and views.

2 Related Work

Stereo Rendering. Stereo rendering is a special case of multi-view rendering, and a naive multi-view rendering can be easily implemented into multiple separate rendering passes using modern graphics APIs, such as OpenGL, DirectX or recent variations (Vulkan or Metal). Therefore, improving the shading efficiency beyond multi-pass rendering becomes a key interest of multi-view rendering research.

A direct extension of stereo rendering research is to coordinate with advances in hardware pipeline and perform two views of transformation and rasterization simultaneously. De Sorbier et al. [16] proposed a simple multi-view rendering method using vertex and geometry shaders to transform geometry for two views in one rendering pass. Recently, NVIDIA released VRWorks^{TM} [17], which was formerly known as GameWorks VR. This SDK contains a suite of APIs, libraries, and features, to tackle high-performance VR rendering. Several graphics features of VRWorks aim at improving the performance of stereo rendering. For example, *Single Pass Stereo* uses a new multi-projection architecture of NVIDIA Pascal-based GPUs to draw geometry only once and then simultaneously project both right-eye and left-eye views of the geometry. However, such a technique only outputs positions with two coordinates in the vertex shader for two views, and it does not optimally consider geometry relations in generating two views. *Multi-Res Shading* and the latest PASCAL-based *Lens Matched Shading* render the image at a resolution that better matches the pixel density of the lens (i.e., correcting the image by scaling down resolutions at image edges). However, this multi-resolution shading is only designed for the distortion of the lens, and it cannot handle the multi-frequency of the shading signals of different scenes, which is one of two main targets of our solution.

To better rasterize geometry between two views, several previous approaches utilized epipolar geometry in different aspects. Adelson et al. [4] derived, analyzed and optimized several classic graphics algorithms, such as scan-line rasterization, Gouraud shading, clipping, and hidden surface elimination, for simultaneous generation of stereo views using projection correspondences. Halle [5] also presented a method for multiple viewpoint rendering by rendering polygons as a multitude of lines in epipolar plane images. Inspired by this idea, Hasselgren and Akenine-Möller [2] proposed a multi-view rasterization architecture that rasterizes each triangle to multiple views simultaneously along a scanline and reuses fragment colors in neighboring views. The proposed method in this study is built upon these approaches and further utilizes epipolar geometry to schedule the tile pair-based rendering and explore multi-rate shading on stereo views.

Another kind of solutions exploit the image space similarity of two views. In general, these solutions only generate one view, and then warp or re-project the view to another view to save the rendering time. However, the direct warping or re-projection of one view usually results in image holes. To fill these holes, different approaches have been

proposed, such as interpolation [6], ray casting [7], blurring of the depth buffer [8], [18], and jointly considering two views [9]. Compared with these image space-based solutions, the proposed solution also reuses shading on image space, but this solution uses complete rasterization of two views and only shares shading between them.

Multi-rate Real-time Rendering. In current graphics hardware and real-time applications, a large portion of the computational resource is spent on executing per-pixel shading on GPUs. Therefore, limiting the cost of shading is a key challenge for both graphics application developers and GPU architects. Many approaches have been proposed with one fundamental idea of reusing shading across pixels.

Multisampling antialiasing (MSAA) [19] is used as a modern GPU feature to reduce shading invocations by reusing one shading result for multiple samples of each rasterized triangle in a pixel. However, in some geometrically complex situations where many triangles intersect a pixel, the shading cost rises dramatically because MSAA must shade each contributing triangle at least once per-pixel. To address this problem, Fatahalian et al. [20] presented a quad-fragment merging technique to enable the sharing of shading computations between adjacent mesh triangles. Ragan-Kelley et al. [12] proposed a decoupled sampling pipeline to separate shading and visibility, in which shading is lazily evaluated over a per-triangle shading grid in image space and cached for reuse. Such an idea of decoupled sampling has also been applied into deferred rendering, and shading is computed after visibility is determined [14], [21]. Crassin et al. [22] proposed an aggregate G-Buffer representation to approximate geometry inside each pixel and decouple the shading rate from the geometric sampling rate.

Shading reuse does not necessarily only depend on geometry complexity, because it also can be performed with multi-frequency of shading signals. A simple implementation [23], [24] is to directly generate a low-resolution buffer storing low-frequency terms of the shading function, and a second rendering pass utilizing these low-frequency terms is then performed for final per-pixel shading. Hasselgren et al. [25] has also proposed a multi-rate shading method by executing culling computations over tiles prior to the pixel shader, and Wang et al. [26] have presented an automatic optimization scheme to transfer per-pixel computations to per-vertex or per-primitive, thereby reducing the shading cost. He et al. [27] then designed a system to allow users to explore different results of multi-rate shader optimization rapidly. Similarly, the work in this study also seeks to reduce shading costs, but this goal is attempted with stereo rendering that shares shading between views. Thus, our contributions are orthogonal and complementary to the presented approaches.

He et al. [1] extended the traditional pipeline and designed two levels of shading stages, namely, coarse and fine fragment shading, to perform multi-rate shading. Vaidyanathan et al. [11] also have proposed a similar idea using coarse pixel shading to restrict and quantize shading rates to a finite set of screen aligned grids. However, these approaches were designed for a single view. It is still non-trivial to reuse shading across coarse pixels at stereo views.

Besides these screen space approaches, shading can also be reused in object space. For example, the offline realistic renderer, Reyes [28], computes and interpolates shading over micropolygons. Burns et al. [29] then proposed reusing shading among a uniform object space grid separate from the micropolygon grid. Clarberg et al. [10] further combined object space shading with multi-frequency shading and designed an object space multi-rate shading pipeline that performs multi-rate computation with different sizes of computation kernels on geometry patches. However, these approaches rely on tessellation patches and cannot reuse shading across original triangles. To address this problem, Hillesland et al. [30] performed variable rate shading at different texture LODs instead of on surfaces. Nevertheless, this approach requires certain pre-parameterization for each object, and this pre-parameterization not be feasible for scenes with many deformable objects or instancing objects, such as in games.

The proposed tile pair-based method computes shading of tiles in screen space and constructs corresponding tile pairs in object space. Therefore, this approach can comprise the shading frequency occurring both in screen and object spaces.

Sort-middle Rendering. Sort-middle rendering or binning rendering has been developed in graphics for quite some time. For example, in the Pixel Planes 5 project [31], many techniques for sort-middle rendering have been validated and invented. Because sort-middle rendering requires a large amount of computation to sort the geometry into screen-space bins or tiles in the middle of the graphics pipeline instead of near the end, sort-middle rendering architecture was barely used at desktop GPUs until recent generations. However, this architecture is popular for embedded GPUs [32], [33] because of the relatively low external memory bandwidth and the modest amount of on-chip memory required. In this work, the traditional sort-middle rendering is extended from one view to stereo views and propose a tile pair-based shading framework. Hasselgren and Akenine-Moller [2] also proposed a rasterization strategy to visit corresponding tiles at two views simultaneously. However, their proposal only rasterizes within one triangle and does not shade the entire scene from a perspective of tile pair-based rendering. The latter requires a newly designed data organization and scheduling scheme, which is the goal of our work.

3 Overview

Given the success of sort-middle shading architecture, the newly proposed shading pipeline shares the fundamental design philosophy that localizes triangle rasterization and pixel shading to small image regions and further extends it to stereo shading by introducing tile pair-based shading. In this new pipeline, tiles at two views are paired according to the underlying geometry correspondences and perform rasterization and shading simultaneously.

Such tile pair-based shading explores a basic observation that two cameras are naturally horizontally aligned in stereo rendering, and they usually have a small disparity. Therefore, pixels shaded at two tiles have high geometry correspondences, i.e., they present a high possibility for shading reuse at the scope of tile pairs. To explore such geometry correspondences among tile pairs, epipolar geometry is employed. In general, epipolar geometry is the

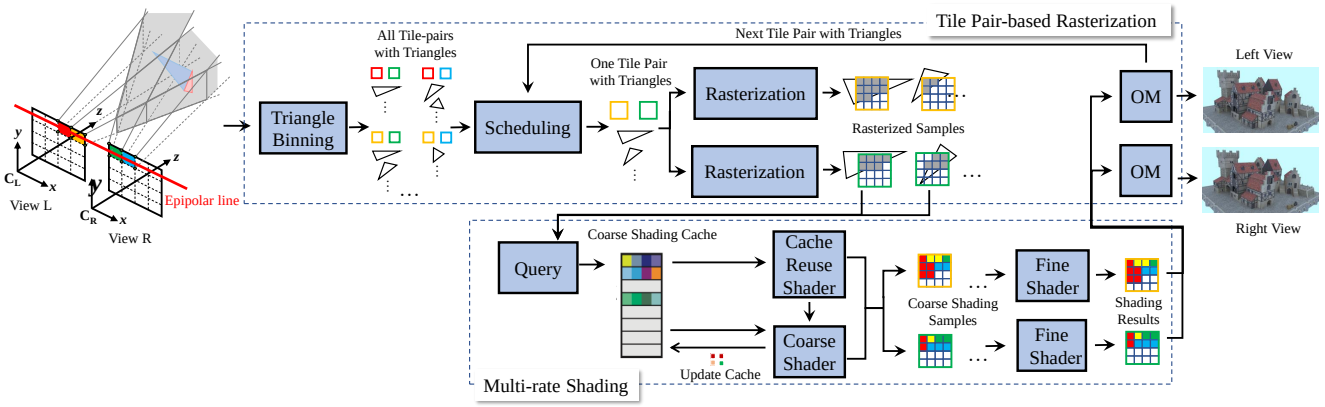


Fig. 2: *An architectural overview of our tile pair-based adaptive multi-rate stereo shading pipeline.*

intrinsic projective geometry between two views [34], leading to a set of constraints [35] between image points.Several of them are used in our tile pair-based stereo shading. When shading tile pairs, the shading of pixels is decoupled with different shading rates, i.e., coarse fragment shading and fine fragment shading. The shading results of coarse fragments of one view are cached and reused at the other view if possible to save shading costs at pixels with low-frequency shading signals among triangles and across two views.

Following the aforementioned idea, a tile pair-based multi-rate stereo shading pipeline is designed, and its architectural overview is shown in Fig. 2. At a high level, the entire shading process is organized into two stages: tile pair-based stereo rasterization and multi-rate shading.

Tile Pair-Based Rasterization. The rasterization stage mainly has four steps. First, stereo views are partitioned into regular tiles and a triangle binning algorithm is used to bin triangles into tiles and create tile pairs. Second, along each horizontal row, the contiguous tile pairs are visited and scheduled for rasterization at a depth-first manner. Third, triangle rasterization is performed at tiles of two views simultaneously to create shading samples. Finally, after all samples are shaded in multi-rate shading stage, they are merged with previous samples to output final images.

Multi-Rate Shading. In this stage, adaptive multi-rate shading is performed on rasterized samples of tile pairs. Following previous multi-rate shading methods [1], [11], the shading is divided into coarse and fine fragment shading to decouple different shading rates. While rasterized coarse samples are processed into this stage, newly designed cache and cache reuse shader are used to determine all possible reusable coarse shaded samples, and if valid cached shading samples exist, new samples are created by interpolating the cached shading samples. Otherwise, coarse fragment shadings are performed and the cache is updated. Fine fragments with low shading frequency are directly interpolated from coarse fragments, but those with high shading frequency are executed in traditional per-pixel shading.

4 Tile Pair-Based Rasterization

This section describes the details of the rasterization stage that integrates epipolar geometry into rasterization and generates shading samples at stereo views in a single draw operation. As shown in Fig. 2, stereo views are first partitioned into regular tiles. Triangles are then binned into these tiles at two views simultaneously. Also, several epipolar geometry constraints are considered in this tile pair-based rasterization. First, the *epipolar constraint* tells us that the shading of one pixel at one view only can be reused at the other view's pixels at the same epipolar line. As a result, tiles are only grouped at the corresponding epipolar line into pairs, greatly reducing the number of tile-pairs. Furthermore, the *order constraint* states that the order of neighboring correspondences on the epipolar line is always preserved for continuous opaque surfaces along that line. Thus, contiguous non-empty tile-pairs could be scheduled one by one along the epipolar line to localize shading calculation and improve the cache access during shading reuse (Sec. 4.2). Finally, the *disparity constraint* reveals that disparity values of pixels are inversely related to scene depth. Consequently, the tile pairs can be approximately sorted from the front to the back based on the difference of tile indexes of one tile pair, further reducing shading costs by combining with z-culling and early z-test . Although such a scheduling algorithm violates the current graphics APIs' ordering guarantees, it should be noted that out-of-order rasterization can be beneficial in many cases and has already been supported by current hardware [36]. Rasterized samples are outputted to the multi-rate shading stage, and once these samples are shaded, they are merged back with samples computed previously at this tile to update the color in the final image.

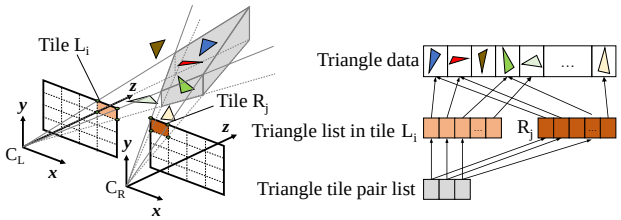


Fig. 3: *An illustration of tile-pair based triangle binning with generated triangle tile list and triangle tile pair list.*

These steps bear some similarities to sort-middle rendering but with several novel extensions for tile pairs. The following section describes the new triangle binning algorithm, presents the novel rasterization scheduling algorithm, and specifies stereo rasterization.

Algorithm 1: *Pseudo-code of tile-pair list construction*

```
1  TraverseEpipolarTile(tri, tilecoord_y)
2    // get leftmost and rightmost covered tile indices for both views
3    [L_s, L_e] = CoveredTiles(tri,tilecoord_y,Left)
4    [R_s, R_e] = CoveredTiles(tri,tilecoord_y,Right)
5    [I_Ls, I_Le] = InsertToTileList(tri,[L_s, L_e],tilecoord_y,Left)
6    [I_Rs, I_Re] = InsertToTileList(tri,[R_s, R_e],tilecoord_y,Right)
7    // only visible in one view, no tile pair
8    if [L_s, L_e] is empty or [R_s, R_e] is empty:
9      return;
10   // calculate PBs in tile boundaries
11   [T_Ls, T_Le+1] = TileBoundaryBarycentric([L_s, L_e],tilecoord_y)
12   [T_Rs, T_Re+1] = TileBoundaryBarycentric([R_s, R_e],tilecoord_y)
13   // calculate efficient measures
14   d = T_Ls+1 − T_Ls
15   compute [E_Ls, E_Le+1] [E_Rs, E_Re+1] as E = d · T
16   L_i = L_s; R_j = R_s
17   while L_i <= L_e and R_j <= R_e:
18     // insert to tile pair triangle list
19     if [E_Li, E_Li+1] overlap [E_Rj, E_Rj+1]:
20           InsertToTilePairList((L_i,R_j), (I_Li,I_Rj))
21     // visit next tile pair
22     if E_L^{i+1} < E_R^{j+1} : i+=1;
23     else if E_L^{i+1} > E_R^{j+1}: j+=1;
24     else: i+=1; j+=1;
```

4.1 Triangle Binning and Tile Pair Construction

Similar to the tile-based rasterization, a triangle list called triangle tile list is built for each tile. In addition, for each tile pair, triangles are further binned into a list called triangle tile pair list, storing triangles that overlap both tiles. Fig. 3 illustrates the data layout to store such two lists, and indexes of triangles in the triangle buffer are first stored in the tile list. Subsequently, indexes of triangles in the tile list are stored in the tile pair list. Such a double index buffer allows us to easily access the triangle tile list while processing the triangle tile pair list.

To bin triangles into these two lists, triangles are processed individually. Given one triangle, all tiles at two views that the triangle covers is computed, and then tile pairs from these tiles are constructed. Note that only two tiles sharing the same portion of geometry are a valid tile pair, meaning that only tiles having same vertical coordinates could be grouped into tile pairs. To distinguish between scanline and tile, we will call these tiles are at same **epipolar tile**. Therefore, the process per triangle and per epipolar tile could be parallelized. Whether two tiles share the same portion of a triangle is based on the test result of their overlaps in the triangle's object space. Specifically, the perspective-correct barycentric coordinates (PBs) are used and the tile-based multi-view traversal algorithm proposed by Hasselgren and Akenine-Möller [2] is adapted. The original tile-based multi-view traversal algorithm is used to sort the order of rasterization of tiles, thereby sorting all texture accesses to improve texture cache performance. However, the proposed method in this work uses the order of tiles and finds tiles sharing the same portion of geometry (i.e., the overlaps of tiles in the PB space). The key difference between the two methods is that the proposed method exploits the the coherence of shading in **stereo view space** defined on the entire scene, while what the previous method exploits the coherence of shading in the **PB space** defined on each triangle.

Alg. 1 shows the pseudo-code to construct the tile-pair list, and an illustrative example is given in Fig. 4. Alg. 1

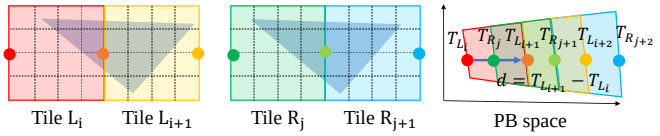


Fig. 4: *An example showing the triangle in screen space and its overlapping tiles in PB space. In this case, total three tile-pairs are created (L_i,R_j), (L_{i+1},R_j) and (L_{i+1},R_{j+1})*

first computes all the covered tiles along this epipolar tile, and add them into corresponding tile lists of views (lines 3-6), where indices of tiles in the tile lists, $[I_{L_s}, I_{L_e}]$ and $[I_{R_s}, I_{R_e}]$, are stored for constructing tile pair list. Then, for each tile, left and right boundary points are chosen at the center epipolar line as reference points (the colored dots in Fig. 4), and their PBs, $\mathbf{T}_{L_i}$ and $\mathbf{T}_{L_{i+1}}$ are then computed (lines 11-12), where L or R indicate the left or right view and i and j are the tile indices. Using reference points, the PB traversal direction $\mathbf{d}$ is computed along the center epipolar line. Note that tiles at two views share the same epipolar line. Therefore, the direction can be computed from only one tile, i.e., the left tile in our algorithm (line 14). Using d, the efficiency measures of reference points are calculated as $E = \mathbf{d} \cdot \mathbf{T}$ (line 15).

If this triangle is only visible to one view, no tile pair is constructed (lines 8-9). Otherwise, the proposed algorithm iteratively visits tiles from left to right. Also, in the PB space, tiles from both views can be sorted according to the efficiency measures, where overlaps of tiles only occur between adjacent tiles. Therefore, the efficiency measure of tiles and pair overlapped left and right tiles in the PB space are incrementally iterated (lines 19-24) and this procedure ends when all visible tiles are visited (line 17).

4.2 Scheduling Using Epipolar Geometry

Once all tile pairs are obtained, the rasterization of triangles on these tile pairs is then scheduled. The tile pair-based rasterization is performed in parallel at different epipolar tiles. For simplicity of exposition, one epipolar tile is used as an example to describe the scheduling approach and the tile pairs are processed from left to right by default. At one epipolar tile, if two tile pairs are not neighbor tile-pairs, the shadings at these two tile-pairs probably cannot be reused because no geometry continuity exists among these tiles. Therefore, to explore the consistent geometry surfaces, we design a two-step algorithm to first finds all neighboring tile pairs, and then uses the neighboring order to schedule rasterization of these tile pairs.

The first step scans all tile pairs at one epipolar tile from left to right. We start with the leftmost tile pair in the left view, which is denoted as (L_s, R_s). It is marked as the start tile-pair of one sequence of neighboring tile pairs, and it is used as a seed to trigger the search of neighboring tile pairs. For each seed, its neighboring tile pairs are fetched by the monotonously increasing index of tiles at two views. For example, (L_{s+1}, R_s), (L_{s+1}, R_{s+1}), and (L_s, R_{s+1}) are neighboring tile-pairs of the tile pair (L_s, R_s), and whether these neighboring tile pairs exist is checked. If any of these neighboring tile pairs are valid tile pairs, they are added to the sequence and marked as seeds to continue the search of neighboring tile pairs. If none of the neighboring tile pairs exist, then geometry continuity is absent at neighboring tile

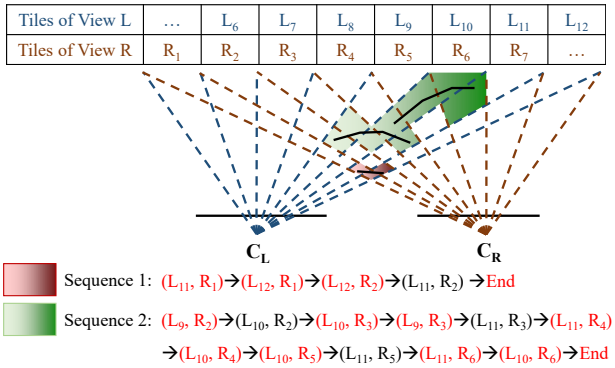


Fig. 5: *An example of tile pair scheduling using epipolar geometry. Two tile pair neighboring sequences are explored and used to schedule rasterization and shading of tile pairs.*

pairs, the search from this tile pair is stopped, and the search is restarted from another seed. The search is performed in a depth-first manner. Once all tile pairs of this sequences of neighboring tile pairs are collected, then another round of searching for other sequences of neighboring tile-pairs is started by selecting the leftmost tile-pair in the left view that has not been processed yet. Fig. 5 shows an example of two sequences of neighboring tile-pairs that have been constructed by the proposed algorithm. The sequence of neighboring tile pairs preserves the continuity of geometry surface, and a depth-first search is used instead of a bread-first search to sort the neighboring tile-pairs in good order even with a complex geometry surface. As shown in Fig. 5, the order of sequence 2 of neighboring tile pairs reflects the local continuity of geometry surface well, even in a scene with multiple layers of geometry surfaces.

Once the sequences of neighboring tile pairs are obtained, naive scheduling can be performed by iteratively processing all sequences as well as sequentially rasterizing individual tile pairs in sequence one by one. However, in the configuration of stereo views, the *disparity constraint* can further help accelerate rasterization and shading by combining with depth related optimizations. Specifically, the larger index of tile at the right view indicates the larger disparity with less scene depth. Also, in the first step, the leftmost unprocessed tile pair was always used to construct a sequence. Thus, the first tile pair of each sequence was sorted by the index of tile at the right view, and the neighboring tile pairs were scheduled under such an order where such an order approximately reveals depths of tile pairs in the sequence. The rasterization of tile pairs with less depths first, usually bring considerable performance gain when enabling depth related optimizations, such as hierarchy-z culling and early-z tests and these optimization techniques are common in modern hardware. Section 6.3 shows the benefits of enabling the scheduling optimization using such a *disparity constraint*.

4.3 Stereo Rasterization

When rasterizing triangles of one tile pair, rasterizations are simultaneously performed at two tiles, and all samples are outputted to our multi-rate shading stage. To avoid re-rasterizing triangles multiple times, once a triangle is first rasterized, this triangle in the triangle tile list is tagged by simply signing the index of this triangle to negative.

5 Multi-rate Shading

Once rasterized samples of triangles are obtained from the rasterization stage, we introduce our multi-rate shading stage that enables adaptive, multi-rate shading on these samples. Specifically, the shading is performed in image space, and the stage is separated into three steps: coarse, cache query and fine fragment steps.

As shown in Fig. 6, given a triangle-screen coverage, a block of four coarse fragments is emitted for shading (shown in blue), where each coarse fragment covers 2×2 or 4×4-pixel regions. A cache query is always performed before the invocation of the coarse fragment shader. If the query is missed, a coarse fragment shader is executed for coarse shading results (Fig. 6(a)). If reusable coarse shading results are queried at the same view, then they are directly resampled to fine fragment shading sample positions and combined with per-pixel texture in fine fragments to generate final shading results (Fig. 6(b)). If the queried results are not at the same view, then the cached values are first interpolated to coarse fragment sample positions at this view and then resampled to fine fragments (Fig. 6(c)).

The following sections describe the on-chip cache and its queries, introduce our cache reuse shader, and show how to use such a shader to reuse cached shading results.

5.1 Cache Queries

The storage and query of the on-chip cache share the concepts of cache design in AMFS [10] that a coarse fragment quad can be described by a quantized shading resolution **n** and the coarse fragment index **q**. In our case, all caching and querying of the coarse fragments are performed on image space instead of patch space, and we add an extra view flag to distinguish left and right views. Specifically, a hash function is used to index the coarse fragment, and it uses the following key:

$$\text{key} = h(q, \mathbf{n}, \mathbf{v}) \tag{1}$$

where h is an appropriately chosen hash function, and **v** represents the left or right view. Based on the selected shading rate, **n** is quantized to the power-of-two, e.g., 2×2 or 4×4, and **q** is the effective screen space position of that quad. Fig. 6 shows three different cases of our cache query. In Fig. 6(a), when one triangle (red one) is rasterized and coarse fragments are emitted, the key is first generated and queried at the coarse fragments' own view (left view in this case) and then it is queried at the other view if the first query is missed. In this case, where both views return cache misses, the coarse fragment shader is invoked and the resultant shading results are updated to cache. In Fig. 6(b), when the query returns a cache hit at the local view, the execution of coarse fragment shader is skipped, but the reuse shader is executed to test whether the cached shading values can be re-sampled to fine signals. In Fig. 6(c), where the cache hit is from the other view, we then project and interpolate coarse fragments from the cached values. Note that a coarse fragment represents fine fragments in 2×2 or 4×4 quad. After the projection, such a rectangle quad is usually distorted to a parallelogram p. To query such a parallelogram, like AMFS, we also quantize the extent of p to the power-of-two to ensure limited grid resolutions. Specifically, we quantize **n** by the projected pixel footprint

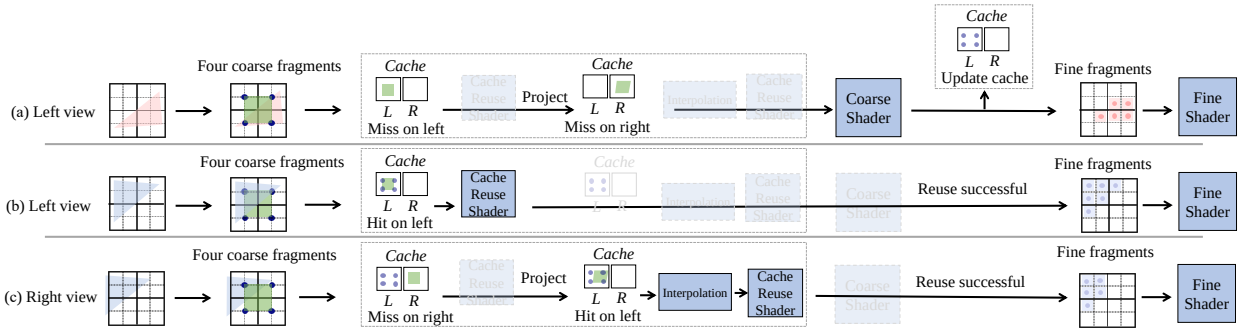


Fig. 6: *Illustration of multi-rate shading process at different cases. (a) After one triangle is rasterized, four coarse fragments are emitted for shading. These coarse fragments are first queried on cache. If a cache miss occurs on both left and right views, then the coarse fragment shader is executed and resampled to fine fragments. (b) If a cache hit occurs at the local view (left view in this case), then the cache reuse shader is executed. If it returns true, then the cached values are reused for fine fragments. (c) If a cache hit occurs at the other view, then the cached values are first interpolated to the local view and then processed for fine fragments.*

Algorithm 2: *An example of cache reuse shader*

```
struct Coarse_In{ float3 norm, pos; }
struct Coarse_Out{ float diff, spec, rdotl; bool refine_flag; }
uniform float3 eye_pos[2], light_dir;
bool reuse(Coarse_In in, Coarse_out out,
int query_view_id,int cache_view_id){
  // evaluate whether cached value can be reused or not
  if(len(in.pos − CachedIN.pos) > THRESHOLD0) return false;
  if(len(in.norm − CachedIN.norm) > THRESHOLD1) return false;
  // cross view check
  if( cache_view_id != query_view_id){
    float3 view = eye_pos[query_view_id] − in.pos;
    float rdotl = dot(reflect(view,in.norm),light_dir);
    if(abs(rdotl − CachedOUT.rdotl)> THRESHOLD2) return false;
  }
  out = CachedOUT;
  return true;}
```

p_x as $n = 1 << log2(p_x)$. The projected size is only quantized at the x-axis because the distortion only exits along x. Fig. 6(c) shows an example of coarse fragment projection and query. The coarse fragment position may not be perfectly aligned with the cached fragments. Therefore, the inputs of cached coarse quad are bilinearly interpolated to align them with the query coarse quad.

Cache data is stored on a fast-on chip memory and operated similar to that proposed by Ragan Kelly et al. [12]. The least recently used cached data is evicted, when the cache is full. In the implementation, a fixed size on-chip shading cache is pre-allocated for each epipolar tile. For each epipolar tile, the cache is only cleared when a neighboring tile-pair sequence is fully processed, because all the shading coherences along this sequence are no longer used.

5.2 Cache Reuse Shader

One issue remains before the queried shading result is used. Specifically, many distinctly different surfaces are mapped on the same pixel region. For example, the surface in (L_{11},R_1) and (L_{11},R_4) in Fig. 5 both map to tile L_{11}, but the shading results should not be reused among them. Thus, after every successful query, a cache reuse shader is used to test whether the shading results of two surfaces are reusable. Our cache reuse shader is designed on the basis of an observation that for certain shading effects (e.g., diffuse and specular lighting and shadows), if the outputs of coarse fragments can be reused across other coarse fragments, then the inputs to these fragment shaders (e.g., fragment sample positions and triangle attributes) should be similar. Thus, the differences of inputs of fragment shaders could be used to predict the similarity of outputs, and outputs can be reused when inputs are similar. Such a frequency check shader bears some similarities to the merge function proposed by Fatahalian et al. [20], but the execution of our shader does not require adjacent information of the geometry; thus, the test can be performed in more general cases, such as shading reuse for different meshes or views.

Alg. 2 lists an example code used. We evaluate whether the lighting computation can be reused by comparing the input of positions and normals, as well as reflected lighting direction at two views. For each executed coarse fragment, the input and output are stored in the cache as **CachedIN** and **CachedOUT**, respectively. Once a cache query from a new coarse fragment is triggered, the cache reuse shader is executed to compare the inputs of this new coarse fragment (i.e., position, normal, and reflected lighting direction), with cached inputs. If the cache reuse query passes, then the cached outputs of coarse fragments are reused; thus, the actual computations of the coarse fragment shader are saved. However, if the test fails, this shader returns false to allow the actual computations of coarse fragments.

Note that in our pipeline, coarse fragments are always shaded in blocks of four to support interpolation and finite difference calculations. Consequently, the actual computations of coarse shader are skipped only when all four coarse fragments are reused. To handle the divergence of executions caused by cache query and reuse shader execution, we adopted the same two-phase scheduling method that has been used in several multi-rate shading papers [1], [10]. Specifically, coarse fragments having different reuse results are sorted to different fragment processing queues. The fragments that fail to reuse cached results are queued for executing fine fragment shaders, and those coarse fragments that can be reused are queued to interpolate shading values.

6 Results

A CPU-based software simulator, adopted from the CPU implementation of work [15] is implemented to evaluate our proposed pipeline. The simulator runs shaders by using SIMD instructions through an abstraction similar to that of the Microsoft HLSL shading language. For coarse and fine fragments, we use an interface similar to that proposed

	Arena	Sponza	Fairy	Castle*
#Tri	138K	265K	170K	1644K
Pixels/Tri	125	161	102	17
Tile 8×8				
#Tile/Mem	48K/1.5	58K/2.7	58K/2.1	46K/5.1
#Tile-pair/Mem	61K/1.2	120K/1.9	75K/1.5	71K/3.5
Tile 16×16				
#Tile/Mem	12K/0.8	14K/1.4	14K/1.1	11K/3.2
#Tile-pair/Mem	14K/0.6	24K/0.9	17K/0.7	16K/1.95
Tile 32×32				
#Tile/Mem	3K/0.5	3.6K/0.9	3.6K/0.8	2.8K/2.3
#Tile-pair/Mem	3.3K/0.3	5K/0.5	4.1K/0.4	3.6K/1.36

TABLE 1: *Statistics of four test scenes (Castle* is a tessellated scene). Columns "#Tile" and "#Tile-pair" refer to the number of tiles and tile pairs, respectively, with their storage cost (Mem in megabytes) presented.*

by He et al. [1]. Several rendering optimizations, such as back-face culling, hierarchical z-culling and early-z test techniques were also used.

The pipeline was tested on five scenes, namely, Arena (Fig. 1) Sponza, Fairy, Castle (Fig. 10) and a small test scene Ogre (Fig. 9) for various tessellation levels and methods. The Arena scene includes a dragon with skinning animation and the Castle scene is subdivided once. In these scenes, different shading effects, such as diffuse and specular lighting [1], [10] and 5×5 PCF shadowing [1], are enabled. The statistics of these four scenes are presented in Table 1. The inter-camera distance is set at 70mm, as suggested by Oculus [37]. All the results are rendered at 2560×1440 resolution (1280×1440 for each view) with a tile configuration of 16×16 pixels unless otherwise specified.

We first analyze the proposed scheduling algorithm. All the memory reads/writes during rasterization and shading are reported and compared with the traditional sort-middle renderer as well as the multi-view rasterization framework [2]. The results of the proposed method were compared with several multiple rate/frequency shading methods, such as that of He et al. [1], AMFS [10], and Texel shading [30]. Also the recommended DYNAMIC4×4 mode is the default configuration in the coarse fragment stage, and the static corner is the coarse sample position when implementing our method and that of He et al. [1]. To compare these methods fairly, we adopt the global cache based shading in AMFS [10] and Texel shading [30] to use a local cache like ours, and employ the same frequency detection proposed in [1] (i.e., check surface flatness, reflection lobe and shadow penumbra). In addition, we enable our tile pair-based scheduling for rasterization while comparing these methods. In this way, the differences of the resultant shading rates of these methods are only from different shading reuse frameworks. Except for the work of He et al. [1], which is not a cache based method, all other methods are backed by a 24 KB fast on-chip shading cache that stores both the inputs (16-bit floats) and the outputs of the coarse fragments (32-bit floats).

The comparison of an entire 150-frame animation is presented in the supplementary video. During these evaluations, the results of different cache and tile sizes, and optimizations used in our method are studied.

6.1 Tile pair Scheduling

In this subsection, we first analyze the memory footprint of our newly proposed tile pair-based scheduling and further compare the bandwidth of our approach with those of the traditional sort-middle rendering and the tiled multi-view rasterization method [2].

Memory Footprint. To utilize the shading coherence between two views, two types of lists are presented: triangle tile list and triangle tile pair list. The latter is newly proposed in our method, and the numbers and memory footprints of these two types of lists at different tile sizes for the four test scenes are presented in Tab. 1. Each entry of triangle tile list and triangle tile pair list is represented by a 32-bit unsigned int and a 16-bit short, respectively. In most cases, the number of tile pairs is less than one and a half of the number of tiles, except in the Sponza scene with tile size of 8×8. This finding indicates that a small tile size configuration in a scene with a high depth complexity should be avoided. The numbers of tiles and tile pairs decrease rapidly as the tile size increases.

Although the number of triangle tile pair lists is higher than the number of triangle tile lists, the memory for storing the former is less than that for the latter because the number of triangles overlapping one tile pair is considerably less than the number of triangles overlapping the entire tile. As shown in Table 1, we only require 50% - 70% memory space of triangle tile lists to store triangle tile pair lists, which is negligible compared with other resources.

Bandwidth. Off-chip memory bandwidth is an important criterion to realize a shading pipeline. To compare the proposed scheduling method with previous methods, such as the traditional sort-middle rendering and the tiled multi-view rasterization method proposed by Hasselgren et al. [2], we collect runtime memory usage by tracing shading executions on per-pixel shading with diffuse texture and shadow effect. The L1 cache system was extensively used at various stages (e.g. vertex cache and depth/color/texture cache) in our simulator, as it is a common practice. Data is accessed with 256 bytes cache line. As suggested by Clarberg et al. [14], the relation of different system costs highly depends on the specific hardware implementation, so different bandwidths are reported separately to better review our design. Overall, four types of data buffers are involved in our pipeline. *Depth+Color* includes the 32-bit depth buffer and the RGBA float color buffer with read/write accesses. The bandwidth of depth and color is compressed four times and two times, respectively, as those proposed in the work of Seiler et al. [38]. *Indices* buffer includes reading/writing the triangle tile and triangle tile pair lists. The *Triangle Data* refers to the bandwidth of triangle data used for triangle rasterization, and the output vertex data generated by the vertex shader. In the case studied in this work, the data comprises eight floats (vertex position, vertex normal and texture coordinate). *Texture* is the total amount of texture accesses during shading. The multi-view rasterization method does not have *Indices*. The standard sort-middle shading also requires four types of data buffers, but its bandwidth of *Indices* does not include the bandwidth of reading triangle tile pair lists.

Fig. 7 shows the results of experiments on different

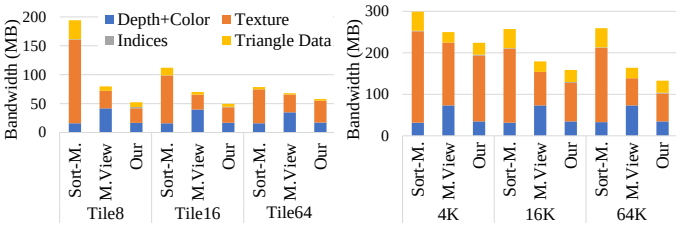


Fig. 7: *Traditional sort-middle framework (Sort-M.), multi-view rasterization method (M.View) proposed in [2] and our tile-pair scheduling are tested on different configurations. Left: Arena scene with fixed 16KB texture cache size. Right: Sponza scene with 16×16 tile. Our tile pair-based shading always has a lower memory bandwidth.*

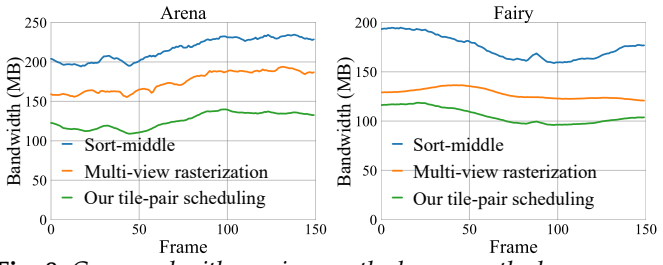


Fig. 8: *Compared with previous methods, our method consumes a considerably lower memory bandwidth over the entire 150-frame animation*

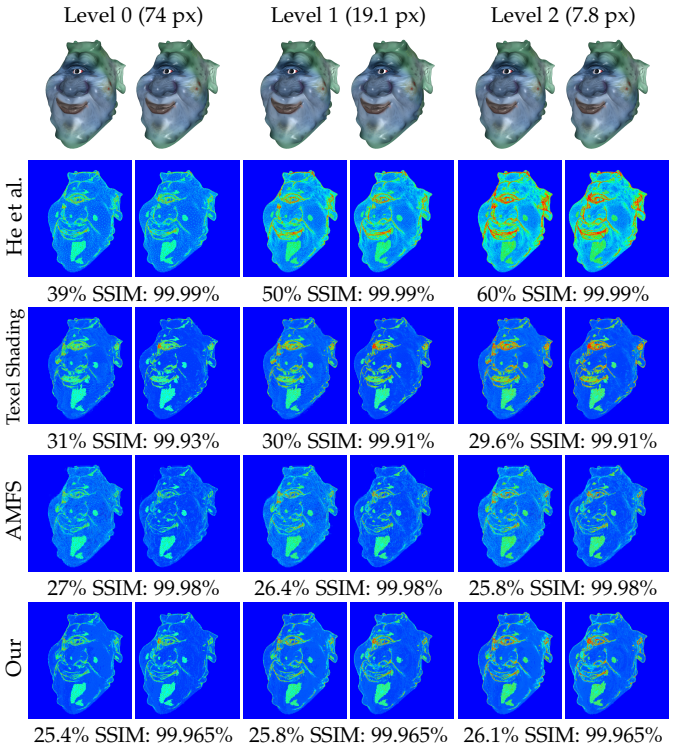


Fig. 9: *The Ogre scene at three levels of PN-triangle [39] subdivision. Similar to two previous methods, Texel shading [30] and AMFS [10], the shading rate of our method does not increase significantly with more levels of subdivision.*

configurations. Two key aspects are explored. In Fig. 7(a), under the same texture cache size (16KB), the effects of different tile configurations are shown. Fig. 7(b) shows that the bandwidth decreases with the increased cache size. It should be noted that because our method requires rasterizing two views simultaneously, for all methods, the size of depth and color cache are set to be able to hold two tiles at the same time. Thus, when the tile size is 16×16, the depth and color cache is 2KB and 8KB, respectively. Of all the methods under different configurations, our tile pair-based approach consumes the least memory bandwidth. The traditional sort-middle method (Sort-M.) requires the least amount for depth/color read/write, but the cache miss rate for accessing texture is high, and the bin-spread is an important issue when the tile is small. In the work of Hasselgren et al. [2] (M.View), they scheduled the rasterization along the texture access order to reduce much of the bandwidth needed for accessing texture. However, their method negatively affects the locality of the framebuffer. Our method requires slightly greater bandwidth to access the depth/color buffer. As it may visit some tiles multiple times. It does not follow the texture access order strictly, so the texture bandwidth is slightly larger than that in [2]. However, our proposed tile pair scheduling has the benefits of both methods, achieving the least bandwidth at all tile configurations. In addition, at a fixed size of texture cache, when the tile size increases, our method closely approaches to traditional sort-middle method, as shown in the 64×64 configuration. Larger texture cache can be beneficial for our method. In Fig. 7(b), when the texture cache size increases, the difference of the texture cache miss rate between our method and that presented by Hasselgren et al. [2] is greatly reduced. Furthermore, Fig. 8 shows that the performance of our method is stable from frame-to-frame and consistently outperforms previous methods.

6.2 Multi-rate shading

In this subsection, we compare several multi-rate approaches, such as that of He et al. [1], AMFS [10], and Texel shading [30] both at shading rates and image quality.

Shading Rates. Comparison of multi-rate shading with traditional per-pixel shading shows that all multi-rate methods remarkably reduce the shading cost by reusing low-frequency shading results, such as shadow umbra and diffuse regions. However, in the work of He et al. [1], when two triangles covere the same region, two executions of coarse shader quads are invoked, as shown in the heat map where higher shading costs are noted at the boundaries of triangles than on the inside. As shown in Fig. 9, the shading costs of He et al.'s method increase with decreased average pixel coverage of a triangle. The results in Fig. 1, 9, and 10 show that our scheme can further reuse shading across triangles and views, thereby allowing us to achieve a considerably lower shading cost (30%∼40% costs compared with that in per-pixel shading, and less than two-thirds of the cost of the method proposed by He et al. [1] in the Sponza scene). Fig. 1(b) shows a detailed breakdown of instructions executed at different stages ("FineInst.","CoarseInst" and "ReuseInst." represent the instructions executed at per-pixel shading, 4×4 or 2×2 coarse shading, and our cache reuse shader execution, respectively). By executing the simple cache reuse shader, we can reuse more than half of the coarse shading result, thereby greatly reducing the shading cost.

We also conduct experiments to compare our method with the multi-rate shading approaches in other spaces,

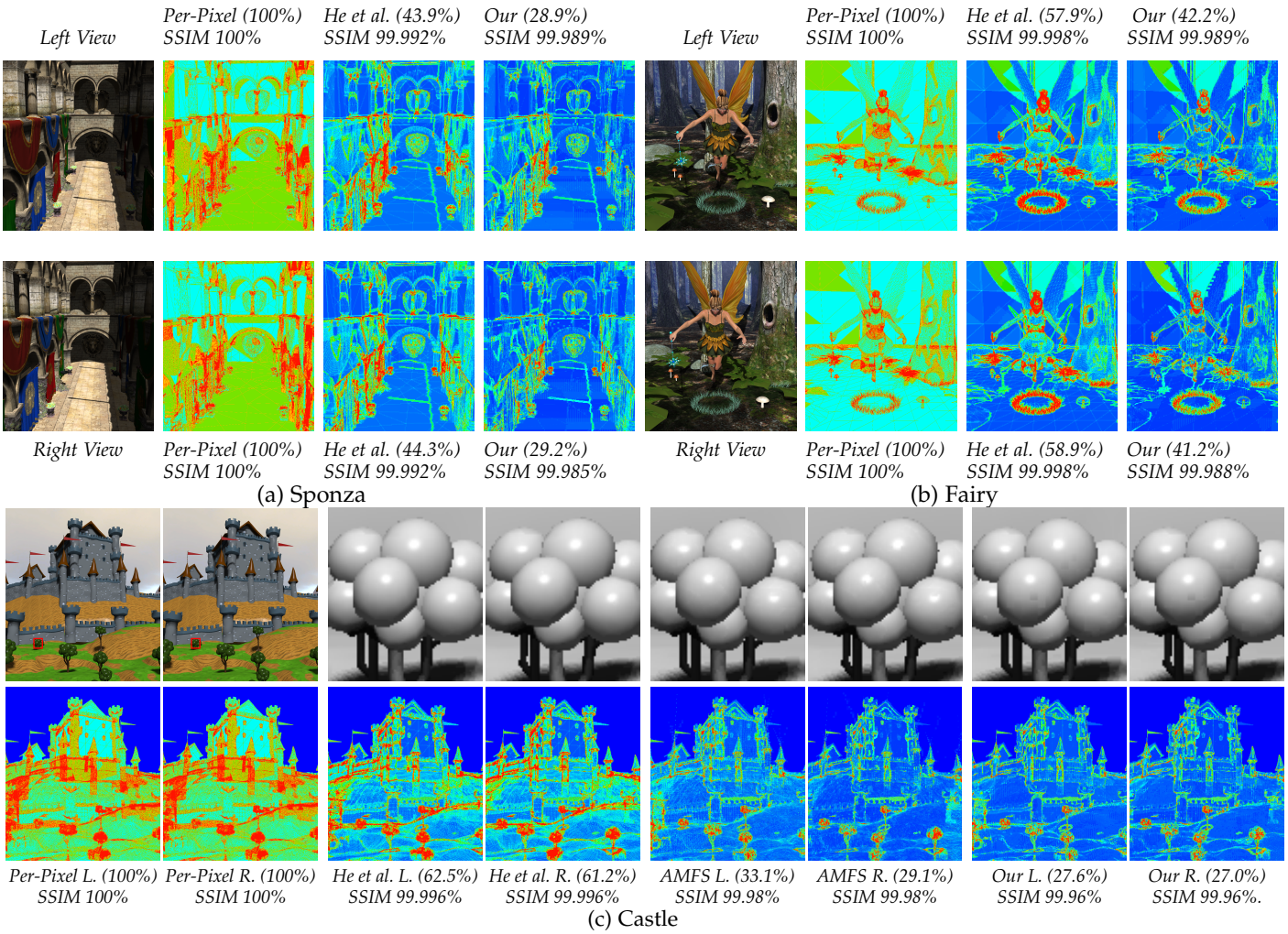


Fig. 10: *Comparison of shading methods in Sponza, Fairy and Castle scenes, relative shading cost is shown in brackets.*

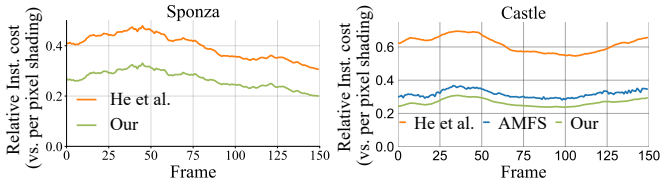


Fig. 11: *Comparison of relative shading costs of several multi-rate shading methods over 150-frame sequence. The shading cost of tile-based shading is set as 100%.*

namely, the patch space used in AMFS [10] and the texture space in Texel shading [30] (we only compare Texel shading in the Ogre model because this method requires a non-overlapping texture parameterization, which is not attainable in other scenes). These two methods perform shading execution and cache query at spaces that are independent of view and can span multiple triangles. Thus, both methods scale well when tessellation level increase. The Texel shading performs worse because it tends to incorrectly estimate the frequency at regions with high parameterization distortion such as surfaces with high curvatures (around Ogre's eye and lips). The shading rate of our method is better than that of AMFS at first level because we can reuse shading results across patch boundaries. However, when triangles became increasingly smaller, the additional overhead of our cache reuse test increase the shading cost slightly. The Castle scene in Fig. 10(c) shows a case in which, compared with AMFS [10], the ability to reuse across patches can further reduce approximately 4% of the shading cost. Compared with these two methods, our approach is more general because it does not have additional requirements regarding the input scene, e.g., tessellation patch or specific parameterization. Fig. 11 shows that the shading costs using our algorithms are stable and consistently lower than that of other methods.

Image Quality. Based on the SSIM errors, all the multi-rate methods preserve very high visual quality (over 99.95% visual similarity). Zoomed insets of different methods are presented in the top row of Fig. 10(c). To better distinguish between methods, we only show the lighting result and adjust the gamma. The error of the multi-rate method [1] is barely noticeable compared with the per-pixel result (thus, the per-pixel result was omitted). Our method can reuse shading across neighboring triangles. In this highly curved sphere, blocks artifacts with our technique are due to we reuse the extrapolation outside previous triangle boundaries. The AMFS has inconsistent shading when reusing shading across the triangle in the patch. Besides, the specular highlight in the right view of AMFS is distorted since it

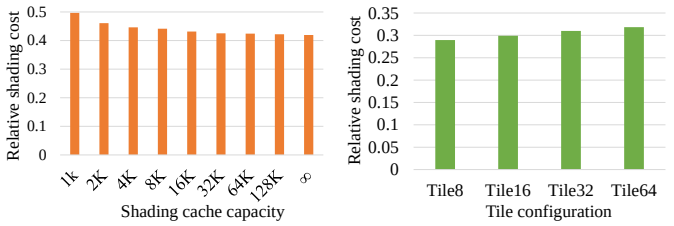
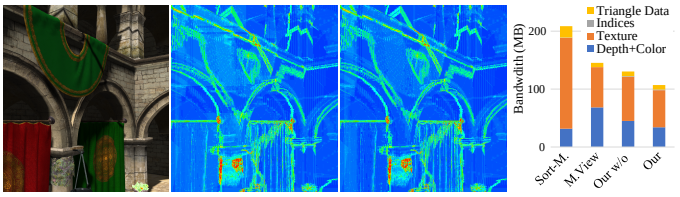


Fig. 12: *We test our method with different shading cache sizes under a fixed 16×16 tile configuration (Left, Fairy scene). A 24 KB cache is usually sufficient to cache both the shading input and output for reuse shading across triangles and views. Additionally, we found that under fixed 24KB cache size (Right, Sponza scene), with larger tile size, the triangle number inside one tile-pair also increases, thereby stressing the fixed size shading cache and reducing the shading reuse ratio. However, the impact is still small.*



Frame 149 *w/o constraint (22%)* *w/ constraint (19.5%)* *Bandwidth*

Fig. 13: *Comparison with or without utilizing disparity constraint in tile pair scheduling.*

directly uses the result of the left view. Our method use the cross view check code (Alg. 2) to avoid the distortion caused by reusing the specular highlight.

6.3 Optimization with Disparity Constraint

In our method, several epipolar geometric constraints are utilized to guide the rasterization and shading of tile pairs. The proposed optimization with disparity can provide additional benefits when depth complexity is high. In Fig. 13, a visualized per-pixel shading cost at one frame with and without such an optimization is presented. The results show that such an optimization of *disparity constraint* can further reduce shading costs and memory bandwidth. For that one frame, shading cost is reduced from 22% to 19.5%, and memory bandwidth is saved 20%.

7 Conclusion and Future Work

We present a novel shading pipeline for stereo rendering that can effectively reduce shading costs and achieve less memory bandwidth. Several novel concepts have been presented and validated in this paper. First, the newly introduced tile pair shading utilizes epipolar geometry to schedule tile pairs, which consider stereo views at rasterizing triangles, thereby better utilizing geometric correspondences among triangles and between views. Second, multi-rate shading performed among tiles and triangles efficiently explores low-frequency shading signals and reduces shading cost. All the aforementioned improvements on runtime data bandwidth, memory storage, and computational performance are essential for modern graphics hardware. We believe that the proposed tile pair-based stereo shading pipeline will inspire future graphics hardware for stereo rendering.

Several directions are also worthy of further investigation. First, it would be interesting to extend our current shading pipeline to general multi-view shading. In our current implementation, we consider only stereo views, in which epipolar lines are naturally aligned. For multiple views, epipolar geometry still holds for two or more views, in which camera centers are at a coplane. Views can be possibly rectified to share epipolar lines; however, this process will cause distortions of pixels and tiles. Solving these distortions and extending the tile pair-based multi-rate shading pipeline to multiple views are interesting future topics.

Second, the use of tile pairs in our shading pipeline partitions the entire scene into spatial grids through tile pair frustums. However, such partitioning is irregular and independent of scene complexity. Therefore, the number of triangles in each tile pair may vary dramatically across a scene. Occasionally, this scenario may become critical for the parallelism of rasterization and shading computation. An improved partitioning in geometric space, instead of by tile pairs, is another interesting future direction for stereo or multi-view rendering.

Third, our approach is designed by improving the rasterization and shading locality through tile pairs in forward rendering pipeline. However, the idea also has potential to be applied in deferred rendering to reduce the cost of stages with geometry rasterization and shading. For example, our tile pair-based rasterization can be used to reduce the memory bandwidth in generating stereo G-buffers. In addition, the scheme of reusing cross view shading can also be adopted for screen space shading pass. However, owing to the difference between forward and deferred rendering, we regard it as the extension of our framework as future work.

Finally, another interesting direction is to combine our pipeline with existing automatic shader simplification methods [26], [27] that enable a system to automatically decompose shaders or compute pixels into coarse and fine fragments to fully utilize the proposed multi-rate shading.

Acknowledgements

We would like to thank all reviewers for their thoughtful comments.WealsowanttothankYiweiHuandJialiPanfor preparing demos and video. This research was partially funded by National Key R&D Program of China (No. 920 2017YFB1002605), NSFC (No. 61872319), Zhe-jiang Provincial NSFC (No. LR18F020002) and the Fundamental Research Funds for the Central Universities (No. 2017FZA5012).

References

[1] Y. He, Y. Gu, and K. Fatahalian, "Extending the graphics pipeline with adaptive, multi-rate shading," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, p. 142, 2014.

[2] J. Hasselgren and T. Akenine-Möller, "An efficient multi-view rasterization architecture." *Rendering Techniques*, vol. 2006, p. 17th, 2006.

[3] J. Carmack, "John carmack's delivers some home truths on latency," http://oculusrift-blog.com/john-carmacks-message-of-latency/, 2014.

[4] S. J. Adelson, J. B. Bentley, I. S. Chong, L. F. Hodges, and J. Winograd, "Simultaneous generation of stereoscopic views," in

Computer Graphics Forum, vol. 10, no. 1. Wiley Online Library, 1991, pp. 3–10.

[5] M. Halle, "Multiple viewpoint rendering," in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. ACM, 1998, pp. 243–254.

[6] S. Fu, H. Bao, and Q. Peng, "An accelerated rendering algorithm for stereoscopic display," *Computers & Graphics*, vol. 20, no. 2, pp. 223–229, 1996.

[7] M. Wan, N. Zhang, H. Qu, and A. E. Kaufman, "Interactive stereoscopic rendering of volumetric environments," *IEEE Transactions on Visualization and Computer Graphics*, vol. 10, no. 1, pp. 15–28, 2004.

[8] L. Zhang and W. J. Tam, "Stereoscopic image generation based on depth images for 3d tv," *IEEE Transactions on broadcasting*, vol. 51, no. 2, pp. 191–199, 2005.

[9] A. Bulbul, Z. Cipiloglu, and T. Capin, "A perceptual approach for stereoscopic rendering optimization," *Computers & Graphics*, vol. 34, no. 2, pp. 145–157, 2010.

[10] P. Clarberg, R. Toth, J. Hasselgren, J. Nilsson, and T. Akenine-Möller, "Amfs: adaptive multi-frequency shading for future graphics processors," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, p. 141, 2014.

[11] K. Vaidyanathan, M. Salvi, R. Toth, T. Foley, T. Akenine-Möller, J. Nilsson, J. Munkberg, J. Hasselgren, M. Sugihara, P. Clarberg *et al.*, "Coarse pixel shading," in *High Performance Graphics*, 2014.

[12] J. Ragan-Kelley, J. Lehtinen, J. Chen, M. Doggett, and F. Durand, "Decoupled sampling for graphics pipelines," *ACM Transactions on Graphics (TOG)*, vol. 30, no. 3, p. 17, 2011.

[13] K. Vaidyanathan, R. Toth, M. Salvi, S. Boulos, and A. Lefohn, "Adaptive image space shading for motion and defocus blur," in *Proceedings of the Fourth ACM SIGGRAPH/Eurographics conference on High-Performance Graphics*. Eurographics Association, 2012, pp. 13–21.

[14] P. Clarberg, R. Toth, and J. Munkberg, "A sort-based deferred shading architecture for decoupled sampling," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, p. 141, 2013.

[15] S. Laine and T. Karras, "High-performance software rasterization on gpus," in *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*. ACM, 2011, pp. 79–88.

[16] F. De Sorbier, V. Nozick, and H. Saito, "Multi-view rendering using gpu for 3-d displays," *GSTF Journal on Computing (JoC)*, vol. 1, no. 1, 2014.

[17] NVIDIA, "NVIDIA VRWorks^{TM}," https://developer.nvidia.com/vrworks, 2016.

[18] C. Fehn, "Depth-image-based rendering (dibr), compression, and transmission for a new approach on 3d-tv," in *Electronic Imaging 2004*. International Society for Optics and Photonics, 2004, pp. 93–104.

[19] K. Akeley, "Reality engine graphics," in *Proceedings of the 20th annual conference on Computer Graphics and Interactive Techniques*. ACM, 1993, pp. 109–116.

[20] K. Fatahalian, S. Boulos, J. Hegarty, K. Akeley, W. R. Mark, H. Moreton, and P. Hanrahan, "Reducing shading on gpus using quad-fragment merging," *ACM Transactions on Graphics (TOG)*, vol. 29, no. 4, p. 67, 2010.

[21] G. Liktor and C. Dachsbacher, "Decoupled deferred shading for hardware rasterization," in *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. ACM, 2012, pp. 143–150.

[22] C. Crassin, M. McGuire, K. Fatahalian, and A. Lefohn, "Aggregate g-buffer anti-aliasing," in *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games*. ACM, 2015, pp. 109–119.

[23] S. Kircher and A. Lawrance, "Inferred lighting: fast dynamic lighting and shadows for opaque and translucent objects," in *Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games*. ACM, 2009, pp. 39–45.

[24] J. V. Natalya Tatarchuk, Chris Tchou, "Mythic science fiction in real time: Destiny rendering engine," in *ACM SIGGRAPH 2013 Courses: Advanced in Real-Time Rendering in Games*, 2013.

[25] J. Hasselgren and T. Akenine-Möller, "Pcu: the programmable culling unit," in *ACM Transactions on Graphics (TOG)*, vol. 26, no. 3. ACM, 2007, p. 92.

[26] R. Wang, X. Yang, Y. Yuan, W. Chen, K. Bala, and H. Bao, "Automatic shader simplification using surface signal approximation," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 6, p. 226, 2014.

[27] Y. He, T. Foley, and K. Fatahalian, "A system for rapid exploration of shader optimization choices," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 112, 2016.

[28] R. L. Cook, L. Carpenter, and E. Catmull, "The reyes image rendering architecture," in *ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4. ACM, 1987, pp. 95–102.

[29] C. A. Burns, K. Fatahalian, and W. R. Mark, "A lazy object-space shading architecture with decoupled sampling," in *Proceedings of the Conference on High Performance Graphics*. Eurographics Association, 2010, pp. 19–28.

[30] K. E. Hillesland and J. C. Yang, "Texel shading," in *Proceedings of the 37th Annual Conference of the European Association for Computer Graphics: Short Papers*, ser. EG '16. Goslar Germany, Germany: Eurographics Association, 2016, pp. 73–76.

[31] H. Fuchs, J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, B. Tebbs, and L. Israel, "Pixel-planes 5: a heterogeneous multiprocessor graphics system using processor-enhanced memories," in *ACM Siggraph Computer Graphics*, vol. 23, no. 3. ACM, 1989, pp. 79–88.

[32] PowerVR, "PowerVR: A master class in graphics technology and optimization," in *Imagination Technologies*, 2012.

[33] ARM, "Mali rendering strategy," http://infocenter.arm.com/, 2016.

[34] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, ISBN: 0521540518, 2004.

[35] A. Koschan, *Eine Methodenbank zur Evaluierung von Stereo-Vision-Verfahren*. Technische Universität Berlin, Fachbereich 20, Informatik, 1991.

[36] KhronosGroup, "Vulkan," https://www.khronos.org/registry/vulkan/, 2015.

[37] V. Oculus, "Oculus rift," *Available from WWW:¡ http://www. oculusvr. com/rift*, 2015.

[38] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin *et al.*, "Larrabee: a many-core x86 architecture for visual computing," in *ACM Transactions on Graphics (TOG)*, vol. 27, no. 3. ACM, 2008, p. 18.

[39] A. Vlachos, J. Peters, C. Boyd, and J. L. Mitchell, "Curved pn triangles," in *Proceedings of the 2001 symposium on Interactive 3D graphics*. ACM, 2001, pp. 159–166.