Temporally Dense Ray Tracing

P. Andersson, J. Nilsson, M. Salvi, J. Spjut, and T. Akenine-Möller

NVIDIA

Abstract

We present a technique for real-time ray tracing with the goal of reaching 240 frames per second or more. The core idea is to trade spatial resolution for faster temporal updates in such a way that the display and human visual system aid in integrating high-quality images. We use a combination of frameless and interleaved rendering concepts together with ideas from temporal antialiasing algorithms and novel building blocks—the major one being adaptive selection of pixel orderings within tiles, which reduces spatiotemporal aliasing significantly. The image quality is verified with a user study. Our work can be used for esports or any kind of rendering where higher frame rates are needed.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Ray Tracing

1. Introduction

Ray tracing has become the dominant visibility algorithm in film and is growing in use for real-time applications thanks to the introduction of the Microsoft DXR and Vulkan ray tracing APIs, and NVIDIA RTX with GPU-hardware acceleration. While ray tracing is becoming more accessible to real-time applications, more performance is needed, particularly to hit the high refresh rates desired for performance-critical applications. One such application is *esports*, or competitive video games, which pit highly skilled professionals against each other and require the athletes to compete both in response time and strategy. In these competitive scenarios, every bit of advantage has the potential to determine which team wins or loses. The latency reduction provided by higher refresh rate displays (e.g., 16.7 ms at 60 Hz becomes 4.2 ms at 240 Hz) has been shown to provide a corresponding competitive advantage in certain esports-related tasks [KSM*19].

Given these potential advantages of higher frame rates and emerging availability of high performance real-time ray tracing, it is natural to want use ray tracing to spread samples across time and enable a higher frequency of final frames, without increasing the total number of samples. *Frameless rendering* is a technique where each ray or sample are given their own unique time [BFMZ94, DWWL05]. While frameless rendering is good at providing continuous updates of the world simulation, it requires running the world simulation for each sample taken, which in practice is a high cost. Therefore, a more moderate solution is to use an approach like *interleaved sampling* [KH01] to cover the spatial sampling desired over some number of frames, each of which has samples located at pseudo-random positions. These samples should then be combined using accumulation buffering [HA90] to produce final frames. Spatiotemporal upsampling of only *shading*, while still using full resolution visibility and material information, can also be used to reduce shading cost [HEMS10]. Temporal an-
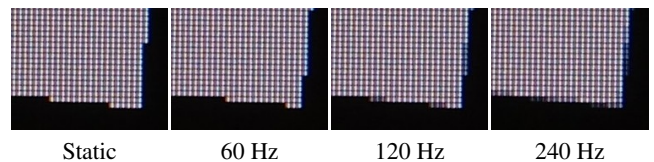

Static	60 Hz	120 Hz	240 Hz

Figure 1: *Static edge as antialiased over time with a jittered single sample per pixel. Photographs taken with 16.67 ms exposure time.*

tialiasing (TAA) [Kar14] uses subpixel offsets and accumulation over many frames and has been widely adopted in video games. TAA uses a variety of heuristics [PSK*16], often tuned per application, to reduce the blurring, ghosting, and other artifacts commonly seen. Checkerboard rendering is another related method where 50% of the pixels are rendered each frame, and reprojection and averaging are used to produce the final frame [EM16].

Our work builds on all the methods mentioned above. In addition, we adapt the algorithms to ray tracing at 240 Hz and present a fundamental insight about how the sampling enumeration inside tiles can reduce edge aliasing substantially if done well and in the right order. Our results include an increase in performance by a factor of approximately $3.5\times$ with increased quality, compared to TAA, and with the possibility to meet the 240 FPS goal, instead of running at 60 FPS, or slightly above.

2. The Visual Power of 240 Hz

It is challenging to convey what 240 Hz displays offer in terms of advantages for rendering via still pictures or using 60 frames per second (FPS) videos. We present two exceptions. Figure 1 attempts to illustrate the integration effect in the human visual system. Four images, each rendered with *one* sample per pixel (SPP) from the standard $4\times$ MSAA pattern, are shown after each other
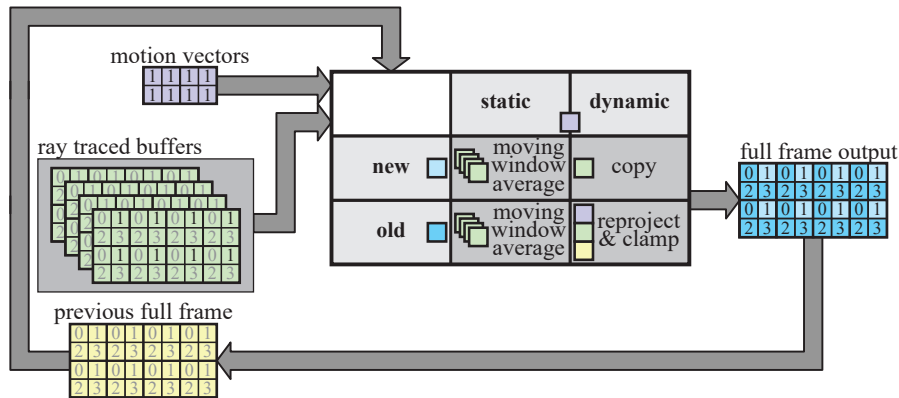
Figure 3: *Our frame generation pipeline. We first render a subframe of lower resolution than the final output, while simultaneously retrieving motion vectors for the corresponding pixels. This subframe together with the previously rendered subframes are then used to composite the final frame. For each pixel in the final frame, we determine if its content is static or dynamic, and whether or not it corresponds to one of the pixels that were just ray traced (new/old). Based on that, as indicated by the center box, different composition methods are used, possibly using the previous frame as input. Black numbers in buffers indicate values that are updated during this frame, while gray numbers indicate information retained from previous frames. Finally, the current frame is copied to the previous frame's buffer.*

Figure 2: *Motion blur reduction with 240 Hz, at 960 pixel per second motion. Images courtesy of Blur Busters [Rej13].*

in succession. The higher the display rate, the more we see the combined, antialised effect of all four images. Even though our images were taken with a camera, humans experience a similar effect. The other exception, shown in Figure 2, illustrates that faster refresh rates reduce the perceived motion blur, which is well known [DER*10, Swe07]. This is an effect that can be captured using a high-speed camera and seen on websites, such as www.testufo.com.

The best way to convey our results is to look at image sequences on a 240 Hz display. Simply averaging four subsequent images of a scene with motion will create a blurred image, which is the opposite effect that a high frame rate display can generate (Figure 2). Therefore, we encourage the reader to go to our website www.240hz.org, which is custom made for this work, and follow the instructions to experience our findings. It is not possible to see the effects that we report on a monitor with less than 240 Hz. In Section 2 on that website, we show four different cases where a 240 Hz display delivers a substantially better visual experience. Section 3 motivates and illustrates our algorithm details, while Section 4 shows a number of scenes rendered using our approach.

3. Our Approach

Temporally dense sampling, i.e., increasing the frame rate, can be afforded if we reduce the spatial sampling rate. Without loss of generality, we restrict ourselves to tracing *one* sample per 2×2 pixels per frame. It is possible to trace n samples in each $w \times h$ tile instead, but that is left for future work. In the figure to the right, one could imagine tracing one sample per red (0) pixel first, followed by one sample per green (1) pixel in the subsequent frame, and so on, resulting in a complete frame, i.e., one sample per pixel, after four iterations. Note that the pixel ordering does not necessarily need to follow the one in the figure. Each tile could also have its own ordering—something we explore in later sections. The collection of all pixels with the same ordering number is called a *subframe*. Each subframe itself does not contain sufficient information to reconstruct a full resolution frame of desired quality. Therefore, we complement the current subframe with pixels from prior subframes, as well as the previous full resolution frame, when we composite a full resolution frame. Superficially, this is a trivial extension to checkerboard rendering [EM16], but it will be shown that pixel ordering should not be done naïvely, and that is key to the success of this approach.

This section describes our frame generation pipeline, which is outlined in Figure 3. Ray tracing of subframes and motion vector generation are presented in Section 3.1, while Section 3.2 explains how the subframes and motion vectors are used to generate the full resolution output frame.

3.1. Ray Tracing and Motion Vector Generation

By sparsely sampling the scene, the number of traced rays and shaded hit points is reduced, thus enabling higher frame rates. For a target frame with resolution $W \times H$, we dispatch only $\frac{W}{2} \times \frac{H}{2}$ rays per subframe. Next, we describe how subframes are ray traced and motion vectors generated.
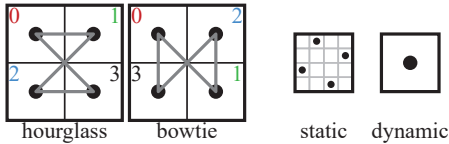
Figure 4: *Left: pixel orderings, called the* hourglass *and* bowtie *patterns, within a 2 × 2 pixel tile. Right: per-pixel sampling patterns in pixels with static content and dynamic content.*
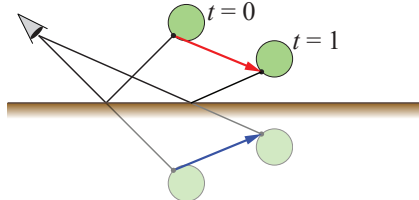

Figure 5: *A green circle moves to the right and a point on the circle is seen in the reflective plane. Instead of computing the motion vector using the primary ray's hit points, we use the reflected vector (blue), which significantly aids reprojection-based algorithms.*

3.1.1. Tracing Subframes

Our algorithm does not dictate a specific method for shading samples. We choose to use Whitted ray tracing [Whi80], as it enables us to include accurate reflections and shadows. Our implementation adopts a recent formulation of ray differentials [AMNA*19] for texture filtering.

The pixel orderings and sampling patterns that we use are shown in Figure 4. Each 2 × 2 tile chooses either an *hourglass* or a *bowtie* pattern, which are different pixel orderings within the tile. This choice is based on the motion in the tile. There are 4! = 24 possible pixel orderings in total, which can be reduced to three basic shapes, namely, a square, a bowtie, and an hourglass. The square did not exhibit any additional benefit over the other two, so we did not use it. Choosing pixel ordering correctly in a tile has a significant impact on the perceived quality of the final frame sequence. How the choice is made, and the implications it has, is further described in Section 3.2.3. As it is based on motion, the selection of per-tile pixel ordering is determined in the final frame generation and passed to the subsequent ray tracing pass. We only make this decision once every four frames in order to always complete a full frame before updating the per-tile patterns. The per-pixel sampling patterns to the right in Figure 4 are the ones we use when the scene is static versus dynamic. This will be discussed in detail in Section 3.2.

3.1.2. Motion Vectors for Reflections

Temporal antialiasing (TAA) [Kar14, PSK*16] reprojects and reuses color samples from the previous frame by using motion vectors (MVs) to track the movement of features anchored to the surface of three-dimensional objects. For mirrors, we compute the reflected motion vector as shown in Figure 5, which can be used for any TAA-based algorithm. While the proposed solution is simple and limited, in the sense that it assumes that the mirror has not moved since the previous frame, it does improve the reprojection results significantly in many cases. There are other methods that
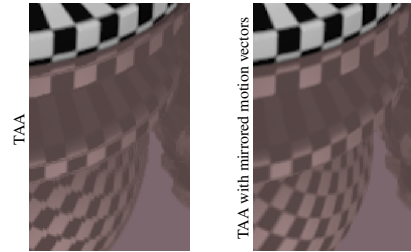

Figure 6: *These images show post-TAA reflections for non-reflected (left) and reflected (right) motion vectors. The top part (black and white) of the images are equal while the reflection is considerably better with reflected motion vectors.*

aim at solving this problem, but a thorough comparison is out of scope of this work. Figure 6, and our website, show the impact of reflected motion vectors with the standard TAA [Kar14] algorithm. We have limited the number of mirrors we can bounce off before computing motion vectors to one. This number could be increased by using, e.g., a chain of Householder matrices, although at a higher cost.

3.2. Final Frame Generation

The final full resolution frame is composited using the most recent subframe and motion vectors, as well as information from prior subframes and the previous full resolution frame. The box at the center of Figure 3 illustrates how the color of each pixel depends on whether its content is deemed to be in motion (static/dynamic), and whether it is part of the most recently rendered subframe (new/old).

3.2.1. Dynamic Pixels

A pixel's content is considered dynamic if the longest screen space motion vector, $\mathbf{m}$, within a 3 × 3 pixel region is nonzero [PSK*16]. The size of this footprint improves detection of fast motion in areas of the image with multiple moving geometries.

While *new* dynamic samples are directly stored in the framebuffer, as shown in Figure 3, *old* and *dynamic* color samples are reprojected and color clamped, in a similar fashion to standard TAA [Kar14]. In particular, the longest motion vector, $\mathbf{m}$, is used to find where the pixel was in the previous frame, and we resample its value from there via a bilinear filter. We then use variance sampling [PSK*16] to compute an axis-aligned bounding box in color space to which we clamp the reprojected pixel.

It is worth noting that extending TAA's reprojection and clamping to our setting is not straightforward. One can, for example, use all immediate neighbors in the variance sampling, which means that old pixel information would be used. Another possibility would be to do variance sampling with samples from a 3 × 3 area in the most recent subframe, but this also reduces the quality of the final frame. These two alternatives are illustrated to the left in Figure 7. Another solution is to construct the box out of the current pixel's immediate neighbors included in the most recent subframe, which is the method we use. Illustrations of this are shown to the right in Figure 7. Note that, in the examples shown in this figure, the most recent samples (marked green in the figure) are positioned regularly
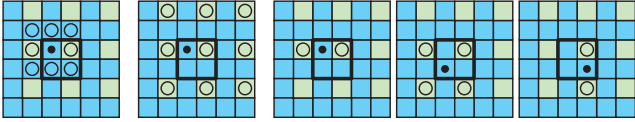
Figure 7: *These figures show variants of neighborhood positions (marked ○) used during variance sampling of a pixel (marked ●). Green pixels are retrieved from the most recent subframe. The left figure shows the situation where all immediate neighbors are used, while the next shows the case where samples are taken from a 3×3 area in the most recent subframe. We use the three patterns to the right, i.e., the ones using the most recent and immediate neighbors.*

Figure 8: *From left to right, for a horizontally moving feature: full frame rendering (spatially dense), rendering with* hourglass*, hourglass with reprojection and clamp, rendering with* bowtie*, and* bowtie *with reprojection and clamp. Note how the rightmost image is most similar to the ground truth image to the very left.*

on the frame for clarity. This is not necessarily the case when we allow for different per-tile pixel orderings. The principle of using only the most recent and immediate neighbors, however, remains. A comparison between the results generated with the three methods is shown on our website.

Furthermore, note that we do not temporally accumulate color samples using an exponential moving average as in TAA, since this tends to generate ghosting effects in pixels with dynamic content, which are disturbing even at 240 FPS. To reduce the aliasing of pixels whose content is static, we instead use a moving window averaging approach, described next.

3.2.2. Static Pixels

The color of a *static* pixel is computed by averaging up to four samples, depending on how long it has been static, effectively resulting in $4\times$ supersampling antialiasing (SSAA). We use a fixed jitter pattern with four samples, as seen in the static pattern to the right in Figure 4. Similar to the tile patterns, the jitter remains the same for four frames, i.e., until each pixel in a tile has been ray traced.

Whenever the camera viewpoint changes or we detect temporal change, we reset averaging in the pixel and instead use the reprojection method discussed in the previous section. As long as averaging is turned off, so is the jitter of our primary rays, and we will then only sample pixel centers to avoid noisy edges in the image. Here, the flexibility of ray tracing enables us to change sampling pattern on a per-pixel basis. As mentioned above, we defer motion vector computations until the next hit point when we hit mirrors. If not, one would incorrectly average samples in, e.g., static mirrors reflecting a moving object.

Temporal change in a pixel includes altered lighting, shadows, indirect illumination, camera motion, transparency, etc., and should possibly inhibit accumulation. For example, one can use the flow methods of Leimkühler et al. [LSR17] to handle many situations. We see this as an open problem, orthogonal to the main task of our research. In our current implementation, we simply use motion vectors as indicators of temporal change.

3.2.3. Adaptive Pixel Ordering Selection

As briefly mentioned in Section 3.1.1, we have discovered that the choice of each tile's pixel ordering, *hourglass* or *bowtie* (Figure 4), is critical to how we perceive the resulting image sequences. In
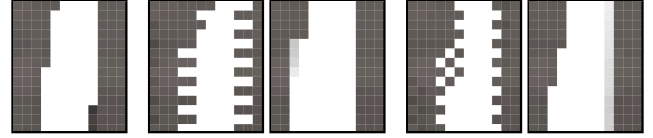
particular, choosing the correct ordering will greatly reduce spatiotemporal aliasing, such as flickering or crawling pixels. This is exemplified in Figure 8, where a horizontally moving feature is rendered spatially sparse and temporally dense, with the *hourglass* and *bowtie* patterns, next to their reprojected and clamped counterparts. In this case, the two pixel long extrusions of the *hourglass* pattern reprojects to an edge that is aligned at the granularity of a tile, which is undesirable. The *bowtie* pattern exhibits a more favorable and perceptually pleasing result at this particular velocity. We can see that there is a strong dependency between the best pattern selection, and on motion direction and speed.

The goal of this section is to find a function, g, which determines each tile's pixel ordering type for the next four frames using some per-tile input. While we believe g to be a function of the tile's motion, $\mathbf{m} = (m_x, m_y)$, and pixel neighborhood contents, $\mathbf{P}$, such as edges, we limit our version of the function, denoted $\hat{g}$, to only consider the tile's motion, such that

$$\hat{g}(\mathbf{m}) \approx g(\mathbf{m}, \mathbf{P}) \in \{hourglass, bowtie\}. \quad (1)$$

We choose the tile's motion to be the same motion as was used to determine whether the tile's contents were static or dynamic, i.e., the longest motion vector in a 3×3 area around the tile's location in the MV buffer.

Our function, $\hat{g}$, first computes the length, ℓ, of the motion vector, and proceeds by removing its integer part, yielding

$$\ell_f = \ell - \lfloor \ell \rfloor, \quad \ell = \|\mathbf{m}\|. \quad (2)$$

We remove the integer part because basing the decision on the absolute motion itself was found to be insufficient. Next, our function determines the motion's angle, α, relative to the x-axis, as

$$\alpha = \mathtt{atan2}(m_y, m_x). \quad (3)$$

With ℓ_f and α introduced, we will now change the signature of $\hat{g}(\mathbf{m})$ to $\hat{g}(\ell_f, \alpha)$, as those are the two parameters that influence the decision. For strictly horizontal motion, i.e., $\alpha \in \{0, \pi\}$, we empirically found that

$$\hat{g}(\ell_f, \alpha) = \begin{cases} hourglass & \text{if } \ell_f \in (0.25, 0.75), \\ bowtie & \text{otherwise,} \end{cases} \quad (4)$$

yielded the desired results. This method effectively segments the scene into different regions, and to base the pixel ordering decision on those is favorable. An example can be seen in Figure 9 and on our website. When the camera is moving and the *hourglass* pattern is used in all tiles, the beer taps on the bar flicker, but not the glasses and bottles on the shelves. If the *bowtie* pattern was used

Figure 9: *An example of the scene segmentation during strictly horizontal motion, where a yellow pixel indicates that its tile is rendered with the* hourglass *pattern, while a blue pixel indicates that its tile is rendered with the* bowtie *pattern.*

instead, the situation would be reversed. As we make the decision based on the fractional part of the motion, we are able to correctly choose per-tile orderings in both regions, as well as for the chairs in the foreground. Similar to the case when we had strictly horizontal motion, we found that with strictly vertical motion, i.e., when $\alpha \in \{-\frac{\pi}{2}, \frac{\pi}{2}\}$, we should instead have

$$\hat{g}(\ell_f, \alpha) = \begin{cases} hourglass & \text{if } \ell_f \in [0, 0.25] \cup [0.75, 1], \\ bowtie & \text{otherwise.} \end{cases} \quad (5)$$

Given the results above, the question arises how to choose pattern when the motion is neither strictly horizontal nor strictly vertical. One option is to base the selection on only the dominant motion direction. That selection method, i.e., the function graph corresponding to $\hat{g}(\ell_f, \alpha)$, is visualized to the left in Figure 10. For all methods depicted there, we only show the function graph in the interval $\alpha \in [0, \frac{\pi}{2}]$, and use symmetry to cover the remainder of possible motion directions. We found that this first approach, i.e., to base the choice of solution on the dominant motion direction, did not yield satisfactory results. Instead, we tried using smoother transitions between the two solutions over $\alpha \in [0, \frac{\pi}{2}]$.

The two illustrations in the middle of Figure 10 both show continuous transitions between the two solutions at $\alpha = 0$ (Equation 4) and $\alpha = \frac{\pi}{2}$ (Equation 5). There are many other possible, continuous transitions, but these two are the simplest. For $\alpha \in [0, \frac{\pi}{4} - \gamma]$, where $\gamma = \frac{\pi}{40}$, the middle-left transition was preferable, while the middle-right transition provided the best results for $\alpha \in [\frac{\pi}{4} + \gamma, \frac{\pi}{2}]$. Between those intervals, i.e., for $\alpha \in (\frac{\pi}{4} - \gamma, \frac{\pi}{4} + \gamma)$, neither transition was satisfactory—both resulted in significant aliasing in different parts of the scene. To combat this, we remove some of the structure in the image by randomizing the pattern choice in that interval. This reduces the flickering artifacts and crawlies. The constant $\gamma = \frac{\pi}{40}$ was found empirically. Putting the above together yields the pixel ordering decision function $\hat{g}(\ell_f, \alpha)$ that we use. The corresponding function graph is shown in the right figure in Figure 10.

On our website (Section 3), we include comparisons of the three rightmost decision functions in Figure 10 for three different camera animations. The first camera has a motion direction $0 < \alpha \leq \frac{\pi}{4} - \gamma$, the second has a motion direction $\frac{\pi}{4} - \gamma < \alpha < \gamma + \frac{\pi}{4}$, and the third has a motion direction $\frac{\pi}{4} + \gamma \leq \alpha < \frac{\pi}{2}$. With the first camera, we see that the middle-left function yields the most satisfactory results, while the results with the third camera suggests that the middle-right function provides the best experience. In the case of the second camera, the best results are those stemming from randomizing the pattern selection. These examples support our decision of using the rightmost pixel ordering decision function illustrated in

Figure 10. Section 3 on our website also demonstrates the adaptive pattern selection for arbitrary motion. Note that many of the methods above were found using much trial and error, and we have reported the best performing techniques we could produce. More research is needed in this field to optimize the pattern selection.

4. Results

Note that we have deliberately chosen *not* to include any images here, since our images need to be viewed in succession, and at 240 FPS, on a 240 Hz display to experience the intended quality. Therefore, we refer to `www.240hz.org`, where we provide cropped image sequences to allow for a replay speed of 240 FPS. All our images were rendered at 1920×1080 pixels on an Acer Predator XB272 monitor, with one shadow ray per light source and up to three recursive reflections, using an NVIDIA RTX 2080 Ti. We have used three scenes, namely, BISTRO (interior), CERBERUS, and LIVINGROOM. BISTRO has 1M triangles and 15 static light sources, CERBERUS has 192k triangles, including one animated robot, as well as three static light sources, and LIVINGROOM has 581k triangles and three static light sources. Furthermore, in all our visualizations, we have activated vertical synchronization (v-sync). Otherwise, if the rendering speed exceeds the display refresh rate, the perceptual integration of frames is disturbed and the perceived quality therefore reduced.

In Section 4 on our website, we show results for the scenes above. For all results, we show vanilla rendering, with and without TAA, at *n* FPS, and our spatially sparse temporally dense rendering at $4 \times n$ FPS. The rationale for this is that our method renders a frame about $3.5\times$ faster than vanilla rendering with TAA. Note that, while we reduce the ray tracing cost to roughly a quarter, the per-frame cost of our filtering is similar to that of TAA. Furthermore, when attempting to render at 240 FPS, possible CPU bottlenecks become more prominent. In such cases, we cannot expect a $3.5\times$ speedup with our method. However, there are certainly more performance enhancements to be made in our implementation, so these numbers could potentially be improved. Other nontrivial CPU and GPU work associated with, e.g., the workings of a game engine, is outside the scope of our work.

The BISTRO scene has, perhaps, the most spectacular results. At 60 FPS, our method looks almost broken, but at 240 FPS, our rendered images are integrated into a perceptually satisfactory experience. We urge the reader to look at the beer taps and the bar counter. This scene is ray traced at about 310 FPS. In the LIVINGROOM scene (290 FPS), we see a large reduction in aliasing with our method and TAA, compared to the vanilla rendering. This can be seen, for example, on the table. As the robot in CERBERUS (260 FPS) runs, we see some flickering in the highlights for the vanilla rendering. This is reduced both with TAA and with our method, though with TAA, the highlights tend to disappear completely. In CERBERUS, the reflected motion vectors (enabled for our method) gives a positive effect in the form of a clearer reflection. This scene also demonstrates the blur that arises without any special handling of dynamic shadows cast on static objects.

As a medium failure case, we have found that rapidly moving, thin features are often hard to render well. To demonstrate this, we
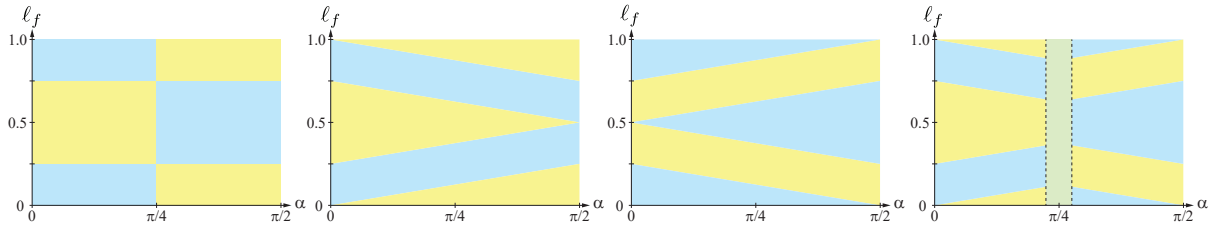
Figure 10: *Four different alternatives for the per-tile pixel ordering decision function, $\hat{g}(\ell_f, \alpha)$, based on the fractional part, ℓ_f, of the motion length, and the motion direction, α. Yellow color indicates that the* hourglass *pattern should be chosen, blue color indicates the* bowtie *pattern, and green indicates that the choice between the two is random. We use the rightmost alternative.*

have rendered HAIRBALL (290 FPS). The results can be found on our website. TAA generates substantially more blur than both other methods and thin geometry may sometimes even disappear. One case where aliasing is not reduced at all is shown in the *bowtie* versus *hourglass* comparison (Section 3 on our website). For the closest chair at the bottom, the top curved edge has retained aliasing. This is, however, a problem for all methods we compared. Another problem that is sometimes visible is what we call *advancing fronts*. When an edge of an object belongs to a region classified as, say, *bowtie* and the background of that edge belongs to *hourglass*, a tile with an edge may be classified as *hourglass* and when the classification switches to *bowtie*, it is too late. This may manifest itself as aliasing on the front of a moving edge.

In a small user study with 35 subjects (33 men, two women), and an age span from 27 to 73 years, we showed three renderings next to each other. A high quality (256 SPP) rendering of BISTRO at 240 FPS was shown in the middle. For each subject, we randomly placed a rendering with 1 SPP at 60 FPS on either the left or right side, and our technique at 240 FPS, on the other side. We excluded TAA in the user study since we wanted to focus our evaluation on the perceptual difference between 60 Hz and 240 Hz, and not on the amount of blur. When asked to identify the rendering that best resembled the ground truth, 94% selected our technique, which indicates a strong preference toward our method.

Another remarkable result is shown in the last link on our website. There, we compare our method, at 240 FPS, to 60 and 240 FPS vanilla rendering. In our opinion, our method is very close to vanilla rendering at 240 FPS, which acts as a *ground truth* image in this case. Recall that our method traces only 25% of the full frame's pixels each frame.

A general comment is that our method, due to its spatially sparse rendering, cannot produce fully temporally coherent image sequences. However, the images are produced at a much higher pace, highlights tend to stay in the images, and textures remain sharp. TAA runs at a lower rate, provides blurry texturing and edges, and even ghosting. What is best is subjective, but if higher frame rates are needed in order to compete in esports, as mentioned in the introduction, our method would likely be a better choice. The integration of frames at 240 Hz is exploited by our algorithm, as we have shown both for edge antialiasing and for improved clamping. We believe our work shows promise, with many avenues for future work, including improved pattern selection, better sampling patterns, code optimization, theoretical explanations, and generalized motion vector computations.

References

[AMNA*19] AKENINE-MÖLLER T., NILSSON J., ANDERSSON M., BARRÉ-BRISEBOIS C., TOTH R., KARRAS T.: Texture Level of Detail Strategies for Real-Time Ray Tracing. In *Ray Tracing Gems*, Haines E., Akenine-Möller T., (Eds.). Apress, 2019, ch. 20. 3

[BFMZ94] BISHOP G., FUCHS H., MCMILLAN L., ZAGIER E. J. S.: Frameless Rendering: Double Buffering Considered Harmful. In *Proceedings of SIGGRAPH* (1994), pp. 175–176. 1

[DER*10] DIDYK P., EISEMANN E., RITSCHEL T., MYSZKOWSKI K., SEIDEL H.-P.: Perceptually-motivated Real-time Temporal Upsampling of 3D Content for High-refresh-rate Displays. *Computer Graphics Forum 29*, 2 (2010), 713–722. 2

[DWWL05] DAYAL A., WOOLLEY C., WATSON B., LUEBKE D.: Adaptive Frameless Rendering. In *Eurographics Symposium on Rendering* (2005), pp. 265–275. 1

[EM16] EL MANSOURI J.: Rendering Tom Clancy's Rainbow Six | Siege. Game Developers Conference, 2016. 1, 2

[HA90] HAEBERLI P., AKELEY K.: The Accumulation Buffer: Hardware Support for High-quality Rendering. In *Proceedings of SIGGRAPH* (1990), pp. 309–318. 1

[HEMS10] HERZOG R., EISEMANN E., MYSZKOWSKI K., SEIDEL H.-P.: Spatio-Temporal upsampling on the GPU. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (February 2010). 1

[Kar14] KARIS B.: High Quality Temporal Supersampling. Advances in Real-Time Rendering in Games, SIGGRAPH Courses, 2014. 1, 3

[KH01] KELLER A., HEIDRICH W.: Interleaved Sampling. In *Eurographics Workshop on Rendering Techniques* (2001), pp. 269–276. 1

[KSM*19] KIM J., SPJUT J., MCGUIRE M., MAJERCIK A., BOUDAOUD B., ALBERT R., LUEBKE D.: Esports Arms Race: Latency and Refresh Rate for Competitive Gaming Tasks. In *Vision Science Society* (2019). 1

[LSR17] LEIMKÜHLER T., SEIDEL H.-P., RITSCHEL T.: Minimal Warping: Planning Incremental Novel-view Synthesis. *Computer Graphics Forum 36*, 4 (2017), 1–14. 4

[PSK*16] PATNEY A., SALVI M., KIM J., KAPLANYAN A., WYMAN C., BENTY N., LUEBKE D., LEFOHN A.: Towards Foveated Rendering for Gaze-tracked Virtual Reality. *ACM Transactions on Graphics 35*, 6 (2016), 179:1–179:12. URL: http://doi.acm.org/10.1145/2980179.2980246, doi:10.1145/2980179.2980246. 1, 3

[Rej13] REJHON M.: PHOTOS: 60Hz vs 120Hz vs ULMB, May 2013. www.blurbusters.com/faq/60vs120vslb/. 2

[Swe07] SWEET B. T.: The Impact of Motion-Induced Blur on Out-the-Window Visual System Performance. In *IMAGE* (2007). 2

[Whi80] WHITTED T.: An Improved Illumination Model for Shaded Display. *Commununications of the ACM 23*, 6 (1980), 343–349. 3