Pacific Graphics 2016
E. Grinspun, B. Bickel, and Y. Dobashi
(Guest Editors)

Volume 35 (2016), Number 7

Pixel History Linear Models for Real-Time Temporal Filtering

Jose A. Iglesias-Guitian[1], Bochang Moon[1], Charalampos Koniaris[1], Eric Smolikowski[2] and Kenny Mitchell[1]

[1]Disney Research (The Walt Disney Company)
[2]Walt Disney Imagineering (The Walt Disney Company)

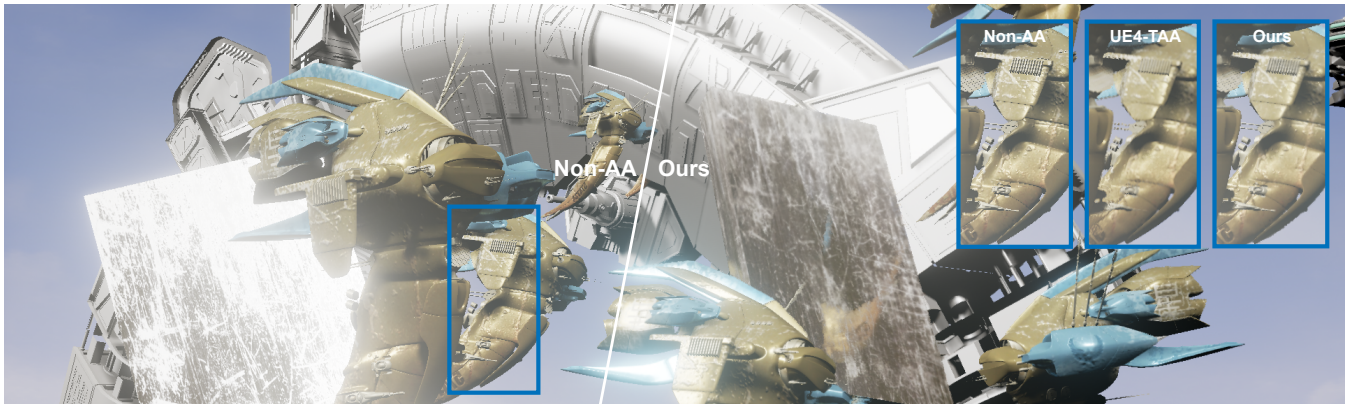

Figure 1: *Our filtering results obtained for frame #*320 *of the* SPACELAND *sequence. This scene showcases a non-linear camera motion, an animated directional light, and various shading effects including shadows and reflections. This scene has been rendered with Unreal Engine 4 [Epi] using* 1 *sample per pixel (Non-AA). The presence of fine geometric details and detailed textures produces lots of temporal aliasing and flickering artifacts. Our filtering method effectively reduces flickering without creating ghosting artifacts (please watch the supplementary video). Moreover, our approach produces less visual overblur (see insets) than the current state-of-the-art solutions for real-time temporal antialiasing, e.g.,* 1.33 *dB better PSNR on average than Unreal Engine temporal filter (UE4-TAA).*

Abstract

We propose a new real-time temporal filtering and antialiasing (AA) method for rasterization graphics pipelines. Our method is based on Pixel History Linear Models (PHLM), a new concept for modeling the history of pixel shading values over time using linear models. Based on PHLM, our method can predict per-pixel variations of the shading function between consecutive frames. This combines temporal reprojection with per-pixel shading predictions in order to provide temporally coherent shading, even in the presence of very noisy input images. Our method can address both spatial and temporal aliasing problems under a unique filtering framework that minimizes filtering error through a recursive least squares algorithm. We demonstrate our method working with a commercial deferred shading engine for rasterization and with our own OpenGL deferred shading renderer. We have implemented our method in GPU and it has shown significant reduction of temporal flicker in very challenging scenarios including foliage rendering, complex non-linear camera motions, dynamic lighting, reflections, shadows and fine geometric details. Our approach, based on PHLM, avoids the creation of visible ghosting artifacts and it reduces the filtering overblur characteristic of temporal deflickering methods. At the same time, the results are comparable to state-of-the-art real-time filters in terms of temporal coherence.

Categories and Subject Descriptors (according to ACM CCS): I.4.3 [Computer Graphics]: Enhancement—Filtering
I.3.3 [Computer Graphics]: Picture/Image Generation—Antialiasing;

1. Introduction

The demand for higher quality content generated in real-time continuously pushes the rasterization graphics hardware to the limit, systematically uncovering fundamental problems. The increasing sophistication and complexity of shading computations create a compromise between available computational resources and achievable spatial and temporal resolutions. As a consequence, spatial and temporal resolutions are often the first candidates to bow to the performance constraints [SYM*12]. Given the limited amount of samples per pixel (spp) that can be used in real-time rendering, undersampling problems easily arise when rendering complex scenarios. For instance, when rendering dense foliage scenes, many geometric details of different leaves, each having a different shading value, may contribute to the same final pixel color. This situation easily derives into very noisy changes of pixel values over time, creating a distracting visual effect, commonly known as temporal aliasing or flicker.

The two main sources of aliasing in rasterization are spatial (e.g. visibility and shading) and temporal (e.g. geometric and specular aliasing). Supersampling is a widely adopted solution to alleviate these problems based on computing higher resolution images and downsampling them to obtain the final filtered result. In rasterization, supersampling usually refers to the spatial and temporal dimensions. On the spatial side, multisampling [Ake93] can be combined with decoupled shading architectures that separate visibility samples from shading in order to minimize rendering costs [RKLC*11]. However, increasing spatial resolution is often not enough to solve temporal flickering problems. In those scenarios, temporal coherence should be exploited to reduce flickering by reusing samples across multiple frames [SYM*12]. Amortizing samples across time, poses the problem of representing history of a pixel value over time. In order to avoid excessive memory requirements, the idea of using exponential history buffers has been widely explored in the literature [HEMS10], even with subpixel accuracy [YNS*09]. These methods typically address temporal flickering by blending the pixel history value with the most updated value produced at the current frame. This blending process is usually driven by heuristic rules and parameter thresholds, often based on comparing pixels depth and motion information for source and target pixel samples. The heuristic nature of this blending process exposes these methods to fundamental problems when dealing with object occlusion and disocclusion situations. For instance, depths may be quite similar, resulting in incorrect color blending of objects in different depth layers and may lead to smearing artifacts known as *ghosting* artifacts, because of the successive repetition of contours with decreasing intensity. Therefore, effectively reducing temporal flickering without creating such visual artifacts is still an open research challenge.

In this paper, we propose a new real-time temporal filtering and antialiasing method for rasterization graphics pipelines. Our method is based on Pixel History Linear Models (PHLM), a new online approach to track shading changes. The key difference between our and previous methods is that PHLM expresses pixel shading history as a linear model instead of using a single color value. By using linear models we can better approximate pixel shading changes over time. In fact, our method drastically reduces the creation of *ghosting* artifacts and reduces the characteristic overblur of current temporal deflickering filters (See Fig. 1). Our filtering framework is orthogonal in operation to other existing techniques (e.g. mipmaps and MSAA) and can be coupled with them to increase the quality of reconstructed pixel values. Specifically, our major technical contributions are:

- A new real-time temporal filtering and antialiasing method for rasterization graphics pipelines. We propose a general spatio-temporal filtering framework based on Recursive Least Squares (RLS) that tracks temporal shading changes based on linear models in an online manner. Our method predicts per-pixel variations of the shading function between consecutive frames and combines temporal reprojection with spatial predictions in order to reconstruct temporally coherent pixel shading.
- We propose Pixel History Linear Models (PHLM), a new idea to model pixel history of shading values over time using linear models. By tracking pixel shading values produced by the rasterization hardware in real-time, PHLM drastically reduce *ghosting* artifacts and significantly alleviate overblur produced by previous real-time temporal deflickering methods. In addition, we propose a robust temporal feature based on lower frequency components as an input predictor for our linear models.

Our method can work even with an input rendered with a small number of samples or in the presence of very noisy shading values. Our approach addresses different sources of potential undersampling artifacts, i.e. spatial and temporal undersamplings. We have demonstrated our method working in real-time in conjunction with a deferred rendering rasterization pipeline, generating higher quality results than state-of-the-art techniques [Lot11, Kar14]. We evaluated temporal flickering based on visual comparisons and we also compared the techniques using peak-signal-to-noise-ratio (PSNR).

2. Related Work

Supersampling and multisampling algorithms. To prevent aliasing caused by undersampling, prefiltering before sampling is theoretically the preferred approach [Pra78]. In practice, texture aliasing in real-time is addressed with mipmaps [Wil83] which contain prefiltered versions of the original texture at different resolutions. Efficient prefiltering techniques can accelerate shading and are therefore a very promising research direction [BN12]. The main adopted solution in the rasterization world to address geometric aliasing is to utilize more samples. Multisampling has been coupled with image feature analysis resulting in many different techniques reviewed in more detail by Jimenez et al. [JGY*11]. These techniques such as Intel MLAA [Res09] and NVidia FXAA [Lot11] have been successfully applied to the antialiasing problem. While these approaches can reduce spatial aliasing artifacts, temporal aliasing still poses a challenge for real-time rendering.

A complementary approach to multisampling is to decouple shading samples from visibility samples [RKLC*11, LD12], reducing over-shading costs by shading only fragments after computing visibility. Wang et al. [WWHS15] further decoupled coverage from visibility in order to achieve better geometric antialiasing. Salvi and Vidimče [SV12] proposed a novel way for decoupling visibility from shading that significantly reduces the number of samples stored and shaded per pixel. A recent example of successful

prefiltering and decoupled shading is the aggregate antialiasing by Crassin et al. [CMFL15], where new hardware capabilities of modern GPUs were used to prefilter visibility samples using a clustering stage before shading. In the context of stochastic rendering, Clarberg et al. [CTM13] proposed a new sort-based architecture for decoupling sampling in deferred shading architectures, ensuring that only non-occluded samples are shaded.

Temporal reprojection techniques. Reprojecting sample information from previous frames is a classic approach, adopted to improve rendering efficiency and increase the number of samples in the temporal dimension. For a complete discussion on existing temporal coherence techniques we refer to the survey by Scherzer et al. [SYM*12]. As an early work, Walter et al. introduced the render cache [WDP99, WDG02], a system that displays images at a faster rate than the one that a renderer can generate complete frames, improving the visual feedback at the cost of producing approximate images during camera and object motions. Reverse reprojection caching by Nehab et al. [NSL*07] is a reprojection framework to reutilize shaded samples from previous frames in the current frame, and thus can avoid expensive shading computations. The amortized supersampling presented by Yang et al. [YNS*09] is a temporal reprojection caching framework, which controls the exponential smoothing factor that determines the decay of previously computed samples over time, in order to account for the temporal changes of shaded values. Spatio-temporal upsampling by Herzog et al. [HEMS10] utilized a regular sampling grid pattern in consecutive frames and feeds these partial results into a bilateral upsampling framework which locally adapts the filtering kernel to the image content. Bowles et al. proposed an iterative image warping algorithm for reprojection [BMS*12] that improves performance under different scenarios, like e.g. real-time stereo rendering for videogames. Yang et al. proposed bidirectional reprojection [YTS*11] that reprojects pixel colors from a future frame as well as previous frames, improving forward reprojection methods especially for disoccluded image regions. Andersson et al. [AHTAM14] focused on optimizing distributed rendering effects such as motion blur and defocus blur. Recently, Karis [Kar14] presented high-quality temporal supersampling, which turned into the main antialiasing solution of Unreal Engine [Epi]. Similarly NVidia presented TXAA and MFAA, based on similar concepts for reusing samples over time [FKS16]. To have a more detailed overview on recent hybrid reconstruction techniques we refer the reader to the recent publication by Drobot [Dro15].

In this paper, we propose a new real-time temporal filter based on a novel representation for pixel history. Our method exploits temporal coherence in pixel history and samples over time, relying on previous techniques for temporal reprojection. The method is output sensitive, meaning that its performance depends only on the number of output pixels. This is a key difference with respect to approaches decoupling visibility and shading, for which performance is directly affected by the number of samples per pixel. With respect to existing methods based on history buffers, we drastically reduce the creation of ghosting artifacts thanks to our online RLS error minimization algorithm. Next, we will provide a detailed description of our framework and of our novel representation for pixel history.

3. Pixel History Representation

Most effective techniques working on flicker reduction are based on the idea of reusing pixel shading and sample information over time. Hence, it becomes critical to have accurate and efficient representations of pixel samples and shading history values. To address this issue, exponential history buffers are often coupled with image-based reprojection techniques. Next, we will briefly refresh exponential history buffers and outline some of their major limitations and why we propose a different representation for pixel history.

Exponential history buffers. State-of-the-art techniques exploit temporal coherence among frames, by averaging coherent pixels in consecutive images over time. The shading values are commonly maintained in history buffers. Given an input color $y(t)$ at frame t a corresponding history value $h(t)$ is updated in an online manner:

$$h(t) = \alpha h(t-1) + (1-\alpha)y(t), \tag{1}$$

where α is an exponential convergence factor that controls the blending between history and current input colors. Finding an optimal value of α for each pixel is a well-known research challenge for interactive applications. To avoid overblurring artifacts caused by mixing incoherent pixel colors, history buffers are often coupled with image-based reprojection techniques [Dro15], as a means to account for the movement of objects or cameras.

A recurring issue when working with temporal reprojection is the side effect of mixing occluded and foreground object colors in the history buffer. This can produce ghosting artifacts, with objects and textures smearing. Most techniques use heuristic rules to deal with the problem, for example by comparing depths or velocities of the original and reprojected samples. However, these heuristic methods still produce visible artifacts that decrease the image quality. Some methods reduce the input variance by clamping the history color against the axis-aligned bounding-box for neighbor color information. This solution, named neighborhood clamping [Mal12, Kar14], demonstrates to be very efficient in removing flicker, however it is known to produce overblurred results. A common assumption of these techniques, that may decrease their sharpness, is the fact that exponential history buffers represent the history of a pixel using a constant value. This kind of flat representation may become insufficient when the input values are particularly noisy, as a constant approximation can easily fail to predict the future pixel shading trend, and furthermore, it can be easily corrupted by input noise.

Advanced pixel history representations. For very noisy input shading, as for example when rendering dense foliage scenes, it is preferable to have more versatile but somehow compressed representations for pixels history, which ultimately are better representations than simply using constant approximations. We got inspired by recent work in Monte Carlo ray tracing denoising, where noisy input images, rendered with a low number of samples, can be denoised *a posteriori* thanks to statistical models fitting the local distribution of the input samples [MIGYM15]. Even if we would like to use a RLS based temporal filtering, as proposed in the previous approach, we could not use the G-buffer data as features, because data could be noisy (e.g., 1 spp) and temporally unstable. Our idea, instead, will be based on having a more stable feature, e.g., using low-frequency color information (more details in Sec. 5). Hence,

we propose to build a statistical model of pixel history, and we have identified the following characteristics that a desirable model for interactive applications should fulfill:

- Accuracy. The history of a pixel should be based on shaded samples that affect exactly the projected area covered by that pixel.
- Up-to-date. The model should continuously assimilate new input samples and incorporate them as part of its history.
- Temporal Stability. The model should produce temporally more stable shading values than the provided input. The produced images should balance between bias and variance errors.

The candidate pixel history model is expected to be reasonably fast to evaluate and compact to store, as this would facilitate its integration with existing rendering systems. Current deflickering filters may create ghosts (as result of the lack of accuracy) in exchange for removing flicker. On the other hand, spatial AA filters do not create any ghost or noticeable overblur (good accuracy), but they are not temporally stable enough. After these observations, we propose to use statistical models for several reasons. First, they can exploit additional features that can help in reducing ghosting and flickering. Second, these models can predict future shading values by simply using input measurements and the current state of those models. Third, a simple model can accurately track shading values, approximating non-linear changes by piecewise linear approximations, often better than constant approximations provided by classic history buffers. Our framework, described next in Sec. 4, and our new method, described in Sec. 5, have been designed to fulfill the three aforementioned basic properties.

Table 1: *Notation used throughout this paper*

Symbol	Description
y	rasterization input image
$y(t)$	rasterization input image at frame t
y_i	shading value for pixel i
$y_i(t)$	shading value for pixel i at frame t
f	ground truth image
$f(t)$	ground truth image at frame t
$\hat{f}(t)$	filtered output image at frame t
$h(t)$	pixel history shading value at frame t
$\xi(t)$	L_2 error with respect to ground truth at frame t
$\hat{\xi}(t)$	estimate of L_2 error with the ground truth at frame t
$\hat{\mathbf{e}}_i(t)$	predicted error vector for pixel i at frame t
$\beta_i(t)$	linear model coefficients for pixel i at frame t
$\mathbf{P}_i(t)$	inverse covariance matrix for pixel i at frame t
$\mathbf{x}_i(t)$	input predictor variable (feature) for pixel i at frame t
Ω	image coordinates space
$v_i(t)$	velocity vector for pixel i at frame t
$\pi_i(t)$	reprojection coordinates for pixel i at frame t
ray_{ori}	*ray* origin
ray_{dir}	*ray* direction
ray_{hit}	first intersection point for *ray*

4. Temporal Filtering Framework

The final purpose of our filtering framework would be to restore the ground truth image f, an aliasing free image that would be generated by using an ideal rasterization system with an infinite number of samples per pixel (spp). The ground truth image can not be computed in practice, therefore it is approximated by another image, called reference, that is generated using a very high but finite number of spp (e.g. 16 or 128). Generating this image is generally not affordable in real-time but serves for comparison purposes. Our filtering framework will use instead an input image y, generated in real-time with a small number of spp, and susceptible of being affected by moderate or severe undersampling artifacts, i.e., variance and flickering.

We first define the basic notation used throughout this paper in Table 1. To represent the value of image functions for a specific pixel, we will use a subscript index. For instance, f_i and y_i would refer respectively to the ground truth and input values at pixel i. Our filtering method will use $y(t)$ as input to produce the filtered output $\hat{f}(t)$, which aims to approximate the ground truth image $f(t)$. When the sampling density used to generate $y(t)$ reveals insufficient, either in space or time, undersampling artifacts easily appear in our filtered image $\hat{f}(t)$ in the form of variance and flickering errors. These errors can be expressed as the L_2 error between the ground truth and the output filtered value as $\xi(t) = \left\| f(t) - \hat{f}(t) \right\|_2$.

Pixel history representation using linear models. We propose a real-time antialiasing filter using linear models. We extend previous work in image denoising [MIGYM15] with a real-time linear model estimation that considers pixel history to reduce temporal flicker. Next, we define pixel history linear models (PHLM), how they represent the history of a pixel, and how they are used to reduce flickering in the final shading values. We define the pixel history for a given input pixel y_i, as the linear model located at a pixel i in the current frame, and we approximate its shading value using the following linear regression:

$$\hat{f}_i = \beta_i^T \mathbf{x}_i, \qquad i = 1, \ldots, d \tag{2}$$

where β_i are the linear model coefficients, and $\mathbf{x}_i$ are the input predictor variables (i.e., some function of the input color y_i). The dimension $d \equiv |\beta|$, will correspond with the number of additional features selected as input for the linear models. In our framework, we store the two main components of these linear models: β_i, a vector of d coefficients, $\beta_i \in \mathbb{R}$; and the inverse covariance matrix for those coefficients $\mathbf{P}_i \in \mathbb{R}^{d \times d}$. With PHLM the history of a pixel at a given frame t will be represented by $\beta_i(t)$ and $\mathbf{P}_i(t)$.

Pixel shading prediction. At each pixel i in the image, we approximate temporal shading changes over time with a linear function. As we previously outlined in Sec. 3, linear models result interesting not only because they can compactly represent pixel history, but more importantly, they can predict the output shading values given the right input predictor variables. We define the input predictor variables $\mathbf{x}_i$ of our linear models as:

$$\mathbf{x}_i = [1, \mathbf{z}_i], \tag{3}$$

where 1 and $\mathbf{z}_i$ are, respectively, estimates of the intercept term and the linear model slopes (i.e., first derivatives). The predicted output shading for a pixel i at frame t, is obtained using the following simplified linear model equation:

$$f_i \approx \hat{f}_i = \beta_i^T \mathbf{x}_i. \tag{4}$$

To increase the quality of these predictions, additional features

$\mathbf{z}_i$ can be introduced as part of the predictor variables. In our case, given the constrained real-time scenario, we need to balance the number of additional features to keep small our model representation footprint. While our models can use any arbitrary feature, we can not use G-buffer features directly (e.g., depth or normals) as they may contain lots of flickering noise given the small number of samples. Therefore, we propose a different feature to remove temporal flicker, based on the low frequency components of the input images (See Sec. 5.1).

Pixel history update. An important aspect of working with history linear models is how to update the linear model representation using current signal measurements $y(t)$, i.e. the image resulting from rasterization rendering. When updating a linear model, one of the complex problems resides in how to weight the amount of input signal to use. This is somehow similar to the problem of setting the convergence factor α when working with exponential history buffers. When updating a linear model at a frame t, the key idea is to use the residual error of our predictions as the amount of update to do using the current image $y(t)$. Ideally, for a given pixel i, we would compute that error by using the values of the ground truth image f_i, and denote the error by $\xi_i(t) = \left\| f_i(t) - \hat{f}_i(t) \right\|_2$. Because the ground truth image is not available, we need to estimate this error as $\hat{\xi}_i(t)$, the error that our model is doing, prior to its update, when predicting the image at frame t. We estimate this error by using the pixel shading value $y_i(t)$ as the most updated measurement available in the place of the ground truth:

$$\hat{\xi}_i(t) = \left\| y_i(t) - \beta_i^T(t-1)\mathbf{x}_i(t) \right\|_2, \tag{5}$$

where $\mathbf{x}(t)$ can be any arbitrary feature computed after knowing $y(t)$. Notice that, even if $y(t)$ is already known at frame t, we are interested in knowing how much error there is with our non-updated model. The main idea behind it is that the new model coefficients $\beta(t)$ should minimize the predicted error. By using the predicted error vector, $\hat{\mathbf{e}}_i(t) = y_i(t) - \beta_i^T(t-1)\mathbf{x}_i(t)$, as residual in the update stage, we expect to reduce predicted errors in the future. More formally, what we propose is to use a recursive least squares (RLS) fitting algorithm [LS87, MIGYM15] to update the linear model coefficients as follows:

$$\begin{aligned} \mathbf{Q}_i(t) &= \frac{\mathbf{P}_i(t-1)\mathbf{x}_i(t)}{\lambda + \mathbf{x}_i^T(t)\mathbf{P}_i(t-1)\mathbf{x}_i(t)}, \\ \beta_i(t) &= \beta_i(t-1) + \mathbf{Q}_i(t)\hat{\mathbf{e}}_i(t), \end{aligned} \tag{6}$$

where λ is a weight to decrease the importance of previous frames values. This is typically fixed to a value near one, we used $\lambda = 0.998$ as widely used in real-time tracking systems [DLHSV08]. The inverse covariance matrix $\mathbf{P}$ can be incrementally updated by RLS using the following recursive equation:

$$\mathbf{P}_i(t) = \lambda^{-1}\left(\mathbf{P}_i(t-1) - \mathbf{Q}_i(t)\mathbf{x}_i^T(t)\mathbf{P}_i(t-1)\right), \tag{7}$$

which fortunately avoids any costly matrix inversion operation.

5. Our Temporal Filtering Method

In this section we introduce a novel filtering algorithm to increase the temporal coherence of image sequences rendered in real-time by rasterization graphics hardware. The key idea is to produce final filtered images from linear model predictions as introduced in Eq. 4. The same equation can be rewritten for a specific frame t as:

$$f_i(t) \approx \hat{f}_i(t) = \beta_i^T(t)\mathbf{x}_i(t). \tag{8}$$

First, we will explain how to provide a robust feature $\mathbf{x}_i$ for temporal denoising (See Sec. 5.1). Next, we will introduce our filtering algorithm based on reprojecting PHLM over time and their selective update according to a given input image at the current frame t (See Sec. 5.2). Our algorithm uses the updated state of PHLM to predict the final output image $\hat{f}(t)$.

5.1. Designing a Robust Temporal Feature

As we already introduced in Sec.4, we need to design a robust temporal feature $\mathbf{z}$ as part of our input predictor $\mathbf{x}$. The chosen feature will determine the behavior of our PHLM. There are two basic properties that a good feature for denoising should have: first, being free of noise, particularly the type of noise that we want to remove, i.e. temporal variance in our case; second, having a high correlation with the current input image. We propose to use an image feature resulting from the online accumulation, supported by an exponential buffer (Eq. 1), of the low frequency color components in the input images. This is an effective way of removing flicker from an image sequence at the cost of creating some bias error. Note that we will use this feature buffer just to drive the predictions of our linear models, and not to create the final output images by itself. Therefore, we can accept some degree of overblur in this feature buffer, as the main objective at this point is to just reduce the temporal variance introduced by the flickering effect. Later on, the online RLS will be in charge of correcting the introduced bias error by automatically adjusting PHLM predictions using the given input.

For the online temporal feature accumulation, we use a per-pixel convergence factor α (See Eq. 1) that takes its values from a heuristic rule based on reported sensitivity of the human visual system to luminance variations [Bar99]. Normally, our method operates in RGB colorspace for error estimation purposes. However, there is an exception when accumulating our temporal feature over time. To compute convergence factors based on contrast sensitivity, we transform RGB samples into luminance values, using the YCgCo colorspace. Perceptual heuristics based on luminance values have been successfully employed to reduce temporal flicker in previous antialiasing methods [Mal12, Kar14]. However, as this feature acts simply as guidance for our linear models, alternative accumulation strategies could be employed interchangeably. In practice, for a given pixel i, we compute $\alpha_i = \phi(v_i, l_i)/(l_i + \Delta_i(Y))$, where v_i is the velocity in screenspace and $\Delta_i(Y)$ is the magnitude of luminance variation in the neighborhood of pixel i. The variable l_i is the minimum distance between the luminance value for the pixel i in the accumulated feature image and the range of luminance described by $\Delta_i(Y)$. The function ϕ is defined as $\phi(v_i, l_i) = a l_i^2 + b v_i l_i^3$, where a and b are constant parameters. In all our tests, we have used $a = 0.125$ and $b = 0.25$, as they proved to work well for all our tested scenarios and were provided in the existing code by authors.

5.2. Main Filtering Algorithm

This section describes our temporal filtering algorithm for which we provide pseudocode in Alg. 1. The algorithm uses the defined temporal feature $\mathbf{z}$ and the velocity information v, required to obtain new pixel coordinates q, resulting from an image-based temporal reprojection method, $q = \pi(v)$. Our filtering framework is general and, therefore, it can use different temporal reprojection techniques [SYM*12].

Algorithm 1 Our Temporal Filter.

procedure TEMPORAL-FILTERING($y(t)$, $\mathbf{z}(t)$, $v(t)$)
  for all $i \in \Omega$ **do**
    $v_i \leftarrow$ Velocity at pixel i
    $\mathbf{z}_i \leftarrow$ Feature at pixel i
    $\mathbf{x}_i \leftarrow [1, \mathbf{z}_i]$
    $q \leftarrow \pi(v_i)$
    if $q \notin \Omega$ **then**
      $q \leftarrow i$
      Reset$(\beta_i, \mathbf{P}_i) \leftarrow y_i$
    else
      $\beta_i \leftarrow \beta_q; \mathbf{P}_i \leftarrow \mathbf{P}_q$
    $\hat{f}_i \leftarrow \beta_i^T \mathbf{x}_i$
    $Conv(\Omega_i) \leftarrow$ Convex Hull in $\Omega_i^{3\times3}$
    $ray_{ori} \leftarrow \hat{f}_i; ray_{dir} \leftarrow (y_i - \hat{f}_i)$
    $ray_{hit} \leftarrow ray.intersect(Conv(\Omega_i))$
    if $\hat{f}_i \notin Conv(\Omega_i)$ **then**
      $\hat{f}_i \leftarrow ray_{ori} + ray_{hit} \times ray_{dir}$
      Reset$(\beta_i, \mathbf{P}_i) \leftarrow \hat{f}_i$
      $\hat{f}_i \leftarrow \beta_i^T \mathbf{x}_i$
    $\hat{\mathbf{e}}_i \leftarrow (y_i - \hat{f}_i)$
    Update$(\beta_i, \mathbf{P}_i, \hat{\mathbf{e}}_i)$
    $\hat{f}_i \leftarrow \beta_i^T \mathbf{x}_i$

Pixel history reprojection. For a given position $i \in \Omega$, our algorithm will obtain the pixel coordinates q resulting from the reprojection method using the information in v_i, that is $q = \pi(v_i)$. As a special case, when our reprojection falls outside the image coordinates space $q \notin \Omega$, the linear model is reset to the input shading value $y_i(t)$. When $q \in \Omega$, we can reuse the linear model placed at pixel q in frame $t-1$, and reproject it to pixel i at frame t. In both cases our method will be ready to predict an initial guess of the output color. First, we need to update the input predictor variable by multiplying the current feature $\mathbf{x}_i = [1, \mathbf{z}_i]$ with the linear model coefficients: $\hat{f}_i(t) \leftarrow \beta_i^T(t)\mathbf{x}_i(t)$. Note that β_i contains initial values from $t-1$ until the model will be updated to the final state for t.

Pixel history correction. Next, our method performs a safety check for our initial prediction $\hat{f}_i(t)$. Intuitively, we would like to know if our initial prediction is close enough to the input values distributed around $y_i(t)$. In general, this can be known by testing whether $\hat{f}_i(t) \in Conv(\Omega_i)$, where $Conv(\Omega_i)$ is the convex hull in a given colorspace (e.g. RGB or YCgCo) of the input shading values around their local neighborhood. Computing an accurate convex hull for each pixel i can be extremely costly, and thus we use an axis aligned bounding box as its approximation in $\Omega_i^{3\times3}$.

At this stage, if our prediction $\hat{f}_i(t)$ is inside this colorspace hull $Conv(\Omega_i)$, our algorithm will consider our prediction valid. In this case, our method will simply estimate the predicted error vector $\hat{\mathbf{e}}_i$, and use it to update our linear model with values from frame $(t-1)$. Finally, the method predicts the final output using the new model coefficients $\beta_i^T(t)$.

On the other hand, when our initial prediction is not considered a good guess, i.e. $\hat{f}_i(t) \notin Conv(\Omega_i)$, our algorithm needs to refine this prediction. In this case, we propose to apply a neighborhood clamping [Mal12, FKS16], adapted to our linear models. For the linear model approach, replacing the history color would correspond to resetting a linear model with a specific color value. After this, we can proceed normally, estimating the new predicted error vector $\hat{\mathbf{e}}_i$, using that estimate to update our linear models, and predicting the final output value $\hat{f}_i(t)$. Challenging cases appear for zoom-in camera movements, where linear models covering subpixel details can be reprojected into wider pixel areas. Thanks to our error estimation, linear model predictions are immediately corrected using the current rasterized input y, reducing excessive blur introduced in the reprojection.

Special case with static scenes. In this scenario $v = \emptyset$, therefore our reprojection function $\pi(v)$ is an identity function which gives as result exactly the same pixel coordinates $\pi(v_i) = i$. Inspired by previous temporal reprojection techniques [YNS*09], our method uses a jittered sampling pattern and weighted moving average accumulation algorithm to converge towards infinite supersampling.

6. Implementation Details

We have implemented our temporal filtering method using CUDA and used one linear model per pixel for all the tested benchmarks. As temporal reprojection technique, we have implemented the reverse reprojection scheme as described in Nehab et al. [NSL*07].

Reset of a linear model to specific values. When the reprojection $\pi(v)$ falls out of the screen or when our linear model predictions $\hat{f}$ are far from the current input y, we need to reset the models to some specific values, either using the input signal y or some result of the neighborhood clamping (See Sec. 5.2). Resetting linear models at pixels in the screen border has a minor visual impact. Potential temporal artifacts reintroduced can be minimized, since our temporal feature stabilizes linear models using neighbor pixel information.

Initialization of the temporal filter. Our filter is initialized in the first frame using the input image information $y(t)$. For a linear model at pixel i, we set all coefficients multiplying the interceptor term to $\beta_i(t) = y_i(t)$ and the rest are cleared to zeroes. For the inverse covariance matrix $\mathbf{P}$, we set all elements to zeroes except the diagonal terms which are initialized with a relatively high value, e.g. 1000. Such value indicates the linear model to weight more the current input signal with respect to the pixel history. Our temporal feature $\mathbf{z}$ is initialized with the input image, $\mathbf{z}(0) = y(0)$.

Linear models storage. Our linear models input predictor $\mathbf{x}_i$ uses a constant intercept term in addition to the temporal feature $\mathbf{z}$ (See Eq. 3). Because in practice our temporal feature can be encoded to separate RGB channels, the final dimension d needed for the input predictor variables is 4. Specifically, one dimension is for the interceptor term and other three are for the temporal feature. The

Method	Spp	Time (ms)	Avg. PSNR (dB) OpenWorld	Avg. PSNR (dB) SpaceLand
Non-AA	1	-	23.45	25.98
FXAA	1	0.23	23.61	26.98
UE4-TAA	1	0.40	21.46	24.78
PHLM (ours)	1	7	23.29	26.11
Reference	16	>1000	-	-

Table 2: *Numerical comparison of different methods in addition to the visual results (*$1920 \times 1080p$*). Image quality is measured across the whole sequences as average PSNR (dB). Execution times in ms are reported for a commercial implementation (*FXAA *and* UE4-TAA*) and our GPU implementation of* PHLM.

number of coefficients per pixel for β is $3 \times d = 12$, and for the inverse covariance matrix $\mathbf{P}$, $d \times d = 16$ elements.

Our GPU implementation. We have implemented a GPU version of our general filtering framework using NVidia CUDA 7.5. After computing our temporal feature $\mathbf{z}$ in one kernel, our main kernel uses it to produce the final output $\hat{f}$ by using one thread for each pixel in the output buffer. Our linear models are stored in GPU texture memory, and thus we can exploit texture cache and hardware bilinear interpolation. We have exploited shared memory to facilitate access to pixel neighbors.

7. Results and Discussion

Flicker is a temporal artifact, perceived as an abrupt change in the reconstructed pixel shading for the same area in consecutive frames. It is observed subjectively based on the temporal contrast sensitivity function of each individual [Bar99]. Visual comparisons have shown so far, that the temporal flicker is more noticeable than spatial aliasing artifacts, and therefore, small amounts of overblurring are often perceived as more acceptable than the flickering effect [YBS08]. Perceptual metrics like SSIM or MS-SSIM are focused on a per-frame analysis, not covering changes over time which are essential for a flicker measurement. Some temporal metrics have been proposed, mostly in the context of block-based video encoding [AČMS10, YBS08] and some covering video with depth [ZY10], however there is not a well established metric for temporal coherence in rendering. Therefore, we have preferred to rely on visual comparisons to assess the level of flickering. We accompany this paper with a supplementary video to show differences between the compared methods. Moreover, we provide per-frame PSNR with respect to the reference as a quantitative measure, especially for cases where visual quality may appear similar. It is important to note that temporal filters always produce some degree of PSNR drop in exchange for removing flicker. We have used the same GPU implementation of our method for all experiments, running on a 3.40 GHz i7 Intel processor and an NVidia GTX 980 GPU with 4 GB of dedicated video memory.

7.1. Comparison to State-of-the-art Antialiasing Filters

We compare our method with state-of-the-art techniques for real-time antialiasing. We choose high quality temporal supersampling [Kar14], which is the current Unreal Engine 4 temporal antialising (UE4-TAA), because the code is publicly available and it implements similar concepts to state-of-the-art temporal filters (e.g. NVidia TXAA or MFAA). Also for a reference, we provide comparisons with a spatial filter FXAA, which is a GPU optimized implementation of morphological antialiasing known for producing similar results to 4xMSAA supersampling algorithms [Lot11]. We demonstrate our filtering results using UE4 rendering system [Epi]. We create two different scenes, OpenWorld and SpaceLand (See Fig. 2), both rendered at full HD resolution (1920×1080p). All our camera animations use non-linear motions, instead of simple panning or rotation. The corresponding reference images have been rendered using 16 spp at 4-times HD resolution (7680×4320p) and downsampled using a Lanczos resampling filter.

In the OpenWorld scene, we present a complex foliage scenario that creates challenging temporal aliasing artifacts. As it can be seen in the supplementary video, both UE4-TAA and our PHLM, provide better temporal coherence when compared to FXAA. This result serves also as an example of the low correlation of PSNR with the visual quality perceived over time (See Fig. 2). A detailed inspection of this sequence also reveals that UE4-TAA has often problems with disoccluded objects, creating the aforementioned ghosting artifacts. This mostly happens whenever an object with similar depth or color, e.g. leaves and branches, are disoccluded (See ghost contours in Fig. 2 OpenWorld UE4-TAA). The reader may also notice at close examination, that UE4-TAA produces a bit blurrier results when compared to our method. This observation is supported by higher PSNR values, 1.8 dB in average, obtained by our method with respect to UE4-TAA in this sequence (See Fig. 3).

The SpaceLand scene contains fine geometric details, animated lighting, reflections and shadows. We have observed a similar trend to OpenWorld in terms of temporal coherence (FXAA with high PSNR but visually strong flicker), ghosting artifacts (See Fig. 1 and Fig. 2 UE4-TAA) and PSNR drop for UE4-TAA (See Fig. 4). Our method does not need to maintain separate reflection or shadow rendering layers, instead, all the shading effects are integrated as part of the input shading to denoise. This general behavior makes possible to automatically denoise many different effects at once and it suits particularly well for effects potentially creating temporal artifacts (e.g. ambient occlusion). For dynamic shading over time, e.g. reflections, our method showed almost noise free results (See Fig. 6). We also report in Table 2 average PSNR across the whole sequences and execution times for the different methods. The average execution time for PHLM is near 7 ms, of which 2.4 ms are used to compute the temporal feature, 4.2 ms to run the main filtering algorithm and the remaining time is spent in thread synchronization and memory copies. In our supplementary video, we show equal time comparisons between FXAA with approximately 2 spp and our method with 1 spp. The results of FXAA still show more flickering than our results. Moreover, our method can be coupled with existing AA methods to produce better results.

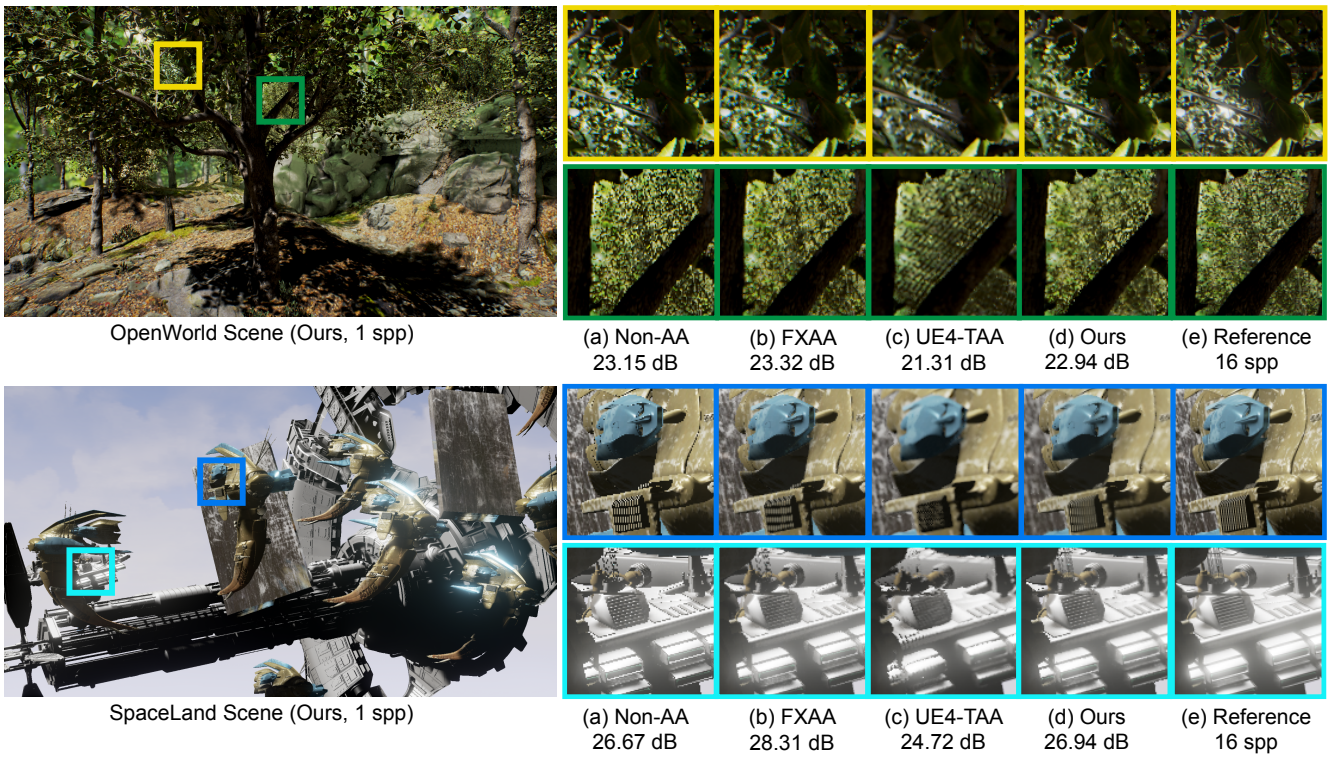

Figure 2: *Visual comparison of different antialiasing methods on the* OpenWorld *and* SpaceLand *scenes using UE4 rendering framework. Full frames #356 for* OpenWorld *and #487 for* SpaceLand *are shown together with insets zoomed on details. PSNR for the whole image is reported for all methods which take Non-AA as the input.*

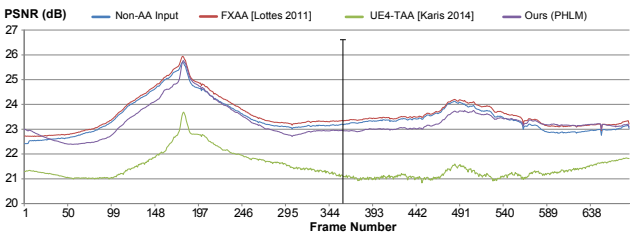

Figure 3: *Per-frame PSNR across the* OpenWorld *sequence for different antialiasing methods. Vertical line at the frame #356 denotes Fig. 2 (top row).*

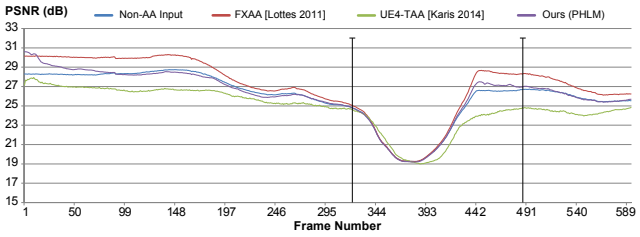

Figure 4: *Per-frame PSNR across the* SpaceLand *sequence for different antialiasing methods. Vertical lines at the frames #320, 487 denote Fig. 1 and Fig. 2 (bottom row) respectively.*

7.2. Qualitative Evaluation for Different Rendering Strategies

In this section, we study the behavior of our filtering method with different strategies for rendering the input image sequences. We have used a custom OpenGL deferred rendering framework with two different sequences, named SanMiguel-Balcony and SanMiguel-Chairs. These scenes present challenging geometric details on balcony and chairs, specular aliasing on tree leaves and plants, and high-frequency texture detail on floor and walls. Both sequences have been rendered at 1024×576p. Reference images have been computed using 128 spp. Our goal is to observe how the results are affected by increasing flicker in the input data. We demonstrate our method with conventional hardware texture filtering ($\times 16$ high-quality anisotropic mipmaps), as this is a widely adopted configuration for rasterization frameworks. Anisotropic mipmaps are very efficient in removing texture aliasing, however their use may produce sightly overblurred results (e.g. doorway stone texture in Fig. 5 SanMiguel-Balcony insets). While using a lower level of the mipmap texture may solve the issue, it may

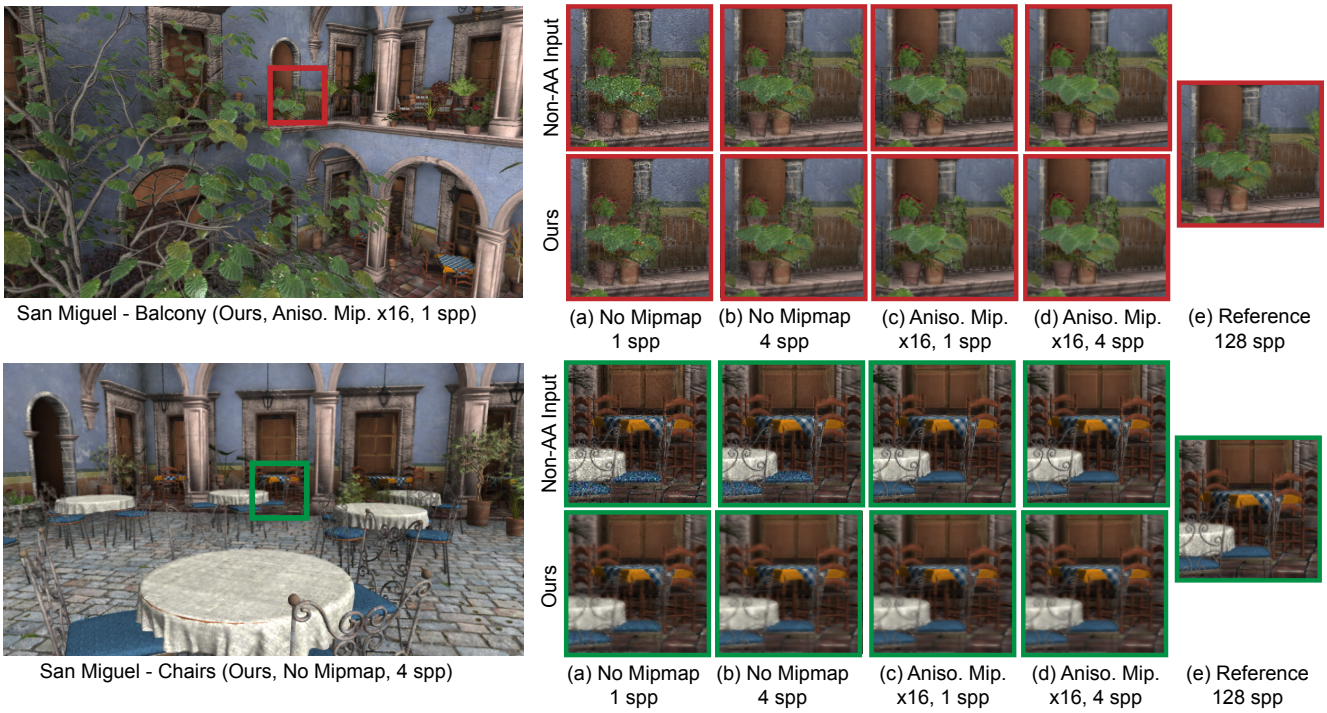

Figure 5: *Visual comparison for images generated with different rendering strategies on the* SANMIGUEL-BALCONY *and* SANMIGUEL-CHAIRS *scenes with an OpenGL rendering framework. We recommend the supplementary video to observe flicker reduction.*

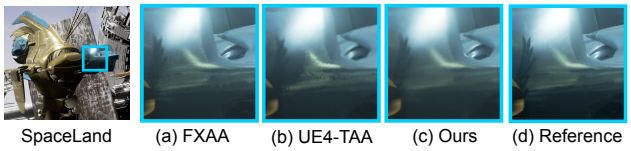

Figure 6: *Visual comparison for dynamic shading effects. We show results for a reflection effect on the spaceship surface.*

also reintroduce flicker as a side effect. We have also tested our method in the presence of severe undersampling by altering the texture sampling mode (e.g. no mipmaps). Moreover, we provide further comparisons for increasing numbers of samples (1 and 4 spp) for all cases. Our experiments show that PHLM consistently improves temporal coherence for all input sequences, using 1 or 4 spp, with or without hardware texture mipmaps (See Fig. 5). In fact, a good behavior is observed even when filtering very noisy input (4 spp using no mipmaps). As it can be seen in Fig. 5 SANMIGUEL-BALCONY, our method produces slightly sharper results without anisotropic mipmaps. For this configuration, the output is temporally quite coherent while the complete lack of texture prefiltering is not noticeable in the image quality (See the supplementary video). We believe this is an interesting result showing the robustness of our method against flicker.

8. Conclusions and Future Work

In this paper, we have proposed a new real-time temporal filtering and antialiasing method for rasterization graphics pipelines. We have proposed a general spatio-temporal filtering framework based on a fast RLS algorithm to track temporal shading changes using linear models. Our method predicts per-pixel variations of the shading function between consecutive frames and combines temporal reprojection with spatial predictions in order to reconstruct temporally coherent pixel shading. Our approach consistently increases the temporal coherence of input images and naturally addresses different sources of potential aliasing artifacts, i.e. spatial and temporal undersamplings. We have also proposed Pixel History Linear Models (PHLM), a new concept for modeling, storing and tracking the history of pixel shading values using linear models. By tracking pixel shading values produced by the rasterization hardware in real-time, PHLM drastically reduces ghosting artifacts and significantly decreases the overblur produced by previous real-time temporal deflickering filters.

As a future work, we would like to further optimize our implementation with better G-buffer layouts, better use of GPU memory hierarchy and exploring strategies for linear model data compression or frameless model updates [IGGM10]. A second research direction for future work is to study the potential benefits of coupling PHLM with different strategies for sampling and rendering to further reduce the levels of overblur. Finally, another interesting

direction will be to extend PHLM with temporal jittering during camera motions.

Acknowledgments

We are thankful to Martin Klaudiny for helping us with the manuscript preparation. We thank Bei Yang for making possible this internal collaboration within Disney. We also thank Nick Swafford, Iván Huerta-Casado and the anonymous reviewers for their insightful feedback. The OPENWORLD and SPACELAND scenes were prepared by Karen Darragh and Joanna Jamrozy. OPENWORLD assets are courtesy of Epic Games Inc. The spaceship models in the SPACELAND scene were obtained from TurboSquid (www.turbosquid.com). The SANMIGUEL scene [McG11] was originally modeled by Guillermo M. Leal Llaguno, and we thank Maggie Kosek for her careful job in remodeling and adapting it. This work was funded by the InnovateUK project #101858.

References

[AČMS10] AYDIN T. O., ČADÍK M., MYSZKOWSKI K., SEIDEL H.-P.: Video quality assessment for computer graphics applications. In *ACM Transactions on Graphics (TOG)* (2010), vol. 29, p. 161. 7

[AHTAM14] ANDERSSON M., HASSELGREN J., TOTH R., AKENINE-MÖILER T.: Adaptive texture space shading for stochastic rendering. In *Computer Graphics Forum* (2014), vol. 33, pp. 341–350. 3

[Ake93] AKELEY K.: Reality engine graphics. In *Proc. of the 20th annual conference on Computer Graphics and Interactive Techniques* (1993), ACM, pp. 109–116. 2

[Bar99] BARTEN P. G.: *Contrast sensitivity of the human eye and its effects on image quality*, vol. 72. SPIE press, 1999. 5, 7

[BMS*12] BOWLES H., MITCHELL K., SUMNER R. W., MOORE J., GROSS M.: Iterative image warping. In *Computer Graphics Forum* (2012), vol. 31, pp. 237–246. 3

[BN12] BRUNETON E., NEYRET F.: A survey of nonlinear prefiltering methods for efficient and accurate surface shading. *IEEE Transactions on Visualization and Computer Graphics 18*, 2 (2012), 242–260. 2

[CMFL15] CRASSIN C., MCGUIRE M., FATAHALIAN K., LEFOHN A.: Aggregate g-buffer anti-aliasing. In *Proc. of the 19th Symposium on Interactive 3D Graphics and Games* (2015), ACM, pp. 109–119. 3

[CTM13] CLARBERG P., TOTH R., MUNKBERG J.: A sort-based deferred shading architecture for decoupled sampling. *ACM Transactions on Graphics (TOG) 32*, 4 (2013), 141. 3

[DLHSV08] DI LIETO N., HAGUENAUER P., SAHLMANN J., VASISHT G.: Adaptive vibration cancellation on large telescopes for stellar interferometry. In *SPIE Astronomical Telescopes+ Instrumentation* (2008), International Society for Optics and Photonics, pp. 70130H–70130H. 5

[Dro15] DROBOT M.: Hybrid reconstruction antialiasing. *GPU Pro 6: Advanced Rendering Techniques* (2015), 101. 3

[Epi] EPIC GAMES: Unreal Engine 4. https://www.unrealengine.com/. Accessed: 2016-05-16. 1, 3, 7

[FKS16] FOLEY T., KAPLANYAN A., SALVI M.: From the lab bench: Real-time rendering advances from NVIDIA Research. *GDC Talk* (2016). 3, 6

[HEMS10] HERZOG R., EISEMANN E., MYSZKOWSKI K., SEIDEL H.-P.: Spatio-temporal upsampling on the GPU. In *Proc. of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games* (2010), ACM, pp. 91–98. 2, 3

[IGGM10] IGLESIAS-GUITIÁN J. A., GOBBETTI E., MARTON F.: View-dependent exploration of massive volumetric models on large-scale light field displays. *The Visual Computer 26*, 6-8 (2010), 1037–1047. 9

[JGY*11] JIMENEZ J., GUTIERREZ D., YANG J., RESHETOV A., DEMOREUILLE P., BERGHOFF T., PERTHUIS C., YU H., MCGUIRE M., LOTTES T., MALAN H., PERSSON E., ANDREEV D., SOUSA T.: Filtering approaches for real-time anti-aliasing. In *ACM SIGGRAPH Courses* (2011). 2

[Kar14] KARIS B.: High-quality temporal supersampling. Advances in real-time rendering in games. In *ACM SIGGRAPH 2014 Courses* (2014), ACM, p. 10. 2, 3, 5, 7

[LD12] LIKTOR G., DACHSBACHER C.: Decoupled deferred shading for hardware rasterization. In *Proc. of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2012), ACM, pp. 143–150. 2

[Lot11] LOTTES T.: *FXAA-Whitepaper*. Tech. rep., NVIDIA, 2011. 2, 7

[LS87] LJUNG L., SÖDERSTRÖM T.: *Theory and practice of recursive identification*. MIT Press, 1987. 5

[Mal12] MALAN H.: Realtime global illumination and reflections in dust 514. Advances in real-time rendering in games. In *ACM SIGGRAPH 2014 Courses* (2012), ACM. 3, 5, 6

[McG11] MCGUIRE M.: Computer Graphics Archive, August 2011. URL: http://graphics.cs.williams.edu/data. 10

[MIGYM15] MOON B., IGLESIAS-GUITIAN J. A., YOON S.-E., MITCHELL K.: Adaptive rendering with linear predictions. *ACM Transactions on Graphics (TOG) 34*, 4 (July 2015), 121:1–121:11. 3, 4, 5

[NSL*07] NEHAB D., SANDER P. V., LAWRENCE J., TATARCHUK N., ISIDORO J. R.: Accelerating real-time shading with reverse reprojection caching. In *Graphics hardware* (2007), vol. 41, pp. 61–62. 3, 6

[Pra78] PRATT W. K.: Digital image processing. *A Wiley-Interscience Publication, New York: Wiley, 1978 1* (1978). 2

[Res09] RESHETOV A.: Morphological antialiasing. In *Proc. of the Conference on High Performance Graphics 2009* (2009), ACM, pp. 109–116. 2

[RKLC*11] RAGAN-KELLEY J., LEHTINEN J., CHEN J., DOGGETT M., DURAND F.: Decoupled sampling for graphics pipelines. *ACM Transactions on Graphics (TOG) 30*, 3 (May 2011), 17:1–17:17. 2

[SV12] SALVI M., VIDIMČE K.: Surface based anti-aliasing. In *Proc. of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2012), ACM, pp. 159–164. 2

[SYM*12] SCHERZER D., YANG L., MATTAUSCH O., NEHAB D., SANDER P. V., WIMMER M., EISEMANN E.: Temporal coherence methods in real-time rendering. In *Computer Graphics Forum* (2012), vol. 31, pp. 2378–2408. 2, 3, 6

[WDG02] WALTER B., DRETTAKIS G., GREENBERG D. P.: Enhancing and optimizing the render cache. In *Proc. of the Eurographics Workshop on Rendering* (2002), ACM Press, pp. 37–42. 3

[WDP99] WALTER B., DRETTAKIS G., PARKER S.: Interactive rendering using the render cache. *Rendering Techniques'99* (1999), 19–30. 3

[Wil83] WILLIAMS L.: Pyramidal parametrics. In *ACM Siggraph Computer Graphics* (1983), vol. 17, ACM, pp. 1–11. 2

[WWHS15] WANG Y., WYMAN C., HE Y., SEN P.: Decoupled coverage anti-aliasing. In *Proc. of the 7th Conference on High-Performance Graphics* (2015), ACM, pp. 33–42. 2

[YBS08] YANG H., BOYCE J. M., STEIN A.: Effective flicker removal from periodic intra frames and accurate flicker measurement. In *Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on* (2008), IEEE, pp. 2868–2871. 7

[YNS*09] YANG L., NEHAB D., SANDER P. V., SITTHI-AMORN P., LAWRENCE J., HOPPE H.: Amortized supersampling. *ACM Transactions on Graphics (TOG) 28*, 5 (2009), 135. 2, 3, 6

[YTS*11] YANG L., TSE Y.-C., SANDER P. V., LAWRENCE J., NEHAB D., HOPPE H., WILKINS C. L.: Image-based bidirectional scene reprojection. *ACM Transactions on Graphics (TOG) 30*, 6 (2011), 150. 3

[ZY10] ZHAO Y., YU L.: A perceptual metric for evaluating quality of synthesized sequences in 3DV system. *Proc. of SPIE 7744* (2010). 7