Eurographics Symposium on Rendering 2008
Steve Marschner and Michael Wimmer
(Guest Editors)

Volume 27 (2008), Number 4

Geometry-Aware Framebuffer Level of Detail

Lei Yang[1] Pedro V. Sander[1] Jason Lawrence[2]

[1]Hong Kong University of Science and Technology [2]University of Virginia

Abstract

This paper introduces a framebuffer level of detail algorithm for controlling the pixel workload in an interactive rendering application. Our basic strategy is to evaluate the shading in a low resolution buffer and, in a second rendering pass, resample this buffer at the desired screen resolution. The size of the lower resolution buffer provides a trade-off between rendering time and the level of detail in the final shading. In order to reduce approximation error we use a feature-preserving reconstruction technique that more faithfully approximates the shading near depth and normal discontinuities. We also demonstrate how intermediate components of the shading can be selectively resized to provide finer-grained control over resource allocation. Finally, we introduce a simple control mechanism that continuously adjusts the amount of resizing necessary to maintain a target framerate. These techniques do not require any preprocessing, are straightforward to implement on modern GPUs, and are shown to provide significant performance gains for several pixel-bound scenes.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation

1. Introduction and Related Work

Programmable graphics hardware has brought with it a steady increase in the complexity of real-time rendering applications, often expressed as a greater number of arithmetic operations and texture accesses required to shade each pixel. In response, researchers have begun investigating fully general and automatic methods for reducing the pixel processing workload at an acceptable amount of approximation error. These include techniques for automatically simplifying shader programs [OKS03, Pel05] and data reuse [ZWL05, DWWL05, NSL*07] which exploit the spatio-temporal coherence in animated image sequences by reusing data generated in previously rendered frames to accelerate the calculation of the shading in the current frame. A related approach is to exploit the spatial coherence within just a single frame by simply rendering the scene to a lower resolution buffer that is then upsampled to the desired screen resolution in a second rendering pass. Introduced as *dynamic video resizing* in the *InfiniteReality* rendering system [MBDM97], this simple technique is effective for controlling the pixel workload, but suffers from noticeable sampling artifacts near silhouettes and other discontinuities in the shading (Figure 2).

This paper extends conventional dynamic resizing in three key ways to provide a framebuffer level of detail (LOD) system. Following Laine et al. [LSK*07] and Sloan et al. [SGNS07], we apply the joint bilateral filter [TM98, KCLU07] to consider differences in scene depth and surface orientation during framebuffer upsampling. Second, we extend traditional resizing to allow computing partial components of the shading at different resolutions. This allows, for example, evaluating a detailed texture map at the full framebuffer resolution while still reaping the performance benefits of resizing a more computationally intensive intermediate calculation. Third, we present a feedback control mechanism that automatically determines the degree of resizing necessary to achieve a target framerate. This approach is analogous to geometry LOD methods [LWC*02] in that it controls one aspect of the rendering workload by reducing the number of pixels that are processed. We evaluate these techniques using several pixel-bound test scenes to demonstrate their benefits over conventional resizing.

Our approach bears a number of similarities to interleaved sampling techniques [SIMP06, LSK*07] which reconstruct a smooth approximation of some component of the shading (e.g., indirect illumination) from a noisy estimate. As this noisy approximation is smoothed to produce the final frame, care must be taken to avoid integrating across discontinuities in scene geometry. The method proposed by Laine et al. [LSK*07] uses a bilateral filter similar to the one we apply here. However, interleaved sampling reduces the average number of samples processed in the shader at every pixel (e.g., virtual point light sources accumulated [LSK*07]) whereas dynamic resizing reduces the number of expensive shader invocations within a single frame. In this respect, they are complimentary approaches for addressing the same problem of reducing shading costs.

The benefit of selectively resizing an expensive component of the shading has been demonstrated in interactive systems that simulate certain aspects of global illumination [SGNS07, HPB06]. In fact, Sloan et al. [SGNS07] also apply a reconstruction filter similar to ours. However, these systems have all focused on optimizing a specific rendering task in which resizing some partial value brings a profitable speed/error trade-off. This paper focuses on the role geometry-aware resizing plays within a general strategy for regulating shading costs and we present a number of different effects for which resizing has clear benefits. Finally, a number of hybrid CPU/GPU cache-based interactive rendering systems have also explored edge-aware framebuffer reconstruction [BWG03, VALBW06]. Although these methods may apply in this context as well, our emphasis is on providing a lightweight method that runs entirely on the GPU and can be easily incorporated into existing real-time systems.

2. Geometry-Aware Framebuffer Resizing

In our approach each frame is rendered in two passes. In the first pass, the scene geometry is rendered to a low resolution buffer L which stores the color, depth, and surface normal at each pixel. In the second pass, the scene geometry is rendered to the framebuffer H at the desired screen resolution using a pixel shader that reconstructs the shading from L. The resizing factor r is equal to the ratio of the width of H relative to that of L; resizing the framebuffer by a factor of $r = 2$ will thus reduce the pixel load by roughly a factor of 4. We next describe each of these passes in detail.

2.1. Rendering L

The original pixel shader is used to render the scene to the low resolution buffer L. In addition to surface color, we also store the depth and surface normal at each pixel (this is similar to the G-buffer used in deferred shading [DWS*88, ST90]). This information allows the second pass to consider discontinuities in the scene geometry when reconstructing the shading. Our current implementation packs this data into a single 16-bit RGBA render target (8-bit RGBA packed on RG, 16-bit depth on B, and 8 bits for each of the normal's x- and y- components on A). Alternatively, this data could be maintained in a separate buffer using multiple render targets.

2.2. Reconstructing H from L

Montrym et al. [MBDM97] simply upsample L to match the resolution of H using a bilinear reconstruction filter. Although this can be performed very efficiently, it leaves visible artifacts along depth boundaries and near high-frequency features of the shading. We use a geometry-aware reconstruction filter inspired by the bilateral filter [TM98] to reduce these artifacts. The basic idea is to modify the reconstruction kernel at each pixel to avoid integrating across shading data associated with different regions in the scene. Specifically, we weight the contribution of each pixel in L toward the reconstruction of each pixel in H based not only on their distance in the image plane, but also on the difference between their respective scene depths and surface normals. At each rendered pixel x_i in H, a pixel shader first computes the surface normal n_i^H and scene depth z_i^H and reconstructs the surface color c_i^H according to

$$c_i^H = \frac{\sum c_j^L \, f(\hat{x}_i, x_j) \, g(|n_i^H - n_j^L|, \sigma_n) \, g(|z_i^H - z_j^L|, \sigma_z)}{\sum f(\hat{x}_i, x_j) \, g(|n_i^H - n_j^L|, \sigma_n) \, g(|z_i^H - z_j^L|, \sigma_z)}, \quad (1)$$

where c_j^L, n_j^L, and z_j^L are the respective counterparts in L and $\hat{x}_i$ is the position of x_i after having been downsampled in each dimension by r. Note that the normals are defined in screen-space so the vector $n = (0, 0, 1)$ points toward the viewer. This sum is carried out over all the pixels in L. The *domain filter* f weights the contribution of each pixel in L according to its distance in the image plane. Instead of modeling this as a Gaussian, as is commonly done with the bilateral filter, we instead use a simple tent function which restricts this sum to only the 2×2 grid of nearest pixels. We found this gave acceptable results at a fraction of the compute overhead. The *range filters* $g(\cdot, \sigma_n)$ and $g(\cdot, \sigma_z)$ are modeled as Gaussians and weight each pixel based on differences in surface normals and scene depths, respectively.

A similar generalization of the bilateral filter was proposed by Toler-Franklin et al. [TFFR07] for manipulating natural images augmented with per-pixel surface normals. They note the importance of accounting for foreshortening when computing the distance between screen-space normals. Following that work, we transform each normal vector (n_x, n_y, n_z) by its z-component $(n_x/n_z, n_y/n_z, 1)$ before computing the Euclidean distance.

The standard deviations σ_n and σ_z are important parameters that determine the extent to which shading information is *shared* between surface regions with different depths and orientations. Figure 1 illustrates the effect of these parameters at different resizing factors. Smaller values of σ_z limit the amount of blending across depth boundaries as can be seen along the silhouette edges of the car. Smaller values of σ_n limit the amount of blending across orientation boundaries like where the rear-view mirror meets the car's body. Although larger values of these parameters allow a larger number of samples to influence each reconstructed pixel, we have found it's almost always desirable to use small values in order to preserve sharp boundaries in the scene geometry. Finally, note that this approach subsumes conventional dynamic resizing which can be obtained by setting $\sigma_n, \sigma_z = \infty$ (leftmost images in Figure 1).

Equation 1 can be efficiently computed in a pixel shader

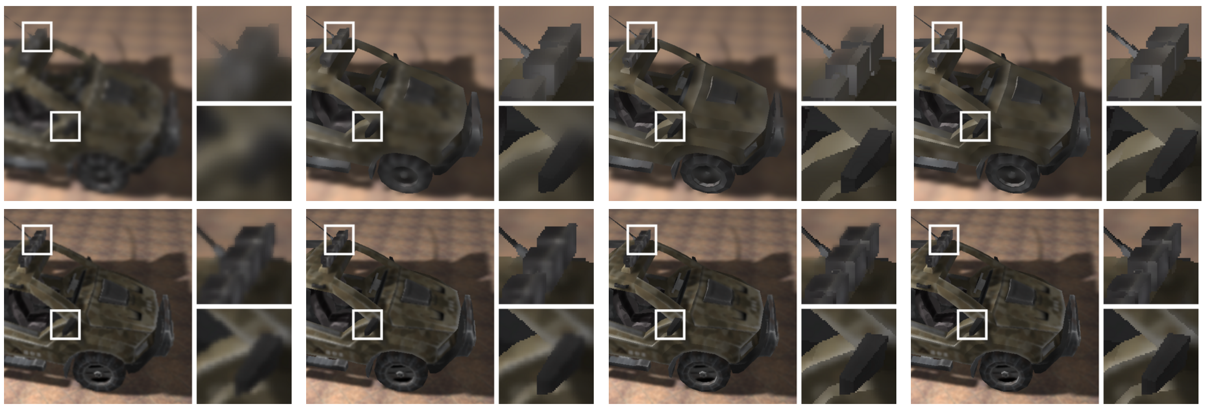
Large σ_z, large σ_n
Small σ_z, large σ_n
Large σ_z, small σ_n
Small σ_z, small σ_n

Figure 1: *Framebuffer resized by a factor of (top row)* $r = 7.3$ *and (bottom row)* $r = 2.0$ *using depth and normal range filters of varying widths.*

since evaluating the two Gaussians can be easily vectorized. Compared to bilinear reconstruction, we have found this adaptive kernel requires very little additional overhead, especially for heavily pixel-bound scenes (Figure 2).

2.3. Fine-Grained Resizing

Our basic approach can be extended to defer the evaluation of certain components in the shading until the second pass. This allows resizing only those calculations that are particularly expensive while evaluating relatively inexpensive calculations at the framebuffer's full resolution. For shaders that combine an expensive calculation that results in a smooth spatial signal (e.g., soft shadows, subsurface scattering approximation) with an inexpensive yet detailed calculation, this can lead to dramatic speedups. This offers finer-grained control over the way resources are allocated to evaluate the shading in each frame and thus greater flexibility for optimizing pixel-bound scenes.

Fine-grained resizing is related to deferred shading [DWS*88, ST90], which renders the attributes of the geometry to multiple screen-sized render targets and then selectively performs additional computation by rendering proxy quadrilaterals. Deferred shading is particularly useful for scenes with high geometric complexity since it supports multi-pass shading without rendering the geometry multiple times. On the other hand, our approach targets scenes with high shading complexity by reducing the resolution of expensive computations and deferring inexpensive yet detailed components until they can be calculated at full resolution. Although our approach does require processing the geometry twice, we do not require rendering all of the attributes of the geometry that are used for shading to multiple render targets like with deferred shading, but rather must only store depths and normals.

3. Automatic Framerate Control

Dynamic resizing presents a clear error/performance tradeoff which is regulated by the resizing factor r. The *Infinite-Reality* system [MBDM97] used a feedback control mechanism to set r based on geometry-raster FIFO statistics provided by the underlying hardware to determine when the application was fill rate limited. We propose a similar approach, but one that requires only the total amount of time spent rendering the previous frame which can be easily monitored by the user-level application. If we assume the pixel processing time is directly proportional to the number of shaded pixels then the time required to render a single frame can be modeled as

$$t \approx Kp + C, \tag{2}$$

where p is the number of pixels processed in L, K is equal to the cost of processing each pixel, and C captures the remaining overhead which includes geometry processing and rendering the final pass. Dynamic resizing is useful for shaders that are highly pixel bound in which case the first term in Equation 2 dominates the sum. If we assume the coverage of the model on the screen does not change then $p \propto wh/r^2$ (for a framebuffer of size $w \times h$) and we obtain

$$t \approx K'(1/r^2) + C \tag{3}$$

where K' accounts for the size of the framebuffer and the percent that is currently shaded. This gives the desired relationship between the change in rendering time Δt and the change in the resize factor Δr:

$$\Delta t \approx K' \, \Delta(1/r^2). \tag{4}$$

We estimate these quantities using the rendering times of the previous and current frames, as well as the previous scale factor; the value K' is experimentally determined using a scene with the expected workload. At runtime, we adjust the scale factor r to produce a Δt that will maintain the target framerate. In practice, we limit the range of Δr to 0.2 between consecutive frames and disallow values below $r = 1.0$.

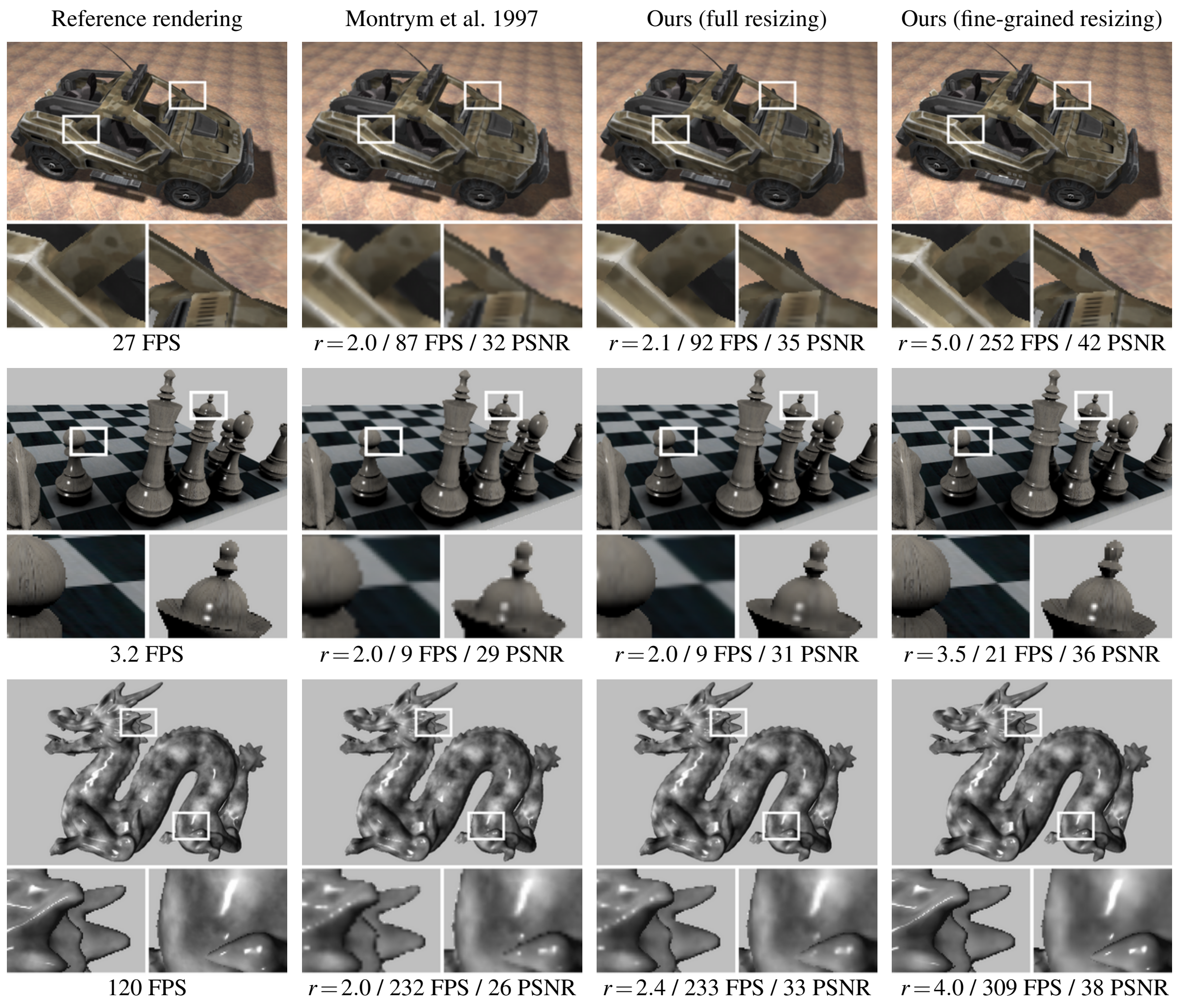
Reference rendering
Montrym et al. 1997
Ours (full resizing)
Ours (fine-grained resizing)
27 FPS
r = 2.0 / 87 FPS / 32 PSNR
r = 2.1 / 92 FPS / 35 PSNR
r = 5.0 / 252 FPS / 42 PSNR
3.2 FPS
r = 2.0 / 9 FPS / 29 PSNR
r = 2.0 / 9 FPS / 31 PSNR
r = 3.5 / 21 FPS / 36 PSNR
120 FPS
r = 2.0 / 232 FPS / 26 PSNR
r = 2.4 / 233 FPS / 33 PSNR
r = 4.0 / 309 FPS / 38 PSNR

Figure 2: *Comparison between (first column) the original shading, (second column) conventional dynamic resizing, (third column) our approach where the full shading is resized, and (fourth column) our approach with fine-grained resizing.*

4. Results

We have evaluated our resizing and LOD techniques using several test scenes with high pixel processing workloads shown in the left column in Figure 2. The *Car* scene combines 9×9 shadow map queries using the percentage-closer filtering (PCF) [RSC87] to generate antialiased shadow boundaries. The *Dragon* scene combines a procedural noise function [Per85] with a simple Phong specular layer. The *Chess* scene computes ambient occlusion at the pixel level using the algorithm described by [HJ07]. All of our experiments were performed using a HP Compaq DX7300 PC equipped with an AMD ATI HD2900XT graphics card.

Figure 2 shows equal time comparisons between conventional resizing [MBDM97] and our geometry-aware resizing. Because the pixel workload in each scene far exceeds the geometry workload, the additional rendering pass required for our approach amounts to a very small overhead and both approaches give significant speedups at even moderate resizing factors of around $r = 2.0$. Our adaptive reconstruction filter clearly does a better job of preserving sharp geometric features in the scene. A limitation of both methods is that high-frequency features in the shading itself are inevitably lost; this is clearly noticeable in the camouflage pattern in the *Car* scene which is represented as a diffuse texture map. However, the evaluation of this component may be deferred to the second pass while still resizing the shadow calculation. Similarly, we can defer the diffuse texture lookup and specular lighting calculation in the *Chess* scene (resizing only the ambient occlusion calculation) as well as the specular calculation in the *Dragon* scene (resizing only the procedural noise calculation). The rightmost column in Figure 2 shows the results of fine-grained resizing. Note that perceptually important features in the

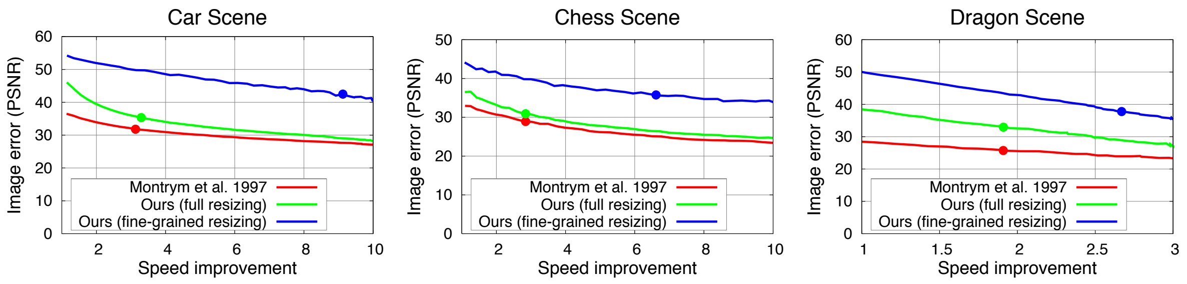

Figure 3: *Image error as a function of speed improvement for the three test scenes. The points that correspond to the images shown in Figure 2 are highlighted.*

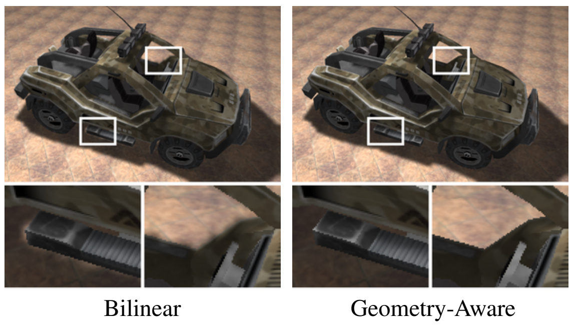

Figure 4: *Using bilinear upsampling with fine-grained resizing can cause the resized elements to "lead" across depth boundaries. Our geometry-aware reconstruction filter eliminates this problem.*

shading are better maintained, but at much higher framerates. Figure 3 plots the performance improvement vs image error (PSNR) for these same scenes and resizing amounts. Our technique provides higher image quality at equal performance improvements as compared to standard resizing in all cases while fine-grained resizing provides a more impressive error/performance trade-off than either. Finally, performing fine-grained resizing with standard bilinear upsampling can cause data to "leak" across silhouette boundaries. Figure 4 illustrates this problem for the *Car* scene where you can clearly see haloing artifacts in the shadows near the car's silhouette edges. These artifacts are avoided with our geometry-aware reconstruction filter.

Figure 5 visualizes our automatic feedback control system being used to control the framerate in the *Car* scene. Note that the framerate oscillates heavily for a fixed resizing factor (Figure 5a), while our technique automatically adjusts the resizing factor in order to maintain a smoother and more constant framerate (Figure 5b).

5. Discussion

It would be possible to build a similar LOD system that resizes calculations performed in a coordinate system other than screen-space. For example, many effects compute elements of the shading in texture-space. Although texture-space resizing would avoid the problems associated with handling geometric discontinuities in the scene, it would introduce others that are avoided by our approach. First, it would require surface parameterizations for all of the models in the scene and the use of different render targets for each visible object. Second, it would still suffer from *texture seams* which is a different type of discontinuity that would need to be addressed during filtering. A key benefit of our screen-space approach is that multiple objects and shaders may all share the same low resolution buffer with minimal modification to the original application.

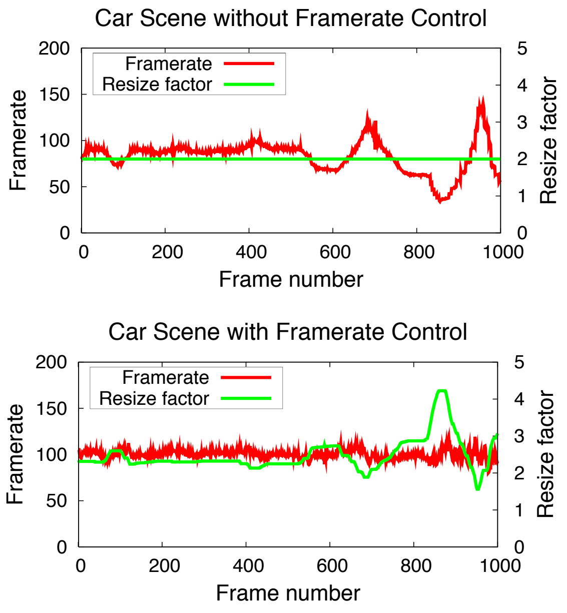

Figure 5: *Framerate and resize factor over a sequence of frames of the Car scene.* ***Top:*** *For a fixed scale factor r, the framerate oscillates heavily.* ***Bottom:*** *Our framerate control system automatically adjusts r to maintain a framerate.*

An important limitation of our approach is that scenes with very fine geometric detail may not be adequately reproduced in the low resolution buffer, resulting in artifacts in the final shading at pixels for which all of the weights in Equation 1 are small. Note that interleaved sampling [SIMP06,LSK*07] does not share this drawback since the shading at every pixel is partially evaluated. Finally, geometry-aware resizing is only appropriate for scenes that are limited by fill rate to such an extent that processing the geometry twice is outweighed by evaluating the shading at its full resolution. While not all scenes will exhibit this characteristic, we have demonstrated examples that do from a range of effects.

6. Conclusion

We have presented a framebuffer level of detail algorithm that provides continuous control over the pixel processing workload in real-time rendering applications. This work extends classic dynamic resizing to use a geometry-aware reconstruction filter and demonstrates how partial components in the shading may be selectively resized. These techniques are straightforward to incorporate into existing applications executing on modern GPUs and were shown to provide notable performance improvements for several pixel-bound scenes. We also presented a simple feedback control mechanism that automatically adjusts the degree of resizing in order to maintain a target framerate.

Acknowledgements

This work was funded by the Hong Kong RGC grant #619207. Jason Lawrence also acknowledges a generous gift from Nvidia. We thank Chris Iacobucci for the car model and the reviewers for their helpful feedback.

References

[BWG03] BALA K., WALTER B., GREENBERG D. P.: Combining edges and points for interactive high-quality rendering. *ACM Transactions on Graphics 22*, 3 (2003), 631–640.

[DWS*88] DEERING M., WINNER S., SCHEDIWY B., DUFFY C., HUNT N.: The triangle processor and normal vector shader: a VLSI system for high performance graphics. In *Computer Graphics (Proceedings of ACM SIGGRAPH 88)* (1988), ACM, pp. 21–30.

[DWWL05] DAYAL A., WOOLLEY C., WATSON B., LUEBKE D.: Adaptive frameless rendering. In *Proceedings of the Eurographics Symposium on Rendering* (2005).

[HJ07] HOBEROCK J., JIA Y.: High-quality ambient occlusion. In *GPU Gems 3*, Nguyen H., (Ed.). Addison-Wesley, 2007, pp. 257–274.

[HPB06] HAŠAN M., PELLACINI F., BALA K.: Direct-to-indirect transfer for cinematic relighting. *ACM Transactions on Graphics 25*, 3 (2006), 1089–1097.

[KCLU07] KOPF J., COHEN M. F., LISCHINSKI D., UYTTENDAELE M.: Joint bilateral upsampling. *ACM Transactions on Graphics 26*, 3 (2007), 96:1–96:5.

[LSK*07] LAINE S., SARANSAARI H., KONTKANEN J., LEHTINEN J., AILA T.: Incremental instant radiosity for real-time indirect illumination. In *Proceedings of the Eurographics Symposium on Rendering* (2007).

[LWC*02] LUEBKE D., WATSON B., COHEN J. D., REDDY M., VARSHNEY A.: *Level of Detail for 3D Graphics*. Elsevier Science Inc., 2002.

[MBDM97] MONTRYM J. S., BAUM D. R., DIGNAM D. L., MIGDAL C. J.: InfiniteReality: A real-time graphics system. In *Proceedings of ACM SIGGRAPH* (1997).

[NSL*07] NEHAB D., SANDER P. V., LAWRENCE J., TATARCHUK N., ISIDORO J. R.: Accelerating real-time shading with reverse reprojection caching. In *Proceedings of Graphics Hardware* (2007).

[OKS03] OLANO M., KUEHNE B., SIMMONS M.: Automatic shader level of detail. In *Proceedings of Graphics Hardware* (2003).

[Pel05] PELLACINI F.: User-configurable automatic shader simplification. *ACM Transacations on Graphics 24*, 3 (2005), 445–452.

[Per85] PERLIN K.: An image synthesizer. In *Computer Graphics (Proceedings of ACM SIGGRAPH)* (1985).

[RSC87] REEVES W. T., SALESIN D. H., COOK R. L.: Rendering antialiased shadows with depth maps. In *Computer Graphics (Proceedings of ACM SIGGRAPH)* (1987), pp. 283–291.

[SGNS07] SLOAN P.-P., GOVINDARAJU N. K., NOWROUZEZAHRAI D., SNYDER J.: Image-based proxy accumulation for real-time soft global illumination. In *Proceedings of Pacific Graphics* (2007).

[SIMP06] SEGOVIA B., IEHL J.-C., MITANCHEY R., PÉROCHE B.: Non-interleaved deferred shading of interleaved sample patterns. In *Proceedings of Graphics Hardware* (2006).

[ST90] SAITO T., TAKAHASHI T.: Comprehensible rendering of 3-d shapes. In *Computer Graphics (Proceedings of ACM SIGGRAPH)* (1990), pp. 197–206.

[TFFR07] TOLER-FRANKLIN C., FINKELSTEIN A., RUSINKIEWICZ S.: Illustration of complex real-world objects using images with normals. In *Proceedings of the International Symposium on Non-Photorealistic Animation and Rendering* (2007).

[TM98] TOMASI C., MANDUCHI R.: Bilateral filtering for gray and color images. In *Proceedings of the International Conference on Computer Vision* (1998), pp. 839–846.

[VALBW06] VELÁZQUEZ-ARMENDÁRIZ E., LEE E., BALA K., WALTER B.: Implementing the render cache and the edge-and-point image on graphics hardware. In *Proceedings of Graphics Interface* (2006), pp. 211–217.

[ZWL05] ZHU T., WANG R., LUEBKE D.: A GPU accelerated render cache. In *Pacific Graphics* (2005).