LETTER

Dynamic Rendering Quality Scaling Based on Resolution Changes

MinKyu KIM[†], ***Member***, **SunHo KI**[††], **YoungDuke SEO**[††], **JinHong PARK**[††a)], ***and*** **ChuShik JHON**[†], ***Nonmembers***

SUMMARY Recently in the mobile graphic industry, ultra-realistic visual qualities with 60fps and limited power budget for GPU have been required. For graphics-heavy applications that run at 30 fps, we easily observed very noticeable flickering artifacts. Further, the workload imposed by high resolutions at high frame rates directly decreases the battery life. Unlike the recent frame rate up sampling algorithms which remedy the flickering but cause inevitable significant overheads to reconstruct intermediate frames, we propose a dynamic rendering quality scaling (DRQS) that includes dynamic rendering based on resolution changes and quality scaling to increase the frame rate with negligible overhead using a transform matrix. Further DRQS reduces the workload up to 32% without human visual-perceptual changes for graphics-light applications.

key words: *GPU real-time rendering, frame rate up-sampling, inter-frame differential estimation, GPU power optimizations*

1. Introduction

Recently in the mobile graphics industry nowadays, there has been a strong trend towards complex rendering for high resolution displays for PC-like realistic visual quality within a very limited computational power budget. Moreover, most of the modern mobile display devices refresh at 60Hz. Although mobile GPU has hardware evolved remarkably, it does not yet satisfy the ever increasing market demand for PC-like high-quality graphics while maintaining a consistent 60 fps. For an application running at 30 fps, we can easily observe very noticeable motion flickering due to fast moving objects and/or cameras. In order to help smooth out the perceived object motions, recent studies are constantly looking for creative ways to up-sample the low frame rate.

Frame interpolation schemes are widely used to optimize image quality. By employing the interpolation scheme, to generate the intermediate frame, two approaches can be utilized: a forward re-projection and a reverse re-projection. In recent studies, a temporal up sampling technique based on the forward re-projection [4] is proposed, which is targeted at higher-frame-rate displays. This technique relies on perceptual findings. In this approach, blurred frame insertion and mesh-based warping are combined to preserve the naturalness of the original sequence for the human visual system (HVS). Based on the reverse re-projection, a spatio-temporal up sampling strategy [5] is proposed, which exploits spatial and temporal redundancy. However, this approach requires many previous frames to be kept in memory and also requires a complex adaptation of the spatial-temporal weighting function for better performance. Recently, [3] this weighting function has been simplified based on a specular lobe similarity and reduced complexity. These approaches give a better image quality than spatial-only or temporal-only up-sampling. However, since they are based on unidirectional re-projection, there are also drawbacks which is caused by interpolation, such as occlusions, lags, and holes which appear in the extrapolated frames. These drawbacks need to be resolved.

To address these problems, an image based bi-directional re-projection technique [2] is proposed that retrieves information from a pair of consecutive rendered frames and interpolates them. Their method uses fixed-point iterations to find a correct pixel correspondence between originally rendered views and interpolated ones. Later, this technique was combined with mesh-based techniques [1]. Although the entire process of the above techniques fits in a pixel shader and can be computed quickly, it requires several additional search heuristics to handle the discontinuity. The most crucial problem is that these methods highly depend on their implementations and the computational cost is very expensive for mobile devices. Additionally, if there are no human perceptible benefits, the workload imposed by a fixed high resolution at a fixed high frame rate is pointless. In other words, the workload can be reduced by scaling the resolution adaptively as long as we can guarantee the visual quality.

In this paper, we propose a dynamic rendering quality scaling (DRQS) technique that increases the frame rate with a very minimal overhead, introducing a combination of the proposed quality scaling algorithm and the novel dynamic rendering algorithm using a transform matrix. This leads to substantial improvements to quality of service. Furthermore, the DRQS can optimize power consumption without human visual-perceptual changes.

2. Dynamic Rendering Quality Scaling

Figure 1 shows the system overview of our solution for performing temporal and spatial resolution enhancement of GPU rendering using the proposed DRQS technique. Our proposed method is located in between graphic libraries and

Manuscript received June 5, 2015.
Manuscript revised August 18, 2015.
Manuscript publicized September 17, 2015.

[†]The authors are with the Department of Computer Science and Engineering, Seoul National University, Korea.
[††]The authors are with LG Electronics, Seoul, Korea.
a) E-mail: jinhong1.park@lge.com

DOI: 10.1587/transinf.2015EDL8130

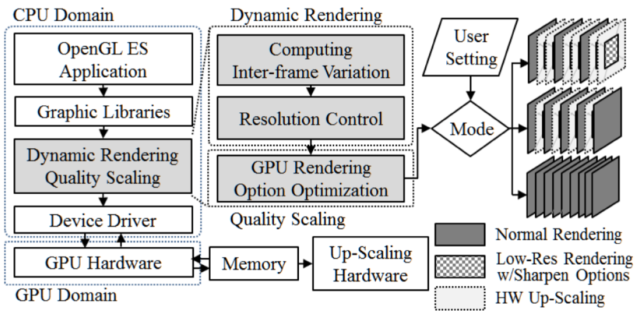

Fig. 1 System overview of dynamic rendering quality scaling

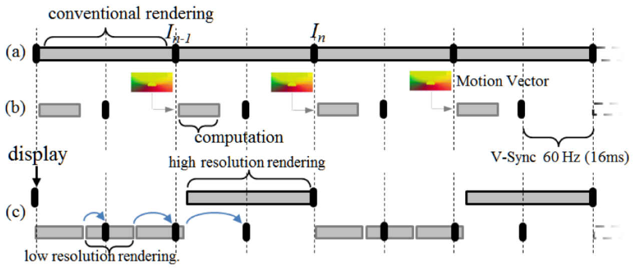

Fig. 2 Dynamic rendering technique for graphics-heavy contents; (a) GPU rendering at 30 fps, (b) Conventional FRUC, (c) DRQS

a device driver in order to capture *gl calls* for computing inter-frame variations and changing the rendering parameters. From these values, the dynamic rendering controls the resolution and the ratio of the low resolution frame to original frames to enhance the temporal resolution. The following sequence, which is quality scaling, configures the rendering parameters to enhance the spatial resolution.

2.1 Dynamic Rendering

As shown in Fig. 2, in order to increase the frame rate of a given example sequence (a), the most recent techniques have been mainly studied to motion-compensate, which commonly consist of two processes: motion estimation (ME) and motion-compensated interpolation (MCI). To generate intermediate frames using these algorithms (b), high computational costs are inevitable. Furthermore, it requires significant additional costs to resolve the well-known issues such as holes, occlusions and lags. Compared to previous studies, we propose a novel technique to naturally increase the frame rate up to 60 fps (c). Compared to conventional FRUC, the additional cost for computing inter-frame variation is negligible.

As shown in Fig. 3, the application requires a substantial GPU workload imposed by a high resolution and a fixed frame rate, 60 fps (a). The idle state was described by a v-sync limit, meaning that the GPU completes rendering jobs of a current frame before the v-sync of a mobile display system and gets into the idle state. The GPU workload can be reduced significantly by employing the proposed dynamic rendering (b), which renders the low resolution frame in between original frames to save power.

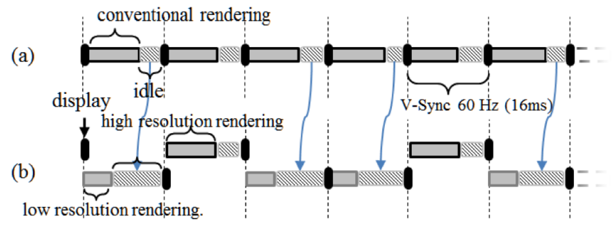

Fig. 3 Dynamic rendering technique for graphics-light contents for (a) GPU rendering at 60fps and (b) DRQS

```
Algorithm:
  Calculating the dynamic rendering parameter for the current frame $F_i$
    input parameter: $F_i$ with GPU rendering status and linked list $L_{i-1}$
    return: $P_i$ which is the ratio of original frames to low resolution frames
1:  function CalcDynamicRenderingParam($F_i$, $L_{i-1}$)
2:    Initialize a linked list; $L_i$
3:    while ( $j$ < Total count $J$ of Objects in $F_i$)
4:      Read the $j^{th}$ Object's ID; $O_{ij}$
5:      Read corresponding MVP matrix; $M_{ij}$
6:      Compute $V_{ij}=M_{ij} * I$
7:      Search $V_{(i-1)j}$ in the linked list $L_{i-1}$ using $O_{ij}$
8:      Compute $D_{ij} = distance(V_{ij}, V_{(i-1)j})$
9:      Insert the $node(O_{ij}, V_{ij}, D_{ij})$ to the linked list $L_i$
10:   Search ($D_{max}$) in the linked list $L_i$
11:   Normalize $D_{max}$ to $D_{nor}$
12:   Calculate the average of $D_{nor}$
13:   Select $P_i$ from comparing the avg. of $D_{nor}$ with pre-defined threshold
14:   $L_{i-1} = L_i$ ; discard the i-1th linked list
15: return $P_i$
```

Fig. 4 Algorithm for calculating of the dynamic rendering parameter

2.2 Algorithm for Computing Inter-Frame Variation

In order to obtain the number of low resolution frames in between original frames, we explicitly measure the amount of scene changes in the consecutive frames using the model view projection (MVP) matrix instead of the high-cost ME/MCI. MVP consists of three separate matrices: the model, view, and projection. This transformation matrix includes rotation, translation, scaling, reflecting, orthographic projection, and perspective projection and is generally used as a combination of several transformations to draw a scene. By retrieving MVP matrixes from *gl calls* and calculating them, we can measure the amount of scene changes. The main steps of algorithm are defined in Fig. 4:

Let F_i be the i^{th} frame in a rendering sequence. J denotes the total number of objects in a single frame F_i, which is the current frame. Since a single frame usually consists of multiple objects and each object is transformed by the corresponding MVP matrix, Object IDs are used to distinguish different objects in a scene. Thus, O_{ij} denotes the j^{th} object ID in the i^{th} frame. Also, the corresponding MVP matrix M_{ij} is denoted accordingly. Let $V_{ij} = (x, y, z, w = 1)$ be a diagnostic vector from multiplying the MVP matrix M_{ij} by the matrix I, where $I = (1, 1, 1, w = 1)$, a vector that expects frame-to-frame coherence using the amount of MVP matrix changes. The reason we used the $(1, 1, 1, w = 1)$ matrix is that MVP matrix multiplied by the $(1, 1, 1, w = 1)$ matrix can result in the extent of changes from the transform

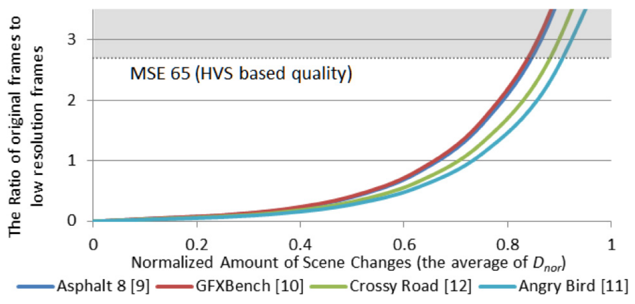

Fig. 5 Curves for the ratio of original frames to low resolution frames

using the MVP matrix. We compute a distance D_{ij} between V_{ij} and $V_{(i-1)j}$ which can represent the amount of each MVP matrix change.

As shown in the algorithm (lines 1-14), we employ the linked list L_i to stores different object IDs and their corresponding MVP matrices in the i^{th} frame and is used for searching the maximum distance values. In line 10-11, we achieve D_{max} in the linked list L_i and normalize D_{max} to D_{nor} by the maximum screen space distance. Then, we calculate the average of D_{nor} (line 12) and compare this value with a fine-grained threshold value (line 13) to achieve P_i which is the ratio of original frames to low resolution frames.

As observed in Fig. 5, we choose the representative applications and explore the variance in amount of scene changes for each application and the ratio of low resolution frame to original frames regarding the mean square error (MSE) as a quality insurance. The ratio of original frames to low resolution frames increases along with the amount of scene changes. As a result, we expect to increase quality of service and reduce the GPU workload respectively. However, when the ratio of original frames to low resolution frames goes above a certain threshold, this ratio cannot satisfy the HVS based visual quality. For example, in the case of graphics-heavy applications, if *gfxbench* [10] and *asphalt 8* [9], if the D_{nor} average is above 0.814 for *gfxbench* [10] and 0.811 for *asphalt 8* [9], then the 1:3 ratio of original frames to low resolution frames is suitable; but this ratio violates the HVS based visual quality [8]. Note that some or all objects can be added or removed from consecutive frames. In this case, the D_{ij} value becomes 1.0, which is equal to D_{max}, meaning that rapid scene changes occurred.

For graphics-light applications, *angry bird* [11] and *crossy road* [12], the curves are similar to the curves from the graphics-heavy applications but are less constrained by the ratio of original frames to low resolution frames. Regarding these curves, note that the low resolution is set to 720p (1280x720) as the default, and the ratio and the resolution can be customized by the user for flexibility.

2.3 Quality Scaling

The Image sharpness is an important factor that depends upon perceived image quality. Blurs in the image can be regarded as artifacts that are present in characters, texts, and static objects. Moreover, it easily indicates the visual quality. Using only dynamic rendering scheme causes inevitable blurring issues due to low resolution rendering and ordinary scaling algorithms. To address this problem, we introduce a novel quality scaling technique to enhance the spatial resolution. To sharpen the texture and edge, we find the optimal parameters for GPU rendering like anti-aliasing levels and texture mipmap levels. These parameters are configured before the rendering starts.

The detailed procedure of changing each parameter for low resolution rendering is described as follows. First, we decrease the multi-sample anti-aliasing rates to get shaper images. This approach can result in more aliased edges but this artifact can be compensated in the up-scaling process. Second, we change the texture mipmap levels to render a low resolution frame to enhance high-frequency details on a magnified texture. The sharpening computes the difference between the magnified finest level (level 0) and the next coarse level (level 1) [7]. A weighted version of the result is added to the magnified finest image to produce the sharpened result. The result is an extrapolation from the level 1 image to the level 0 image. The weighting factor applied to the difference f is a function of the magnification factor $f(L)$ where L denotes a LOD (level of detail) value. The equation to compute the texel color T from the top two texture levels and current magnification is:

$$T_{sharp} = (1 + f(L))T_0 - f(L)T_1 \tag{1}$$

where T_{sharp} is the new texel color, T_0 is the magnified texel color at level 0 and T_1 is the magnified texel color at level 1. We only consider mipmap levels of 0 to 1 for sharpen rendering, because the base mipmap image at level 0 is the finest, sharpest, unfiltered image. The $f(L)$ function takes the LOD value and produces a weighting factor between 0 and 1. Thus, when we use Eq. (1), we can get a sharper image for low resolution frames. Of course, this approach can increase the memory traffic for texturing. However, we believe this increased memory traffic is negligible because texture images commonly used in mobile applications are usually compressed using ASTC, PVRTC, ETC2, and others. Furthermore, we demonstrate that our method can be easily integrated into other existing methods, which is one of the benefits in terms of an implementation.

3. Experimental Results

In this section, we explore several applications that are mainly categorized into two scenarios: graphics-heavy contents and graphics-light contents based on the v-sync (v-blank) limit (60fps). Our experiments were examined using the *ZeBu* emulator [6] (1.6 GHz CA15 quad cores, 400MHz Rogue GPU, 2 GB RAM). We choose one of the most popular GPU benchmarks called *gfxbench* [10] and one 3D game called *asphalt 8* [9] as representative graphics-heavy applications. Suppose that the original sequence is running at 40 fps and the frame rate is increased to 60 fps by these different methods. To evaluate performance, we first implement our methods and previous studies as well as naïve bilinear

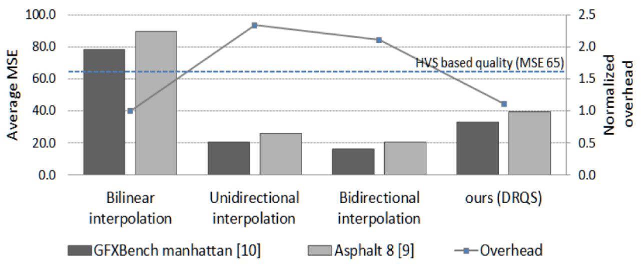

Fig. 6 Our results of performance analysis and computational overhead for graphics-heavy contents

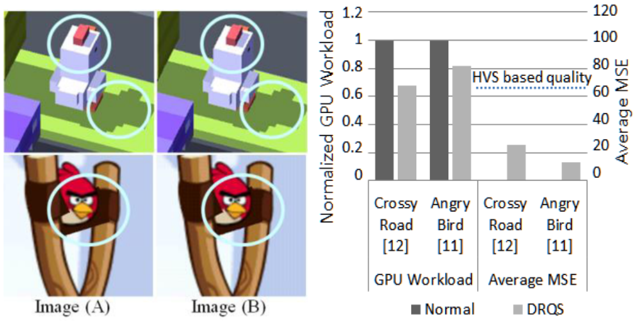

Fig. 8 Comparison of the golden image and ours: (A) golden image, (B) DRQS results (left) and experimental results of quality loss and GPU workload reduction (right) for graphics-light contents.

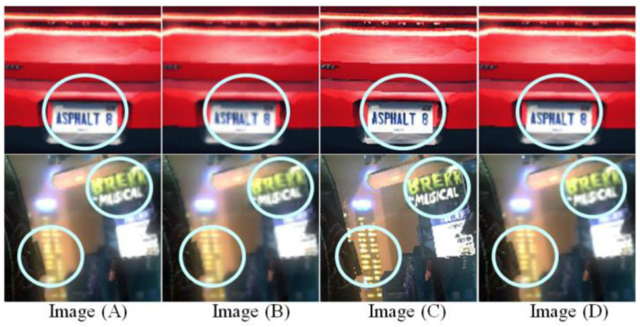

Fig. 7 Comparison of the golden image and ours: (A) golden image, (B) bilinear upscaling, (C) sharpen rendering, and (D) final DRQS results.

interpolation.

For the most recent interpolation methods to obtain pixel-accurate results, one is based on the unidirectional re-projection by the image warping technique and the other is based bidirectional re-projection technique. Figure 6 shows the experimental results of measuring the average quality in MSE and the computational cost. Since both applications are a fill-bounded scene consisting of moving characters and a camera with complex fragment shading operations, we found that the results from both exhibit a similar pattern. In terms of the quality, both previous studies achieved relatively good results with increasing frame rates. However, both algorithms show 2x the high computational overhead, which directly causes to power consumption, compared to the naïve bilinear interpolation. Thus, they are not satisfactory for mobile devices. In contrast, our approach has a significant advantage in terms of additional costs. According to our preliminary comparisons, our approach requires negligible overhead but the other methods use up to quad CA15 cores in our experiments. Nevertheless, ours maintains the HVS based high visual quality [8].

In Fig. 7, we compare and show the closed-ups images of *gfxbench* [10] (lower row) and *Asphalt 8* [9] (upper row) to enhance the spatial resolution by applying the proposed quality scaling algorithm. To compensate for the blurring artifact as shown in Image (B) due to the low resolution rendering, Images (B) and (C) show the difference before and after the proposed sharpen rendering is applied. As a result of the proposed quality scaling, we achieve the final output, Image (D). For graphics-light applications, we chose two popular games running at 60fps: *angry birds* [11] with an image changed little over time and *crossy road* [12] where there are strong changes. Figure 8 shows the experimental results of quality loss and GPU workload reduction. For more dynamic scenes, our algorithm is more robust in reducing the workload and naturally saves more power. As a trade-off between power efficiency and quality, our results prove that it always guarantees HVS based visual quality [8] even though there is some quality loss.

4. Conclusion

In this paper, we have demonstrated a novel technique of combining the Dynamic Rendering algorithm with the Quality Scaling algorithm to enhance the spatial and temporal resolution of very practical scenarios. Our approach has many advantages. First, unlike the recent frame rate up conversion algorithms, our proposed solution does not require high computational costs to naturally increase the frame rate, while satisfying the HVS based visual quality and 60 fps. Second, as increasing the importance of reducing the power consumption, reducing the GPU workload is very crucial, so for graphics-light contents, our solution reduces the workload by up to 32% in GPU-based rendering with acceptable quality loss based on HVS. Thus, our proposed solution is well suited for mobile devices, meaning that it provides a better quality of service within the limited power budget.

References

[1] H. Bowles, K. Mitchell, R.W. Sumner, J. Moore, and M. Gross, "Iterative image warping," Computer graphics forum, vol.31, no.2, pp.237–246. May 2012.
[2] L. Yang, Y.-C. Tse, P.V. Sander, J. Lawrence, D. Nehab, H. Hoppe, and C.L. Wilkins, "Image-based bidirectional scene reprojection," ACM Transactions on Graphics, vol.30, no.6, p.150, Dec. 2011.
[3] Y. Tokuyoshi, "Specular lobe aware upsampling based on spherical Gaussians," ACM SIGGRAPH 2013 Posters, p.107, July 2013.
[4] P. Didyk, E. Eisemann, T. Ritschel, K. Myszkowski, and H.-P. Seidel, "Perceptually-motivated Real-time Temporal Upsampling of 3D Content for High-refresh-rate Displays," Computer Graphics Forum, vol.29, no.2, pp.713–722, May 2010.
[5] R. Herzog, E. Eisemann, K. Myszkowski, and H.-P. Seidel, "Spa-

tio-temporal upsampling on the GPU," ACM SIGGRAPH on Interactive 3D Graphics and Games, pp.91–98, Feb. 2010.
[6] "The Synopsys suite of ZeBu emulation solutions," http://www.synopsys.com/Tools/Verification/hardware-verification/emulation/Pages/default.aspx, accessed Feb. 2015.
[7] T. McReynolds and D. Blythe, Advanced graphics programming using OpenGL, Elsevier, 2005.
[8] E. Cerqueira, M. Curado, and M. Leszczuk, Future Multimedia Networking, Springer-Verlag Berlin, 2010.
[9] "Asphalt 8: Airborne HD," http://www.gameloft.com/android-games/asphalt-8-free, Gameloft, accessed Feb. 2015.
[10] "GFXBenchmarks," https://gfxbench.com, Kishonti, accessed Feb. 2015.
[11] "Angry Birds," https://www.angrybirds.com, Rovio, accessed Feb. 2015.
[12] "Crossy Road," www.crossyroad.com, Yodo1, accessed Feb. 2015.