UDC 004.92
DOI: https://doi.org/10.17721/1812-5409.2025/1.19

Rostyslav PIKULSKY, PhD Student
ORCID ID: 0000-0001-8973-8084
e-mail: rostyslav.pikulsky@gmail.com
Taras Shevchenko National University of Kyiv, Kyiv, Ukraine

DEFERRED RENDERING METHODOLOGIES ANALYSIS AND COMPARISON

Real-time photorealistic rendering has always been one of the long-standing goals in computer graphics. In particular, this objective often goes down to low-latency photorealistic two-dimensional image generation based on analytic three-dimensional environment description and spectator location and view parameters using a typical personal computer with consumer graphics hardware with relatively limited computing capability. The fundamental problems one has to solve to perform rendering are main view visibility determination and shading, i. e. finding the environment surface element that is observed in the particular pixel of the resulting image and calculating the amount of light this surface element reflects in the observer's direction for all image's pixels. Due to the computation capability constraints of the target hardware, it is crucial to perform these operations as efficiently as possible.

This work is aimed at analysis and experimental comparison of the modern methodologies used to tackle main view visibility determination and shading tasks. Due to modern lighting calculation methods' complexity and high geometric detail of the virtual environments, forward rendering has become impractical as a general-purpose rendering approach and only remains in use when specific geometry or material types are involved. Deferred rendering with G-buffer generation has become a state-of-the-art solution for products demanding high visual quality and fidelity. G-buffer generation can be performed in different ways including rasterized G-buffer generation and visibility buffer method. A theoretical overview and comparison of these techniques are presented in this work. Also, we use a demonstration application we implemented to perform an experimental comparison of G-buffer generation techniques in various conditions. Our experimental results can be used as a guidance when designing production rendering solutions.

K e y w o r d s : ***computer graphics, real-time rendering, forward rendering, deferred rendering, G-buffer, visibility buffer.***

AMS 2020 classification: 68U05.

Introduction

Computer graphics is an important area of computer science nowadays with rendering being one of the key objectives. Rendering (Pharr, Jakob, & Humphreys, 2016) in computer graphics generates an image of the analytically described environment viewed by a certain observer with a given location, orientation, and view properties. It allows computer users to observe virtual visual worlds. It is thus an integral part of computer simulation and modeling software packages, animated films and cinematic effects production software, computer games, etc. Real-time photorealistic rendering is a particularly crucial and relevant case – it is intended to interactively produce a photorealistic image sequence for the environment and observer that can change over time based on behavior algorithms and user's input, which can provide a real-life-like visual experience to the user. On the other hand, a typical interactive graphics application has to produce frames with a high enough refresh rate (60 frames per second or more) using the consumer graphics adapter in a personal computer, smartphone, or other device the application runs on. Since the demand for higher visual fidelity keeps increasing and the computational capabilities and memory size of graphics processors are limited, real-time photorealistic rendering techniques and algorithms remain relevant topics for further research aimed at minimizing the rendering software packages latency (time to build one frame) and the amount of memory required for their operation.

Different rendering approaches have been used for interactive rendering. A forward rendering method was one of the first and simplest methods used. Later, deferred rendering (Deering et al., 1988) became an important rendering technique replacing forward rendering in most scenarios (Pharr, & Fernando, 2005, chap. 9; Karis, 2013). The idea behind it is to generate a G-buffer – a set of large screen images containing the visible surface elements data (position, normal, material properties, etc.), which are then used to calculate lighting by a separate deferred procedure. G-buffer generation is typically much faster than complete lighting calculation, so this scheme provides significant speedup to the rendering algorithm, especially when using complex lighting models and processing scenes with complex geometry. However, due to the increasing complexity of virtual environments, the G-buffer generation stage can still pose a performance issue so it is rather important to perform it most efficiently.

This article provides an overview of the forward and deferred rendering methodologies and compares their advantages and disadvantages. Also, we perform an experimental comparison of two common G-buffer generation algorithms which are rasterized G-buffer and visibility buffer generation, and analyze the applicability of both techniques in different scenarios. These testing results can be used as a guidance when designing production rendering solutions.

1. Forward and deferred rendering

In general, interactive rendering intends continuous image (frame) generation of a given dynamic environment from a certain dynamic observer's point of view. The frame generation process can be separated into two fundamental stages: visibility determination and shading. Visibility determination is finding the environment surface element observed in the particular pixel of the resulting frame. Shading means calculating the amount of light this surface element reflects in the observer's direction. Rendering a frame is essentially solving visibility determination and shading tasks for every pixel of the resulting frame. It should be noted that while both of these problems are important, visibility determination is a primary one since it is impossible to perform any shading and generate a resulting image if visible surface elements are unknown. On the other hand, shading itself can be simplified at the cost of the photorealism and visual quality of the rendering solution. Due to this, initial graphics hardware with three-dimensional graphics acceleration implemented robust visibility determination in the first place while shading solutions were gradually evolving with graphics cards' hardware capabilities.

Graphics processing units use rasterizers to tackle visibility determination problem. Rasterizers are fixed-function hardware blocks that take the two-dimensional triangle defined in image space as input and output a set of pixels of the image

the input triangle covers. To use these blocks, graphics processors implement a graphics pipeline, which consists of three common stages:

- Vertex processing stage: it takes geometry data as input (for example, three-dimensional environment geometry) and transforms it into two-dimensional geometry defined in the image space.
- Rasterization stage: it uses hardware rasterizers to determine a set of covered image pixels based on the vertex processing stage's output.
- Pixel processing stage: it performs a certain program for every resulting pixel of the rasterization stage which must output some value to the output image or discard the processing. To avoid occluded triangle color output, the graphics pipeline can maintain a minimum camera-surface distance image (depth buffer, Z-buffer) to discard or overwrite existing results based on distance comparison.

Forward rendering is a rendering method that uses a graphics pipeline to rasterize all the environment geometry and performs shading in the pixel processing stage in these pipelines. This rendering method was implemented in the first non-programmable three-dimensional graphics accelerators and remains relevant for specific rendering scenarios. Its most important advantages are simplicity, wide hardware support (it only requires a basic graphics pipeline to be performed), and low memory consumption, as only base output and depth buffers are necessary.

The rise of the graphics hardware computing power and the expansion of its technical capabilities opened possibilities to increase the geometry scale and complexity and switch to advanced lighting models, which drastically improved the visual quality of virtual environments (Burley, 2012; Walter et al., 2007). However, the forward rendering method turned out to be ineffective in these conditions because new lighting models required a significantly larger amount of calculations. One reason for this is triangle overlap which reduces pixel processing efficiency as it often leads to duplicate pixel processing in the overlap area when triangles are processed in the order from the farthest to the closest to the observer. This phenomenon becomes common when using the pixel processing stage to perform computation-intensive lighting models and rendering complex geometry with a severe amount of triangle overlap, resulting in a crucial performance penalty. Another reason for performance degradation is the unbalanced workload of the rasterizers and multiprocessors that perform computationally heavy lighting calculations, which reduces general graphics pipeline processing throughput. The deferred rendering method (Deering et al., 1988; Liktor, & Dachsbacher, 2012; Mara, McGuire, & Luebke, 2013) was developed to tackle these problems. Instead of immediately calculating the lighting during geometry rasterization, this method uses a graphics pipeline pixel processing stage to generate a G-buffer – a set of textures that match the main screen in size and contain visible surface element parameters corresponding to a particular pixel. After G-buffer generation, the lighting is calculated once for every pixel by a separate procedure that uses G-buffer data as input. Example G-buffer images are displayed in Figure 1; they preserve typical surface parameters including the color of the surface element, the surface normal, the type of light reflection model, etc. Determining these values is usually significantly cheaper than lighting calculation in terms of operations to be performed. Hence, the triangle overlap issue's influence on performance reduces drastically. Another key advantage of this deferred shading method is compatibility with advanced shading algorithms that are impossible to implement in a pixel processing stage. The main disadvantage of deferred shading is the increased memory consumption required for G-buffer maintenance and performance overhead produced by writing data to G-buffer textures and reading from it. These may become notable performance issues in case many different surface parameters are required for shading. G-buffer management also causes problems with anti-aliasing support for deferred rendering due to severe memory consumption and processing requirement increase. Various techniques were developed to decrease the negative effect of this issue (Kerzner, & Salvi, 2014; Crassin et al., 2016).

Recently, the method of generating a G-buffer using a visibility buffer (V-buffer) has become popular (Burns, & Hunt, 2013). This method addresses the shortcomings of the classical rasterized G-buffer deferred rendering, which poses a performance issue when using a G-buffer to store an extensive amount of data. It narrows the data written during the pixel processing stage when rasterizing environment geometry to the minimal identifier of the rasterized triangle – the resulting buffer containing these identifiers is called visibility buffer. This significantly speeds up the geometry rasterization stage since such identifiers need a small amount of memory for storage and usually do not require any calculations in the pixel processing stage to be obtained. The G-buffer generation is performed by a separate procedure that takes the visibility buffer as an input and uses environment data to reconstruct the visible surface element for a particular pixel and calculate necessary parameters. To implement full simultaneous access to all the environment data required, modern graphics processor programming model support is necessary which slightly limits the applicability of the visibility buffer method for the old hardware. Surface element reconstruction requires software repeat of the operations performed at the rasterization stage (for example, transforming the vertices of the triangle, determining the point of its intersection with the corresponding pixel's sight ray, and others), which are not necessary when using rasterized G-buffer method. Therefore, the visibility buffer approach efficiency may be lower on a small amount of geometry that does not cause significant triangle overlap during rasterization, or simple lighting models that do not require a large number of geometric parameters. Hence, choosing the optimal G-buffer generation technique is a subtle compromise that needs careful research and testing.

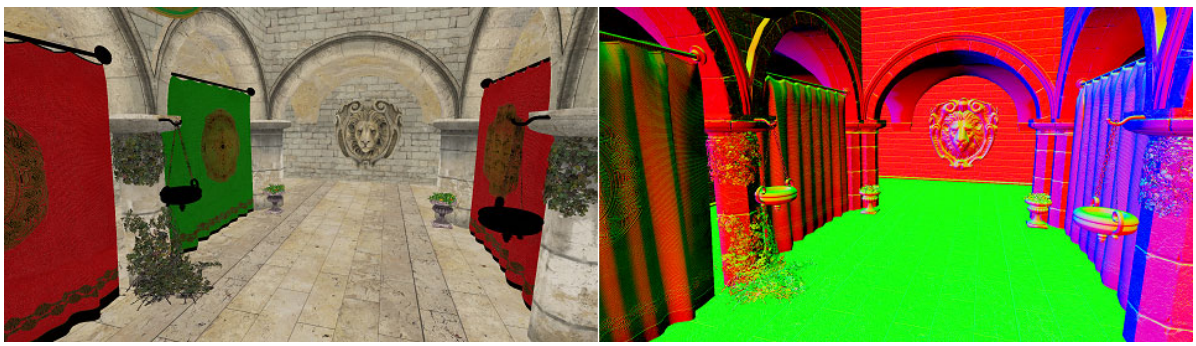

Fig. 1. G-buffer example. The left image contains diffuse color, right image preserves the surface normal (each of the 3D normal's components is encoded in one of the image's color channels)

2. Testing and comparison

We implemented a demonstration application to perform an experimental comparison of two G-buffer generation methods described above. To achieve decent analysis variability, we performed testing with different options for the following parameters:

- Frame resolution: two common screen resolutions were used: 1920x1080 pixels (FullHD) and 2560x1440 (2K).
- Geometry amount: testing was performed using several common demonstration scenes containing different numbers of triangles: Cornell Box (36 triangles), Sponza (262267 triangles), and Bistro (2829801 triangles).
- G-buffer memory size: to simulate testing of different lighting models, three G-buffer layouts with common surface element parameters were implemented, each requiring 16, 32, and 64 bytes per pixel respectively.

The testing was performed on a personal computer with an NVidia RTX 2080 Super graphics card. Results are shown in Tab. 1 and 2. Each table corresponds to the screen resolution specified in the description; the table rows contain the time measurements for the specified scene, while the columns contain the measurements for the rendering using the specified G-buffer generation method with the G-buffer of a certain size. For example, the column labeled "Raster, 16 bpp" corresponds to rasterized G-buffer deferred rendering with a G-buffer size of 16 bytes per pixel, and the column "V-buffer, 64 bpp" corresponds to visibility buffer deferred rendering with a G-buffer size of 64 bytes per pixel. All time measurements are given in milliseconds.

Table 1

Time measurements for 1920x1080 resolution

Time, ms	Raster, 16bpp	V-buffer, 16bpp	Raster, 32bpp	V-buffer, 32bpp	Raster, 64bpp	V-buffer, 64bpp
Cornell box	0.10	0.31	0.17	0.35	0.28	0.41
Sponza	0.35	0.37	0.58	0.48	0.89	0.54
Bistro	0.61	0.62	0.99	0.67	1.74	0.71

Table 2

Time measurements for 2560x1440 resolution

Time, ms	Raster, 16bpp	V-buffer, 16bpp	Raster, 32bpp	V-buffer, 32bpp	Raster, 64bpp	V-buffer, 64bpp
Cornell box	0.21	0.54	0.35	0.62	0.52	0.75
Sponza	0.63	0.67	1.07	0.76	1.53	0.88
Bistro	1.10	1.03	1.78	1.16	3.42	1.24

As can be observed in the testing results, the visibility buffer turns out to be a powerful optimization when rendering complex environments with high geometry detail and using advanced lighting models that require large G-buffers. However, when dealing with smaller environment geometry amounts, classical deferred shading noticeably outperforms the visibility buffer approach due to surface element reconstruction calculations needed by the visibility buffer method which adds an unavoidable fixed performance price to it. Visibility buffer approach relative performance also tends to slightly improve with increasing screen resolution, because shortcomings of classic deferred rendering become more apparent when processing a larger number of pixels.

Discussion and conclusions

In this work, common rendering methodologies were reviewed and compared, including forward rendering and deferred rendering with its G-buffer generation algorithm variations – rasterized G-buffer and visibility buffer approaches. In addition to the theoretical discussion, a demonstration application was developed and an experimental comparison of G-buffer generation methods was performed in different data and working conditions. Experiments show that the rasterized G-buffer method is more efficient for simple graphics libraries designed to process simple visual environments, while the visibility buffer turns out to be superior when rendering complex scenes using advanced lighting models. Based on the testing data, the relative performance of different G-buffer generation approaches can be predicted for other rendering applications.

References

Burley, B. (2012). *Physically-based shading at Disney.* Disney. https://media.disneyanimation.com/uploads/production/publication_asset/48/asset/s2012_pbs_disney_brdf_notes_v3.pdf

Burns, C. A., & Hunt, W. A. (2013). The Visibility Buffer: A Cache-Friendly Approach to Deferred Shading. *Journal of Computer Graphics Techniques*, 2(2), 55–69. http://jcgt.org/published/0002/02/04/

Crassin, C., McGuire, M., Fatahalian, K., & Lefohn, A. (2016). Aggregate G-Buffer Anti-Aliasing. *IEEE Transactions on Visualization and Computer Graphics*, 22(10), 2215–2228. https://doi.org/10.1109/TVCG.2016.2586073

Deering, M., Winner, S., Schediwy, B., Duffy, C., & Hunt, N. (1988). The triangle processor and normal vector shader: a VLSI system for high performance graphics. *SIGGRAPH Computer Graphics*, 22(4), 21–30. https://doi.org/10.1145/378456.378468

Karis, B. (2013). *Real Shading in Unreal Engine 4.* Epic Games. https://cdn2.unrealengine.com/Resources/files/2013SiggraphPresentationsNotes-26915738.pdf

Kerzner, E., & Salvi, M. (2014). Streaming G-Buffer compression for multi-sample anti-aliasing. In *HPG '14: Proceedings of High Performance Graphics* (pp. 1–7). Eurographics Association.

Liktor, G., & Dachsbacher, C. (2012). Decoupled deferred shading for hardware rasterization. In Spencer, S. N. (Ed.), *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (pp. 143–150). Association for Computing Machinery. https://doi.org/10.1145/2159616.2159640

Mara, M., McGuire, M., & Luebke, D. (2013). *Lighting Deep G-Buffers: A single-pass, layered depth images with minimum separation applied to indirect illumination.* NVIDIA Corporation. https://research.nvidia.com/sites/default/files/pubs/2013-12_Lighting-Deep-G-Buffers/Mara2013DeepGBuffer.pdf

Pharr, M., & Fernando, R. (2005). *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems).* Addison-Wesley Professional.

Pharr, M., Jakob, W., & Humphreys, G. (2016). *Physically Based Rendering: From Theory to Implementation* (3rd ed.). Morgan Kaufmann Publishers Inc.

Walter, B., Marschner, S. R., Li, H., & Torrance, K. E. (2007). Microfacet models for refraction through rough surfaces. In J. Kautz, & S. Pattanaik (Eds.), *Proceedings of the 18th Eurographics Conference on Rendering Techniques* (pp. 195–206). Eurographics Association.

Отримано редакцією журналу / Received: 06.12.24
Прорецензовано / Revised: 07.03.25
Схвалено до друку / Accepted: 13.03.25

Ростислав ПІКУЛЬСЬКИЙ, асп.
ORCID ID: 0000-0001-8973-8084
e-mail: rostyslav.pikulsky@gmail.com
Київський національний університет імені Тараса Шевченка, Київ, Україна

АНАЛІЗ І ПОРІВНЯННЯ МЕТОДОЛОГІЙ ВІДКЛАДЕНОГО РЕНДЕРИНГУ

Фотореалістичний рендеринг у реальному часі є однією з основних проблем у сфері комп'ютерної графіки. На практиці вона часто зводиться до генерації анімованої послідовності фотореалістичних зображень з низькою затримкою на основі аналітичного опису тривимірного віртуального середовища та параметрів спостерігача з використанням типового персонального комп'ютера або іншого пристрою з графічним обладнанням, що має відносно обмежені обчислювальні можливості. Фундаментальними задачами, які необхідно розв'язати для виконання рендерингу, є визначення видимості головного виду та затінення для кожного пікселя зображення, тобто пошук елемента поверхні середовища, що спостерігаємо в конкретному пікселі результуючого зображення та обчислення кількості світла, яке цей елемент поверхні віддзеркалює в напрямку спостерігача. Через обмежені обчислювальні можливості цільового обладнання вкрай важливо виконувати ці операції якомога ефективніше.

Пропоновану роботу присвячено аналізу й експериментальному порівнянню сучасних методологій, які використовують для розв'язання завдань визначення видимості та затінення головного виду. Через складність сучасних методів розрахунку освітлення та високу геометричну деталізацію віртуальних середовищ прямий рендеринг поступово втратив практичність як загальновживаний підхід до рендерингу та нині використовується лише в разі необхідності оброблення специфічних типів геометрії та матеріалів. Відкладений рендеринг з використанням G-буфера став найактуальнішим рішенням для графічних додатків, що потребують високої візуальної якості та точності. Генерація G-буфера може бути виконана різними способами, такими як метод растеризації G-буфера та метод буфера видимості. У дослідженні представлено також теоретичний огляд і порівняння цих методик. Крім того, розроблено демонстраційну програму та застосовано її для здійснення експериментального порівняння методів генерації G-буфера в різних умовах. Результати експериментів можна використовувати для прогнозування ефективності зазначених методів під час розроблення бібліотек рендерингу.

К л ю ч о в і с л о в а : ***комп'ютерна графіка, рендеринг у реальному часі, прямий рендеринг, відкладений рендеринг, G-буфер, буфер видимості.***

Автор заявляє про відсутність конфлікту інтересів. Спонсори не брали участі в розробленні дослідження; у зборі, аналізі чи інтерпретації даних; у написанні рукопису; в рішенні про публікацію результатів.

The author declares no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses or interpretation of data; in the writing of the manuscript; in the decision to publish the results.