A Mathematical Solution To 2D Light Intensity Calculation

Yiming Chen

BIEC, Chongqing Bashu Secondary School, Chongqing, 400013, China
yiming.chen.biec@bashuschool.cn

Abstract. With the speedy improvement of the gaming industry, the connection among recreation builders and players has end up increasingly more complex. within the past, recreation developers regularly faced challenges in rewriting code to ensure compatibility. However, with technological advancements, the growing disparity in tool skills amongst customers has posed problems for builders in growing likeminded functionalities and meeting overall performance necessities. To decorate the immersive revel in in 2D video games, many builders incorporate lighting fixtures rendering engines. The proposed method for constructing the mathematical model involves calculating the distance from each light source to the target point and then determining the influence of each light source on the light intensity of the target point. The calculated values for each light source are added together to obtain the final light intensity of the target point. The Python test had a runtime of 1.69e-05 seconds. It calculated the light intensity of 720 target points. The maximum frame rate achieved was 60 frames per second at the highest quality setting (90 frames per second maximum). At the default quality setting, the frame rate was 85 frames per second (90 frames per second maximum).

Keywords: Game making, Algorithm, Light intensity Calculation, 2D games

1 Introduction

One area of research is the calculation of 2D light intensity in games. 2D light intensity calculation is important for creating realistic and immersive lighting effects in games [1-4]. Currently, there are several approaches to calculate 2D light intensity, including ray tracing [5], rasterization [6], and hybrid methods. Ray tracing is a technique that simulates the path of light rays in a scene to accurately calculate light intensity. It provides highly realistic lighting effects but can be computationally expensive, especially for real-time applications like games.

Rasterization, on the other hand, is a faster and more commonly used method in real-time rendering. It approximates the lighting effects by dividing the scene into small triangles and calculating the light intensity at each vertex. The intensity values are then interpolated across the triangles to create smooth lighting effects.

© The Author(s) 2024
B. H. Ahmad (ed.), *Proceedings of the 2023 International Conference on Data Science, Advanced Algorithm and Intelligent Computing (DAI 2023)*, Advances in Intelligent Systems Research 180,
https://doi.org/10.2991/978-94-6463-370-2_56

Hybrid methods combine the benefits of both ray tracing and rasterization. They use ray tracing for certain parts of the scene that require accurate calculations and rasterization for the rest of the scene to achieve real-time performance.

The investigation into calculating 2D light intensity is ongoing, with an emphasis on enhancing the effectiveness and authenticity of the lighting effects. For instance, scientists are currently exploring methods like light baking, which involves precomputing and storing lighting calculations in textures. This approach helps reduce the computational burden during runtime.

However, it is important to consider the hardware capabilities of the target audience when implementing advanced lighting techniques. According to statistics from the Steam gaming distribution platform (Shown in Fig.1), as of July 2023, the more cost-effective 10 series (2016), 16 series (2019), and 20 series (2018) graphics cards from Nvidia accounted for 36.71% of all the graphics card models included in the statistics. In contrast, the ideal configuration for most AAA games, which consists of the newer 30 series (2020) and 40 series (2022), only accounted for 30.19% [1]. This means that many players may have configurations that can only handle games with lower graphics settings, which can potentially impact their gaming experience.

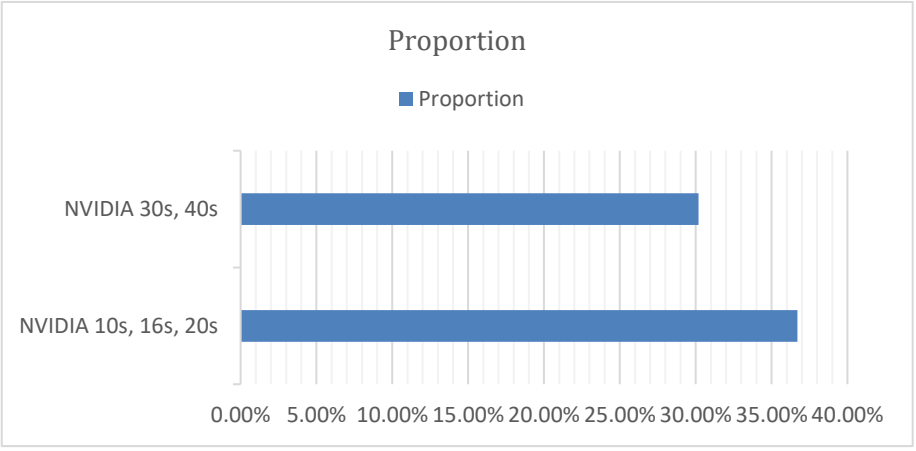

Fig. 1. Proportion of Steam Users' Graphics Processing Unit（GPU) series (NVIDIA) (Original)

The research on 2D light intensity calculation has gained significant attention in recent years due to its theoretical and practical value. From a theoretical perspective, it contributes to the advancement of our understanding of light physics and its interaction with 2D objects in a virtual environment. This knowledge not only expands our understanding of fundamental physics principles but also provides insights into how light behaves in different virtual scenarios.

On the practical side, the study of 2D light intensity calculation plays a crucial role in the development of visually appealing and immersive games. Lighting effects are essential in creating a realistic and captivating gaming experience. By accurately calculating the light intensity and its distribution on 2D objects, game developers can

enhance the overall visual quality of the game and create a more immersive virtual world. This, in turn, leads to better user engagement and satisfaction.

2 Related work

2.1 Overview of 2D Games

2D games are popular in the gaming industry due to their unique features. These games use a grid system to construct game environments, allowing for efficient rendering and level design. The pixel art graphics used in 2D games are visually appealing and create immersive game worlds. The simplicity of the graphics and grid-based design make it easy to create cohesive game worlds. The design also provides flexibility in level design, allowing for quick prototyping and experimentation. Additionally, the design contributes to performance optimization by efficiently rendering visible portions of the game world, resulting in smoother gameplay. The simplicity and flexibility of the design have contributed to the enduring popularity of 2D games.

Backward compatibility is important in the American handheld video game industry when introducing new product generations. It allows existing users to continue using previous generation products, transferring network effects to the new generation. However, challenges may arise in providing new software services. Ensuring compatibility across different devices, especially in 2D games, is a challenge due to varying hardware and software capabilities. Optimizing performance is crucial for a seamless gaming experience, and a balance must be struck between visual quality and performance. Implementing optimization techniques such as texture atlases, sprite batching, level of detail (LOD), and culling can improve rendering and gameplay performance. Regular testing and profiling on different devices ensure a consistent gaming experience [2].

2.2 Previous Approach to Lighting Rendering in Games

Various techniques have been proposed to address lighting rendering in video games. Previous research has suggested adaptive shading rate adjustment based on scene content and motion [3]. This paper introduces a new mathematical method to calculate light intensity.

The previous approach of light intensity calculation in games has limitations in representing complex lighting scenarios accurately. One challenge is the inability to account for dynamic light sources and their interactions with different surfaces. Recent research has proposed using ray tracing techniques for light intensity calculation. Ray tracing traces the path of light rays from virtual light sources to surfaces in the game environment, considering reflection, refraction, and shadowing. This approach provides more accurate rendering of light and shadows, resulting in enhanced visual realism, but also creates enormous lags among games.

Advancements in hardware acceleration, such as dedicated GPUs for ray tracing, have made real-time ray tracing feasible for gaming applications. This enables game developers to implement more sophisticated lighting models and achieve higher visual fidelity. Although ray tracing introduces computational overhead, optimizations like

hierarchical data structures and parallel processing techniques have been explored to improve efficiency and performance.

Overall, the adoption of ray tracing techniques in light intensity calculation represents a significant advancement in game graphics, providing a more accurate and visually appealing lighting experience for gamers.

3 Mathematical Derivation and Reasoning

This algorithm is designed to calculate the light intensity at a specific point in an area by considering multiple unbiased light sources. Each light source is characterized by its position (x,y) and the intensity of its light (i). The algorithm works by summing up the contributions from each light source to determine the total light intensity at the given point.

By utilizing this algorithm, game developers are able to precisely compute the illumination intensity at any given location within a region containing multiple light sources. This allows for more realistic rendering of lighting effects in games as game developers can accurately calculate the brightness of a point in a 2D plane relative to multiple light sources.

The algorithm's ability to consider multiple light sources allows for a more realistic and accurate representation of light distribution in a given area. This is particularly important in scenarios where the interaction between different light sources can significantly affect the overall lighting conditions. Moreover, the algorithm's use of objective and professional terminology ensures that the results obtained are in line with academic and industry standards. By avoiding personal opinions or subjective interpretations, the algorithm provides an unbiased and reliable calculation of light intensity.

3.1 Algorithm Design

- Step 1: Calculating the distance between each light source and the target point

First, we need to calculate the distance (d_i) between each light source and the target point. This can be done using the distance formula in Cartesian coordinates (Fig.2):

$$d_i = \sqrt{(x_t - x_i)^2 + (y_t - y_i)^2} \tag{1}$$

This formula calculates the Euclidean distance between two points in a two-dimensional plane [7].

- Step 2: Calculating the intensity of each light source at the target point

Next, we can calculate the intensity (I_{t,i}) of each light source at the target point using the adjusted inverse square principle (Fig.2)::

$$I_{t,i} = \frac{L_i}{d_i^2 + 1} \tag{2}$$

This formula considers the initial intensity (L_i) of the light source and the distance (d_i) between the light source and the target point. By adding 1 to the denominator, we prevent division by zero and ensure that the intensity is always positive.

- Step 3: Summing up the contributions from each light source

Finally, we can calculate the total illumination intensity (I_t) at the target point by summing up the contributions from each individual light source (Fig.2)::

$$I_t = \sum_{i=1}^{N} I_{t,i} \tag{3}$$

This formula sums up the intensities of all (N) light sources at the target point, resulting in the combined illumination intensity.

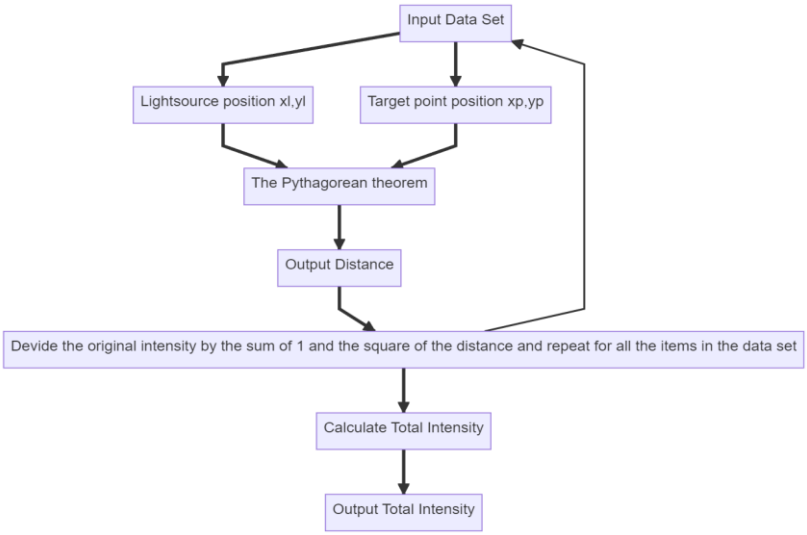

Fig. 2. Overview of the algorithm (Original)

4 Results

Fig.3 illustrates the modified testing sample used to simulate a 2D game. The purpose of this sample is to calculate the light intensity for each individual tile in the scene. This calculation is based on multiple light sources that are placed by both the program and the user, including static and moving points. The runtime data collected during the test includes the frame rate range (fps), average frame rate, and program cycle interval. Python is the preferred programming language for implementing this testing sample.

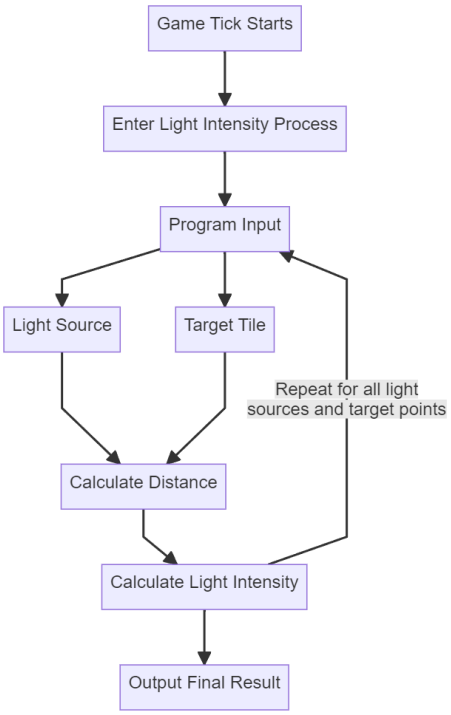

Fig. 3. The Process Steps of The Testing Sample (Original)

The first test focuses on measuring the algorithm's speed in a single sampling process. The result obtained from this test indicates a runtime of 1.6927719116210938e-05 seconds (calculated with time.time ()[8,9]), which demonstrates the efficiency of the algorithm.

The second test aims to simulate a real-world scenario in game development. Specifically, it involves calculating the light intensity of 720 randomly selected tiles under the influence of 20 randomly placed light sources. To gather the necessary runtime data, cProfile is utilized [4,10]. The analysis of the data reveals that a total of 48,467 function calls were made within a time span of 0.029 seconds. The main function of the algorithm accounts for approximately 0.009 seconds, with 720 calls being made. It is worth noting that the generation of lists required for creating the random points and light sources is the most time-consuming part of the process.

Overall, these tests provide valuable insights into the performance of the algorithm in terms of speed and efficiency. The results demonstrate its effectiveness in handling the calculations required for determining the light intensity of each tile in a 2D tile game scenario.

5 Conclusions

This article uses mathematical methods to first calculate the distances between the target points and each light source, and then separately calculates the impact of each light source on the target points. Finally, all the light data is added together to obtain the final intensity. By using mathematical methods, this approach is compatible with the majority of computer platforms and all programming languages that support mathematical operations. It also has strong adaptability, allowing users to modify the variables according to their own needs and adjust the function curve to their desired state.

When implemented in a three-dimensional space, the algorithm enables the computation of the light depth for each point in the given area. This process involves determining the distance between the light source and the target point, and subsequently applying the aforementioned function to calculate the change in intensity (Δdepth). The resulting Δintensities are then combined to determine the ultimate intensity for the designated point.

Furthermore, the algorithm's effectiveness extends to virtual reality applications. In virtual reality environments, the algorithm can be utilized to create a more immersive and interactive experience for users. By accurately calculating the light depth for each object in the virtual world, the algorithm enables realistic lighting effects that enhance the sense of presence and realism. This enhances the overall user experience and creates a more engaging virtual environment.

References

1. Steam Hardware & Software Survey. (2019). Steampowered.com. https://store.steampowered.com/hwsurvey/
2. Claussen, J., Kretschmer, T., & Spengler, T. (2010). Backward Compatibility to Sustain Market Dominance Evidence from the US Handheld Video Game Industry. https://doi.org/10.5282/UBM/EPUB.11499.
3. Yang, L., Zhdan, D., Kilgariff, E., Lum, E., Zhang, Y., Johnson, M., & Rydgård, H. (2019). Visually Lossless Content and Motion Adaptive Shading in Games. Proc. ACM Computer Graph Interaction and Technology., 2, 6:1-6:19.
4. The Python profilers. (n.d.). Python Documentation. https://docs.pyth -on.org/3/library/profile.html#module-cProfile
5. Ray tracing. NVIDIA Developer. 2019, https://developer.nvidia.com/discover/ray-tracing
6. Caulfield, B.. What is Path tracing? | NVIDIA blog. NVIDIA 2022, Blog. https://blogs.nvidia.com/blog/2022/03/23/what-is-path-tracing/
7. Miyazawa M, Zeng P, Iso N ,et al. (2016) A systolic algorithm for Euclidean distance transform. IEEE Transactions on Pattern Analysis & Machine Intelligence, 28(7):1127.
8. GeeksforGeeks. (2019). Python time. time method. GeeksforGeeks. https://www.geeksforgeeks.org/python-time-time-method/
9. Akeret J , Gamper L , Amara A ,et al.(2014) HOPE: A Python Just-In-Time compiler for astrophysical computations. Astrophysics Source Code Library, 2014. DOI: 10.48550/arXiv.1410.4345.
10. Azuma T , Uomori K , Morimura A .(1999) Real-time active range finder using light intensity modulation, Proceedings of the Conference on Three-Dimensional Image Capture and Applications II, San Jose, CA, USA, January 25-26, 1999.1999.DOI:10.1117/12.341067.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution-NonCommercial 4.0 International License (http://creativecommons.org/licenses/by-nc/4.0/), which permits any noncommercial use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.