Smaller than Pixels: Rendering Millions of Stars in Real-Time

S. Schneegans[1], A. Kreskowski[2], and A. Gerndt[1,3]

[1]University of Bremen, Germany
[2]Bauhaus-Universität Weimar, Germany
[3]German Aerospace Center (DLR)

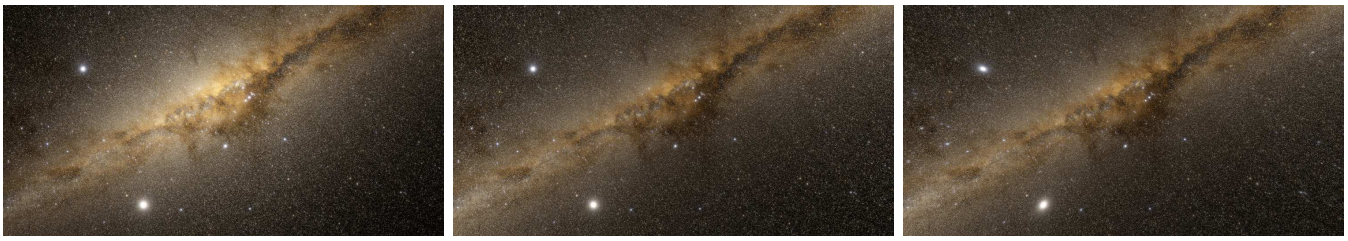

Figure 1: *With naive point-based rendering, stars in the centre of the screen will be too bright (left). To correct this, the projected size of the stars has to be incorporated in the luminance computation (centre). For perspectively correct glaring, the glare must not be computed in screen space, because else the stars will look flat in immersive scenarios (right, observe the now correct ellipsoidal glare around bright stars). For these images, 50 million stars from the Gaia catalogue were rendered by our proposed software rasterizer. About 22 millions stars were inside the frustum. Rendering the right view took about* 8 ms *at 4K resolution on an RTX 4070 Super.*

Abstract

Many applications need to display realistic stars. However, rendering stars with their correct luminance is surprisingly difficult: Usually, stars are so far away from the observer, that they appear smaller than a single pixel. As one can not visualize objects smaller than a pixel, one has to either distribute a star's luminance over an entire pixel or draw some kind of proxy geometry for the star. We also have to consider that pixels at the edge of the screen cover a smaller portion of the observer's field of view than pixels in the centre. Hence, single-pixel stars at the edge of the screen have to be drawn proportionally brighter than those in the centre. This is especially important for virtual-reality or dome renderings, where the field of view is large. In this paper, we compare different rendering techniques for stars and show how to compute their luminance based on the solid angle covered by their geometric proxies. This includes point-based stars, and various types of camera-aligned billboards. In addition, we present a software rasterizer which outperforms these classic rendering techniques in almost all cases. Furthermore, we show how a perception-based glare filter can be used to efficiently distribute a star's luminance to neighbouring pixels. Our implementation is part of the open-source space-visualization software CosmoScout VR.

CCS Concepts

• ***Computing methodologies*** → *Real-time simulation;*

1. Introduction

As instruments for observing the universe have become more powerful, astronomers have collected ever-growing catalogues of stars. There is a substantial interest in visualizing these stars in virtual environments for scientific, educational, and entertainment applications. Also, for software-in-the-loop simulations of space missions realistic star rendering is essential. The approach presented in this paper has been developed for the VaMEx3-RGE project, in which technology is developed for a future mission to Mars where an autonomous robotic swarm will explore the Martian surface. In this project, the presented approach is used in an immersive mission-control tool for mission planning and monitoring. In such applications, the correct rendering of stars is not trivial. The first challenge is to render individual stars with their correct luminance in a perspectively correct manner. If this is done incorrectly, stars will appear with varying brightness depending on their position on the screen, or with an elliptical shape (see *Figure 1*). The second challenge is to rasterize many stars efficiently without causing a bottleneck in the rendering pipeline. The final challenge is proper glaring to make the high dynamic range of stars perceivable.

DOI: 10.2312/egs.20251029

Figure 2: *All eight images of Orion's belt share the same total luminance prior to tone-mapping. Upper row: Single-pixel stars produce aliasing issues when jumping between discrete pixel positions and their mutual luminance differences are difficult to perceive (a). With our software rasterizer, the aliasing is mitigated, but the luminance is still hard to see (b). If discs are used as representation, the visualization gets clearer (c). Larger billboards with an additional glare produce better results but worse performance (d). The bottom row uses the same configurations as above but 20 % of the luminance is contributed by the glare filter presented in* Section 5. *This makes bright stars more easy to spot especially for (a) & (b).*

2. Background and Related Work

In some applications, stars can be visualized as static background images. However, this is not suitable if the user can travel to them. In this case, stars have to be drawn as individual point light sources. Often, they are not drawn as single pixels but as small discs or larger billboards. This improves the visualization as the human visual system is not good at distinguishing the luminance of very small light sources [Bla46]. Adding an artificial glare helps to increase the perceived luminance of overexposed areas [KMS05]. However, this requires rather large billboards and, with many stars in the scene, may result in high overdraw and therefore drastically reduced rendering performance.

Some space visualizations such as Cosmographia [Sem] and Celestia [Cel] employ screen-aligned billboards to represent stars. This approach has the drawback that stars will be circular on the screen which makes them appear more elliptical towards the edge of the screen. Other tools, such as CosmoScout VR [SZGG22] or GaiaSky [SJMS18] draw stars as camera-facing billboards to avoid this issue. However, neither of the authors describe how they compute the brightness of the billboards. This step however is crucial to achieve a correct compositing with other lit objects in the scene.

We use the *Hipparcos* star catalogue and the *Gaia* star catalogue. Together, these catalogues contain more than 1.8 billion stars. Level-of-detail algorithms exist to render such large star catalogues in real-time [SJMS18]. Such algorithms select a subset of stars to render based on their brightness and distance to the camera. In this paper, we do not aim to present a new level-of-detail approach but rather focus on how to correctly and efficiently rasterize many stars at once. Our approach is orthogonal to level-of-detail approaches and can therefore be used in combination with existing level-of-detail algorithms, potentially increasing the number of stars such a system can render. Hence, we will not attempt to render the full Gaia catalogue but demonstrate our methods on a few tens of millions of the brightest stars.

We implemented all techniques in the visualization software CosmoScout VR [SFGG25]. In this software, the luminance of the scene is written to a 32 bit floating point render target. When all scene geometry has been rendered, the average luminance is computed by a series of parallel reductions using compute shaders. It is then used to compute an exposure value according to [PTYG00]. Ultimately, filmic tone mapping [Dui] is used to reduce the dynamic range before displaying the image.

3. The First Challenge: Rendering a Single Star

We can not directly visualize a star as it usually appears smaller than a pixel. Instead, some sort of proxy representation has to be chosen. The smallest possible representation is the pixel that the star appears on. Single-pixel stars promise a good performance but require a glare filter and produce artefacts when jumping between pixel positions. Small anti-aliased camera-aligned discs provide better visual quality but are more expensive to render.

3.1. Star Representations

We implemented point-based stars both using pixel-sized point primitives and using a custom software rasterizer. The latter is described in *Section 4*. Camera-aligned discs were implemented using a geometry shader emitting a quad for each star. A fragment shader then draws a normalized circular gradient to create an anti-aliased disc. We also implemented a variant which produces larger billboards with a built-in glare effect (see *Figure 2*).

The solid angle A of the disc is a configurable parameter. Choosing the right size is difficult as smaller stars are more likely to create flickering but increase the drawing performance. We also observed that stars which are off-screen produce still a significant amount of performance overhead. Using the geometry shader to cull stars outside the frustum can help to reduce this overhead. The performance is evaluated in more detail in *Section 4.2*.

3.2. Luminance and Colour of a Star Representation

For each star from the catalogue, we know its absolute magnitude mag_{abs}, its distance to the camera d in parsecs, and its effective black body temperature T_{eff} in Kelvin. The star's colour is computed based on T_{eff} according to [HH21]. The apparent magnitude $mag_v = mag_{abs} + 5 \cdot log_{10}(0.1 \cdot d)$ describes how bright the star appears from the observer's point of view. We use this to compute the surface brightness S of the star representation using its solid angle A in arcseconds^2. This can be used to compute the luminance L of the star representation in cd/m^2:

$$S = mag_v + 2.5 \cdot log_{10}(A) \tag{1}$$

$$L = 10.8 \cdot 10^4 \cdot 10^{-0.4 \cdot S} \tag{2}$$

For single-pixel stars, the solid angle A of a pixel needs to be computed. As this is smaller at the edge of the screen as opposed to the center, point-based stars are rendered brighter towards the border. From the perspective of the user, this is correct. If discs are used, the solid angle is constant for all stars, but their projected size differs depending on the screen-space position. Hence, points and discs will contribute the same luminance to the render target regardless of their screen-space position.

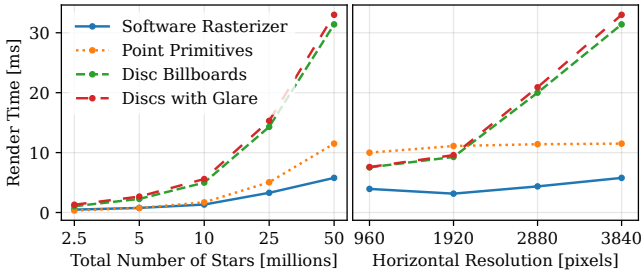

Figure 3: *Star-render times (without glare computation) for a 16:9 view of the Milky Way core. About 25 % of the stars were inside the frustum. The left chart has been recorded at 4k resolution, the right chart with 50 million stars, both on an RTX 4070 Super.*

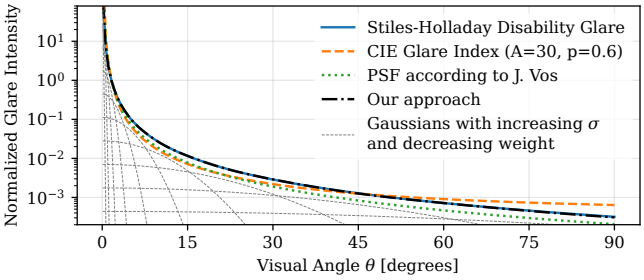

Figure 4: *Our glare function is very similar to empirically deduced glare functions from the literature. It is the sum of several Gaussian kernels with consecutively doubling standard deviations and halving weights. Graphs are normalized to 1 at* $\theta = 2°$.

4. The Second Challenge: Rendering Many Stars

As large star representations are expensive to render, we propose to render the stars as small as possible and add a glare effect in a post-processing step. This way, our approach scales much better with the number of stars in the scene.

4.1. A Software Rasterizer for Stars

As mentioned in *Section 3*, drawing stars as single pixels results in artifacts. Hardware smoothing of points exists, but the implementation is vendor specific, so we cannot know how much luminance a smooth point will actually contribute to the render target. Multisampling is another option, but it is computationally expensive. Because of this and since small primitives tend to stall the GPU pipeline [KSW21], we wrote a software rasterizer for stars.

Our rasterizer uses a compute shader with a spin-lock to atomically update the render target, similar to [GKLR13]. For each star, we upload five floating point values: its 3D position, mag_{abs}, and T_{eff}. Each thread processes one star. First, it computes the screen-space position of the star, and if it is outside the frustum, processing is stopped. Then it evaluates how much luminance would need to be written to a single pixel and distributes this over a 2x2 pixel area, weighted by the distance of the star to the pixel centers. This gives a smooth transition when a star moves across the screen. We pack both the luminance and T_{eff} as half floats into a 32-bit integer texture. The luminance is blended additively, T_{eff} is mixed with the previous value using a weighted average. The mapping from T_{eff} to colour is actually not linear which leads to a small colour shift in areas where many stars overlap. In the future, we plan to investigate packing the colour directly into the 16 bits. Finally, the texture is drawn with a full-screen pass to the main render target, computing the colour based on the mixed temperature value.

4.2. Performance Evaluation

Figure 3 shows render times for different resolutions and various numbers of stars. The software rasterizer scales better with the stars count than native point rendering. It also performs better across all tested resolutions. Interestingly, the performance decreases for small screen resolutions. This is most likely due to an increased number of collisions when writing to a texel of the atomic image.

5. The Final Challenge: Adding a Proper Glare

Experiments have shown that the amount of perceived veiling luminance (or glare) as a function of the angle θ in degrees between the visual axis and a point light source roughly follows $L_{veil}(\theta) \propto 1/\theta^2$ [Vos03]. *Figure 4* graphs several models for this relationship incorporating factors such as age or ocular pigmentation which have been proposed in the literature. As it is not viable to convolve the entire frame with a large 2D kernel, glare is often simulated using multiple 1D Gaussians [KMS05]. In fact, we observed that $1/\theta^2$ specifically can be approximated with a series of weighted Gaussians with exponentially increasing standard deviations. *Equation 3* shows a standard Gaussian with an additional weight factor of $1/\sigma$:

$$f(\theta,\sigma) = \frac{1}{\sigma^2\sqrt{2\pi}} e^{-\frac{\theta^2}{2\sigma^2}} \tag{3}$$

Summing multiple of these weighted Gaussians produces a function which converges to c/θ^2 with c being a constant factor. Even if this is often implemented similarly, we are not aware of any work which has explicitly shown this relationship:

$$\lim_{n\to\infty} \sum_{i=-n}^{n} f\left(\theta, 2^i\right) \sim \frac{1}{\theta^2} \tag{4}$$

This function and the individual Gaussians are graphed in *Figure 4*. The maximum relative error compared to c/θ^2 is less than 1% for $0.2° \leqslant \theta \leqslant 90°$ already for $n = 10$. This emphasizes that simulating glare with a series of weighted Gaussians is a good approximation for the veiling luminance perceived by the human visual system.

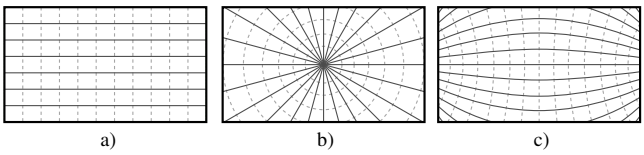

Figure 5: *Usually, Gaussion kernels are decomposed into a horizontal (solid) and a vertical (dashed) pass in screen space (a). Decomposing into a radial and a circular component would produce perspectively correct results, but suffers from a singularity at the vanishing point of the view axis (b). We use an alternative pattern in view space which is more stable and produces similar results (c).*

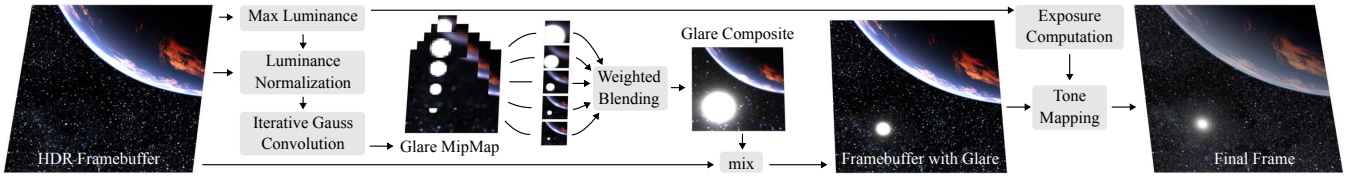

Figure 6: *As the resolution of the mipmap pyramid used for glare computation halves with each level, the standard deviation of the Gaussian effectively doubles. The levels are blended using consecutively halved weights. Star brightness is exaggerated for visualization purposes.*

5.1. Implementation Details

Our glare-computation pipeline is shown in *Figure 6*. The glare is computed using a mipmap pyramid. The base level of the pyramid has half the resolution of the render target. The levels of the pyramid are updated with several compute passes and contain successively more blurred versions of the scene. In a final pass, the mipmap levels are combined using efficient bi-cubic texture lookups [SH05] with the weight of each level being $1/2^i$. The result is then mixed with the render target.

We implemented two variants: The first decomposes the Gaussian kernel into a horizontal and a vertical component in screen space. This produces a circular glare which is very fast as only a few samples are required but not perspectively correct. The second variant decomposes the Gaussian kernel in view space (see *Figure 5*). This produces a more realistic glare but is computationally more expensive. Example images are shown in *Figure 1*.

5.2. Performance Evaluation

As we know the maximum scene luminance for the current frame, we can scale the glare luminance to fit into half floats. Hardware bilinear interpolation can be used to sample the average of four pixels in the previous level of the pyramid. Also, the glare pyramid can be computed with a relatively low resolution and still produce a smooth glare thanks to the bi-cubic texture filtering. Computing the screen-space glare took about 0.34 ms at Full-HD resolution and 0.74 ms at 4k resolution on an RTX 4070 Super. The asymmetric correct glare took about 0.41 ms and 1.38 ms respectively.

6. Summary and Future Work

In this paper, we have evaluated several techniques for rendering stars in a space simulation. We have shown that rendering stars as small as possible and adding glare in a post-processing step scales best with the number of stars and usually provides the best performance. For this, we proposed a software rasterizer for stars which performs better than native point-based rendering. We have also shown that a high-quality glare effect can be achieved on modern hardware with very little performance overhead.

In the future, we plan to investigate whether a tile-based rasterizer could further improve the performance. Avoiding the spin-lock like [SKW22] could also be beneficial, however an implementation of atomic additive color blending will not be trivial. Also, adding a level-of-detail algorithm to our system so that we can visualize the full Gaia star catalogue is an interesting avenue for future work.

Acknowledgements

This work is supported by the German Aerospace Center (DLR) Space Administration with financial means of the German Federal Ministry of Economic Affairs and Climate Action (BMWK) on the basis of a decision by the German Bundestag as part of the VaMEx3-RGE project (50NA2204A). Open Access funding enabled and organized by Projekt DEAL.

References

[Bla46] BLACKWELL H. R.: Contrast thresholds of the human eye. *J. Opt. Soc. Am. 36*, 11 (Nov 1946), 624–643. 2

[Cel] CELESTIA DEVELOPMENT TEAM: Celestia: Real-Time 3D Visualization of Space. Open-Source Software. Accessed: 2024-12-13. URL: https://celestiaproject.space/. 2

[Dui] DUIKER H.-P.: Filmic tonemapping for real-time rendering. *Proceedings of ACM SIGGRAPH Courses. ACM 28*, 701–711. 2

[GKLR13] GÜNTHER C., KANZOK T., LINSEN L., ROSENTHAL P.: A GPGPU-based pipeline for accelerated rendering of point clouds. *Journal of WSCG 21* (2013). 3

[HH21] HARRE J.-V., HELLER R.: Digital color codes of stars. *Astronomische Nachrichten 342*, 3 (2021), 578–587. 2

[KMS05] KRAWCZYK G., MYSZKOWSKI K., SEIDEL H.-P.: Perceptual effects in real-time tone mapping. In *Proceedings of the 21st Spring Conference on Computer Graphics* (New York, NY, USA, 2005), SCCG '05, Association for Computing Machinery, p. 195–202. 2, 3

[KSW21] KARIS B., STUBBE R., WIHLIDAL G.: A deep dive into Nanite virtualized geometry. In *ACM SIGGRAPH 2021 Courses, Advances in Real-Time Rendering in Games, Part 1* (2021), ACM. 3

[PTYG00] PATTANAIK S. N., TUMBLIN J., YEE H., GREENBERG D. P.: Time-dependent visual adaptation for fast realistic image display. In *Proceedings of SIGGRAPH '00* (USA, 2000), ACM, pp. 47–54. 2

[Sem] SEMENOV B.: WebGeocalc and cosmographia: Modern tools to access OPS SPICE data. In *2018 SpaceOps Conference*, p. 2366. 2

[SFGG25] SCHNEEGANS S., FLATKEN M., GILG J., GERNDT A.: CosmoScout VR, Jan. 2025. doi:10.5281/zenodo.14748678. 2

[SH05] SIGG C., HADWIGER M.: Fast third-order texture filtering. *GPU gems 2* (2005), 313–329. 4

[SJMS18] SAGRISTÀ A., JORDAN S., MÜLLER T., SADLO F.: Gaia Sky: Navigating the GAIA catalog. *IEEE transactions on visualization and computer graphics 25*, 1 (2018), 1070–1079. 2

[SKW22] SCHÜTZ M., KERBL B., WIMMER M.: Software rasterization of 2 billion points in real time. *Proc. ACM Comput. Graph. Interact. Tech. 5*, 3 (July 2022). 4

[SZGG22] SCHNEEGANS S., ZEUMER M., GILG J., GERNDT A.: CosmoScout VR: A Modular 3D Solar System Based on SPICE. In *2022 IEEE Aerospace Conference (AERO)* (2022), pp. 1–13. 2

[Vos03] VOS J. J.: On the cause of disability glare and its dependence on glare angle, age and ocular pigmentation. *Clinical and experimental optometry 86*, 6 (2003), 363–370. 3