UTRECHT UNIVERSITY

Artificial Intelligence

Bachelor Thesis Artificial Intelligence

Enhancing Gaming Experience: Image Outpainting Applied to Asynchronous Reprojection

First examiner:

Peter Vangorp

Second examiner:

Yupei Du

Candidate:

Tsjerk Piter Walinga

0962414

February 1, 2024

Abstract

This study introduces a novel approach that combines asynchronous reprojection and image outpainting to accelerate game rendering. The literature review outlines prevalent AI-based techniques, leading to the proposed method. In implementation, we detail the process of using the current frame for outpainting, generating expanded images, and employing asynchronous reprojection. Testing reveals that this combination maintains superior temporal cohesion compared to baseline and an alternative. Execution time assessments suggest potential for further optimization. The results demonstrate the potential effectiveness of the proposed method in enhancing game rendering performance. Considerations for future optimization and other possible techniques are discussed.

Contents

1. Introduction

The increasing demand for realistic game graphics is straining GPU performance, making it challenging to achieve playable frame rates. A significant issue associated with low frame rates in games is the sluggish sensation experienced while looking around[1]. Asynchronous reprojection has emerged as a potential solution, where the act of looking around occurs independently of game rendering. This reuses the previous frame for the generation of a new frame by projecting them on the new camera viewport. This is done in parallel with the normal rendering and shown between fully rendered frames if the next frame is not ready on time[2]. This technique is currently mostly used for virtual reality applications. However, this technique introduces a drawback: missing information, since elements outside the rendered view cannot be displayed without undergoing rendering. The objective of this research is to address this information gap by leveraging existing AI image outpainting techniques and exploring if this avenue of improving frame rate is worth exploring more. Image outpainting is the act of expanding the image outwardly. In contrast image inpainting fills in parts on the inside of an image[3], which is for example used in Photoshop for removing unwanted objects[4]. Both can be used in slightly different ways to address this information gap. We chose for outpainting, this decision is further discussed in chapter 3.

1.1 Research question

Can the integration of AI image outpainting with asynchronous reprojection contribute to a higher frame rate?

2. Literature

There have been a lot of attempts at using AI for speeding up game rendering. In this section we will be discussing a few, their trade-offs, the basis of the techniques we use and how our attempt differs.

2.1 AI based rendering acceleration techniques

2.1.1 Upscaling

By far the most popular AI based technique for speeding up game rendering has been upscaling, it is now included in most new games and all big graphics card manufacturers have their own version (although not all versions use AI techniques). For example, DLSS (deep learning super sampling) by Nvidia is a spatial upscaler that uses motion vectors and the current low resolution frame to generate a higher resolution output using two convolutional neural networks. One as a preprocessing step for the second which does the upscaling[5]. Adding an extra layer of processing to the rendering does add extra overhead increasing input latency, however in almost all cases this is canceled out by the faster rendering of the frame itself[6].

2.1.2 Denoising

Denoising, a technique aimed at reducing noise in an image, can be implemented in various ways. The simplest method involves blurring the image, albeit at the expense of sharpness. Consequently, a multitude of techniques have been developed, with some leveraging machine learning approaches as evidenced by studies such as [7] and [8]. Denoising is specifically used for path-traced or ray-traced rendering because those techniques rely on averaging lots of rays to create smooth looking results. Normally there is a need for a lot of samples to get rid of the noise in path-traced or ray-traced games.

Denoising allows us to lower the sample count which speeds up the rendering.[9] This technique is increasingly prevalent as more games are opting for ray-traced elements for the rendering[10][11].

2.1.3 Frame Interpolation

Frame interpolation is a technique used in computer graphics and video processing that aims to improve visual experiences by generating intermediary frames between existing ones, instead of focusing on making the frames easier to render[12]. This method enhances the frame rate of the game by generating complete frames in between fully rendered frames. Frame interpolation involves algorithms that analyze existing frames, extrapolating information to generate intermediary frames. Techniques such as optical flow-based methods or machine learning-driven approaches are commonly used for this purpose. For example in DLSS 3.0 which is a machine learning-driven approach that estimates the frame between the current and the prior frame, using both frames, an optical flow field generated as a preprocessing step and game engine data[13]. Using frame interpolation to increase frame rate does come with a slight downside: The interpolated frame is based on the current and previous frame, which means showing the current frame is delayed so an interpolated frame can be shown in between. This introduces extra input latency for the user, which is one of the original benefits of a higher frame rate.

2.2 Asynchronous Reprojection

Asynchronous reprojection has until now almost been exclusively used for VR purposes. In VR it is essential to maintain a high frame rate, because low frame rates can lead to motion sickness due to the resulting unresponsiveness to the user motion [14]. This is why asynchronous reprojection was developed, a technique in which frames are inserted between fully rendered frames by reprojecting the current frame based on the camera movement, even during high system load. This is done by doing it asynchronously from the rendering, allowing rotational movement to occur while the next

frame is being rendered. Anynchronous reprojection helps frame drops by ensuring a high frame rate experienced by the user even while the game is unable to draw new frames[2].

2.2.1 Making sense of the names

A lot of different companies in the VR space have used different names for techniques based on asynchronous reprojection, this is our attempt at providing an overview of them and how they differ.

2.2.1.1 Timewarp

Timewarp is the term used by Meta for describing reprojection that is helping reduce latency. Orientation Timewarp is the term for a variant that only uses the rotational change in head position to reproject the frame. There are 2 versions an asynchronous and synchronous version, referred to ATW and STW respectively. Where the asynchronous version is decoupled from the rendering, the synchronous version always applies it right after a completed frame [15][16].

2.2.1.2 Asynchronous Spacewarp 1.0

Asynchronous Spacewarp is the term used by Meta for a supplementary technology to ATW, that tries to extrapolate the character movement, controller movement and the player's own positional movement. It is intended to use next to ATW and is often referred to as ASW or ASW 1.0 [17].

2.2.1.3 Positional Timewarp

Also referred to as PTW, a technique used by Meta that leverages depth information for a sparse-parallax-mapping technique to correct for rotation and translational movement[16]. Parallax mapping moves parts of an image at different rates based on their distance, so further objects move less than close objects, which mimics the effect of actually moving the view.

2.2.1.4 Asynchronous Spacewarp 2.0

A version of ASW that uses PTW instead of ATW. Whenever a game does not provide depth information ASW 1.0 is used instead[16].

2.2.1.5 Motion Smoothing

Developed by Valve, this technique extrapolates the next frame by estimating the motion based on the last two frames. Intended as an improvement upon asynchronous reprojection, it allows for rotational and translational movement but can in some cases introduce visual artifacts[18].

2.2.1.6 Interleaved reprojection

Used by Valve, this is a fallback for systems that cannot support asynchronous reprojection. Interleaved reprojection reprojects every other frame using rotation only reprojection. This does the same as STW[19].

2.3 Outpainting

Outpainting, the act of expanding an image outwards, has so far mostly been used for creative purposes, because there is no single obvious use case. It has for example been proposed as a way to create 360-degree images for use in the backgrounds of cgi or games[20] and for creating textures for 3d models [21]. The use of outpainting based on AI has especially risen since the rise in popularity of generative AI models such as Dall-e 2[22] and Stable diffusion[23] which also have the ability to outpaint[24][25]. These recent models employ diffusion-based techniques, tasking the model with progressively refining noise in small increments. In contrast, many earlier models relied on Generative Adversarial Networks (GANs). GANs operate within a machine learning framework where two models, such as an image generator and a discriminator, compete. For instance, the image generator aims to deceive the discriminator into believing its generated images are authentic, facilitating mutual improvement in their respective tasks [26].

There have also been video outpainting developed, however those are

not directly applicable to our use case as real-time outpainting is required, and the subsequent frames are unknown. Consequently, real-time video outpainting for games is not possible due to not knowing how the next frames will look. Despite this limitation, insights from video outpainting techniques, particularly those addressing temporal coherence issues, could prove valuable for enhancing our proposed technique, given that temporal cohesion poses a similar challenge in our approach [27].

2.4 Combination of asynchronous reprojection and Outpainting

The combination of asynchronous reprojection with outpainting is different from the AI based techniques we have discussed by being able to create extra frames while the next frame is still being rendered. This means that it could possibly have a much greater impact on the frame rate compared to other techniques due to it being based on the asynchronous reprojection; frame interpolation has only been used for inserting 1 frame in between fully rendered frames and the other techniques only make the normal rendering faster. Techniques that are based on making the normal rendering faster like upscaling a lower resolution frame have the drawback of still having to wait on the rendering, which means that those techniques will not help anytime a frame takes longer for some reason. The combination of asynchronous reprojection and outpainting also does not add an extra delay to the normal rendering like the other techniques, so it should not have an impact on input latency.

3. Method

3.1 Implementation

The implementation of the combination of asynchronous reprojection and image outpainting is fairly straightforward and does not need any scene specific information: The current frame is used as input for the image outpainting method, which then produces an expanded image. This resulting image is then used for the reprojection and shown if the next fully rendered frame is not ready yet. This whole process is, as the name implies, done asynchronously from the normal rendering.

3.1.1 Outpainting strategies

We do still have a decision with differing trade-offs to make here:

- Only outpaint fully rendered images; so, the outpainting is only done once for every fully rendered frame.
- Also apply the outpainting to the reprojected frames we have inserted between fully rendered frames (of which thus a part is already outpainted)

The first approach is faster because the outpainting only has to be done once and is then reused for subsequent reprojections (until the next fully rendered frame), this way we could also look back towards the center of the fully rendered frame and it would reuse the fully rendered frame. This approach does have a problem: once we look far enough outside of the original view there is not enough extra information created by the outpainting which results in the edge where we are trying to look to not contain any information, which results (in our implementation) in black. If this were to occur often it could be quite distracting to the user. The second approach does not have this problem because it continuously generates more infor-

mation outside of the known area, however this comes at the cost of performance because the image outpainting has to be applied every time we would like to reproject the image. With different outpainting techniques these tradeoffs could look slightly different, for example in a technique that can outpaint with a mask such that only the part we need is outpainted the second approach could be better. Conversely, a technique that can outpaint to an arbitrary size the first approach could be tuned to never have the problem in real world situations (for example by changing the outpainting size depending on the mouse movement speed). A hybrid approach that only expands the outpainting if needed could also be interesting however this could add extra overhead due to the extra check involved. In our proof of concept we opted for the first approach mainly due to the speed of the technique used (see chapter 4 for more information on this).

3.1.2 Inpainting or outpainting

We could also use image inpainting on the frame after the reprojection, then we would know exactly what has to be filled in. This would in a similar way as the second approach get rid of the problem of the first approach discussed in the section above. However preliminary testing on our part showed unfavorable results, the image inpainting method we tried[28] left an obvious edge and seemed unable to produce a satisfactory result when inpainting on the edge of an image.

3.2 The proof of concept

To identify possible roadblocks we made a proof of concept for which we used the opensource game engine Godot for its extensibility and ease of use. We found an opensource implementation of asynchronous reprojection for godot[29]. We only used 3 degrees of freedom reprojection, which means only rotation is possible between fully rendered frames and no translation. This is so there is no additional depth information needed for the reprojection and things that are hidden behind other objects do not become a problem. We used it on non-vr games for ease of development. As explained in

the section above, the approach where outpainting is only applied once between fully rendered frames was chosen. The utilized outpainting method processed 128x128 images, extending them to 192x192 as detailed in [30]. For this proof of concept, we constrained the game resolution to 128x128. However, this fixed resolution is suboptimal for practical games. To address this limitation, an outpainting method capable of handling arbitrary resolutions or incorporating additional downscaling and upscaling steps should be implemented. Without such flexibility, users would be confined to the specified resolution of 128x128. This resolution is significantly lower than the widely adopted 1920x1080 resolution, identified as the most common according to a Hardware Survey, where 59.58% of users (at the time of writing) opted for this resolution [31].

4. Results

4.1 Testing

The resulting program was tested for temporal cohesion, how much the inpainting differs per frame. As testing temporal cohesion provides a measure of how much flickering there is in the filled in areas and thus how distracting it will be for the user. It was also tested for execution time, specifically considering if it is less than needed for real time rendered applications (60 frames per second, which is seen as the standard in this context due to most monitors having a refresh rate of 60hz), as exceeding this threshold would render it impractical for use in games. We did not test against ground truth because that only tests how well the outpainting technique performs on images instead of this specific use case, this is also already done in the paper of the technique we use. For testing we have taken screenshots from a game, downscaled them and made a panning image sequence out of it, then on every image we applied the outpainting. The difference is a value between 0 and 1 that is calculated per pixel, for example the difference between black (0,0,0) and white (255,255,255) would be 1 and the difference between black (0,0,0) and red (255,0,0) would be 0.33. This difference is then averaged over all pixels being compared; so all differences added up and divided by the number of white pixels in the mask. This difference was calculated between all subsequent images in the image sequence using the corresponding mask. This area should – in an ideal scenario – stay the same because the mask shifts over one pixel to the bottom left and the panning motion move over one pixel to the top right. It was tested in this way to simulate looking around in a game. As seen in Figure 4.1, the mask is an L shape that moves in the opposite direction of the panning motion such that the compared area should contain the same information in the ideal scenario.

This was done on image sequences of 16 images per scene, which re-

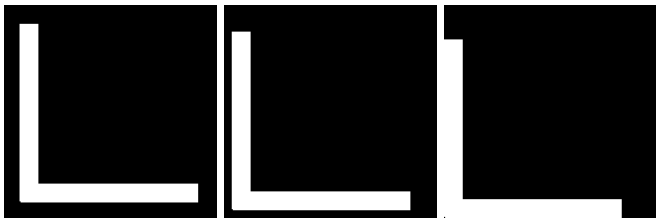

Figure 4.1: Three frames of the mask

sulted in 15 differences between the images per scene as shown in Figure 4.5. This testing was then done on three different techniques for filling in the borders:

- The outpainting technique
- Random noise, that is different per frame, as a baseline. This was provides a good baseline because if what we are doing is worse in difference per frame than random noise it would not make much sense.
- Stretching the last pixel over the sides. This is used as an example of a simple technique that does not create obvious borders. We did not stretch it over the corners because there is no single obvious way to do this, the lack of the corners does mean that the stretching method has a lower difference result as the corners do not change.

4.2 Examples

To give an idea of how the 3 different techniques look we will provide an example for each technique on the same scene – Figure 4.2, Figure 4.3 and Figure 4.4 – These show two subsequent images of the sequence we tested on with the three techniques discussed above.

Figure 4.2: Two subsequent testing images using the outpainting

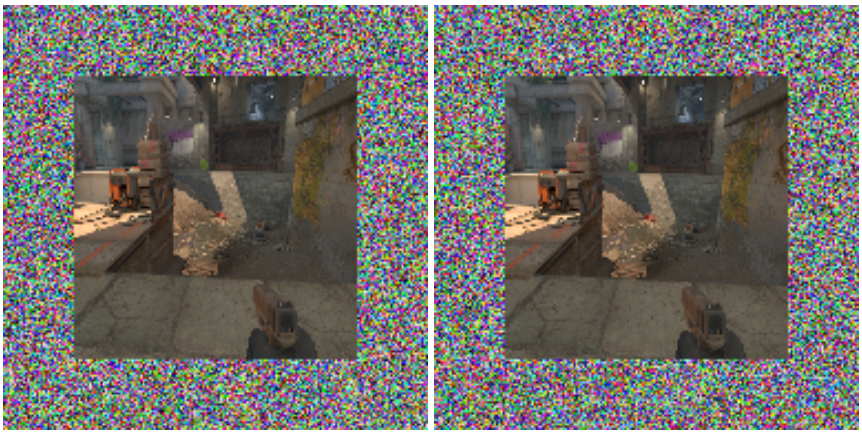

Figure 4.3: Two subsequent testing images with the noise method

Figure 4.4: Two subsequent testing images where the last pixel has been stretched over the sides

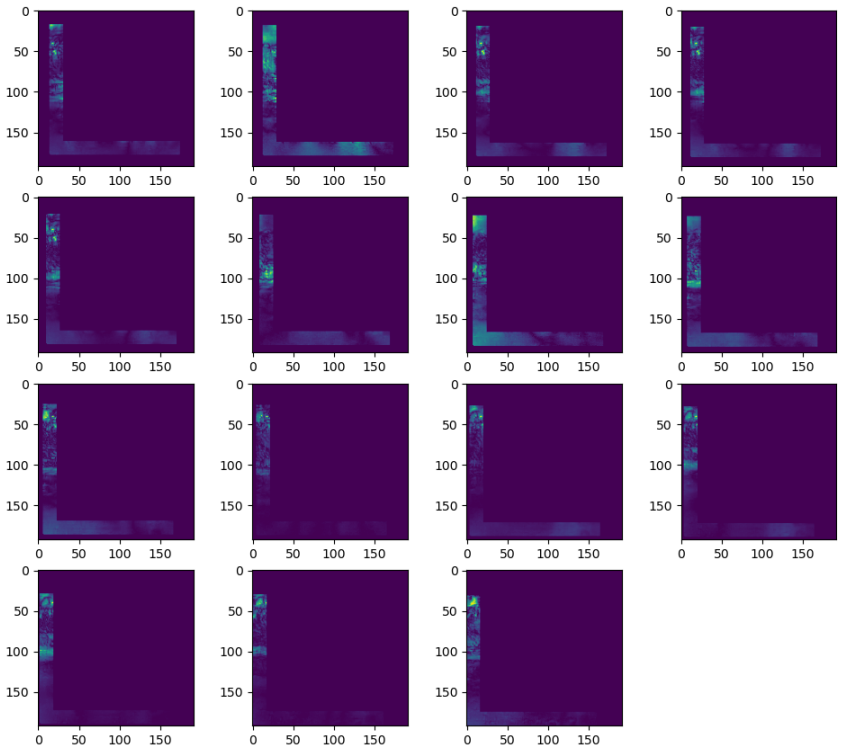

Figure 4.5: Differences per frame from a test scene using the outpainting

4.3 Quality

From testing on 11 samples of different scenes in a recent game title (Counter-Strike 2) the following results were observed:

Scene	Outpainting	Noise	Stretching
0	0.0399	0.2215	0.0569
1	0.0431	0.2231	0.0809
2	0.0635	0.2220	0.1179
3	0.0174	0.2223	0.0273
4	0.1036	0.2212	0.1070
5	0.0459	0.2218	0.0725
6	0.0293	0.2223	0.0535
7	0.0351	0.2220	0.0522
8	0.0484	0.2215	0.0857
9	0.0499	0.2230	0.0743
10	0.1109	0.2228	0.1356
Average	0.0534	0.2221	0.0785
Minimum	0.0090	0.2173	0.0154
Maximum	0.2125	0.2278	0.2267
Variance	0.0014	4.585E-06	0.0015

Table 4.1: Results per different scenes per technique.

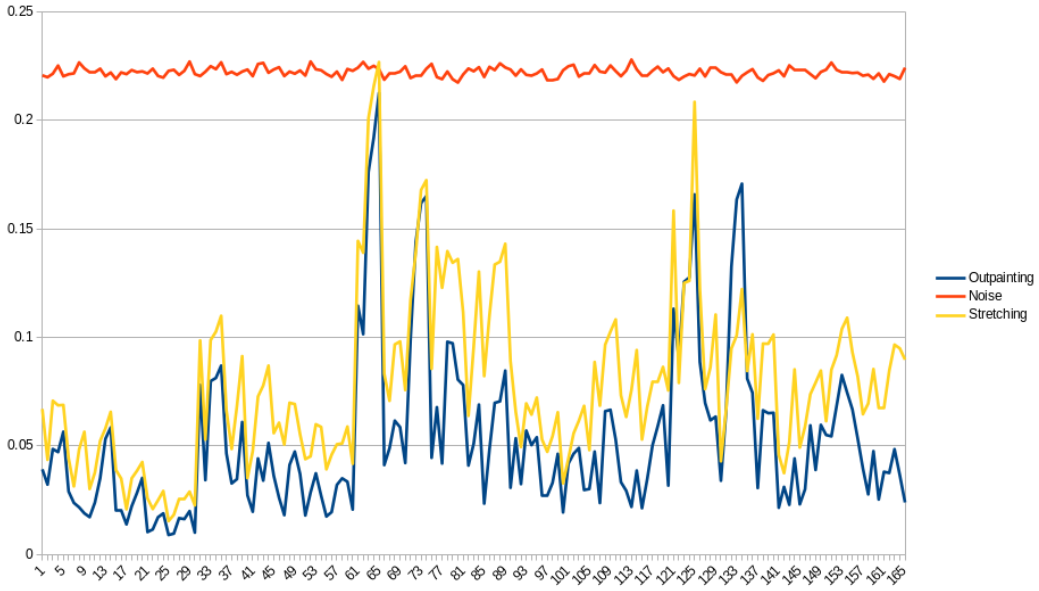

Figure 4.6: The differences from all frames plotted

When we average the results shown in Figure 4.6 we get the results

shown in Figure 4.7. For an overview of the performance on different scenes we have included Table 4.1.

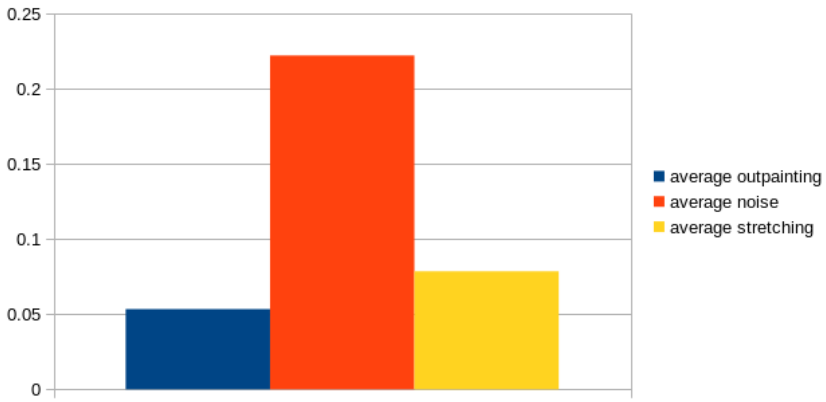

Figure 4.7: The consistency of outpainting compared to random noise and stretching the last pixel to the border

These are the average differences between the masked areas in subsequent frames, where lower is better. To make these results more intuitive you could see them as percentages, e.g. the compared area from the outpainting technique differed 5.336% on average. As shown the outpainting technique changes significantly less than the other techniques, even with the lack of corners on the stretching technique. The lower average difference would make it less distracting for the user when used in games.

4.3.1 Limitations

There are of course limitations of only testing the difference between frames, for example if we filled in the outside area with a single color and kept this the same it would get a perfect score while not giving us the result we want (because the edge would be obvious and distracting). This is why we opted to test against random noise as a baseline instead of not filling in anything. The stretching method was also chosen for this reason, it gives us a simple technique that does change per frame. The stretching technique also does not produce an obvious border which makes it even more compelling to compare against. -

4.4 Speed

After optimizing the outpainting technique of our choice by removing features we did not need for our purpose such as arbitrary size of input images and re-setting the evaluation mode every time, we tested the speed: The average of 100 runs was 0.084 seconds on the CPU and 0.014 seconds when using the GPU for the PyTorch operations. These results translate into roughly 12 and 71 frames per second respectively. Having the PyTorch operations run on the GPU does mean a faster execution time – in our case an almost sixfold improvement – although this does mean that it takes up GPU resources which we want to avoid due to that being the most common bottleneck for modern games. The system we tested on was a Lenovo legion 5 pro 16iah7h laptop running an intel i7 12700H and an RTX 3070TI. The runs where we got the averages from consisted of making a masked image of the input, doing the outpainting and blending it with the starting image, as these steps would need to be done every time a new image is outpainted in a gaming scenario. The model and starting image were already loaded in RAM and resulting images were not saved. This is so it better simulates an in-game scenario in which the model would be pre-loaded during loading times, the starting images in RAM due to rendering and the resulting images discarded after being shown.

5. Further Research

5.1 Different techniques

From the insights gained from our research we got more ideas for different techniques that could be worth exploring, we will briefly discuss them here.

5.1.1 Upscaling wider lower resolution view

Further research could focus on using existing upscaling techniques (e.g. FSR[32] or DLSS[5]) to upscale a wider view from a low resolution for use in filling in the missing information of the re-projection. These techniques have become a lot faster than real time which would make it possible to insert more frames between fully rendered frames and thus create a smoother experience.

5.2 Improving this technique

Our proposed technique also leaves a lot to be desired, the temporal coherence could still be improved, currently only the rotational movement of the player view is smoother, the technique is still too slow to be used in practice when we are not using the GPU and when the player is looking around too quickly at a low frame rate he could run into missing information. In this section we will propose some ways further research can try to fix some of these remaining problems.

5.2.1 Game specific training

An option for improving our technique would be using game or even scene specific training to potentially improve the temporal coherence and accu-

racy of the result. For example by having a model per game level. This would help by making the model more specific. Training of these models could be automated by taking random realistic (so from the same points in space that the player could take) viewpoints. These viewpoints should then be taken with the field of view the size of what we would like to outpaint to, and then use these and a cut in version as the training data for our model. Creating game or scene specific models could add a lot of extra complexity or at the very least a lot of extra training time to game development so this might not be ideal.

5.2.2 More degrees of freedom

Extending our technique to 6 degrees of freedom, so including movement and not only rotation, and using AI inpainting[28] to also fill in the missing parts behind objects would also be an interesting follow up. One way of doing this is by using the positional timewarp[16] or the motion smoothing[18] techniques we discussed in chapter 2.

5.2.3 Model made for this purpose

If you were to create an image outpainting model specifically for the purpose of use for asynchronous reprojection you would be able to focus on speed and making the result as least distracting as possible instead of only on accuracy as current models do. As discussed previously in chapter 3, a model with an arbitrary outpainting size could also benefit this technique by for example extrapolating how fast the player is looking around per frame to determine the outpainting size needed to never have the player run into any missing information. Because game engines often create extra information such as depth information, this could also be used to generate a more accurate and/or a more consistent result, just like how DLSS 2.0 uses motion vectors[5] or how DLSS 3.0 uses the optical flow as a preprocessing step[13]. Adding additional information has already been shown in research in the past to produce a qualitative improvement in image outpainting[21]. More information can also improve the temporal coherence of the technique as

shown in video outpainting techniques [27].

5.2.4 Stretching past the outpainting

An additional step which could be taken to never have the player run into an obvious border of missing information would be stretching the last pixel of the outpainted area far enough to fill it in. This would be like the stretching method we tested against but with the additional benefit of the extra accuracy and consistency of the outpainting for the first part of the missing information needed for the asynchronous reprojection. This could be a good compromise if outpainting farther means less accuracy or a slower result.

6. Conclusion

There have been a lot of attempts at using AI to speed up game rendering such as upscaling, denoising and frame interpolation. In the VR scene there are lots of different methods such as asynchronous reprojection and similar techniques for making the experience smoother for the user. We combined the basic asynchronous reprojection with image outpainting to fill in the gaps. We have detailed the straightforward process of using the current frame for outpainting, generating expanded images, and employing asynchronous reprojection for displaying these images during rendering.

Testing focuses on temporal cohesion and execution time. Temporal cohesion is evaluated by calculating differences between the outpainting in subsequent frames, revealing that the outpainting technique used maintains temporal coherence better than the baseline of random noise and also better than stretching the last pixel over the border. Execution time was also measured, indicating further optimisation to be needed for the used technique to be useful.

This indicates that this might be a new way to make for a better experience, but further research is needed to make it possible. Multiple ways in which further research could be done have been discussed and should provide a good starting point for anyone building upon our research.

So can the integration of AI image outpainting with asynchronous reprojection contribute to a higher frame rate? Yes, if it becomes fast enough, it should not be too distracting.

Bibliography

[1] M. Claypool, K. Claypool, and F. Damaa, "The effects of frame rate and resolution on users playing first person shooter games," *Proceedings of SPIE - The International Society for Optical Engineering*, vol. 6071, Jan. 2006. DOI: 10.1117/12.648609.

[2] Google, *Asynchronous reprojection*. [Online]. Available: https://developers.google.com/vr/discover/async-reprojection.

[3] Y.-C. Cheng, C. H. Lin, H.-Y. Lee, J. Ren, S. Tulyakov, and M.-H. Yang, "In&out: Diverse image outpainting via gan inversion," vol. abs/2104.00675, 2021.

[4] Adobe, *Remove objects from your photos with content-aware fill*, Jul. 2023. [Online]. Available: https://helpx.adobe.com/photoshop/using/content-aware-fill.html.

[5] A. Burnes, *Nvidia dlss 2.0: A big leap in ai rendering*, Accessed: January 28, 2024, Mar. 2020. [Online]. Available: https://www.nvidia.com/en-us/geforce/news/nvidia-dlss-2-0-a-big-leap-in-ai-rendering.

[6] H. Unboxed, *Does dlss hurt input latency?* Accessed: January 27, 2024, Sep. 2021. [Online]. Available: https://www.youtube.com/watch?v=osLDDl3HLQQ.

[7] Y.-Q. Wang and J.-M. Morel, "Can a single image denoising neural network handle all levels of gaussian noise?" *IEEE Signal Processing Letters*, vol. 21, no. 9, pp. 1150–1153, 2014. DOI: 10.1109/LSP.2014.2314613.

[8] B. Liu and S.-i. Kamata, "Combined convolutional neural network for highly compressed images denoising," in *2020 Joint 9th International Conference on Informatics, Electronics Vision (ICIEV) and 2020 4th International Conference on Imaging, Vision Pattern Recognition (icIVPR)*, 2020, pp. 1–7. DOI: 10.1109/ICIEVicIVPR48672.2020.9306597.

[9] C. R. A. Chaitanya, A. S. Kaplanyan, C. Schied, *et al.*, "Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder," *ACM Trans. Graph.*, vol. 36, no. 4, Jul. 2017, ISSN: 0730-0301. DOI: 10.1145/3072959.3073601. [Online]. Available: https://doi.org/10.1145/3072959.3073601.

[10] J. Roach, "All ray tracing games on pc: Amd radeon and nvidia rtx ray tracing," *digitaltrends*, Oct. 2023, Accessed: January 28, 2024. [Online]. Available: https://www.digitaltrends.com/computing/games-support-nvidia-ray-tracing.

[11] A. Burnes, *Nvidia dlss 3.5: Enhancing ray tracing with ai; coming this fall to alan wake 2, cyberpunk 2077: Phantom liberty, portal with rtx*

more, Accessed: January 28, 2024, Aug. 2023. [Online]. Available: https://www.nvidia.com/en-us/geforce/news/nvidia-dlss-3-5-ray-reconstruction/.

[12] A. S. Parihar, D. Varshney, K. Pandya, and A. Aggarwal, "A comprehensive survey on video frame interpolation techniques," *The Visual Computer*, vol. 38, no. 1, pp. 295–319, Jan. 2021, ISSN: 1432-2315. DOI: 10.1007/s00371-020-02016-y. [Online]. Available: http://dx.doi.org/10.1007/s00371-020-02016-y.

[13] H. C. Lin and A. Burnes, *Nvidia dlss 3: Ai-powered performance multiplier boosts frame rates by up to 4x*, Accessed: January 27, 2024, Sep. 2020. [Online]. Available: https://www.nvidia.com/en-us/geforce/news/dlss3-ai-powered-neural-graphics-innovations.

[14] J. Wang, R. Shi, W. Zheng, W. Xie, D. Kao, and H.-N. Liang, "Effect of frame rate on user experience, performance, and simulator sickness in virtual reality," *IEEE Transactions on Visualization and Computer Graphics*, vol. 29, no. 5, pp. 2478–2488, 2023. DOI: 10.1109/TVCG.2023.3247057.

[15] M. Antonov, *Asynchronous timewarp examined*, Accessed: January 28, 2024, Mar. 2015. [Online]. Available: https://developer.oculus.com/blog/asynchronous-timewarp-examined.

[16] D. Beeler and V. Aksoy, *Developer guide to asw 2.0*, Accessed: January 28, 2024, Aug. 2019. [Online]. Available: https://developer.oculus.com/blog/developer-guide-to-asw-20.

[17] E. H. Dean Beeler and P. Pedriana, *Asynchronous spacewarp*, Accessed: January 28, 2024, Nov. 2016. [Online]. Available: https://developer.oculus.com/blog/asynchronous-spacewarp.

[18] Valve, *Introducing steamvr motion smoothing*, Accessed: January 28, 2024, Nov. 2018. [Online]. Available: https://steamcommunity.com/games/250820/announcements/detail/1705071932992003492.

[19] A. Vlachos, "Advanced vr rendering performance," Accessed: January 28, 2024, Mar. 2016. [Online]. Available: https://alex.vlachos.com/graphics/Alex_Vlachos_Advanced_VR_Rendering_Performance_GDC2016.pdf.

[20] N. Akimoto, Y. Matsuo, and Y. Aoki, "Diverse plausible 360-degree image outpainting for efficient 3dcg background creation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2022, pp. 11 441–11 450.

[21] C. Shi, Y. Ren, X. Li, I. Mumtaz, Z. Jin, and H. Ren, "Image outpainting guided by prior structure information," *Pattern Recognition Letters*, vol. 164, pp. 112–118, 2022, ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2022.10.030. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167865522003294.

[22] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, *Hierarchical text-conditional image generation with clip latents*, 2022. arXiv: 2204.06125 [cs.CV].

[23] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, *High-resolution image synthesis with latent diffusion models*, 2022. arXiv: 2112.10752 [cs.CV].

[24] OpenAI. "DALL-E: Introducing Outpainting." Accessed: January 27, 2024. (2022), [Online]. Available: https://openai.com/blog/dall-e-introducing-outpainting.

[25] AUTOMATIC1111. "Stable-diffusion-webui." Accessed: January 27, 2024. (2022), [Online]. Available: https://github.com/AUTOMATIC1111/stable-diffusion-webui.

[26] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 53–65, 2018. DOI: 10.1109/MSP.2017.2765202.

[27] L. Dehan, W. Van Ranst, P. Vandewalle, and T. Goedemé, "Complete and temporally consistent video outpainting," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Jun. 2022, pp. 687–695.

[28] G. Liu, F. A. Reda, K. J. Shih, T.-C. Wang, A. Tao, and B. Catanzaro, *Image inpainting for irregular holes using partial convolutions*, 2018. arXiv: 1804.07723 [cs.CV].

[29] Ogilliland, *Godot-asynchronous-reprojection*, Dec. 2022. [Online]. Available: https://github.com/ogilliland/godot-asynchronous-reprojection.

[30] B. V. Hoorick, "Image outpainting and harmonization using generative adversarial networks," *CoRR*, vol. abs/1912.10960, 2019. arXiv: 1912.10960. [Online]. Available: http://arxiv.org/abs/1912.10960.

[31] "Steam hardware & software survey." Accessed: January 27, 2024. (2024), [Online]. Available: https://store.steampowered.com/hwsurvey/Steam-Hardware-Software-Survey-Welcome-to-Steam.

[32] Kurbeco, *Fidelityfx super resolution 2.2.2 (fsr2)*, Jul. 2023. [Online]. Available: https://github.com/GPUOpen-LibrariesAndSDKs/FidelityFX-SDK/blob/main/docs/techniques/super-resolution-temporal.md.