Scholarly Horizons: University of Minnesota, Morris Undergraduate Journal

Volume 10 | Issue 2 Article 4

June 2023

Deep-Learning Realtime Upsampling Techniques in Video Games

Biruk Mengistu
University of Minnesota Morris

Recommended Citation

Mengistu, Biruk (2023) "Deep-Learning Realtime Upsampling Techniques in Video Games," *Scholarly Horizons: University of Minnesota, Morris Undergraduate Journal*: Vol. 10: Iss. 2, Article 4.
DOI: https://doi.org/10.61366/2576-2176.1127
Available at: https://digitalcommons.morris.umn.edu/horizons/vol10/iss2/4

Deep-Learning Realtime Upsampling Techniques in Video Games

Biruk Mengistu
mengi024@morris.umn.edu
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA

Abstract

This paper addresses the challenge of keeping up with the ever-increasing graphical complexity of video games and introduces a deep-learning approach to mitigating it. As games get more and more demanding in terms of their graphics, it becomes increasingly difficult to maintain high-quality images while also ensuring good performance. This is where deep learning super sampling (DLSS) comes in. The paper explains how DLSS works, including the use of convolutional autoencoder neural networks and various other techniques and technologies. It also covers how the network is trained and optimized, as well as how it incorporates temporal antialiasing and frame generation techniques to enhance the final image quality. We will also discuss the effectiveness of these techniques as well as compare their performance to running at native resolutions.

Keywords: deep learning super sampling (DLSS), convolutional autoencoder neural networks, antialiasing, frame generation, real-time rendering, autoencoders, tensor cores, motion vectors, latency, optical flow

1 Introduction

Video games are a popular form of entertainment that require high computational power and graphical fidelity to deliver immersive and realistic experiences. However, achieving both high performance and high quality in video games is challenging due to the trade-off between video resolution and frame rate.

Video resolution refers to the number of pixels that make up an image on a screen, while frame rate refers to the number of images displayed per second (frames per second, or fps). Higher resolutions provide sharper and more detailed images, but they also demand more processing power from the graphics processing unit (GPU), which can lower the frame rate. Lower frame rates can cause stuttering or lagging effects that impair player performance and enjoyment.

Conversely, higher frame rates provide smoother and more responsive gameplay experiences, especially in fast-paced action games where split-second reactions can be crucial for winning or losing. However, increasing the frame rate often requires reducing the resolution or lowering the graphical settings of a game, which can degrade its visual quality and appeal.

In recent history, increases in graphical processing power have advanced at a slower pace compared to the introduction of more and more demanding graphical effects and resolutions. This has called for a different way of dealing with this issue rather than simply increasing graphical processing power — super-resolution (SR).

SR is a technique where a game is rendered at a lower resolution, then upscaled using a neural network, anti-aliased, and presented to the user at a higher resolution, with low to negligible effects on image quality [4]. A great example of SR is the widely-used *Deep Learning Super Sampling (DLSS)* technique developed by *NVIDIA*, whose techniques and technologies are covered in this paper.

2 Background

Before getting into DLSS, we will first discuss how games are rendered. While the curved edges of real-life objects are relatively smooth until you get down to the atomic level, any object rendered in-game is composed of a large but finite number of vertices — later converted to a series of triangles, that are combined to create the illusion of a smooth surface [5]. This brings us to the *Graphics Rendering Pipeline*, a series of steps a graphics system takes to render a 3D object to a 2D screen [14]. These steps are carried out not on the CPU, but rather on the *Graphics Processing Unit (GPU)* — a piece of hardware that does calculations in a highly parallel manner, unlike the limited parallelization of a CPU.

The series of raw vertices are first passed through the *Vertex Processor* (see Figure 1), which takes a series of *primitives* (defined as inputs to the pipeline built from vertices and can be *triangles, lines, or points*) and places their coordinates in the 3D space. The transformed vertices and primitives are then passed through what is called a *rasterizer*. A *rasterizer* converts the primitives into a series of *fragments*, which are 3D representations of 2D pixels. So, if we have two separate fragments overlapping each other, the fragment closer to the camera (a smaller delta between the z coordinate of the camera and the fragment) will get rendered.

These fragments are then passed on to the *fragment processor*, which calculates the final color of the fragments given the input parameters (lighting, texture information, etc.). Finally, these fragments undergo *output merging*, where the fragments are converted into a 2D grid of pixels.

https://digitalcommons.morris.umn.edu/horizons/vol10/iss2/4
DOI: 10.61366/2576-2176.1127

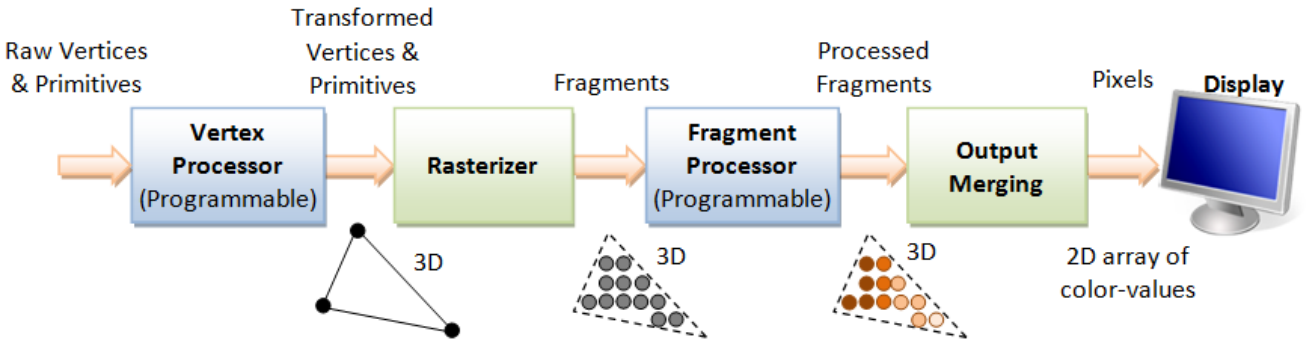

Figure 1. Steps in the Graphics Rendering Pipeline [3]

This brings us to a phenomenon known as *aliasing*. Aliasing is an undesirable visual artifact that occurs when an image is displayed or sampled at a resolution lower than what is required to accurately represent the original image, resulting in distortion and/or pixelation. In Figure 2, The pixels form a stair-like pattern along the edges of the line, creating a jagged appearance. This effect reduces the perceived quality and realism of the graphics.

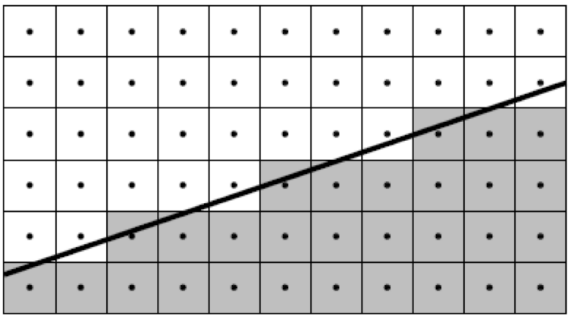

Figure 2. Aliasing [10]

Possible ways to counter aliasing are to increase the resolution (which would make the checkerboard-like pattern small and negligible) or by employing a technique known as *anti-aliasing* [4], which we will discuss in more detail in Section 3.2 - both of which require more processing power on the client computer. Higher resolutions in particular can require significantly more processing power to render. A resolution of 3840x2160 (4K), which is getting more and more popular recently, requires *four times* the processing power compared to the more conventional resolution of 1920x1080 (1080p).

2.1 The Need for a New Approach

In the same period when the gaming industry has transitioned to higher resolutions, there have also been newer and more demanding effects in games due to the demand for realism. This presents a problem - while advances in hardware have been able to keep up with advances in graphical fidelity in the past, that is simply not the case anymore. Especially if we seek to reap the best graphical fidelity that games have to offer as of now [4].

Recognizing this challenge, *NVIDIA* released a set of deep-learning super-sampling techniques known as DLSS (*Deep Learning Super Sampling*) starting in 2018. We will be looking at each major version of DLSS as well as the different techniques and improvements in each.

3 Deep Learning Super Sampling Techniques

As previously mentioned, DLSS is a *set* of technologies and techniques to upsample and enhance the image quality of video game graphics. Different versions of DLSS use a different combination or variations of technologies. So far, there have been three major versions of DLSS. We will discuss the technologies used in each version next.

3.1 DLSS 1.0

Being relatively the simplest and earliest form of DLSS, 1.0 is primarily an image upscaler that utilizes *Convolutional Auto-encoder Neural Networks.*

3.1.1 Convolutional Auto-encoder Neural Networks.

A convolutional auto-encoder neural network is a type of neural network that is great at dealing with image data. The neural network used in DLSS features a set of nodes arranged in layers - an input layer, a number of convolutional layers, and an output layer. *Nodes* are individual computational units that perform mathematical operations on the input data, while layers are collections of nodes that are organized in a specific way to extract features from the input. Each node in a layer is connected to every node in the previous layer, and every node in the next layer is connected to every node in the current layer.

The main purpose of the network is to take many inputs with desired outputs, get trained on those inputs, and produce accurate outputs for new inputs it has not yet seen. In

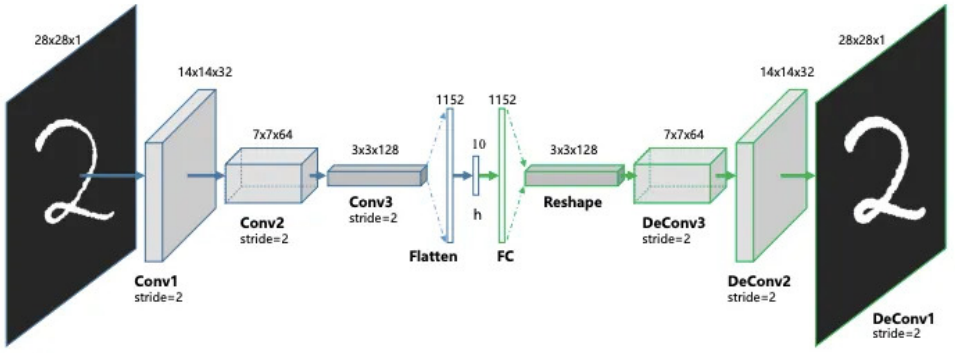

Figure 3. Autoencoder Convolutional Neural Networks [12]

the context of DLSS, the trained convolutional auto-encoder neural network is used to upscale lower-resolution images to higher resolutions, while maintaining as much detail as possible. Once trained, the network can be used to upscale low-resolution frames in real-time, making it a useful tool for applications such as video games and virtual reality.

Instead of the more typical "one-dimensional" layers, here we have a set of two-dimensional layers that help preserve spatial information in the input. If we were to input the pixels in the image as an array of pixels, for example, we would lose crucial spatial information in the reference image. This allows it to be better suited to process image data - as it is able to learn local patterns such as corners and edges in the image.

The *"Convolutional"* part of the name refers to its use of a mathematical operation called *convolution*, where a filter (a matrix of initially random values of a certain size) is passed over the input data and a dot product is computed between the filter and a certain part of the input [2]. The *dot product* is an arithmetic operation executed on a pair of matrices that results in a singular numerical value. To compute the dot product of two matrices, their corresponding components are multiplied together and then summed up to a single value. This dot product is then passed on to the next layer in the network.

To accelerate these calculations, DLSS utilizes what are called *tensor cores*, specialized processors that are well suited for complex matrix multiplications at high speed. Tensor cores significantly speed up the process of training deep learning models due to their specialization in matrix operations.

The *auto-encoder* part of the name refers to the way the network uses two processes, *encoding* and *decoding* to learn to upscale images.

Figure 3 shows an example of this encoder-decoder network. In the figure, the neural network takes in a high-resolution image and starts the encoding process, passing it through different convolutional layers, compressing the image representation further at each step until it gets to the *bottleneck*, the narrowest portion of the network where the image is at its highest compression. Once it gets to the decoder, a symmetrical series of steps will try to reconstruct the full-size image from the compressed representation.

There are two stages to using this neural network. One is known as *training* while the other is known as *inference*. The training stage of this process is run on NVIDIA's supercomputers, where high-resolution frames are fed into the network and outputs are compared to inputs. The aim here is for the outputs to be as close to the inputs as much as possible. To achieve this, the aforementioned filter values are modified as well as values in the network called *weights*, which are values that represent the strength of the connections between nodes. The network looks at the output image, tweaks its filters and weights to get the output as close as possible to the input image, then repeats this process over and over again - getting better and better at recreating the original image.

This brings us to the inference stage. Here, a pre-trained network (more specifically, the decoder portion of the auto-encoder) is fed lower-resolution images, which are then upscaled using the decoder to a final, higher-resolution image. This is the portion of the process that is done on the client's computer. Since the training has already been done, the client computer only does one pass per frame through the decoder, making inference significantly less computationally demanding than training.

In DLSS 1.0, the network is trained on a *per-game* basis. So after a game is nearly done with development but is not yet released, game developers will send game data to NVIDIA, where supercomputers are used to train the model. The models are then shipped out to consumers in the form of graphics driver updates. This means that consumers will need to update their graphics drivers to the version that is necessary for the game to run, which is one of the cons of this iteration of DLSS.

In addition to being trained on regular frames, the network is also trained on frames where the image is rotated and flipped, color channels are randomized, noise is added, and/or the image is zoomed in [7]. This helps train the model on more scenarios, which can improve image quality in different in-game contexts and situations. However, it is worth noting that training the model in more scenarios than what is likely to actually occur in game can result in worse results, so there is a good balance to be reached.

In addition to current frame image data, DLSS's neural network utilizes *motion vectors*. Motion vectors are used in video game engines to represent the motion of in-game objects. DLSS can use these motion vectors to better predict the the appearance of objects in motion while upscaling.

While DLSS 1.0 was a welcome addition to make demanding games more accessible to the public, it was not without its cons. Besides the aforementioned drawback of training models on a per-game basis, 1.0 simply was not at a stage where consumers could use it without experiencing a perceptible hit to graphical fidelity. In particular, it struggled in scenarios with fast movement or minute detail, resulting in distracting visual artifacts that led to a mixed response among consumers.

A big source of the limitations that DLSS 1.0 had had to do with the fact that it was a form of image upscaling called *Single Image Super Resolution* - a form of upscaling that uses a *single* low-resolution image to generate a high resolution one [15]. This means that besides motion vectors, which don't provide additional image detail, the only other input to the network is a low-resolution image. In order to upscale the low-resolution image, the network has to create information that wasn't there in the first place, it has to *hallucinate* - a term used to describe when a neural network generates data that are not present in the original dataset, often with its outputs seeming false, unrealistic, or artificial. So, while the network can make an educated guess about the missing information, the resulting output may not always be accurate. As a result of these factors, DLSS 2.0 was introduced as a superior successor.

3.2 DLSS 2.0

DLSS 2.0 improved on the previous version by modifying previous approaches and adding others. Firstly, the neural network in this version uses data from more sources than before. This includes not only the current frame and motion vectors, but also temporal data such as previous frames, and depth buffers; as well as data about exposure and brightness of the game scene.

Secondly, DLSS 2.0 uses a model that is *fully generalized*. This means that the neural network can produce high-quality output across various games without needing to be trained on a per-game basis. With this change, the technology can scale more effectively to more games, while also making it easier for game developers to release their games without waiting for NVIDIA to train the model for each game.

Thirdly, and most importantly, DLSS 2.0 uses *Multi-Frame Super Resolution* [8]. Here, the neural network is trained to use *temporal data* (data from previous low-resolution frames) in addition to the current frame to upscale the current low-resolution frame. Due to the fact that it collects samples from previous frames, the neural network can better restore missing details that were lacking from the low-resolution image without hallucinating details that weren't there or creating visual artifacts. In addition to this, DLSS 2.0 also uses deep learning to do *anti-aliasing* in what is called Deep Learning Anti-Aliasing (DLAA). Anti-aliasing is a technique that aims to reduce the jagged appearance of curved edges in images. An example of this can be seen in Figure 4.

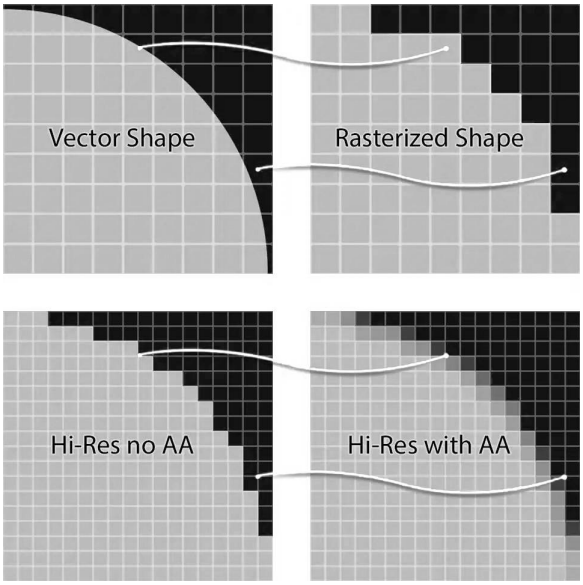

Figure 4. Anti-aliasing [11]

DLAA uses the neural network's ability to learn patterns in high-resolution images to estimate how edges should look in low-resolution images when they are smoothed out. By doing this within the upscaling process, it creates a smoother appearance of edges in the final upscaled image without the need for additional post-processing steps. After inputting the low-resolution frame into the network, the network performs upscaling while fixing the jagged edges caused by aliasing. It is able to do this because the network was trained on images with much higher resolution - therefore, it is able to make a good estimation as to how a smooth edge should look. A comparison of a vector shape, its rasterized image, as well as a higher resolution version of the image with and without anti-aliasing can be seen in Figure 4.

3.3 DLSS 3.0

Version 3.0 improves upon the previous version by introducing a new technology called *optical-flow frame generation*. Here, a third artificial frame is inserted between every two 'real' frames that smoothly transitions between the two frames. This can result in significantly higher frame rates for a given resolution, albeit with added latency. *Latency* refers to the delay between a player's action (such as pressing a button on a controller or keyboard) and the corresponding response on the screen.

Optical flow works by utilizing two consecutive frames and motion vectors just like in 2.0, however, it also takes advantage of *optical flow field*. Optical flow field refers to the recorded velocity and orientation of the movement of individual pixels as they shift from one frame to another. While motion vectors come from the game engine, optical flow is measured from rendered pixels. DLSS 3.0 takes advantage of both to better display motion as well as do a better job of predicting a third, intermediary frame. Figure 5 shows frame generation without optical flow.

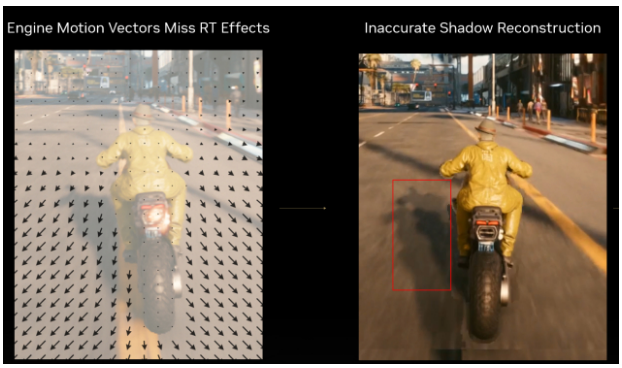

Figure 5. Frame generation with just motion vectors [6]

In Figure 5, motion vectors from the game engine indicate that the shadow is in motion, which it technically is in relation to the game world. However, it is fixed on the screen in relation to the in-game camera viewport. This results in a smearing effect known as *ghosting*, which is undesirable. Using both optical flow *and* motion vectors mitigates this issue by predicting motion for areas of the screen whose optical flow and motion vectors show movement while reducing movement in areas of the screen whose optical flow shows less motion. This can be seen in Figure 6.

Figure 6. Frame generation with motion vectors and optical flow [6]

However, since frame generation has to delay rendering a frame while it renders an intermediary frame, added *latency* is inevitable.

To help offset the added latency caused by frame generation, NVIDIA introduced *Reflex*, a set of technologies that aim to reduce input latency. This is achieved by providing game developers with a Reflex Software Development Kit (SDK). Reflex keeps the CPU and GPU in sync to minimize the number of pre-rendered frames that are generated by the CPU before being rendered by the GPU. *Pre-rendered frames* are frames that have been generated in advance by the CPU, but have to wait for the GPU to render them, causing latency. This is especially the case in situations where games are run at higher resolutions and graphics settings, where the GPU is the bottleneck. This delays user input already processed by the CPU from being rendered on screen. Reflex counters this by keeping the GPU and CPU in sync, getting rid of the frame queue and consequently, the latency caused by frames waiting to be rendered. By offsetting the latency introduced by frame generation, players can take advantage of even greater performance while keeping the latency to manageable levels.

4 Results and Performance

Different versions of DLSS have had different levels of efficacy, both in performance and visual fidelity. We will now discuss how well each version does at mimicking running a game at native resolution, as well as the performance gains in each version. For versions after 2.0, DLSS also offers three settings the user can toggle between — *Quality, Balanced, and Performance*, with each one offering 2X, 3X and 4X upscaling respectively. This means that running a game at 4K resolution on DLSS Performance settings renders the game at 1080p (a quarter of the number of pixels compared to 4K) and upscales it to 4K. We will be looking at the differences between these settings.

The original DLSS noticeably improved performance with 30-40% gains in FPS (frames per second) on average [13].

However, it was not without a significant decline in visual fidelity. Softened textures and visual artifacts such as noise and ghosting were very frequently seen, which caused DLSS 1.0 to have a mixed response among consumers. Figures 7 and 8 show the performance increase and fidelity decrease respectively below.

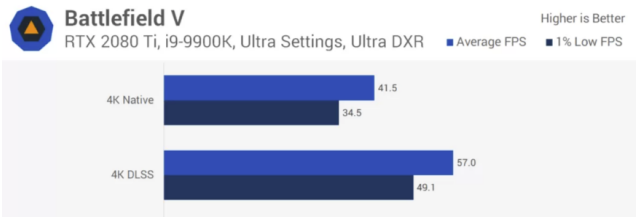

Figure 7. Comparison of Performance in *Battlefield V* at 4K, with and without DLSS[13]

In figure 8, textures on the bark of the tree on the left are much blurrier on DLSS. Significant blurring and loss of detail are also apparent on other objects in the scene, such as the definition of the leaves in the bush behind the tree, which don't appear separated in DLSS. Objects in motion also exhibited significant ghosting, leaving a trail behind them.

Figure 8. Image quality comparison with native rendering [1]

Version 2.0 addressed several of the issues present in 1.0. There were fewer visual artifacts, and the overly soft appearance of 1.0 was also minimized. The upscaler does a better job of filling in missing detail by utilizing data from previous frames. Performance in Quality mode was comparable to the previous version of DLSS [1]. Switching to balanced or performance mode offers better performance. See figure 9.

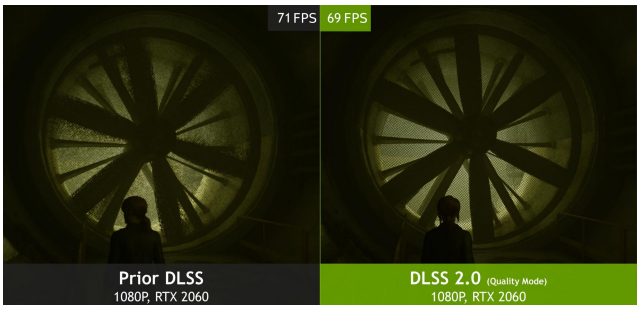

Figure 9. Image quality comparison between DLSS 1.0 and 2.0 [1]

Due to optical flow frame generation, DLSS 3.0 improves performance even further, approaching 2 and 3X the average FPS compared to native rendering in certain games and settings[9] while image quality remains comparable to DLSS 2.0. See figure 10.

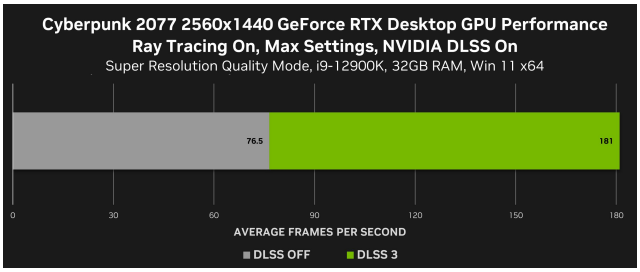

Figure 10. DLSS 3.0 on vs off comparison in *Cyberpunk 2077* at 1440p resolution [1]

5 Conclusion

The growing complexity of graphics in video games has posed a significant challenge in achieving both high-quality image output and optimal performance at the same time. The conventional way of addressing this issue is to create more powerful hardware that can keep up with the needed processing power. However, DLSS proposes a different approach to this problem by utilizing deep learning to improve performance and offer higher frame rates. It achieves this by using technologies such as single and multi-frame super-resolution, optical flow frame generation, and CPU/GPU synchronization. As games continue to demand higher and higher processing power, DLSS presents a way for modern hardware to keep up with these new demands.

References

[1] A. Burnes. 2020. NVIDIA DLSS 2.0: A big leap in AI rendering. https://www.nvidia.com/en-us/geforce/news/nvidia-dlss-2-0-a-big-leap-in-ai-rendering/. Retrieved February 19, 2023.

[2] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, and Tsuhan Chen. 2018. Recent advances in convolutional

neural networks. *Pattern Recognition* 77 (2018), 354–377. https://doi.org/10.1016/j.patcog.2017.10.013

[3] Chua Hock-Chuan. 2013. 3D Graphics Pipeline. https://www3.ntu.edu.sg/home/ehchua/programming/opengl/images/Graphics3D_Pipe.png [Online; accessed 17-March-2023].

[4] K. Kapse. 2021. An Overview of Current Deep Learned Rendering Technologies. In *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*. IEEE. https://ieeexplore.ieee.org/document/9441822

[5] Mozilla. 2023. Explaining basic 3D theory - Game development | MDN. https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_on_the_web/Basic_theory. Accessed: 2023-03-16.

[6] NVIDIA. [n. d.]. DLSS 3.0: AI-Powered Neural Graphics Innovations. https://www.nvidia.com/en-us/geforce/news/dlss3-ai-powered-neural-graphics-innovations/. [Online; accessed 14-April-2023].

[7] NVIDIA. 2019. Truly Next-Gen: Adding Deep Learning to Your Game. https://www.gdcvault.com/play/1026184/Truly-Next-Gen-Adding-Deep. Accessed on 22 March 2023.

[8] NVIDIA. 2020. High-Performance Computing and AI Innovations at GTC 2020. https://developer.nvidia.com/gtc/2020/video/s22698. Accessed on 23 March 2023.

[9] NVIDIA. 2022. Cyberpunk 2077 DLSS 3 Update Out Now. https://www.nvidia.com/en-us/geforce/news/cyberpunk-2077-dlss-3-update-out-now. Accessed on 14 April 2023.

[10] opengl notes. 2022. Aliasing. https://opengl-notes.readthedocs.io/en/latest/_images/aliasing.png

[11] RenderStuff. 2016. Antialiasing and Color Mapping in V-Ray Tutorial. https://renderstuff.com/tutorials/antialiasing-and-color-mapping-in-v-ray-tutorial-158/. [Online; accessed 14-April-2023].

[12] Ankit Sharma. 2018. Convolutional Autoencoders for Image Noise Reduction. *Towards Data Science* (2018). https://towardsdatascience.com/convolutional-autoencoders-for-image-noise-reduction-32fce9fc1763

[13] TechSpot. 2019. Nvidia RTX DLSS in Battlefield V: A Big Performance Boost, But With a Quality Cost. https://www.techspot.com/article/1794-nvidia-rtx-dlss-battlefield/. [Accessed 14-April-2023].

[14] Wikipedia. 2023. Graphics Pipeline — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Graphics_pipeline [Online; accessed 17-March-2023].

[15] Wenming Yang, Xuechen Zhang, Yapeng Tian, Wei Wang, Jing-Hao Xue, and Qingmin Liao. 2019. Deep Learning for Single Image Super-Resolution: A Brief Review. *IEEE Transactions on Multimedia* 21, 12 (2019), 3106–3121. https://doi.org/10.1109/TMM.2019.2919431