Neural Supersampling for Real-time Rendering

A Pytorch implementation (2021)

Pierre THIEL · Guillaume HAERINCK · Alexys LE SAINT

Made for Project Management class by Chaohui WANG at Université Gustave Eiffel

Abstract Realtime supersampling has the ambition to reconstruct a high resolution frame from a low resolution input created by a rendering engine. This operation needs to bear a lower computational cost than what would be done by the engine while maintaining a similar level of quality. With the power of machine learning and the proper architecture and input data, it is now possible to approach and even achieve such results. In this paper we will review the current state of the art and present our own implementation[18] of the NSRR presented by Facebook's research team.

Keywords deep learning · rendering · upsampling · superresolution

1 INTRODUCTION

Video games and similar interactive experiences have known an exponential graphics enhancement over the past decade. According to Moore's law the increase of computing power allowed the developers to have bigger and more detailed scenes with finer effects. We see each year cutting edge algorithms[7] being further improved upon by the game industry and the major vendors offering various hardware and drivers upgrades to support them. Some of these effects are even encompassing real-time raytracing, something that could only be dreamt of years ago.

Yet to support these advances on a wider range of devices it is also necessary to provide software solutions to reduce the cost of a frame. Moving some of the workload from the CPU to the GPU has been a valuable step. We also see many improvements done to dynamically vary the level of details[8] and processing based on the scene cost and camera position. Lastly there has been usage of machine learning to enhance the visual quality[4] for a lower computational cost for denoising and supersampling.

Université Gustave Eiffel, M2 image science

Fig. 1 One of the sample from the NSRR paper [1]

Deep learning super sampling (DLSS)[5] is a state of the art technology for real-time supersampling developed by NVidia. It is already in use for many high-profile video games and has proven to be more efficient than traditional methods. However as a proprietary solution there are no implementation details available to the public. A team at facebook research recently released a paper[1] presenting their architecture for a similar technology, but their algorithm wasn't released with it. In this paper, we present an open-source implementation of their algorithm using PyTorch.

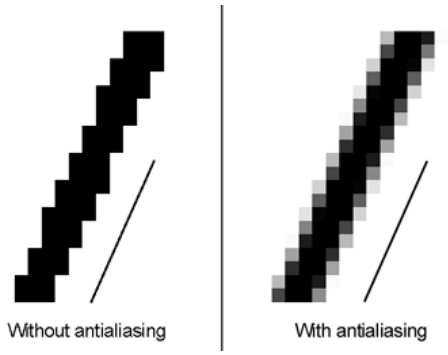
2 RELATED WORK

Real-time rendering is all about trade-offs and choices. You have a limited amount of computing power available for each frame and you have to dispatch this power wisely. On the consumer-side, we see this through the graphics settings of a game where we can adjust the several effects to match our computer power. The choice of the output resolution is one of the most impactful settings, both in terms of cost and visual quality. In the next section, we will present methods used to improve or adjust the quality of the final image when upscaling the resolution is either too costly or not feasible.

2.1 Traditional methods

2.1.1 Antialiasing

Antialiasing is a family of techniques used extensively in real time rendering. Its goal is to smooth-out the image in order to prevent the jagged edges effect. Such artifacts happen when the resolution of the image is too low, which can be also due to the hard limit from the display resolution. There is no best antialiasing algorithm as they each offer advantages in terms of quality and performance, which is why we will review the usage of the most popular ones.



One of the first anti-aliasing algorithms invented relies on upscaling the target image in order to average the pixels on the lower output resolution. The color is averaged from multiple samples taken from the source region. Multiple patterns containing more or less samples exist. Supersampling anti-aliasing (SSAA) gives great results but is very heavy on performance. The multisample anti-aliasing (MSAA) is an improvement of this technique. Instead of upsampling the whole frame, the MSAA only upsample pixels for which triangles cover enough space on the sample pattern. Further improvements and implementation of this technique are known as CSAA from NVidia and EQAA from AMD.

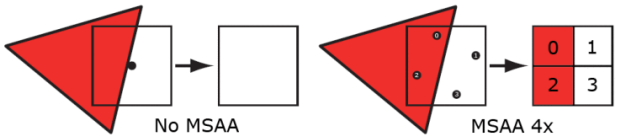


Fig. 2 MSAA typical pattern [2]

Another thread of work relies on the morphological analysis of the image. This technique has the advantage to be easier to implement as it only needs the final frame while being easier on performance. First developed by intel labs in 2009[9], the morphological antialiasing (MLAA) will look for discontinuities in the image and blend the neighborhood of these patterns. A more performant implementation is known as fast approximate anti-aliasing (FXAA)[10] and was released by NVidia in 2009. The technique has been improved twice by a team led by Jorge Jimenez in 2011[12] and finally in 2012[11] with the groundbreaking SMAA.

To further improve the results of these algorithms, in particular to ensure that there is smooth motion, some implementation relies on the previous frames. This is known as temporal anti-aliasing (TXAA) and became popular in game engines since 2010 according to Yang and Liu[13]. One popular implementation has been presented during the Siggraph 2014 conference by Brian Karis from Epic Games[14].

2.1.2 Adaptive resolution

Video games are non-linear experiences which means that the workload to process has no reason to be constant over time. This gap between heavy scenes and smaller ones can be seen as wasted processing time that could be better used to improve the quality of the game. For their popular title Fortnite, Epic Games developed an algorithm[15] to dynamically change the output resolution of the game based on the workload. Released along Unreal Engine 4.19, the newer version of the screen percentage uses temporal anti-aliasing to perform an upsampling of better quality.

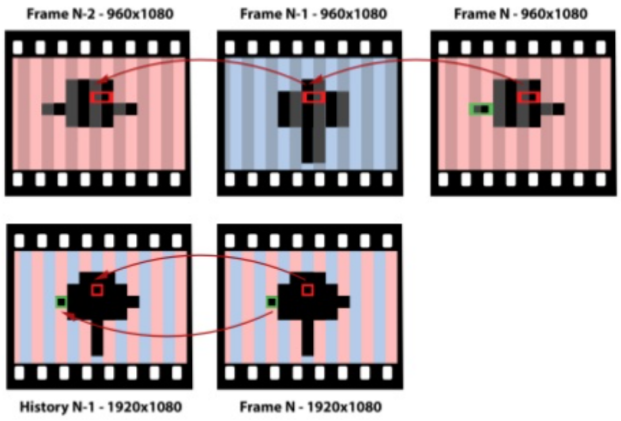


Fig. 3 Alternate rendering in Killzone [3]

AMD is offering a similar toolset through their FidelityFX[8] package which offers a variable shading algorithm capable of varying the update rate of a portion of the image based on its movements. In 2014 the game Killzone Shadow Fall made the bold move of alternating the render of odd and even pixels to gain performance in multiplayer[3]. With this technique, they sacrificed

a bit of rendering quality for a large frame rate gain. With the emergence of virtual reality, searchers found out that it was possible to render the border of the image at a lower quality without being noticed by the user, it is called foveated rendering.

While these solutions are taking place on the final image, there is also much to say about the adaptive resolution of the objects in the game world. Level of details techniques (LOD) through the use of tessellation is a well known process widely used in the industry. An improvement of this pipeline has been proposed by NVidia in 2018 through the invention of mesh shaders[16]. Some details from the new Unreal Engine 5 renderer[17] called nanite shows that with the help of this new pipeline and severe compressing algorithm it is capable of streaming a large number of high poly meshes and adapting them to the current framerate. More details are expected by the mid 2021 when the new version of the engine is released.

2.2 Machine learning methods

With an impressive breakthrough made during the 2012 ILSVR challenge[22], machine learning based methods became the new go-to for image restoration and segmentation algorithms. While these methods are not new, the novel access to large dataset and the increase of computational power through GPU gave machine learning the much-needed tools to shine. Machine learning methods share the same process (dataset transformation, training and testing) but they differ on their neural network architecture. In this section, we will review and compare popular architecture used for image supersampling.

2.2.1 Convolutional Neural Networks (CNN)

Given an input image, the role of the CNN is to reduce it into a shape that is easier to process, while conserving the features of the image used for predictions. The CNN architecture[23] is composed of a series of convolution and activation layers. During the 2018 Siggraph, Microsoft presented a CNN for real-time supersampling through their DirectML[4] suite running on the game Forza Horizon 3. With their network, they were able to upsample 1080p to 4k in real time with convincing results.

Their network is fairly small in size with 2 samples layers 6 groups of convolution-normalisation-relu activation. The role of the first layers is to extract the feature of the input image (edges, corners, ridges, etc) and outputs high-resolution patches. These patches are then mapped to a higher representation and finally reconstructed to the 3-channels output image.

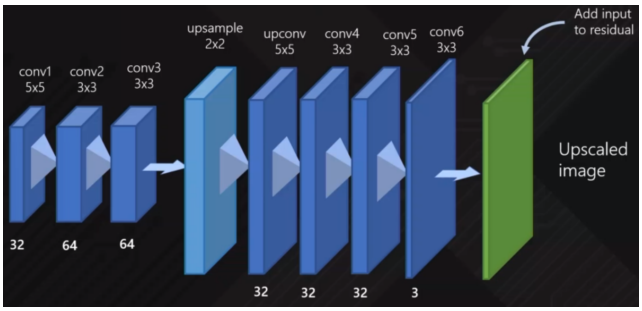


Fig. 4 Super resolution CNN architecture in DirectML [4]

2.2.2 Auto-Encoders

Auto-Encoders are made of two parts : an encoder which extracts the features of an image and a decoder which reconstructs a higher dimensional image based on the extracted features. The network is often symmetrical, and its center is notably small in dimension as the outputed features are smaller than the input image. Auto-Encoders, just like CNN, relies heavily on convolutional layers, but they also often add pooling layers to further reduce the image dimension during the encoding stage.

This architecture is used in many image restoration fields and often modified to include skip connections for the network to conserve the details of the image. This method, popularized with the U-Net paper in 2015[24], consists of connecting layers of the encoder to the corresponding layers of the decoder. As the encoder is reducing the dimension of the image, some details are inevitably lost in the procedure. By concatenating results from previous stages of the encoder to the decoder, we can recover this kind of data.

For example DeepFovea[25] in 2019 by Kaplanyan uses a U-Net with resblocks and several recurrent procedures to reconstruct foveated images. This recovery of lost details, added with the decoder structure, allows autoencoders to be more versatile than simple CNNs. They also greatly reduce training time of the network.

2.2.3 Recurrent Neural Networks (RNN)

Recurrent neural networks are used to improve the prediction in a sequence of data such as tracking a moving subject in a video. The basic idea[26] is to keep in memory a result from a specific step of the network and mix it with new input during the next epoch. This approach is great for short-term memory but lacks the ability to remember too old data. This problem is called the Vanishing Gradient and some architecture such as

long short term memory (LSTM) and gated recurrent units (GRU) have been created to handle it.

The LSTM were introduced by Hochreiter and Schmidhuber[27] as a way for the network to remember information for long periods of time. It is composed of 4 layers and 3 gates that have the role to store or remove some of the information that goes through the cell. Further information can be found on colah's blog[28]. RNN mechanisms are necessary for video based data-sets as they provide further temporal stability and more information about the overall context of the sequence.

More recent papers have refined the RNN structure into self-attention modules[29]. The role of these modules is to take into account all of the previous inputs instead of the few last ones in order to increase the weight of the most valuable input areas[30]. However this mechanism has a computational cost which can make their use problematic for real-time algorithms.

2.2.4 The case of DLSS

NVidia deep learning super sampling technology is currently the state of the art in this field by a fair margin. While information about the network architecture is limited, it is now possible to download and use the official SDK or easily install a plugin for Unreal Engine 4. To use it however, one must have an RTX graphic card which is difficult to acquire for the time being. With the low-resolution frame alongside its motion vectors and depth buffer, the network outputs a high resolution frame.

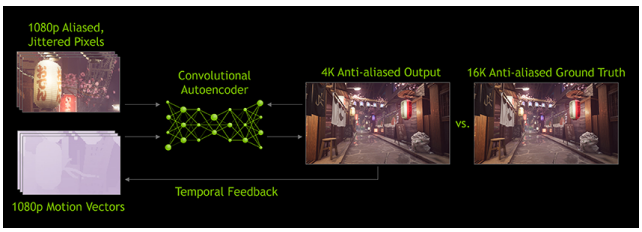


Fig. 5 The DLSS 2.0 architecture [5]

From the information made available in the SDK programming guide we know that the network is an auto-encoder with recurrent methods applied through the motion vectors. Both SDK and plugin comes pre-trained with nons-game-specific content, which means that we do not have to re-train the network to run it on our application. This fact alongside the impressive performance, runtime and ease of use makes this technology far above the current rival algorithms. AMD recently announced an addition for Superresolution in their FidelityFX package designed to be an open-sourced direct concurrent to DLSS but the said algorithm isn't available yet.

3 DATASET GENERATION

As there is currently no publicly available dataset for our use case we had to generate one from the unity game engine[19]. As for DLSS, we generate the depth buffer and motion vector alongside the rgb frame for a pair of low and high resolution. We have total control over our dataset so we could generate increasingly complex scenes to train our network on. Generating a dataset from a game engine also has the advantage of simplifying the integration of the network into an actual application.

3.1 Characteristics of Depth buffer and Motion vector

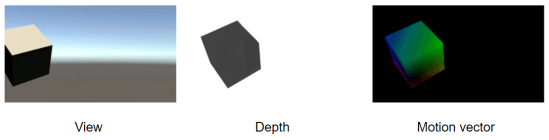


Fig. 6 An example from our generated dataset

3.1.1 Mipmap

To prevent artifacts in the render, it is important to disable the mipmap texture compression by 0 in the renderer settings. Such artifacts are particularly visible on high frequency patterns such as a meadow scene with a lot of grass.

3.1.2 Motion vector

The motion vector is generated on a per-pixel motion basis and in screen-space. The motion vector maps a pixel from the current frame to its position in the previous frame. Such movement describes movement on the x and y axis with the upper left corner as the [0, 0] coordinate and the bottom right the actual resolution (for a 1080p surface, this would be [1919, 1079]). Such movements can be negative and even represent floating values.

An HLSL implementation for the Unreal Engine is available in the DLSS programming guide p.13. In our implementation, you can decide to transfer this rgb movement into a HSV color as the NSRR paper says it provides better performances. It is notable to say that

motion vectors are also called optical flows, we haven't found a specific difference between the two terms.

3.1.3 Depth buffer

The depth buffer near the plane is 0.0 while the far plane is 1.0. By default, it is encoded in a 16bits image but it is adjustable to 32bits.

3.2 With Unity Engine

Unity offers a legacy ML-ImageSynthesis project[20] to export several types of frames from the engine. We modified and improved this project on a public repository[19]. The project is able to run on the Unity 2019 LTS with the built-in render pipeline. To record the frames, one simply has to attach a script to the camera and press play. The user can decide the framerate and the passes exported. This flexibility allows to record player driven gameplay, movement driven by scripted timeline or even randomize movements of a camera with flocking-like algorithms.

3.3 With Unreal Engine 4

Unreal engine allows to export custom passes easily and setup shaders for these custom passes. The engine offers a motion vector export by default but we found out that the generated data was incorrect as presented in this pull request[21]. We modified these shaders with custom HLSL nodes but results were still unconvincing so we decided to stick with Unity. It is important to note that the DLSS plugin by NVidia adds similar modifications to compute their motion vector as described in their documentation.

4 METHOD

While the NSRR paper bases itself on previous work, their network architecture is one of a kind and presents multiple mechanisms working with each other for an additional complexity. For the three students we are, some of the implementation details are lacking and led us to some adaptation in multiple areas of the architecture. This means that, while we will do our best to be more explicit than the original paper, some of the information we provide might differ from the original implementation.

4.1 Problem setting

Neural Supersampling for Real-time rendering has been presented by a team from Facebook Reality Labs in 2020. Their goal was to provide an open-sourced alternative for the NVidia DLSS and use it for the Oculus VR helmets in order to gain a much needed performance boost. This use case provides multiple challenges that the team had to tackle and incorporate in their algorithm, while being cautious of the cost to keep real time performances and no hardware specifics.

The first problem resides in the temporal stability of the image. Between two close images, the supersampling algorithm might change some of the patterns on the screen and make a noticeable noisy effect. The only way to handle this problem is to take into account the previous frames through the usage of motion vectors. The NSRR implementation goes even further by stacking the three previous frames and feed them into the reconstruction network.

Another problem presented by the NSRR team is that there is no available data to feed into the network to represent in detail the lighting, shading and occlusion changes that happen in a scene. To handle such cases they are passing each frame through several steps of zero-upsampling and backward warping with the help of previous frames to reweight the image into a finer result.

4.2 Network Architecture

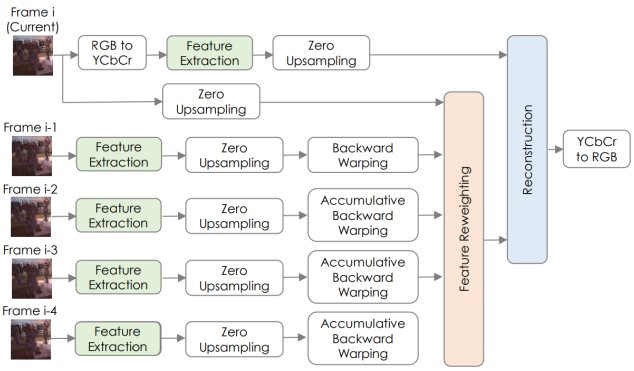
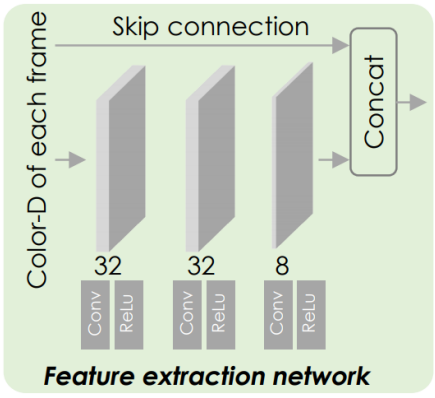


Fig. 7 The NSRR architecture as presented in the original paper [1]

The NSRR is made of 4 major modules that are called at different steps of the pipeline. There is the Feature extraction, the temporal reprojection composed of zero upsampling and backward warping, the feature reweighting and finally the reconstruction. These modules will take 1 to 4 tensors in entry as the network

has recurrent mechanisms based on the use of the last 4 frames to reweight the final results. One noticeable detail is that while the network takes in and outputs an rgb frame, it works itself in the YCbCr space so a conversion is happening at the entry and the exit of the pipeline as explained in part 5.

4.2.1 Feature extraction



The feature extraction is a simple ResBlock with an additional group convolution + ReLU right before the final concatenation. It takes a frame in entry an rgb frame with its depth value as a 4 dimensional tensor which is convoluted to 32 and 8 dimension. As it is a resblock, a copy of the entry is concatenated with the results giving a final 12 dimensional tensor for the exit. While the schema shows a direct link with this module to the next steps, we believe that this result is not used until the reconstruction module.

4.2.2 Temporal re-projection

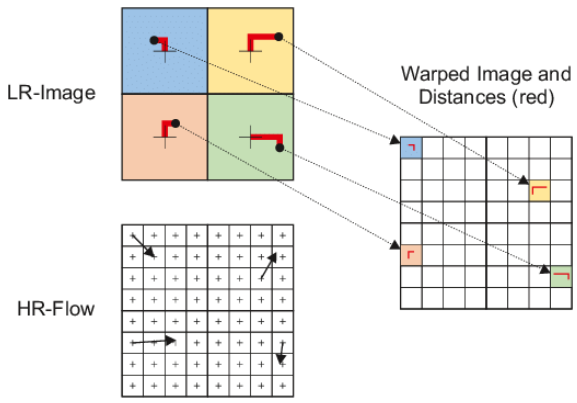


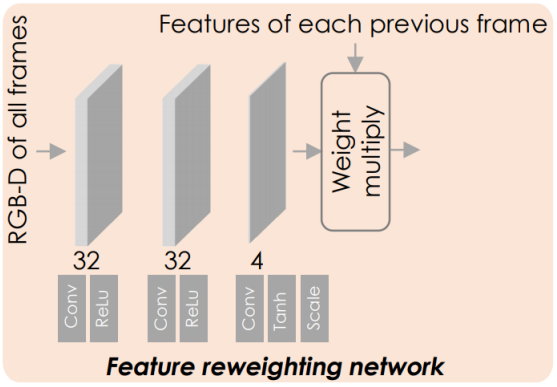
Fig. 8 Illustration of the backward warping operation [6]

Temporal re-projection is made of zero upsampling which is followed by backward warping. These are two non-machine learning methods that are used to reduce artifacts such as the occlusion of previous pixels. From our understanding this module uses the rgb network input and the motion vector image (rgb image) rather than the feature extraction output. It outputs the zero-upsampled original frame alongside the backward warped zero-upsampled frame shaped from the 4 previous frames making up a final output of 8 dimensions.

Zero upsampling is simply scaling the input tensor into a higher resolution by mapping the pixels from the low resolution to the high resolution. We leave the missing coordinates at 0 value. In our case we double the dimension size, so to get the position of the pixel [3, 2] into the high resolution we simply have to multiply by 2 which gives us [6, 4]. However for the motion vector we apply a bilinear filtering to the same scaling parameter.

The idea of backward warping is to add the pixels from the previous frame into the current one with the help of the motion vector image. The process is as follows : we look at the image coordinates, read the flow for each pixel. Then we copy the pixel value of the corresponding pixel (x + u) in that initial coordinate (x) in the resulting frame. In the network this step is done iteratively for the 4 previous frames.

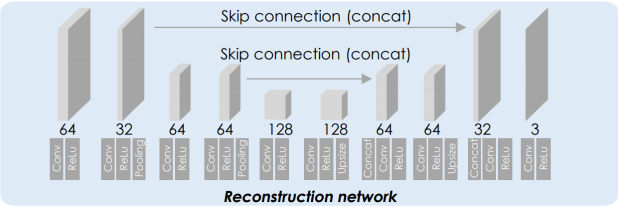
4.2.3 Feature reweighting



The feature reweighting module allows us to take into account dynamic disocclusions and shading changes between frames. It takes as input a 8 dimensional tensor shaped from the two rgb-d images from the temporal reprojection, applies 3 convolution and finally outputs a 4 dimensional tensor with values remapped from [-1, 1] to [0, 10] values. A step called weight multiply is then applied to this output, we are unsure about the exact process but we believe that it is a multiplication

between this tensor with the 4 previous outputs from this module kept in memory.

4.2.4 Reconstruction



The reconstruction network is a modified u-net architecture which takes as input a 8 dimensional tensor shaped from the concatenation between the feature reweighting output and the feature extraction output and finally outputs an rgb image. This network is a 10 layer symmetrical structure made of groups of 2 blocks of convolution + ReLU with a final pooling/upsampling. The pooling / upscale factor is 2 and the bottleneck is at 128 dimensions. We had problems of dimensions when we tried to apply the concat step after each upsize so we had to crop the tensors in order to match the right dimensions, which is a common step for a u-net structure.

4.3 Color space conversion

As explained before the NSRR team found out that the network gave better results when working with input images in the YCbCr color space (around 0.1dB improvements in PSNR), so the final step is to convert the reconstruction output to an RGB frame.

5 RESULTS

We weren't able to finish and test our whole network in time due to external constraints and the project internal complexity. We developed[18] every module presented in the paper and linked them together as they were supposed to be. However we know that some parameters for our layers in the u-net for the reconstruction layers are incorrect due to runtime errors and we are also unsure about our reweighting module implementation. There is not much to fix but we simply weren't able to do so yet.

We were also unable to test DLSS as our team members weren't able to find an RTX card in their surroundings, but we did test some supersampling algorithms without further comparison as our main focus was to finish our implementation. What we found out is that while machine learning methods were able to provide results of quality, they need the proper hardware to be run on. The Microsoft DirectML sample only ran at 10fps on a GTX 1060 for an 4x upscale while the same with bilinear filtering would be outputted at 300fps for at lesser quality but not justifying this gap in performance.

The unit test for our network implementation also did run for more than 16ms, breaking the real-time constraint that the network is supposed to bear. In comparison the SMAA implementation by Jorge Jimenez is only taking 1ms of rendering time with the same hardware. But with the proper hardware, machine learning supersampling does create wonder. From the Unreal Engine livestream and several other sources, we can see that toggling between the TAA and the DLSS offers a performance gain of over 40

6 CONCLUSION

Realtime supersampling has seen some incredible advances in the past years and it is certain that such a process will become ubiquitous in the near future. However to support these algorithms the target audience needs a specific set of hardware which only reserves such technologies to the high-end consumers and the last console generation. We feel frustrated to haven't been able to fix the remaining stages of our implementation but we are at the same time satisfied by the amount of information discovered for this work to happen.

References

1. Lei XIAO, Salah NOURI, Matt CHAPMAN, Alexander FIX, Douglas LANMAN, Anton KAPLANYAN Neural Supersampling for Real-time Rendering , Facebook Reality Labs (2020)
2. Akenine-Mller, Tomas and Haines, Eric and Hoffman, Naty, Real-Time Rendering, Fourth Edition, 2018, A. K. Peters, Ltd.
3. Michal Valient, Taking Killzone Shadow Fall Image Quality Into The Next Generation, 2014
4. Adrian Tsai, Accelerating GPU inferencing with DirectML and DirectX 12, Siggraph 2018, NVidia
5. Andrew Burnes, NVIDIA DLSS 2.0: A Big Leap in AI Rendering, 2020, NVidia
6. Osamal Makansi, End-to-End Learning of Video Super-Resolution with Motion Compensation, 2017
7. Advances in realtime rendering in 3D graphics and games, Siggraph
8. Fidelity FX, AMD
9. MLAA: Efficiently Moving Antialiasing from the GPU to the CPU, Intel, 2009
10. Timothy Lottes, FXAA, NVidia, 2009

11. Jorge Jimenez, Jose l. Echevarria, Tiago Sousa, Diego Gutierrez, SMAA: Enhanced Subpixel Morphological Antialiasing, 2012, Crytek
12. Jorge Jimenez, Belen Masia, Jose l. Echevarria, fernando Navarro, Diego Gutierrez, Practical Morphological Anti-Aliasing, 2011, Lionhead Studios
13. Lei Yang, Shiqiu Liu, Marco Salvi, A Survey of Temporal Antialiasing Techniques, NVidia, 2020
14. Brian Karis, HIGH-QUALITY TEMPORAL SUPER-SAMPLING, 2014, Siggraph, Epic Games
15. Screen Percentage with Temporal Upsample, Unreal engine 4 documentation
16. Christoph Kubisch, Introduction to Turing Mesh Shaders, 2018, NVidia
17. Unreal Engine 5 Revealed, 2020, Epic Games
18. Pierre THIEL·Guillaume HAERINCK·Alexys LE SAINT, NSRR-PyTorch, 2021
19. Pierre THIEL·Guillaume HAERINCK·Alexys LE SAINT, Unity ML Dataset, 2021
20. Image Synthesis for Machine Learning, Unity, 2017
21. Fix velocity computed from camera motion, github, Epic games
22. Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks , University of Toronto, 2012
23. Vincent Dumoulin, Francesco Visin, A guide to convolution arithmetic for deep learning, Université de Montréal, 2018
24. Olaf Ronneberger, Philipp Fischer, Thomas Brox, U-Net: Convolutional Networks for Biomedical Image Segmentation, University of Freiburg, 2015
25. Anton S. Kaplanyan, Anton, Sochenov, Thomas Leimkuhler, Mikhail Okunev, Todd, Goodall, Gizen Rufo, DeepFovea: Neural Reconstruction for Foveated Rendering and Video Compression using Learned Statistics of Natural Videos, Facbook reality labs, 2019
26. Michael Phi, Illustrated Guide to Recurrent Neural Networks, Medium, 2018
27. Seep Hochreiter, Jurgen Schmidhuber, Long short-term memory, 1997
28. Colah's blog, Understanding LSTM Networks, 2015
29. Prajit Ramachandran, Niki Parmar, Ashish Vaswani, Irwan Bello, Anselm Levskaya, Jonathon Shlens, Stand alon self attention in vision models, Google Research, 2019
30. Diganta Misra, Attention Mechanisms in Computer Vision: CBAM PaperspaceBlog, 2020