Adaptive Encoder Settings for Interactive Remote Visualisation on High-Resolution Displays

Florian Frieß* Mathias Landwehr Valentin Bruder Steffen Frey Thomas Ertl

University of Stuttgart, Germany

ABSTRACT

We present an approach that dynamically adapts encoder settings for image tiles to yield the best possible quality for a given bandwidth. This reduces the overall size of the image while preserving details. Our application determines the encoding settings in two steps. In the first step, we predict the quality and size of the tiles for different encoding settings using a convolutional neural network. In the second step, we assign the optimal encoder setting to each tile, so that the overall size of the image is lower than a predetermined threshold. Commonly, for tiles that contain complicated structures, a high quality setting is used in order to prevent major information loss, while quality settings are lowered for others to keep the size below the threshold. We demonstrate that we can reduce the overall size of the image while preserving the details in areas of interest using the example of both particle and volume visualisation applications.

Index Terms: Human-centered computing—Visualization—Visualization application domains—Scientific visualization; Machine learning—Machine learning approaches—Neural networks

1 INTRODUCTION

Remote visualisation is a common practice nowadays, when it comes to displaying large amounts of data, mainly resulting from simulations that are run on large-scale supercomputers. A typical scenario in remote visualisation is to render the simulation on a cluster and send the encoded images to a client where they are decoded again. Despite recent advances, visualising and analysing such simulations remotely at interactive rates remains to be a challenge for high-resolution set-ups. One of the main challenges when using this approach is maintaining a low latency, especially if the data shall be interactively explorable. A natural approach to reduce latency is to send less data, for instance by strongly compressing the images with encoding settings that produce a low-quality output. However, using low-quality encoding settings may greatly impact the user experience (e.g., loss of fine details).

To mitigate this, we present an approach to dynamically adapt encoder settings on a per image-tile-basis, for each display node, to optimise the quality for a given bandwidth. Our goal is to allow for high-quality screen capture sessions between two high resolution powerwalls, i.e., to preserve areas that contain fine structures and reduce the amount of encoding artefacts in those areas, while highly compressing uniform areas. For this, we split the rendered frame of each display node into several, equally large tiles. We then use a convolutional neural network (CNN) to predict (1) the size of each tile after encoding as well as (2) the quality in terms of similarity to the original image. We do this prediction for several different encoding settings.

Based on those predictions, we optimise the encoding process for maximum quality (i.e., similarity to the original image), under the constraint of the available maximum bandwidth. After the encoding settings have been determined, all tiles are encoded on GPUs and then sliced to fit into UDP packets. The client receives, and if necessary reorders or drops the packets containing the slices via a point-to-point network connection. Once an encoded tile of one display node is completely received, it is decoded and added to the final image that is then displayed.

*e-mail: florian.friess@visus.uni-stuttgart.de

2 RELATED WORK

Remote visualisation has become an essential component in research concerned with big data and high-performance computing (HPC). Especially the growing size of data sets as well as increased node numbers in HPC have contributed to a decrease in practicability of local visual data analysis for large scale simulations. Several approaches and systems for remote visualisation have been proposed in recent years [3, 14]. There is also a sizable body of work concerned with mitigating bandwidth limitations. One direction is the use of adaptive sampling techniques [1, 10], while others focus on using image compression techniques [5,8]. Pajak et al. [13] use augmented video information to efficiently compress and stream images of dynamic 3D models. Moreland et al. [12] present an approach targeted towards visualisation, where they use level-of-detail techniques to provide interactive rendering regardless of the network performance. Also targeted towards scientific visualisation in a remote setup is the technique presented by Frey et al. [4]. They integrate sampling and compression techniques to balance visualisation and transfer to optimise image quality.

CNNs have proven to be very good at classification and localisation tasks [6, 9], but can also be used to predict the quality of images. Several approaches have been proposed for this task. Li et al. [11] developed an image quality assessment algorithm that utilises a regression neural network. They use their technique to predict image quality that is relative to human subjectivity by applying a range of different distortion types. Kang et al. [7] use a CNN to predict image quality without a reference image, instead they use image patches as input to their network. Furthermore, they combine feature learning and regression as well as an optimisation process to estimate image quality in terms of human perception. A similar approach was proposed by Bosse et al. [2]. They use a deep neural network to predict image quality that works close to human perception, also feeding image patches into the network. While all of these techniques use neural networks for image quality predictions, their goal is different from ours. Our objective is to use the predictions in order to determine the optimal encoder setting with respect to a bandwidth limit, while their main goal is to judge the perceptual visual quality.

3 METHOD

Our algorithm to share the screen of our powerwall remotely, consists of the following four steps (Fig. 1), which will be detailed below:

1. Capture the last rendered frame
2. Split the frame into tiles and convert them to the *NV12* image format
3. Determine the encoder setting for each tile
4. Encode the tiles and send them to the client

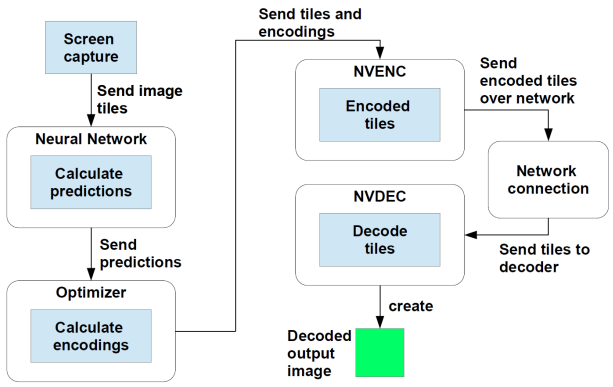


Figure 1: Overview of the pipeline of our algorithm. The screen capture part acquires the images, then they are subdivided into tiles and passed on to the neural network and the optimiser. Based on the determined predictions the tiles are encoded and sent to the client where they are decoded and displayed.

The first step is to acquire the last frame that was rendered on every display node. Our display nodes have a resolution of 1200×4096 and use the portrait mode, therefore their original resolution is 4096×1200. We use the Desktop Duplication API by Microsoft that outputs a texture with the content of the last frame. The pixel data within the texture has the format *BGRA*. This texture has the size 4096×1200 but the content is rotated counter-clockwise by 90 degrees, since the display nodes use the portrait mode.

The second step is to split the texture into tiles and to convert each tile from *BGRA* to *NV12*. Since the tiles of size 512×240 do not cover the whole rotated texture we rotate the texture back to the resolution, 1200×4096, of the display nodes while converting the colour format. This results in 40 tiles per display node and 400 tiles for the whole powerwall. In addition to the conversion of the image format to *NV12*, we also need to convert the tiles to greyscale since our CNN works on greyscale images. To lower the time needed for predictions we batch the input tiles for the network to multiple bigger tiles. The colour conversions are done on the GPU using a compute shader. The input of the shader is the acquired texture from the first step and the output is a texture array with 40 sub-textures of the size 240×768 and a second texture array with 2 sub-textures of the size 4800×768. The first texture array contains the tiles in the *NV12* format and the second texture array contains the greyscale tiles. The second texture array is used as input for the neural network and contains a batch of 20 tiles. We use the fact that the *NV12* format contains the luminance in the first $width \times height$ pixel and use that as the greyscale representation of the tile.

The third part in our algorithm determines the encoder setting for each tile. For this, we first use a CNN to predict the quality and size of the encoded tiles for three encoding settings: *LOW*, *MEDIUM* and *HIGH* (see Table 1). We then use an optimiser to determine the best possible quality for all tiles based on those predictions and the threshold for the overall size of the tiles

The fourth step is the encoding of the tiles based on the results of our optimiser. For the encoding we use the *NVENC* API by Nvidia that allows for h264 encoding on the GPU. We create three encoders with the settings *LOW*, *MEDIUM* and *HIGH* and provide each with a queue that contains the converted tiles for the encoder. The tiles are queued and encoded in parallel in order to keep the latency as low as possible. The encoded tiles are sliced by the encoder and each slice is sent over to the client via a dedicated node in the cluster, henceforth *streaming node*. The *streaming node* is connected to the display nodes via an infiniband network and uses MPI for communication. It receives the slices from all nodes and forwards them using point-to-point UDP connections. Each packet contains the binary data of the slice and a header with additional information. The header consists of the packet number, the timestamp, the numbers of the frame and the slice as well as the overall count of slices for that frame. Both, the timestamp and the packet number, are used to reorder the packets on the client side. On the client the sliced tiles need to be combined to one frame again based on the frame number, the slice number and the slice count for that frame. There are multiple queues that each provide a buffer where the slices are combined to complete frames again. Every received slice is copied to the corresponding buffer in the queue, the offset is determined by the slice number. Once a frame is complete, it is passed on to the *NVDEC* API that handles the decoding. There is one decoder for every tile of the original image, i.e. 400 decoders in our setup. If a frame is incomplete because of packet loss, it is dropped and the buffer of that queue is used for the next frame. The decoded tiles are copied into the display texture based on their ID, the tile in the top left corner has the ID 0. Based on their ID and the number of tiles cnt_x and cnt_y in the x and y-direction, the x and y position on the image texture can be determined via $x = ID \ \% \ cnt_x$ and $y = \left\lfloor \frac{ID}{cnt_y} \right\rfloor$, respectively.

Table 1: Overview of the different encoder settings.

setting	preset	CONSTQP	avg. bitrate	max. bitrate
LOW	*DEFAULT*	40	645120	645120
MEDIUM	*HIGH QUALITY*	30	7741440	7741440
HIGH	*LOSSLESS*	20	17418240	69672960

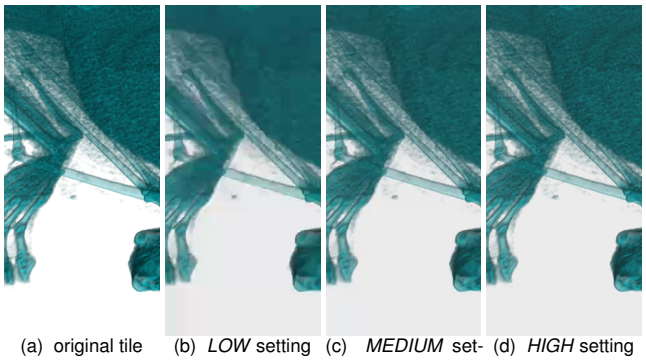
3.1 Encoder Settings

We use three different encoder settings, Table 1 gives an overview of their properties. The three settings produce encoded images of different quality and size. The *LOW* setting produces images with a severe loss of quality but a very small size. This setting should therefore only be used for tiles containing very little or no fine structures, otherwise details are lost. The *MEDIUM* setting is a compromise between quality and size. While there is still a loss of quality, it is not as severe as when using the *LOW* setting. It should be used for tiles that are between *LOW* and *HIGH* tiles based on the amount of structures they contain. The *HIGH* setting performs a lossless encoding and therefore produces the best quality but also the largest images sizes. For bandwidth optimisation, it should only be used for tiles containing fine structures to prevent information loss. All encoding settings use the constant quality mode, therefore the entire frame is encoded using the *CONSTQP* value. This value can be set in the range of $[0,51]$ with 0 giving the best quality and 51 the worst. For the *LOW* setting, we adapted the *DEFAULT* preset of the *NVENC* API and changed the *CONSTQP* value to 40, thus reducing the quality and the size of the encoded frame. For the *MEDIUM* setting we adapted the *HIGH QUALITY* preset and changed the *CONSTQP* value to 30, therefore achieving a medium-quality frame. For the *HIGH* setting we adapted the *LOSSLESS* preset. The *CONSTQP* value we use is 20, which is sufficient to preserve the structures in the tile while not increasing the size of the encoded tiles too much.

Fig. 2 shows a comparison between the three settings and the original tile. The *LOW* setting reduces the size of the tile but all of the finer structures are lost. With the *MEDIUM* setting the quality is better but there are still structures missing. These structures are only present in the tile that was encoded using the *HIGH* setting.

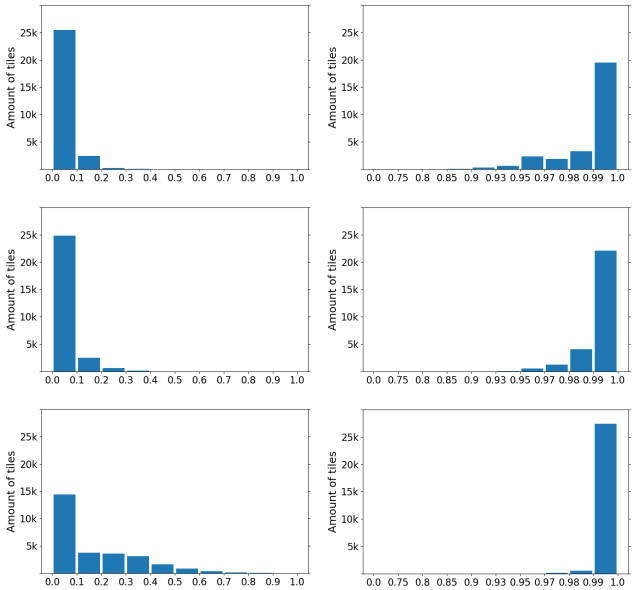
3.2 Prediction of Compressed Tile Size and Quality

The major part of the algorithm is to find an optimal encoding setting for each tile based on the quality and size. In order to achieve this, we need to analyse the different tiles and check if they contain a lot of fine structures. For that purpose, we need an algorithm that is able



(a) original tile (b) *LOW* setting (c) *MEDIUM* setting (d) *HIGH* setting

Figure 2: Comparison between the three different encoding settings.



(a) normalised tile sizes (b) SSIM values

Figure 3: The distribution of the SSIM values and the normalised tile sizes for the encoding settings. The rows show the *LOW*, *MEDIUM* and *HIGH* setting from top to bottom.

to predict the quality of images after encoding as well as the size of the encoded image. We decided to use CNN-based regression to predict the size and quality of an image tile for each encoder setting. The output of our network is a vector containing six values, one for the expected quality and one for the expected size after the encoding with each of the three settings. In combination with our optimiser we can decide on the encoder setting for each tile.

We use the the structural similarity index (SSIM) to assess image quality by comparing the original to the encoded image [15]. SSIM takes the luminance, contrast and also the structures of two images into account in order to compute a numerical closeness indicator. This indicator can be rescaled into the range $[0,1]$, with 1 representing an identical image and 0 the complete opposite.

For the training of the network we used 22560 training images from different volume rending simulations and 5632 evaluation images. Each of these images was divided into tiles with the size of 240×512 and then each tile was encoded three times using the three different encoder settings. For each of those three, we computed the SSIM values with respect to the original tile. In addition to the SSIM values we also stored the size of the encoded images. Both

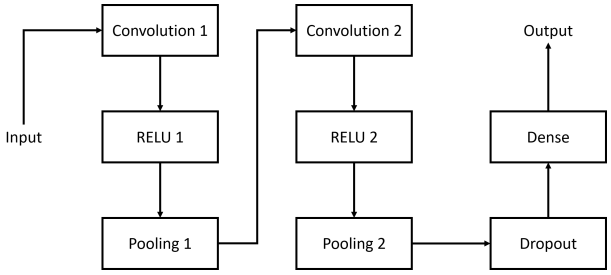


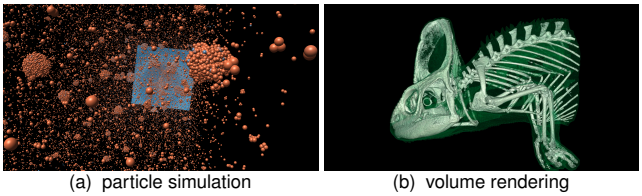
Figure 4: Overview of all layers of the convolution neural network we use to predict the quality and size of the tiles.

were used as labels during the training phase of the network. Fig. 3 shows the SSIM value distribution of the training data. It is clearly visible that, for the *HIGH* encoding setting, the SSIM values are between 0.95 and 1.0, while most tiles are between 0.99 and 1.0. For the other two settings the range is bigger and less tiles are close to 1.0, i.e. equal to the original tile.

Our network consists of two pairs of convolution and pooling layers as shown in Fig. 4, followed by the dropout and dense layer. The network is deliberately kept simple, in order to reduce the time needed to predict the encoder setting for all tiles. We use tiles of the size 240×512 as input and the output is a vector with six values, the predicted SSIM and size values for each encoder setting. Since our network was trained with greyscale tiles, the input also needs to be greyscale. Since they are already converted to the *NV12* image format during the splitting step, we use the luminance part of the converted tiles as the grey value representation. To further reduce the time needed for predictions, the tiles can be combined to batches, i.e. combining multiple tiles into one bigger input. This batch is of the size $t * 240 \times 512$ with t being the number of tiles in the batch. The first layer (Convolution 1) performs a convolution with a kernel size of 7×7 and uses 32 kernels, then a RELU (RELU 1) activation function is applied, resulting in a feature map of size $32 \times 240 \times 512$. The first pooling layer (Pooling 1) performs a maxpooling with a kernel of size 4×4 pixel and stride of 4, effectively reducing the size to a batch of $32 \times 60 \times 128$. This allows the network to still detect features that are in the general area of an image patch instead of requiring it to be at the exact location, therefore speeding up the computation time by reducing the size of the feature map. Afterwards the second convolution layer (Convolution 2), as well as a RELU (RELU 2) activation function, is applied. This time with 64 kernels but the same size as in the first convolution layer, resulting in a feature map of the size 60×128. The second pooling layer (Pooling 2) applies a 4×4 kernel reducing the size to $64 \times 15 \times 32$. In the first dense layer the output of the second pooling layer is flattened to a one dimensional vector containing 2048 units of 1 pixel each. Additionally a dropout layer drops 20% of the activations to prevent overfitting. The last layer is another dense layer that condenses the vector to the desired output size of six values.

3.3 Optimisation of Encoder Settings

Our goal is to optimise the quality of the tiles for a given size threshold T, i.e. we want to have the highest possible encoding setting per tile while keeping the overall size of the tiles below the threshold. This is equivalent to the multiple-choice knapsack problem and can be optimally solved by minimising the objective function: *minimise* $\sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x_{ij} * (1 - SSIM_{ij})^2$. With N being the number of tiles and M the number of encoding settings. The first constraint arises from the fact that we can take exactly one setting for each image tile: $\forall_j \in N : \sum_{i=0}^{M-1} x_{ij} = 1, x \in \{0,1\}$. The second constraint restricts the overall size of all tiles to be less or equal to the given threshold: $\sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x_{ij} * SIZE_{ij} \leq T$ Since



(a) particle simulation (b) volume rendering

Figure 5: On frame from each test.

we want to solve this problem faster than the ILP we use a greedy approach that approximates the optimal solution. For this, we first optimise the quality locally on each display node by taking all the predicted SSIM and size values into account. The optimiser first sets the encoding setting of all tiles to *HIGH* and sorts the tiles, in descending order, according to the SSIM value of the *MEDIUM* encoding setting. Then for all tiles that have an SSIM value bigger than the defined threshold of 0.975 the encoding setting is reduced to *MEDIUM*. After that, the tiles are sorted again in descending order, according to the SSIM value of either *LOW* or *MEDIUM*, depending on the current encoding setting of the tile. This process is repeated until there are no tiles left where the encoding setting can be reduced without loosing too much quality. Then, the overall size of the tiles is computed based on the predicted sizes and the assigned encoding settings. If this value is above the size threshold, the encoding settings of the tiles are reduced until the condition is met. For this, the optimiser uses the tiles with the lowest difference in the predicted SSIM values between the current encoding setting and the setting one level below that. This ensures that we keep the best possible quality while staying below the given size threshold.

4 Results

We implemented a prototype of our approach and evaluated its performance by transferring images between two workstations. We used one of the display nodes as the source, which is equipped with an Intel Xeon E5-2640 v3 CPU with 256GB RAM and a Quadro M6000 GPU and changed the resolution to 4096×1200 pixel, in order to simulate a small powerwall. We performed two tests and measured the bandwidth of the connection between the display node and the client machine, see table 2. In addition to the bandwidth test we analyze the performance of our prototype and the quality it delivered as well as how good the settings where adapted by the network and optimiser. For both test we target a maximum bandwidth of $100\frac{Mbits}{s}$, and determine threshold T on this basis.

The first test uses 40 seconds of a particle simulation that depicts a laser firing on an aluminium block. Numerous small particles and bigger clusters are separated from the block, i.e., there are lots of small details in this simulation that should be preserved (cf. Fig. 5 (left)). We looped the simulation for two minutes to measure the average, minimum and maximum bandwidth over time steps of one second. The *LOW* encoding setting does not preserve many details and the final image has an SSIM value of 0.90. The *MEDIUM* and *HIGH* setting perform much better when it comes to preserving the details of the simulation with an SSIM value of 0.92 and 0.96 respectively. Our prototype preserves most of the details with the SSIM value of 0.93. Fig. 6 shows a comparison between the *HIGH* setting and our algorithm. For that part of the whole image there is no visible difference between the two because our algorithm chooses the *HIGH* settings for the tiles in that part of the image.

The second test is a thirty second long interaction with a volume rendering that we also looped for two minutes. The rendered data set is a high resolution CT-scan of a chameleon, Fig. 5 (right) shows an example image of the sequence. Renderings in this sequence contain a mix of more detailed areas and areas with less details, which should be optimal for our algorithm.

Table 2: Overview of the average (avg) and maximum (max) bandwidth, in MBit/s needed for the tests.

Test	*LOW*	*MEDIUM*	*HIGH*	Our algorithm
Particle (avg)	29.19	86.63	310.92	26.35
Particle (max)	46.13	133.98	544.55	42.21
Volume (avg)	4.00	13.90	143.23	10.83
Volume (max)	6.49	22.18	255.94	21.72

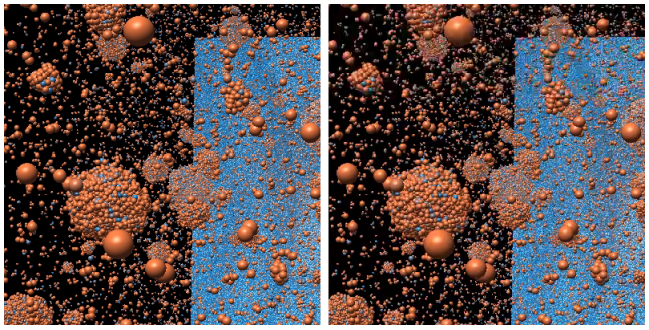


Figure 6: Comparison between the *HIGH* setting and our algorithm for a cropped part of the whole frame. Differences between the two images only become apparent at high zoom levels (cf. Fig. 7).

Table 2 gives an overview of the average and maximum bandwidth needed for both test cases, when using our technique compared to using the three encoding settings globally. We achieve a frame rate between 15 and 20 frames per second, compared to the 30 frames per second for the other three encoding settings. On average the node requires $50ms$ to capture, subdivide, convert and predict the encoding settings. Predicting the quality and size for all tiles is the most time consuming step, which requires $40ms$ for all tiles, the optimiser on the other hand requires less than $0.01ms$. The adaptation of the settings for a four minute run of the particle dataset produces on average 28.76% *LOW* encoded tiles, 69.08% *MEDIUM* encoded tiles and 2.16% *HIGH* encoded tiles.

Table 3: Analysis of the accuracy of the trained CNN for predicting quality (SSIM) and tile size.

SSIM	MSE	AAE	STD	max AAE
LOW	$2.43 * 10^{-5}$	0.0026	0.0041	0.0576
MEDIUM	$1.19 * 10^{-5}$	0.0018	0.0029	0.0308
HIGH	$5.38 * 10^{-6}$	0.0016	0.0016	0.0156
tile size	MSE	AAE	STD	max AAE
LOW	$7.95 * 10^{-5}$	0.0046	0.0076	0.1134
MEDIUM	0.0001	0.0049	0.0093	0.1566
HIGH	0.0011	0.0180	0.0281	0.2279

Table 3 shows the mean squared error (MSE), the average absolute error (AAE), the standard deviation (STD) of the AAE and the maximum AAE for both, the SSIM values and the tile sizes. It can be seen that MSE and AAE decrease for the SSIM predictions with higher quality, but increase for the tile size predictions with higher quality. The main reason for this is the range of occurring SSIM values, which are close to 1.0. The tile sizes on the other hand are mostly located at the lower end of the spectrum, due to the normalisation process, and have a bigger pool of potential values which also results in higher error values (see Fig. 3). We cannot directly calculate the accuracy for a regression model. The closest we can get is an approximation for the possible error rate by taking the potential value range and the average absolute error into account.

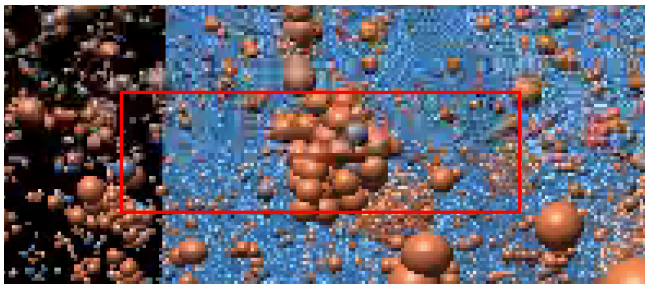


Figure 7: Zoom in on the boundary between two tiles (highlighted by the red rectangle). Enlarging the area by 500% allows to see the transition between a tile encoded with the *HIGH* setting (bottom tile) and the *LOW* setting (top tile).

For all 6 values the approximated error rate is roughly around 5%. So the network is by no means perfect, but the results are good enough to calculate the possible SSIM values and tile sizes given the fact that it also has to predict as fast as possible.

5 Discussion and Conclusion

Our prototype shows that we can achieve a good overall quality while preserving details in areas of interest. For the first test it uses the lowest average and maximum bandwidth while delivering a slightly better overall quality than the *MEDIUM* setting. The *MEDIUM* encoding setting for the whole input image requires $3\times$ the bandwidth of our algorithm on average, while producing a similar output. The difference between the used bandwidth comes from the fact that our algorithm chooses the *LOW* encoding setting for some tiles in order to save bandwidth. This allows it to use the *HIGH* setting for tiles with fine structures to improve the overall quality of the final image. Only the *LOW* setting and our algorithm stay below the target of $100\frac{Mbits}{s}$ during the tests. The *MEDIUM* setting is above that limit from time to time and the *HIGH* setting is above the limit for both tests. The lower average and maximum bandwidth in the first test, compared to the *LOW* setting stems from the fact that the optimisations of the h264 encoder are used for each tile individually and therefore further reduce the size of the encoded tiles. For the second test only the *HIGH* encoding setting is over the limit. The low bandwidth needed by all settings comes from the fact that for most of the time only a small part of the image contains structures. This time the *LOW* encoding setting has a lower average and maximum bandwidth than our algorithm but the quality is also much lower. Again, we achieve a quality that is between the quality of the *MEDIUM* and *HIGH* encoding setting, while using less bandwidth than these two settings. The boundaries between tiles of different settings produce artefacts but they only become clearly visible at high zoom levels. Fig. 7 shows the boundary between two tiles, one encoded with the *HIGH* and the other encoded with the *LOW* setting. In order to reduce these artefact we could enforce that settings of neighbouring tiles only differ by one level, i.e. no neighbouring tiles have *HIGH* and *LOW* settings.

We developed a novel approach that dynamically adapts encoder settings for image tiles. Our method uses a CNN to predict the quality and size of image tiles. It uses that information to determine the optimal setting for each tile in order to yield the best possible quality for a given bandwidth. One major contribution of our approach is that we use the predictions of a CNN to adapt the encoder setting for each image tile separately. Our algorithm works independent of the displayed visualisation, since it uses the structure of the image tiles to predict the quality. In contrast to adaptive compression approaches in JPEG2000, our algorithm uses a quality metric to directly determine the encoding setting compared to the heuristic approach of the JPEG2000 technique. With respect to staying below a certain threshold, we only need to encode once while typical JPEG2000 implementations rely on iteratively encoding until the criterion is satisfied. Finally we showed the feasibility of our method, using a prototype of the algorithm, for two different test cases. In the future, we want to evaluate the prototype to work with our powerwall so that we can achieve the goal of sharing the full resolution of 10800×4096 with a second powerwall while preserving the quality of the images and keeping the bandwidth to a minimum.

Acknowledgments

The authors would like to thank the German Research Foundation (DFG) for supporting the project within projects INF and A02 of SFB/Transregio 161, and the Cluster of Excellence in Simulation Technology (EXC 310/1) at the University of Stuttgart.

References

[1] M. R. Bolin and G. W. Meyer. A frequency based ray tracer. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pp. 409–418. ACM, 1995.

[2] S. Bosse, D. Maniry, T. Wiegand, and W. Samek. A deep neural network for image quality assessment. In *2016 IEEE International Conference on Image Processing (ICIP)*, pp. 3773–3777, 2016. doi: 10.1109/ICIP.2016.7533065

[3] J. Diepstraten, M. Gorke, and T. Ertl. Remote line rendering for mobile devices. In *Computer Graphics International*, pp. 454–461. IEEE, 2004.

[4] S. Frey, F. Sadlo, and T. Ertl. Balanced sampling and compression for remote visualization. In *SIGGRAPH Asia 2015 Visualization in High Performance Computing*, p. 1. ACM, 2015.

[5] R. Herzog, S. Kinuwaki, K. Myszkowski, and H.-P. Seidel. Render2mpeg: A perception-based framework towards integrating rendering and video compression. In *Computer Graphics Forum*, vol. 27, pp. 183–192. Wiley Online Library, 2008.

[6] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. *CoRR*, abs/1709.01507, 2017.

[7] L. Kang, P. Ye, Y. Li, and D. Doermann. Convolutional neural networks for no-reference image quality assessment. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1733–1740, 2014. doi: 10.1109/CVPR.2014.224

[8] D. Koller, M. Turitzin, M. Levoy, M. Tarini, G. Croccia, P. Cignoni, and R. Scopigno. Protected interactive 3d graphics via remote rendering. In *Transactions on Graphics (TOG)*, vol. 23, pp. 695–703. ACM, 2004.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds., *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012.

[10] M. Levoy. Volume rendering by adaptive refinement. *The Visual Computer*, 6(1):2–7, 1990.

[11] C. Li, A. C. Bovik, and X. Wu. Blind image quality assessment using a general regression neural network. *IEEE Transactions on Neural Networks*, 22(5):793–799, 2011. doi: 10.1109/TNN.2011.2120620

[12] K. Moreland, D. Lepage, D. Koller, and G. Humphreys. Remote rendering for ultrascale data. In *Journal of Physics: Conference Series*, vol. 125, p. 012096. IOP Publishing, 2008.

[13] D. Pajak, R. Herzog, E. Eisemann, K. Myszkowski, and H.-P. Seidel. Scalable remote rendering with depth and motion-flow augmented streaming. In *Computer Graphics Forum*, vol. 30, pp. 415–424. Wiley Online Library, 2011.

[14] S. Shi and C.-H. Hsu. A survey of interactive remote rendering systems. *ACM Computing Surveys (CSUR)*, 47(4):57, 2015.

[15] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.