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. 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.

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
The first step is to acquire the last frame that was rendered on every display node. Our display nodes have a resolution of 1200 \times 4096 and use the portrait mode, therefore their original resolution is 4096 \times 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 \times 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 \times 240 do not cover the whole rotated texture we rotate the texture back to the resolution, 1200 \times 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 to the network for 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 \times 768 and a second texture array with 2 sub-textures of the size 4800 \times 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 cntx and cnty 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.

### 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

<table>
<thead>
<tr>
<th>setting</th>
<th>preset</th>
<th>CONSTQP</th>
<th>avg. bitrate</th>
<th>max. bitrate</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOW</td>
<td>DEFAULT</td>
<td>40</td>
<td>645120</td>
<td>645120</td>
</tr>
<tr>
<td>MEDIUM</td>
<td>HIGH QUALITY</td>
<td>30</td>
<td>7741440</td>
<td>7741440</td>
</tr>
<tr>
<td>HIGH</td>
<td>LOSSLESS</td>
<td>20</td>
<td>17418240</td>
<td>17418240</td>
</tr>
</tbody>
</table>

Table 1: Overview of the different encoder settings.
we can decide on the encoder setting for each tile. With each of the three settings. In combination with our optimiser the output of our network is a vector containing six values, one for the SSIM values with respect to the original tile. In addition to the different encoder settings. For each of those three, we computed the size of the encoded images. Each of these images was divided into tiles with the size of 240 × 512 from different volume rendering 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 encoder settings. 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 240 × 512 as input and the output is a vector with six values, the predicted SSIM and size values for each encoder setting. Since we decided to use CNN-based regression to predict the quality of images after encoding as well as the size, we 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. In combination with our optimiser we can decide on the encoder setting for each tile.

We use 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 rendering 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

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 240 × 512 as input and the output is a vector with six values, the predicted SSIM and size values for each encoder setting. Since we decided to use CNN-based regression to predict the quality of images after encoding as well as the size, we 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 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: $\min \sum_{i=0}^{N} \sum_{j=0}^{M-1} x_{ij} \cdot (1 - \text{SSIM}_{ij})^2$. 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: $\sum_{i=0}^{N} \sum_{j=0}^{M-1} x_{ij} \cdot \text{SIZE}_{ij} \leq T$.
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 MBit/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.

<table>
<thead>
<tr>
<th>Test</th>
<th>LOW</th>
<th>MEDIUM</th>
<th>HIGH</th>
<th>Our algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Particle (avg)</td>
<td>29.19</td>
<td>86.63</td>
<td>310.92</td>
<td>26.35</td>
</tr>
<tr>
<td>Particle (max)</td>
<td>46.13</td>
<td>133.98</td>
<td>544.55</td>
<td>42.21</td>
</tr>
<tr>
<td>Volume (avg)</td>
<td>4.00</td>
<td>13.90</td>
<td>143.23</td>
<td>10.83</td>
</tr>
<tr>
<td>Volume (max)</td>
<td>6.49</td>
<td>22.18</td>
<td>235.94</td>
<td>21.72</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>SSIM</th>
<th>MSE</th>
<th>AAE</th>
<th>STD</th>
<th>max AAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOW</td>
<td>2.43 × 10^{-5}</td>
<td>0.0026</td>
<td>0.0041</td>
<td>0.0576</td>
</tr>
<tr>
<td>MEDIUM</td>
<td>1.19 × 10^{-5}</td>
<td>0.0018</td>
<td>0.0029</td>
<td>0.0308</td>
</tr>
<tr>
<td>HIGH</td>
<td>5.38 × 10^{-6}</td>
<td>0.0016</td>
<td>0.0016</td>
<td>0.0156</td>
</tr>
</tbody>
</table>

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.
low 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


