Rixels: Towards Secure Interactive 3D Graphics in Engineering Clouds

Altenhofen, Christian; Dietrich, Andreas; Stork, André; and Fellner, Dieter

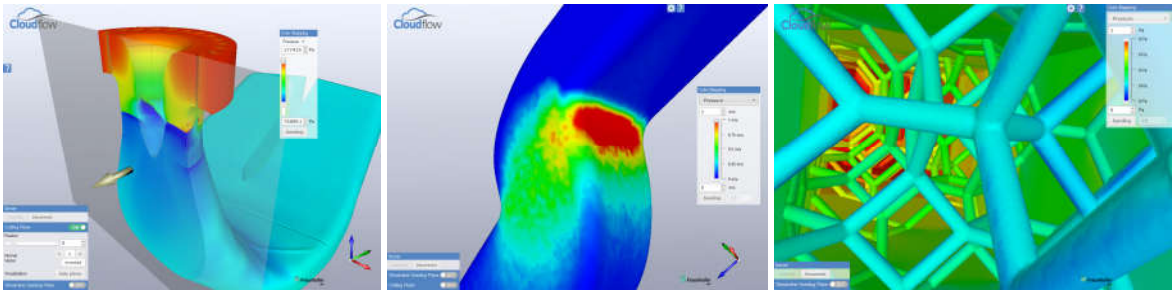

Figure 1: Three images showing the rich pixels approach used to visualize 3D simulation results in a web client.

Abstract: ***Cloud computing rekindles old and imposes new challenges on remote visualization especially for interactive 3D graphics applications, e.g., in engineering and/or in entertainment. In this paper we present and discuss an approach entitled 'rich pixels' (short 'rixels') that balances the requirements concerning security and interactivity with the possibilities of hardware-accelerated post-processing and rendering, both on the server side as well as on the client side using WebGL.***

Index Terms: ***Remote rendering, application sharing, rich pixels, web-based rendering, cloud computing, scientific visualization, remote post-processing***

1. INTRODUCTION

CLOUD computing is more and more digging into interactive applications where not only elastic computing on big data is required but graphical interaction with 3D models / 3D content.

With the virtualization of computing power, also the demand for virtualization of the rendering power increased. This stimulates developments such as NVIDIA's GRID graphics [4] able to serve more than one 'user' with hardware-accelerated rendering. To this end, classical image or video encoding and transmission techniques are adopted to transfer screen content to the client machine. This approach can be combined with any off-the-shelf application. However, it requires transmitting new pictures with every single user interaction and is thus prone to create high network traffic. In parallel, WebGL [5] has become a standard feature of all web browsers and provides hardware-accelerated rendering from within the web browser – a feature which was hardly conceivable few years ago. With WebGL, 3D content can be rendered interactively after sending the graphical information (the 3D model, the simulations results, etc.) to the web client – as long as the power of the client machine is sufficient for the current model complexity. Still, compared to desktop applications, web applications have limitations, e.g., speed/flexibility of JavaScript or the limited feature set of WebGL compared to OpenGL. More importantly, the transmission of large datasets – common in cloud computing – to the client can take a considerable amount of time, so that 'time to first picture' may be much longer than in an application sharing approach where the data resides on the server and only images are being sent. However, for many applications the biggest 'problem' with this approach is the fact that the 3D data is being sent to the client, which opens the possibility to grab confidential content from the network in a form close to the original model data – a real acceptance hurdle for many professional users.

These observations (virtualization and hardware acceleration on the server side and the availability of WebGL on the client side) have motivated us to develop an approach that we entitled 'rich pixels' or 'rixels'. In our rich pixels approach, data resides on the server and we use hardware-accelerated server-side post-processing and rendering based on OpenGL with all its functionality, flexibility and performance.

Manuscript received August 1, 2015. This work was supported by the EU-project CloudFlow (FP7-2013-NMP-ICT-FoF-609100) which is co-funded by the 7th Framework Program of the European Union, project number 609100. More information can be found at http://www.eu-cloudflow.eu/.

The result is not transferred as a screen shot (RGB image), but as a set of pixels enriched with depth information and arbitrary additional attributes read from the server's (deep) framebuffer. These rich pixels are securely transmitted to the client and visualized in a web browser using WebGL. Visualization is not only limited to rendering. Because of the availability of pixel attributes, it is possible to instantly perform post-processing operations also on the client.

This hybrid approach – a combination of ideas from image transmission and local WebGL rendering and processing – promises to better serve the partially contradictory requirements:

- **Low network load:** Only data that is needed should be transferred. It also should not be necessary always having to keep up a high-bandwidth connection.
- **Interactive behavior:** The visualization client should always remain responsive without freezing or stuttering.
- **Security:** It should be possible to visually inspect a model, but without having to send the original data to the client.
- **Best use of HW rendering resources:** Not only hardware on the server side, but also any available resources on the client side, such as GPUs, should be leveraged.
- **Broad range of client devices:** A client application should not depend on specific hardware or operating systems. Ideally, it should work within a web browser.

We benchmarked our approach and found performance benefits of a factor of up to 19.8 in comparison with a standard application sharing solution like UltraVNC.

1.1. Contributions

The approach presented in this paper provides the following main contributions in the fields of application sharing and remote rendering:

- Combination of powerful hardware-accelerated server-side rendering with flexible web-based client-side rendering.
- Client-side navigation without additional data transfer from the server because pixels are enriched with depth values.
- Shader-based post-processing on the client using additional pixel attributes.
- Security of the 3D model since only rich pixels are sent from which the 3D model cannot be easily reconstructed.

1.2. *Paper Outline*

The remainder of the paper is structured as follows:

Section 2 gives an overview of existing technologies and approaches in the fields of application sharing and remote rendering. Section 3 describes our approach of rich pixels in detail while section 4 presents the client-server architecture used in our setup. Section 5 introduces remote post-processing as an example use case for our technique and section 6 provides multiple measurements of network traffic comparing our approach with the commonly used application sharing software UltraVNC. Section 7 concludes this paper by reflecting on the presented approach and summing up its advantages and current limitations. Finally, section 8 presents ideas for future improvements.

2. Related Work

The approach proposed in this paper draws from a variety of previous techniques. In the following sections we will provide a brief overview of fields our system is related to.

2.1. Remote Rendering

Remote rendering, i.e., producing an image on one device and displaying it on another network-connected device, has been an important research topic for decades, and a plethora of different techniques and systems have been proposed. We will not go into detail here and instead refer the interested reader to Shi and Hsu [1], who provide a comprehensive survey of remote rendering systems. According to them, remote rendering can be categorized into a continuum of model-based and image-based systems, where on the one end of the spectrum the original model data is transferred and on the other end only the final images are transmitted. The approach proposed here represents a mixed or hybrid technique where image data augmented with model information is being sent.

2.2. Application Sharing

Probably the most widely used remote rendering variant is application sharing, i.e., running a complete application on a remote server, and displaying the results on a local machine using a purely image-based transfer mechanism. A popular protocol for doing this is the Virtual Network Computing (VNC) system [2], which allows for controlling a remote machine over a network by sending input device events and replicating the remote display on a local workstation by receiving and displaying RGB images with changing content generated by the remotely running application through the received input events. A prominent open source implementation of VNC is UltraVNC [3], which we use for comparison. While VNC can support arbitrary applications, it relies on the availability of

sufficiently high network bandwidth, in particular when compression artifacts and interaction latency are not tolerable.

2.3. Virtualized Hardware Acceleration

A recent trend in cloud computing is the increasing availability of graphics hardware accelerators on the server side. In addition, current state-of-the-art solutions, such as NVIDIA GRID [4], enable the virtualization of graphics processors, so that a single physical GPU can serve multiple client devices. This is realized by providing user processes with their own address space, command buffers, and rendering contexts. The GRID management software is integrated into a desktop hypervisor, where for each virtual machine an instance of a virtual GPU device driver exists. GPUs with GRID support use on-chip hardware video encoders to generate a H.264 compressed bitstream of desktop images. However, this does not allow the images to carry additional data, such as depth or other attributes.

2.4. Web-based Visualization

Another current development is the possibility to leverage graphics hardware acceleration from within web browsers. In order to access 3D graphics capabilities in a web browser without the need for an additional plug-in, the Web Graphics Library (WebGL) provides a JavaScript API. WebGL [5] is based on OpenGL ES 2.0 and allows for employing shaders written in GLSL. In our system WebGL is used in conjunction with X3DOM [6], which enable handling of 3D scenes in HTML5 and to manipulate them through the HTML document object model (DOM).

Although a large number of remote visualization solutions exist, none of them provides a perfect fit for our requirements. Particularly, being able to support arbitrary client devices (desktops, as well as tablets, phones, etc.), but also to deal with low network bandwidth by performing model inspection and post-processing on the client side, lead to the design of the rich pixels system.

3. Rich Pixels Concept

In comparison to the usual transmission of RGB images (one byte for each channel: red (R), green (G), and blue (B) per pixel), we transmit framebuffer content, more precisely content of a render target, containing the visible pixels (after the painter's algorithm / rasterization) together with their positional information (x, y, z) and additional attributes. These additional attributes could for example be different IDs for different parts of the geometry, or physical values from simulation results (like pressure or velocity), and can be used for further processing on the client side.

3.1. Bandwidth Reduction

Since background pixels are discarded in our approach, the amount of data actually being sent to the client strongly depends on the size, shape, and orientation of the content on the server's virtual screen.

Especially when rendering only the outlines or the wireframe of a 3D object, the required bandwidth is much lower than it would be when transferring the entire framebuffer. In contrast, the worst case would be to have a solid model that covers every single pixel of the virtual screen. Therefore, the size of the framebuffer/render target (total number of pixels) can be used as an upper bound for the amount of data needed to transmit one single frame. The exact value depends on how many additional attributes are sent per rich pixel.

3.2. Advantages of Sending Additional Information with each Pixel

With rich pixels, the partial 3D model gives enough visual reference for the user to allow proper orientation under rotations of limited extent. The user can navigate the scene without requiring a new image from the server with each interaction. This offers a built-in preview mechanism when rotating the scene on the client. An example can be seen in Figure 2 where a rotation around a 3D dataset is shown. The rotation shown in the first three images is done

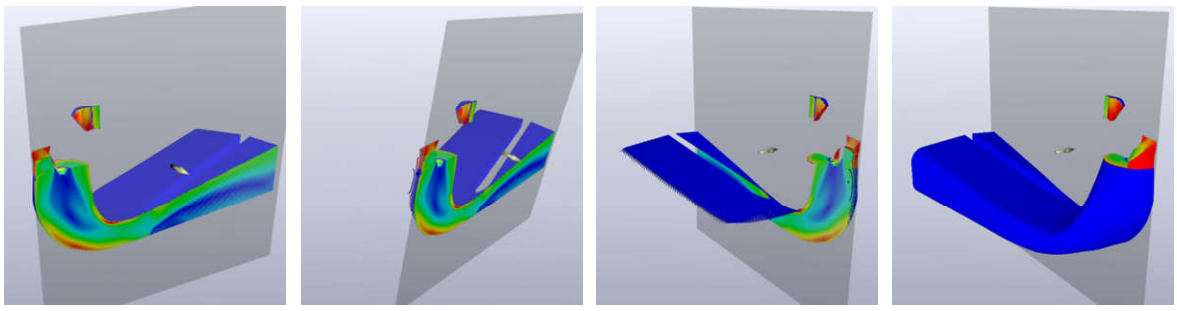

Figure 2: Rotation done on a dataset containing simulation results with RPP's preview feature activated. As the rotation angle increases, not much of the original geometry is visible anymore and the user's orientation in the dataset might get lost (3rd image).

locally whereas for the last image, new data from the server has been requested.

Since we transmit additional attributes with the pixels, we can render them in different ways. This is especially important for scientific visualization where in many cases pixels need to be false-colored depending on the mapping the user wants between their attributes and colors. This mapping can be done directly in the web browser using shaders, again without requesting a new image from the server. So the user could inspect simulation results by changing the color mapping interactively without the need to request any new data from the server as long as he does not change the camera.

3.3. SECURITY

When it comes to cloud environments, confidentiality and security of intellectual property is a common concern. In the field of cloud-based remote visualization, this corresponds to the security of the 3D model that has to be visualized. In our approach, the pixels themselves contain much less information about the real model than the full topological set of triangles typically transferred to WebGL applications. Thus, the 3D model cannot be fully downloaded by the client and cannot be intercepted during transmission. The main data resides save on the server.

In summary, the two core mechanisms of the rich pixels approach – transmitting only relevant pixels and augmenting them with auxiliary information to aid reconstruction and post-processing – allow for realizing all the initial requirements listed in chapter 1, therefore avoiding drawbacks of existing solutions in this regard.

4. CLIENT-SERVER ARCHITECTURE

Rich pixels are created by a server application using hardware-accelerated off-screen rendering with OpenGL and are then transmitted to the client application. The client itself is a website based on HTML and JavaScript that renders the rich pixels as a 3D point cloud using X3DOM and WebGL. The underlying transmission is handled using the WebSocket protocol combined with the *Protobuf* package [10]. *Protobuf* (full name 'Protocol Buffers') is a platform-independent open source framework developed by Google for efficient serialization and deserialization of structured data in multiple programming languages. For secure communication, the WebSocket connection is configured to use WebSockets over SSL/TLS (wss:// instead of ws://) for encrypted communication. In addition, symmetric or asymmetric encryption mechanisms can be applied on the rich image data directly.

Figure 3 shows a diagram describing the architecture and its components.

The server back-end can be controlled by sending specified commands from the client using dedicated Protobuf messages. The most common command is to update the orientation of the camera during user interaction. Therefore, the current camera matrix is transmitted to the server, the scene is rendered on the server using this camera matrix, and the resulting rich image is sent back to the client. As many of these commands can be produced in a short amount of time by moving the camera around, they might be issued faster as they can be processed in a low bandwidth environment. To avoid stacking of commands on the server side, the server back-end offers a queuing system that allows skipping intermediate messages of the same type. For the camera example, all received camera matrices except the most current one would be discarded when processing the next command.

As an alternative to queuing and discarding intermediate camera positions, the command can also be issued only once after the camera interaction is finished using the preview feature described in section 3.2.

Since only the 3D screen content is handled, this approach requires the re-implementation of a Web-based user interface to control the remote application. This additional effort is the pay-load

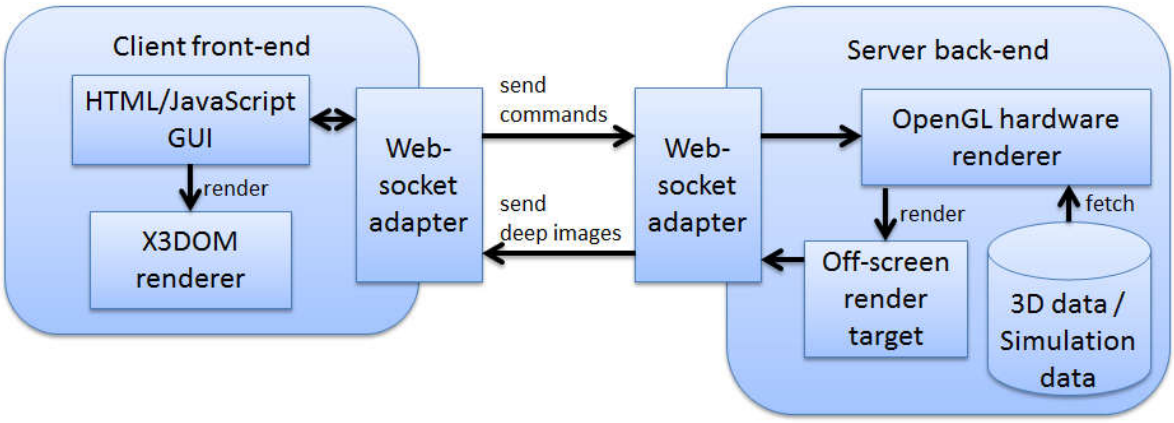

Figure 3: Client-server architecture for web-based rich pixel transmission and rendering.

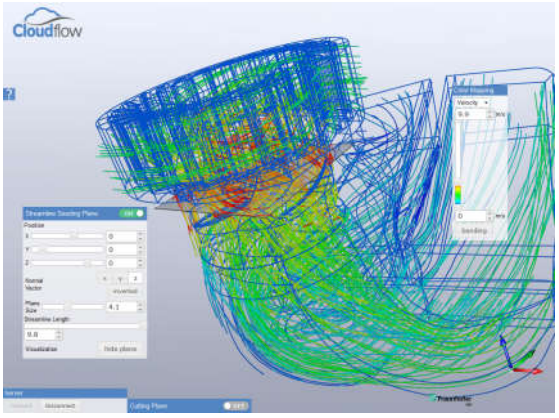

Figure 4: Stream lines computed on a dataset containing results of a CFD simulation. The stream lines are calculated and rendered on the server and are then transmitted to a web client as a set of rich pixels.

for the advantages of our approach and using WebGL on the client. However, not the entire server-side user interface might be needed in the remote scenario. Therefore, different subsets of the server's functionality could be offered on different client websites; one for each class of users.

The WebGL/JavaScript implementation of X3DOM provides generic interaction and navigation methods [6]. These include mouse based camera navigation modes. For example, in 'Walk' mode, pressing the left button and moving the mouse pushes the camera forward. Based on an updated camera position, the server computes a new frame and sends the resulting rixels to the client. In addition, the JQuery user interface provides buttons and sliders to control the post-processing operations (stream line generation, color mapping, etc.) on the client and server sides.

5. Remote Post-Processing

Engineering clouds are especially needed where large datasets are handled or where in conjunction huge simulation models are calculated / solved. To do this, engineering clouds offer high-performance computing (HPC) clusters. The storage and compute resources provided by a standard client machine are in most cases not sufficient to calculate, process, and inspect this amount of data. Here is where our rich pixel approach can best show its benefits. With this in mind, we have developed and tested a remote post-processing (RPP) setup for 3D simulation results.

The RPP system consists of a server back-end, which can perform complex post-processing tasks like calculating cutting planes or seeding stream lines in a simulation result dataset and prepares the actual rendering, as well as of a client application which shows the final results in a web page. Instead of transferring the RGB image from the server to the client as it is done in VNC-like approaches, the server application renders its scene into a framebuffer object and sends its rich pixels to the client, where they are rendered as a set of colored 3D points in the client web page. Figure 4 shows an example of the RPP client rendering a set of rich pixels containing 3D geometry and stream lines.

As the mentioned post-processing features are very compute intensive, a powerful multi-threaded hardware is needed in order to run them at interactive computation times. When computing a cutting plane or tracing stream lines through the domain, the entire volumetric dataset has to be taken into account. In most engineering use cases these datasets consist of hundreds of megabytes or gigabytes of geometry and simulation results and therefore would take too long to be transferred completely to a remote client. Even if the data could be transferred, the client in most cases would not have sufficient memory and compute resources to process it on its own. Additionally, the requirements for GPU resources for rendering the complete dataset are much higher than for rendering rich images.

Despite all the technical limitations of a remote client, sending the simulated geometry and the simulation results completely to a remote client in a cloud based scenario is not desired by most engineering companies. The data is a highly valuable asset to these companies and must not be transferred freely over the internet.

For the presented scenario of remote post-processing of large simulation data, the rich pixels approach is well suited, since the data resides save on the server where enough compute and rendering resources are available while it can be used with a flexible web client with low requirements.

6. Evaluation

In order to compare the performance of rich pixels with VNC common standard application sharing software – we use a post-processing scenario. A set of network traffic measurements have been conducted to answer the question where the rich pixels approach shows benefits in practice.

6.1. Test Setup

In order to gather representative measurements, a test suite has been designed that covers the typical use cases of a post-processing session analyzing CFD simulation results. The task sequence consists of the following steps:

a) Idle for 15 seconds (doing nothing):

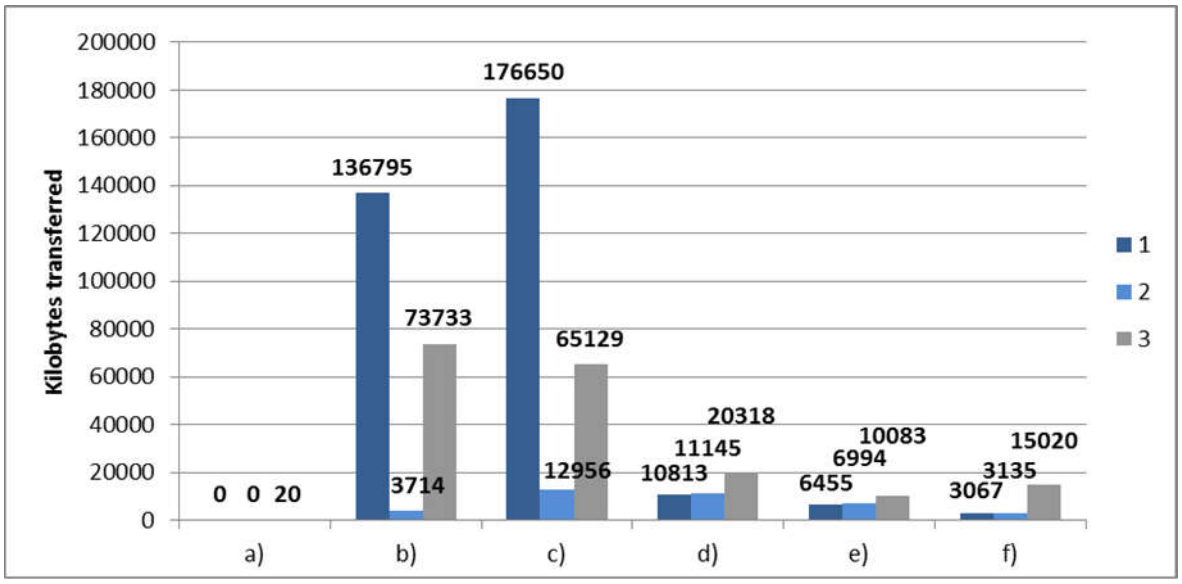

Figure 5: Amount of network traffic created by the different actions performed with RPP (1, 2) and VNC (3) as described in section 6.1. These actions are idling (a), rotating (b), zooming (c), color mapping (d), cutting (e) and calculating stream lines (f).

We simulated a user in front of the screen doing nothing for 15 seconds. The goal is to determine the amount of data transmitted to keep the connection alive and to maintain the current status.

b) Rotating the camera:
This step performs a rotation of the camera around the 3D model (from left to right and back to its initial state).

c) Zooming in and out:
In this step, the camera is zoomed out a certain extend and zoomed back in again to the initial position.

d) Changing the color mapping:
In this step we simulate the user changing the mapping between values and colors of the CFD result. It consists of changing the interval of the color ramp several times and of changing the displayed physical quantity of the velocity vector (from magnitude to "w" component).

e) Positioning the cutting plane:
In this step, the position of the cutting plane is varied and the amount of transmitted data is monitored.

f) Parametrizing stream lines:
Finally, we vary the position and the parameters of a seeding plane from which stream lines are calculated through the CFD field: first, we move the seeding plane to different positions inside the dataset; secondly, we modify the seeding parameters (seeding direction, size of seeding plane and number of seeds).

This sequence of actions has been executed in three different setups to gather information on corresponding network traffic:

1. The RPP client with the preview feature turned off.
2. The RPP client with the preview feature turned on.
3. A direct connection to the post-processing server using UltraVNC

To measure the pure network traffic, thus deducting the required bandwidth for interactive user experience, the client and server machines were placed in a local network and connected via 1 GBit/s Ethernet. To collect information about the amount of data sent from the server to the client, the network monitoring tool WireShark 1.12.4 was used.

6.2. RESULTS

Figure 5 summarizes the results of our network traffic measuring. It shows the six steps of our test sequence in the three settings described above. It clearly demonstrates that advantages and disadvantages of rich pixels compared to VNC depend on the step and on the configuration of our rich pixel application.

Step a) reveals that the RPP client creates no network traffic when idling, whereas UltraVNC still exchanges data in this case. However, the amount of data is small (20 KB).

When transmitting images due to camera rotation or translation (step b and c), VNC is roughly two to three times more efficient as the web-based RPP client using rich pixels. This mainly results from the fact, that we do not yet compress the rich pixel information and each

pixel is carrying more information than in the VNC case.

While VNC needs to send an image with each camera movement, we can exploit the rich pixel information under camera movement as declared in 2 – preview feature enabled. While moving the camera the rich pixels can still be transformed perspectively correct using the local graphics hardware on the client machine giving the user enough orientation as long as the camera manipulation does not exceed certain limits. When finishing the interaction (releasing the mouse button), a new rich pixel image is being calculated from the current perspective and transmitted. By only transmitting the new dataset once the camera interaction has finished, it reduces the amount of network traffic remarkably. In this test, the reduction factor is 13.6 for zooming and 36.8 for rotation. In general, the actual speed-up factors may vary depending on the user interactions. However, if the rotation angle is too big without requesting a new set of rich pixels, user orientation in the 3D scene may get lost (see Figure 2).

When performing interaction with GUI elements, the rich pixel approach outperforms VNC by a factor of 1.5 to 5 (as in steps d, e, and f). The explanation is straight-forward, since the GUI of the rich pixel client is web-based, there is no need to communication with the remote application at all while only acting on the 2D GUI. In contrast, VNC has to display any change in a 2D GUI element such as a button, a menu, a pop-up, etc. on the client machine by transmitting the corresponding image. The rich pixel client only sends the corresponding command once the Web-GUI interaction has finished. The processing (e.g., cutting the volumetric flow field, seeding and integration of stream lines, etc.) is performed on the server. The rich pixel server only sends a new image after the calculation is done, while VNC is sending a constant image stream also under 2D GUI interactions.

This benefit has to be ‘paid’ by developing a corresponding Web-GUI for the remote application while VNC supports off-the-shelf applications as they are.

7. Conclusion

We have presented a novel approach to remote rendering and remote post-processing in a cloud scenario by utilizing both the server’s and the client’s rendering capabilities in a combined fashion, which works well for modern cloud-based engineering use cases. By securely sending rich pixels instead of RGB images, we can interact with the data on the client to some extent without requesting new images from the server every time. Client-side shader-based post-processing is a big advantage with high potential benefit of this approach. The 3D model itself resides on the server and therefore cannot be downloaded directly or intercepted during transmission.

The comparison with UltraVNC reveals that the benefits and drawbacks of rich pixels depend on the use case. For continuous camera movements, VNC is still up to three times faster than our approach. Even with compression, we do not expect to fully compensate this benefit since we simply have more data to transmit per pixel that we then exploit in local web-based hardware-accelerated rendering. However, the fact that we do not need to transmit any background pixels and no 2D GUI updates is another pro for rich pixels. Anyway, for discrete actions, rich pixels outperform VNC by a factor of up to 19.8.

An obvious limitation of our technique is due to the point based approximation of altered (rotated, translated, zoomed) views, which results in visual artifacts stemming from missing scene geometry. When reducing bandwidth by simultaneously increasing the intervals between requests of new rixels, more and more parts of the scene are lost. Similarly, image-based methods suffer from increased visual artifacts when applying lossy image compression. However, they always provide a complete image of the 3D scene. In the case of extensive bandwidth reduction, artifacts caused by our approach might be perceived as more objectionable by domain users. It would be an avenue of future research to conduct user studies and to evaluate an acceptable performance–quality tradeoff.

8. Future Work

There are several points, where our approach could be improved in order to further reduce network traffic especially for steps b) and c).

8.1. Compression

In its current state, the rich pixel data is transmitted exactly as it is stored in the server’s frame buffer without any compression. Of course, this is not optimal and could be improved by applying or adapting different image/data compression techniques (see [7] for details). When trying to do this, the compression and decompression time has to be balanced w.r.t. the potentially reduced duration for transmission of the compressed data.

8.2. Progressive Transmission

The set of rich pixels can also be regarded as a large 3D point cloud. With this mental image in mind, progressive streaming methods like the ones presented in [8] and [9] could be applied.

This would further increase the responsiveness and interactivity of the rich pixel client application especially for large-resolution render targets and / or low-bandwidth environments.

8.3. Application to Other Domains

The presented architecture is not tied to the simulation domain. This use case was chosen because this is an obvious one to have additional values besides the color (like stress, temperature, pressure, or velocity). Of course, any other dataset that provides positional information in combination with extra attributes could be displayed. For example, an interesting application would be the visualization of large cultural heritage objects, where in addition to the spatial location of a pixel not only color values, but also surface characteristics could be transferred.

Acknowledgment

We would like to thank Daniel Weber and Thomas Gierlinger from the Fraunhofer Institute for Computer Graphics Research IGD, Felix Loosmann from the Graduate School of Excellence Computational Engineering GSC CE, as well as Sven Müller and Martin Becker from Stellba Hydro GmbH & Co KG for providing 3D models and simulation results used to develop and test our approach, and their additional support.

References

[1] Shi, S., Hsu, C. H., "A Survey of Interactive Remote Rendering Systems," *ACM, ACM Computing Surveys (CSUR)* 47.4, 2015, p. 57

[2] Richardson, T., Stafford-Fraser, Q., Wood, K. R., Hopper, A., "Virtual network computing," *IEEE, Internet Computing*, 1998, pp. 33–38.

[3] Ultra VNC remote access tools. www.uvnc.com.

[4] NVIDIA, "NVIDIA GRID: Graphics Accelerated VDI with the Visual Performance of a Workstation," http://www.nvidia.de/content/grid/resources/White_paper_graphics_accelerated_VDI_v1.pdf.

[5] Khronos Group, "OpenGL ES 2.0 for the Web," https://www.khronos.org/webgl.

[6] Behr, J., Eschler, P., Jung, Y., Zöllner, M., "X3DOM: a DOM-based HTML5/X3D integration model," *ACM, Proceedings of the 14th International Conference on 3D Web Technology*, 2009, pp. 127-135.

[7] Ratanaworabhan, P., Ke, J., Burtscher, M., "Fast lossless compression of scientific floating-point data," *IEEE, Data Compression Conference 2006, Proceedings*, 2006, pp. 133-142

[8] Limper, M., Thöner, M., Behr, J., Fellner, D. W., "SRC-a streamable format for generalized web-based 3D data transmission," *ACM, Proceedings of the 19th International ACM Conference on 3D Web Technologies*, 2014, pp. 35-43

[9] Evans, A., Agenjo, J., Blat, J, "Web-based visualisation of on-set point cloud data," *ACM, Proceedings of the 11th European Conference on Visual Media Production,* 2014, p. 10

[10] Protocol Buffers https://developers.google.com/protocol-buffers/

CHRISTIAN ALTENHOFEN is a PhD student at the Technische Universität Darmstadt and associated with the Graduate School of Excellence Computational Engineering GSC CE. Dr. Andreas Dietrich and Christian Altenhofen are research associates in the department for Interactive Engineering Technologies at the Fraunhofer Institute for Computer Graphics Research IGD. Prof. Dr. André Stork is head of the department for Interactive Engineering Technologies at the Fraunhofer Institute for Computer Graphics Research IGD and professor at the Technische Universität Darmstadt. Prof. Dr. techn. Dieter W. Fellner is director of the Fraunhofer Institute for Computer Graphics Research IGD and professor at the Technische Universität Darmstadt. He is affiliated with the Graz University of Technology where he chairs the Institute of Computer Graphics and Knowledge Visualization.