Image Reconstruction from Spatially Non-uniform Samples

Adam Siekawa*
Supervised by: Radoslaw Mantiuk†

Institute of Computer Science
West Pomeranian University of Technology
Szczecin / Poland

Abstract

We evaluate methods for image reconstruction from spatially non-uniform samples. Such distribution is characteristic for location of cones on the human retina. Transformation from spatially non-uniform samples to raster image in Cartesian coordinates is obligatory step for the retinal rendering techniques. We present two methods in our work, one using triangle mesh renderer and second using image post processing operation called a push-pull. In the first method vertices are placed at corresponding sample positions in screen space, which allows us to perform fast triangular interpolation of values on a GPU. Second method is based on image pyramid processing which filters out blank pixels during downsampling and fills them during upsampling. We evaluate the performance and quality of reconstruction using sample data generated by the GPU-accelerated ray tracer. As it is work in progress, a map of the non-uniform sample distribution and a map of triangles are generated off-line during preprocessing.

Keywords: non-uniform image sampling, image reconstruction, gaze-dependent rendering, gaze-contingent display, retinal rendering

1 Introduction

Contemporary rendering algorithms use sampling in the regular Cartesian coordinate system. Since rendered image is supposed to be displayed on a flat and rectangular display it is an intuitive choice for sample distribution for raster image.

With increasing popularity of Virtual Reality (VR) and the Head Mounted Displays (HMD), a non-uniform sample distribution schemas are applicable. For example, in the most modern HMDs image is rendered using barrel distortion, which is required for visualizing the image properly on such devices.

Eye tracking becomes a popular way of aiding rendering systems by rendering scenes with varying sample density. This sampling schema uses lower density of samples in the parafoveal region of vision [4, 9, 8], which can be non-linearly reduced due to decreasing the contrast sensitivity with the eccentricity on the human retina. Lower sensitivity is caused by non-uniform distribution of light receptor cells, called cones. On the fovea there are 150 thousand cones per square millimetre, but for eccentricity equal to 10 degrees this number falls down to below 10 thousand.

Although models of rendering for peripheral regions are not yet matured [7], we can assume that it is possible to significantly reduce number of samples needed to render these regions. This approach can also be supported by limitations of HMDs, in which lenses optical distortions introduce even more distortion in the rendered light field.

In this work we evaluate two different reconstruction techniques that transform rendered image samples from the non-uniform, so called retinal space to the Cartesian space of the displayed image. In the first method vertices are placed at corresponding sample positions in the screen space. Then, the triangulated mesh is rendered using GPU. Second method is based on image pyramid processing which filters out blank pixels during downsampling and filling them during upsampling.

We evaluate reconstruction accuracy by comparing reconstructed image to the image where each pixel is rendered using complete sampling in the Cartesian space. We test the performance of the evaluated reconstruction techniques. Both tests are conducted for varying number of the non-uniformly distributed samples, showing accuracy and performance trade-off for the real-time rendering scenarios.

Ray tracing is a popular rendering method which simulates light behaviour. It works by sending rays into the scene that are collecting radiance. Ray tracing allows to render accurate reflection, refractions and shadows, which results in photo realistic images. Unfortunately, due to slow performance it finds most of the usage in cinematography and CAD visualisations. Since this method can produce physically accurate images it becomes more appealing as a rendering solution for VR applications.

Our work aims to significantly improve ray tracing performance by using fast non-uniform sample distribution. Such combination can lead to better, more realistic image synthesis on HMDs without visible image degradation at the same rendering time needed per frame.

In Section 2, we show how the rendering techniques

*asiekawa@wi.zut.edu.pl
†rmantiuk@wi.zut.edu.pl

can benefit from the non-uniform sampling approach. In Section 3 we describe the evaluated reconstruction techniques; triangle mesh rendering and a push-pull technique. In Section 4 we present a testbed and stimuli used for experiments we performed, compare results obtained in both reconstruction techniques and evaluate their performance. The paper ends with conclusions and future work in section 5.

2 Background

A number of approaches have been proposed in the literature that reconstruct a signal from a reduced number of samples. In this work we omit these studies and focus on the real time reconstruction techniques. For the high quality approaches that, however, require a significant time overhead and can not be used in the real time graphics application, we refer the reader to Yaroslavsky et al. work [11].

Adaptive image sampling. Non-uniform image sampling can be used in the rendering techniques, in which one wants to reduce sampling in the regions that contribute less to the output image. For instance, coarser sampling can be applied in low frequency regions of the scene. The opposite approach with increased number of samples can be used to enhance complex objects. Such approach is applied in the adaptive anti-aliasing techniques [2].

As proposed in Gunter et al. [4], if the viewing direction of the observer are known, the number of rendered pixels can be significantly reduced by rendering three low resolution images at different fields of view instead of one high resolution image. Images rendered for the wider view angle are then magnified and combined with lower field of view images. The later ones depict details in the area surrounding the gaze position. This technique reduces sampling rate for the peripheral regions but keeps the deterioration of image quality invisible for observer.

Another technique proposed by Stengel et al. [9] aims to reduce shading complexity in the deferred shading technique [2]. The spatial sampling is constant for the whole image but the material shaders are simplified for peripheral pixels. This technique reduces the shading time up to 80%.

The gaze-dependent ray tracing was proposed in [8]. A varying size of pixels allows to reduce sampling for image areas distant from the observer's gaze location. The sizes of individual pixels are determined by gaze-dependent contrast sensitivity function, which models how human contrast sensitivity is decreased with eccentricity [5].

Ultra-wide displays. Recent advancements in display technology resulted in wide spread of Virtual Reality hardware. Unfortunately Head Mounted Devices (HMDs) require high resolution displays in order to produce acceptable visual experience. Consequently, VR system require high end GPUs to provide acceptable graphics quality. Additionally, images displayed on such devices need to be transformed by applying barrel distortion as shown in Fig. 1. Such transformation suggests that less samples might be needed on more distorted regions of the image. For these regions a non-uniform sampling is a natural way to speed-up rendering.

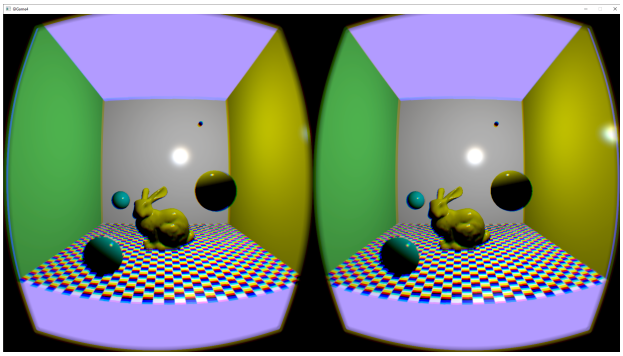

Figure 1: Example of the rendering required by the HMD stereo display. Generated for Oculus DK2.

3 Reconstruction Techniques

Ray tracing is not limited to the uniform sampling schema, because one has a full control over the ray origin and direction. This allows to render images based on the non-uniform sampling algorithms with a negligible impact on the rendering performance. Since the ray tracing performance depends on the number of traced rays, one can choose a sample distribution that reduces the overall number of rays. However, the next step is required to transform the spatially non-uniform samples to the Cartesian coordinates, which can be displayed on the screen. The goal is to use a reconstruction technique, which introduces the lowest possible distortions to the original signal.

In this Section we present two reconstruction techniques that address this problem: *push-pull* and *triangle mesh rendering*. The input in both techniques is the RGB image with pixels located in random positions (see Fig. 2). We pre-compute this spatially non-uniform pixel distribution using the quasi-random locations (see Section 4.1) but, in future work, we plan to implement a distribution, which mimics the sampling schema of the human eye.

3.1 Push-Pull Technique

The *push-pull* technique has been introduced by Gortler et al. [3]. We simplified the implementation of this technique to achieve better performance.

In the *push* phase, the mipmapping is applied. The image is downsampled by a factor of two and stored for later use. During downsampling, four corresponding pixels are averaged but the pixels that are marked as empty (do not

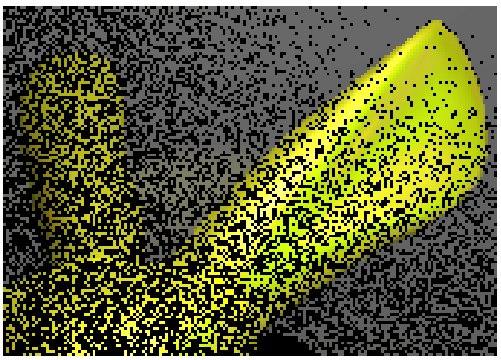

Figure 2: Part of the RGB image consisting of the quasi-randomly distributed samples. The black areas indicate a lack of pixels.

contain a sample value computed by ray tracer) are ignored (see Fig. 3, left). If all four pixels are marked as empty, we also mark destination pixel in the lower level as empty and set its color to 0. This operation is performed recursively taking newly calculated image as input until we cannot halve image size anymore. It is recommended to ensure that image dimensions are a powers of two - this can be achieved by inserting missing columns and rows to the image. It is assumed that at the lowest level there are no empty pixels.

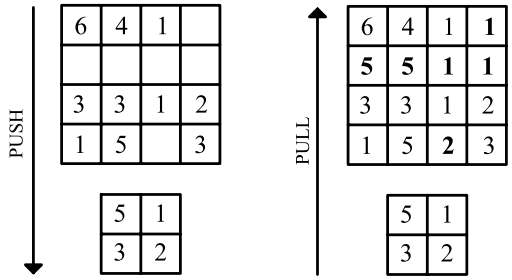

Figure 3: The push-pull technique.

The *pull* phase starts from the second level from the coarsest level of the pyramid. If pixel in this image is marked as empty, it takes color of the corresponding pixel from the coarser level of the pyramid (see Fig. 3, right). This procedure is repeated by climbing to the top of the pyramid.

Source 1 presents details of the push-pull technique. This listing shows two OpenCL kernel functions executed in subsequent steps of the algorithm. *Push* function takes as input *mask* image containing information on the sample locations and *image* containing colors of the samples. Results are stored in *maskL* and *imageL* variables, that corresponds to mask and image from the coarser level of the pyramid. *w* and *h* are the total width and height of the input image, respectively. Pixels from the input image are averaged using 2x2 block filter but the pixel color is added only if the sample exists at the corresponding mask location (lines 15-19). If there is a valid sample in 2x2 pixel block, the averaged color is stored in the coarser level of the pyramid and 1 is written into lower level mask. Otherwise, the mask is set to 0.

Pull function uses the results generated by the *Push* function. If there are no valid sample in a given pixel location, the pixel color is copied from the coarser level. For the valid samples the pixel color is not modified.

Source 1: OpenCL implementation of the push-pull algorithm.

```
1  __kernel void Push(__read_only image2d_t mask,
       __read_only image2d_t image, __write_only
       image2d_t maskL, __write_only image2d_t
       imageL, int w, int h)
2  {
3    const int2 txc = { get_global_id(0),
         get_global_id(1) };
4    int idx = txc.x * h + txc.y;
5    float4 sum = 0.0f;
6    int numSamples = 0;
7    for (int x = 0; x <= 1; x++)
8    {
9      for (int y = 0; y <= 1; y++)
10     {
11       int2 tn;
12       tn.x = txc.x * 2 + x;
13       tn.y = txc.y * 2 + y;
14       float4 m = read_imagef(mask, sampler,
             tn);
15       if (m.x > 0.0f)
16       {
17         numSamples += 1;
18         sum += read_imagef(image, sampler, tn);
19       }
20     }
21   }
22   if (numSamples != 0)
23   {
24     write_imagef(imageL, txc, ans /
           (float4)numSamples);
25     write_imagef(maskL, txc, (float4)1.0f);
26   }
27   else
28   {
29     write_imagef(maskL, txc, (float4)0.0f);
30   }
31 }
32
33 __kernel void Pull(__read_only image2d_t mask,
       __write_only image2d_t image, __read_only
       image2d_t maskL, __read_only image2d_t
       imageL, int width, int height)
34 {
35   const int2 txc = { get_global_id(0),
         get_global_id(1) };
36   int idx = txc.x * height + txc.y;
37   float4 m = read_imagef(mask, sampler, txc);
38   if (m.x > 0)
39     return;
40   write_imagef(image, txc, read_imagef(imageL,
         sampler, txc / 2));
41 }
```

3.2 Triangle Mesh Rendering

The non-uniform map of samples can be triangulated and rendered using the standard forward rendering accelerated by GPU. The triangulation is a time consuming process, which is hard to execute in real time. However, as it is done in our work, the triangle mesh can be computed in the preprocessing and then used for the ray tracing rendering.

In our implementation a map of samples (presented e.g. in Fig. 5) is converted to the triangle mesh using the Delaunay triangulation technique. Each sample in the map becomes a vertex in the mesh (see example in Fig. 4). This mesh is pre-computed and read from file during initialization of our real-time ray tracer. Ray tracer traces rays passing through the vertices of the mesh and stores colors of the corresponding pixels. During the actual triangle mesh rendering, colors inside the triangles are interpolated in screen space using the barycentric interpolation.

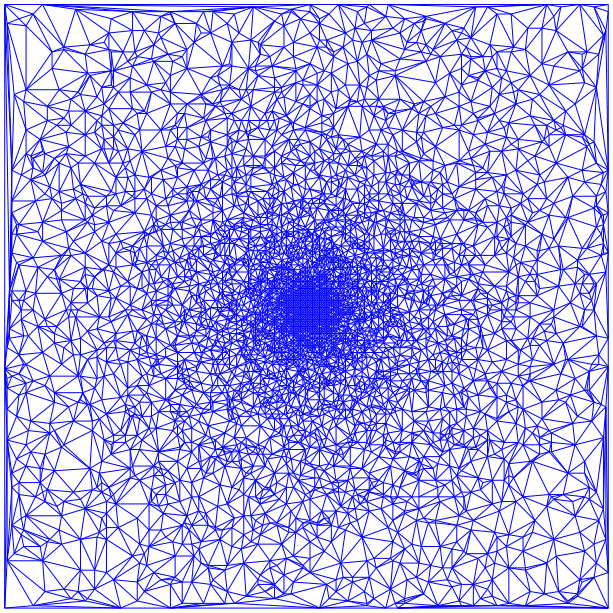

Figure 4: Example of the triangle mesh generated for the non-uniform distribution of 6590 samples.

4 Accuracy and performance tests

In this section we evaluate accuracy of the proposed image reconstruction techniques. The reconstructed images are compared to the reference image rendered for the regular Cartesian sampling. We also discuss the performance benefits resulting from the non-uniform sampling.

4.1 Testbed and stimuli

The tests have been performed using three non-uniform sample maps containing 10%, 30% and 50% of the pixels of the original image (see Fig. 5). This reference image had a resolution of 1920x1080 pixels.

Pixel coverage	Push-Pull	Mesh rendering
10%	0.9879	0.9830
30%	0.9965	0.9946
50%	0.9983	0.9954

Table 1: SSIM indices.

To evaluate the accuracy of the reconstruction, we compared the images after reconstruction to the reference images using the Structural Similarity Index Metric (SSIM) [10] and computed the global SSIM index. SSIM predicts perceptual difference between images, i.e. the difference between the images, which have been discovered by average human observer. SSIM can replace the time consuming perceptual experiments, in which people have to manually indicate differences between the images.

The sample maps were generated off-line using Matlab script. We generated quasi-random distribution of samples, which mimics the fovea-oriented distribution of the photoreceptors on the human retina. The samples are distributed with non-linear density, which depends on the eccentricity (distance from the fovea). In our implementation, the sample maps were exported to the binary image, in which 1 indicates that there is a sample at a given location, and 0 that there was not rendered sample. The Matlab script also generated a triangle mesh based on the distribution of the samples. The mesh was normalized to fit the OpenGL screen coordinates.

Custom renderer

We tested the reconstruction techniques in a custom GPU-based ray tracer implemented using OpenCL library (version 1.2). The primary rays were generated for individual samples in the sample map (or vertices in the triangle mesh). The renderer uses the Radeon Rays library [1] for calculation of the ray triangle intersection and computation of the output color. The color values were directly stored in the OpenGL Frame Buffer Object and the missing pixels were computed using the push-pull technique. Alternatively, the colors of the samples per each vertex were stored in the Vertex Buffer Object and this triangle mesh was used to render the image using OpenGL shader.

Test scene consisted of a single omnidirectional light source and Lambertian or Phong-based [6] BRDFs (see examples in Fig. 7).

4.2 Accuracy tests

In Fig. 7 the reference image is presented along with its reconstructions based on 10% of the samples. The push-pull technique generates the jagged edges, which are especially noticeable in regions distant from the fovea (see Fig. 7, center). The mesh rendering produces similar artefacts but they are less visible due to barycentric interpolation (see

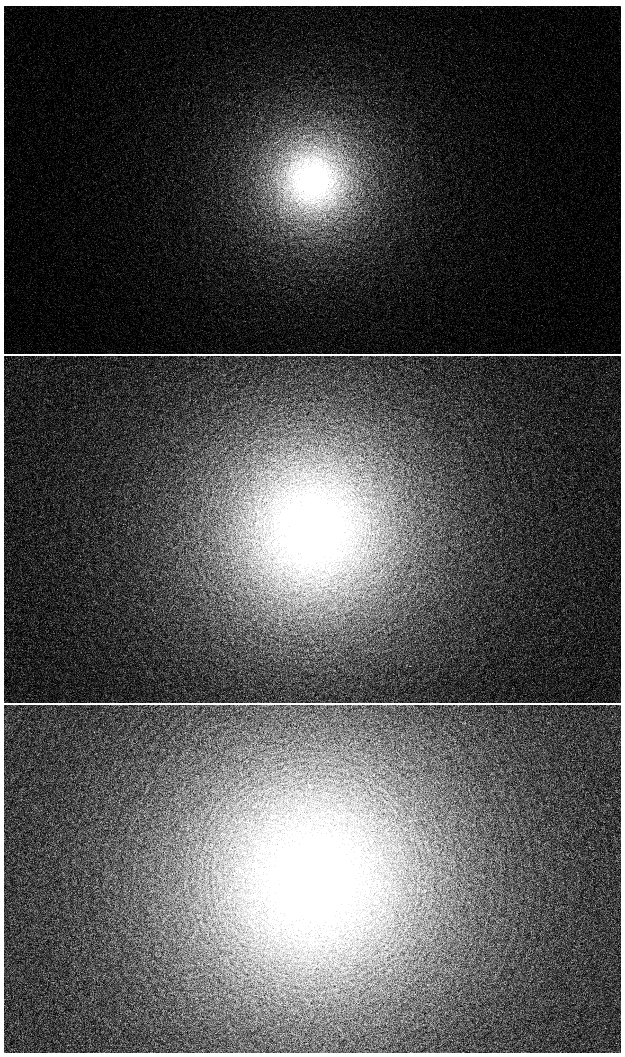

Figure 5: Maps with non-uniformly distributed samples. From top to bottom: 10%, 30% and 50% of a number of pixels from the original image.

Fig. 7, bottom). However, the interpolation introduces distortions and blurs the image.

Tab. 1 presents the SSIM indices for both push-pull and mesh reconstruction techniques. The SSIM index values close to 1 could indicate a negligable difference in the appearance of the reference and reconstructed images. However, the distortion maps presented in Fig. 6 depict noticeable difference in peripheral regions of the images and at the edges of the objects. These artefacts cause strong aliasing and, as we tested in a pilot study, are easily noticeable even in the periphery.

The reconstruction generates images of higher quality for more samples, however, it reduces the performance (see Sect. 4.3). We assume that the aliasing caused by the push-pull technique can be reduced by filtering the image during the pull phase. We plan to implement the variable size kernel to reduce the image blurring after this reconstruction. The kernel size should depend on pyramid level,

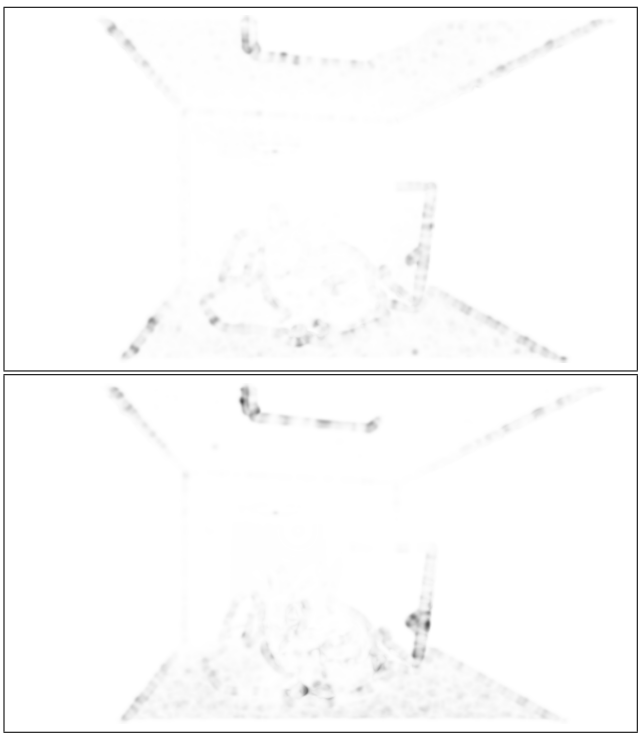

Figure 6: The SSIM distortion maps for the push-pull (top) and mesh rendering (bottom) reconstruction techniques. The maps generated for reconstruction based on 10% of samples.

i.e. for the detailed levels should be smaller than for the coarser levels. For the mesh rendering technique a typical anti-aliasing can be applied with the assumption that rays originating from the less dense regions should have higher scattering. We also plan to implement alternative shading technique, which will use the normal vectors at mesh vertices to reduce aliasing.

4.3 Performance tests

The rendering times are presented in Table 2. Our custom ray tracer renders the full frame image (1920x1080 pixels) using regular sampling in 30 ms. The rendering times for reduced number of samples are presented in the second column of Table 2. The reconstruction techniques require additional overhead depicted in column 3 and 4, which results in the total rendering speed-ups presented in the brackets.

The push-pull performance is constant for all cases because this technique is executed in the screen space and depends only on the image resolution. For 10% of samples the overall rendering time is reduced 5.5-times. In this case, we achieved 8.36-times performance boost for 10% of samples.

For the mesh rendering, lower number of samples reduces the mesh complexity and speed-ups both the ray tracing and the actual mesh rendering.

Percentage of samples	Rendering time	Push-Pull overhead	Mesh rendering overhead
10%	3.4 ms	2 ms (5.5x)	0.19 ms (8.36x)
30%	9.1 ms	2 ms (2.7x)	0.675 ms (3.06x)
50%	14.8 ms	2 ms (1.78x)	1 ms (1.9x)

Table 2: Results of the performance tests. Times in milliseconds. Ray tracing time doesn't include Push-Pull or Mesh overhead. Values in brackets shows performance improvement against the full frame rendering.

We tested the performance of the reconstruction techniques using a PC with the AMD RX 480 8GB GPU.

5 Conclusions and Future Work

In this work we introduced two methods for image reconstruction from non-uniform sample distribution. Both methods were tested in terms of accuracy and performance, achieving promising results. Due to reduced number of samples, rendering speed was increased by a large degree, enabling 1080p image rendering above 60 frames per second in all our test cases for test scene.

In the future work we plan to extend the algorithm with more accurate cone distribution model. Another issue that needs to be addressed is noticeable aliasing in push-pull method. One of potential methods to solve this problem is introduction of varying blurring kernel at each level of image pyramid. In triangle mesh rendering there's potential for better quality in alternative algorithms for triangulating 2D sample sets. More testing needs to be done on actual head mounted displays in order to determine effectiveness of presented techniques.

Acknowledgments.

The project was partially funded by the Polish National Science Centre (decision number DEC-2013/09/B/ST6/02270).

References

[1] Advanced Micro Devices, Inc. Radeon-rays library, version 2.0, 2016.

[2] Tomas Akenine-Möller, Eric Haines, and Natty Hoffman. *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., Natick, MA, USA, 2008.

[3] Steven J Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F Cohen. The lumigraph. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 43–54. ACM, 1996.

[4] Brian Guenter, Mark Finch, Steven Drucker, Desney Tan, and John Snyder. Foveated 3d graphics. *ACM Trans. Graph.*, 31(6):164:1–164:10, 2012.

[5] Eli Peli, Jian Yang, and Robert B Goldstein. Image invariance with changes in size: The role of peripheral contrast thresholds. *JOSA A*, 8(11):1762–1774, 1991.

[6] Bui Tuong Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, 1975.

[7] Ruth Rosenholtz. What your visual system sees where you are not looking. In *Human vision and electronic imaging*, page 786510, 2011.

[8] Adam Siekawa. Gaze-dependent ray tracing. In *Proceedings of Central European Seminar on Computer Graphics (non-peer-reviewed)*, pages 155–160. TU Wien, 2014.

[9] Michael Stengel and Marcus Magnor. Gaze-contingent computational displays: Boosting perceptual fidelity. *IEEE Signal Processing Magazine*, 33(5):139–148, 2016.

[10] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[11] Leonid P Yaroslavsky, Gil Shabat, Benjamin G Salomon, Ianir A Ideses, and Barak Fishbain. Nonuniform sampling, image recovery from sparse data and the discrete sampling theorem. *JOSA A*, 26(3):566–575, 2009.

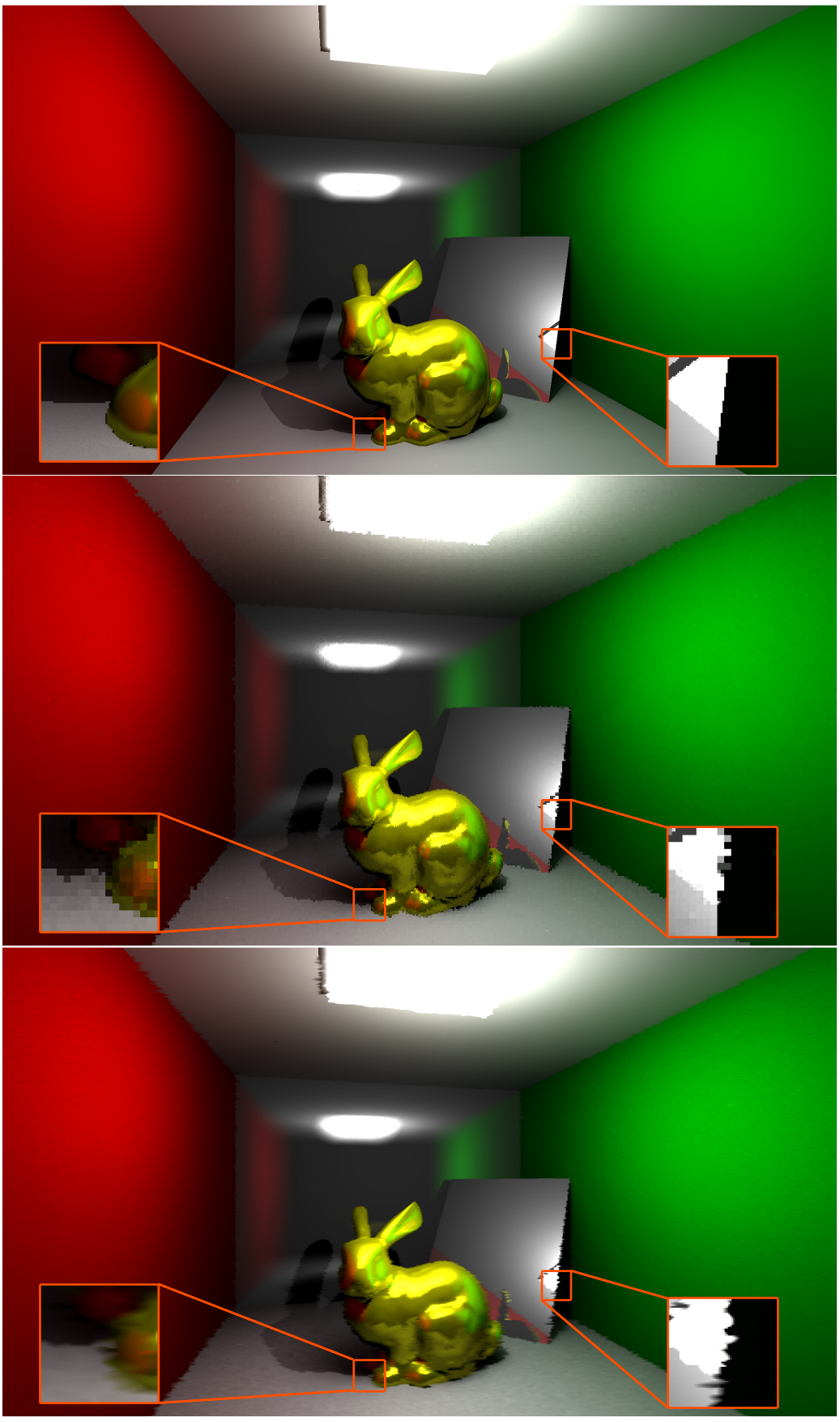

Figure 7: Example renderings (from top): reference image, image reconstructed from 10% of samples using the push-pull technique, triangle mesh rendering for 10% of samples. Two regions were magnified to depict artifacts produced by the reconstruction techniques.