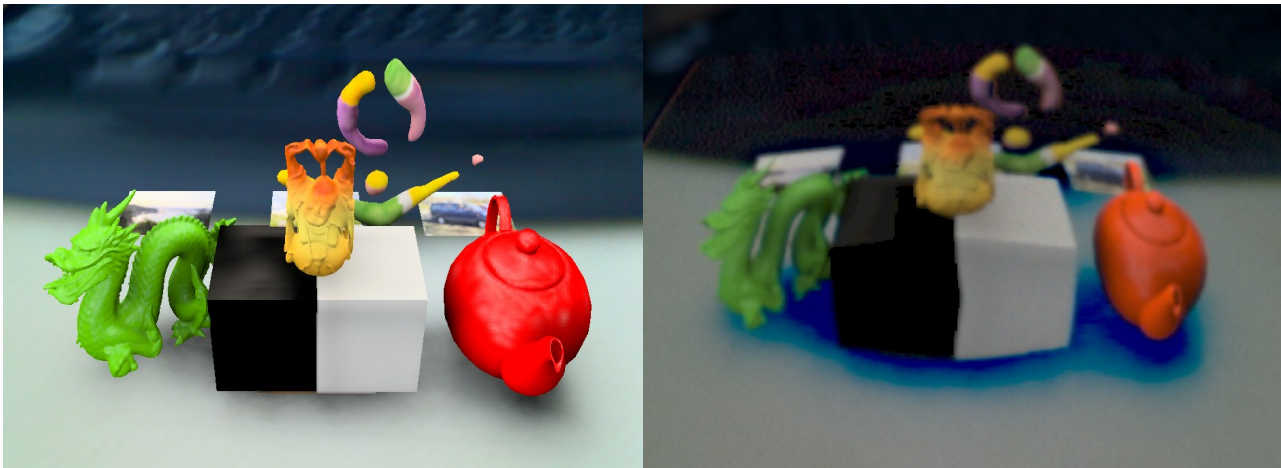


Implementing Camera Artefacts into **RESHADE**



The Goal:

Simulate camera artefacts for virtual objects as if they would have been captured by a low cost camera. Steps to reproduce the effects.

- lens distortion
- motion blur
- gaussian blur
- desaturation and color mixing
- noise
- chroma split and squash

A more detailed description of the steps can be found in [Klein et al.].

The main problem to implement was that RESHADE is based on the idea of DIFFERENTIAL RENDERING [Knecht et al.], so the process of combining virtual and real images was more complicated than just adding them together.

In essence, three buffers are used, a virtual image, a real image and a mask. So to make all the effects possible and display them correctly after computation, they had to be applied to all three buffers during their combination stage in the pipeline.

The effects were implemented as a post-processing step after the buffers were rendered, so everything could be done in the fragment program stage using hlsl shaders.

Lens distortion:

Lens distortion was implemented using a cartesian-to-polar-coordinate conversion and adjusting the radius.

Desaturation

When photons arrive at a sensor cell energy can flow to the neighbouring cells which leads to color aberrations and desaturated colors. To simulate this one step involved taking a percentage of two color channels and adding them to the destination channel. So, i.e., the resulting red channel had also portions of the blue and green channels.

Motion Blur:

To make the virtual object look like they have been captured by a real camera, one important step is to introduce motion blur. This was implemented by leveraging temporal coherence between two consecutive frames. The directed screenspace velocity of the frame since the last frame had already been computed during a prior stage so the implementation was straight forward.

Downsampling / Gaussian Blur:

Low-cost cameras don't have a high resolution so to make virtual objects look "captured" the image of them had to be down sampled. This was accomplished by rendering the image into a smaller render target.

Gaussian Blur was implemented using a two-step algorithm which involves first filtering the image horizontally and then filtering the result vertically, but it can also be done vice versa. This leads to a fewer computation calls than filtering two-dimensionally.

Bayer masking

To simulate a physical camera, the sensor had to be imitated. The bayer mask of the resulting image was recreated splitting the color channels. Every channel got its own noise function applied to and was horizontally gaussian blurred.

Chroma split and squash

To decrease the resolution of the chromaticity information the so far computed image had to be converted to Yuv space. The illumination information was kept at full resolution and the chroma information downsampled horizontally to one forth.

Controls:

motion blur can be adjusted by pressing *SPACE* + *M* and *SPACE* + *N* keys to increase or decrease the velocity factor.

Gaussian blur kernel width can be increased and decreased using *SPACE* + *K* and *SPACE* + *J* keys.

Radial distortion can be steered using *SPACE* + *I* and *SPACE* + *U* keys.

Literature

KLEIN, G., AND MURRAY, D. W. 2010. Simulating low-cost cameras for augmented reality compositing. *IEEE Transactions on Visualization and Computer Graphics* 16, 369–380.

KNECHT, M., TRAXLER, C., MATTAUSCH, O., PURGATHOFER, W., AND WIMMER, M. 2010. Differential instant radiosity for mixed reality. In *Proceedings of the 2010 IEEE International Symposium on Mixed and Augmented Reality (ISMAR 2010)*, 99–107.