Interactive Augmentation of Live Images using a HDR Stereo Camera

Matthias Korn, Maik Stange, Andreas von Arb, Lisa Blum, Michael Kreil,
Kathrin-Jennifer Kunze, Jens Anhenn, Timo Wallrath, Thorsten Grosch

Institute for Computational Visualistics
University of Koblenz-Landau

Abstract: Adding virtual objects to real environments plays an important role in today's computer graphics: Typical examples are virtual furniture in a real room and virtual characters in real movies. For a believable appearance, consistent lighting of the virtual objects is required. We present an augmented reality system that displays virtual objects with consistent illumination and shadows in the image of a simple webcam. We use two high dynamic range video cameras with fisheye lenses permanently recording the environment illumination. A sampling algorithm selects a few bright parts in one of the wide angle images and the corresponding points in the second camera image. The 3D position can then be calculated using epipolar geometry. Finally, the selected point lights are used in a multi pass algorithm to draw the virtual object with shadows. To validate our approach, we compare the appearance and shadows of the synthetic objects with real objects.

Keywords: High Dynamic Range, Augmented Reality, Fisheye Epipolar Geometry

1 Intro / Previous work

Adding virtual objects in real images has a long tradition in computer graphics: The pioneering work was done by Fournier et al. [FGR93] who invented the differential rendering technique for augmenting a real photograph with consistent lighting, based on a radiosity simulation. Several improvements exist for improving the realism and speed of the augmentation. Debevec [Deb98] introduced the HDR light probe for illumination with natural light. A similar approach was used by Sato et al. [SSI99], using a fisheye lens for capturing the environment illumination. On the other hand, Gibson et al. [GCHH03] have shown that real-time augmentation of photographs is possible with graphics the hardware. Nowadays, there are two different types of illumination for rendering with natural light: Precomputed Radiance Transfer (PRT) and sampling-based approaches. PRT techniques transform the environment map in an orthonormal basis which enables a fast illumination requiring only a simple dot product for each vertex. This method is well suited for diffuse objects in slow-varying lighting environments, but it omits high frequency content in the environ-

ment map. Consequently, the resulting shadows are blurry. Moreover, this method precomputes a transfer function for each vertex and is thus limited to rigid objects. On the other hand, sampling techniques can display sharp shadows and dynamic objects but they require many rendering passes for a good appearance. Although both techniques produce believable results for natural illumination, they are mainly limited to static environment maps, and an extension to dynamic illumination is not straightforward.

Recently, Havran et al. [HSK+05] created a system for real-time illumination of virtual objects using one HDR camera with a fisheye lens. Light sources are extracted from the HDR images and the lighting is computed with the graphics hardware using a multi pass algorithm. However, this system is limited to distant illumination, which is a wrong assumption for in-door environments. Our system is able to reconstruct both the brightness and position of the light sources. Moreover, the illuminated object is displayed in the image of a webcam using ARToolkit marker tracking [KB99]. To determine the 3D position of a common feature in two photographs, the epipolar geometry can be used [Fau93].

The rest of the paper is organized as follows: In section 2 we describe our approach for interactive illumination of a virtual object, in section 3 follows our hardware setup. Section 4 describes the sampling algorithm, the epipolar geometry for fisheye lenses follows in section 5. Section 6 describes the multi pass rendering algorithm for displaying the virtual object in the image of a webcam. We show our results in section 7 before we conclude in section 8.

2 Our approach

Our system consists of three cameras: Two HDR video cameras (HRDC IMS Chips), directed upwards with fisheye lenses and a normal webcam. The webcam is directed towards a marker where the virtual object should be placed at. First, we select a few bright pixels in one camera image. To detect the corresponding points in the second image, we calculate the epipolar curve in the second camera image. Because of the wide-angle lens, the possible corresponding points are not placed on a straight line. After detection of the most similar point on the epipolar curve, we calculate the 3D position using triangulation. The resulting set of point lights is used for illuminating the virtual object with the graphics hardware. For each point light, we calculate the illumination and a hard shadow using the shadow volume of the virtual object. We assume that we have a 3D model of the local scene around the virtual object for displaying the shadow on real objects. Because we omitted a certain amount of environment light by choosing a few samples, we calculate an ambient term representing the missing light. The correct camera pose for inserting the virtual object is calculated using a marker and ARToolkit. The three steps for augmentation are described in detail in the next sections.

3 Setup

The two HDR cameras are attached to a table with a distance of about 30cm between them. The position of the virtual object is assumed to be in the middle of the cameras. Thus the marker has to be placed at this position to reduce calibration complexity (see fig. 1).

Figure 1: Webcam, HDR-cameras with marker, augmented scene on display

4 Sampling

Because we developed the whole project on our own, the light source detection does not use any known algorithms, like the blue noise method [ODJ04]. Our system includes three implemented algorithms.

4.1 Simple light selection

Our first algorithm is a simple selection of bright pixels. The first one is important to see whether the other algorithms find the brightest light sources in the scene. It locates the n brightest pixels in the picture and provides their color values and pixel coordinates. The parameter n can be selected by the user as a time/quality tradeoff. This simple selection causes problems in the case of multiple light sources. For example, if there are two fluorescent tubes, it will only detect the one with the brightest pixels, while the other one will never be found, as can be seen in figure 2 on the left.

4.2 Threshold sampling

The second algorithm solves this problem – it samples a uniformly distributed subset of all pixels (around 20000) to find n light sources. The asset compared with the first algorithm is the detection of more than one bright light source. As shown in the middle of figure 2, it is now possible to find both fluorescent tubes as light source and not only the brightest light source. To speed up

these two algorithms there is a threshold value which decides whether the pixel is bright enough to process it or not. We use the average brightness value of all pixels used for ambient light calculation weighted by a variable value as a threshold value. The main problem of this algorithm is the resampling ”from scratch” which causes different sampling positions for every new frame. This leads to ”flickering” effects.

4.3 Cached samples

The third algorithm solves these problems. It uses a cache for the light sources. After passing the sampling there are n additional light source pixels found. These other light source pixels will be compared with the cache and only one single light can replace another if it fulfills some restrictions: For a replacement, the new light must be in the proximity of an old light position and it must be brighter. Figure 2 on the right shows the radius around the old light in which a brighter light could replace it. This is also needed to assure that not all lights converge to the brightest ones after some time. The disadvantage of this cache is that the appearance of a new light, which is not located in

Figure 2: The left image shows the results of the first algorithm, which returns the n brightest pixels. Detected light positions are represented by small circles. All sampling positions are located within a single fluorescent tube. Results of the threshold sampling algorithm shows the image in the middle. A better sampling distribution is obtained here, note that we found every fluorescent tube. Detected sampling positions of the final algorithm with cache in the right image. Each fluorescent tube contains some samples. The large circles around the sampling positions represent the acceptance region for a new sample.

the radius around an old light, will not be detected. This happens, for example, when switching on a flashlight. Therefore, we search for bright pixels during the calculation of the ambient term. If there is a pixel which is much brighter than the cache pixel, a cache reset is done. A cache reset means clearing the cache and resampling to find new light sources. If a new light was found it will be registered and saved. To get the ambient term we calculate the mean value of all pixels.

5 3D Geometry

As mentioned previously, we are using two HDR cameras with fisheye lenses. In order to record the illumination of the environment with underexposed and overexposed areas, we decided to use two HDR cameras. Due to the fisheye lenses, it is possible to cover a large range to locate as many light sources as possible. This approach should lead to a more realistic illumination of the virtual objects.
While working we realized that the project entails some difficulties. The images are distorted because of the fisheye lenses. As a result the epipolar lines turned out to be in fact epipolar curves. Projecting the images on a plane for equalization led to blurry effects and is not very meaningful for aperture angles greater than 180 degrees. Additionally, the fisheye lenses are not precisely centered to the camera chips, so the images are lightly displaced.
Last but not least we can not assure that the camera axes are parallel because we fix the cameras on the table with simple screw clamps. Therefore, it was not possible to use known methods of epipolar geometry. Our approach to solve these problems will be explained in the following sections.

5.1 Fisheye lens

A pinhole camera or other distortion free lens systems are easy to describe mathematically. Unfortunately fisheye lenses are more complex.
Usually, the mapping function for fisheye lenses is $r = c \cdot \theta$, where r is the distance of a point from the image centre, θ is the angle from the optical axis and c is a constant that is dependent on the focal length and the size of the image (see fig. 3).
We used a simple dot pattern (fig. 4) to confirm that our fisheye lenses satisfy this mapping function accurately to the subpixel.

5.2 3D Calibration

As mentioned above, we do not know any intrinsic or extrinsic parameters of both cameras. So we needed a calibration system that calculates all camera parameters before the AR-system is used. Most of the known calibration systems use grid or chess board patterns [Zha00]. But these systems are not suitable for fisheye lenses, because the pattern covers only a small part of the field of view. Other systems use special equipment and are complex to use or to implement. But we preferred a low-budget solution, that works automatically and is easy to implement.
First, we describe the projection mathematically in three steps:

1. Rotating (M_{camera}) and translating (v_{camera}) a 3D vector v into the coordinate system of the

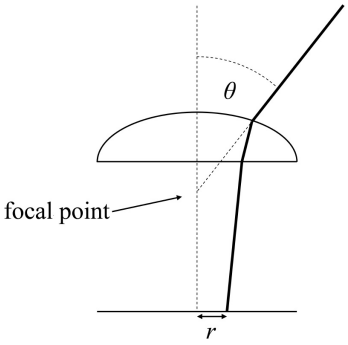

Figure 3: θ and r in a fisheye lens

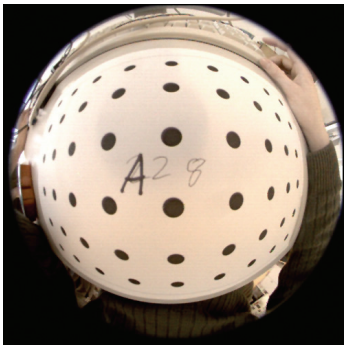

Figure 4: Fisheye lens calibration with a simple dot pattern

camera (extrinsic parameters).

$$v' = M_{camera} \cdot (v - v_{camera}) \tag{1}$$

2. Projecting the 3D vector $v' = (x,y,z)^T$ into 2D (mapping function).

$$v'' = \frac{\arctan(\frac{r}{z})}{r} \cdot (x,y)^T, \qquad \text{where } r = \sqrt{x^2+y^2} \tag{2}$$

3. Scaling (s) and translating (v_{center}) the 2D vector v to match size and displacement of the field of view of the camera (intrinsic parameters).

$$v''' = v'' \cdot s + v_{center} \tag{3}$$

These three steps describe the projection of a given 3D point onto its pixel coordinates in the image. But they require knowledge of the extrinsic and intrinsic parameters.
One possibility to calculate the camera parameters is to locate lights with known 3D coordinates (v_{3D}) in the camera images and to optimize the camera parameters so that calculated 2D coordinates (v_{calc}) and located 2D coordinates (v_{loc}) match.

$$v_{loc,left} = projection_{left}(v_{3D}) \quad v_{loc,right} = projection_{right}(v_{3D}) \tag{4}$$

Unfortunately, this requires a special system of calibration lights with known 3D coordinates (i.e. eight lights along the edges of a cube). It would be great if the knowledge of the 3D coordinates is not necessary.
Every used calibration light has three unknown parameters (v_{3D}), but adds four new equations (two 2D vector equations). So effectively, every calibration light provides one new piece of information. If enough lights are used, the set of equations is solvable.

Our system is used as follows: First we use a flashlight to draw circles around the cameras. The light point is tracked automatically in both camera pictures. Figure 5 shows an accumulation of all pictures captured during calibration. After recording n pictures we have n pixel coordinates in

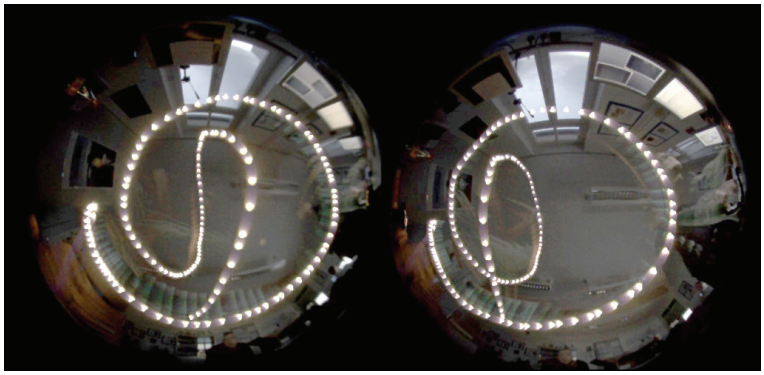

Figure 5: Accumulation of calibration pictures

the left image corresponding to n pixel coordinates in the right image. Now we estimate the 3D coordinates of the n light points and the intrinsic and extrinsic parameters of both cameras. The 3D coordinates are projected onto the left and right images in consideration of the estimated intrinsic and extrinsic parameters. Then the error of our approximation is calculated. The error value is simply calculated by summation of the squared distances between the estimated and the measured 2D pixel coordinates.
To minimize the error we have to optimize the unknown variables (3D coordinates of the n light points and the camera parameters) with a simple conjugated gradient optimization method [Pre92]. When the optimization error achieves the minimum (after 2 seconds for 10 calibration lights) a very good approximation of the intrinsic and extrinsic parameters is found. The resulting 3D coordinates of the calibration light points are not used.
The set of equations is not uniquely solvable. The following assumptions had to be made:

1. The focal point of the left camera is located at coordinates $(-1,0,0)^T$.
2. The focal point of the right camera is located at coordinates $(1,0,0)^T$.
3. The left camera can not rotate around the axis through the focal points. So it is pointed upwards, can only tilt toward or off the right camera and rotate around their camera axis.

5.3 Epipolar Geometry

After detecting the pixel coordinates of the lights in the left camera image, the corresponding epipolar lines in the right image are calculated. Unfortunately, there is no direct equation for these epipolar lines. The only way to find them is to calculate the related view ray for a pixel in the left

image as a 3D vector and project 3D points of this vector onto the right image. This is obviously inefficient, because many steps with complex trigonometric functions slow the algorithm down. To assure real time performance, some parameters are precomputed. The basic idea is to dissect the space into epipolar planes, (fig. 6) so every 3D point in space belongs to one of these planes. Additionally, each pixel in both images belongs to one epipolar curve and therefore to one epipolar plane. The following arrays are precomputed once after 3D calibration:

Figure 6: Epipolar planes, light source samples and corresponding epipolar curves

1. An array that assigns to each pixel in the left image the corresponding epipolar plane. In our case, we use 500 planes over 180 degrees, so that there is a plane every 0.35 degrees.
2. An array that assigns to each epipolar plane the coordinates of all the pixels of the epipolar line in the right image. Therefore, epipolar lines do not have to be calculated anymore, because all pixel coordinates are precomputed and stored in this array.
3. An array that assigns to each pixel in the left image the length of the epipolar line in the right image. I.e.: A pixel in the left image corresponds to an epipolar plane, that corresponds to an epipolar line, that consists of 500 pixel coordinates. But the pixel in the left image corresponds to an epipolar line, that consists of only 250 pixel coordinates, because the 250th pixel matches the infinitely far point of the view ray.
4. An array that assigns to each pixel in the left image the corresponding direction vector of the view ray.
5. An array that assigns to each pixel in the left image the angle of inclination of the view ray.
6. An array that assigns to each pixel in the right image the angle of inclination of the view ray.

Approximately 500 equidistant epipolar planes are placed into the half space. Afterwards, for each located light source the epipolar curve is looked up in the precomputed arrays. The resulting epipolar curve is sampled to find the most similar pixel, depending on color. Finally, the 3D position can be easily calculated by using the last three arrays.

6 Rendering

The rendered image is composed of three parts. First, the video image captured by the webcam. Secondly, virtual objects lit with the light information retrieved in the previous steps are put into the captured scene. And third, shadows cast by the virtual objects are added.
Capturing the webcam image is greatly simplified using ARToolKit [KB99]. It provides methods for capturing and rendering a webcam image, as well as a simple pattern recognition. Virtual objects can be placed into the real environment if a certain pattern is recognized.
We do not want iterations of our multi pass algorithm to be seen on the screen. Therefore an off-screen rendering context is used to store all iterations – starting with the webcam image. Pbuffers are common render-to-texture contexts. The class RenderTexture[1] uses pbuffers and provides a convenient off-screen rendering context in OpenGL, including color, depth and stencil buffers. 32-bit buffers are used to handle the HDR input data.
If ARToolkit detects a marker, virtual geometry is superimposed on the webcam image. Geometry is loaded via a simple VRML-loader, which parses and processes basic VRML-files. Virtual objects are illuminated using the phong [Pho75] lighting model. All lighting calculations are implemented using Cg [MGAK03] as a shading language. In order to blend the geometry realistically into the environment, all light information passed to the shaders, such as light position and intensity as well as the global ambient light component are retrieved from the HDR cameras and thus reflect the actual illumination of the object's environment. Our graphic engine can handle an arbitrary number of point light sources. For every light the calculations are made separately by the phong shader. The result is virtual geometry lit by one point light source which is stored in the RenderTexture buffer. A second RenderTexture object accumulates the results of the lighting and shadow calculations (see below), adding the current rendering pass to the overall scene.
Shadows are implemented using stencil shadow volumes. Here the stencil buffer provided by the RenderTexture class is useful as off-screen shadow volumes can be implemented straightforward. Because there is no 3D representation of the actual environment, a plane is drawn into the RenderTexture's depth buffer. This plane is coplanar with the ARToolkit marker and used for all shadow calculations. As a setback, the surrounding area of the marker has to be planar for the shadows to be realistic because all shadows are cast onto this virtual plane.
Until now, HDR values provided by the HDR cameras are used to perform all lighting calculations. In order to render the final image to the screen, these HDR values must be mapped to displayable values. Conventional screens are able to display RGB values in the range [0,1]. A simple tone mapping shader maps the HDR values into this range.

$$r' = (\frac{r_{HDR}}{lum_{max}})^{\gamma} \tag{5}$$

[1] http://www.markmark.net/misc/rendertexture.html

To retrieve the final RGB values, exemplary the red component r' in the formula above, the HDR values r_{HDR} are divided by the maximum luminance lum_{max} found in the accumulated scene. The gamma exponent γ adjusts the tone mapping graph avoiding a linear mapping of the HDR values to enhance the overall appearance of the image.

7 Results

Our final system is interactive on a PC with Pentium IV 3.2GHz CPU and GeForce 6600GT GPU. It is possible to detect up to 128 light sources interactively. To find these light sources we are working with around 20000 samples that correspond to 10% of the image pixels. Real 3D positions of all these lights will be found and will affect illumination as well as shadows. Figure 7 shows the original scene with a self-made cube on the left and the augmented scene on the right. As can be seen from figure 8 the effect of real 3D positions changes size and position of the shadow. The left part shows a light close to the cube and the right part shows the same light faraway of the cube. Using 12 light sources we achieve a framerate of 5.5fps. Finding and displaying 128 light sources achieves a framerate of 0.8fps. The system is theoretically limited by the 32-bit buffer adding up the HDR values of all lights. But given a maximum HDR value of one light around 4000, the maximum number of lights would be at least more than one million. In practice the interactive ability of our system is lost a lot earlier, depending primarily on our graphic hardware equipment, for example the number of pixel pipelines or GPU frequency.

Figure 7: The left image shows the real scene, with a cube made of paper, the right image shows the same scene with a virtual cube. The shadow is nearly the same.

Figure 8: The image on the left shows the virtual cube with its shadow and a light source very close to it, the image on the right shows the same scene with the light source farther away from the virtual cube. Note the non-parallel shadow boundaries.

8 Conclusions & Future Work

We introduced a system for interactive illumination of virtual objects in real environment. Light sources will be recognized and 3D positions are computed. Shadows change in size and appearance depending on the object-light source-distance. The implemented and explained sampling algorithms to find the light sources of the environment was sufficient for testing and developing the whole project. For further research more and better sampling algorithms should be implemented. Lights need to be more stable without losing the ability of finding moving lights so more complex algorithms should be used. At the moment, all lights are represented as point lights. An implementation of area lights to represent large light sources in conjunction with soft shadow algorithms would result in an appearance much closer to reality. More complex tone mapping algorithms would also enhance the scene. For example an algorithm which considers the camera response function to adapt the low dynamic range webcam image to the HDR illumination of the virtual geometry. In order to handle all kinds of VRML-files, a more extensive and reliable VRML-loader is required. Furthermore it would be possible to move the marker of the virtual object if we track the HDR cameras with an additional marker. These two markers have to be visible in the webcam image.

References

[Deb98] Paul Debevec. Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. *Computer Graphics*, 32(Annual Conference Series):189–198, 1998.

[Fau93] Olivier Faugeras. *Three-dimensional computer vision: a geometric viewpoint*. MIT Press, Cambridge, MA, USA, 1993.

[FGR93] A. Fournier, A. S. Gunawan, and C. Romanzin. Common illumination between real and computer generated scenes. In *Graphics Interface*, pages 254–262, 1993.

[GCHH03] Simon Gibson, Jon Cook, Toby Howard, and Roger Hubbold. Rapid shadow generation in real-world lighting environments. In P. Christensen and D. Cohen-Or, editors, *Proceedings of the 14th Eurographics Workshop on Rendering*, pages 219–229, Aire-la-Ville, Switzerland, June 25–27 2003. Eurographics Association.

[HSK+05] Vlastimil Havran, Miloslaw Smyk, Grzegorz Krawczyk, Karol Myszkowski, and Hans-Peter Seidel. Importance Sampling for Video Environment Maps. In Kavita Bala and Philip Dutré, editors, *Eurographics Symposium on Rendering 2005*, pages 31–42, Konstanz, Germany, 2005. ACM SIGGRAPH.

[KB99] Hirokazu Kato and Mark Billinghurst. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. *iwar*, 00:85, 1999.

[MGAK03] William R. Mark, R. Steven Glanville, Kurt Akeley, and Mark J. Kilgard. Cg: a system for programming graphics hardware in a c-like language. *ACM Trans. Graph.*, 22(3):896–907, 2003.

[ODJ04] Victor Ostromoukhov, Charles Donohue, and Pierre-Marc Jodoin. Fast hierarchical importance sampling with blue noise properties. *ACM Trans. Graph.*, 23(3):488–495, 2004.

[Pho75] Bui Tuong Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6):311–317, 1975.

[Pre92] William H. Press. Conjugate gradient. In *Numerical Recipes in C. The Art of Scientific Computing*. Cambridge University Press, 1992.

[SSI99] Imari Sato, Yoichi Sato, and Katsushi Ikeuchi. Acquiring a radiance distribution to superimpose virtual objects onto a real scene. *IEEE Trans. Vis. Comput. Graph.*, 5(1):1–12, 1999.

[Zha00] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.