

Fast Hologram Synthesis for 3D Geometry Models using Graphics Hardware

Christoph Petz and Marcus Magnor

Max-Planck-Institut für Informatik
Graphics - Optics - Vision
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
Email: petz@mpii.de

ABSTRACT

The holographic visualization of three-dimensional object geometry still represents a major challenge in computational holography research. Besides the development of suitable holographic display devices, the fast calculation of the hologram's interference pattern for complex-shaped three-dimensional objects is an important pre-requisite of any interactive holographic display system. We present a fast method for rendering full-parallax holograms using a standard PC with a consumer-market graphics card. To calculate the hologram of a 3D object, scaled and translated versions of the interference pattern of simple primitives, e.g. point sources, are superimposed. The hologram is built up completely on-board the graphics card. To avoid numerical inaccuracies due to limited frame-buffer resolution, we use a hierarchical approach. Using an NVidia Geforce4 graphics card, the proposed algorithm takes 1.0 second to calculate the 512x512-pixel hologram of 1024 primitives.

Keywords: Graphics Hardware, Complex Texture blending, Synthetic Holograms

1. INTRODUCTION

Holography¹ is an intriguing technique for visualizing 3D objects. When viewing a hologram, the light wave emerging from an object is reproduced exactly, thus giving the observer a realistic depth impression of the recorded 3D scene. The visualization of virtual, computer-generated 3D geometry models using a kind of holographic display is an appealing task. Besides the development of holographic display devices, the great computational requirements for calculating the holographic pattern represents a scientific challenge.

In the early days of creating computer generated holograms, the calculated hologram pattern was recorded on a piece of film, scaled and chemically processed before being viewed. The generation of the holographic pattern was measured in hours or days. For an interactive holographic display,² however, interactive frame rates are needed.

During the last years, low-end graphics hardware improved considerably in speed and programmability. Today's graphics cards represent a highly parallel processing scheme with high bandwidth to graphics memory. Investigating the generation of holograms on graphics hardware therefore seems a promising endeavour.

Lucente and Galyean³ were the first who used a graphics workstation for hologram generation. They reduced the computational effort by creating holograms without vertical parallax. They superimposed real-valued wavefields⁴ using the workstation's accumulation buffer. Ritter et al.⁵ introduced an algorithm for the complex-valued calculation of full parallax holograms using a graphics workstation's accumulation buffer as well. For the superposition of 2D complex-valued wavefields, they encode these fields into texture images. Each field, is rendered to the framebuffer, and the result is added to the accumulation buffer. The accumulation buffer usually has a higher accuracy, e.g., 25 bits compared to the framebuffer's accuracy of 8 bits. Therefore, the superposition of a large number of 8-bit wave fields can be computed without loss of accuracy. However, only high-end and expensive graphics workstations exhibit an accumulation buffer. To generate holograms on consumer-market graphics cards, only 8-bit accurate calculations are currently available.

In this paper, we present a new method for fast generation of synthetic holograms using the graphics hardware of a standard PC. We also encode complex wave fields into texture images for the superposition of these fields, but the superposition is calculated in a new manner. We prevent inaccuracy problems by utilizing a hierarchical

method for combining the wave fields. A number of wave fields are combined in a single rendering step by exploiting the graphics hardware multitexturing capabilities. Our method doesn't need a hardware accelerated accumulation buffer.

The paper is organized as follows. In Sect. 2 we give a short introduction to the theory of computer generated holography. We present our algorithm for hologram generation in Sect. 3. Afterwards, in Sect. 4 we give details of the implementation using graphics hardware. Experimental results are presented in Sect. 5. We conclude with a review of our results in Sect. 6.

2. HOLOGRAM CREATION

The process of generating *computer generated holograms* is based on appropriately simulating the physical hologram recording process. To create a hologram of an arbitrary object, the beam of a coherent light source is divided in two parts: One part illuminates the object, the other part serves as reference beam. The reflected light from the object and the light from the reference beam interfere in the hologram plane, and a photographic plate records the interference pattern. The hologram is obtained by developing the photographic plate. The main concern in synthetic hologram generation is the interference pattern simulation of the light of the complex-valued object wave with the reference beam in the hologram plane.

For our calculations, the hologram is located in the xy -plane, and the desired object is placed near the z -axis at a distance z_0 , illuminated by a coherent light source. The complex reference wave in the hologram plane is $R(x, y)$, and the wave reflected by the object is denoted by $O(x, y)$. The encoded hologram is then:

$$H(x, y) = |O(x, y) + R(x, y)|^2 \quad (1)$$

The complex-valued wave field $O(x, y)$ in the hologram plane emanating from the object's surface can be expressed using the Rayleigh-Sommerfeld diffraction formula⁶

$$O(x, y) = \frac{1}{i\lambda} \int_S o(\vec{s}) \frac{\exp(ik|\vec{r}(\vec{s})|)}{|\vec{r}(\vec{s})|} \cos \alpha d\vec{s}, \quad (2)$$

where $o(\vec{s})$ is the reflected field strength from the object surface point \vec{s} , \vec{r} is the vector pointing from \vec{s} to (x, y) in the hologram plane, and α is the angle between incident object illumination and \vec{r} . The light wavelength is λ , and $k = 2\pi/\lambda$ denotes the wave number.

Assuming that the distance z_0 between object and hologram is large compared to the size of the hologram, $\cos \alpha$ and the distance $|\vec{r}|$ in the denominator can be considered constant over the entire hologram area. The field strength $o(\vec{s})$ can then incorporate these constant factors. Discretization of the integral formula Eq. (2) then leads to the expression

$$O(x, y) = \sum_{\vec{s}_i \in S} o(\vec{s}_i) \exp(ik|\vec{r}(\vec{s}_i)|) \quad (3)$$

The sum is defined over a set S of sample points on the object's surface. According to Huygen's principle, the object wave can be approximated by the superposition of the elementary wavefronts emerging from the sample points.

The complex wave pattern of a point source in the hologram plane is called a Fresnel-Zone-Plate (FZP) (see Fig. 1). It consists of concentric rings around the (x, y) position of the point source. The ratio of the n th ring of two point sources p_r and p having distances z_r and z from the hologram plane is

$$\frac{\text{radius}(p, n)}{\text{radius}(p_r, n)} = \sqrt{\frac{z}{z_r}} \sqrt{1 + \frac{n\lambda(z_r - z)}{n\lambda z + 2zz_r}} \approx \sqrt{\frac{z}{z_r}}. \quad (4)$$

Assuming $z, z_r \gg \lambda$, the ratio can be approximated by $\sqrt{z/z_r}$, independently of n . For example, for $n = 500$, $\lambda = 632 * 10^{-9} m$, $z = 0.1 m$ and $z_r = 0.2 m$ the approximation error is only 0.04%. Therefore, a translation of the point source along the z direction can be approximated by simply scaling the original FZP. The complete object wave $O(x, y)$ in Eq. (3) can then be constructed by superimposing scaled and translated versions of only one reference FZP.

3. FAST HOLOGRAM SYNTHESIS ON GRAPHICS HARDWARE

Our algorithm computes the hologram of an object by decomposing it into a set S of primitives. The complex waves of these primitives are superimposed in the hologram plane. Afterwards, the reference wave is added. To obtain the final hologram, the intensity of the wave pattern is calculated. Our algorithm is tailored to use off-the-shelf graphics hardware and can be implemented using standard *OpenGL*.⁷

In a nutshell, the algorithm consists of the following steps:

1. Create the complex wave patterns for all primitives.
2. Hierarchically superimpose these wave patterns.
3. Add a reference wave.
4. Calculate the interference pattern's intensity.

Steps 1 and 2 are the most costly operations and are therefore of greatest interest for graphics-hardware acceleration. While steps 3 and 4 are not as time-critical, we are also able to compute these steps entirely in graphics hardware, thus calculating the entire hologram on the graphics card. Given a holographic display device, it could be directly hooked up to the graphics card's display output.

3.1. Primitive wave field

The object wave of an extended object is the superposition of the wave field emerging from the primitives $s \in S$ the object is decomposed into. Therefore, the calculation of the wave field for each primitive is the starting point for calculating the object wave. We describe in detail the wave field calculation of an object built up by a number of point primitives. The extension to primitives representing line segments or even triangular elements can be done in the same way.^{5,8,9} In compliance with Ritter et al.,⁹ we call the interference pattern of a primitive its *holographic equivalent*.

Our algorithm computes the hologram of a set S of sample points on the object's surface. The holographic equivalent of a sample point can be approximated by translating and scaling the pre-calculated pattern of a reference point, Sect. 2. Let $p_i \in S$ be an enumeration of the sample points and $N = |S|$ the cardinality of S . A sample point $p_i = (x_i, y_i, z_i, a_i)$ consists of the sample point coordinates and the amplitude of the emitting field strength a_i . The hologram is located in the xy -plane and centered around $(0, 0, 0)$. The distance between two pixels in the hologram plane is δ , the light wavelength is λ .

The complex wave field of a reference point $(0, 0, z_r, 1)$ in the xy plane is a FZP centered around the origin. In a pre-computation step, the reference point's wave pattern is created by calculating the phase function in the xy -plane according to Eq. (3). The choice of z_r is arbitrary, but for highest accuracy, it should be similar to the depth of the sample points p_i . Due to symmetry, only a quarter of the pattern has to be stored. The pre-calculated pattern should be larger than the desired hologram size. The complex-valued wave pattern in the hologram plane is stored in two two-dimensional arrays corresponding to the real and the imaginary part.

The wave pattern of an arbitrary sample point $p_i = (x_i, y_i, z_i, a_i)$ can be approximated from the reference point pattern: x_i and y_i correspond to a translation of the FZP, while the distance z_i yields a scaling of the pattern by $\sqrt{z_i/z_r}$, Eq. (4). The field strength can be adjusted by a simple multiplication of each sample point with a_i . In Sect. 4 we show that these operations can be performed very efficiently using graphics hardware.

The wave pattern of p_i in the hologram plane for a hologram of width h_w and height h_h is then a translated, scaled and clipped version of the pre-computed reference point pattern. The rectangular region with the coordinates

$$\begin{array}{ll}
 \text{upper left corner} & \left(x_i - \frac{h_w}{2} \sqrt{\frac{z_i}{z_r}}, y_i - \frac{h_h}{2} \sqrt{\frac{z_i}{z_r}}\right) \\
 \text{lower right corner} & \left(x_i + \frac{h_w}{2} \sqrt{\frac{z_i}{z_r}}, y_i + \frac{h_h}{2} \sqrt{\frac{z_i}{z_r}}\right)
 \end{array} \tag{5}$$

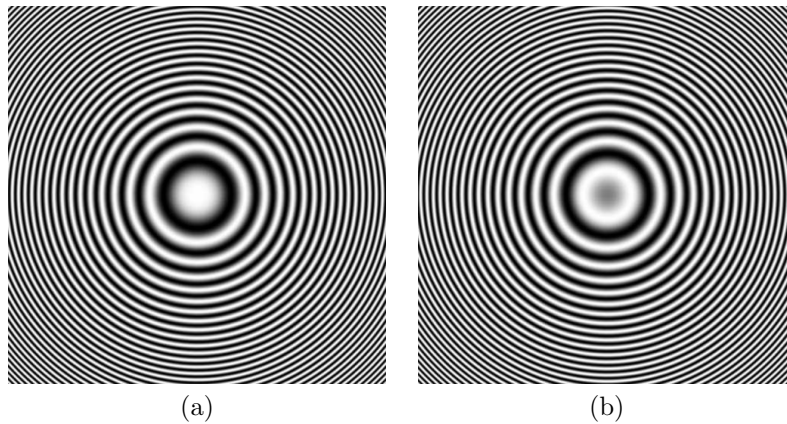


Figure 1. the real and imaginary part of a fresnel zone plate

is copied to the hologram plane, thus yielding the holographic equivalent of p_i .

To improve the quality of computer generated holograms for display purposes, it is advantageous to select the phase of the light emitted from an object's surface point appropriately. To simulate a rough object surface, the phase distribution of the sample points are chosen randomly. Therefore, a set of different FZPs for the reference point is constructed with different phase values, so the phase can be varied.

3.2. Hierarchical superposition of wave fields

With the set of holographic equivalents H_0 created so far, all parts of the final object wave are available. The remaining task is the superposition of the wave fields. The main concern is the limited 8-bit computational accuracy provided on graphics hardware. To deal with this restriction, and at the same time to preserve the characteristics of the primitive wave fields in the final object wave, we propose a scheme of hierarchical superposition.

The data format provided by graphics hardware is commonly 8-bit, fixed-point. Within this limitation we want to calculate the superposition of all holographic equivalents,

$$O(x, y) \propto \sum_{h_i \in H_0} h_i(x, y) \tag{6}$$

For good hologram quality, the number of sample points must be much greater than 256. Therefore, the impact of a single element h_i on O is less than one bit in the data type's precision, hence a direct summation as suggested by Eq. (6) cannot yield the desired result.

Our scheme of hierarchical superposition makes it possible to combine an arbitrarily large number of wave fields. Starting with the set H_0 of holographic equivalents, we superimpose groups of T elements at a time. Thus, we obtain a smaller set H_1 of wave fields, which elements are each the superposition of T elements of H_0 . This process is iterated, until a single element, the complete object wave remains. In every combining step, the number of involved input fields T is small, so rounding errors due to the 8-bit precision is small. However, more memory is required to store the intermediate results.

Crucial for the quality of the resulting hologram is high contrast. Combining a very large number of primitive wave fields nearly averages out the resulting wave to zero. However, it is these finely grained differences that makes up the holographic information. The fixed-point data format is not capable to account for this automatically. Therefore, every superposition operation should re-scale the intermediate result to exploit the full range of the data type. To ensure that every initial holographic equivalent has the same impact on the final result, the scaling factor must be constant for all superpositions going from $H_i \mapsto H_{i+1}$.

3.3. Final processing step

Compared to calculating the object wave field, the remaining steps to finish the hologram are of little account. Even so, it is possible to perform the whole hologram generation process entirely in graphics hardware, programmed using standard OpenGL without vendor-specific extensions.

The superposition of a reference wave to the calculated object wave is done in just the same way as combining the primitives' complex wave fields before. The reference wave can be seen as another holographic equivalent, except that it is superimposed with the final object wave. It can be of arbitrary shape and orientation, e.g., an inclined planar wave or a wave of a point source. The final hologram is obtained by calculating the intensity of the wave field in the hologram plane.

4. GRAPHICS HARDWARE

For hologram generation, the main challenge is the superposition of a huge number of complex wavefields as denoted by Eq. (3). It is appealing to exploit the capabilities of modern graphics hardware for this purpose. A modern GPU has much greater memory bandwidth than a modern CPU. For many tasks, the graphics hardware has parallel processing units, and many operations work with vector-valued data. Graphics hardware is optimized for rendering geometry for a 2D output device and is therefore not as flexibly programmable as a normal processor. In this section, we show in detail which operations can be used to map graphics hardware capabilities to hologram generation.

Important for encoding performance is the number of available texture units according to the OpenGL variable `MAX_TEXTURE_UNITS`. We denote this hardware constant by T . The color depth of the framebuffer influences the precision of the calculation. Most graphics hardware support a maximum color depth of 8 bits per channel. The color resolution of the texture format should match those of the framebuffer. Color is commonly specified in RGBA format, yielding 4 parallel 8-bit channels to work with.

4.1. Multi-texturing

In our proposed algorithm, we rely on the multi-texturing capabilities of OpenGL. For multi-texturing, more than one set of texture coordinates can be assigned to a vertex. The maximum number of textures per vertex depends on the graphics card's number of available texture units T . The texture units are arranged sequentially in the rendering pipeline. Each fragment passes them one by one, and each unit can change the fragment color by calculating on its input values. The input to a texture unit is the fragment's primary color, the previous texture unit's output and the current texture value. Depending on its configuration, each texture unit computes one of a fixed set of operations.

We use multi-texturing to perform arithmetics with arrays of numbers. For generating synthetic holograms, we need to implement the following two equations:

$$R_1(x, y) := \sum_{i=1}^T A_i(x, y) \quad (7)$$

$$\text{and } R_2(x, y) := \sum_{i=1}^T \frac{1}{T} \alpha_i A_i(x, y) \quad (8)$$

where T is the number of available texture units, the input fields are assumed to be $A_i(x, y) \in [-1, 1]$, and factors α_i are restricted to $[0, 1]$ because of a technical reason described in more detail later.

Due to the graphics card color resolution limitation, the data types of the components of the input arrays A_i and the result arrays R_i are 8-bit, fixed-point numbers with the co-domain $[-1, 1]$. Therefore, the accuracy of the operations are limited. In case of Eq. (7), all components of the result are clamped to $[-1, 1]$, so the array R_1 can only be evaluated accurately if the components of the arrays A_i are appropriate small. Because of the division in Eq. (8), R_2 always exhibits rounding inaccuracies in the last digit.

The computation is performed using OpenGL's rendering capabilities. The width and the height of the framebuffer must match the input array sizes. We render a rectangle R covering the entire framebuffer, e.g.

1024 x 1024 pixels for a hologram of this size. The input fields $A_i(x, y)$ are coded into images and stored in the texture units. The texture coordinates of R are set such that each texture image covers the entire rectangle. The calculation is performed on a per-pixel basis. For each pixel, the corresponding values of the arrays A_i are assigned to the current texture values of the texture units. The calculation depends on the configuration of the texture environments. The result is calculated in one rendering pass and stored in the framebuffer.

The field A_i is encoded in one color channel of the texture image. All color values have to be in the range $[0, 1]$, so we scale and bias the input values accordingly. Each color channel is treated independently, and it is possible to evaluate the expressions with vector-valued input fields. In our application for hologram generation, we encode the real and imaginary part of the complex-valued wave pattern in the red and green color channels, respectively.

For Eq. (7), the texture units are configured to perform the operation `GL_ADD_SIGNED` with the current texture value and the value of the last stage as parameters. This operation performs an addition, interpreting the parameters as signed values. The first texture unit just returns the texture value.

For the calculation of Eq. (8), the texture units are configured to perform the operation `GL_INTERPOLATE`, performing

$$Arg_0 * Arg_2 + Arg_1 * (1 - Arg_2). \quad (9)$$

The arguments are set as follows: Arg_0 is the value of the current texture, Arg_1 the output of the previous texture unit or the fragment's primary color for the first texture unit. Arg_2 is a constant, specific to each texture unit. For the i th texture unit, the constant is

$$Arg_2^i = \begin{cases} \frac{\alpha_i}{T} * \prod_{k=i+1}^T \frac{1}{1 - Arg_2^k} & : \quad 1 \leq i < T \\ \frac{\alpha_T}{T} & : \quad i = T \end{cases} \quad (10)$$

Because Arg_2 is restricted to $[0, 1]$, α_i is also restricted to this interval. The primary color is set to 0.5 to correct the shifted value of zero. An excerpt of the sourcecode can be seen in Fig. 2.

So far, we have shown how to implement two operations for combining up to T vector-valued arrays on graphics hardware in a single rendering step. The input wave fields must be stored as texture images. Compared to an implementation running on a CPU, the execution time is very fast. We use these operations to superimpose complex-valued wave fields. With operation Eq. (8) we can weight the input wave fields. Unfortunately, the factors α_i are restricted to $[0, 1]$, so there is no continuous transition to Eq. (7). As we show in the next section, both operations are used in tandem for creating the hologram.

4.2. Hologram generation

In this section, we explain how our hologram generation algorithm is implemented. Starting with a set S of primitives, we create the according set H_0 of holographic equivalents on graphics hardware by translating, scaling and clipping the reference point's FZP, Sect. 3.1. The elements of H_i are superimposed groupwise to create the set H_{i+1} , until a single holographic equivalent for all elements of S , the object wave field, remains. Finally, the reference wave is superimposed, and the intensity of the resulting wave pattern is calculated to yield the final hologram.

In a preparation step, the wave field of a reference FZP is calculated and encoded into a texture image. This reference wave field must be large enough to cover the clipped patterns for all point-primitives of S . The pattern for each point is obtained by scaling and translating the reference FZP as described in Sect. 3.1 on the graphics card. Due to the `GL_MIRRORED_REPEAT` texture wrap mode, only a quarter of the reference wave has to be stored in graphics memory. This preparation step has to be done only once for an arbitrary number of generated holograms.

The elements of set H_0 are not stored explicitly, but they are created on-the-fly when needed. We use the superposition operation Eq. (8). The input wave fields are given as texture coordinates of the reference FZP's

Resolution	# of primitives	Rendering time (s)
512 × 512	1,024	0.96
512 × 512	4,096	2.82
512 × 512	16,384	10.74
512 × 512	65,536	41.7
1024 × 1024	1,024	3.86
1024 × 1024	4,096	11.1
1024 × 1024	16,384	55
1024 × 1024	65,536	270

Table 1. Dependency of rendering time on resolution and number of primitives. The times are measured for an nVidia GeForce 4 Ti 4600.

texture image. The factors α_i are set according to the brightness of the sample points. In one step, T elements of S are combined into one element of H_1 .

In the subsequent merging step from H_i to H_{i+1} , we can decide whether to use Eq. (7) or Eq. (8) as the superposition operation. In the case of simple addition Eq. (7), the contrast of the result will be enhanced by a factor of T . However, the result must fit into the range $[-1, 1]$ in order to prevent clamping. Otherwise, the operation Eq. (8) with $\alpha_1 = 1$ averages the input patterns, with the drawback of losing some accuracy in the last digit.

When simply combining a large number of wave fields, the wave pattern very likely averages out to zero. Since the theoretical maximum is very unlikely, however, in some of the transitions, contrast enhancement can be performed without exceeding the available range of values. The accuracy error rises with the number of times Eq. (8) is applied.

The superposition of the whole set H_0 of holographic equivalents is implemented recursively, thus achieving a hierarchical summation. The intermediate results are stored as texture images with the same size as the final hologram. In the best case, each combining step combines T wave fields to exploit the graphics hardware's number of texture units most efficiently. The number of required textures for intermediate results is then $1 + T * (\log_T(N) - 1)$. The recursion depth is limited to $\lceil \log_T(N) \rceil$.

The wave pattern of the reference wave is encoded into a texture and superimposed with the calculated object wave. Afterwards, the intensity in the hologram plane must be obtained. Because the calculation of $|u + iv|^2 = u^2 + v^2$ can be separated in the real and the imaginary part, the OpenGL *pixelMap* functionality can be used for mapping $x \mapsto x^2$, and the sum of the mapping is read out of the framebuffer using GL_LUMINANCE.

In the case of a planar, perpendicular reference wave, the steps of adding the reference wave and mapping the complex field to intensities can be combined into the mapping step, because the reference wave is then independent of the x and y position in the hologram plane.

5. RESULTS

Our hardware-accelerated algorithm for hologram generation renders a 512x512 pixel hologram of 1024 point samples in 960 ms, using an nVidia GeForce 4 Ti 4600 graphics card. The algorithm scales about linearly in the number of point samples as well as in hologram size, as can be seen in Table 1. Compared to the results published by Ritter et al.,⁹ our algorithm computes the hologram of point sources considerably faster.

Holograms generated for display purposes must be built up from a very large number of sample points. The calculation round-off error should be as small as possible. We presented an algorithm for combining a large number of holographic equivalents using graphics hardware despite the limitation to 8-bit resolution. The calculation of the object wave is the crucial part in evaluating the accuracy of the hologram. The remaining hologram-generation steps, adding a reference wave and calculating the intensity, are the same for every calculated object wave, and thus introduce a constant error.

Input: texture = array of input fields
 T = number of field to add
 α_i are assumed to be 1
 Output: framebuffer = result of the operation.

```
glClear(GL_COLOR_BUFFER_BIT);
forEach texture  $i \in [0, T - 1]$ :
    glActiveTexture(GL_TEXTURE0 + i);
    glBindTexture(GL_TEXTURE_2D, texture[i]);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_COMBINE);
```

in the case of simple addition Eq. (7):

```
if ( $i = 0$ ):
    glTexEnvf(GL_TEXTURE_ENV, GL_COMBINE_RGB, GL_REPLACE);
else ( $i \neq 0$ ):
    glTexEnvf(GL_TEXTURE_ENV, GL_COMBINE_RGB, GL_ADD_SIGNED);
```

in the case of weighted average Eq. (8):

```
glTexEnvfv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_COLOR, 1.0/( $i + 1$ ));
if ( $i = 0$ ):
    glTexEnvf(GL_TEXTURE_ENV, GL_SOURCE1_RGB, GL_PRIMARY_COLOR);
    glTexEnvf(GL_TEXTURE_ENV, GL_OPERAND1_RGB, GL_SRC_COLOR);
glTexEnvf(GL_TEXTURE_ENV, GL_COMBINE_RGB, GL_INTERPOLATE);
glTexEnvf(GL_TEXTURE_ENV, GL_SOURCE2_RGB, GL_CONSTANT);
glTexEnvf(GL_TEXTURE_ENV, GL_OPERAND2_RGB, GL_SRC_ALPHA);
```

in both cases:

```
glBegin(GL_POLYGON);
glColor4f(0.5, 0.5, 0.5, 0.5);
forEach texture  $i \in [0, T - 1]$ : glMultiTexCoord2f(GL_TEXTURE0+i, 0.0, 0.0);
glVertex2f(-1.0, -1.0);
forEach texture  $i \in [0, T - 1]$ : glMultiTexCoord2f(GL_TEXTURE0+i, 1.0, 0.0);
glVertex2f(1.0, -1.0);
forEach texture  $i \in [0, T - 1]$ : glMultiTexCoord2f(GL_TEXTURE0+i, 1.0, 1.0);
glVertex2f(1.0, 1.0);
forEach texture  $i \in [0, T - 1]$ : glMultiTexCoord2f(GL_TEXTURE0+i, 0.0, 1.0);
glVertex2f(-1.0, 1.0);
glEnd();
```

Figure 2. Sourcecode with OpenGL commands for the operations Eq. (7) and Eq. (8) using multitextures.

# of point samples	avg. error	max. error	avg. error (<i>ec</i>)	max. error (<i>ec</i>)
1024	0.035	0.09	0.01	0.04
4096	0.066	0.17	0.011	0.04
16384	0.11	0.28	0.019	0.05
65536	0.17	0.44	0.014	0.05

Table 2. Comparison of the average and maximum accuracy error according to Eq. (11). The abbreviation *ec* means "enhanced contrast", and shows the influence of an appropriate scaling of the object wave on the accuracy of the calculation.

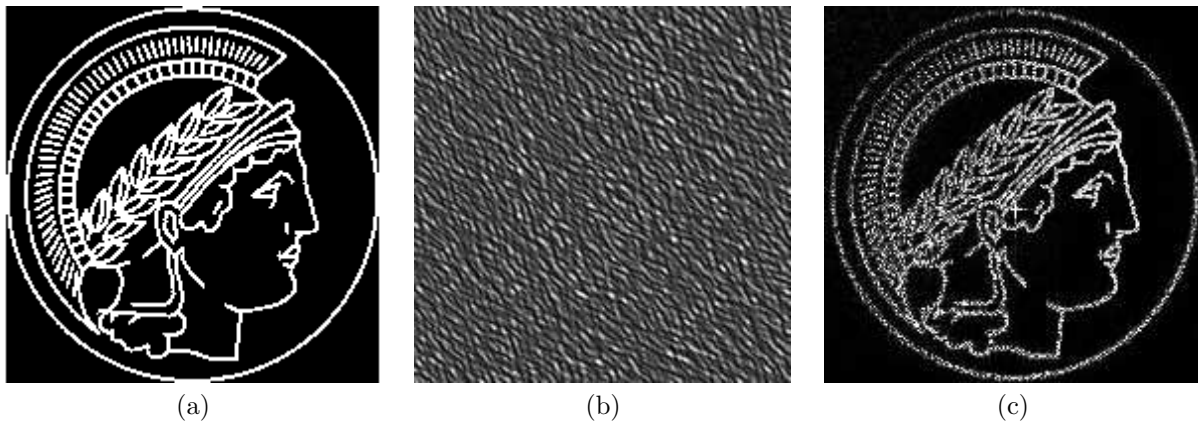


Figure 3. input image (a), hologram (b) and reconstruction (c) of an emblem of the greek goddess Minerva. The hologram is constructed with 8192 sample points in 5.8 s, and has a size of 512^2 image points. It is reconstructed using the Fresnel approximation.

For evaluating the accuracy, we compare the resulting object wave calculated by the graphics card with a calculation using a double precision floating point format on CPU. We measure the contrast error for a hologram point (x, y) as

$$\frac{|O_D(x, y) - O(x, y)|}{\max_{x', y'} |O(x', y')|}, \quad (11)$$

where $O_D(x, y)$ is the object wave calculated with double precision, and $O(x, y)$ is the object wave calculated using graphics hardware.

Table 2 indicates that our algorithm calculates the object wave with high accuracy. The average error of a superposition of 16384 holographic equivalents introduces an error of 1.9%. The contrast enhancement has a high impact on the accuracy.

Fig. 3 shows a hologram of an image and its digital reconstruction. In the hologram construction process, the image is placed opposite to the hologram plane, and a salted plane wave is used as reference wave, yielding an off-axis hologram. The digital reconstruction is calculated using the Fresnel approximation.⁶ Fig. 4 shows the reconstruction of a hologram of objects at different depths.

6. CONCLUSIONS

We have presented a novel method for creating holograms using a hardware accelerated implementation for off-the-shelf PC graphics hardware. The constructed holograms preserve full parallax and can be used for display purposes. Compared to other methods, our method does not require an accumulation buffer. The performance gain, over previously reported systems, is considerable.

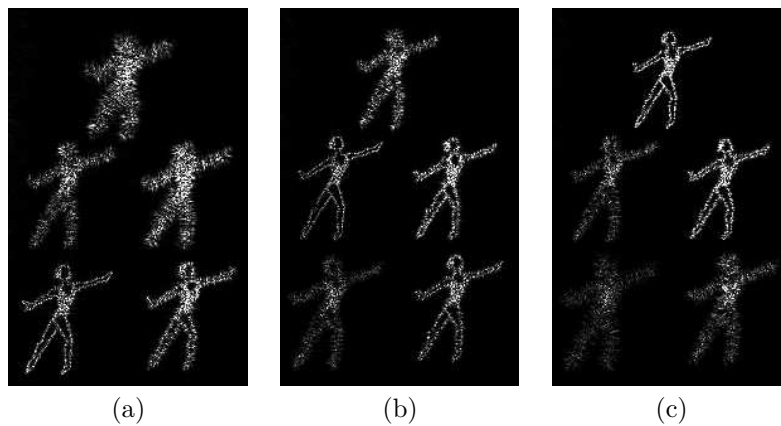


Figure 4. Fresnel approximated reconstructed view of an hologram with 5 dancers at different distances. Nearest to the hologram plane is the dancer in the lower-left corner at a distance of 1.26 cm, farthestmost is the upper dancer at 1.64 cm. The hologram is constructed with 4096 sample points in an off-axis geometry. Reconstruction: (a) nearest dancer in focus, (b) middle-left dancer in focus and (c) upper dancer in focus.

REFERENCES

1. P. Hariharan, *Optical Holography*, Cambridge University Press, 1984.
2. M. Lucente, "Interactive three-dimensional holographic displays: seeing the future in depth," *Computer Graphics* **31**(2), 1997.
3. M. Lucente and T. A. Galyean, "Rendering interactive holographic images," in *Proceedings of the Conference on Computer Graphics (SIGGRAPH-95)*, R. Cook, ed., pp. 387–394, ACM Press, August 6–11 1995.
4. A. D. Stein, Z. Wang, and J. S. Leigh, "Computer-generated holograms: A simplified ray-tracing approach," *Computers in Physics* **6**(4), pp. 389–392, 1992.
5. A. Ritter, J. Boettger, O. Deussen, M. Koenig, and T. Strothotte, "Hardware-based rendering of full-parallax synthetic holograms," *Applied Optics* **38**(8), pp. 1364–1369, 1999.
6. J. W. Goodman, *Introduction to Fourier Optics*, The McGraw-Hill Companies, 2nd ed., 1996.
7. M. Segal and K. Akeley, *The OpenGL Graphics System: A Specification (Version 1.4)*, Silicon Graphics, Inc., 2002.
8. M. Koenig, O. Deussen, and T. Strothotte, "Texture-based hologram generation using triangles," in *Proceedings of SPIE*, **4296**, 2001.
9. A. Ritter, J. Böttger, O. Deussen, and T. Strothotte, "Fast texture-based interference for synthetic holography," Tech. Rep. 3/98, Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik, 1998.