Computer generated hologram from point cloud using graphics processor

Rick H.-Y. Chen and Timothy D. Wilkinson*

Electrical Engineering Division, Department of Engineering, University of Cambridge, 9 JJ Thomson Avenue, Cambridge CB3 0FA, UK

*Corresponding author: tdw13@cam.ac.uk

Received 14 August 2009; revised 10 November 2009; accepted 27 November 2009; posted 30 November 2009 (Doc. ID 115705); published 10 December 2009

Computer generated holography is an extremely demanding and complex task when it comes to providing realistic reconstructions with full parallax, occlusion, and shadowing. We present an algorithm designed for data-parallel computing on modern graphics processing units to alleviate the computational burden. We apply Gaussian interpolation to create a continuous surface representation from discrete input object points. The algorithm maintains a potential occluder list for each individual hologram plane sample to keep the number of visibility tests to a minimum. We experimented with two approximations that simplify and accelerate occlusion computation. It is observed that letting several neighboring hologram plane samples share visibility information on object points leads to significantly faster computation without causing noticeable artifacts in the reconstructed images. Computing a reduced sample set via nonuniform sampling is also found to be an effective acceleration technique. © 2009 Optical Society of America

OCIS codes: 090.1760, 090.2870, 090.5694, 100.6890.

1. Introduction

Computer generated holography offers the possibility of producing realistic three-dimensional (3D) images of synthetic objects, but it is a hugely challenging task. Despite decades of advancements in computer technology since the inception of computer generated holograms (CGHs), the complexity of computing light transportation and interaction is such that generating holograms of photorealistic 3D scenes in real time remains unattainable. Owing to the revolutionary advancements in programmable graphics processing units (GPUs) in recent years, however, we are getting closer to achieving this goal than ever before. The power and versatility of modern graphics cards is gradually attracting the interest of the holography community, as we see an increasing number of papers reporting hologram generation carried out on GPUs [1–8].

Computer generated holography can be categorized into two general classes: the image-order and the object-order approaches. An example of the image-order approach is ray casting [5,8], where rays are cast from each hologram plane sample into the object volume, the nearest ray-object intersection points are searched for, and their contributions to the sample are calculated. The number of rays has a predominant effect on the computation time. The object-order approach, on the other hand, starts with the object primitives, be it points, lines, or planar segments, and propagates the complex field scattered by the primitives toward the hologram plane. The speed of hologram generation is influenced primarily by the primitive count. References [9–12] are examples of this approach.

Both approaches have their pros and cons. The ray-casting method scales well with object complexity and handles occlusion computation efficiently. Its problem lies in sampling. Ray casting, which is intrinsically a sampling process, is more likely than not to return a different set of ray-object intersection points to each hologram plane sample even in the absence of any occlusion. This is intuitively not an accurate depiction, as we would expect an object point, without being occluded, to contribute to a wide

© 2009 Optical Society of America

© 2009 Optical Society of America
range of samples in the hologram plane. The more commonly seen object-order approach has no such issue, but it does not facilitate occlusion. Naturally, we want to combine the best of the two in a hybrid approach. This hybrid method, while fundamentally object-ordered, derives its occlusion calculation from geometrical rays, and it is implemented under the framework of parallel computation to take advantage of GPUs’ capabilities.

This paper presents a parallel algorithm for fast synthesis of full-parallax holograms with hidden surface removal from point cloud input. The theory of CGH calculation is given in Section 2, including discussions on object modeling and light wave propagation. A brief introduction to computing with GPUs is given in Section 3. Section 4 presents an overview of the algorithm, followed by discussions on several acceleration techniques and their implementation in Section 5. We put it all together in Section 6 with pseudocodes, and we conclude with optical reconstructions and some observations in Section 7.

2. Computer Generated Hologram Theory

A. Complex Field of a Point Emitter

A synthetic 3D object is modeled as a collection of geometric primitives. Primitives that have been used for hologram generation include point emitters [1,6,7], lines [13], and polygons [10,11]. We adopt the point emitter as the primitive primarily for its simplicity in occlusion computation.

Two models exist in describing the complex field radiated by a point source at any point in space. The first model treats the primitive as an isotropic, spherical-wave-emitting point source [14]. The second model follows the diffraction theory of light and describes the field scattered from the primitive by either the Fresnel–Kirchhoff or the Rayleigh–Sommerfeld diffraction formula.

The main difference between these two treatments comes down to the obliquity factor present in the diffraction formulas, which suppresses backward radiation. The isotropic point source model assumes the primitive radiates equally in all directions, while in the diffraction based model the primitive radiates strongly in the forward direction (normal to the wavefront) but with lesser side radiation and no back radiation. Intuitively, this would seem to suggest that the diffraction based model would exhibit a more specular effect, and the isotropic point model would be more diffuse.

Rayleigh–Sommerfeld diffraction of the field radiated by a point emitter can be calculated using angular spectra of plane waves [15], as

$$ s(x', y') = F^{-1} \left\{ F \left\{ A \exp(j\varphi) \otimes \delta(x-x_0, y-y_0) \right\} \right\} \times \exp \left\{ -j\frac{2\pi}{\lambda} z_d \sqrt{1 - (\lambda \mu)^2 - (\lambda v)^2} \right\}, $$

where $F\{\cdot\}$ denotes Fourier transformation and $\otimes$ is the convolution operator; $A$ and $\varphi$ define the amplitude and phase of the emitter, $(x_0, y_0)$ is the emitter’s coordinates, $\lambda$ is the wavelength, $z_d$ is the propagation distance, and $\mu$ and $v$ are the spatial frequencies. Rearranging the expression above, we get

$$ s(x', y') = A \exp(j\varphi) F^{-1} \times \left\{ \exp \left\{ -j\frac{2\pi}{\lambda} z_d \sqrt{1 - (\lambda \mu)^2 - (\lambda v)^2} \right\} \right\} \otimes \delta(x-x_0, y-y_0). \quad (1) $$

The term inside the inverse Fourier transform brackets is the free-space transfer function, sometimes called the wave spread function (WSF). Convolving with a Dirac delta essentially shifts and centers the function around the delta impulse. Therefore, if we know the inverse Fourier transform of the WSF in advance, finding the diffracted field involves simply scaling it with the emitter’s amplitude and phase factor, followed by a shifting operation. We come back to this when we go into acceleration methods in Section 5.

B. Lighting

Lighting creates shades and shadow that form one of the key visual cues that inspires the impression of three-dimensionality. The bidirectional reflectance distribution function (BRDF) was developed to describe how an illuminating light is reflected by an opaque surface [16]. Most BRDF models separate reflection into three components: ambient, diffuse, and specular. Ambient reflection arises from light from the surrounding environment, and it radiates equally in all directions. An excellent survey of some of the most popular models for diffuse and specular reflections can be found in [17,18]. The following paragraph is devoted to a particular model to implement, namely, Lambertian reflection.

Lambertian reflectance represents perfect diffuse reflection. A Lambertian surface exhibits isotropic surface reflectance, scattering incident light equally in all directions. The Lambertian BRDF is thus a constant given by [18]

$$ \frac{I_r}{I_i} = \max\{L_i \cdot \hat{n}, 0\}, \quad (3) $$

where $I_r$ and $I_i$ are the intensity of light reflected off and incident on a surface point $p$, respectively; $L_i$ is the unit vector pointing from $p$ to the light source; $\hat{n}$ is the unit normal vector at $p$, respectively; $\otimes$ is the dot product operator. Figure 1 helps to elucidate this description. Rearranging Eq. (3) allows one to work out the intensity reflected, with the minimum bound to zero by the max{ } operator in the case that $L_i \cdot \hat{n}$ returns a negative value. Not all rough, diffuse surfaces can be adequately approximated as a Lambertian surface, but it is nonetheless a good general model.
We have incorporated ambient and Lambertian reflection into object modeling. To exhibit Lambertian reflection, the amplitude of the point primitive is weighted by \( I_n \) calculated according to Eq. (3). The result is then added on top of an ambient light value \( A_{\text{amb}} \) set by the user. It should be emphasized that no distinction is made between light emitted and reflected by the object in this work. While the constituent primitives are said to be light-emitting point sources, their amplitudes are being modified by the BRDF model. This is merely a simplification, just as we have simplified the lighting model to omit light transmission.

C. Fourier Hologram

A Fourier hologram is generated by Fourier transforming the complex field of the object in the hologram plane. Unlike holograms produced by interference with a reference wave, a Fresnel hologram for example, the Fourier hologram is a complex function; the spatial light modulator (SLM) used for displaying the hologram, however, can only represent either the amplitude or the phase information. Fortunately, the spectral phase of a signal tends to contain more information about the signal than the spectral amplitude in Fourier representation [19]. If the spectral amplitude is relatively smooth, the spectral phase carries the majority of the essential information on the nature of the signal and the amplitude can be eliminated entirely. A Fourier hologram with more or less uniform amplitude can be achieved by assigning a random, independent phase to the synthetic object data, effectively making the object diffusive [20], specifically by setting the phase \( \varphi \) of Eq. (2) random.

Although the CGHs we generated and describe in this paper are of the Fourier type, we should stress that the algorithm presented here is not restricted to Fourier holograms only. In fact, every step in the algorithm up to the last Fourier transform stage is identical for both Fourier and Fresnel holograms. Another point we emphasize is the distinction we make between “hologram pixels” and “hologram plane samples.” In our usage, the first term refers to pixels of the CGH, while the second term refers to sampling points in the \( x-y \) plane at \( z = 0 \) with no reference to the actual CGH itself.

3. Parallel Computing on the Graphics Processing Unit

The GPU is a powerful parallel machine complete with its own memory on the graphics card. A modern GPU has more than a hundred processor cores that manage multiple threads of execution concurrently. The theoretical peak performance of today’s high-end GPU is reaching one teraflops (floating point operations per second) in single precision, roughly 10 times that of a quad-core CPU [21]. For this reason, researchers from different fields have been drawn to general purpose computation on graphics processing units (GPGPU), trying to harness the power of GPUs to solve their computationally intensive problems. Hologram generation on computer is one of those problems that can benefit from GPGPU.

In the early days of GPGPU, shading languages such as GLSL were used to program the vertex and/or fragment processors of the graphics pipeline to carry out specific tasks. Programmers use graphics application program interfaces (APIs) like OpenGL to access and control the programmable GPU. Since GPUs were designed for graphics rendering and not general purpose computing, researchers had to map whatever problems they want to solve into an abstraction of image rendering. GPGPU techniques were more like a clever trick to exploit the power of GPUs. It is a completely different paradigm for researchers unfamiliar with computer graphics, and they face a steep learning curve.

NVIDIA’s Compute Unified Device Architecture, or CUDA, is a parallel computing architecture designed specifically for GPGPU. Unlike with shader programming where the GPGPU is carried out under a graphics “shell,” CUDA provides an environment where programming can be done under a familiar paradigm. CUDA is supported only by NVIDIA’s graphics card from the G80 series onward. This is because the new framework not only requires a software change, but additional hardware is added to the chip where a dedicated silicon area is devoted to general-purpose programming. On the software side, CUDA introduces a small additional set of extensions on top of the C language. The syntax is easy to pick up, but the real challenge is in designing a CGH algorithm that utilizes the full potential of the GPU.

4. Basic Algorithm

A. Parallel Computing

There are two paths for parallelization. The first is point driven, where multiple object points are processed concurrently. The second path is sample driven and leads to parallel computation of multiple hologram plane samples.

The biggest difficulty with the point-driven method is the prevention of race condition. A point may be visible and thus contributing to multiple hologram plane samples. It is highly possible that multiple points being processed simultaneously may write
to the same hologram plane sample at the same time, resulting in unpredictable behavior. There is a way around this by using atomic operations. An atomic operation serializes multiple write-to operations and executes them one after another. So far, all but one atomic operation in CUDA are limited to integers only, so to use floating point values would require some clever programming get-around. Even if floating point atomic operations are supported in the future, performance will suffer because the sequential nature of atomic operations could easily result in a bottleneck.

Point-driven parallelization inherently suits the object-order approach, whereas sample-driven parallelization is a natural fit for the image-order approach and, as we see below, for our hybrid approach as well.

B. Implementation with Compute Unified Device Architecture

The hologram plane (HP) is sampled by a grid of \( M \times N \) nodes. We launch a CUDA kernel where each HP sample is assigned to a separate thread. A kernel is a program written by the programmer that gets executed on many threads simultaneously. The kernel takes an array of point IDs, point coordinates, and complex amplitudes as input, and loops over the entire point array to compute the complex amplitude of HP samples. We process the points in front-to-back order, meaning that we deal with points closest to the hologram plane first and progress our way to the furthest points.

At the core of the kernel lies the visibility test that culs occluded points for each individual HP sample. A point is visible to a HP sample if the straight-line path linking the two is not obstructed by any other points. To determine the visibility of a point to any HP sample, it is not necessary to check every point in the object volume for occlusion, but only those lying closer to the hologram plane than the point under question. The mathematics is straightforward. Essentially we calculate and compare the angles between the point-sample lines and the HP surface normal, for example, \( \theta_1 \) and \( \theta_2 \) in Fig. 2. Only two dimensions are shown in this illustrative example for the sake of brevity, but extension to 3D is trivial. We call these angles the ray angles, since they define the direction of the rays going from HP sample to each point. Since a point may be visible to some HP samples while occluded from others, when we talk about the visibility of a point, we always explicitly or implicitly assume it is in respect to some particular sample.

After the tests, depending on the thread of execution (recall that one thread is assigned to compute for each HP sample), a point may be visible or occluded from view. If it is visible, its radiated field to the HP sample is calculated based on Eq. (1) and added to the sample’s complex amplitude. This process is then repeated on the next point in the input array until all points have been dealt with in this way.

5. Methods of Acceleration

Apart from parallelizing computations, there are several other techniques we can employ to accelerate hologram generation. First of all, there is obviously the number of input points that we would want to minimize. This may not always be possible as it requires optimizing the construction of the 3D point cloud model or resampling of an existing continuous-surface model, and the time penalty we take for doing so may outweigh the benefit. Also obvious and indeed commonly used is the technique of precomputing complex calculations offline and retrieving the results later on-the-fly by a table lookup. Wave propagation is a perfect target for this acceleration scheme. Techniques also exist that target visibility computation, aiming to simplify the visibility test as much as possible and minimize the number of tests. Lastly, we could also try to reduce the number of HP samples, and hence the total amount of computation carried out, by exploiting spatial and temporal coherence and by applying sampling techniques. We discuss the implementation of these techniques one by one in the following.

A. Point Interpolation

From continuous-surface 3D scene geometry to collection of primitives, sampling is at the core of modeling regardless of the form of geometric primitive. A more densely sampled point set better approximates a surface but also increases computational load. Apart from the time issue, there may be occasions where the input point cloud does not have adequate resolution. Under this situation, we would have to rely on data interpolation to fill the void in between sparsely distributed points. Suitably chosen interpolation kernels, centered on each point and overlapping, could make up a good continuous object representation.
Gaussian interpolation is a common interpolation scheme in image processing and in computer graphics. The Gaussian interpolation kernel is a low-pass filter that smooths out the signal it convolves with. The discrete points are thereby blurred and blended together to form a smooth continuous representation. Ideally a 3D interpolation kernel should be used, but that would complicate the wave propagation calculation substantially. At this stage we will limit ourselves to a 2D version,

\[ g(x,y) = \exp \left( -\frac{x^2 + y^2}{2\sigma^2} \right), \]  

where \( \sigma \) is the standard deviation that controls the width of the Gaussian distribution.

Interpolation is not so simple in the case of hologram synthesis, where not only the amplitude but also the phase of the point must be considered. The interpolation kernel in effect turns a discrete point into a finite surface area. For computational reasons, the continuous Gaussian has to be sampled. The area covered by the Gaussian kernel is populated by these samples or points that are interpolated from the mother point. Only the amplitude of the samples is interpolated. To model a diffusive surface, each individual interpolated point is given a random phase \( \rho(x,y) \), and the interpolation kernel becomes

\[ g'(x,y) = \exp \left[ -\frac{x^2 + y^2}{2\sigma^2} \right] \rho(x,y). \]  

A Gaussian function extends to infinity, but for our purpose we define a radius \( R \) and restrict our interpolation kernel to this limit. We want the radius to be such that the edge of the kernel is just touching the adjacent point. The kernel radius is directly proportional to the \( \sigma \) parameter. Figure 3 demonstrates how the choice of the parameter \( \sigma \) affects the resulting point interpolation. As \( \sigma \) increases, so does the radius of the interpolation kernel, and neighboring points would blend together over a longer separation distance.

B. Precomputed Lookup Table

This is perhaps the most popular acceleration technique for computing the CGH in serial fashion [1,22,23], and it still works very effectively in data-parallel computing. In Section 2 we have already presented the basis for implementing this technique, but to take the interpolation kernel into account, we modify the diffraction formula to

\[ s(x',y') = A \exp(j\varphi)b(x',y') \otimes \delta(x-x_o,y-y_o), \]  

where

\[ b(x',y') = F^{-1}\left\{ F\{ g(x,y) \} \times \exp\left( -j2\pi z_d \sqrt{1 - (\lambda \mu)^2 - (\lambda \nu)^2} \right) \right\}. \]  

Equation (7) gives the field of a point (and its interpolation kernel) with unity amplitude and zero initial phase after propagating \( z_d \) meters in free space. We call \( b(x',y') \) the base distribution. The 3D object volume is sliced into \( L \) parallel planes perpendicular to the \( z \) axis, cutting the axis at discrete location \( z_1, z_2 \ldots z_L \). Each plane has a base distribution of its own, calculated according to Eq. (7) using the value of the \( z \) axis at where it is cut. These distributions are precomputed and compiled into a lookup table (LUT), then copied into the GPU's global memory. Since a base distribution is symmetrical about the origin, storing just one quarter of the complete distribution in the LUT is sufficient.

Note that the random phase vector assigned to the interpolation kernel in Eq. (5) is identical for all base distributions. It ensures intrapoint phase incoherence, that is, phase incoherence among interpolated points under the same interpolation kernel and their mother point. After being multiplied by the mother point’s phase factor (which is also random), the combined randomness ensures interpoint phase incoherence as well, that is, incoherence among points of different interpolation groups.

A schematic of how to use the LUT is presented in Fig. 4 with a simplified 1D example. To find the complex amplitude an arbitrary object point \( p \) bestows on a HP sample \( s \) using the LUT; first the most appropriate base distribution in the LUT is selected from the point’s \( z \) coordinate based on nearest neighbor approximation. Then the horizontal sample-to-point distance is used as an index to retrieve a particular element from the selected base distribution (again with nearest neighbor approximation). After being scaled by the point’s amplitude and phase factor \( A \exp(j\varphi) \), the returned value is the complex amplitude we are after.

The LUT is very much application dependent. The one we built for the object models shown in Section 7 contains 350 base distributions with a \( z \)-distance increment of 20 \( \mu \)m. A base distribution has up to 640 \( \times \) 512 elements, and each element corresponds to a sampling point in the hologram plane spaced 13.62 \( \mu \)m apart. In other words, the object volume is sampled by a grid with a sampling pitch of 20 \( \mu \)m \( \times \) 13.62 \( \mu \)m \( \times \) 13.62 \( \mu \)m. Any object point from
the input point cloud that lies between grid nodes will be effectively “snapped” to the nearest node during LUT operation due to nearest neighbor approximations mentioned earlier.

C. Reducing the Number of Visibility Tests

We want to minimize the number of points involved in the visibility test. The visibility of a point is determined by checking it against a list of potential occluders. A potential occluder is any point that is known to be visible. A point that is found to be invisible to a particular HP sample is nonexistent as far as that sample is concerned, and we shall cull it from the list of potential occluders so that it will not appear in subsequent visibility tests for later points. Conversely, if the point is found to be visible, it is added to the list and will be used in the visibility test for the next point down the input point array. This requires the algorithm to maintain a list of all visible points for every HP sample. The lists could occupy a large amount of memory space, and this may be a problem.

D. Visibility Approximations

Determining point visibility is slightly more complicated with the addition of the interpolation kernel. Now instead of just being a point, a point and its interpolation kernel occupies a circular region and blocks out an extended volume behind it. The first approximation we make is to disallow partial occlusion. To a HP sample, a point and its interpolation kernel will either be entirely occluded or entirely visible. There are several choices for setting the criteria for visibility, as illustrated in Fig. 5. At the most conservative level, the entire interpolation kernel of a point has to be inside the shaded region for the point to be considered invisible and culled. A less conservative approximation deems a point invisible only if the point itself is within the shaded region. The most aggressive strategy labels the point as invisible if any part of its interpolation kernel overlaps with the shaded region. Aggressive culling may result in holes, whereas conservative culling, on the other hand, causes color bleeding. Aggressively culling the points will help to increase the speed of computation since fewer points will be processed to compute their contribution to the field in the hologram plane. Something midway between these two extremes offers the best accuracy, so we adopt the less conservative criteria for occlusion culling.

Previously in Subsection 4.B, occlusion between two points \( p_1 \) and \( p_2 \) in relation to HP sample \( s \) is determined simply by checking whether their ray angles are equal. This method does not lend itself

---

**Fig. 4.** (Color online) Schematics illustrating (a) construction of LUT and (b) complex amplitude retrieval.

**Fig. 5.** The illustration on the left shows the interpolation kernel centered on point \( p \) with radius \( R \) and four boundary points marked out. In the middle is an illustration of the occlusion culling strategy, from (a)–(c), showing conservative to aggressive culling. (d) Point \( p_2 \) is invisible from \( s \) if \( p_2' \) is within a radius \( R \) distance of \( p_1 \).
to easy incorporation of the above occlusion criteria. Alternatively, we look for the point where ray \( sp_2 \) intersects the plane containing \( p_1 \) and its interpolation kernel, denoted \( p'_2 \) in Fig. 5(d). According to the occlusion criteria above, point \( p_2 \) is occluded from \( s \) if and only if the distance from \( p'_2 \) to \( p_1 \) is smaller than the kernel radius \( R \). The \((x'_2, y'_2)\) coordinates of \( p'_2 \) are simply

\[
x'_2 = \frac{x_1(x_2 - x_s)}{x_2} + x_s, \quad y'_2 = \frac{y_1(y_2 - y_s)}{x_2} + y_s,
\]

where the subscripts 1, 2, and \( s \) refer to points \( p_1 \) and \( p_2 \), and sample \( s \), respectively.

We may further reduce the amount of computation by dividing HP samples into squares of four, which we call the common visibility group (CVG). HP samples within a CVG share the same set of visible points; the visibility of points is determined with respect to the center of the CVG and applies to all members of the group. The complex amplitude of the four samples in a group is computed individually from the common visible point set.

E. Nonuniform Sampling

So far the complex field in the hologram plane has been evaluated at uniformly distributed sampling points. The idea of this scheme is to lower the overall computational load by reducing the number of samples that must be evaluated. A good sampling scheme allows one to generate CGH using only a subset of the original samples, yet its effect on the reconstructed image quality is minimal. For this, a better sampling scheme than uniform sampling is needed.

Nonuniform sampling has been employed in image and speech analysis as an effective sampling strategy that offers a better trade-off between reduction in sample count and accuracy of the reconstructed signal than does uniform sampling. With the same number of samples, the spectral amplitude of a randomly sampled signal is contaminated with low-power white noise [24], in contrast to the case of a uniformly sampled signal corrupted by irreversible aliasing noise. This advantage is not obvious in our case, where the random phase assumed by object points has the effect of smoothing out the spectral amplitude, which is discarded and not even used in generating our CGH. Nevertheless, a nonuniform sampling scheme, random sampling, was tested. Empirical results, presented in Section 7, support our hypothesis that it produces CGH reconstructions of visual quality superior to that of the uniform sampling scheme.

Our nonuniform sampling scheme is implemented as follows. First we build a table of coordinates of random sampling points in the hologram plane, precomputed offline and copied into GPU memory. Then we simply launch our CUDA kernel, but this time assigning a thread to each random sampling point rather than to each HP sample of the regular \( M \times N \) grid. This acceleration technique is seemingly incompatible with the common visibility group approximation because of the random nature of the sample distribution, and we do not want the samples to conglomerate in little groups.

F. Band-Limited Impulse Response

The finite computation window in Eq. 5 imposes a limitation on the system’s bandwidth, which in turn restricts the fan-out angle of a point’s radiation field [5]. From this we can derive the area of the hologram plane that could potentially receive radiation from an arbitrary point given the point's coordinates, or vice versa, the frustum of object volume encompassing points that could potentially contribute to any arbitrary HP sample. Carrying out visibility tests for points lying outside the frustum is futile and expensive. Thus, before subjecting a point to any visibility test, the algorithm first checks to ensure that it is positioned within the frustum; if not, the point is culled without wasting further processing time.

6. Summary

Pseudocode for the complete algorithm is listed in Algorithm 1 (Table 1 and Algorithm 2 (Table 2), including all of the acceleration methods with the exception of group visibility approximation and nonuniform sampling. Sorting of the object points from front-to-back is achieved through the radix sort implementation provided by the CUDA Data Parallel Primitives library [25], based on [26]. Similarly, a discrete Fourier transform on a GPU is provided by the CUDA Fast Fourier Transform library [27].

7. Results and Discussion

This work is performed on an NVIDIA GeForce 9800 GX2 graphics card. Optical reconstruction was carried out on a reflective, binary spatial-light modulator (SLM) with 1280 by 1024 pixels and a 13.62 \( \mu \)m pixel pitch. The reconstruction was made with a collimated red laser and an optical setup shown schematically in Fig. 6. More details on the optical setup can be found in [28].

Figure 7 shows optical reconstructions of a knight chess piece with 2386 points based on three different

<table>
<thead>
<tr>
<th>Table 1. Algorithm 1. Skeleton of CPU Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Load precomputed LUT and copy to GPU global memory</td>
</tr>
<tr>
<td>2. Load input point data, calculate lighting, and copy data to GPU global memory</td>
</tr>
<tr>
<td>3. Launch CUDA kernel to sort input points according to their z coordinates in descending order</td>
</tr>
<tr>
<td>4. Launch CUDA kernel for ( M \times N ) number of threads to compute the complex field in hologram plane</td>
</tr>
<tr>
<td>5. Launch CUDA kernel to Fourier transform the results of step 4 and quantize to create CGH</td>
</tr>
<tr>
<td>6. Display CGH on SLM</td>
</tr>
</tbody>
</table>

20 December 2009 / Vol. 48, No. 36 / APPLIED OPTICS 6847
σ values and Gaussian kernel radii. We have already seen the effect of point interpolation on regular point arrays in Fig. 3. The optimum σ value depends on the model and its point distribution. The knight in Fig. 7(a) with σ = 1 and a kernel radius of 5 sample pitch is patchy, showing large empty regions, especially in its neck and base where points are further apart. Increasing σ and kernel radius produced more continuous reconstructions in Figs. 7(b) and 7(c), even though the input point cloud was exactly the same as before. The reconstruction of a model without lighting is given in Fig. 7(d) as a comparison to show the importance of lighting.

In Fig. 8 a bishop chess piece is shown cut through by a slanted opaque rectangle. Both reconstructions of CGHs generated with and without a common group visibility approximation provide accurate occlusion. Looking closely, minor degradation in image quality could be detected and some finer details were lost in the approximated CGH, but we would consider the loss to be minimal and barely noticeable by the human eye.

Reconstructions of CGHs created with different sampling schemes are shown in Fig. 9. Figure 9(a) shows the original unsampled CGH, and for Figs. 9(b) and 9(c), 50% of the hologram plane samples have been removed by nonuniform and regular sampling, respectively. As expected, the noise level in Fig. 9(b) is significantly higher than in 9(a), and while it is not quite homogeneous throughout the image, the energy of the noise is spread over a fairly large extent. There is no apparent structure to the noise, and the target image is clearly discernible. The same is not true of the regular sampling scheme; noise is concentrated in bounded regions overlapping the target image. This type of noise, known as aliasing, is much more visually disturbing as is evident in Fig. 9(e).

---

Table 2. Algorithm 2. Skeleton of the CUDA Kernel

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Initialize potential occluder list, $&lt;\text{Occl}&gt;$</td>
</tr>
<tr>
<td>2.</td>
<td>for all points p do</td>
</tr>
<tr>
<td>3.</td>
<td>Compute the distance between point $p$ and HP sample $s$</td>
</tr>
<tr>
<td>4.</td>
<td>Compute $p'$s ray angles</td>
</tr>
<tr>
<td>5.</td>
<td>if $p$'s ray-angles ≤ frustum's bounding angles do</td>
</tr>
<tr>
<td>6.</td>
<td>for all points $p_\text{occl}$ in $&lt;\text{Occl}&gt;$ do</td>
</tr>
<tr>
<td>7.</td>
<td>Retrieve $p_\text{occl}$ coordinates from $&lt;\text{Occl}&gt;$</td>
</tr>
<tr>
<td>8.</td>
<td>Find the point $p_\text{occl}$ where ray $s_2$ intersects the $z = z_p$ plane</td>
</tr>
<tr>
<td>9.</td>
<td>if $p_\text{occl}$ to $p$ distance ≤ interpolation kernel's radius $R$ do</td>
</tr>
<tr>
<td>10.</td>
<td>Flag $p$ as being invisible</td>
</tr>
<tr>
<td>11.</td>
<td>Terminate for loop</td>
</tr>
<tr>
<td>12.</td>
<td>end if</td>
</tr>
<tr>
<td>13.</td>
<td>end for</td>
</tr>
<tr>
<td>14.</td>
<td>end if</td>
</tr>
<tr>
<td>15.</td>
<td>if $p$ is not flagged do</td>
</tr>
<tr>
<td>16.</td>
<td>Compute LUT indices</td>
</tr>
<tr>
<td>17.</td>
<td>Retrieve appropriate base distribution element $b$ from LUT</td>
</tr>
<tr>
<td>18.</td>
<td>Complex amplitude at $s$ due to $p = A \exp(j\phi) \times b$</td>
</tr>
<tr>
<td>19.</td>
<td>Add $p$ to potential occluder list $&lt;\text{Occl}&gt;$</td>
</tr>
<tr>
<td>20.</td>
<td>end if</td>
</tr>
<tr>
<td>21.</td>
<td>end for</td>
</tr>
</tbody>
</table>

The kernel is executed on multiple threads concurrently, where each thread is responsible for computing the complex amplitude of one hologram plane sample.

---

Fig. 7. (Color online) Knight chess piece with varying σ value and Gaussian interpolation kernel radius: (a) $\sigma = 1$, kernel radius = 5 sample pitch; (b) $\sigma = 5$, kernel radius = 15 sample pitch; (c) $\sigma = 10$, kernel radius = 35 sample pitch. (d) Same kernel parameters as (c) but without lighting.
Overall, there is a general “blurriness” in all of the images displayed. Obviously the interpolation scheme blurs the image, but it is only part of the answer; apodization of the laser beam and the SLM also contribute to the blurriness. The laser beam has an intensity profile of a Gaussian, while the finite SLM’s extent can be expressed mathematically as a rect function; the shape of the image pixel after Fourier reconstruction is thus a combination of Gaussian and sinc, rather than a well defined point.

Table 3 summarizes the computation times for a variety of models and acceleration techniques. The computation time increases linearly with both the number of points and the number of CUDA threads. With each thread handling one individual hologram plane sample, both the visibility approximation and the nonuniform sampling technique gain a speedup by reducing the number of samples that must be calculated, even though the common visibility group approximation still computes the diffracted light field on each hologram plane sample individually. This proves that the computational complexity is dominated by the visibility test. Admittedly, the scene construction was rudimentary, and we should expect a longer computation time for more complex scenes with realistic lighting and textures applied. Nevertheless, the results suggest that this algorithm is a competitive alternative to the ray-casting approach. Also included in the table for comparison are the computation times of a sequential version of the CGH algorithm (with no CVG or nonuniform sampling approximation) running on a 1.8 GHz AMD Athlon 64 CPU. The parallel GPU algorithm is some 14 to 15 times faster than its CPU counterpart.

8. Conclusions
We have described and demonstrated a method of rapidly generating full-parallax CGHs. Our ultimate goal is to be able to synthesize holograms using a physically accurate description of light propagation in real time. For this reason, we developed an algorithm that exploits the data-parallel computing capability of GPUs using NVIDIA’s CUDA. Our proof-of-concept system supports geometric occlusion as well as a basic lighting model of perfect diffuse reflection. In order to make up a continuous object representation from discrete points, we applied a Gaussian interpolation kernel to each point. The characteristics of the interpolation kernel have a major influence on the appearance of the reconstructed image; depending on the point distribution and density of the point cloud model, the interpolation kernel should be optimized individually for each model. We used a precomputed LUT to replace expensive diffraction computation on-the-fly, and incorporated a point culling strategy based on potential occluder

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>No. of Threads</th>
<th>Bishop Chess Piece (7592 Points)</th>
<th>Guitar (4694 Points)</th>
<th>Knight Chess Piece (2386 Points)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>655360</td>
<td>20.1</td>
<td>14</td>
<td>7.12</td>
</tr>
<tr>
<td>Nonuniform Sampling</td>
<td>327680</td>
<td>10.3</td>
<td>7.07</td>
<td>3.58</td>
</tr>
<tr>
<td>CVG Approximation</td>
<td>163840</td>
<td>5.07</td>
<td>3.54</td>
<td>1.78</td>
</tr>
<tr>
<td>CPU (sequential version of “Original”)</td>
<td>-</td>
<td>300.55</td>
<td>208.22</td>
<td>104.88</td>
</tr>
</tbody>
</table>

Fig. 8. (Color online) Bishop chess piece with a slanted opaque sheet cutting through it, showing clearly the effect of occlusion: (a) no approximation and (b) group visibility approximation.

Fig. 9. (Color online) Optical reconstruction of CGH generated with different sampling schemes: (a) no sampling, (b) nonuniform sampling, and (c) uniform sampling.
lists and the bandwidth limitation of the system to avoid unnecessary computation.

We could accelerate the hologram generation process further either by assuming that a group of neighboring hologram plane samples share the same set of visible points or by computing only a subset of the entire samples chosen by nonuniform sampling. Both acceleration techniques effectively reduce the number of point visibility tests that dominate the computational complexity of the process and achieve a linear speedup. Optical reconstructions show little to acceptable degradation in image quality for these two techniques. We conclude that the proposed algorithm compares advantageously to other fast hologram synthesis methods.

For future work we would like to extend the interpolation kernel to 3D. A quantitative metric for evaluating reconstructed image quality needs to be developed, as well as a more rigorous study in nonuniform or random sampling with application in Fourier hologram generation. It may also be worthwhile to investigate ways that could combine a nonuniform sampling scheme with visibility group approximation to perhaps reduce the computational load further or to improve the reconstruction quality.

The models are courtesy of the Princeton Shape Benchmark [29]. We thank Neils Hübner for the program that loads CGHs onto the SLM, and Jon Freeman and Wesley Hsu for their help in photography. We are also grateful to NVIDIA for sponsoring the graphics card.

References


