# Interactive light source position estimation for augmented reality with an RGB-D camera

**Citation for published version:**
Boom, BJ, Orts-Escolano, S, Ning, XX, McDonagh, S, Sandilands, P & Fisher, RB 2017, 'Interactive light source position estimation for augmented reality with an RGB-D camera', *Computer Animation and Virtual Worlds*, vol. 28, no. 1, e1686. https://doi.org/10.1002/cav.1686

**Digital Object Identifier (DOI):**
10.1002/cav.1686

**Link:**
Link to publication record in Edinburgh Research Explorer

**Document Version:**
Peer reviewed version

**Published In:**
Computer Animation and Virtual Worlds

# Interactive light source position estimation for augmented reality with an RGB-D camera

Bastiaan J. Boom[a,*], Sergio Orts-Escolano[b], Xin X. Ning[a],

Steven McDonagh[a], Peter Sandilands[a], Robert B. Fisher[a]

[a]School of Informatics, University of Edinburgh,

[b]Dpt. Computer Technology and Computation, University of Alicante,

[*]Email: bas.boom12@gmail.com, bboom@inf.ed.ac.uk

### Abstract

The first hybrid CPU-GPU based method for estimating a point light source position in a scene recorded by an RGB-D camera is presented. The image and depth information from the Kinect is enough to estimate a light position in a scene, which allows for the rendering of synthetic objects into a scene that appear realistic enough for augmented reality purposes. This method does not require a light probe or other physical device. To make this method suitable for augmented reality, we developed a hybrid implementation that performs light estimation in under one second. This is sufficient for most augmented reality scenarios because both the position of the light source and the position of the Kinect are typically fixed. The method is able to estimate the angle of the light source with an average error of 20 degrees. By rendering synthetic objects into the recorded scene, we illustrate that this accuracy is good enough for the rendered objects to look realistic.

**Keywords:** Light Source Estimation, Augmented Reality, GPU implementation, RGB-D camera

# Introduction

The appearance of objects in a scene depends on their illumination. In augmented reality, this illumination is often not taken into account, which makes any rendered synthetic objects look unrealistic. A Kinect sensor can obtain a depth map of the scene, enabling more realistic interactions between real objects and augmented objects [1, 2]. The goal of our work is to estimate the illuminant position in the scene based on the intensity image and depth map provided by the Kinect sensor for the purpose of improving the augmented reality experience (see Figure 1). In this paper, an estimate of the illuminant position is computed based on the Kinect's depth and intensity images without any human annotations of light sources or known objects in the scene. This work uses the Kinect, however, the techniques are not limited to this sensor and any RGB-D camera which provides a registered multispectral image and depth map allows the estimation of the illuminant in the scene.

This paper is focussed towards the application of augmented reality, although the estimation of the illuminant is applicable in multiple domains (like surface improvement, scene understanding, illumination invariant scenes, etc). We have developed a methodology to estimate the location of a single point light source based on the image and the depth map provided by the Kinect, first described in [3]. In this paper, we focus on making it applicable for augmented reality, by building a hybrid CPU-GPU (i.e. "hybrid" in the remainder of the paper) implementation for some of the methods using the Point Cloud Library. We show that a light source estimation can be performed in 1 second, which, in combination with other methods (like Simultaneous Localization And Mapping), can lead to a better Augmented Reality experience.

The main contributions of this paper are:

**GPU implementation of methodology:** We show that it is possible to run the method introduced in [3] on a GPU, processing the incoming Kinect data with an average speed of more than 1 frame per second. This interactive implementation has potential benefits to graphics applications and game designers.

**Creation of dataset:** A public dataset is created to measure and verify the performance of our methodology. In this dataset, we measure the light position and angle relative to the Kinect scenes. The dataset contains multiple scenes recorded by the Kinect, where we have known positions of the light sources, the scene surface points and registered colour data. This dataset will be made publicly available after publication[1].

**Extended experiments:** New experiments are added to investigate the influence of the hybrid implementation on the results. The hybrid implementation requires different algorithmic choices which affect both the accuracy and speed. The two key properties that allow the presented algorithm to work are: 1) Most surfaces can be roughly approximated as Lambertian, and so the surface normals estimated from range data allow an approximate synthesis of the appearance given a light source position. In this case, the surface appearance provides partial constraints on the position of the light source. 2) Segmented surface patches based on colour and distance tend to have the same albedo.

[1] (http://www.dtic.ua.es/jgpu12/lightEstimation/, password: lightEstimationEdi)

This leads to an algorithm that, given an intensity image with associated depth and surface normals at every point, renders images from different hypothesised light source positions and selects the position whose synthesised image best matches the original image. The algorithm can be summarised as follows:

1. Given a candidate light source position, estimate the albedo at every image pixel (using the surface normal and light source position)

2. Given segmented image regions, combine the individual albedo estimates to obtain an estimated albedo for the entire surface segment

3. Using the estimated segment albedos, point surface normals and light source position, synthesise an image of the scene

4. Compute the error between the synthesised and captured real images

5. Optimize the light source position to minimize the error

## Related work

In this research, we focussed on improving a part of the augmented reality experience i.e. the illumination of rendered synthetic objects in a scene. In this case, the research question is how to obtain an estimate of the light source position which can be used by rendering software. We assume that other steps necessary for the augmented reality experience, like positional tracking (e.g. Simultaneous Localization And Mapping or Parallel Tracking and Mapping) and rendering software are available. In our literature study and the remainder of this paper, we will mainly focus on the estimation of the illuminant. This is an old research subject, where, based on the image information and often scene geometry, information about the illuminant can be estimated. Both the overview paper of [4] and [5] discussed that inverse rendering is a problem with multiple unknowns, which can be the lighting, texture, geometry and BRDF, where [5] uses spherical harmonics to put these problems into a signal processing framework. Most papers assume a known geometry in the scene that allow them to estimate one or multiple unknowns. The approach described in [6, 7, 8, 9] estimates the illuminant by taking advantage of the cast shadows and specular reflections of known geometry in the scene. Extensions to these papers are performed in [10] which only needs boundaries of an object and [11] which assume different reflectance models. Recent work by [12] is inspired by the human visual system to use the object silhouette to estimate multiple light sources from a single image. In [13, 14], a stereo setup is used together with a probe sphere (in their case a white shiny ball) to determine multiple area light sources. The work of [15] determines the illuminant of a scene accurately using a mirror surface probe sphere to determine light maps of the scene from different exposed photographs. These light maps are a more dynamic way of modelling the illumination than assuming some synthetic light source. Augmented reality based on mirror surface probe spheres to estimate the illumination is performed by [16, 17, 18]. In Computational Color Constancy, the main interest is the color of the light source, however this often also requires some estimation of the light direction (good overview papers are [19, 20]). There is work in estimating the illumination in outdoor scenes, often for augmented reality, without probe spheres [21, 22, 23], using properties of sunlight and shadows. Recent work of [24] uses movement of the camera and known properties of sunlight to determine outdoor illumination conditions. Augmented reality in outdoor scenes is studied by [25, 26], where the detection of the light source in the outdoor environment is performed using GPS and compass information. A 3D sensor is used for realistic shadow rendering. Another clue to estimate the illumination in a scene are the shadows which has been used in [27, 28, 29]. In [30], a double camera system is used for augmented reality, where the first camera films the actual scene while a fish-eye camera is used for filming the position of the light on the ceiling. Rendering based on photographs where light sources, i.e. windows and ceiling lights, are annotated by users is performed in [31, 32, 17], which is related to the virtual light design of buildings by optimal placing light sources [33].

Recent work on illuminant estimation by [34, 35], tries to decompose the RGB-D input into albedo and shading fields in order to explain the scene. Other work focusses on using the light estimates for shape from shading to improve the depth maps given by the RGB-D cameras [36]. Our previous work [3] focusses more on using the illuminant estimation in augmented reality, where computation time and compatibility with rendering sofware are more important than a perfect explanation of the underlying scene.

The paper is structured as following: 1) The theory of this method is first describe and 2) we implement a hybrid method to allow interactive use. 3) Experiments in realistic scenes show that we can estimate the light direction accurately to 20 degrees. 4) Finally, examples of synthetic objects, rendered realistically into real images are shown

## Explanation of Light Source Estimation Method

As the work of [5] already shows, inverse rendering is a difficult problem, which requires certain assumptions to cope with all the unknowns. This method presented here assumes diffuse surfaces (Lambertian reflectance model), uniform albedo over contiguous regions, a single light source and no cast shadows. A distinction between cast (caused by the blockage of the light source by the object) and attached shadows (surface patches facing away from the light source) is made. Cast shadows are not modelled because their presence is difficult to predict. For example, there may be objects that are not visible in the camera's field of view, but which cast shadows onto the observed scene. Attached shadows are modelled as will be discussed later. The **first assumption** is that the interaction between light source and object can be modelled by the Lambertian reflectance model. This gives us the following equation:

$$I_o(\mathbf{p}) = \rho(\mathbf{p}) \min(\mathbf{n}(\mathbf{p})\mathbf{s}(\mathbf{p})^T i, 0) \qquad (1)$$

According to the Lambertian reflectance model (Equation 1), the image intensity $I_o$ at pixel $\mathbf{p} = \{x, y\}$ can be calculated given the albedo $\rho$, the surface normal $\mathbf{n}(\mathbf{p})$ of the object at $\mathbf{p}$ and the direction $\mathbf{s}(\mathbf{p})$ and intensity $i$ of the light [2].

$$\mathbf{s}(\mathbf{p}) = \frac{\mathbf{l} - \mathbf{x}(\mathbf{p})}{||\mathbf{l} - \mathbf{x}(\mathbf{p})||} \tag{2}$$

where $\mathbf{l}$ is the light source position and $\mathbf{x}(\mathbf{p})$ is the 3D position of point $\mathbf{p}$ in the depth map. We assume that there is only a single dominant point light source present in the scene. The Kinect sensor gives the image intensities $I$ and a 3D depth map. A small proportion of image intensity values do not have associated depth information (these image intensities are not used in our method). Given the 3D depth map, the surface normals $\mathbf{n}$ are computed using [37]. A schematic representation of our method is given in Figure 2 that shows the 3D depth map and the surface normals. The remaining unknown variables in the Lambertian Reflectance model are the albedo $\rho$ and the direction $\mathbf{s}$ and intensity $i$ of the light, making this equation still underdetermined. The **second assumption** is that the albedo of objects is similar in contiguous regions and that different albedo values on the same object are often easy to distinguish. Our methodology uses this assumption by using a colour-based segmentation method Figure 2 step b shows an example of the segmentation), dividing the set of all image pixels $P$ into contiguous segments (subsets) $P = \{R_1, ..., R_N\}$ that have a similar albedo $\rho$. Because the albedo is nearly constant in a segment, the estimated albedo $\rho$ at each pixel $\mathbf{p} \in R_j$ is set to the average albedo in that segment. This allows us to estimate the albedo of segment $R_j$ (Equation 3) for an arbitary light source $\mathbf{s}_r(\mathbf{p}), i_r$.

[2] For simplicity, we assume $i$ is constant but real scenes are affected by light attenuation of $\frac{1}{d^2}$ where $d$ is the distance. In our experiment the effects of the lambertian equation were more dominant.

$$\rho_{R_j} = \frac{1}{|R_j|} \sum_{\mathbf{p} \in R_j} \frac{I_o(\mathbf{p})}{\mathbf{n}(\mathbf{p})\mathbf{s}_r(\mathbf{p})^T i_r} \tag{3}$$

The main observation this paper uses is that *in the original image, a segment is likely to contain a gradient in the image intensity due to the light source position and the normals of the surface, while the albedo is similar for the entire segment.* By optimising the light source position, a better explanation of this intensity gradient can be found. We combine information from all segments $R_j$ with $j = \{1, ..., N\}$ to obtain the full reconstructed image $I_r$. The light source position is determined by minimizing (Equation 5) the difference between the original intensity image $I_o$ and the reconstructed image $I_r$ (Equation 4).

$$I_r(\mathbf{p}) = \rho_{R_j} \min(\mathbf{n}(\mathbf{p})\mathbf{s}(\mathbf{p})^T i, 0), \mathbf{p} \in R_j \tag{4}$$

$$\{\mathbf{s}(\mathbf{p}), i\} = \arg \min_{\{\mathbf{s}(\mathbf{p}), i\}} \sum_{\mathbf{p} \in P} ||I_o(\mathbf{p}) - I_r(\mathbf{p})|| \tag{5}$$

We search for the light source position that reconstructs an image $I_r$ (bottom-right panel in Figure 2) that is closest to the original image $I_o$ (bottom-left). For these images only intensity pixels with depth information are considered. The next section gives more details on the segmentation methods used to obtain the segments $R_1, ..., R_N$.

## Segmentation

Segmentation (Figure 2, step b) is used to find regions that have approximately the same albedo, where we assume that the same image color implies the same albedo. In principle, any colour-based image segmentation method can be used to obtain the regions we consider to have the same albedo. Our experiments on the CPU are performed using the colour-based segmentation method described in [38], where we remove segments that are smaller than $\lambda = 100$ pixels because they are often noisy and do not contain enough gradient information necessary for the minimization (Figure 2, step i).

## Error Function

The error function $E$ (Figure 2, step j) is minimized (Figure 2, step i) to find the light position. The error function is the $L2$-norm between the original image intensity $I_o$ and the reconstructed image intensity $I_r$ (Equation 4), which can be minimized over parameters $(\mathbf{s}(\mathbf{p}), i)$ of the light source position (Equation **??**) : By using a reconstructed image, it is relatively easy to visually verify how well the observed scene can be explained by this method. To reconstruct the image $I_r$ given the Lambertian model (Equation 4), the normals $\mathbf{n}$ and albedo $\rho_{R_j}$ of the objects are needed together with the light direction $\mathbf{s}(\mathbf{p})$ and light intensity $i$. From the depth map, we are able to compute the normal using [37]. In order to minimize the error function, the light source parameters $(\mathbf{s}(\mathbf{p}), i)$ are the search parameters. The minimization method searches the $(\mathbf{s}(\mathbf{p}), i)$ space to minimize Equation **??**, leaving the albedo as the only unknown. Given an estimate of the parameters, $(\mathbf{s}_r(\mathbf{p}), i_r)$, the albedo $\rho_r$ can be computed for every position $\mathbf{p}$ using the Lambertian equation:

$$\rho_r(\mathbf{p}) = \frac{I_o}{\mathbf{n}(\mathbf{p})\mathbf{s}_r(\mathbf{p})^T i_r} \tag{6}$$

Given the assumption that the albedo $\rho_r$ of segment $R_j$ is the same for all positions in the segment $\mathbf{p} \in R_j$, the albedo of the segment $\rho_{R_j}$ can be estimated by taking the mean or median $\rho_r$ at all positions where the reflectance is larger than zero ($\mathbf{n}(\mathbf{p})\mathbf{s}_r(\mathbf{p})^T i_r > 0$). However, if the normals $\mathbf{n}$ are almost perpendicalur to the light direction $\mathbf{s}$, the albedo estimate becomes very unstable, which makes the median a better estimate for $\rho_{R_j}$. Notice that by estimating the albedo using the light source, the albedo varies inversely with the light intensity $i_r$. Therefore, the light intensity is arbitrarily set to $i_r = 1$.

Given the estimated albedo $\rho_{R_j}$ for each segment, we can synthesise the entire reconstructed image $I_r$. In the case of an attached shadow pixel $\mathbf{p}$ ($\mathbf{n}(\mathbf{p})\mathbf{s}_r(\mathbf{p})^T i_r <= 0$), the synthesised image intensity values are set to zero, while cast shadows are not taken into account. The difference between the reconstructed image $I_r$ and the original image intensity $I_o$ (see Equation 5) is then minimized. The error field $E()$ is visualized by computing the error function in a grid over the scene (Figure 5). The large balls shows that there is a global minimum in a position near the measured light position.

### Search Method

To find the light parameters, several search strategies can be used. In the previous section, we observed that albedo and intensity are related allowing us to set the light intensity ($i_r = 1$). In the case of a point light source, we can thus optimize over the variables ($x, y, z$-position of $\mathbf{l}$). Then, using the position of the light source $\mathbf{l}$, compute the light direction $\mathbf{s}(\mathbf{p})$ for every position $\mathbf{p}$. The light position is found in the case of a CPU usng the downhill simplex method [39]. The viewpoint of the Kinect is used as the initial light source position, which allows an initial estimate of the albedo using Equation 6. The initial stepsize of the downhill simplex method to search for a point light source position $\mathbf{l}$ is 10 centimeters and the minimization will continue until convergence.

## Hybrid implementation

Many works have taken advantage of massively parallel architectures like the GPU for obtaining interactive frame rates when doing point cloud processing. In [1] the main part of the algorithm is implemented on the GPU enabling interactive 3D reconstruction and interaction with the environment using a moving depth camera like the Kinect. The authors in [40] leverage the computing power of the GPU to perform a textured 6 DoF (Degrees Of Freedom) reconstruction in real-time, performing the preprocessing and the main core of the algorithm on the GPU.

The main focus of this section is on the implementation of our light estimation algorithm and the extended GPU pipeline which is critical for enabling the estimation of the current light source position at interactive rates. Interactive frame rates also allow rendering realistic shadows in augmented reality scenes, thereby creating more realistic scenes.

For the hybrid implementation, searching in a large grid of possible light source positions in parallel is easier because there are no dependencies in each iteration unlike with the downhill simplex method. The light position in the scene typically does not change from frame to frame which allows us to refine the grid search (i.e. by initializing based on previous interation's optimum) given the next frame, enabling us to determine the light position accurately within a couple of frames. However, in our experiments only a single frame is used to compare the downhill simplex (CPU) with the grid sampling (GPU).

Not every part of the CPU implementation is easy to perform on the GPU, for instance calculation of the median of the albedo of the segments is an expensive operation on the GPU. We address this by introducting a "robust mean", removing all albedo values above a certain threshold ($\tau = 2.5$), which gave experimentally better estimates than the normal mean and offers comparative accuracy performance to the median.

The current CPU segmentation method can not be run in parallel [41], where [41] mentions also alternative parallel image segmentation techniques. In this paper, a parallel image segmentation is given that also use the depth information from the Kinect sensor which is a hybrid CPU-GPU method. The depth information is not necessarily related to the albedo, but we assume depth discontinuities provide useful additional clues for object boundary segmentation. This segmentation methods will be discussed in the subsection

“Segmentation”.

## Workflow

Most of the steps involved in the method are point-wise and therefore are well suited to be implemented on the GPU. While running this method on the CPU requires computing each point sequentially, the GPU performs the computation at many points in parallel, accelerating the execution time of each step considerably. In this case not only the processing of the point cloud is performed on the GPU, but also the pre-processing of the RGB-D map is performed on the GPU projecting the depth map obtained from the sensor to a point cloud. Figure 3 shows the workflow for the GPU implementation from raw depth and color maps to estimated light source position. Each of these steps is executed in parallel on the GPU using the CUDA language [42] and the PCL (PointCloud Library) [43], which offers various algorithms and data structures for point cloud processing and visualization.

## Preprocessing of Depth and Color Information

The first step to be performed on the raw depth map is an upsampling filtering technique [44] due to the noise present in the depth maps provided by the Kinect sensor. We used the traditional Bilateral Upsampling filter implemented on the GPU [45]. For every frame $t$ obtained from the sensor, we launch as many CUDA threads as there are pixels in the depth map. Each CUDA thread computes in parallel a different pixel $\mathbf{p} = \{x, y\}$. After applying the denoising method each GPU thread projects a specific depth value to a 3D vertex in the camera's coordinate system using the intrinsic calibration parameters of the RGB-D sensor. This allows us to obtain an organized vertex map computed in parallel. Corresponding normal vectors for each vertex are also computed by each GPU thread by performing PCA over a local point neighbourhood in parallel. The normal vector is the outward facing Eigenvector with the smallest Eigenvalue.

The normal estimation is accelerated considerably on the GPU. Moreover, the normal estimation step takes advantage of the organized point cloud provided by the Kinect Sensor to perform a faster nearest neighbour (NN) search. Thanks to the organization of the point cloud a fixed sized window can be used as the search space for NN search and thus considerably accelerate the normal estimation process. Running this algorithm on the CPU would not be able to provide interactive processing rates due to the complexity of the algorithm and the number of points that must be calculated sequentially.

Finally the RGB color map obtained from the Kinect sensor is also processed in parallel, performing color space conversion to the HSV space and then to grayscale. These two representations will be used later in the segmentation and light position estimation modules.

## Segmentation

To achieve interactive performance in the hybrid implementation, the segmentation step has been adapted, so that it can be partially computed in parallel on the GPU. As the initially proposed segmentation algorithm is iterative, for the GPU implementation we propose a hybrid approach capable of computing the segmentation partially on the GPU and afterwards computing the region extraction algorithm on the CPU. We extract patches of points with similar color, curvature and close proximity in Euclidean distance. The depth information previously obtained (and already hosted in the GPU memory) is used to aid the segmentation process.

The proposed algorithm uses the same approach as other region growing based algorithms [46, 47, 48]. The main difference is the parallel computation of distances in different spaces using the pixel's neighbourhood and the use of all available information: Euclidean distances, curvature and colorimetric distances. The similarity function between pixels is given by:

$$belongs_{region} = (||p-q|| \leq T_d) \wedge (|\overline{C_r} - C_q| \leq T_c) \wedge (|n_p \times n_q| \leq \cos \epsilon_{\theta_n}) \qquad (7)$$

where $(||p-q|| \leq T_d)$ is a constraint based on the Euclidean distance between points $p$ and $q$. $T_d$ is obtained in real-time based on the point cloud resolution: the mean distance for each point $p$ in its eight-neighbourhood $k$ is calculatedB. Next, based on the average of these mean distances and the standard deviation, threshold $T_d$ is given by: $T_d = \overline{d_k} + \sigma_d$, where $\overline{d_k}$ is the mean distance and $\sigma_d$ is the standard deviation of all computed distances. The established threshold for the maximum angle between the two normal vectors is $\epsilon_{\theta_n}$. This is calculated in the same way as $T_d$, obtaining an angle threshold using the average of mean angles in an eight-neighbourhood and the standard deviation. The target point will belong to the actual region if it is close in Euclidean space to the actual region and if the angle between normal vectors of both points are lower than an established threshold $\epsilon_{\theta_n}$. Furthermore, the colorimetric distance (in HSV) between the mean color $\overline{C_r}$ of the actual region $R_a$ and the targeted point $q$ is calculated. Colorimetric distance is calculated using the following metric as discussed in [49], where $C_r = (h_r, s_r, v_r)$ and $C_q = (h_q, s_q, v_q)$

$$|\overline{C_r} - C_q| = [(s_r v_r \cos h_r - s_q v_q \cos h_q)^2 + (s_r v_r \sin h_r - s_q v_q \sin h_q)^2 + (v_r - v_q)^2]^{(\frac{1}{2})} \quad (8)$$

The proposed GPU segmentation algorithm takes advantage of the matrix organization in which the 3D points are stored, and the fact that they are already stored in the GPU memory. In this way, it is possible to calculate distances in Euclidean, curvature and colorimetric spaces for each point in the scene in parallel. Point distances are calculated relative to a point's direct neighbours in the vertical, horizontal and diagonal directions, resulting in eight distances for each pixel. The pre-computation of these distances on the GPU is then used in a region growing based segmentation using increasing distances in all spaces, deciding if neighbouring points must be added to the current region. This last step is performed on the CPU and uses all the precomputed distances in order to considerably accelerate this process. Furthermore, with the aim of removing the segments with a certain level of noise, segments with a number of points less than $\lambda = 100$ pixels, previously defined, are deleted.

The resulting implementation of this step yields a running time below 100ms allowing continuation with the next step on the GPU. This efficient version allows significantly lower runtimes compared to the CPU version and therefore the entire system is capable of running at interactive rates.

In this paper we proposed an accelerated segmentation algorithm based on the classical region-growing segmentation approach. The proposed implementation is defined as a hybrid one; the first part of the algorithm (computing pixel distances between each pixel and its neighbours) is executed in parallel on the GPU. This considerably accelerates the runtime. The second part (pixel grouping) cannot be parallelized and therefore, it is computed on the CPU taking advantage of GPU precomputed distances. Table 2 shows how the compute distances step is accelerated by a factor of 7-25, reducing the computational time from 100ms to in the best case 4ms and therefore reducing the runtime of the whole segmentation process. The pixel grouping component is executed on the CPU, while computing cloud resolution is performed by the GPU. By overlapping the use of both processors we optimally make use of available computing resources.

Figure 4 shows an example of the segmentation produced by the proposed hybrid implementation. Results obtained are reasonably good, allowing the detection of regions in the scene with different color and geometric properties. Further results regarding the runtime and speed-up obtained compared with the CPU implementation are provided in the result sections.

## Grid-based error minimization search

The algorithm for estimating the position of the light source in the scene is highly parallel. The different steps that are performed and how they are implemented on the GPU are described below.

### Error estimation

Assuming a position of the point light source in the scene, it is necessary to calculate, for every point on each scene surface, the distance and direction vector from the scene point to this assumed light source position. This step is executed in parallel with one CUDA thread computing the Euclidean distance and the direction vector for each point. Using this data, we also compute the albedo factor for every point of the scene in parallel (Equation 6). Finally, using the extracted regions and the previously calculated albedo, the proposed error function (Equation **??**) is computed.

### Grid-base error minimization

To find the light source position with the minimum error, we perform a grid-based search (see Figure 5) in the scene calculating the error in different positions. The possible light source positions are distributed uniformly using a three-dimensional voxel grid (that forms a cube). The error function is computed in parallel in every voxel of the constructed grid. The GPU implementation of the error function allows us to perform this search in less than 1s for a grid of 125 uniform-size voxels with side length 1 meters.

### Dynamic approach for error minimization over time

The estimated position of the light is refined every iteration $t$, where the position found in $t-1$ is used in $t$ to re-estimate the albedo for the grid-base search error minimization. To improve position accuracy after every iteration the grid and its voxel size are decremented in successive frames by an established factor, $0.75$ in the experiments, until the position converges. We stop iterating (convergence) over this process when the voxel size is smaller than 10-15 centimetres (6 iterations considering an initial grid size of $4 \times 4 \times 4$ metres).

The refinement process consists in the iterative execution of the last step of the proposed method: Grid-based energy minimization search. This step is iteratively executed reducing the search space (grid size) and therefore increasing the accuracy of the estimated light source position. The light source position computed in the previous iteration is used as the centroid for the next iteration (new and reduced search space). Thanks to the accelerated and parallelized CUDA implementation of The Grid-based energy minimization search step we are able to run this step repeated times on the GPU decreasing the search space and increasing results accuracy.

# Experiments

## Experimental Setup

To test the method, experiments are performed with different light sources and different scenes. To measure the accuracy of our method, we recorded several different scenes under different known illumination conditions. Most scenes are recorded with a single light source, which in our case is a 60 or 100 watts light bulb or a spotlight (see Figure 6). The distance between two salient control points in the scene on the ground plane are measured together with the height of the light source above the floor. Based on three hand-picked points in the Kinect data, we reconstruct the ground plane allowing us to determine the Kinect coordinates for the true light source position. These positions are used as Groundtruth in our experiments to evaluate the performance of estimating the light source position and angle. Some of the scenes that have been captured with the Kinect are shown in Figure 7. For most scenes, we made multiple captures under different illumination conditions. This dataset is available at http://www.dtic.ua.es/jgpu12/lightEstimation/ (password: lightEstimationEdi).

## Measurement

To measure the accuracy of our method, the difference in the **angle** between the estimated position and the measured position is calculated. Angle comparison is performed by measuring for every pixel (with related 3D position) in the scene “the angle between the measured light source and the estimated light source". For the rendering of objects, having the angle correct seems to be the most important component. In the scene, we can also measure the **distance** between the estimated and the measured position of the light source. This is however deemed less important than the angle. For instance, the spotlight light source is not exactly a point light source so it is often estimated with a low angle error but at a distance from the scene larger than the measured light source. The **speed (ms)** of the algorithm is the final measure, where we compare the CPU implementation with several hybrid CPU-GPU implementations.

## Accuracy Results

Figure 7 shows some of the recorded scenes together with the estimated previous CPU method, estimated current hybrid method and measured light positions. Our dataset consists of 23 different recordings of 6 different scenes illuminated with different light sources. The two scenes in the top rows in Figure 7 show the potential of this method for estimating the light source position and angle. Determining the distance accurately seems to be difficult, while the angle is often correct which is more important for rendering objects realistically in the scene. The scene at the bottom of Figure 7 has a large difference in angle and distance between the estimated and measured light position. The main reason for the large difference seems to be the segmentation, which is probably the main cause of errors. The segmentation sometimes fails to segment multiple objects with different albedos incorrectly, and therefore assumes these objects have the same albedo, which causes errors in the reconstructed image. The difference between the previous CPU and current hybrid method often comes from diffences in the segmentation methods.

In the preceding theory section, different alternatives were proposed for computing the albedo (using mean, robust mean or median). These alternatives have implications on the speed of the method. In this section, we compare the different alternatives for the computation of the albedo, search method and segmentation method. Our default method on the CPU for the computation of the albedo is the median, for search the default is the downhill simplex method and for segmentation it is the Graph-Based Image Segmentation where the only variable parameter is $K$ = 200, where a larger value of $K$ creates larger image segments. For the hybrid method, we use the robust mean, grid search and region growing segmentation.

**Computation of the Albedo**: We suggest using the mean, robust mean or median in order to compute the albedo of a segment. In Figure 8, the accuracy in distance and difference in angles is shown averaged over all scenes. The distance computed using the median has a large standard devation. It is especially difficult to perform accurately for scenes with a large distance to the light source or scenes with spotlights. Figure 8 shows that the angle is much more accurate for the median than the mean, which is more important for this application. However, the median is a more expensive operation on the GPU, because it applies a sort over the entire set of values. Because of this, we experimented with a robust mean, removing all albedo values above a certain threshold ($\tau = 2.5$), which gives better estimates than the normal mean and offers comparative accuracy performance to the median (Figure 8) while using a computationally cheaper mean operation.

**Search Methods**: Two search methods are used for finding the light position, namely a grid search for the hybrid method and the downhill simplex method for the CPU method. Although the grid size can be set in our software, for the experiments a grid from the Kinect sensor of $-2$ metres to $2$ metres is used. A comparison between the downhill simplex method and the grid search is given in Figure 9. The downhill simplex method performs slightly worse than the grid search, however the grid search seems to have a larger variance, which seems to be mainly due to mistakes in the segmentation allowing multiple illumination explanations to be likely for a scene.

**Segmentation Methods**: The segmentation methods are different between the previous CPU and hybrid method. On the previous CPU method, the Graph-Based Image Segmentation [38] is performed both with only the image and on both image and depth information with two parameter settings ($K$ = 100 and $K$ = 200) where a large value of $K$ creates larger albedo segments. On the hybrid method, a different segmentation method is used on both the image and depth information. A comparison of these methods is given in Figure 9, where the median performance of the different segmentation methods is very similar. The depth information makes the angle slightly better but also has a larger standard devation. The implementation of the segmentation, which is not Graph-Based but uses region hybrid growing gives very similar results to the CPU implementations, which shows that our method does not depend on a single segmentation methodology.

## Performance (ms)

The hybrid version of the proposed method has been tested on a desktop machine with an Intel Core i3 540 3.07Ghz and different CUDA capable devices. The hybrid implementation was first developed on a laptop machine equipped with an Intel Core i5 3210M 2.5 Ghz and a CUDA compatible GPU. Table 1 shows different models that have been used and their main features. We used different models ranging from the integrated GPU on a laptop to a more advanced model, demonstrating that the hybrid implementation can be executed on different GPUs and obtains good runtimes on all systems experimented with.

The performance obtained by the hybrid implementation allow us to executed the proposed method faster than 1 sec. In Table 2 we can see the different steps that have been accelerated using the GPU and their different runtime and the speed-ups achieved for the different graphics boards. The obtained acceleration is relative to a CPU implementation of the proposed method.

The best performance was obtained with the graphics board with the largest number of CUDA cores (GTX480) and the largest memory bandwidth, performing $\sim 6 - 7\times$ faster than the CPU implementation. This allows our light source position estimation technique to be used for demanding realistic rendering applications with interactive rates.

The overall speed-up is not so high compared to some of the individual steps because although the light estimation process itself is parallelized at pixel level, we still have to traverse all voxels in the grid and estimate the light position in each voxel (brute-force). A more complex approach could be developed overlapping the computation in parallel over different voxels (task parallelism). We discarded this approach because in experiments we obtained full occupancy of GPU processors with the current implementation, so this high level of parallelism would need more powerful GPUs.

Another interesting aspect of the results shown in Table 2 is that the hybrid implementation allows us to compute operations that are prohibitively slow on the CPU such as normal estimation using PCA and the depth map noise reduction using bilateral filtering.

# Rendering a Synthetic Object into the Scene

## Technical Details

By estimating the light source position, we are able to realistically render a new object (see Figure 10 and 11) into each RGB image created by the Kinect, such that the new object appears natural in terms of lighting, shading and shadows. To do this, we first create an approximate surface over the coloured depth data using [50]. This surface gives us the geometry of the scene. In order to render the new object, we first use this geometry to design a non-penetrating motion for the object using keyframe animation in Autodesk Maya. Using the light position and properties estimated by our method, we can create a point light element in Maya. This light position is already in the aligned camera space, so no additional calculation needs to be performed on the light location. A virtual camera that matches the Kinect's properties is created, using the specifications provided at [51]. The important values were horizontal field of view and film aspect ratio. These were set to 62.7 degrees and 1.33 respectively. In order to keep the camera model simple, we did not apply any depth-of-field or motion blurring effects. For rendering the artificial images, we used the NVIDIA Mental Ray raytracing system [52]. We rendered a separate shadow pass, where only the shadows on the background surface (that we generated from the depth data) and on the inserted objects were calculated. This results in a 640x480 render where non-transparent colours are the values to remove from the unshadowed image. Self-shadowing is disabled for the background geometry in order to prevent the duplication of shadows in the final image. We then rendered an unshadowed pass using the light positions, with only the new character or object visible. For this reason we call this second pass the character render. Using these two renders we can finally composite the character render, the shadow render and the original RGB image from the Kinect together to create the final image. We used the "replace" blend mode for the character render over the original Kinect image, and the "subtract" blend mode for the shadows on top of these two. Using the subtract mode allows us to use the original colour data from the Kinect, whilst computing the shadow effect for each pixel per frame, giving us realistic shadows that depend only on the quality of the reconstructed geometry.

### Resulting scenes

Although the estimation of the light source will never be entirely accurate, with Figures 10 and 11 we show that a reasonable estimation is often sufficient. The experience of having objects that are rendered whilst taking into account the illumination information in the scene will in most cases be enough. In Figures 10 and 11, other synthetic objects are rendered into the scene to show the potential of this method. This conclusion is further more supported by the paper of [53], which investigates the human perception on estimating the light angle showing that humans can not detect a error in the angle of 20 degrees. In the supplemental materials videos are included showing that the object can interact with the scene. Although this material is developed offline, given the speed of our methodology we should also be able to connect these methods to augmented reality software in the future. Figure 10 shows that the shadows interact with the environment, showing that the shadows take into account the mesh of the background objects. Figure 11 shows that you can replace objects or people, although this was done offline. Using object segmentation this can also be done online.

## Discussion

In this paper, a method for estimating the light source position is described for the application of rendering synthetic objects in a scene. A new method for estimating the light source position based on 3D depth data with a registered colour image (e.g. given by a Kinect sensor) has been developed. We show that this estimation can be performed at speeds on the order of once per second (718 ms) with our hybrid implementation. Often both the Kinect position and light source position do not change much over time, which makes this implementation useful for the application of augmented reality. We verified our method using both OpenGL rendering software and with a dataset of real scenes with measured light position. The experiments show that the angle of the light source can be estimated with an average error of 20 degrees between measured and estimated light source positions. These light source estimates are good enough for rendering synthetic objects into the scene with realistic looking illumination conditions.

Although this work is limited to the estimation of a single point light source, we noticed that this simple assumption often also works in scenes with more difficult illumination conditions like fluorescent tubes, reconstructing an image that explains the scene as well as possible. Although the Kinect is not often used in outdoor environments, estimating directional light sources like the sun should be possible by checking if the distance of the light source to the scene becomes very large and checking if a directional light source in the same direction is able to give a better minimization of the error function. The minimization procedure can also be extended to search for multiple light sources, however this adds complexity to the minimization, making it less attractive for augmented reality. More complex reflectance models which include ambient light (small amounts of light that is scattered about the entire scene) and specularity can be used, however many more parameters are needed since some of these properties are surface dependent. Initial simulations showed minor effects, where ambient light causes the estimated light source position to be estimated as further away from the scene, while specular reflectance had the reverse effect. The light estimation is however often averaged over different kinds of reflective surfaces. Shadow detection and ray tracing of known objects can enhance the light estimation, but is also an expensive operation, which should be performed after we reduced the search space. None the less, the effects of cast shadows are limited as shadow areas are often segmented separately and they are not a very large part of the scene, resulting in having only a small effect in the error function.

Future work can focus on other applications where illumination estimation is important, like improving 3D surface using shape from shading or creating illumination invariant features for scene recognition.

## Acknowledgments

This work is partially supported by the Fish4Knowledge project, which is funded by the European Union 7th Framework Programme [FP7/2007-2013], by the HiPEAC Network of Excellence, by Valencian Government grant BEFPI/2012/056 and by EPSRC (EP/P504902/1, EP/H012338/1)

## References

[1] Shahram Izadi, Richard A. Newcombe, David Kim, Otmar Hilliges, David Molyneaux, Steve Hodges, Pushmeet Kohli, Jamie Shotton, Andrew J. Davison, and Andrew W.

Fitzgibbon. Kinectfusion: real-time dynamic 3d surface reconstruction and interaction. In *SIGGRAPH Talks*, pages 23:1–23:1, 2011.

[2] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew W. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *ISMAR*, pages 127–136, 2011.

[3] Bastiaan J. Boom, Sergio Orts-Escolano, Xin X. Ning, Steven McDonagh, Peter Sandilands, and Robert B. Fisher. Point light source estimation based on scenes recorded by a rgb-d camera. In *British Machine Vision Conference*, 2013.

[4] Gustavo Patow and Xavier Pueyo. A survey of inverse rendering problems. In *Computer graphics forum*, volume 22, pages 663–687. Wiley Online Library, 2003.

[5] Ravi Ramamoorthi and Pat Hanrahan. A signal-processing framework for inverse rendering. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 117–128, New York, NY, USA, 2001. ACM.

[6] Pierre Poulin and Alain Fournier. Lights from highlights and shadows. In *Proceedings of the 1992 Symposium on Interactive 3D Graphics*, I3D '92, pages 31–38, New York, NY, USA, 1992. ACM.

[7] Pierre Poulin, Karim Ratib, and Marco Jacques. Sketching shadows and highlights to position lights. In *Proceedings of the 1997 Conference on Computer Graphics International*, CGI '97, pages 56–, Washington, DC, USA, 1997. IEEE Computer Society.

[8] Yang Wang and Dimitris Samaras. Estimation of multiple directional light sources for synthesis of mixed reality images. In *Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*, PG '02, pages 38–47, Washington, DC, USA, 2002. IEEE Computer Society.

[9] Yang Wang and Dimitris Samaras. Estimation of multiple directional light sources for synthesis of augmented reality images. *Graphical Models (Special Issue on Pacific Graphics)*, 65(4):185–205, July 2003.

[10] Yuanzhen Li, S. Lin, Hanqing Lu, and Heung-Yeung Shum. Multiple-cue illumination estimation in textured scenes. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 1366 –1373 vol.2, oct. 2003.

[11] Kenji Hara, Ko Nishino, and Katsushi Ikeuchi. Light source position and reflectance estimation from a single view without the distant illumination assumption. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(4):493–505, April 2005.

[12] Jorge Lopez-Moreno, Sunil Hadap, Erik Reinhard, and Diego Gutierrez. Compositing images through light source detection. *Computers & Graphics*, 34(6):698–707, 2010.

Graphics for Serious Games Computer Graphics in Spain: a Selection of Papers from {CEIG} 2009 Selected Papers from the {SIGGRAPH} Asia Education Program.

[13] Wei Zhou and Chandra Kambhamettu. Estimation of the size and location of multiple area light sources. In *Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 3 - Volume 03*, ICPR '04, pages 214–217, Washington, DC, USA, 2004. IEEE Computer Society.

[14] Wei Zhou and Chandra Kambhamettu. A unified framework for scene illuminant estimation. *Image Vision Comput.*, 26(3):415–429, March 2008.

[15] Paul Debevec. Rendering synthetic objects into real scenes: bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 189–198, New York, NY, USA, 1998. ACM.

[16] S. Gibson, T.L.J. Howard, and R.J. Hubbold. Image-based photometric reconstruction for mixed reality. In *SIGGRAPH 2001 Sketches and Applications Program*, August 2001.

[17] Kusuma Agusanto, Li Li, Zhu Chuangui, and Ng Wan Sing. Photorealistic rendering for augmented reality using environment illumination. In *Proceedings of the 2nd IEEE/ACM International Symposium on Mixed and Augmented Reality*, ISMAR '03, pages 208–216, Washington, DC, USA, 2003. IEEE Computer Society.

[18] S. Heymann, A. Smolic, K. Müller, and B. Froehlich. Illumination reconstruction from real-time video for interactive augmented reality. In *International Conference on Video and Image Processing*, 2005.

[19] Steven D. Hordley. Scene illuminant estimation: Past, present, and future. *Color Research & Application*, 31(4):303–314, 2006.

[20] A. Gijsenij, T. Gevers, and J. van de Weijer. Computational color constancy: Survey and experiments. *Image Processing, IEEE Transactions on*, 20(9):2475 –2489, sept. 2011.

[21] Yanli Liu, Xueying Qin, Songhua Xu, Eihachiro Nakamae, and Qunsheng Peng. Light source estimation of outdoor scenes for mixed reality. *Vis. Comput.*, 25(5-7):637–646, April 2009.

[22] Yanli Liu, Xueying Qin, Guanyu Xing, and Qunsheng Peng. A new approach to outdoor illumination estimation based on statistical analysis for augmented reality. *Comput. Animat. Virtual Worlds*, 21(34):321–330, May 2010.

[23] Jean-Franois Lalonde, Alexei A. Efros, and Srinivasa G. Narasimhan. Estimating the natural illumination conditions from a single outdoor image. *International Journal of Computer Vision*, 98:123–145, 2012.

[24] Yanli Liu and Xavier Granier. Online tracking of outdoor lighting variations for augmented reality with moving cameras. *IEEE Transactions on Visualization and Computer Graphics*, 18:573–580, 2012.

[25] Claus B. Madsen and Michael Nielsen. Towards probe-less augmented reality - a position paper. In *GRAPP*, pages 255–261, 2008.

[26] Claus B. Madsen and Brajesh Behari Lal. *Probeless Illumination Estimation for Outdoor Augmented Reality*. Augmented Reality, 2010.

[27] I. Sato, Y. Sato, and K. Ikeuchi. Illumination distribution from shadows. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 1, pages –312 Vol. 1, 1999.

[28] Imari Sato, Yoichi Sato, and Katsushi Ikeuchi. Illumination from shadows. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(3):290–300, March 2003.

[29] A. Panagopoulos, T.F.Y. Vicente, and D. Samaras. Illumination estimation from shadow borders. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 798 –805, nov. 2011.

[30] J. Frahm, K. Koeser, D. Grest, and R. Koch. Markerless augmented reality with light source estimation for direct illumination. In *Visual Media Production, 2005. CVMP 2005. The 2nd IEE European Conference on*, pages 211 – 220, 2005.

[31] Céline Loscos, Marie-Claude Frasson, George Drettakis, Bruce Walter, Xavier Granier, and Pierre Poulin. Interactive virtual relighting and remodeling of real scenes. In *Rendering Techniques 99*, pages 329–340. Springer, 1999.

[32] Céline Loscos, George Drettakis, and Luc Robert. Interactive virtual relighting of real scenes. *Visualization and Computer Graphics, IEEE Transactions on*, 6(4):289–305, 2000.

[33] António Cardoso Costa, António Augusto Sousa, and Fernando Nunes Ferreira. Lighting design: A goal based approach using optimisation. In *Proceedings of the 10th Eurographics Conference on Rendering*, EGWR'99, pages 317–328, Aire-la-Ville, Switzerland, Switzerland, 1999. Eurographics Association.

[34] Jonathan T. Barron and Jitendra Malik. Intrinsic scene properties from a single rgb-d image. *CVPR*, 2013.

[35] Qifeng Chen and V. Koltun. A simple model for intrinsic image decomposition with depth cues. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 241–248, Dec 2013.

[36] L.-F. Yu, S.-K. Yeung, Y.-W. Tai, and S. Lin. Shading-based shape refinement of rgb-d images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[37] Toby. P. Breckon and Robert. B. Fisher. Environment authentication through 3d structural analysis. In *Image Analysis and Recognition*, volume 3211 of *Lecture Notes in Computer Science*, pages 680–687. Springer Berlin Heidelberg, 2004.

[38] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *Int. J. Comput. Vision*, 59(2):167–181, September 2004.

[39] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.

[40] Dominik Neumann, Felix Lugauer, Sebastian Bauer, Jakob Wasza, and Joachim Hornegger. Real-time RGB-D mapping and 3-D modeling on the GPU using the random ball cover data structure. In *ICCV Workshops*, pages 1161–1167. IEEE, 2011.

[41] Jan Wassenberg, Wolfgang Middelmann, and Peter Sanders. An efficient parallel algorithm for graph-based image segmentation. In *Proceedings of the 13th International Conference on Computer Analysis of Images and Patterns*, CAIP '09, pages 1003–1010, Berlin, Heidelberg, 2009. Springer-Verlag.

[42] NVIDIA. *NVIDIA CUDA Programming Guide 4.2*. 2008.

[43] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[44] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Proceedings of the Sixth International Conference on Computer Vision*, ICCV '98, pages 839–846, Washington, DC, USA, 1998. IEEE Computer Society.

[45] Derek Chan, Hylke Buisman, Christian Theobalt, and Sebastian Thrun. A Noise-Aware Filter for Real-Time Depth Upsampling. In *Workshop on Multi-camera and Multi-modal Sensor Fusion Algorithms and Applications - M2SFA2 2008*, Marseille, France, 2008.

[46] Yinghui Xiao Qingming Zhan, Yubin Liang. Color-based segmentation of point clouds. In *ISPRS Volume XXXVIII-3/W8*, 2009.

[47] Dirk Holz and Sven Behnke. Fast range image segmentation and smoothing using approximate surface reconstruction and region growing. In *Proceedings of the 12th International Conference on Intelligent Autonomous Systems (IAS)*, Jeju Island, Korea, June 2012.

[48] T. Rabbani, F. A. van den Heuvel, and G. Vosselmann. Segmentation of point clouds using smoothness constraint. In *IEVM06*, 2006.

[49] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer graphics: principles and practice (2nd ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[50] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics Symposium on Geometry Processing*, SGP '06, pages 61–70, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[51] Kurt Konolige and Patrick Mihelich. Technical description of kinect calibration. http://www.ros.org/wiki/kinect_calibration/technical.

[52] NVIDIA. Mental ray: Rendering imagination visible. http://www.mentalimages.com/products/mental-ray.html.

[53] Jorge Lopez-Moreno, Veronica Sundstedt, Francisco Sangorrin, and Diego Gutierrez. Measuring the perception of light inconsistencies. In *Proceedings of the 7th Symposium on Applied Perception in Graphics and Visualization*, APGV '10, pages 25–32, New York, NY, USA, 2010. ACM.

**Bastiaan J. Boom** received his bachelor engineering degree in Computer Science from the Hogeschool van Amsterdam in 2002 and a the Master degree from the Free University Amsterdam in Computer Science in 2005 He has a PhD degree from the University of Twente (2010), where he specialised in the fields of biometrics. He was Research Associated at the University of Edinburgh and is currently working for Cyclomedia, where his research interests are in computer vision and machine learning.

**Sergio Orts-Escolano** received his B.Sc and M.Sc in Computer Science from the University of Alicante (Spain) in 2010 and 2011 respectively. He is currently a researcher with the Department of Computer Technology at the University of Alicante. His research interests include 3D vision, surveillance systems, parallel computing on GPUs and neural networks.

**Xin Xin Ning** received his M.Sc (Hons) degreein Computer Science from the University of Edinburgh in 2012. Currently, he is studying at the Entertainment Technology Center at Carnegie Mellon University.

**Steven McDonagh** received the BSc degree in Computer Science and Artificial Intelligence from The University of Edinburgh in 2008 and a PhD degree in 2015, His interests span a variety of topics in computer vision, image processing and machine learning and is currently working for Disney Research His current work focuses on the analysis and implementation of multi-view registration algorithms, range data processing and geometric modelling.

**Peter Sandilands** studied for the Ph.D. under Dr. Taku Komura at the School of Informatics in the University of Edinburgh, previously receiving his BSc (Hons) in Artificial Intelligence and Computer Science from the same institution. In 2010, he won the ScotlandIS Young Software Engineer of the Year award for his work on visual and auditory systems of the Sony AIBO and in 2012 won Best Student Paper at the Motion in Games conference. His current research focus is on capture and generation of close interactions between actors and objects. He now works in industry for Rockstar North.

**Prof. Robert B. Fisher** received a B.S. with Honors (Mathematics) from California Institute of Technology (1974) and a M.S. (Computer Science) from Stanford University (1978). He received his PhD from University of Edinburgh (1987), investigating computer vision. Since then, Bob has been an academic at Edinburgh University, now in the School of Informatics, where helped found the Institute of Perception, Action and Behaviour.

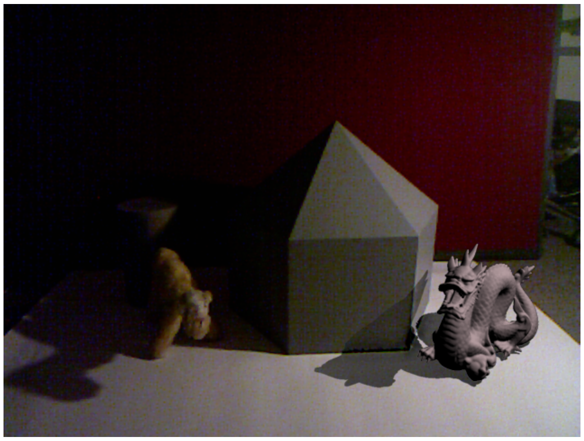

Figure 1: A synthetic object (dragon, right) that is rendered into a real-world scene recorded with the Kinect, where the illumination of the synthetic object is similar to the scene and the rendered shadows take into account the geometry of the scene based on an estimated light source position determined using only the intensity image and depth information.

| **Device Model** | **CUDA cores** | **Global Mem** | **Bandwidth Mem** |
|---:|---:|---:|---:|
| Quadro 2k | 192 | 1 GB | 41.6 GB/s |
| GeForce GTX 480 | 480 | 1.5 GB | 177.4 GB/s |
| GeForce GT630M | 96 | 1 GB | 32 GB/s |

Table 1: CUDA capable devices used in experiments.

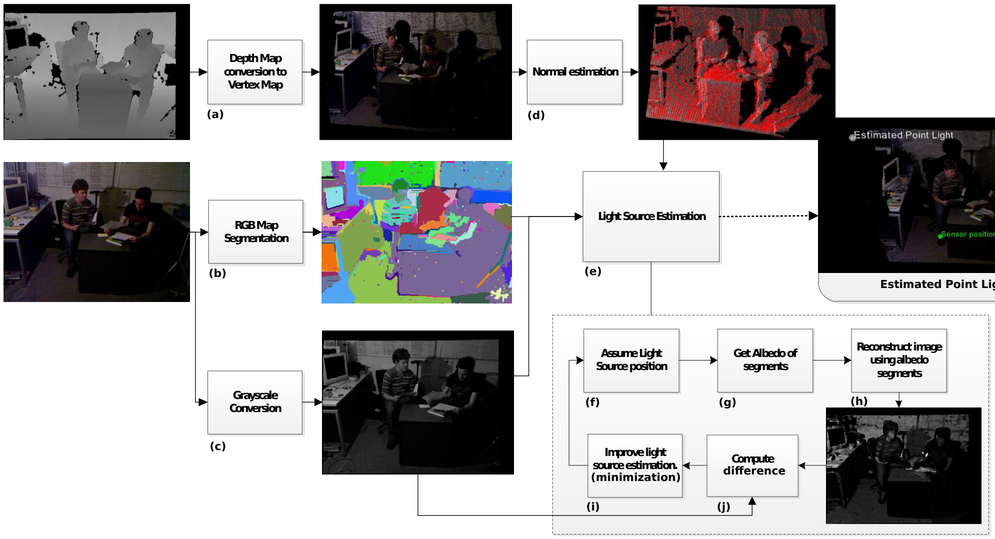

Figure 2: A schematic representation of the method to estimate the light source position, where given the input (depth map and image shown left), we compute the normals (top-right) and segments (middle). Afterwards we search for the light source position that gives the best reconstructed image (bottom-right) by minimizing the distance with the original image (bottom-middle), allowing us to find the light source direction shown in (middle-right). The method in box e is expanded to show the contained submethods (boxes f to j)

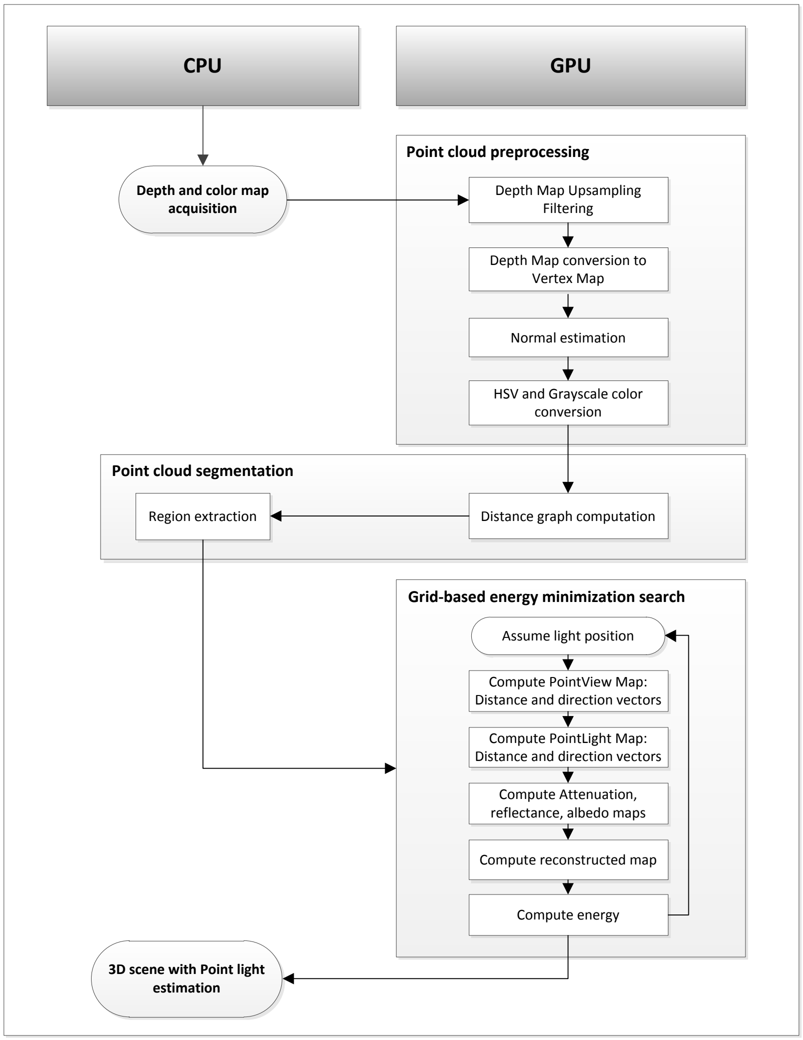
CPU
GPU
Depth and color map acquisition
Point cloud preprocessing
Depth Map Upsampling Filtering
Depth Map conversion to Vertex Map
Normal estimation
HSV and Grayscale color conversion
Point cloud segmentation
Region extraction
Distance graph computation
Grid-based energy minimization search
Assume light position
Compute PointView Map: Distance and direction vectors
Compute PointLight Map: Distance and direction vectors
Compute Attenuation, reflectance, albedo maps
Compute reconstructed map
Compute energy
3D scene with Point light estimation

Figure 3: Extended work flow diagram for computing the Light Source Position Estimation algorithm on the GPU. Note that most of the steps have been moved to the GPU in order to achieve a runtime faster than 1s on an off-the-shelf consumer GPU.

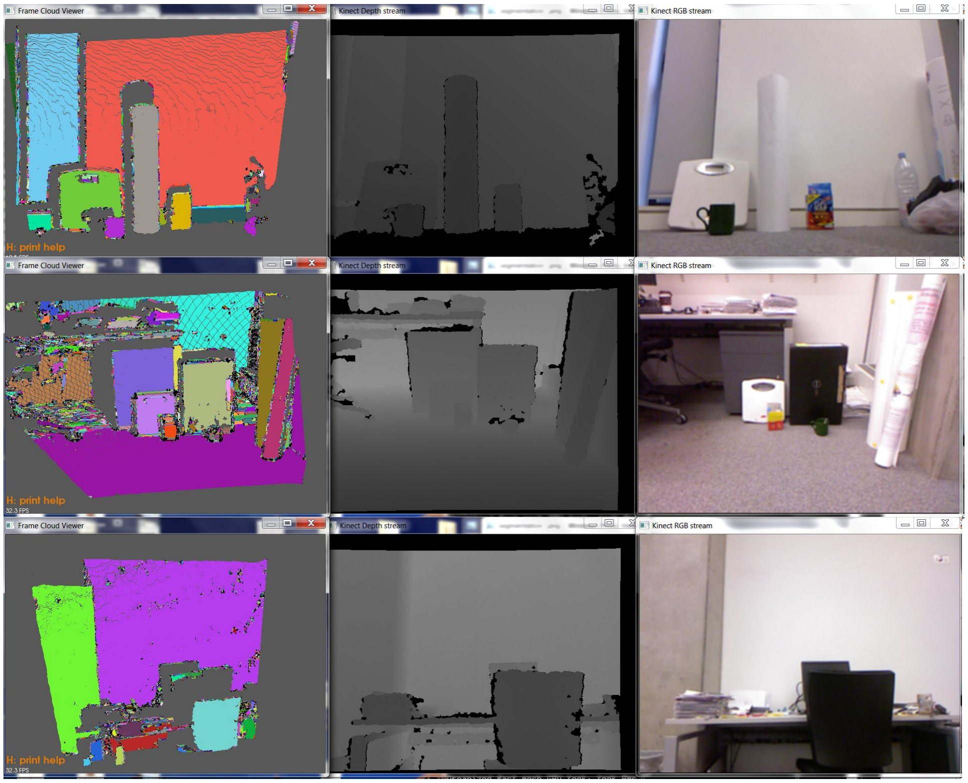

Figure 4: Point cloud segmentation examples. Left column: Segmented point cloud. Center column: Depth map. Right column: Color map smoothed using bilateral filtering. Segmentation results are satisfactory allowing real-time scene region extraction.

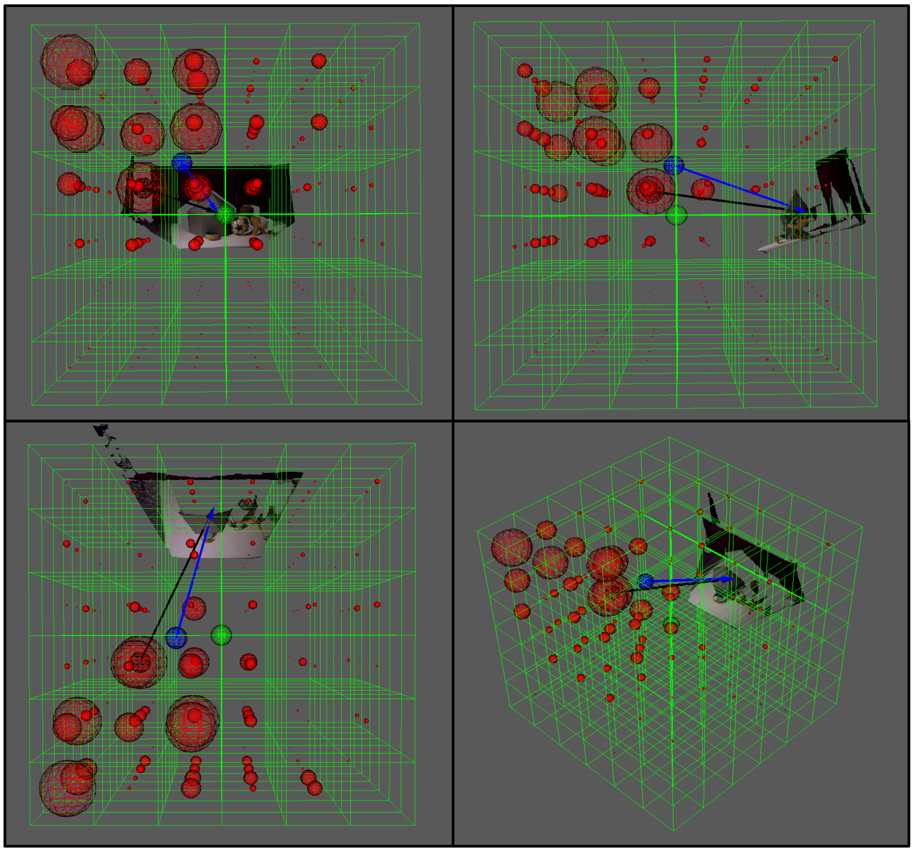

Figure 5: The error function given a point light source is shown, where the scene is shown from different angles. The green sphere is the camera position, while the blue sphere is the measured position of the light source with a blue arrow toward the middle of scene. In this figure, the error is computed in a grid of $-2$ to $2$ meters with steps of $\frac{2}{3}$ meters. The red spheres are the points where the errors are sampled and the sphere radius is inversely proportional to the error for visualization purposes. The largest red sphere shows the minimum error where black arrow is the global minimum direction for this grid, which is very close to the measured light position.

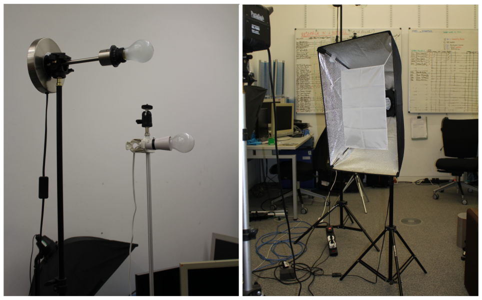

Figure 6: The light sources used to record all the scenes. The left photo shows the 60 and 100 watt light bulbs. The right photo shows the spotlight.

| Step | GT630M | GTX480 | Quadro2k | CPU | GT630M | GT480 | Quadro2k |
|---:|---:|---:|---:|---:|---:|---:|---:|
| Bilateral filtering of depth map | 11 ms | **5 ms** | 8 ms | 1008 ms | 91.63x | **201.6x** | 126x |
| Point cloud projection | 2 ms | **1 ms** | 1 ms | 50 ms | 25x | **50x** | 50x |
| Normal estimation | 9 ms | **1 ms** | 8 ms | 190 ms | 21.11x | **190x** | 23.75x |
| Compute distances graph for segmentation | 13 ms | **4 ms** | 11 ms | 101 ms | 7.76x | **25.25x** | 9.18x |
| Compute cloud resolution | 7 ms | **4 ms** | 6 ms | 330 ms | 47.14x | **82.5x** | 55x |
| Compute error given point light source | 10 ms | **9 ms** | 12 ms | 50 ms | 5x | **5.55x** | 4.16x |
| Grid-based error minimization | 595 ms | **583** ms | 818 ms | 3056 ms | 5.13x | **5.24x** | 3.73x |
| Total execution time | 778 ms | **718 ms** | 958 ms | 4855 ms | 6.24x | **6.86x** | 5.06x |

Table 2: Runtime comparison and speed-up obtained for the proposed method using different graphics boards. The fastest runtime was achieved by the graphics board NVIDIA GTX480 running the algorithm 6.86 times faster than on a conventional CPU.

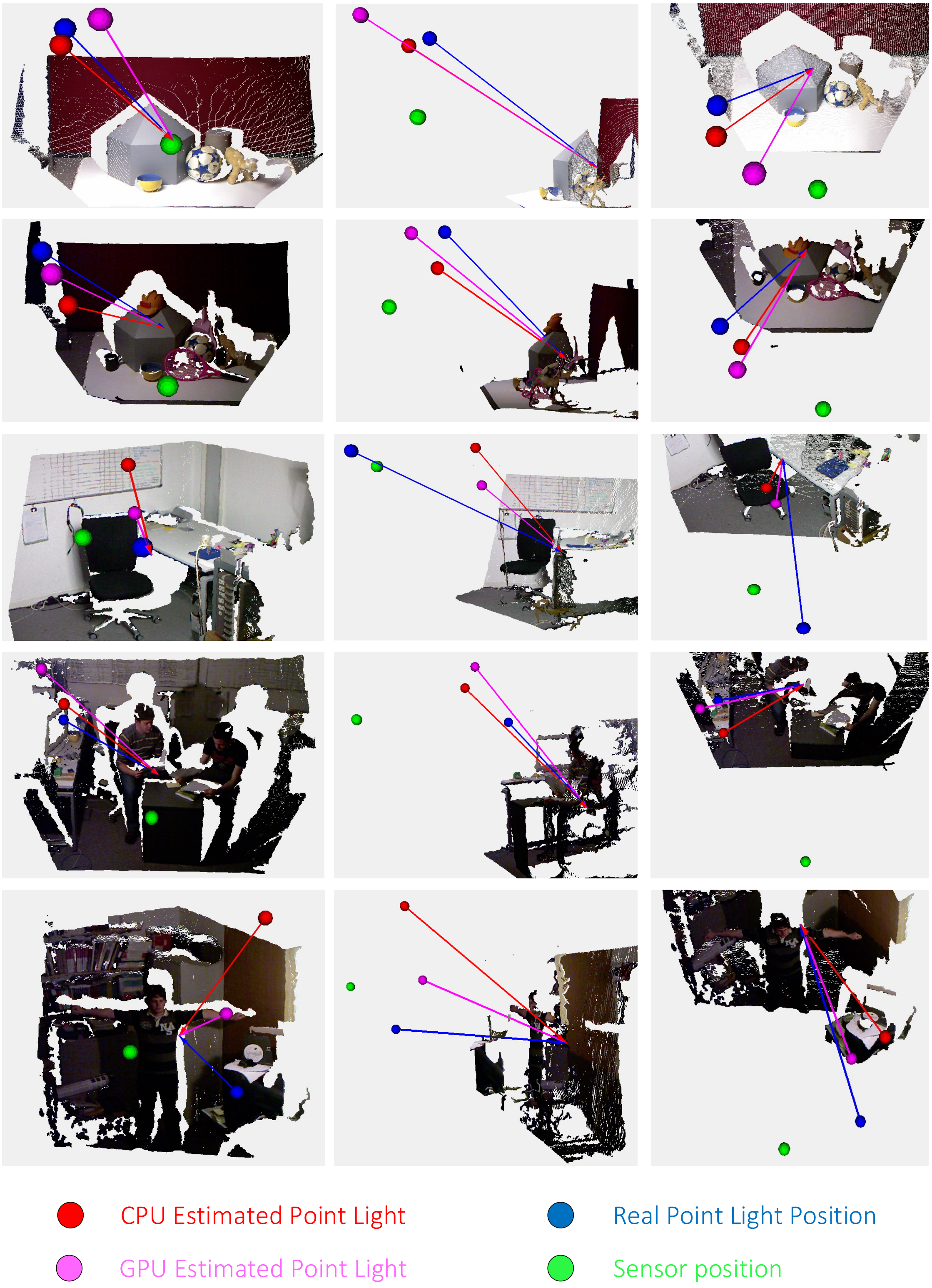
● CPU Estimated Point Light
● Real Point Light Position
● GPU Estimated Point Light
● Sensor position

Figure 7: Four different scenes (one scene per row) of our dataset together with the camera position (green ball) and the CPU estimated (red ball), GPU estimated (purple ball) and

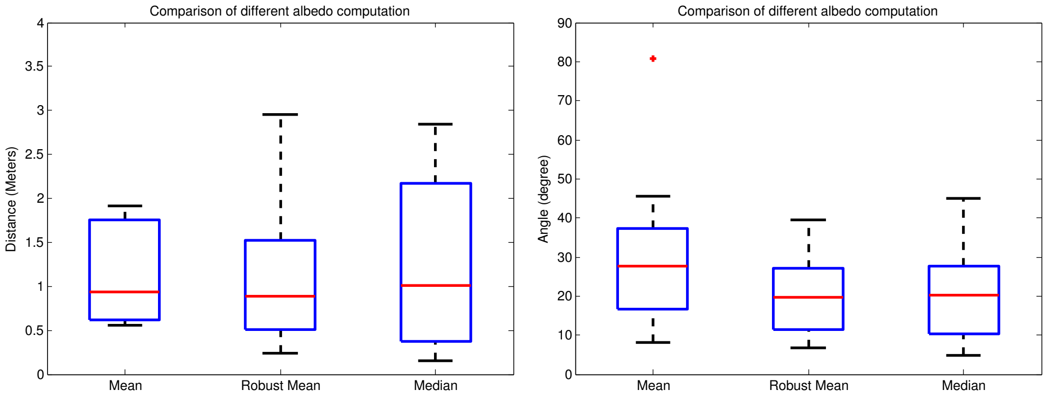

Figure 8: The boxplot of the error in distance and angle between the measured and estimated light source position, for computing the albedo using the mean, robust mean, median. These are obtained from our dataset of 23 different recordings on 6 different scenes.

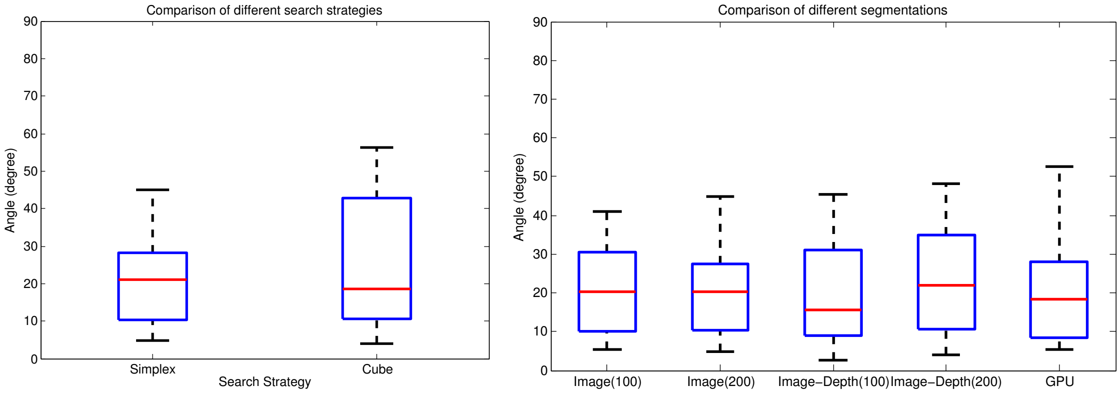

Figure 9: The boxplot of the error in angle between the measured and estimated light source position, left: using different search strategies, right: using different segmentation methods. These are obtained from our dataset of 23 different recordings on 6 different scenes.

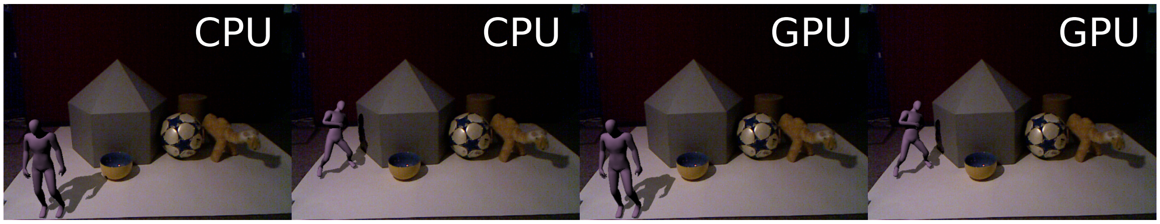

Figure 10: Two frames of a virtual object (character) rendered into the scene showing the interaction between the shadows of the inserted character and the background objects and the difference between CPU and GPU (very small).

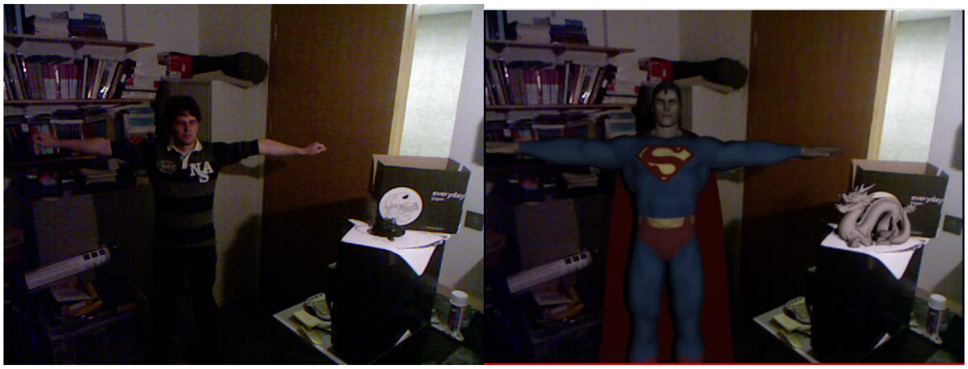

Figure 11: Replacing the person in the scene with a virtual character, by taking into account the lighting of the scene (can you spot the second addition?).