Journal of Real-Time Image Processing manuscript No.
(will be inserted by the editor)

Live User-Guided Depth Map Estimation for Single Images

Márcio C. F. Macedo · Antônio L. Apolinário Jr.

Received: date / Accepted: date

Abstract The availability of depth information in an image enables the simulation of distinct visual effects (*e.g.,* refocus, desaturation, haze) that are related to the distance of the camera to the objects in the scene. To generate depth from color data in single images, existing techniques typically use learning-based strategies or require user-guided depth annotations. Learning-based techniques suffer from generality issues, while user-guided techniques solve a costly optimization problem that prevents a real-time feedback of the depth map generated from the user annotation. In this paper, we overcome the latter problem and propose a GPU-based algorithm that provides live feedback on the output depth map estimated during the user annotation. We follow previous work and treat the depth map estimation as a 2D Poisson problem that can be optimized using a sparse linear solver. However, we change the way that the sparse linear coefficients are computed to favor a more smooth, spatially coherent depth map, able to provide the desired visual effects. Moreover, our approach is designed to run almost entirely on the GPU, achieving real-time performance even for high-resolution images.

Keywords Depth Map · Poisson Equation · Parallel Processing · Real Time

1 Introduction

The increasing popularity of virtual and augmented reality devices (*e.g.,* Facebook Oculus, Microsoft HoloLens) that provide 3D visualization experiences shows that users are interested in applications that augment their perceived depth with respect to a 2D content. Even so, an image is still a 2D matrix that stores color data per pixel, and most of the 2D content available today consists of photographs taken by users and that do not store depth data.

Visual effects achieved using split-depth images [21] and wiggle stereoscopy [28] use motion to improve depth perception in video sequences, but require depth data or at least a pair of left/right images to provide the desired visual effect. The Ken Burns effect requires only a combination of zooming and panning effects to improve depth perception for single images, although the existence of depth information eases its design [27]. Therefore, the availability of depth data is important to augment depth perception for 2D applications.

Single image editing is a challenging task that has been an active topic of research in the literature, with solutions to assist the user in tasks such as image colorization [44] and image synthesis [42]. Depth-based image editing allows the generation of visual effects (*e.g.,* refocus, desaturation, haze) that augment the depth perception and work on the basis of depth data. However, the problem of recovering depth from the color data of a single image is ill-posed. One strategy to make this problem tractable relies on user-guided depth annotation to assist in the depth estimation. Even though semi-automatic techniques provide visually plausible results, they require user intervention and typically do not work in real time. Hence, the user must first annotate the image with depth scribbles and wait a few seconds before the algorithm provides the output depth map (top of Figure 1). Any corrections to be done in the annotation will require the user to wait another few seconds before the display of the final result.

M. Macedo
E-mail: marciocfmacedo@gmail.com

A. Apolinário
E-mail: antonio.apolinario@ufba.br

Federal University of Bahia, Bahia, Brazil

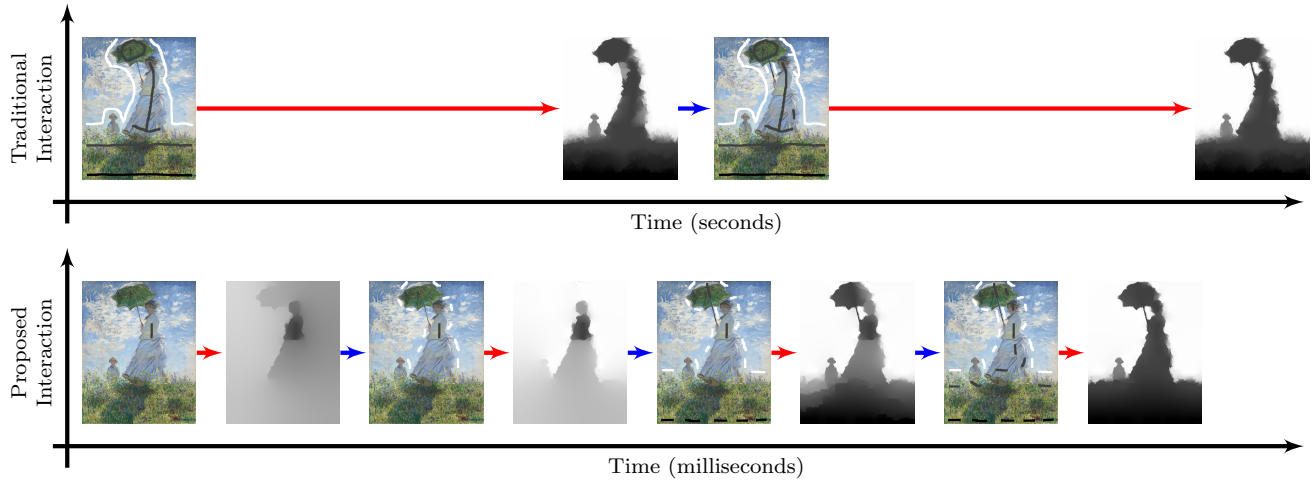


Fig. 1 An illustrated view of user interaction models for depth map annotation. Red arrows indicate the time spent by the algorithm to estimate a depth map. Blue arrows indicate the time spent by the user to annotate an image. (Top) User-guided depth map estimation methods typically expect the user to first annotate the input image, and then solve a costly optimization problem to generate a dense depth map. Small corrections in the annotation require the user to wait a few seconds to see the final result. (Bottom) In our work, we give live previews of the depth map for each user annotation, allowing the user to add less scribbles and/or spend less time to generate the idealized depth map. Top depth maps were generated using the method of Liao *et al.* [22]. Bottom depth maps were generated using our method. Image: Claude Monet's "Woman with a Parasol".

In this work, we propose an algorithm for live user-guided depth map estimation for single images. Depth map estimation is treated as a 2D Poisson problem [22], where user depth scribbles are constraints of a sparse linear system, optimized to generate an edge-aware depth map. By designing an energy function to favor the generation of smooth, spatially coherent depth maps, and by implementing an efficient hierarchical sparse iterative solver to minimize such an energy function on the Graphics Processing Unit (GPU), we are able to generate visually pleasant depth maps in real time, enabling the user to see the live result of his/her depth annotations, as illustrated at the bottom of Figure 1.

The main **contribution** of this work is an interactive user-guided depth map estimation process that uses an efficient GPU implementation to produce smooth depth maps one order of magnitude faster than related work, allowing a user to annotate and interactively refine a depth map on the basis of the live feedback given by the application (the bottom of Figure 1 shows an illustrative example of this contribution).

2 Related Work

Given the considerable amount of work already proposed in the field of depth map estimation, several strategies can be employed to estimate real-world depth data.

The most accurate and expensive strategy for depth map estimation relies on the usage of specialized hardware setup to capture the depth data. Depth sensors based on stereo vision [12,1], structured light [8] or time-of-flight [19] technologies are able to capture and provide high-quality depth data even at real-time frame rates, but cannot recover depth from legacy content taken by the user with a commodity camera.

If the legacy content taken by the user is a video, Structure from Motion (SfM), Simultaneous Localization and Mapping (SLAM) and Visual Odometry (VO) techniques [9,38,35] coupled with a depth propagation approach [14,39,26] are able to take into consideration the spatial coherency and relative motion between consecutive color frames to estimate temporally coherent depth maps. However, these methods require at least a small motion between frames to work properly.

If the legacy content taken by the user is a single image, like a photograph, deep learning [20] is commonly used to estimate a depth map from a single image [17]. Since the work of Saxena *et al.* [36], a large training dataset containing color and depth data is used to train a deep neural network able to estimate the depth of any single image [24]. Inspired by the fields of Shape from Shading (SfS) [15] and Depth from Defocus (DfD) [31], depth cues such as shading [3] and defocus [11] are typically used to aid such a learning process. Thus, deep learning techniques are automatic and provide accurate results when tested on images that are similar (*e.g.,* taken in the same environment, with the same visibility conditions) to the ones present in the training dataset, but fail to generalize for complex, unseen scenes.

As an alternative to learning-based strategies, approaches based on user guidance have been proposed

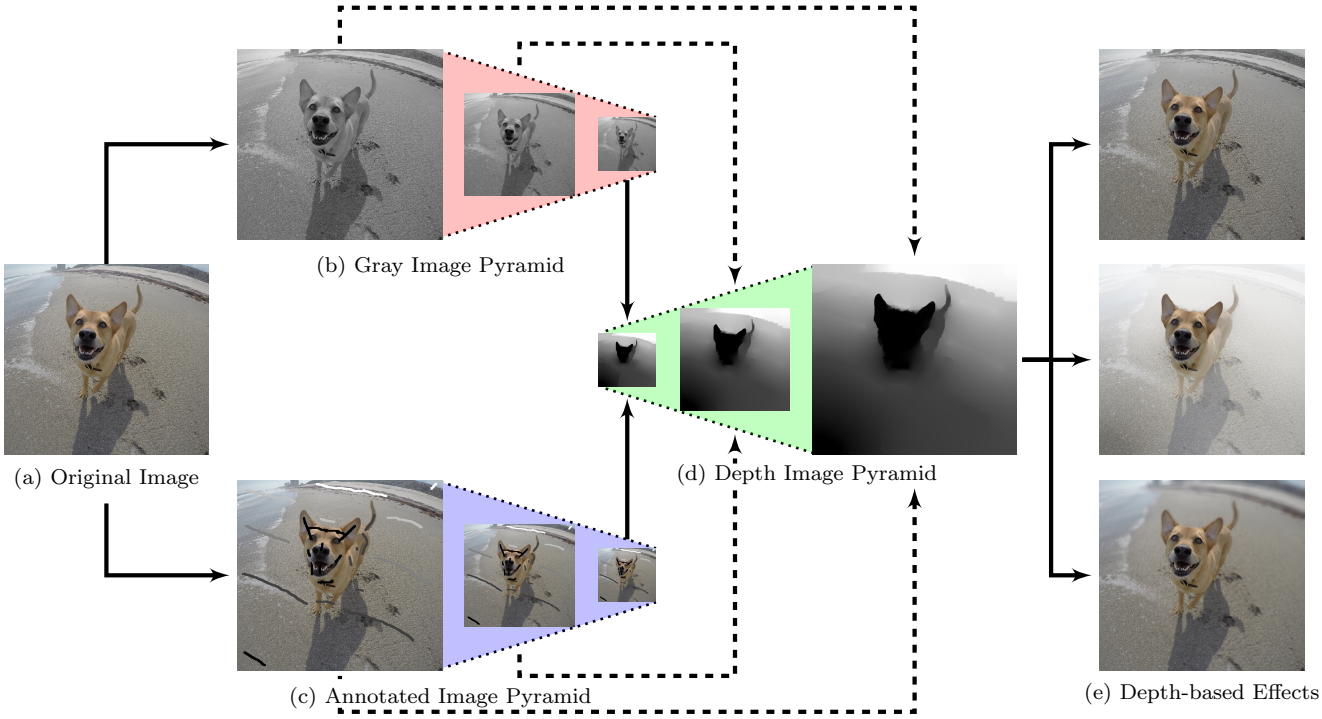


Fig. 2 An overview of the proposed solution. Given an input image (a), we convert it to grayscale (b) and allow the user to annotate the input image with sparse depth scribbles (c). Depth scribbles and gray intensities are used to provide an edge-aware depth map estimation (d). Image pyramids (b, c, d) are built to accelerate depth propagation between neighbour pixels in a multi-scale, bottom-up fashion. Hence, depth-based effects, such as desaturation (top of (e)), haze (middle of (e)) and refocus (bottom of (e)), can be rendered based on the generated depth map. Image (a) is courtesy of Pxfuel.

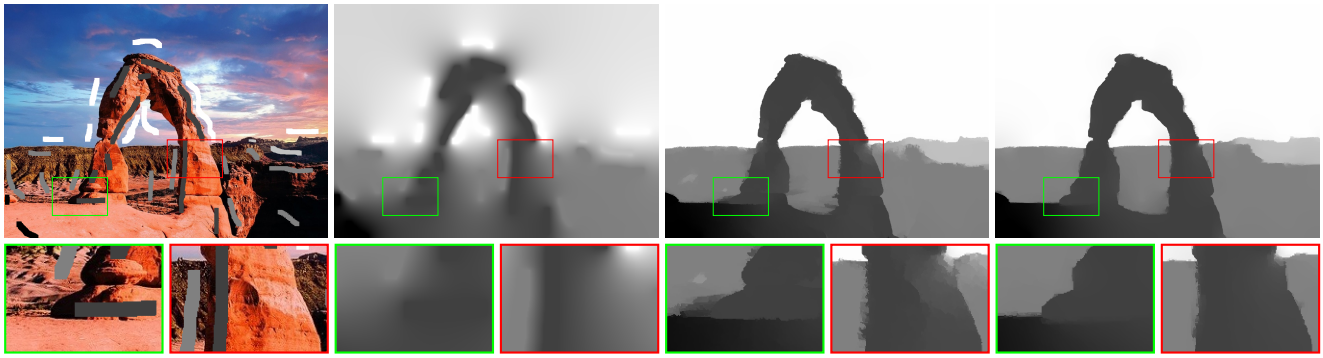
to allow semi-automatic depth estimation from a single image. In general, these approaches operate directly on the image domain, allowing the user to paint the color image with sparse depth scribbles that are propagated in an edge-aware manner to generate a dense depth map. Such a map is later used to generate stereographic content or depth-based effects, such as haze.

Some approaches formulate the depth propagation as a graph-based optimization problem, where the color image is considered as a 4-connected graph, with nodes representing pixels, valued-edges representing the similarity between neighbour pixels, and user depth scribbles being the constraints of the problem. Some works [32, 43] take inspiration by image segmentation methods to propagate depth in the graph. Lopez *et al.* [25] assign edge weights inversely proportional to image gradients, favoring depth propagation at regions with similar color intensities, and reducing depth propagation at the border regions.

Gerrits *et al.* [10] use depth and normal constraints to guide the depth propagation. Liao *et al.* [22] treat the depth propagation problem as an anisotropic Poisson diffusion, and propose several tools (*e.g.,* relative depth, directional guidance) to aid the depth editing. Wang *et al.* [41] propagate depth using a colorization by optimization approach, weighted on the basis of a distance transform of the user depth scribbles and an edge-aware smoothness term. Lin *et al.* [23] adapt the bilateral filter to provide a progressive depth map estimation in a single image. Iizuka *et al.* [16] use superpixel-based segmentation and geodesic distance computation to propagate depth in the entire image.

One problem of the semi-automatic works mentioned above is that they were not designed to provide real-time performance for the depth map estimation, preventing the user to promptly see whether the idealized depth map matches the depth map estimated (top of Figure 1). With a live feedback of the estimated depth map, the user could use less scribbles to achieve the desired effect with a reduced time, as illustrated at the bottom of Figure 1, and the sparse-to-dense propagation method could be adapted for real-time applications, such as monocular augmented reality [14].

We take advantage of the high processing power of a GPU to propose an approach for live user-guided depth map estimation. Inspired by related work [22], we treat the depth map estimation as an anisotropic 2D Poisson problem, and propose an algorithm to solve that problem efficiently on the GPU, almost one order of magnitude faster than the fastest previous work, while generating visually pleasing results.



(a) (b) (c) (d)

Fig. 3 A depth annotated image (a), and the corresponding depth maps generated by solving Equation (3) with $\omega = 1$ (b), ω estimated from (4) (c), and ω estimated from (5) (d). Depth maps were generated by a sparse iterative solver with the same number of iterations. Image (a) by Pete Linforth from Pixabay.

(a) Original Image I (b) Gray Image G (c) Depth Image D (d) Edge Detection on G (e) Edge Detection on D

Fig. 4 Given an input image I (a) and its corresponding grayscale version G (b), traditional edge detection algorithms (*e.g.,* Canny in (d)) are not robust enough to detect only the outer edges that separate the distinct parts of the object in the scene (d). By applying an edge detection algorithm over a depth map D (c) estimated from I, the task of outer edge detection becomes easier (e).

3 User-Guided Depth Annotation

An overview of our proposal is shown in Figure 2. Our algorithm starts with a single, colored image, like the one shown in Figure 2-(a). To generate depth from color data (Figure 2-(d)), we make use of a grayscale representation of the original image (Figure 2-(b)), to provide edge-aware depth map estimation, and sparse depth scribbles annotated on top of the original image (Figure 2-(c)), to guide the depth propagation. Image pyramids (Figures 2-(b, c, d)) are used to speed up the convergence of a sparse iterative solver and provide real-time performance. Distinct depth-based effects, such as desaturation (top of Figure 2-(e)), haze (middle of Figure 2-(e)) and refocus (bottom of Figure 2-(e)) may be rendered on the basis of the visually plausible depth map generated by our algorithm.

Let us denote I as a colored image with m rows and n columns, where each pixel $\mathsf{I}(i,\ j) = [\mathsf{I}_r(i,\ j),\ \mathsf{I}_g(i,\ j),\ \mathsf{I}_b(i,\ j)]^T$, with $i \in [1, n]$ and $j \in [1, m]$, stores red, green and blue color channels, respectively (Figure 2-(a)). Our main goal is to estimate a single-channel depth map D (Figure 2-(d)) from I, where each pixel $\mathsf{D}(i,\ j) \in [0, d_{\max}]$ stores a distance value of its corresponding part of the scene to the camera.

The problem of estimating D from I without prior knowledge of the scene is ill-posed, since the depth data of the 3D scene are lost when projected onto pixels in a 2D image plane. To perform the inverse projection and recover the 3D data given a 2D image, one would need to know the depth of each pixel in advance. To make this problem tractable, we allow the user to define the content of a discrete depth scribble image S, that stores, for each annotated pixel $\mathsf{S}(i,\ j) \in [0, d_{\max}]$, a user-defined depth intensity. By default, non-annotated pixels in S can be assigned to any value $\in [0, d_{\max}]$, since one can use an additional binary image to separate annotated from non-annotated pixels. D is estimated by the propagation of the depth intensities in S over I. An example of visualization of annotated pixels of S shown on top of I can be seen in Figure 2-(c).

The most common strategy to estimate depth from color data using a set of sparse depth scribbles is by solving a 2D Poisson problem [29, 4, 22, 14]. In this work, we follow that same approach and mathematically define the depth intensity propagation on the basis of a 2D Laplace equation, in the form of $\Delta\mathsf{D} = 0$, with Δ

as the Laplace operator, whose discretization is

$$\sum_{(k,l)\in N(i,j)} \mathsf{D}(i,j) - \mathsf{D}(k,l) = 0, \quad (1)$$

where $N(i,j)$ stores the indices of the 4-connected neighbourhood of the pixel $\mathsf{D}(i,j)$, and the depth scribbles S are hard constraints in this formulation, such that $\mathsf{D}(i,j) = \mathsf{S}(i,j)$ for every pixel $\mathsf{S}(i,j)$ that was directly annotated by the user. Our algorithm is designed to allow the user to progressively annotate the depth map. Hence, we initialize $\mathsf{D}(i,j) = d_{\max}$ for non-annotated pixels, since we assume by default that $\mathsf{D}(i,j) = 0$ is used for foreground pixels that are close to the camera, and $\mathsf{D}(i,j) = d_{\max}$ is used for background pixels that are far from the camera. During the user annotation, the values of $\mathsf{D}(i,j)$ for non-annotated pixels are the ones output by the optimization solver.

The solution of (1) favors a smooth propagation of S along the entire image (Figure 3-(b)), since each neighbour contributes equally to the depth value assigned to a pixel. However, we need to add an edge-aware coefficient in (1) to limit the depth propagation to the boundary of the objects in I. To do so, we convert I to a single-channel grayscale image G (Figure 2-(b))

$$\mathsf{G}(i,j) = 0.299\mathsf{I}_r(i,j) + 0.587\mathsf{I}_g(i,j) + 0.114\mathsf{I}_b(i,j) \quad (2)$$

and formulate the edge-aware depth propagation [22]

$$\sum_{(k,l)\in N(i,j)} \omega(i,j,k,l)(\mathsf{D}(i,j) - \mathsf{D}(k,l)) = 0, \quad (3)$$

where $\omega(i,j,k,l) \in [0,1]$, or simply ω, is a coefficient that measures the difference of gray intensities between neighbour pixels $\mathsf{G}(i,j)$ and $\mathsf{G}(k,l)$

$$\omega = \exp(-\beta|\mathsf{G}(i,j) - \mathsf{G}(k,l)|), \quad (4)$$

and β is a user-defined parameter to control the influence of the difference between gray intensities in the depth propagation.

According to (4), ω is close to 0 for neighbour pixels located at the object's boundaries (*i.e.,* where neighbour gray intensities are too different from each other), and is 1 for neighbour pixels with the same gray intensity (*i.e.,* $\mathsf{G}(i,j) = \mathsf{G}(k,l)$).

By solving (3), we are able to greatly improve the visual quality of D, as shown in Figure 3-(c). However, performance and visual quality issues still remain when using such an approach to estimate depth from color data.

Equation (3) can be solved through the construction of a constrained sparse linear system. As discussed in Section 5.2, even for a low-resolution (*e.g.,* 480p) image, the majority of the available sparse iterative solvers do not achieve convergence in real time, hampering the use of this approach for interactive depth map estimation. To reduce such a problem, we take inspiration from multigrid acceleration techniques [6,34] to solve (3) hierarchically on the basis of image pyramids.

Let us assume that $\mathsf{G}^{(p)}$ (Figure 2-(b)), $\mathsf{S}^{(p)}$ (Figure 2-(c)) and $\mathsf{D}^{(p)}$ (Figure 2-(d)) are gray, user-annotated and depth images that, at the level $p \in [0,\alpha]$ of the pyramid, have $\frac{m}{2^p}$ rows and $\frac{n}{2^p}$ columns. Then, we first solve (3) in the coarsest level of the pyramid ($p = \alpha$) and iteratively use the data computed in a coarser level p of the pyramid as an initial guess to solve (3) for the next finer level $p-1$ of the pyramid. This strategy accelerates the convergence of an iterative solver, but is insufficient to provide real-time performance when running on the CPU. In Section 4, we describe our efficient GPU implementation that brings real-time performance for the depth propagation.

The second problem of Equation (3) is related to visual quality. Leaking artifacts may occur if the gray intensity of distinct parts of the scene are not too different from each other, or noise artifacts may appear if an object is mostly composed of distinct gray intensities, as shown in the closeups of Figure 3-(c). To minimize such a problem, we make use of an edge detection algorithm that detects only the outer edges that separate the objects present in the scene, maximizing the depth propagation inside an object, while minimizing the depth propagation between distinct parts of the scene.

Holynski and Kopf [14] have already noted that the traditional edge detection algorithms (*e.g.,* Canny [7]) are not able to detect accurately the outer edges of an object in G (Figure 4-(d)). To solve that problem, they take advantage of the motion between consecutive frames to extract the outer edges. However, their approach cannot be applied to single images, since there is no motion or multiple frames available.

Taking advantage of the multi-scale representation illustrated in Figure 2, used to speed up the convergence of a sparse iterative solver, one can see that even in the coarsest level of the pyramid, where $\mathsf{G}^{(\alpha)}$ is mostly composed of the low-frequency details of $\mathsf{G}^{(0)}$, the depth map $\mathsf{D}^{(\alpha)}$ computed by solving (3) already provides a clear separation between the distinct regions of interest, according to the depths annotated in S. In this case, the boundaries that separate the regions of interest can be easily detected by edge detection algorithms in D, as shown in Figure 4-(e). So, rather than relying solely on the difference between gray intensities when computing ω, we change (4), such that

$$\omega^{(p)} = \begin{cases} \exp(-\beta|\mathsf{G}^{(p)}(i,j) - \mathsf{G}^{(p)}(k,l)|) & \text{if } |\mathsf{D}^{(p)}(i,j) - \mathsf{D}^{(p)}(k,l)| > \gamma \\ 1 & \text{otherwise,} \end{cases} \quad (5)$$

Algorithm 1 Live user-guided depth map estimation

```
1: //preprocessing
2: I ← loadInputImage;
3: α ← estimateMaximumPyramidLevel(I);
4: W ← precomputeWeightTable(β);
5: G^(0) ← convertColorToGrayscale(I);
6: G ← buildPyramid(G);
7: D^(α) ← initializeDepthMap(d_max);
8:
9: //live depth map estimation
10: for each frame do
11:     if user has provided a new annotation then
12:         S^(0) ← updateAnnotatedImage;
13:         S ← buildPyramid(S);
14:         D^(α) ← updateDepthMap(S^(α));
15:     end if
16:     if solver convergence has not been achieved then
17:         for p from α to 1 do
18:             D^(p) ← runSolver(D^(p), G^(p), S^(p), W, p, α);
19:             D^(p−1) ← upsample(D^(p));
20:         end for
21:         D^(0) ← runSolver(D^(0), G^(0), S^(0), W, p, α);
22:     end if
23: end for
```

where $\mathsf{D}^{(p)}$ is an upsampled version of a $\mathsf{D}^{(p+1)}$ previously estimated, p goes from $\alpha - 1$ to 0, since at the coarsest level α, the first $\mathsf{D}^{(\alpha)}$ must be computed using (4), and γ is a depth threshold

$$\gamma = \begin{cases} 4 & \text{if } 0 < p < \alpha \\ 0 & \text{else if } p = 0 \end{cases} \tag{6}$$

A discussion about the values chosen for γ can be seen in Section 5.5.

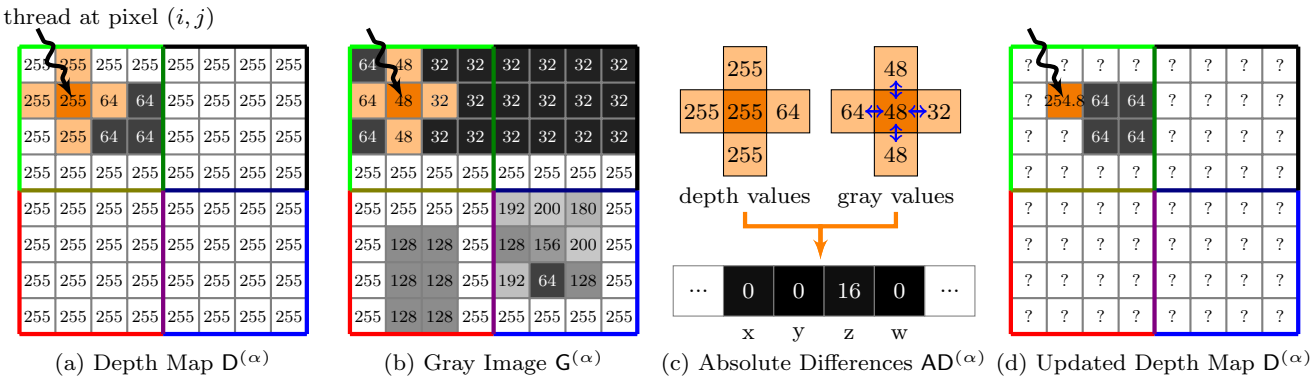
In this updated definition of ω, pixels propagate their depths uniformly on the interior side of an object, similar to the effect achieved when using (1), and the propagation intensity decreases as long as a pixel becomes closer to the boundary of an object. The usage of both G and D images during the solution of the discrete Poisson problem favors spatial coherency and minimize the presence of artifacts during the depth propagation, as depicted in Figure 3-(d).

4 GPU-Based Live Depth Annotation

In this section, we describe our GPU implementation to achieve real-time performance for the user-guided depth map estimation method presented in Section 3. The proposed process is shown in Algorithm 1. By default, GPU images are stored in pitched memory to favor coalesced memory access on the GPU. Also, our image processing kernels associate each thread launched by the application with its corresponding pixel in an image.

In our algorithm, the CPU is mostly used to run part of the preprocessing stage (Lines 2-7), to control the main loop of the application (Lines 10-23), and to call the GPU kernels for parallel processing. In the preprocessing stage, we load the input image I (Figure 2-(a), Line 2) on the CPU and estimate the maximum pyramid level $\alpha = \log(\frac{\min(m,n)}{45} + 1)$ (Line 3), for an input image with m rows and n columns. Assuming that every pixel $\mathsf{G}(i,j) \in [0, 255]$, we know that the integer absolute difference of gray intensities $|\mathsf{G}(i,j) - \mathsf{G}(k,l)|$ lies in the integer interval $[0, 255]$. Given that β is known a priori, we can precompute an array W, on the CPU (Line 4), with the possible values that ω (5) may assume. Hence, $\mathsf{W}(x) = \exp(-x\beta), 0 \leq x \leq 255$. W is loaded as a constant read-only memory on the GPU, to avoid the cost of an intensive evaluation of the exponential function on the GPU solver kernel (Lines 18 and 21). Afterwards, I is copied from CPU to GPU and a GPU kernel is launched to perform the grayscale conversion (2) of I to $\mathsf{G}^{(0)}$ (Line 5). Then, the Gaussian pyramid of G is built on the GPU (Line 6, Figure 2-(b)). Finally, a kernel initializes the coarsest depth map $\mathsf{D}^{(\alpha)}$ with a default value of $d_{\max}$ (Line 7).

In the main loop (Line 10), for each frame, we check the validity of two conditions: whether a new depth annotation has been provided by the user (Line 11) and whether the depth propagation has achieved convergence (Line 16). If the first condition is valid, a GPU kernel paints S with user-defined sparse depth scribbles (Line 12). Then, an image pyramid of S is built as a set of mipmaps that stores, per pixel, maximum rather than average values around corresponding 2×2 neighbours of the previous level on the GPU (Line 13, Figure 2-(c)). Annotated pixels at the coarsest level of S are used to initialize or update $\mathsf{D}^{(\alpha)}$ with the constraints of the solver (Line 14, Figure 2-(d)). Next, we proceed with the depth propagation. If that process has not achieved convergence, we densify $\mathsf{D}^{(\alpha)}$ by solving (3) on the GPU (Lines 18 and 21), and initialize non-annotated pixels in $\mathsf{D}^{(p-1)}$ as an upsampled version of the non-annotated pixels in $\mathsf{D}^{(p)}$ (Line 19, Figure 2-(d)), in order to make sure that the previously computed depth map is used to accelerate the convergence of the solver in a finer pyramid level. Solver convergence is achieved when the residual error is below a tolerance *tol*. However, to keep the real-time performance during the depth propagation, we limit the number of solver iterations per pyramid level to *it*. Hence, even if an accurate depth map cannot be estimated in a single frame, we provide the feedback on D to the user and promote its progressive refinement in the next frames, until convergence measured by *tol* has been achieved.



(a) Depth Map $\mathsf{D}^{(\alpha)}$ (b) Gray Image $\mathsf{G}^{(\alpha)}$ (c) Absolute Differences $\mathsf{AD}^{(\alpha)}$ (d) Updated Depth Map $\mathsf{D}^{(\alpha)}$

Fig. 5 An overview of a single iteration of our parallel solver for the coarsest level α of the image pyramid with resolution 16×16. Given an annotated depth map (a) and a grayscale image (b), we divide both images into CUDA blocks (inside the colored contours) of size $\mathrm{B} \times \mathrm{B}$ (in this example, $\mathrm{B} = 4$ for visualization purpose) and load each block into the shared memory. Then, each GPU thread stores integer absolute differences (blue arrows in (c)) between its pixel (i, j) (strong orange squares) and its neighbour grayscale values (weak orange squares in (b)) in an **int4** type global memory array $\mathsf{AD}^{(\alpha)}$ (see the x, y, z, and w variables in (c)), whenever the depth difference between neighbours is above a depth threshold (assume here $\gamma = 0$, (6)), as required in (5). Afterwards, each thread associated with a non-annotated pixel (pixels labeled with '?' in (d)) accesses neighbour depth values (weak orange squares in (a)) and corresponding ω values (5), computed on the basis of $\mathsf{AD}^{(\alpha)}$ and the precomputed weight table W (*i.e.*, $\omega^{(\alpha)} = \mathsf{W}(\mathsf{AD}^{(\alpha)}(i, j))$) to minimize (3) and update the depth map (d) until convergence or a maximum number of iterations has been achieved.

Both *tol* and *it* are user-defined values, and the ones that we have used in this work are shown in Table 1.

The depth propagation is provided by our parallel hierarchical sparse iterative solver, whose overview is illustrated in Figure 5. To minimize (3) iteratively, we need to compute the weight coefficients $\omega^{(p)}$ (5) per pyramid level p, to separate distinct regions of the image. These weights are estimated on the basis of gray and depth intensities. Knowing that the gray image remains unchanged during the depth propagation, we can avoid the computation of the absolute differences in (5) for every iteration of the solver kernel. Hence, a GPU kernel is launched once per pyramid level p, to store in an **int4** type global memory array $\mathsf{AD}^{(p)}$, the integer absolute differences of gray intensities between a pixel and its 4-connected neighbourhood whenever the depth difference between neighbours is above γ (6), otherwise, 0 is stored, since $\omega = \exp(0) = 1$ (5). To do so, we divide the image domain of both $\mathsf{D}^{(p)}$ (Figure 5-(a)) and $\mathsf{G}^{(p)}$ (Figure 5-(b)) into CUDA blocks of size $\mathrm{B} \times \mathrm{B}$. In Figure 5, those blocks are located inside the colored contours, and $\mathrm{B} = 4$ only for visualization purpose. In our application, we use $\mathrm{B} = 16$, as shown in Table 1. For every block, we load the corresponding contents of $\mathsf{G}^{(p)}$ and $\mathsf{D}^{(p)}$ into shared memories. Then, each GPU thread at pixel (i, j) (strong orange squares in Figure 5) stores the grayscale absolute differences of a pixel and its 4-connected neighbours into an **int4** variable in $\mathsf{AD}^{(p)}$ (Figure 5-(c)).

For each solver iteration, run for each thread associated with a non-annotated pixel (marked as '?' in Figure 5-(d)), corresponding $\omega^{(p)}$ values (5) are accessed using the values stored in $\mathsf{AD}^{(p)}$ as keys to the precomputed weight table W, (*i.e.*, $\omega^{(p)} = \mathsf{W}(\mathsf{AD}^{(p)}(i, j))$). Once with the proper $\omega^{(p)}$ values estimated and $\mathsf{D}^{(p)}$ loaded into the shared memory, each thread is able to compute a new depth value that minimizes (3), as shown in Figure 5-(d). Therefore, a new depth map $\mathsf{D}^{(p)}$ is computed, and threads are synchronized per solver iteration to load consistent, updated neighbour depth values. A Jacobi solver accelerated with the Chebyshev semi-iterative method [40] is used to solve (3), due to its fast convergence and performance (more details in Section 5.2). The output of the Algorithm 1 is the depth image pyramid, as shown in Figure 2-(d).

Table 1 List of parameters.

Symbol	Description	Value
$d_{\max}$	maximum depth intensity	255
β	influence of G on D	0.4
B	GPU block width	16
tol	solver tolerance	10^{-5}
it	solver max. num. iterations	$1000/2^{\alpha-p}$

Our approach supports not only the real-time depth estimation of D on the GPU, but also the real-time visualization of depth-based effects computed on the basis of D (Figure 2-(e)). The generation of those visualizations also runs on the GPU, and allow the user to see, at the same time, both the generated depth map and depth-based effects during the live depth annotation of I. More details on this topic are given in Section 5.3.

Table 2 A ranking of the averaged processing time (in seconds) obtained by our approach using distinct GPU sparse iterative solvers able to solve (3) and generate depth images with different resolutions.

	Image Resolution (s)			
Solver	**480p**	**720p**	**1080p**	**2160p**
ViennaCL BiCGS	0.861	1.734	3.250	10.558
Paralution BiCGS	1.485	1.793	2.566	6.967
CUSP BiCGS	0.446	0.546	0.737	1.755
Gauss-Seidel	0.044	0.054	0.080	0.142
Jacobi+Chebyshev	**0.026**	**0.034**	**0.050**	**0.100**
Jacobi	**0.025**	**0.032**	**0.045**	**0.088**

5 Results

In this section, we evaluate the visual quality and the processing time obtained by our approach in the task of live user-guided depth map estimation for single images.

5.1 Experimental Setup

To run the experiments of this paper, we used an NVIDIA GeForce GTX Titan X and an Intel Core^{TM} i7-3770K CPU (3.50 GHz) with 8GB RAM. To implement our technique, we have used OpenCV 2.3.1 [5] for image processing and CUDA 8.0 [18] for parallel processing.

We compare our approach with the unoptimized, but accurate nonlocal random walks algorithm [43], which is open source[1], and with our implementation of the proposal of Liao *et al.* [22]. To perform that comparison, we have sparsely annotated three images available in the Middlebury dataset [37] using their ground-truth depths in the scribbles. The list of parameters used to run all the experiments is shown in Table 1. We refer the reader to the accompanying video to see the temporal coherency of our method and to the supplementary document for additional comparisons between our approach and related work for distinct datasets.

5.2 Solver Evaluation

There are several solvers available on the literature able to solve (3) efficiently. In this paper, we have performed an evaluation of the processing time required by different iterative solvers when estimating D for distinct image resolutions. Hence, we have chosen to test the most popular sparse iterative solvers implemented on the GPU, since all the CPU implementations tested in our work provided non-interactive performance. We have tested the following solvers: Jacobi, Jacobi combined with the Chebyshev semi-iterative approach [40] (Jacobi+Chebyshev), Gauss-Seidel combined with Successive Over-Relaxation, implemented on the GPU using red-black ordering [30], and bi conjugate gradient stabilized (BiCGS). We tested their implementation in distinct GPU libraries: CUSP[2], Paralution[3], and ViennaCL [33]. In this case, we considered that a solver has achieved convergence if either one of the two criteria of error tolerance *tol* or maximum number of iterations *it* per pyramid level has been achieved. The values used for both criteria are available in Table 1.

As listed in Table 2, the GPU implementation of the bi conjugate gradient stabilized solver provided by CUSP library was the fastest one that we have tested using an external library, but even that optimized implementation required more than 400 milliseconds to solve (3), being far from achieving the desired real-time performance. In this sense, the fastest solver evaluated in our work was the Jacobi, that could achieve real-time performance for images up to the full high-definition resolution ($1080p$) and interactive performance for 4k ($2160p$) images. However, the Jacobi solver is well-known in the literature due to its reduced convergence speed [6,34,40]. Hence, we have chosen to use in our work the second fastest solver listed in Table 2: the Jacobi solver accelerated with the semi-iterative Chebyshev method [40]. That acceleration scheme is more than one order of magnitude faster than most of the other solvers listed in Table 2, greatly improves the convergence rate of the Jacobi solver, and produces visually plausible depth maps able to simulate real-time depth-based effects.

5.3 Visual Quality Evaluation

In Figure 6, we evaluate the accuracy of the depth images generated by our approach and related work given the same input images and depth annotations. Intensity differences are evaluated in terms of the root mean squared error (RMSE).

In the Baby image shown on the left of Figure 6, the work of Liao *et al.* (Figure 6-(e)) generates more noisy artifacts along the edges of the scene than our approach (Figure 6-(f)). The work of Yuan *et al.* (Figure 6-(d)) minimizes those artifacts, despite incorrectly assigning a depth value near the right hand of the Baby doll.

For the Midd scenario shown in the middle of Figure 6, all the approaches recover a depth map that resembles the ground-truth (Figure 6-(c)), but our work and the work of Liao *et al.* are slightly more prone to fail in the measurement of appropriate depth values along the edges of the objects in the scene (Figures 6-(e, f)),

[1] https://github.com/tcyhx/NRW

[2] https://cusplibrary.github.io/

[3] https://www.paralution.com/

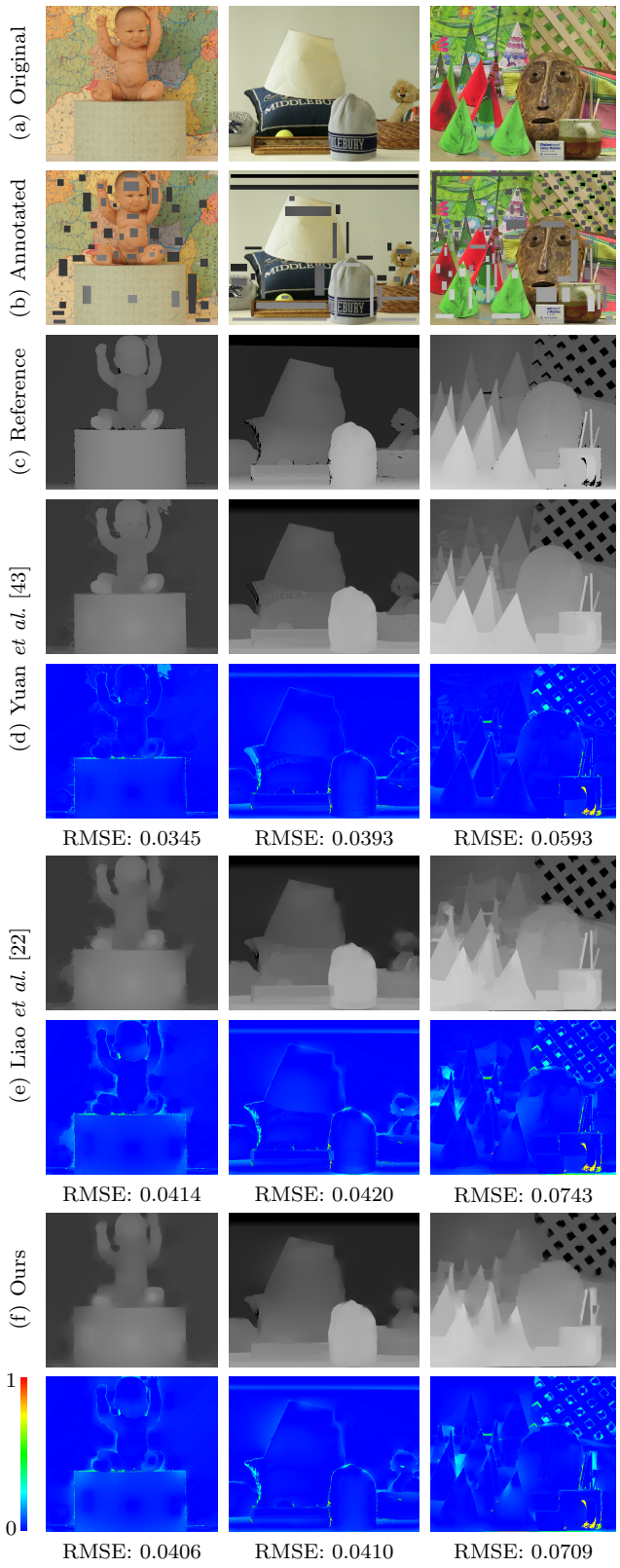


Fig. 6 A comparison between the depth maps generated by related work (d, e) and our approach (f) for the sparsely depth annotated (b) Baby (left, 620×555 resolution), Midd (middle, 683 × 555 resolution) and Cones (right, 450 × 375 resolution) scenes of the Middlebury [37] dataset (a). False color maps show intensity differences to the reference depth map (c).

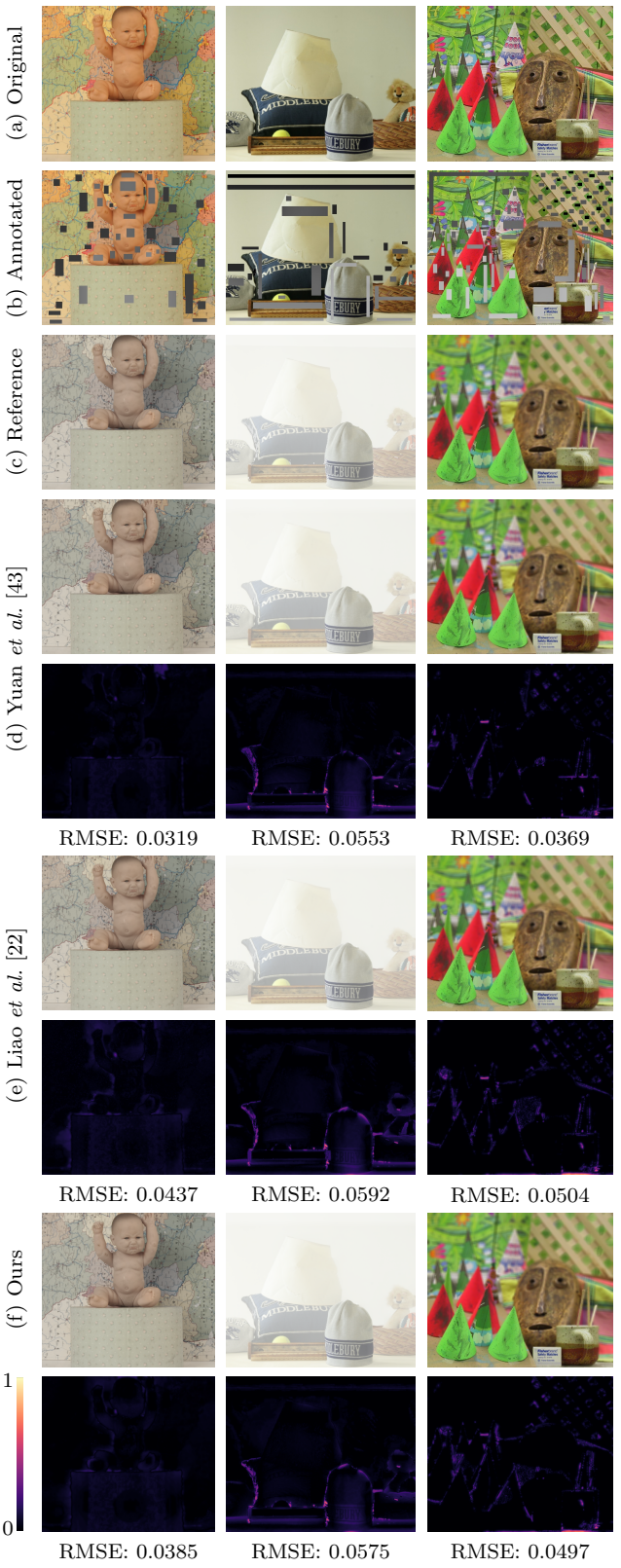


Fig. 7 A comparison between desaturation (left), haze (middle) and refocus (right) depth-based effects generated by related work (d, e) and our approach (f) for the sparsely depth annotated (b) Baby (left, 620×555 resolution), Midd (middle, 683 × 555 resolution) and Cones (right, 450 × 375 resolution) scenes of the Middlebury [37] dataset (a). False color maps show perceptual differences to the reference image (c).

mainly due to the color similarity between the white object and the background wall (Figure 6-(b)).

For the challenging Cones scenario in the right of Figure 6, our work and the work of Liao *et al.* (Figures 6-(e, f)) produce noisy depth maps that fail to capture high-frequency details present in the reference depth map (Figure 6-(c)), artifacts that are minimized by the approach of Yuan *et al.* (Figure 6-(d)).

In this work, our main focus is not only the generation of high-quality depth maps given a set of sparse depth annotations, but also, the generation of depth-based effects on the basis of those estimated depth maps. As shown in Figure 2, we implemented three different depth-based effects on the GPU to demonstrate, in practice, how depth maps can be used to augment depth perception in single images.

The haze effect [13], shown at the top of Figure 2-(e) and in the middle of Figure 7, uses the scene depth to exponentially attenuate scene radiance in I. The desaturation effect, visible in the middle of Figure 2-(e) and in the left of Figure 7, uses depth as a weight to linearly interpolate the input image I and the grayscale image G. The refocus effect, displayed in the bottom of Figure 2-(e) and in the right of Figure 7, consists on the convolution of an anisotropic kernel over the input image, whose kernel order varies in a per-pixel basis, according to the distance of the center pixel to the camera.

In Figure 7, we applied the depth-based effects over the original images of Figure 6, to evaluate how much the depth map quality influences on the quality of the final rendered image. To assist us in that evaluation, we have estimated the RMSE of the perceptual differences (using the FLIP metric [2]) between the reference images (Figure 7-(c)) and the ones generated by each depth estimation method. One can see that all of the methods produce depth-based effects perceptually too much similar to each other, with the ones produced by Yuan *et al.* [43] slightly more accurate than the ones. That general observation is also held for the distinct scenes evaluated in the supplementary document.

5.4 Performance Evaluation

We have evaluated the processing times obtained by our approach and related work when estimating the depth maps for three scenes of the Middlebury dataset, shown in Figure 6. The work of Yuan *et al.* provides performance far from real time for the user-guided depth map estimation, requiring more than **5 seconds** to generate the depth maps. The work of Liao *et al.* is faster than the work of Yuan *et al.*, but still demanded more than **500 milliseconds** to compute the depth maps. Our work is faster than related work since, when running solely on the CPU, it required approximately **200 milliseconds** to solve the depth estimation problem, and when implemented on the GPU, it achieved real-time performance in all the tested scenarios, running in less than **30 milliseconds** to generate the maps of Figure 6.

We analyzed our parallel implementation described in Section 4 using a GPU profiler, and could measure that more than 95% of the time spent by our algorithm is dedicated to run the Jacobi + Chebyshev solver, the costly step of our algorithm.

5.5 Discussion

The main goal of this work is to propose a new algorithm to generate depth maps, in real time, on the basis of a single color image. As we show in Figure 2, the availability of a depth map eases the task of image editing with respect to the generation of distinct depth-based effects, such as desaturation, haze, and refocus. In this sense, while automatic solutions provided by deep learning, for instance [36, 24, 11], could also generate depth maps from single color images, we believe that a real-time, user-assisted method for depth map generation gives more control to the user about how the depth map can be generated (*e.g.,* realistically, artistically), while enabling a prompt visualization of the generated depth map and desired depth-based effect, and while working even if the image to be edited does not match the characteristics of the datasets on which learning algorithms have been trained.

As shown in Figure 6-(f), given a sparse depth annotation of an input image, our algorithm is able to recover a smooth depth map that resembles the ground-truth, mainly in terms of its low-frequency details, but that deviates from the ground-truth in regions with high-frequency details (*i.e.,* edges). Even so, the estimated results presented here and in the supplementary document show that our approach is, in general, numerically (Figure 6-(f)) and perceptually (Figure 7-(f)) more accurate than the work of Liao *et al.*. In practice, Figure 7 shows that the depth-based effects produced by our approach are coherent with the sparse depth annotations available and are also similar to the ones produced by related work. In this sense, the main advantage of our approach is that we are able to produce depth maps in real time, achieving, for instance, 30 frames per second in average to generate depth maps for high-resolution ($720p$) images. We could show that our work, the only one implemented on the GPU, is able to generate visually pleasant depth maps and corresponding depth-based effects at real-time frame rates,

enabling the user to see a live feedback on the depth map (and on the depth-based effect, if desired) generated on the basis of his/her sparse depth scribbles, enhancing the user control over the depth annotation.

Our proposed approach is useful for real-time applications that demand plausible depth maps, such as augmented reality, or for artistic image editing applications that work on the basis of depth annotations to improve depth perception in single images, and aim a more interactive algorithm able to provide a live feedback of the depth annotations.

As for the parameters of our algorithm, at the coarsest level ($p = \alpha$) we fall back to the weight coefficient as determined in (4), since the depth map is too coarse to be used to detect edges, or has a default initialization in the first iteration of the algorithm. Then, we use $\gamma = 4$ at coarser levels of the pyramid ($0 < p < \alpha$) to favor the generation of a smooth depth map, and use $\gamma = 0$ in the finest pyramid level ($p = 0$), to consider high-frequency details in the last iterations of the depth propagation algorithm. Moreover, our solver takes advantage of the coarse-to-fine strategy used to propagate depth for each pyramid level, to reduce by a half the number of iterations *it* needed to solve (3) per pyramid level. Hence, we spend more iterations to solve (3) in the coarsest pyramid level, and just a few iterations to include the high-frequency details of the finest pyramid level in the final depth map, achieving a balance between accuracy and performance.

Similarly to the majority of the related work in the field of user-guided depth map estimation, our algorithm produces wrong and incoherent depth maps given a set of incorrect depth annotations, but this problem can be solved by the use of an undo operation. Moreover, our algorithm is not robust under the presence of neighbour objects with similar color intensities, a problem that is visible in the Midd scenario in the middle of Figure 6, where the white object and the wall have similar color intensities. A more detailed, dense annotation over the input image may alleviate depth artifacts generated along edges and favor the representation of high-frequency details in the output depth map. Also, by providing a live feedback of the generated depth map, we enable the user to correct or improve the depth annotation as soon as any of those limitations is perceived.

6 Conclusion and Future Work

In this paper, we have presented an algorithm that allows a user to annotate a single image with sparse depth scribbles and generate a plausible depth map in real time, on the basis of a GPU-based edge-aware propagation of the sparse depth scribbles annotated on the input image. We have shown that we could generate visually pleasant depth maps at approximately 30 frames per second even for high-resolution images. In this sense, our algorithm is suitable for applications that demand real-time feedback and work with legacy content in the form of single images.

For future work, one could make use of more efficient solvers to reduce the processing time of our solution. Moreover, we believe that our work could be integrated with alternative constraints (*e.g.,* relative depth [22], equality/inequality [25]) to guide the depth map estimation. Our approach could be tested to propagate depth in monocular augmented reality applications [14], where sparse depth points generated by a SLAM algorithm would be used as a basis to generate the depth map. Also, a user study could be conducted to evaluate the impact of the live feedback provided by our solution in terms of the quality, usability and performance of the depth annotation process.

Acknowledgements We are thankful to Yuan *et al.* [43] for gently sharing the source code of their depth map estimation algorithm. This research is financially supported by the Postdoctoral National Program of Coordenação de Aperfeiçoamento de Pessoal do Nível Superior (PNPD/CAPES). The graphics card used in the experimental setup was provided by NVIDIA through the GPU Education Center. This is a post-peer-review, pre-copyedit version of an article published in Journal of Real-Time Image Processing. The final authenticated version is available online at this link[4]

Conflict of interest

The authors declare that they have no conflict of interest.

Code availability

The source code can be found at this link[5].

References

1. Aguilera, C.A., Aguilera, C., Navarro, C.A., Sappa, A.D.: Fast CNN Stereo Depth Estimation through Embedded GPU Devices. Sensors (Basel) **20**(11) (2020)
2. Andersson, P., Nilsson, J., Akenine-Moller, T., Oskarsson, M., Astrom, K., Fairchild, M.: ꟻLIP: A Difference Evaluator for Alternating Images. ACM Comput. Graph. Interact. Tech. **3**(2), 15:1–15:23 (2020)
3. Bednarik, J., Fua, P., Salzmann, M.: Learning to Reconstruct Texture-Less Deformable Surfaces from a Single View. In: Proceedings of the 3DV, pp. 606–615 (2018)

[4] https://doi.org/10.1007/s11554-020-01055-x
[5] https://github.com/MarcioCerqueira/RealTimeDepthDiffusion

4. Bezerra, H., Eisemann, E., DeCarlo, D., Thollot, J.: Diffusion Constraints for Vector Graphics. In: Proceedings of the NPAR, pp. 35–42. ACM, New York, NY, USA (2010). DOI 10.1145/1809939.1809944
5. Bradski, G., Kaehler, A.: Learning OpenCV: Computer Vision in C++ with the OpenCV Library, 2nd edn. O'Reilly Media, Inc. (2013)
6. Briggs, W.L., Henson, V.E., McCormick, S.F.: A Multigrid Tutorial (2nd Ed.). Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2000)
7. Canny, J.: A Computational Approach to Edge Detection. IEEE Trans. Pattern Anal. Mach. Intell. **8**(6), 679–698 (1986). DOI 10.1109/TPAMI.1986.4767851
8. Cruz, L., Lucio, D., Velho, L.: Kinect and RGBD Images: Challenges and Applications. In: Proceedings of the SIBGRAPI Tutorials, pp. 36–49 (2012)
9. Engel, J., Koltun, V., Cremers, D.: Direct Sparse Odometry. IEEE Trans. Pattern Anal. Mach. Intell. **PP**(99), 1–1 (2017). DOI 10.1109/TPAMI.2017.2658577
10. Gerrits, M., De Decker, B., Ancuti, C., Haber, T., Ancuti, C., Mertens, T., Bekaert, P.: Stroke-based creation of depth maps. In: Proceedings of the ICME, pp. 1–6 (2011). DOI 10.1109/ICME.2011.6012006
11. Gur, S., Wolf, L.: Single Image Depth Estimation Trained via Depth from Defocus Cues. In: Proceedings of the CVPR (2019). DOI 10.1109/CVPR.2019.00787
12. Hamzah, R., Ibrahim, H.: Literature Survey on Stereo Vision Disparity Map Algorithms. Journal of Sensors **2016**, 1–23 (2016). DOI 10.1155/2016/8742920
13. He, K., Sun, J., Tang, X.: Single Image Haze Removal Using Dark Channel Prior. IEEE Trans. Pattern Anal. Mach. Intell. **33**(12), 2341–2353 (2011)
14. Holynski, A., Kopf, J.: Fast Depth Densification for Occlusion-aware Augmented Reality. ACM Trans. Graph. **37**(6), 194:1–194:11 (2018)
15. Horn, B.K.P., Brooks, M.J. (eds.): Shape from Shading. MIT Press, Cambridge, MA, USA (1989)
16. Iizuka, S., Endo, Y., Kanamori, Y., Mitani, J., Fukui, Y.: Efficient Depth Propagation for Constructing a Layered Depth Image from a Single Image. Computer Graphics Forum **33**(7), 279–288 (2014)
17. Khan, F., Salahuddin, S., Javidnia, H.: Deep Learning-Based Monocular Depth Estimation Methods-A State-of-the-Art Review. Sensors (Basel) **20**(8) (2020)
18. Kirk, D.B., Hwu, W.m.W.: Programming Massively Parallel Processors, Third Edition: A Hands-on Approach, 3rd edn. Morgan Kaufmann Publishers Inc. (2016)
19. Kolb, A., Barth, E., Koch, R., Larsen, R.: Time-of-Flight Cameras in Computer Graphics. Computer Graphics Forum **29**(1), 141–159 (2010)
20. LeCun, Y., Bengio, Y., Hinton, G.: Deep Learning. Nature **521**(7553), 436–444 (2015)
21. Liao, J., Eisemann, M., Eisemann, E.: Split-Depth Image Generation and Optimization. Computer Graphics Forum **36**(7), 175–182 (2017). DOI 10.1111/cgf.13283
22. Liao, J., Shen, S., Eisemann, E.: Depth annotations: Designing depth of a single image for depth-based effects. Computers & Graphics **71**, 180 – 188 (2018)
23. Lin, Y.H., Tsai, M.H., Wu, J.L.: Depth sculpturing for 2d paintings: A progressive depth map completion framework. Journal of Visual Communication and Image Representation **25**(4), 670 – 678 (2014)
24. Liu, F., Shen, C., Lin, G., Reid, I.: Learning Depth from Single Monocular Images Using Deep Convolutional Neural Fields. IEEE Trans. Pattern Anal. Mach. Intell. **38**(10), 2024–2039 (2016)
25. Lopez, A., Garces, E., Gutierrez, D.: Depth from a Single Image Through User Interaction. In: Proceedings of the CEIG. The Eurographics Association (2014)
26. Luo, X., Huang, J.B., Szeliski, R., Matzen, K., Kopf, J.: Consistent Video Depth Estimation. ACM Trans. Graph. **39**(4) (2020)
27. Niklaus, S., Mai, L., Yang, J., Liu, F.: 3D Ken Burns Effect from a Single Image. ACM Trans. Graph. (2019)
28. Ogawa, N., Narumi, T., Hirose, M.: Swinging 3D Lamps: A Projection Technique to Convert a Static 2D Picture to 3D Using Wiggle Stereoscopy. In: Proceedings of the ACM SIGGRAPH Posters, pp. 29:1–29:2 (2017)
29. Orzan, A., Bousseau, A., Winnemöller, H., Barla, P., Thollot, J., Salesin, D.: Diffusion Curves: A Vector Representation for Smooth-shaded Images. ACM Trans. Graph. **27**(3), 92:1–92:8 (2008)
30. Pall, P., Nylén, O., Fratarcangeli, M.: Fast Quadrangular Mass-spring Systems Using Red-black Ordering. In: Proceedings of the VRIPHYS, pp. 37–43 (2018)
31. Pentland, A.P.: A New Sense for Depth of Field. IEEE Trans. Pattern Anal. Mach. Intell. **9**(4), 523–531 (1987)
32. Phan, R., Androutsos, D.: Robust Semi-Automatic Depth Map Generation in Unconstrained Images and Video Sequences for 2D to Stereoscopic 3D Conversion. IEEE Transactions on Multimedia **16**(1), 122–136 (2014)
33. Rupp, K., Tillet, P., Rudolf, F., Weinbub, J., Grasser, T., Jngel, A.: ViennaCL-Linear Algebra Library for Multi- and Many-Core Architectures. SIAM Journal on Scientific Computing (2016-10-27). DOI 10.1137/15m1026419
34. Saad, Y.: Iterative Methods for Sparse Linear Systems, 2nd edn. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2003)
35. Saputra, M.R.U., Markham, A., Trigoni, N.: Visual SLAM and Structure from Motion in Dynamic Environments: A Survey. ACM Comput. Surv. **51**(2), 37:1–37:36 (2018). DOI 10.1145/3177853
36. Saxena, A., Sun, M., Ng, A.Y.: Make3D: Learning 3D Scene Structure from a Single Still Image. IEEE Trans. Pattern Anal. Mach. Intell. **31**(5), 824–840 (2009)
37. Scharstein, D., Szeliski, R., Zabih, R.: A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. In: Proceedings of the SMBV, pp. 131–140 (2001). DOI 10.1109/SMBV.2001.988771
38. Song, J., Wang, J., Zhao, L., Huang, S., Dissanayake, G.: MIS-SLAM: Real-Time Large-Scale Dense Deformable SLAM System in Minimal Invasive Surgery Based on Heterogeneous Computing. IEEE Robot. Autom. Lett. **3**(4), 4068–4075 (2018). DOI 10.1109/LRA.2018.2856519
39. Valentin, J., Kowdle, A., Barron, J.T., Wadhwa, N., Dzitsiuk, M., Schoenberg, M., Verma, V., Csaszar, A., Turner, E., Dryanovski, I., Afonso, J., Pascoal, J., Tsotsos, K., Leung, M., Schmidt, M., Guleryuz, O., Khamis, S., Tankovitch, V., Fanello, S., Izadi, S., Rhemann, C.: Depth from Motion for Smartphone AR. ACM Trans. Graph. **37**(6), 193:1–193:19 (2018)
40. Wang, H.: A Chebyshev Semi-iterative Approach for Accelerating Projective and Position-based Dynamics. ACM Trans. Graph. **34**(6), 246:1–246:9 (2015)
41. Wang, O., Lang, M., Frei, M., Hornung, A., Smolic, A., Gross, M.: StereoBrush: Interactive 2D to 3D Conversion Using Discontinuous Warps. In: Proceedings of the SBIM, pp. 47–54 (2011)
42. Wang, T., Liu, M., Zhu, J., Tao, A., Kautz, J., Catanzaro, B.: High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs. In: Proceedings of the CVPR, pp. 8798–8807 (2018)

43. Yuan, H., Wu, S., Cheng, P., An, P., Bao, S.: Nonlocal Random Walks Algorithm for Semi-Automatic 2D-to-3D Image Conversion. IEEE Signal Process. Lett. **22**(3), 371–374 (2015). DOI 10.1109/LSP.2014.2359643
44. Zhang, R., Zhu, J.Y., Isola, P., Geng, X., Lin, A.S., Yu, T., Efros, A.A.: Real-time User-guided Image Colorization with Learned Deep Priors. ACM Trans. Graph. **36**(4), 119:1–119:11 (2017)