Volume xx (200y), Number z, pp. 1–16

From Noon to Sunset: Interactive Rendering, Relighting, and Recoloring of Landscape Photographs by Modifying Solar Position

M. Türe[1], M. E. Çıklabakkal[2], A. Erdem[3], E. Erdem[4], P. Satılmış[4], and A. O. Akyüz[5]

[1]Taleworlds Entertainment, Turkey
[2]University of Waterloo, Canada
[3]Koç University, Turkey
[4]Hacettepe University, Turkey
[5]Middle East Technical University, Turkey

Abstract

Image editing is a commonly studied problem in computer graphics. Despite the presence of many advanced editing tools, there is no satisfactory solution to controllably update the position of the sun using a single image. This problem is made complicated by the presence of clouds, complex landscapes, and the atmospheric effects that must be accounted for. In this paper, we tackle this problem starting with only a single photograph. With the user clicking on the initial position of the sun, our algorithm performs several estimation and segmentation processes for finding the horizon, scene depth, clouds, and the sky line. After this initial process, the user can make both fine-scale and drastic changes on the position of the sun: it can be set beneath the mountains or moved behind the clouds practically turning a midday photograph into a sunset (or vice-versa). We leverage a precomputed atmospheric scattering algorithm to make all of these changes not only realistic but also real-time. We demonstrate our results using both clear and cloudy skies, showing how to add, remove, and relight clouds, all the while allowing for advanced effects such as scattering, shadows, light shafts, and lens flares.

CCS Concepts
• ***Computing methodologies*** → ***Rendering; Image manipulation;***

1. Introduction

Sunsets and sunrises are among the most popular photographic subjects. However, capturing those moments are not simple especially if the photographer is also concerned with adjusting other scene elements to achieve a good composition. All important scene elements must be positioned properly and desired poses must be attained for all live subjects. The camera parameters must be set correctly, all the while accounting for the overall appearance shifts due to the continuous motion of the sun. Most importantly, decisions must be made quickly as there is generally a limited time-frame during which an ideal composition can be obtained. It is not uncommon for professional photographers to plan and wait for hours, even days, to capture compelling sunset and sunrise pictures.

In this paper, we propose an image editing technique particularly suited for modifying the appearance of images with large skies and visible sun. With our technique, the position of the sun in the sky can be updated freely, which allows for both small and large scale modifications to the original picture (Figure 1). Small scale modifications can be made if the user is generally satisfied with the photograph, but wants to make some subtle adjustments to the sun's position. Our technique also allows drastic modifications such as moving the sun behind the horizon, mountains, and clouds, bringing it back up, changing its both azimuth and elevation as well as entirely removing the sun or adding it to a picture that is devoid of it. Such modifications entail overall changes to the photograph so that the entire photograph remains consistent with respect to the sun's updated state. We show that such effects are not only possible but can be applied in real-time to allow an interactive solution.

Our algorithm is comprised of three key stages namely segmentation, rendering, and recoloring. In the segmentation stage, the important components such as the foreground, sun, sky, and horizon are detected, and a soft segmentation mask for the clouds are computed. In the rendering stage, using the sun's position and the estimated camera parameters, a precomputed atmospheric scattering algorithm [Ele09] is applied to re-render the sky together with the clouds [Sch16]. The recoloring stage involves color transfer from the original sky and recoloring of the the foreground to make the entire photograph consistent with the updated position of the sun and the sky. With these three stages, the proposed algorithm can produce compelling results for a wide range of input photographs. To this end, the key contributions of the current work are:

Input Image Our Results for Different Sun Positions

Figure 1: Our algorithm allows modifying the position of the Sun, as well as its size and several other parameters, in an input photograph. The left-most image in this figure is the original photograph and the other images are our edits performed in real-time. The appearance of the sky regions and the foreground is automatically updated to make it consistent with the updated Sun position. Input image kindly shared at https://pixabay.com/en/patagonia-road-mountains-2428981.

- The first single-image-based sun position modification algorithm,
- Realistic handling of various appearance effects that stem from updating the sun's position,
- A real-time implementation allowing all modifications to be fully interactive.

In the following, we first review the related work followed by an algorithmic overview and details of each step. We then demonstrate our visual results, compare it with alternatives, discuss its limitations, and outline future research directions.

2. Related Work

The proposed work is closely related to studies that aim to perform advanced image editing tasks using a single image, such as Poisson image editing [PGB03], image-based material editing [KRFB06], time-of-day modification from a single picture [SPDF13], and sky replacement [TSL*16]. However, to our knowledge, the proposed work is the first attempt to modify the sun's position directly in a picture in which the sun is directly visible.

Image editing, as it refers to altering the appearance of images by inserting new objects and/or modifying and removing existing ones, is an extensively studied subject within computer graphics. A powerful technique for image editing involves solving Poisson equations to perform seamless cloning between images or to perform seamless appearance edits within a single image [PGB03]. This technique is extended for homogeneous Poisson equations, also known as Laplace equations, for seamless cloning [JCW09]. Other studies have shown that solving for Poisson equations are not required for seamless cloning and edits as long as correct boundary conditions and smooth changes within the interior regions are satisfied [FHL*09].

Existing image editing techniques are not directly applicable in the case of solar modifications as proposed in the current paper. This is because most image editing algorithms are concerned with making static and local edits by means object insertions and/or removals, whereas solar modifications require dynamic and global changes as a result of the changing sun position.

Image Relighting and Recoloring. Related to image editing are techniques that involve relighting or recoloring the entire image or selected parts of it to simulate various lighting and material changes. These techniques range from advanced methods that aim to separate an image into its intrinsic layers [ED04, BBS14, RDL*15, MZRT16] to simple methods that update pixel values using various heuristics. For relighting, some algorithms require a second image as a source of illumination or colors [RAGS01, WAM02]. Other algorithms require user strokes to provide hints to allow relighting of different objects in different a manner [LFUS06]. Auto-generated or user-assisted alpha mattes are used to update the materials of objects and relight them using their new material properties [KRFB06]. Lately, some researchers also proposed deep learning based image relighting methods, which have the ability to modify the visual appearance of the images of single objects [XSHR18], indoor [MGAD19] and outdoor [PGZ*19, YME*20] scenes or portrait images [SBT*19, ZHSJ19]. However, all these recent techniques require a large set of training images, which are photographed with some predetermined lighting directions, and/or some auxiliary data such as depth maps or 3d geometry.

Time of Day Modification. There are also several relighting studies that simulate the effects of time-of-day on a captured photograph. Chandra et al. developed a technique that can simulate time-of-day variations in appearance of aerial images [CAT06]. Their method, however, requires the orientation of the surfaces as well as the location, date, and time at which the photograph was taken to be known. Madsen and Laursen proposed a real-time technique for image-based relighting [ML04]. However, their work also requires a large amount of input information to be known such as the 3D model of the scene that is registered to the input photograph as well as the original lighting conditions of the scene. In a more recent approach, Balci and Gudukbay proposed to track the sun's position from time-lapse videos and use this information to insert virtual objects in real videos [BG17]. However, their method requires a time-lapse video instead of a single image, as input. Furthermore, some manual intervention is required to extract the scene geometry from the input video. Here we should also note that there are some recent studies that aim at generating a time-lapse video from a single image [NMC*19, CCC20, LSK*20].

In another related study, Shih et al. [SPDF13] proposed an algorithm to change the time of day of a photo. This algorithm uses a data-driven approach. The color transfer from one frame of the video (for example sunrise) to another (night) is learned by a model. A matching time-lapse video is found for the input image

and the affine color transfer for that frame is applied to the input image. More recently, Karacan et al. [KAEE19] and Anokhin et al. [ASK*20] have developed image-to-image translation methods that are based on deep generative networks and have the ability to make daytime changes to a given scene image.

Sky Replacement. As one of the most related algorithms to the current work, Tsai et al. [TSL*16] proposed an algorithm to change the sky with another one from a suitable image. First, the scene is segmented by a fully convolutional neural network (FCNN) [LSD15]. Afterwards, the result of the FCNN is refined to match it with the input resolution. Then, by using the semantic information about the image, images with a matching sky box are found to be used as a replacement for the sky in the original image. To correctly update the color of the foreground, semantic data from both images are used.

Similar to Tsai et al., Rawat et al. [RGSN18] also aimed to replace the sky of a photograph to make it visually more appealing. However, different from the earlier work, their focus was finding the most appropriate sky that would not require a color correction to be performed after the sky replacement. More recently, Liu et al. [LGZ*20] proposed a learning based framework to disentangle an input image into temporally-varying illumination and permanent scene elements. It then becomes possible to change the temporal factors while keeping the permanent elements fixed. However, this work is more suitable for city-scapes where time-lapse imagery from Google Street View is available. It also does not aim to create complex atmospheric scattering and cloud shading effects, as we do in the current work.

Single-Image Depth Estimation. Some of the aforementioned methods, as well as our method, require an estimate of the scene depth for more accurate lighting computations. Predicting depth from single view images has been one of the challenging problems in computer vision and has received much attention. Especially with the introduction of deep learning based methods, researchers have been trying to solve this problem by casting it as an image-wide regression problem [EPF14, LSL15, LRB*16, RT16]. One of the most robust methods for this problem is the MegaDepth algorithm [LS18]. The method differs from the other existing works in several aspects. First and foremost, it is trained on a new large-scale dataset, which is more suitable for estimating depth of natural scenes. The previous depth datasets such as NYU [SHKF12], Make3D [SSN09], and KITTI [Gei12] contain depth images that are captured via either RGB-D sensors such as Kinect, which is limited to indoor use or laser scanning devices such as LIDAR which results in sparse depth maps. On the other hand, MegaDepth is collected from images with overlapping viewpoints that are available in photo collections found on the web. Specifically, the authors proposed to employ structure-from-motion (SfM) and multi-view stereo (MVS) methods to obtain dense depth maps for both indoor and outdoor images. They also employ semantic segmentation to remove noisy MVS depths especially in the foreground regions by taking into account dynamic objects such as people, cars, etc. or hard-to-reconstruct objects such as sky regions.

Sky Segmentation. Various algorithms have been proposed for segmenting the sky from the foreground using a single image. Some algorithms are not specific to skies but aim to perform per-pixel semantic segmentation by using FCNNs [LSD15, ZZP*17, ZSQ*17]. Most of these techniques work at a lower resolution and require a separate refinement stage. To this end, the initial segmentation results can be refined by using a dense conditional random field (CRF) model [KK11] or the GrabCut algorithm [RKB04]. However, as indicated by Mihail et al. [MWBJ16], the vast variation in the sky due to lighting, presence and types of clouds, the effect of the sun, flare, and other camera induced effects an entirely robust solution is difficult to obtain. There are also some algorithms that are designed to only separate the foreground from the sky region, for example to be used in ground robot navigation [SW13]. These algorithms are simpler and arguably more suitable for our purpose as we require a bimodal separation between the sky and the foreground, rather than a dense segmentation of the image into multiple categories.

Cloud Detection. Blue light scatters more in the atmosphere, which causes a high blue over red channel, B/R, ratio for the sky pixels compared to the cloud pixels. Therefore, the B/R channel ratio has been commonly used in literature to detect the cloud pixels in the sky [JH87, KJS91, PML*03]. There has been much research to improve the classification accuracy based on this method such as using $B-R$ or normalized B/R ratio [HMS10, LLY11, YLMY12, CNT*15]. Some other techniques such as partial least squares regression, bag-of-words, random forest, support vector machine, and Bayesian classification have also been tried to improve the cloud detection accuracy [YML*16, DLW17, HYCL17]. However, considering the sky lighting depends on many parameters such as sky haziness, turbidity, sun direction, and ground reflectance, no single solution produces accurate results on every cloudy image.

In summary, the analysis of the previous work reveals that while there are various image editing, relighting, and recoloring methods supported by various estimation and segmentation algorithms, no algorithm has hitherto been proposed to modify the position of the sun where the sun and clouds are directly visible by using only a single image and to simulate relighting and recoloring effects that are necessitated by it. Furthermore, despite producing appealing results, existing relighting methods such as [PGZ*19, YME*20, IZZE17, ASK*20] do not allow for fine-scale modifications of the solar position. Instead they are designed to change the entire illumination of the scene by replacing the sky with a new one. In contrast, our approach allows for both fine-scale modifications such as moving the sun by incremental amounts to a desired target position (Figure 1) as well as large-scale modifications such as adding, removing clouds, and introducing advanced lighting effects (Figure 18).

3. Algorithmic Overview

Our algorithm is composed of three stages: segmentation (Section 4), rendering (Section 5), and recoloring (Section 6). Our algorithm starts with a user click indicating the approximate position of the sun in an input picture. Using this input, we detect the sun pixels with a simple flood-fill algorithm, assuming that the sun is initially not occluded by clouds. It is also possible to add sun to an image that is devoid of it. The user presses another button to distinguish this case. This initial input also triggers a series of computations during which the sky, clouds, foreground, depth, and the

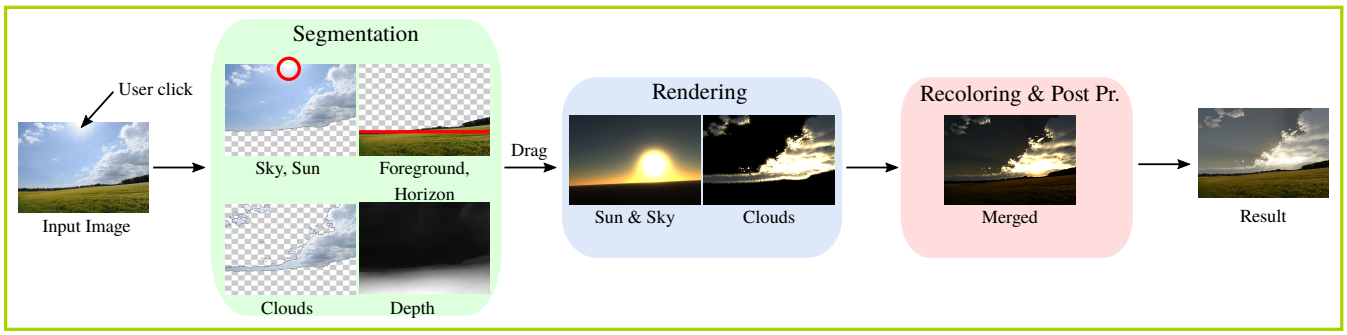


Figure 2: The main workflow of our algorithm: the input to our algorithm is a single image. With the user clicking on the sun position, we perform a series of computations to segment important information about the image. Then as the user drags the sun across the sky, atmospheric scattering and cloud rendering are simulated. Finally the results are merged, post-processed, and tone mapped for display purposes.

horizon line are detected along with the orientation of the camera that captured the picture. This stage of the processing, which we call preprocessing and segmentation, is the only stage that is not interactive and may take up to several seconds depending on the image resolution. The segmented data as well as the depth map is used in this stage to create advanced lighting effects.

Once the previous information is computed, the next stage, called rendering, begins with the user dragging the sun from its initial position to a different region in the sky. The user can move the sun anywhere in the sky in an interactive manner; the occlusion behind the clouds and the foreground are automatically computed by our algorithm. The rendering stage involves a computer-generated sky rendering based on physically-based precomputed atmospheric scattering and cloud rendering algorithms. We provide the horizon line and the camera parameters extracted in the previous stage to this algorithm, which in turn renders both the sky, clouds, and the sun in its new position. To achieve more plausible images, effects such as light shafts and shadows can be introduced during this stage by using the extracted depth map (Section 5.3).

The final stage involves merging the rendered sky with the existing one and recoloring the photograph. This stage involves several computations such as transferring the colors from the original sky to the rendered one to maintain consistency with the original photograph and recoloring of the foreground based on the updated sun position and current sky color. The details of each step of our algorithm are elaborated in the following sections. The overall workflow is depicted in Figure 2.

4. Preprocessing and Segmentation

4.1. Horizon Line Detection

This stage starts with detection of the horizon line. The horizon line is defined as a set of pixels, p, whose backprojection on the camera coordinate system is orthogonal to the gravity vector's representation in this system [WZJ16]:

$$p^T K_c^{-T} R_c [0,1,0]^T = 0, \quad (1)$$

where K_c represents the intrinsic camera matrix and R_c is the camera's orientation matrix. Despite the availability of a multitude of techniques [LGvGRM14, Tar09, XOH13], we decided to use a more recent deep learning based algorithm, which was trained on a large database involving natural images with labeled horizon lines [WZJ16]. This algorithm provides the location of the horizontal line using two parameters, namely (θ, ρ), where θ is the angle that it makes with the x-axis of the image and ρ represents the perpendicular distance from the image origin to the horizon line. Several horizon lines extracted by this algorithm for our test images are shown in Figure 3.

Figure 3: Several horizon lines extracted using Workman et al.'s method [WZJ16].

4.2. Estimating the Scene Depth

We use scene depth for casting the shadows of more distant objects on the objects that are closer to the camera. To estimate the scene depth, we use a deep learning based method recently proposed by Li and Snavely [LS18] for single view depth prediction. This method is trained on a large scale dataset called MegaDepth, which contains images with overlapping viewpoints that are available in photo collections found on the web. Two depth maps extracted using this method for our sample scenes are shown in Figure 4. As can be seen by these images, the method works robustly for estimating the depth of natural scenes.

4.3. Camera Parameters

There are several camera parameters that effect the visual appearance of the captured image. These include but are not limited to the camera's response curve, exposure time, aperture size, noise characteristics, the altitude at which the image is taken, orientation, focal length as well as the field-of-view. Although some studies allow for estimating some of these parameters from a single im-

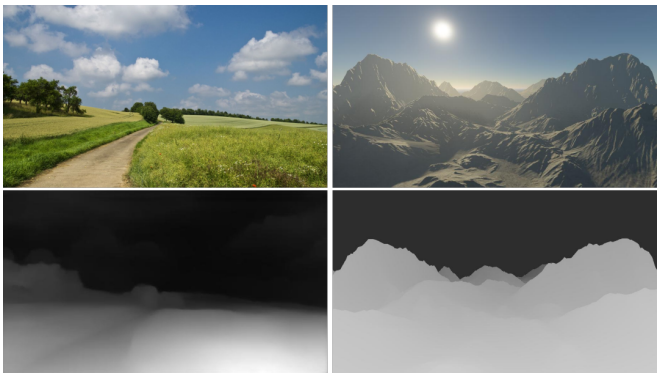
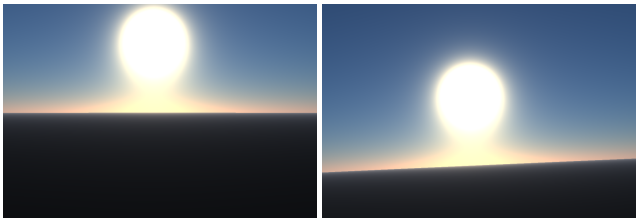


Figure 4: Two depth maps computed using Li and Snavely's single-view depth predictor [LS18].



(a) Initial sky rendering (b) Matching the horizon line

Figure 5: The camera is rotated such that the horizon line of the rendered sky (a) matches the horizon line of the input image (b).

age [LFSK06, WGZ*15, YZZ*17, LP17, LLC*20], estimating all of them reliably is a significantly difficult problem.

Given that our goal is to re-render the sky using a modified sun position, we note that accurate estimation of these parameters is not critical. The most important camera parameter is the camera's orientation, which should match the orientation of the real camera that captured the picture. We compute the camera orientation such that the horizon line of the rendered sky matches the horizon line computed from the image. To this end, we first rotate the camera around the z-axis such that the rendered and real horizon lines become parallel. We then rotate the camera around the x-axis to make the horizon lines collinear. A sample result depicting the rendered sky before and after this process is shown in Figure 5. In our current implementation, we do not use field-of-view (FOV) estimation for the camera. We experimentally set it to a fixed value of 80° vertically, which we found to yield plausible results.

4.4. Sky Segmentation

For segmenting the sky from the foreground, we use a sky segmentation algorithm that was originally developed for ground robot navigation [SW13] with some modifications. In this algorithm, the goal is to find a sky border $b_t(x)$ that separates the image I into two regions:

$$I(x,y) = \begin{cases} \text{sky (s)}, y < b_t(x), \\ \text{ground (g)}, y \geq b_t(x), \end{cases} \quad (2)$$

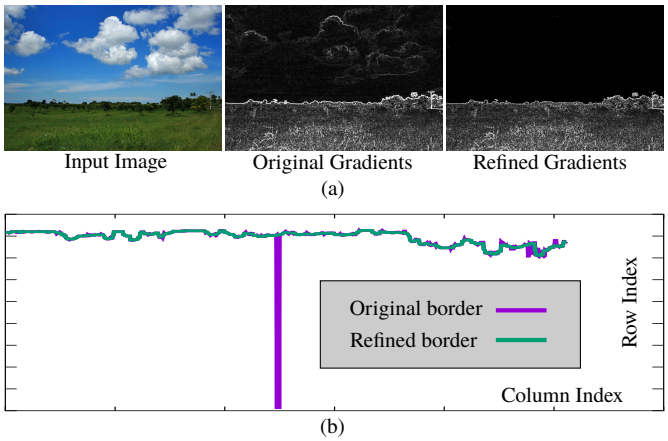


Figure 6: (a) By resetting the gradients caused by the clouds, we obtain a more reliable sky border. (b) Median filtering the border removes outliers further improving the skyline.

such that the separation maximizes the following energy function:

$$J_t = \frac{1}{\gamma|\Sigma_s| + |\Sigma_g| + \gamma|\lambda_1^s| + |\lambda_1^g|}, \quad (3)$$

where $|\Sigma_s|$ and $|\Sigma_g|$ represent the determinant of the covariance matrices of sky and ground pixels and λ_1's representing their largest eigenvalues. γ is a parameter that emphasizes the homogeneity in the sky region and a value of 2 is proposed in the original algorithm. Maximizing J_t corresponds to minimizing the intra-class variance of sky and ground pixels.

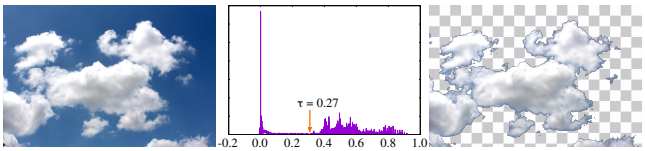
The border function, $b_t(x)$, is found by detecting the first image gradient whose magnitude is greater than t. To this end, for each column x, the gradient image ΔI is traversed from top to bottom to find the first row y such that $\Delta I(x,y) > t$. The border value is then set to this value: $b_t(x) = y$. In the original algorithm, the threshold is varied in range $t \in [5, 600]$ with a step size of 5. The threshold value that yields the largest energy is taken to be the optimal threshold, and the corresponding border function is used as the sky border:

$$b_{\text{opt}} = \arg\max_{b_t} J_t. \quad (4)$$

We observed although this algorithm works well for clear skies, the presence of clouds often create strong gradients resulting in an incorrect sky segmentation. Furthermore, because the border at each pixel is computed independent of the neighboring pixels sudden variations in the border values are possible. To counteract these two problems, first we reset the image gradients to zero for which the blue channel value exceeds a threshold. Secondly, we apply median filtering on the sky border to remove outliers. The effect of these refinements are demonstrated in Figure 6 for a sample picture.

4.5. Cloud Detection

We perform cloud detection on the sky pixels after the sky has been segmented. After experimenting with several algorithms, we found the HYTA cloud segmentation algorithm [LLY11] to work sufficiently well for our purposes. It is fast and has a single intuitive



(a) Input image (b) Normalized B/R histogram (c) Cloud mask

Figure 7: This input image (a) has a bimodal normalized B/R histogram (b). As the standard deviation of the normalized ratio image was greater $T_s = 0.03$, MCE thresholding was applied to obtain the cloud mask in (c).

parameter that can be controlled by the user if the default segmentation results are not satisfactory. This algorithm is based on computing the normalized blue over red ratio,

$$\lambda = \frac{B}{R}, \qquad \lambda_N = \frac{\lambda - 1}{\lambda + 1}, \tag{5}$$

such that sky pixels with $\lambda_N < \tau$ are assumed to represent clouds. Note that in the above formulation, the red component's value is increased by one of it is equal to zero and therefore $\lambda_N \in [-1, 254/256]$. Furthermore, the algorithm operates on display-encoded (i.e. gamma-corrected) pixel values. The selection of the threshold value follows a binary decision schema. If the standard deviation of the λ_N image is less than or equal to $T_s = 0.03$, the histogram is assumed to be unimodal and a fixed threshold of $\tau = 0.250$ is used. On the other hand, if the standard deviation is larger, the histogram is assumed to be bimodal. In this case, the threshold value is found such that the cross entropy between the original image and the segmented image is minimized, an approach known as minimum cross entropy (MCE) thresholding [LL93].

5. Rendering

The rendering process is comprised of two stages. The first stage involves the rendering of the atmosphere with the sun in its new position. This stage is intertwined with a second stage, which involves rendering of the clouds – if clouds are present in the input image. The reason for the interplay between these two stages is that the lighting of the clouds depend on the sun position, but at the same time presence of clouds induce certain effects such as light shafts.

5.1. Precomputed Atmospheric Scattering

The color of the sky depends on the scattering of the sunlight due to microscopic particles and molecules floating in the sky. Two types of scattering phenomena are distinguished: Rayleigh and Mie scattering [RKAJ08]. Rayleigh scattering is wavelength dependent. The upper atmosphere contains relatively smaller particles and these cause blue light to be scattered and reach our eyes. During sunset and sunrise, the optical path that must be traversed by sunlight becomes larger. This causes the blue light to get scattered to almost extinction and the sky takes a red/orange hue. Mie scattering is caused by larger aerosol particles and it is responsible for the overall lightening and darkening of the sky based on the sun position.

Mathematically, the light arriving at a viewer at an altitude of $h(x)$ can be modeled as:

$$L = \tau L_d + L_{\text{ins}}, \tag{6}$$

where L_d is the direct light, L_{ins} is the in-scattered light, and τ is the extinction factor. Both τ and L_{ins} depend on the optical path, s, of the sunlight that travels within the atmosphere:

$$\tau(s) = \exp\left(-\int_0^s \beta_m(h(x))dx\right) \exp\left(-\int_0^s \beta_r(h(x))dx\right), \tag{7}$$

where the β coefficients represent the Mie and Rayleigh scattering factors as a function of the altitude at which the scattering event occurs. Scattering also has an angular dependency in addition to altitude:

$$\beta(\omega_i, \omega_o, h(x)) = \beta^0(\omega_i, \omega_o)\beta(h(x)), \tag{8}$$

where the angular and altitude dependencies are decoupled. The angular dependency, $\beta^0(\omega_i, \omega_o)$, is often termed as the phase function and it models how much of the light coming from direction ω_i is scattered toward the direction ω_o. The amount of light scattered at point x along ω_o is then given by:

$$L_{\text{ins}}(x) = \beta_m(h(x)) \int_\Omega L_s(w)\beta_m^0(\omega_i, \omega_o)dw + \tag{9}$$

$$\beta_r(h(x)) \int_\Omega L_s(w)\beta_r^0(\omega_i, \omega_o)dw, \tag{10}$$

where L_s represents the inscattered light at that point. Integrating this over the full optical path yields:

$$L_{\text{ins}} = \int_0^s L_{\text{ins}}(x)\tau(s)dx. \tag{11}$$

Precomputed atmospheric rendering techniques evaluate these integrals using ray marching techniques. The computed results are stored in lookup tables for different observer altitudes and all possible sunlight and viewing directions. It then becomes possible to use these tables as texture maps, and sample from them in real-time based on the current sun position and view direction. Several methods exist in the literature. We used Elek's approach [Ele09][†], which is a simplification of Bruneton and Neyret's [BN08] work.

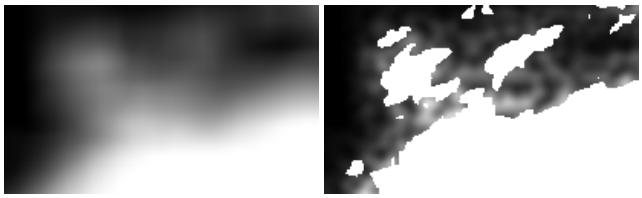
5.2. Cloud Shading

For realistic cloud shading we used the method described by Schneider [Sch16], which is originally designed for the game "Horizon: Zero Dawn". This method proposes a volumetric cloud modeling using fractal Brownian motion (FBM). FBM involves computing the sum of series of a noise function with increasing frequency and decreasing amplitude. Different from the earlier approaches, it combines two noise types namely Perlin [Per85] and Worley [Wor96] to create more realistic cloud shapes.

At the core of this algorithm lies a weather texture, comprised of $\mathbf{C_w}$, $\mathbf{R_w}$, and $\mathbf{T_w}$ vectors. These vectors, which are stored as the color channels of a three-channel texture image in ($\mathbf{C_w}$, $\mathbf{R_w}$, $\mathbf{T_w}$) order, control the following properties of the rendered clouds:

[†] We used Michal Skalsky's implementation available at https://github.com/SlightlyMad/AtmosphericScattering.

Figure 8: A sample weather texture shown on the left, together with rendered stratus (middle) and cumulus (right) clouds obtained by changing the $\mathbf{T_w}$ value.



(a) Gaussian blur (b) Our result

Figure 9: Softening the cloud boundaries is important for realistic blending between clouds and sky. Naïve Gaussian blur causes over-blurring (a), whereas our approach preserves the overall cloud shape while producing a semi-random and smooth transition at the border (b).

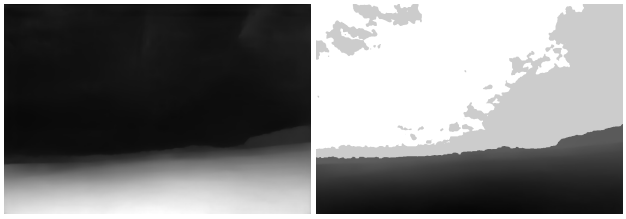
$\mathbf{C_w}$: Percentage of cloud coverage in the sky
$\mathbf{R_w}$: Probability of rain
$\mathbf{T_w}$: Cloud type

Note that these are per-pixel maps. In our case, we already know the location of the clouds therefore we set the $\mathbf{C_w}$ channel to one for all pixels. This causes clouds to be rendered in the entire sky. We then apply a mask to select the regions in which the original image contains clouds. For realistic appearance, we use a soft cloud mask as explained in the next section. We set the $\mathbf{R_w}$ channel to zero as the scenes we deal with are typically not rainy. We leave the cloud type $\mathbf{T_w}$ as a user-parameter for which the value of 0 indicates stratus, 0.5 stratocumulus, and 1 cumulus clouds. A sample weather texture is shown in Figure 8 together with rendered stratus and cumulus clouds obtained by changing the $\mathbf{T_w}$ from 0 to 1. The sampling of cloud density is based on ray marching, which is typical for volumetric rendering algorithms[‡].

5.2.1. Computation of Soft-cloud Mask

Using a binary cloud mask leads to an unrealistic transition between the sky and cloud regions. In contrast, simply blurring the cloud mask produces clouds that are too smooth and unrealistic. To compute a soft-cloud mask, we first blur the original cloud mask using a Gaussian kernel with a large size. We then subtract the original mask from the result to extract only the cloud boundaries. We then create a Perlin noise texture and only select the parts that correspond to these boundaries. We then multiply the boundary texture with the Perlin noise texture to semi-randomly modulate the transition in the boundary regions. Finally, we recompose the cloud mask texture by adding the modulated boundary texture. We show the difference of this result from simply applying a Gaussian blur to the cloud mask in Figure 9.

[‡] We used Ben Hopkins's implementation available at https://github.com/kode80/kode80CloudsUnity3D.



(a) Initial depth estimation (b) Refined and inverted depth map

Figure 10: We recompute the input depth map (left) so that the most distant object is the sun, followed by the sky, clouds, and the foreground. Note that depth values are inverted in the final depth map (right) so that more distant pixels have higher values.

5.3. Creating Lighting Effects

The presence of the per-pixel depth information allows us to render advanced effects such as light shafts and shadows. In Section 4.2, we explained our method of depth estimation from a single image. Although this method works well for foreground objects, it fails to properly capture the depth of clouds. Despite their precise depth not being important, clouds need to be positioned between the sun and the foreground in order to be able to cast shadows on the foreground. To achieve this, we define start and end values for both the foreground and the clouds. We ensure that the start value of the clouds exceeds the end value of the foreground. This makes the foreground pixels closest to the viewer, followed by the clouds, sky, and then the sun. Figure 10 shows the input and the final depth maps using this technique. Note that the final depth map is inverted so that distant pixels have higher values.

With the depth map available clouds and distant objects can be made to cast shadows on the foreground using standard shadow mapping techniques. If a pixel is under-shadow[§], we march a ray from the pixel's reconstructed 3D position in the sun direction, attenuating the light intensity based on the sampled cloud density

[§] Whether a pixel is under shadow or not can be understood by comparing the depth of this pixel with the value stored in the depth buffer, both with respect to the sun position.

along this ray. The limitation of our algorithm regarding the shadows is that as we cannot remove the existing shadows, the user would need to be careful to avoid introducing conflicting cues. This could be problematic if the input image already has strong directional shadows that are cast in different directions than the newly introduced ones.

It is also possible to add light shafts during the rendering of the atmosphere. To achieve this, ray marching is performed in two directions. For every pixel, we first march along the viewing ray and at every step we march along the sun direction. If clouds are encountered the illumination is attenuated by using cloud density.

6. Recoloring

Rendering part of our algorithm deals with recomputing the colors of the sky commensurate with the updated sun position. In the recoloring phase, we first update the statistical distribution of these colors to match it with the distribution in the original image. Secondly, we update the colorfulness and contrast of the foreground to match it with the final sky color.

6.1. Color Transfer

Color transfer algorithms aim to impart the color distribution of a target photograph into a source one while preserving the content of the latter [FPC*14]. In our case, we do not know the colorimetric and radiometric properties of the camera that captured the original photograph. These properties may include the response curve, white balance settings, and sensor sensitivities of the camera. When we render the sky using a physically based atmospheric scattering algorithm, it is implausible to expect that the colors of the rendered sky match to those of the original photograph – even if the sun is at the same position. To prevent this sudden change that would happen as soon as we switch from the input sky to the rendered one, we decided to use Reinhard et al.'s [RAGS01] color transfer algorithm, which suffices for our purposes.

This algorithm aims to match the statistical distribution (mean and standard deviation) of colors between two images after transforming them to the $L\alpha\beta$ color space [RCC98]. In our case, by using our sky mask, we compute and transfer the distributions only for the sky pixels. However, applying color transfer irrespective of the sun's new position would create incorrect colors; we want the sky to *change* color based on the sun's distance to the horizon. To this end, we modulate the degree of this transfer based on the Euclidean distance between the sun's original and current position. Assuming that C_s and C_r represent the original and rendered sky colors, the final sky colors C'_s are computed as:

$$C'_s = tC_r + (1-t)f(C_s, C_r), \tag{12}$$

where f is the color transfer function and $t \in [0,1]$ is a smoothly varied distance measure between the original and updated sun position.

6.2. Foreground Colors

Until now, we focused on computing the sky colors. For a realistic result, however, the foreground colors must be duly updated. Let $\bar{C}_s$ and $\bar{C}_f$ represent the mean colors of the sky and foreground regions in the original picture, respectively. Denote their per-component ratio by $Q = \bar{C}_f / \bar{C}_s$. After updating the sky color as explained in Section 6.1, we recompute the mean foreground color to preserve this ratio, i.e. $\bar{C}'_f = Q \times \bar{C}'_s$ where $\times$ represents per-component multiplication. To achieve this, we scale each foreground pixel by the factor of $\bar{C}'_f / \bar{C}_f$:

$$C'_f = \left(\frac{\bar{C}'_f}{\bar{C}_f} \times C_f \right). \tag{13}$$

This produces the desired darkening effect as the sun approaches the horizon.

Another key observation about outdoor lighting is that the direct sun contribution is much more powerful than the scattered sky colors and the transmitted sun rays through the clouds. As a result, when sun is occluded by clouds or sets toward the horizon, the overall color saturation of foreground objects decrease, an phenomenon known as the Hunt effect [Fai13]. To model this phenomena, we introduce a new variable $s \in [0,1]$ which is a normalized cosine multiplied power of the visible sun pixels. Then, we employ a colorfulness modulation formula inspired by tone mapping [Sch95] to reduce the saturation of the final foreground colors:

$$C''_f = \left(\frac{C'_f}{L'_f} \right)^s L'_f, \tag{14}$$

where L'_f is the luminance computed from the color C'_f after Equation 13. As the sun is occluded with clouds or approaches the horizon, $s < 1$ will result in reduced colorfulness.

6.3. Postprocessing

After the recoloring stage, we merge the foreground layer with the sky layer using Laplacian blending [BA83] to achieve a smooth transition between the two layers. Furthermore, the rendering of the sky and clouds take place in the linear high dynamic range (HDR) domain. To make the input image compatible with linear lighting, it is presented as an sRGB texture so that it will be linearized during texture fetches. The final HDR result that is obtained after Laplacian blending is tone mapped for display purposes. To this end, any shader-friendly tone mapping operator (TMO) can be used. We used the photographic TMO [RSSF02] as it can be easily implemented in the pixel shader, and the quality of the results suffices for our purposes.

7. Results

We present our results under several conditions that pose different degrees of challenges for our algorithm. First, we share results for clear-sky images with and without sun. These images are easier to process due to lack of clouds. Second, we demonstrate results for images with cloudy skies. Third, we present and discuss results for advanced lighting effects that include shadows, sun-shafts, and sun-flare.

(a) Input (b) Sky mask (c) Depth (d) Sun at original position (e) Sunset

Figure 11: A sample result for an input image with clear sky. The horizon line is overlayed in red in (a) for visualization purposes. Two outputs of our algorithm at different sun altitudes are shown in (d) and (e).

(a) Input image (b) After minimal sun movement

(c) Towards sunset (d) Sunset

Figure 12: Input image (top-left) and relighted outputs after incremental sun movements.

(a) Input image (b) Initial rendering result

(c) Sunset toward open sky (d) Sunset behind the clouds

Figure 13: The input photograph is on the left and results with different sun positions are on the right. Note the realistic shading of the clouds with respect to different sun positions.

(a) Input image (b) Our rendering result

Figure 14: Detection and rendering of thin clouds pose challenges for our algorithm. The input image is on the left and our result is on the right.

7.1. Clear Sky

This setting represents the simpler case for our algorithm due to the lack of clouds. We show a sample result in Figure 11. Here (a) shows the input image with the horizon line in red for visualization purposes. The sky and depth masks are shown in (b) and (c). Two outputs of our algorithm for different sun elevation angles are shown in (d) and (e). For this example, cloud masks were set to all zeros.

We share another result in Figure 12. Here, the top-left image is the original photograph and other images are our results for different sun positions. It can be seen that as the sun sets towards the horizon, the sky as well as the foreground pixels take an orange tint. This result also showcases a limitation of our algorithm. In the real-world, the sea would take a more orange color due to its highly reflective surface. We cannot account for this effect as we do not further segment foreground object into different categories.

7.2. Cloudy Sky

This category represents the more difficult case for our algorithm. It requires an accurate segmentation of cloud pixels as well as realistic rendering of the sky and shading of the clouds. We share a sample result for this case in Figure 13. It can be seen in this figure that the sky and clouds are realistically shaded and the foreground colors are properly updated for different sun positions.

The most challenging part of cloudy skies is the difficulty of matching the appearance of clouds to those in the original picture. In our experiments we found it to be the most difficult to match the appearance of thin clouds. An example is shown in Figure 14. In this example, the rendered clouds fail to capture the appearance of clouds in the original picture.

(a) Input (b) Result without sun (c) Result with sun

Figure 15: The clouds in the input photograph (left) are removed in the rendering results. The sun may be left outside (middle) or inside (right) the scene.

(a) Input (b) Cloud mask (c) Result

Figure 16: The clear sky in the input image (a) can be turned into a cloudy one (c) by adding clouds using a cloud mask (b).

7.3. Cloud Removal and Addition

Our algorithm also allows the removal and addition of clouds. Removal of clouds is achieved by rendering only the sky and leaving out the clouds. The sun may be positioned inside or outside of the photograph as shown in Figure 15. Inserting clouds can be achieved by first setting the desired parameters the clouds (Section 5.2) and rendering them according to these parameters. A custom cloud mask can be used to specify where the clouds should be present in the photograph. This is illustrated in Figure 16.

7.4. Advanced Lighting Effects

As explained in Section 5.3, previously estimated depth map can be used to add plausible effects to the output image. Adding shadows is one of them and, Figure 17 (a) and (b) show the difference between disabling and enabling shadows on the foreground. Another effect which can be achieved is to add light shafts that creates dark and light patterns on the sky pixels as can be seen in Figure 17 (c).

As a detailed illustration of both the merits and limitations of our algorithm, we use an image captured by our own digital camera (Figure 18). The top-left image (a) represents the input with the horizon line overlayed in red. The depth map is shown to the right (b). In the top-right corner (c), we can see the gradient map from which the sky mask is computed. The sky mask shown in (d) fails to capture some of the fine details of the tree branches – these parts will be replaced with the rendered sky. (e) shows the initial result that is computed without color transfer. Note the color difference of the sky between (a) and (e). As shown in (f), the sky as well as the foreground becomes closer to the input image after applying color transfer. The bottom row shows the sun moved closer to the horizon and with various effects enabled. Shadows are shown in (g). Note that there is some inconsistency between the original shadows and the rendered ones. Light shafts on the sky can be observed when the sun sets behind the building (h). Finally, artistic effects such as light shafts can be added using the shadow map texture and the orientation between the sun position and the camera (i).

Operation	Time (ms)
Precomputed atmospheric scattering	0.57
Volumetric cloud rendering	2.05
Light shaft computation	1.54
Shadow map computation	3.01
Mean sky color computation	123.34
Color transfer	0.41
Laplacian blending	1.01
Tone mapping	1.17
Other	4.21
Total	137.31

Table 1: Time taken by the different stages of the rendering and recoloring parts of our algorithm for an input image size of 1210×907. Note that the majority of the time is spent on mean color computation for color transfer, which is not a GPU-friendly operation.

7.5. Time-lapse Videos

Finally, using our algorithm we can create time-lapse videos by varying the sun position at every frame. We invite the readers to view several examples in electronic supplementary materials.

7.6. Run-time Performance

The run-time performance of our algorithm is high enough to allow real-time frame rates. In our tests, we used a system that has an Intel Core i7 CPU running at 3.60 GHz and Nvidia GTX 1050 Ti GPU. The implementation was made on Unity3D version 2019.4.9f1. Table 1 shows the time taken by different parts of our algorithm to render a single frame of size 1210×907 (Figure 18). As can be seen from this table, the majority of the time is spent on computing the mean sky color to be used in color transfer. Other parts of our algorithm take only a few milliseconds each. The implementations of all of these operations use either compute or pixel shaders.

7.7. Comparison

Our algorithm allows both small and large modifications of the sun position to simulate the appearance of the same scene in different times of the day. While we are not aware of any direct alternatives to our work, in this section we compare our results with three prominent works for day to night translation [IZZE17], style transfer [ASK*20], and scene relighting [YME*20]. As can be seen in Figure 19, despite producing appealing results, these algorithms do not allow for the type of modifications that are possible with our algorithm. In particular, the image in (c) appears too dark and the sun is not visible. Moreover, it has nearly half the resolution of the input image as the method accepts fixed size inputs. The image in (d) looks more appealing but note that new clouds are introduced and there is a false sun image at the original position of the sun. The image in (e) fails to convey the sunset style of the style source image. Note that for both (d) and (e) if a different style image was used as input entirely different outputs would be obtained. In contrast, our results for the same scene shown in (f) as well as in Figures 13 and 17 suggest that more finely controlled modifications that adhere to the original scene content can be achieved.

(a) Initial result (b) Shadows (c) Light shafts

Figure 17: Shadows and light shafts can be added to improve realism.

(a) Input (b) Depth Map (c) Gradient Map

(d) Sky Mask (e) Without Color Transfer (f) With Color Transfer

(g) Shadows (h) Shadows + Light Shafts (i) Shadows + Light Shafts + Lens Flare

Figure 18: A detailed example illustrating various stages of our algorithm. See the text for details.

(a) Input (b) Style source image (c) Isola et al.

(d) Anokhin et al. (e) Yu et al. (f) Our result

Figure 19: (a) The input image. (b) Style source image. (c) The result of pix2pix algorithm by Isola et al. [IZZE17] using the trained model of day2night. (d) The result of Anokhin et al. [ASK*20] using the style source image. (e) The result of Yu et al. [YME*20] using the style source image. Our results for the same input image are shown in Figures 13 and 17. The aspect ratio of (c) is different due to the model's limitation. The style source image was kindly shared at https://www.pexels.com/photo/brown-and-green-grass-field-during-sunset-1237119.

Finally, we compare our results against a ground-truth time-lapse sequence. In Figure 20, the top-left image in (a) shows a real photograph from a time-lapse sequence with sun having risen above what is known as *Ayers Rock* in Uluru region of Australia. The top-right image in (b) shows an earlier frame from the same sequence. This is a typical application scenario of our algorithm: the photographer misses the moment of a sunrise or sunset and wants to apply image editing to recreate that missed moment. Our results are shared in (c), (d), and (e). In (c), we render a clear sky to remain faithful to the original sky in the input photograph. In (d) and (e), we add different amounts of cloud coverage for artistic effects. While our results are not identical to the ground-truth, they appear to create plausible depictions of the input environment for different sun positions. In particular, a viewer without access to the ground-truth image may not notice the difference between our results and real-photographs of the scene.

7.8. Limitations

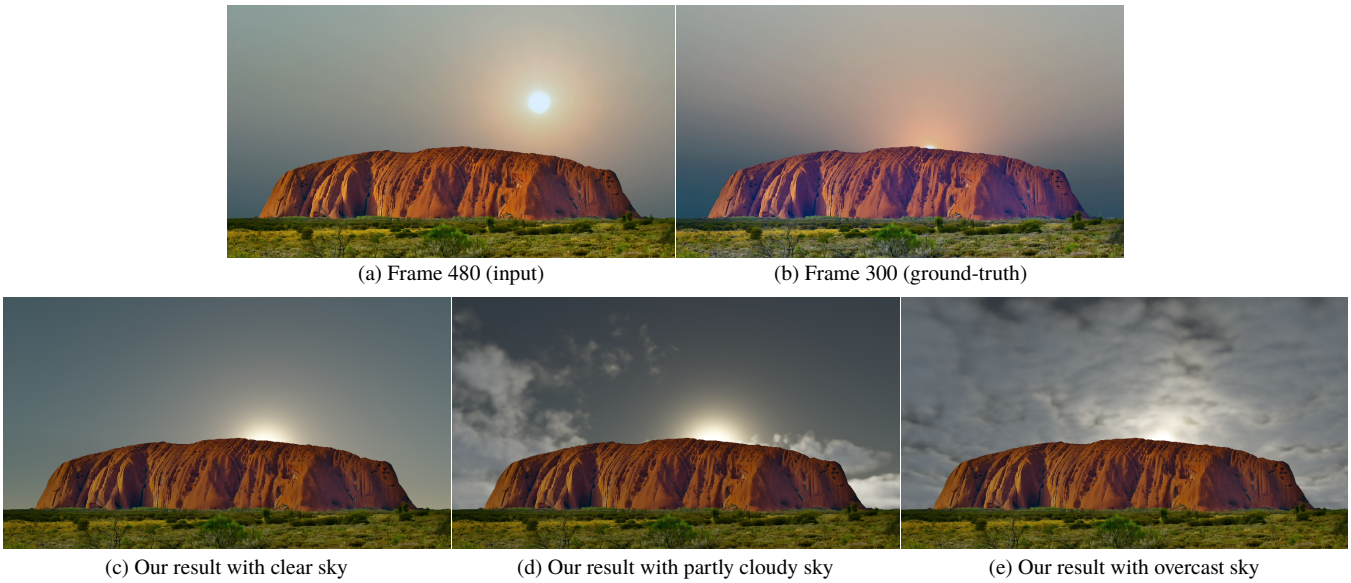
Since we do not perform a dense semantic segmentation of the foreground into multiple object categories, we cannot distinguish shading differences between different scene elements. For example, we cannot add specular highlights to water surfaces that are consistent with the updated sun position, nor can we make such a surface attain a dominantly orange hue akin to the sky. State-of-the-art semantic segmentation models [ZSQ*17, ZZP*19, RBK21, LLC*21] may be employed to enable these effects. These algorithms may also prove to be beneficial for segmenting of the sky from the foreground as well as segmenting of the clouds from the sky.

As another limitation of our approach, we could not find a plausible way to drive the rendered cloud density from the density of the real clouds in the input image. This poses two challenges where one first needs to estimate the original cloud density and then estimate the parameters of the noise function so that the original density constraints are satisfied. This is a challenging problem and some solutions are outlined in a relevant survey of procedural noise functions [LLC*10].

Finally, as stated in Section 5.3, our algorithm cannot remove the existing shadows before introducing new ones. Although this problem can theoretically be tackled with shadow detection and removal algorithms [SSL12], a fully automatic solution that can work under large disconnected shadow regions with arbitrary shapes appears to be elusive [HZYG17].

8. Discussion and Future Work

In this paper, we attempt to address a difficult image editing problem, which is the modification of the sun position using a single input image. To solve this problem, we need to solve multiple subproblems such as horizon line, sky, and cloud detection, depth estimation, precomputed atmospheric scattering, cloud shading, and recoloring. As our algorithm depends on all of these techniques, it also borrows the union of their parameters. However, in practice we



(a) Frame 480 (input) (b) Frame 300 (ground-truth)

(c) Our result with clear sky (d) Our result with partly cloudy sky (e) Our result with overcast sky

Figure 20: (a) The input frame. (b) An earlier ground-truth frame. (c) Our result by moving the sun from the input frame location to the ground-truth frame location. (d, e) Adding varying amounts of cloud coverage for artistic effects. The source time-lapse sequence was kindly shared at https://www.videezy.com/abstract/45554-uluru-sunrise-australia.

found most of the algorithms to work with their default parameters especially those based on deep learning such as the horizon line and single image depth estimation.

As for sky segmentation, the only parameter that requires tweaking is the gradient reset threshold explained in Section 4.4. Usually, a default value of 128 works together with a median filtering size of 3 pixels. Reducing this threshold clears more gradients pushing the sky border lower in the vertical direction. Increasing it has the opposite effect. The size of the median filter can be increased depending on the width of the outliers (see Figure 6). However, sky segmentation takes only a few seconds and therefore multiple settings can be quickly tried.

Cloud detection is another part of our algorithm that may require parameter tweaking. Automatically computed threshold values do not always work, incorrectly labeling some sky pixels as clouds or cloud pixels as sky. In this case, we set a user defined threshold value. Increasing the threshold parameter marks more pixels as clouds and decreasing it has the opposite effect (see Section 4.5).

As for the rendering of the sky and clouds, there are multiple parameters that may affect the quality of the results. These include but are not limited to sunlight intensity, Rayleigh and Mie scattering coefficients, phase function, cloud distance, the proportion of ambient to direct illumination, coverage texture, number of samples in ray marching, shadow map resolution, tone mapping parameters, etc. Because of a large number of parameters with confounding effects, we found it best to use the physically-based default values.

Given these issues, it usually takes less than five minutes to create satisfactory results starting with a single input image. We believe this is still significantly faster and yields more realistic results compared to those that can be obtained with existing image editing tools. Furthermore, because the rendering and recoloring parts of our algorithm are real-time, placing the sun to the desired part of the photograph takes a negligible amount of time.

There are several future research directions that can improve the realism of the results. First, we do not remove shadows on the foreground. This may create conflicting cues when the sun is moved to a new position as the new shadows may be cast in a different direction than the existing ones. Secondly, we only approximate the foreground lighting. It should be possible to use the gradient of the depth image to create view space normals and then use this information for more accurate foreground lighting. Finally, the current algorithm cannot deal with surfaces with different reflective properties in the foreground. For example, specular highlights on the water surface will remain at the same position even if the sun is moved. Accounting for these effects requires more accurate and detailed partitioning of the foreground pixels into different object categories.

Acknowledgements

We thank Michal Skalsky and Ben Hopkins for making their precomputed atmospheric scattering and cloud rendering implementations publicly available. Without their implementations as starting points, it would have been a daunting task to implement these algorithms from scratch. The same goes for Zhengqi Li and Noah Snavely for “MegaDepth” and Scott Workman, Menghua Zhai, and Nathan Jacobs for “Horizon Lines in the Wild” implementations. This work was supported in part by GEBIP 2018 Award of the Turkish Academy of Sciences to E. Erdem, BAGEP 2021 Award of the Science Academy to A. Erdem.

References

[ASK*20] ANOKHIN I., SOLOVEV P., KORZHENKOV D., KHARLAMOV A., KHAKHULIN T., SILVESTROV A., NIKOLENKO S., LEMPITSKY V., STERKIN G.: High-resolution daytime translation without domain labels. In *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), pp. 7488–7497. 3, 10, 12

[BA83] BURT P. J., ADELSON E. H.: A multiresolution spline with application to image mosaics. *ACM Transactions on Graphics (TOG) 2*, 4 (1983), 217–236. 8

[BBS14] BELL S., BALA K., SNAVELY N.: Intrinsic images in the wild. *ACM Trans. on Graphics (SIGGRAPH) 33*, 4 (2014). 2

[BG17] BALCI H., GÜDÜKBAY U.: Sun position estimation and tracking for virtual object placement in time-lapse videos. *Signal, Image and Video Processing 11*, 5 (Jul 2017), 817–824. 2

[BN08] BRUNETON E., NEYRET F.: Precomputed atmospheric scattering. In *Computer Graphics Forum* (2008), vol. 27, Wiley Online Library, pp. 1079–1086. 6

[CAT06] CHANDRA K., ADABALA N., TOYAMA K.: Aerial image relighting: simulating time of day variations. In *Advances in Computer Graphics*. Springer, 2006, pp. 594–605. 2

[CCC20] CHENG C.-C., CHEN H.-Y., CHIU W.-C.: Time flies: Animating a still image with time-lapse video as reference. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2020). 2

[CNT*15] CHAUVIN R., NOU J., THIL S., TRAORE A., GRIEU S.: Cloud detection methodology based on a sky-imaging system. *Energy Procedia 69* (2015), 1970–1980. 3

[DLW17] DEV S., LEE Y. H., WINKLER S.: Color-based segmentation of sky/cloud images from ground-based cameras. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 10*, 1 (2017), 231–242. 3

[ED04] EISEMANN E., DURAND F.: Flash photography enhancement via intrinsic relighting. In *ACM transactions on graphics (TOG)* (2004), vol. 23, ACM, pp. 673–678. 2

[Ele09] ELEK O.: Rendering parametrizable planetary atmospheres with multiple scattering in real-time. In *Proceedings of the Central European Seminar on Computer Graphics* (2009), Citeseer. 1, 6

[EPF14] EIGEN D., PUHRSCH C., FERGUS R.: Depth map prediction from a single image using a multi-scale deep network. In *Proc. NIPS* (2014). 3

[Fai13] FAIRCHILD M. D.: *Color appearance models*. John Wiley & Sons, 2013. 8

[FHL*09] FARBMAN Z., HOFFER G., LIPMAN Y., COHEN-OR D., LISCHINSKI D.: Coordinates for instant image cloning. *ACM Transactions on Graphics (TOG) 28*, 3 (2009), 67. 2

[FPC*14] FARIDUL H. S., POULI T., CHAMARET C., STAUDER J., TRÉMEAU A., REINHARD E., ET AL.: A survey of color mapping and its applications. *Eurographics (State of the Art Reports) 3* (2014), 2. 8

[Gei12] GEIGER A.: Are we ready for autonomous driving? the kitti vision benchmark suite. In *Proc. CVPR* (2012). 3

[HMS10] HEINLE A., MACKE A., SRIVASTAV A.: Automatic cloud classification of whole sky images. *Atmospheric Measurement Tech. 3*, 3 (2010), 557–567. 3

[HYCL17] HSU-YUNG C., CHIH-LUNG L.: Cloud detection in all-sky images via multi-scale neighborhood features and multiple supervised learning techniques. *Atmospheric Measurement Tech. 10*, 1 (2017), 199. 3

[HZYG17] HE K., ZHEN R., YAN J., GE Y.: Single-image shadow removal using 3D intensity surface modeling. *IEEE Transactions on Image Processing 26*, 12 (2017), 6046–6060. doi:10.1109/TIP.2017.2751142. 12

[IZZE17] ISOLA P., ZHU J.-Y., ZHOU T., EFROS A. A.: Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017), pp. 1125–1134. 3, 10, 12

[JCW09] JESCHKE S., CLINE D., WONKA P.: A GPU Laplacian solver for diffusion curves and poisson image editing. In *ACM Transactions on Graphics (TOG)* (2009), vol. 28, ACM, p. 116. 2

[JH87] JOHNSON R., HERING W. S.: Automated cloud cover measurements with a solid-state imaging system. In *Proceedings of the Cloud Impacts on DOD Operations and Systems?1987, Workshop* (1987), pp. 59–69. 3

[KAEE19] KARACAN L., AKATA Z., ERDEM A., ERDEM E.: Manipulating attributes of natural scenes via hallucination. *ACM Transactions on Graphics 39*, 1 (Nov. 2019). 3

[KJS91] KOEHLER T., JOHNSON R., SHIELDS J.: Status of the whole sky imager database. In *Proceedings of the Cloud Impacts on DOD Operations and Systems, 1991 Conference* (1991), pp. 77–80. 3

[KK11] KRÄHENBÜHL P., KOLTUN V.: Efficient inference in fully connected CRFs with Gaussian edge potentials. In *Advances in neural information processing systems* (2011), pp. 109–117. 3

[KRFB06] KHAN E. A., REINHARD E., FLEMING R. W., BÜLTHOFF H. H.: Image-based material editing. *ACM Transactions on Graphics (TOG) 25*, 3 (2006), 654–663. 2

[LFSK06] LIU C., FREEMAN W. T., SZELISKI R., KANG S. B.: Noise estimation from a single image. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)* (2006), vol. 1, IEEE, pp. 901–908. 5

[LFUS06] LISCHINSKI D., FARBMAN Z., UYTTENDAELE M., SZELISKI R.: Interactive local adjustment of tonal values. In *ACM Transactions on Graphics (TOG)* (2006), vol. 25, ACM, pp. 646–653. 2

[LGvGRM14] LEZAMA J., GROMPONE VON GIOI R., RANDALL G., MOREL J.-M.: Finding vanishing points via point alignments in image primal and dual domains. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2014), pp. 509–515. 4

[LGZ*20] LIU A., GINOSAR S., ZHOU T., EFROS A. A., SNAVELY N.: Learning to factorize and relight a city. In *ECCV* (2020). 3

[LL93] LI C. H., LEE C.: Minimum cross entropy thresholding. *Pattern recognition 26*, 4 (1993), 617–625. 6

[LLC*10] LAGAE A., LEFEBVRE S., COOK R., DEROSE T., DRETTAKIS G., EBERT D. S., LEWIS J. P., PERLIN K., ZWICKER M.: A survey of procedural noise functions. In *Computer Graphics Forum* (2010), vol. 29, Wiley Online Library, pp. 2579–2600. 12

[LLC*20] LIU Y.-L., LAI W.-S., CHEN Y.-S., KAO Y.-L., YANG M.-H., CHUANG Y.-Y., HUANG J.-B.: Single-image HDR reconstruction by learning to reverse the camera pipeline. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2020). 5

[LLC*21] LIU Z., LIN Y., CAO Y., HU H., WEI Y., ZHANG Z., LIN S., GUO B.: Swin transformer: Hierarchical vision transformer using shifted windows. *arXiv preprint arXiv:2103.14030* (2021). 12

[LLY11] LI Q., LU W., YANG J.: A hybrid thresholding algorithm for cloud detection on ground-based color images. *Journal of atmospheric and oceanic technology 28*, 10 (2011), 1286–1296. 3, 5

[LP17] LI H., PEERS P.: CRF-net: Single image radiometric calibration using CNNs. In *Proceedings of the 14th European Conference on Visual Media Production (CVMP 2017)* (2017), pp. 1–9. 5

[LRB*16] LAINA I., RUPPRECHT C., BELAGIANNIS V., TOMBARI F., NAVAB N.: Deeper depth prediction with fully convolutional residual networks. In *Proc. 3DV* (2016). 3

[LS18] LI Z., SNAVELY N.: Megadepth: Learning single-view depth prediction from internet photos. In *Proc. CVPR* (2018). 3, 4, 5

[LSD15] LONG J., SHELHAMER E., DARRELL T.: Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2015), pp. 3431–3440. 3

[LSK*20] LOGACHEVA E., SUVOROV R., KHOMENKO O., MASHIKHIN A., LEMPITSKY V.: Deeplandscape: Adversarial modeling of landscape video. In *European Conference on Computer Vision (ECCV)* (2020). 2

[LSL15] LIU F., SHEN C., LIN G.: Deep convolutional neural fields for depth estimation from a single image. In *Proc. CVPR* (2015). 3

[MGAD19] MURMANN L., GHARBI M., AITTALA M., DURAND F.: A dataset of multi-illumination images in the wild. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)* (October 2019). 2

[ML04] MADSEN C. B., LAURSEN R.: Image relighting: Getting the sun to set in an image taken at noon. In *13th Danish Conference on Pattern Recognition and Image Analysis* (2004), Citeseer, pp. 13–20. 2

[MWBJ16] MIHAIL R. P., WORKMAN S., BESSINGER Z., JACOBS N.: Sky segmentation in the wild: An empirical study. In *Applications of Computer Vision (WACV), 2016 IEEE Winter Conference on* (2016), IEEE, pp. 1–6. 3

[MZRT16] MEKA A., ZOLLHÖFER M., RICHARDT C., THEOBALT C.: Live intrinsic video. *ACM Trans. on Graph. (TOG) 35*, 4 (2016), 109. 2

[NMC*19] NAM S., MA C., CHAI M., BRENDEL W., XU N., KIM S. J.: End-to-end time-lapse video synthesis from a single outdoor image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2019). 2

[Per85] PERLIN K.: An image synthesizer. *ACM Siggraph Computer Graphics 19*, 3 (1985), 287–296. 6

[PGB03] PÉREZ P., GANGNET M., BLAKE A.: Poisson image editing. *ACM Transactions on graphics (TOG) 22*, 3 (2003), 313–318. 2

[PGZ*19] PHILIP J., GHARBI M., ZHOU T., EFROS A., DRETTAKIS G.: Multi-view relighting using a geometry-aware network. *ACM Transactions on Graphics (SIGGRAPH Conference Proceedings) 38*, 4 (July 2019). 2, 3

[PML*03] PFISTER G., MCKENZIE R., LILEY J., THOMAS A., FORGAN B., LONG C. N.: Cloud coverage based on all-sky imaging and its impact on surface solar irradiance. *Journal of Applied Meteorology 42*, 10 (2003), 1421–1434. 3

[RAGS01] REINHARD E., ASHIKHMIN M., GOOCH B., SHIRLEY P.: Color transfer between images. *IEEE Computer graphics and applications 21*, 5 (2001), 34–41. 2, 8

[RBK21] RANFTL R., BOCHKOVSKIY A., KOLTUN V.: Vision transformers for dense prediction. *arXiv:2103.13413* (2021). 12

[RCC98] RUDERMAN D. L., CRONIN T. W., CHIAO C.-C.: Statistics of cone responses to natural images: implications for visual coding. *JOSA A 15*, 8 (1998), 2036–2045. 8

[RDL*15] REN P., DONG Y., LIN S., TONG X., GUO B.: Image based relighting using neural networks. *ACM Transactions on Graphics (TOG) 34*, 4 (2015), 111. 2

[RGSN18] RAWAT S., GAIROLA S., SHAH R., NARAYANAN P. J.: Find me a sky: A data-driven method for color-consistent sky search and replacement. In *International Conference on MultiMedia Modeling* (Cham, 2018), Springer International Publishing, pp. 216–228. 3

[RKAJ08] REINHARD E., KHAN E. A., AKYUZ A. O., JOHNSON G.: *Color imaging: fundamentals and applications*. CRC Press, 2008. 6

[RKB04] ROTHER C., KOLMOGOROV V., BLAKE A.: Grabcut: Interactive foreground extraction using iterated graph cuts. In *ACM transactions on graphics (TOG)* (2004), vol. 23, ACM, pp. 309–314. 3

[RSSF02] REINHARD E., STARK M., SHIRLEY P., FERWERDA J.: Photographic tone reproduction for digital images. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), pp. 267–276. 8

[RT16] ROY A., TODOROVIC S.: Monocular depth estimation using neural regression forest. In *Proc. CVPR* (2016). 3

[SBT*19] SUN T., BARRON T. J., TSAI Y.-T., XU Z., YU X., FYFFE G., RHEMANN C., BUSCH J., DEBEVEC P., RAMAMOORTHI R.: Single image portrait relighting. *ACM Transactions on Graphics (TOG)* (2019). 2

[Sch95] SCHLICK C.: Quantization techniques for visualization of high dynamic range pictures. In *Photorealistic Rendering Techniques*. Springer, 1995, pp. 7–20. 8

[Sch16] SCHNEIDER A.: Real-time volumetric cloudscapes. *GPU Pro 7: Advanced Rendering Techniques 97* (2016). 1, 6

[SHKF12] SILBERMAN N., HOIEM D., KOHLI P., FERGUS R.: Indoor segmentation and support inference from RGBD images. In *Proc. ECCV* (2012). 3

[SPDF13] SHIH Y., PARIS S., DURAND F., FREEMAN W. T.: Data-driven hallucination of different times of day from a single outdoor photo. *ACM Transactions on Graphics (TOG) 32*, 6 (2013), 200. 2

[SSL12] SANIN A., SANDERSON C., LOVELL B. C.: Shadow detection: A survey and comparative evaluation of recent methods. *Pattern recognition 45*, 4 (2012), 1684–1695. 12

[SSN09] SAXENA A., SUN M., NG. A. Y.: Make3D: Learning 3D scene structure from a single still image. *IEEE Trans. Pattern Analysis and Machine Intelligence 31*, 5 (2009), 824–840. 3

[SW13] SHEN Y., WANG Q.: Sky region detection in a single image for autonomous ground robot navigation. *International Journal of Advanced Robotic Systems 10*, 10 (2013), 362. 3, 5

[Tar09] TARDIF J.-P.: Non-iterative approach for fast and accurate vanishing point detection. In *Computer Vision, 2009 IEEE 12th International Conference on* (2009), IEEE, pp. 1250–1257. 4

[TSL*16] TSAI Y.-H., SHEN X., LIN Z., SUNKAVALLI K., YANG M.-H.: Sky is not the limit: Semantic-aware sky replacement. *ACM Transactions on Graphics (TOG) 35*, 4 (2016), 149. 2, 3

[WAM02] WELSH T., ASHIKHMIN M., MUELLER K.: Transferring color to greyscale images. In *ACM Transactions on Graphics (TOG)* (2002), vol. 21, ACM, pp. 277–280. 2

[WGZ*15] WORKMAN S., GREENWELL C., ZHAI M., BALTENBERGER R., JACOBS N.: Deepfocal: A method for direct focal length estimation. In *2015 IEEE International Conference on Image Processing (ICIP)* (2015), IEEE, pp. 1369–1373. 5

[Wor96] WORLEY S.: A cellular texture basis function. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (1996), pp. 291–294. 6

[WZJ16] WORKMAN S., ZHAI M., JACOBS N.: Horizon lines in the wild. *arXiv preprint arXiv:1604.02129* (2016). 4

[XOH13] XU Y., OH S., HOOGS A.: A minimum error vanishing point detection approach for uncalibrated monocular images of man-made environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2013), pp. 1376–1383. 4

[XSHR18] XU Z., SUNKAVALLI K., HADAP S., RAMAMOORTHI R.: Deep image-based relighting from optimal sparse samples. *ACM Transactions on Graphics (TOG) 37*, 4 (2018), 126. 2

[YLMY12] YANG J., LU W., MA Y., YAO W.: An automated cirrus cloud detection method for a ground-based cloud image. *Journal of Atmospheric and Oceanic Technology 29*, 4 (2012), 527–537. 3

[YME*20] YU Y., MEKA A., ELGHARIB M., SEIDEL H.-P., C.THEOBALT, SMITH W.: Self-supervised outdoor scene relighting. In *Proc. of ECCV* (2020). 2, 3, 10, 12

[YML*16] YANG J., MIN Q., LU W., MA Y., YAO W., LU T., DU J., LIU G.: A total sky cloud detection method using real clear sky background. *Atmospheric Measurement Tech. 9*, 2 (2016), 587–597. 3

[YZZ*17] YAN H., ZHANG Y., ZHANG S., ZHAO S., ZHANG L.: Focal length estimation guided with object distribution on focalens dataset. *Journal of Electronic Imaging 26*, 3 (2017), 033018. 5

[ZHSJ19] ZHOU H., HADAP S., SUNKAVALLI K., JACOBS D. W.: Deep single-image portrait relighting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)* (October 2019). 2

[ZSQ*17] ZHAO H., SHI J., QI X., WANG X., JIA J.: Pyramid scene parsing network. In *CVPR* (2017). 3, 12

[ZZP*17] ZHOU B., ZHAO H., PUIG X., FIDLER S., BARRIUSO A., TORRALBA A.: Scene parsing through ADE20K dataset. In *Proc. CVPR* (2017). 3

[ZZP*19] ZHOU B., ZHAO H., PUIG X., XIAO T., FIDLER S., BARRIUSO A., TORRALBA A.: Semantic understanding of scenes through the ADE20K dataset. *International Journal of Computer Vision 127*, 3 (2019), 302–321. 12