Cassidy All
Katherine Carter
Bijan Jourabchi

1 Abstract

In this paper we implement a method for detecting an altered photo originally proposed by Kee et al. [6]. We examine whether the shadows and shading in an image are consistent with one another. We develop linear systems from physical models and solve them using singular value decomposition, least-norm minimization, and other techniques for matrix manipulation. We use this method on a picture we took and an altered composite we created. We found that the methods proposed by Kee et al. were successful in determining the altered image in our experiments [6].

2 Attribution

Bijan Jourabchi and Cassidy All implemented methods, performed experiments, and wrote code; Katherine Carter analyzed results and generated image vectors; all three contributed to the paper.

3 Introduction

The U.S. Surgeon General recently issued a call for a "warning label" on social media platforms [7]. Doctored images, created using Artificial Intelligence (AI) or popular photo-editing tools like Facetune have become increasingly common. In the course of day-to-day social media usage, a survey found that 71% of people surveyed would not post a photo on social media without manipulating it first [1]. The increased prevalence of image manipulation on social media has profound negative mental health implications [9] [3]. Additionally, images are a prominent and growing misinformation/disinformation vector, especially with the increasing accessibility of image manipulation using AI [2]. Thus, tools to detect the presence of image manipulation are of interest to many stakeholders: they have profound implications for mental health, democracy, and more.

4 Methods

How do we determine whether a photo has been altered? Clues can be found in the shading/shadows present on objects in an image. Our approach, drawn from [6], attempts to determine whether objects in an image were photographed under consistent lighting conditions—or—if the image is a fabricated composite. In this paper we develop approaches for constraining the direction of a single point light source based on the shading/shadows present on objects in the image. By placing these constraints into a single linear system and determining whether a solution exists we can determine whether the objects in an image could be consistent with having been photographed under the same lighting conditions. If they are not, we can reasonably conclude the image is a forgery.

First, we describe constraints on a potential light source from cast and attached shadows. We develop half-plane and line inequalities from a manually-specified physical model. These inequalities form a linear system which we solve using least-square approximation to minimize an included error term.

Second, we describe estimating 3-D surface normals from a 2-D image (not necessarily easy) and the radiance of an object. Together, these calculations introduce an additional geometric construction to the image and enable analysis of characteristics of objects' surfaces.

Third, we develop 3-D shading constraints by estimating a linear system using Singular Value Decomposition (SVD). Then, we note how another application of SVD can be used to project 3-D constraints onto the 2-D image plane by taking advantage of orthonormal vectors.

Finally, we apply these techniques to a real image we took, and a fabricated composite including a baseball which was pictured under significantly different lighting conditions. See those images in Figure 1.

5 Assumptions

First, we begin with some simplifying assumptions drawn from [4] and [6]. We assume our surface of interest is (1) Lambertian; (2) has constant reflectivity; (3) is illuminated by an infinite point light source; (4) the angle between the surface normal and light direction is between 0°and 90°; (5) the camera's optical center is aligned with the center of the image; and (6) the radiance of a select image patch is uniform across the patch, and $\mathbf{r}$ is selected uniformly at random from a patch. (1) and (2) are simplifying assumptions we will use to construct our expressions to constrain light direction. We use (3) and (6) to simplify radiance calculations for the shading of objects, shadow analysis would be possible using a local light source. (4) and (5) restrict

Figure 1: Test Images

our working quadrants. These are the primary assumptions which enable (relatively) simple physically-derived lighting models. We touch on other assumptions, primarily relating to sources of uncertainty, as they appear in this paper.

6 Shadows

Examining the shadows present in an image also provides linear inequalities which we can use for additional clues as to an image's authenticity. In this section we develop models for both cast and attached shadows, as well as discussing potential sources of uncertainty in their calculation.

6.1 Modeling Cast Shadows

Cast shadows are what we traditionally think of as shadows—a 2D projection of a "dark spot" on a surface caused by an object blocking a light source. In a 2-D scene cast shadows can be modeled by a "wedge" shape representing potential light directions. This "wedge" is constrained by a point in the shadow and lines along the boundary of the occluding object. We provide an example in Figure 2. Kee et al. note that this relationship holds regardless of object geometry and for a light source at any

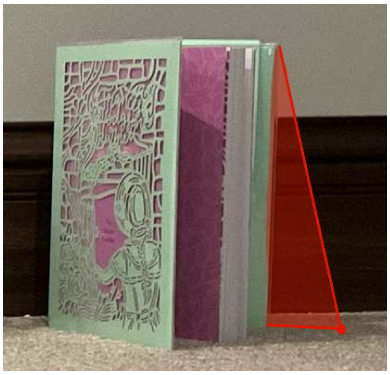

Figure 2: Example Cast Shadow Constraint

distance—meaning it is squarely within the bounds of our given assumptions [6].

6.1.1 Theoretical Approach

First, let us gather two inequalities useful for describing a "wedge-like" constraint on a light source.

$$n_i{}^1 \cdot x - n_i{}^1 \cdot p_i \geq 0 \text{ and } n_i{}^2 \cdot x - n_i{}^2 \cdot p_i \geq 0 \tag{1}$$

Together, these inequalities describe the region between two lines sharing a point $\mathbf{p}_i$ by their implicit normals $\mathbf{n}_i{}^1$ and $\mathbf{n}_i{}^2$. We formalize a collection of these constraints as a system of inequalities,

$$\begin{pmatrix} n_1 \\ n_2 \\ \vdots \\ n_m \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} n_1 \cdot p_1 \\ n_2 \cdot p_2 \\ \vdots \\ n_m \cdot p_m \end{pmatrix} \geq 0 \tag{2}$$

$$Nx - P \geq 0 \tag{3}$$

At this stage, we make our first steps to address potential error. If our scene is real and perfectly-described this system should always have a solution. However, given the difficulty of accurately manually specifying surface normals we introduce an error-mitigating variable $\mathbf{s}$. We let $\mathbf{s}$ by a column vector with length m and let our system be greater than or equal to $\mathbf{s}$. Then, our system should be consistent when $\mathbf{s} = 0$, otherwise, we have a term that allows us to estimate deviation from consistency and compare images with greater confidence. We describe this modified system below.

$$\begin{pmatrix} N & I \\ 0 & I \end{pmatrix} \begin{pmatrix} x \\ s \end{pmatrix} - \begin{pmatrix} P \\ 0 \end{pmatrix} \geq 0 \tag{4}$$

Now, we have re-formulated the problem as an optimization problem attempting to minimize the L_1 norm of $\mathbf{s}$. Note: we choose to minimize the L_1 norm for consistency with the original paper [5] and because the L_1 norm describes more possible solutions which is important in this application. Finally, before we continue we note that their is a sign ambiguity in our constraint. If the light source is behind the camera our inequality will be inverted (this makes intuitive sense, as it represents a 180° shift). Thus, we consider our system satisfied if we can find a solution to either:

$$Nx - P \geq -s \;\; or \;\; -Nx + P \geq -s \tag{5}$$

Ultimately, these inequalities describe a "wedge" in which a potential light source could be located. We can also express this constraint as attempting to find a line between a point in a cast shadow and the occluding object. That line is expressed as the simple inequality,

$$\begin{aligned} s &= \alpha d + p \\ \alpha &\geq 0 \end{aligned} \tag{6}$$

Here, $\mathbf{p}$ is a point in shadow while $\mathbf{d}$ is a vector representing the line between a the point in shadow and a point on the object. This defines a line constraining the light source $\mathbf{s}$. Note that α is a constant which allows us to "vary" the distance between the objects and define a line. Our model constrains direction rather than distance—which would require significantly more assumptions about a given light source. We provide this alternative formulation of the linear constraints generated by cast shadows to note that this is identical to the linear constraint for shading we express in (25). The core of our approach is highly versatile and can be applied to a wide variety of geometries so long as we can develop an appropriate model.

6.1.2 Forensic Process

Specifying cast shadow constraints requires manual specification of three things: a point within the shadow, and two lines connecting the point to the edges of the occluding object. This introduces a source of potential error—that selected points poorly specify a given shadow. The specific decisions made specifying constraints could have an impact on their ultimate suitability. Kee et al. did not propose additional solutions to combat this problem beyond the introduction of an error term [5].Thus, we proceed assuming that the introduction of constraints from multiple objects will lead to a system which is an accurate representation of potential constraints.

We illustrate this process in Figure 3. The red point and lines represent the manually specified shadowed region while the yellow shaded region represents the "wedge" constraint on potential light sources from this object. Readers may note that, technically, our inequalities allow for a point light source within the red shaded region, although this would not make physical sense. In fact, so long as multiple constraints exist, this is impossible if the image is illuminated by a single light source as we assume. Thus, we do not have to further complicate our model to account for this.

In order to model the constraints on a light source from cast shadows we selected a point in the shadowed region and generated two normal vectors specifying a wedge. We selected the point and defined lines using Desmos. We used Wolfram Alpha for calculations of the normal vectors. After developing constraints for each of the objects in our image we created a program formalizing (4). This is a somewhat unique problem. Given that the problem is likely not homogeneous we cannot solve the system directly. Instead, we try to find the minimum-norm solution within the image of the matrix describing our system. The most obvious approach would be the implementation of a simple root-finding routine. However, this would also be computationally expensive and scale poorly for larger matrices. Instead, we can use tools from linear algebra! We compute the

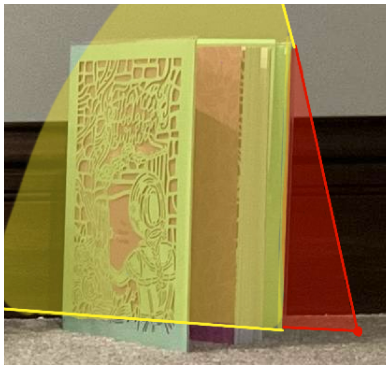

Figure 3: Constraint of Potential Light Source from Cast Shadows

least-norm solution to $s = \begin{pmatrix} 0 & 0 & ... & 0_m \end{pmatrix}$ and solve for $\mathbf{s}$ in order to minimize the L_1 norm of S. Because our norm does not "weight" terms differently, we can confidently apply least-norm approximation here. Code for this process can be found in 11.1 and our results are described in 8.

6.2 Modeling Attached Shadows

Attached shadows are produced when an object occludes light from itself and are most common on folds or locally convex geometry [5]. Think of how the moon "casts" a shadow on itself. Attached shadows occur when a surface normal makes an angle greater than 90°with a light source. This makes sense—if the angle of the surface normal is greater than 90°the light is not "continuing" along the object, instead, it is occluded.

6.2.1 Theoretical Approach

Substantively, attached shadows are incredibly similar to cast shadows. The primary difference is that they offer less information. Thus, we are only able to constrain a light source to a half-plane rather than the "wedge" we developed earlier. Still, we are able to use identical constraints as to those developed in (1), except that we gain one inequality rather than two. However, we can add those inequalities to the system we constructed above, (2) and proceed identically with (3) (4) (5).

We can also express this half-plane with the inequality,

$$s \cdot d - p \cdot d \geq 0 \tag{7}$$

This equation describes a half-plane along which the light source $\mathbf{s}$ must lie. $\mathbf{p}$ is a point where the surface normal is greater than 90° from the direction of the light, as constrained by other equations, and $\mathbf{d}$ is a vector orthogonal to a "terminating" half-plane along an attached shadow. Clearly, this inequality constrains the direction of $\mathbf{s}$ to directions consistent with the attached shadows present. Note that this describes a constraint of the same form as (26) for shading. Once again, the geometric underpinnings of this approach are remarkably consistent.

6.2.2 Forensic Process

To model the constraints originating from attached shadows we only need to specify two thing: a point within an attached shadow and a line forming a 90° with that point. From this line we can derive an implicit normal consistent across the entirety of the line and, theoretically, forming a similarly consistent "terminating" plane on the surface resulting in occlusion. Again, results may vary slightly depending on the exact points specified. For our test images we did our best to select points which would maximally constrain a potential light source.

We used Desmos to select points along the attached shadow and generate a perpendicular line to the point. We used Wolfram Alpha to calculate corresponding normal vectors. We added these additional normal vectors/points to the system we attempt to find a solution for as described in 6.1.2. See Figure 4 for an illustration of what a sample point/line looked like, and the constraint it generated.

6.3 Uncertainty

Largely, uncertainty arises because it "may be difficult to accurately specify a cast shadow constraint due to finite resolution, ambiguity in matching a cast shadow to its corresponding object, or when the shadow is indistinct" [6]. However, because the constraints from attached shadows are so easily specified we are fairly confident in those constraints. Thus, we are content to address potential uncertainty by specifying generous "wedge-based' constraints and optimizing our error term $\mathbf{s}$.

Figure 4: Constraint of Potential Light Source from Attached Shadows

7 Shading

7.1 Estimating Normal Vectors

Crucially, we want to be able to estimate the direction of a point light source from the objects in our image. To do this, we need a way to describe an objects interaction with a light source geometrically. A common approach is to define functions for light direction using the normal vectors of objects in an image. Thus, the first problem we're presented with is how to estimate these normal vectors.

The primary difficulty arises from the fact that we are attempting to estimate 3-D geometry from a 2-D image. Surface normals should be vectors of the form $\begin{pmatrix} x & y & z \end{pmatrix}^T$. However, this is generally impossible to estimate from a 2-D image unless you have multiple images or an object with known geometry. Instead, we can simplify the problem further by estimating surface normals along an occluding boundary, an approach described by [8]. Occluding boundaries are the smooth curves along which an object obscures itself. For example, the Earth's horizon is an occluding contour. Along occluding boundaries the z-component of a surface normal is zero because a viewer's perspective is along the z-axis. This allows us to simplify our normal vectors, reducing the necessary dimensions $\begin{pmatrix} x & y & 0 \end{pmatrix}^T$. The number of points needed to accurately estimate normal vectors using this lower-dimensional contour varies depending on the exact error function applied, however, accurate estimates should occur when the points included are ≥ 3 [4].

We use the methodology described in [4] to manually describe our occluding boundaries as existing algorithms are computationally expensive and, for our purposes, provide an unnecessary level of grain. We manually identified occluding boundaries in an image, partitioned each boundary into eight segments, manually identified three points on each segment, and fit those points to a quadratic curve. Then, using these quadratic curves we were able to analytically estimate surface normals. Specifically, we calculate the tangent line at the center-most point on the segment and find the perpendicular unit vector to the tangent line at that point. This results in a good approximation of the normal vectors along the occluding contour of an image. See Figure 5 for a depiction of this process and the actual boundary we used for an object in our test image. Boundaries were manually segmented using a free photo-editing tool, points were selected using Desmos, and further calculations were done using Wolfram Alpha. We report the matrix representations of approximated normal vectors for the objects in our test images below.

$$
\begin{aligned}
N_{box} &= \begin{pmatrix} .20867 & .977986 \\ .172534 & .985003 \\ .388549 & .921428 \\ .185988 & .982552 \\ .180088 & .983651 \\ .1113 & .993787 \\ .239325 & -.970939 \\ .935695 & .35281 \end{pmatrix} \quad N_{heel} = \begin{pmatrix} 0.821522 & 0.570177 \\ 0.998265 & -0.0588896 \\ 0.363304 & 0.931671 \\ 0.303697 & 0.952769 \\ 0.70921 & 0.704998 \\ 0.936248 & -0.351339 \\ 0.721686 & 0.692221 \\ 0.993827 & -0.110939 \end{pmatrix} \\
N_{book} &= \begin{pmatrix} 0.999157 & -0.042555 \\ 0.999157 & -0.042555 \\ 0.999157 & -0.042555 \\ 0.999157 & -0.042555 \\ 0.999157 & -0.042555 \\ 0.999157 & -0.042555 \\ 0.999157 & -0.042555 \\ 0.999157 & -0.042555 \end{pmatrix} \quad N_{ball} = \begin{pmatrix} .0929284 & .995673 \\ .121927 & .992539 \\ .137655 & .99048 \\ .181355 & .983418 \\ .177949 & .98404 \\ .207478 & .97824 \\ .806337 & .591456 \\ .130778 & -0.991412 \end{pmatrix}
\end{aligned}
\tag{8}
$$

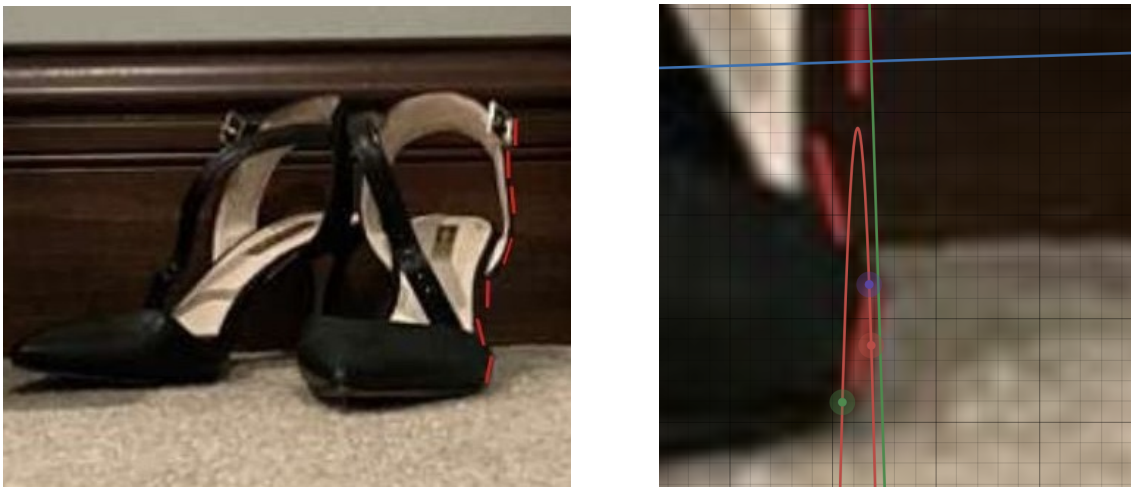

Figure 5: Normal Vector Approximation

7.2 Estimating Radiance

Unfortunately, we cannot directly measure lighting intensity from our occluding boundaries because, of course, the boundary is occluded. However, Nillius & Eklundh note that it should be sufficient to measure intensity close to the border [8]. Thus, we proceed with the methodology described in [4]. We assume constant reflectivity along the surface so we only need to make a single measurement, one pixel away, from the boundary of radiance to compute the radiance for that entire patch. To do this, we convert the image to grayscale and measure the intensity of a randomly-chosen pixel near the border, normalizing it by dividing by 255. We report these values below.

$$r_{box} = 0.2745098039 \quad r_{heel} = 0.1843137255 \quad r_{book} = 0.231372549 \quad r_{ball} = 0.4666666667 \tag{9}$$

7.3 Developing 3-D Constraints for a Point Light Source

Kee et al. outlines the process we utilized for creating shading constraints [6]. Analyzing the images' shading constraints begins with breaking the image into image patches that correspond to a surface normal vector $\mathbf{n} = (n_x, n_y, n_z)^T$. To develop our shading constraints, we must estimate where our light source is, $\mathbf{s} = (s_x, s_y, s_z)^T$. The radiance of a light patch is given by

$$r = \mathbf{n} \cdot \mathbf{s} + a \tag{10}$$

where a is defined as the ambient term—a variable constant representing indirect illumination. The equations for each surface normal and corresponding radiance form a matrix:

$$\begin{pmatrix} n_1^T & 1 \\ n_2^T & 1 \\ . & . \\ . & . \\ . & . \\ n_k^T & 1 \end{pmatrix} \begin{pmatrix} \mathbf{s} \\ a \end{pmatrix} = \mathbf{r} \tag{11}$$

$$\mathbf{Ne} = \mathbf{r}$$

When the matrix $\mathbf{N}$ is full rank, $\mathbf{e}$ can be solved for directly. This is not always the case, but while it might not be possible to determine an exact solution, it is possible to constrain the solution to a space of potential solutions.

Radiance is strictly modeled in 3-D—it would be incredibly difficult to create a solely 2-D model. Thus, our normal vectors should be three-dimensional. Determining a 3-D normal vector from a 2-D image is difficult and unreliable. However, by restricting ourselves to points along an occluding contour, a 3-D normal can be reasonably estimated from a 2-D normal. As we noted above, along an occluding contour the z-element is usually assumed to be zero. However, this assumption does not strictly hold because a camera will distort these boundaries. To support less ideal surface normals we implicitly compute n_z given camera information. This approach is outlined in [6]. We let $\mathbf{n}$ and $\mathbf{s}$ be defined in a camera coordinate system, where the camera faces down along the z-axis with an origin at the optical center of the camera. Thus, we define 2-D origin at the center of our image and the z-axis lies along the camera. Thus, we can impute n_z using properties of our imagined camera. The 3D

surface normal can be estimated with:

$$\mathbf{n} = \begin{pmatrix} n_x \\ n_y \\ \frac{1}{f}\mathbf{n} \cdot (\mathbf{x} - \mathbf{c}) \end{pmatrix} \tag{12}$$

Here, f is the cameras focal length. Usually this information would be unknown and we would let f be a free variable. However, since we created the test images we know that $f = 24$mm for an IPhone. Having appropriately scaled $\mathbf{n}$ we take the dot product with $\mathbf{x}$, the 2-D location of each normal with respect to the image center $c = 0$.

Before continuing, we also note that Kee et al. claim that this lighting model can be estimated for k ≥ 4 image patches, although little explanation is given as to why [6].

We presume $\mathbf{N}$ is rank deficient, and proceed by performing singular value decomposition $N = USV^T$ where $\mathbf{S}$ is a diagonal matrix of singular values $\sigma_{1\ldots n}$ while $\mathbf{U}$ and $\mathbf{V}^T$ consist of the left and right singular vectors. As we know, $\mathbf{V}$ also forms an orthonormal basis for our lighting parameters $\mathbf{e}$.

$$V = \begin{pmatrix} v_1 & v_2 & v_3 & v_4 \end{pmatrix} = \begin{pmatrix} w_1 & w_2 & w_3 & w_4 \\ b_1 & b_2 & b_3 & b_4 \end{pmatrix} \tag{13}$$

This orthonormal basis will become the foundation for constraining our light source in 3D. Our lighting parameters $\mathbf{e}$ can be estimated with

$$e_0 = N^+ r \tag{14}$$
$$N^+ = VS^{-1}U^T$$

when N is singular, where N^+ is the pseudo-inverse of N. We let $\mathbf{e}_0$ be our estimated lighting parameters, where $\mathbf{e}_0 = \begin{pmatrix} s_0 \\ a \end{pmatrix}$. Our possible lighting solutions differ depending on the number of degenerate singular values, $\sigma_k/\sigma_1 \approx 0$. We outline those possibilities below.

Given no degenerate values, e_0 can be fully estimated using (14).

When one singular value is degenerate, $\sigma_4/\sigma_1 \approx 0$, our light can be constrained to a line in 3-D, which corresponds to a half-circle of directions, and can be given by

$$\mathbf{s} = s_0 + \alpha w_4 \tag{15}$$

When two singular values are degenerate, $\sigma_4/\sigma_1 \approx 0$ and $\sigma_3/\sigma_1 \approx 0$, our light source can be constrained to a plane in 3D, corresponding to a half circle of directions, given by

$$\mathbf{s} = s_0 + \alpha w_4 + \beta w_3 \tag{16}$$

Finally, given three degenerate singular values, the kernel of N spans all of 3-D space. Thus we have no constraint on our light source.

7.4 Projecting 3-D Constraints onto the 2-D Image Plane

Light directions in 3-D can be considered as points at infinity in 2-D, thus projecting our 3-D lighting constraints onto our 2-D image plane. Under an idealized camera with focal length f and image center $\mathbf{c}$, the light source in the direction of $\mathbf{s}$ projects to a point in the 2D plane:

$$\tilde{s} = \begin{pmatrix} f & 0 & -c_x \\ 0 & f & -c_y \\ 0 & 0 & -1 \end{pmatrix} s \tag{17}$$
$$\tilde{s} = K\mathbf{s}$$

Here, $\mathbf{K}$ is the intrinsic matrix of the camera and the image light source $\tilde{s}$ is given in homogeneous coordinates. When the linear system is full rank, $\mathbf{s}$ can be solved directly and projected to a single point on the image plane. Again, we develop several cases depending on the number of degenerate singular values.

When one singular value is degenerate, the light direction is constrained to a half-circle of directions in 3D. These directions project onto a line into the image plane:

$$\tilde{s} = K(s_0 + \alpha w_4) \tag{18}$$

This line can be described by a position $\mathbf{p}$ and direction $\mathbf{d}$ in the image plane:

$$\mathbf{p} = Kw_4 \tag{19}$$

$$\mathbf{d} = sign(s_{0_z})(\mathbf{p} - s_0) \tag{20}$$

The direction **d** is a normal vector indicating the direction of a point light source with respect to the imagined camera. For consistency, when the estimated light direction is in front of the camera($s_{0_z} < 0$)—we define **d** as the line segment connection **p** to s_0, and vice versa if $s_{0_z} > 0$.

When two singular values are degenerate, the light direction is constrained to a half sphere in 3-D, and a plane in 2-D.

$$\tilde{s} = K(s_0 + \alpha w_4) + \beta w_3 \tag{21}$$

Although a plane encompasses the entire image, it can be split into two half-planes, one for light directions in front of the camera and one for light directions behind the camera. These half planes are described by **p** and **d** where $\tilde{h}$ is the line that separates the two half-planes ($\tilde{h} = Kw_4 \times Kw_3$):

$$\mathbf{p} = Kw_4 \tag{22}$$

$$\mathbf{d} = sign(\tilde{h} \cdot s_0)\tilde{h} \tag{23}$$

7.5 Formalizing Constraints

Together, we evaluate whether the shading constraints allow for a single physically consistent light source. Thus, we combine them into a single linear system. When the light position can be constrained to a single point in the image, the following equality is placed on the light source**s**,

$$s = Ks_0 \tag{24}$$

When the light position is constrained to a line the following constraint is placed on **s**,

$$\begin{aligned} s &= \alpha d + p \\ \alpha &\geq 0 \end{aligned} \tag{25}$$

Readers should note that this is identical to the inequality we described in (6.1.1). Finally, when the light source is constrained by a half-plane, the following inequality is placed on **s**.

$$s \cdot d - p \cdot d \geq 0 \tag{26}$$

Again, this is identical to inequality we described in (7).

A viable solution to the system indicates that shading in an image is consistent. When a solution cannot be found, one or more of the constraints is invalid, and the shading is not consistent. When the light source is behind the camera, $s_{0_z} > 0$ all constraint inequalities are reversed, and shading is determined to be consistent if the reversed inequalities are satisfied. Because the formalized constraints take the same form as those for shadows, we can construct a linear system consisting of more constraints than were we to use either approach alone.

7.6 Uncertainty

Similarly to cast shadow constraints, uncertainty in shading and point light estimation constraints arises due to poor image resolution, as well as "deviations from the assumptions of Lambertian reflectance and a distant light source" [6]. Uncertainty is more likely to occur when our **N** matrix is not well conditioned, and thus doesn't provide a good estimation for the 3D model. Thus uncertainty is most common in the cases where our light source is constrained to a line or a half plane [6]. We can check the condition of our constraints using our singular values. The condition of our line constraint is given by $\kappa_L = \sigma_3/\sigma_1$ and our plane is given by $\kappa_P = \sigma_2/\sigma_1$ [6]. The higher these kappa values are, the less conditioned the constraint is, and the more uncertainty we have in our result.

8 Results

8.1 Shadows

First, we present the matrices **N** and **P** as described in (2) using the techniques we outlined in 6.1.2 and 6.2.2. They represent a combined linear system of constraints from both cast and attached shadows.

$$N_{real} = \begin{pmatrix} -0.139038 & 0.990287 \\ 0.0167039 & 0.99986 \\ -0.110262 & 0.99393 \\ 0.0443534 & 0.999016 \\ -0.0900761 & 0.995935 \\ 0.00868626 & -0.999962 \\ -0.026519 & 0.999648 \\ -0.0816932 & 0.996658 \end{pmatrix} \quad P_{real} = \begin{pmatrix} 0.05646653 \\ 0.75290322 \\ 0.13712383 \\ 1.07495521 \\ 0.10060033 \\ -0.65636249 \\ 3.48015984 \\ 3.0599393 \end{pmatrix}$$

$$N_{fake} = \begin{pmatrix} -0.139038 & 0.990287 \\ 0.0167039 & 0.99986 \\ -0.110262 & 0.99393 \\ 0.0443534 & 0.999016 \\ -0.0900761 & 0.995935 \\ 0.00868626 & -0.999962 \\ -0.118596 & 0.992943 \\ 0.185023 & 0.982734 \\ -0.026519 & 0.999648 \\ -0.0816932 & 0.996658 \\ -0.0995431 & 0.995033 \end{pmatrix} \quad P_{fake} = \begin{pmatrix} 0.05646653 \\ 0.75290322 \\ 0.13712383 \\ 1.07495521 \\ -0.10060033 \\ -0.65636249 \\ 1.92596513 \\ 4.06240416 \\ 3.48015984 \\ 3.0599393 \\ -1.35008784 \end{pmatrix} \tag{27}$$

Notice that the forgery adds an additional three constraints which should make the linear system impossible/more difficult to satisfy. In order to minimize the error term $\mathbf{s}$ we represent (2) as a least-squares-esque minimization problem utilizing the L_1 norm rather than the standard Euclidean norm.

$$min||Nx - P|| \tag{28}$$

$$\begin{aligned} &\text{minimize } 1^T v \\ &\text{subject to } \begin{pmatrix} N & -I \\ -N & -I \end{pmatrix} \begin{pmatrix} x \\ v \end{pmatrix} \leq \begin{pmatrix} P \\ -P \end{pmatrix} \end{aligned} \tag{29}$$

We implement solving with a linear program utilizing the Python library *cvxpy* for convex optimization. Code is available in 11.1. Ultimately, we found that $\mathbf{s}_{real} = 6.28$ for the real image and $\mathbf{s}_{fake} = 11.84$. Although $\mathbf{s}_{real}$ deviates significantly from zero, that could be expected given the abundant sources of uncertainty we identified above. More importantly, $\mathbf{s}_{fake}$ is much larger. Thus, we validate that this method is able to distinguish between true images and forgeries—although the bounds/confidence for that distinction would require further experimentation.

8.2 Shading

Given the volume of calculations required to develop shading constraints we use the box object from the test image to provide an illustrative example of the process. Our full results are reported on below and the code is available in 11.2. We start by estimating a 3-D normal vector from our 2-D vectors using the method outlined in (12). We used a free photo editing software to find the vector $\mathbf{x}$ with respect to the center of the image and calculated:

$$A_{box} = \begin{pmatrix} n_1^T \\ n_2^T \\ n_3^T \\ n_4^T \\ n_5^T \\ n_6^T \\ n_7^T \\ n_8^T \end{pmatrix} = \begin{pmatrix} .20867 & .977986 & 0.0127 \\ .172534 & .985003 & 0.0110 \\ .388549 & .921428 & 0.0080 \\ .185988 & .982552 & 0.0053 \\ .180088 & .983651 & 0.0020 \\ .1113 & .993787 & 0 \\ .239325 & -.970939 & 0.0026 \\ .935695 & .35281 & -0.0044 \end{pmatrix} \tag{30}$$

We then formed N_{box}, a linear system representing the constraints on a single light source from radiance, according to (11):

$$N_{box} = \begin{pmatrix} .20867 & .977986 & 0.0127 & 1 \\ .172534 & .985003 & 0.0110 & 1 \\ .388549 & .921428 & 0.0080 & 1 \\ .185988 & .982552 & 0.0053 & 1 \\ .180088 & .983651 & 0.0020 & 1 \\ .1113 & .993787 & 0 & 1 \\ .239325 & -.970939 & 0.0026 & 1 \\ .935695 & .35281 & -0.0044 & 1 \end{pmatrix} \tag{31}$$

We define r_box using the experimentally determined values from (9) as a vector of length m.

$$r_{box} = \begin{pmatrix} 0.2745 & 0.2745 & 0.2745 & 0.2745 & 0.2745 & 0.2745 & 0.2745 & 0.2745 \end{pmatrix} \tag{32}$$

Next, we perform a singular value decomposition, $N_{box} = USV^T$ in order to determine the number of degenerate singular nodes. We find our S matrix to be:

$$S = \begin{pmatrix} 3.6390 & 0 & 0 & 0 \\ 0 & 1.5311 & 0 & 0 \\ 0 & 0 & 0.6433 & 0 \\ 0 & 0 & 0 & 0.0125 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{33}$$

Our singular values are then determined to be $\sigma_1 = 3.6390$, $\sigma_2 = 1.5311$, $\sigma_3 = 0.6433$, and $\sigma_4 = 0.0125$. A degenerate singular value is defined as $\sigma_k/\sigma_1 \approx 0$ [6]. Since $\sigma_4 = 0.0125 \not\approx 0$ we observe that our matrix has no degenerate singular value, and thus our light source is constrained to a point source in 2D. Our next step is to solve for e_0, our least norm solution to the linear system in (11). To find the least squares solution, we first find the reduced SVD, $N = P\Sigma Q^T$ such that our diagonal matrix of singular values, Σ, is square and possesses an inverse, Σ^{-1}. Our pseudo-inverse is then given by $N^+ = Q\Sigma^{-1}P^T$. It follows that e_0 is given by

$$e_0 = N^+ r \tag{34}$$

$$N^+_{box} = \begin{pmatrix} 0.2907 & 0.0777 & 0.4740 & -0.2642 & -0.5 & -0.8299 & -0.3878 & 1.1394 \\ 0.0046 & 0.0265 & 0.0526 & 0.0974 & 0.1386 & 0.1650 & -0.5257 & 0.0410 \\ 41.6715 & 28.22 & 24.0430 & -7.6022 & -29.2841 & -47.5259 & 2.8969 & -12.4191 \\ -0.1593 & -0.0467 & -0.1644 & 0.1766 & 0.3216 & 0.4889 & 0.5724 & -0.1892 \end{pmatrix} \tag{35}$$

$$e_0 = \begin{pmatrix} 0.1554 \times 10^{-14} \\ 0.0173 \times 10^{-14} \\ 0.0083 \times 10^{-14} \\ 0.2745 \end{pmatrix} \tag{36}$$

$$s_0 = \begin{pmatrix} 0.0333 \times 10^{-14} \\ 0.0099 \times 10^{-14} \\ 0.1776 \times 10^{-14} \end{pmatrix} \tag{37}$$

With this, we now have everything we need to project our 3-D constraint onto our 2-D image plane. We begin by initializing the intrinsic camera matrix K defined in (17). We assume c_x and c_y to both be equal to 0, as we define the origin to be at the center of the image. IPhone cameras have a focal length of 24mm, giving us

$$K = \begin{pmatrix} 24 & 0 & 0 \\ 0 & 24 & 0 \\ 0 & 0 & -1 \end{pmatrix} \tag{38}$$

To determine the point constraint, we evaluate (17) and solve for $\tilde{s}$. We find that

$$\tilde{s} = \begin{pmatrix} 0.7994 \times 10^{-13} \\ 0.2373 \times 10^{-13} \\ -0.1776 \times 10^{-13} \end{pmatrix} \tag{39}$$

Where $\tilde{s}$ is in homogeneous coordinates, meaning our true s_x and s_y values are given by

$$s_x = \frac{\tilde{s_x}}{\tilde{s_z}} \quad s_y = \frac{\tilde{s_y}}{\tilde{s_z}} \tag{40}$$

Finally, we can constrain the location of our light source to a single point, $s_{box} = \begin{pmatrix} -4.5 \\ -1.3359 \end{pmatrix}$. We continue this process for our remaining objects until we have a shading constraint for each. We end up with the constraints $s_{heel} = \begin{pmatrix} -4.33 \\ -1.9167 \end{pmatrix}$, $s_{ball} = \begin{pmatrix} 0.7305 \\ -0.0520 \end{pmatrix}$, $s_{book} = s_{0,book} + \alpha w_{4,book} + \beta w_{3,book} = \begin{pmatrix} 0.1156 \\ -0.0049 \\ 3.0198 \times 10^{-14} \end{pmatrix} + \alpha w_{4,book} + \beta w_{3,book}$ with position $\mathbf{p}$ and direction $\mathbf{d}$ of

$$\mathbf{p} = \begin{pmatrix} -16.9630 \\ -0.4901 \\ -0.7067 \end{pmatrix} \text{ and } \mathbf{d} = \begin{pmatrix} 16.9537 \\ 0.4898 \\ -407.2807 \end{pmatrix}$$

Clearly, the constraints for s_{ball} occupy an entirely different area of the image to locations for a potential light source derived from genuine objects. While there is some variation, and the genuine objects do not technically ascribe a single consistent solution—they are close enough that we chalk it up to uncertainty. If our results were less stark we would proceed with measuring the distance between the constrained points using an appropriate norm.

8.3 Bringing it all Together

Ultimately, we only have a single additional constraint to add to the linear system we described earlier in (31)—the half-plane described by the book. Given that additional constraint, we re-compute our error function $\mathbf{s}$ and compare to constraints offered by both shading and shadows.

The introduction of the additional shading-based constraint introduced significant noise, $\mathbf{s}\tilde{1}2.72$. Thus, it actually made our classification worse. We believe this may be because the processes for calculating both constraints were substantially different and human error may have been introduced in image sizing, alignment, vector calculation, and more. It could also be because the shading constraint for the book is complicated by the fact that it has internal convex geometry (within the covers) which violates model assumptions.

However, disregarding the shading constraint offered by the book, we plot the approximate point constraints offered by shading constraints and approximate shadow constraints.

Figure 6: Final Scene

The rough locations of the point constraints from shading (yellow circles) are clearly within the boundaries established by the three real objects. In contrast, the projected point constraint from the composite baseball in deeply inconsistent with the constraints from shadows. Thus, this demonstrated how the composition of constriants from both shading and shadows can be an effective tool for detecting manipulated images.

9 Discussion and Conclusion

We have shown a method for determining if a photo is manipulated by measuring whether the shadows and shading are consistent with each other.

This method of creating constraints for shadows and shading could be extended to other types of forged photos. An example is crochet art and seeing if a work is made by AI or if it is made by a human. AI has trouble replicating the look and consistency of a crochet piece of art. A method involving linear systems could be used to see whether or not the crochet art could be produced by a human. Some constraints can be put on the stitches in the photo of the work to see if they are consistent with what a stitch produced by a human would look like. Constraints can also be put on the way that the stitches are attached.

Of course, this method has limitations. If a professional forger were to ensure that the shading and shadows were consistent with each other, it would render this method useless. This is something to take into account when using this method. It is also susceptible to a number of manually-specified constraints which introduce human error.

Future work could explore increased automation, better methods of enforcing internal consistency, and more advanced ways of dealing with error. Also, the introduction of new physical models could relax the assumptions necessary—making the methods described here more broadly applicable.

It will be fascinating to see what image forgeries will look like in the coming years. It remains to be determined whether technology to create them will outpace methods of finding forgeries.

10 References

[1] Georgia Aspinall. 'I Was Obsessed With Facetune': 71% Of People Won't Post A Picture Online Without Photoshopping It – That Needs To Change, Aug 2020.

[2] Nicholas Dufour, Arkanath Pathak, Pouya Samangouei, Nikki Hariri, Shashi Deshetti, Andrew Dudfield, Christopher Guess, Pablo Hernández Escayola, Bobby Tran, Mevan Babakar, et al. Ammeba: A large-scale survey and dataset of media-based misinformation in-the-wild. *arXiv preprint arXiv:2405.11697*, 2024.

[3] Rosalind Gill. Changing the perfect picture: Smartphones, social media and appearance pressures, Mar 2021.

[4] Micah K Johnson and Hany Farid. Exposing digital forgeries by detecting inconsistencies in lighting. In *Proceedings of the 7th workshop on Multimedia and security*, pages 1–10, 2005.

[5] Eric Kee, James F O'Brien, and Hany Farid. Exposing photo manipulation with inconsistent shadows. *ACM Transactions on Graphics (ToG)*, 32(3):1–12, 2013.

[6] Eric Kee, James F O'brien, and Hany Farid. Exposing photo manipulation from shading and shadows. *ACM Trans. Graph.*, 33(5):165–1, 2014.

[7] Vivek H. Murthy. Surgeon General: Why I'm Calling for a Warning Label on Social Media Platforms, Jun 2024.

[8] Peter Nillius and Jan-Olof Eklundh. Automatic estimation of the projected light source direction. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I. IEEE, 2001.

[9] Phillip Ozimek, Semina Lainas, Hans-Werner Bierhoff, and Elke Rohmann. How photo editing in social media shapes self-perceived attractiveness and self-esteem via self-objectification and physical appearance comparisons. *BMC Psychology*, 11(1), Apr 2023.

11 Appendix

11.1 Finding Solutions to Shadow Constraints Using Least-Norm Approximation

shadow_constraints

July 22, 2024

[ ]:
```
# req. packages
import numpy as np
import cvxpy as cp
```

[ ]:
```
# define manually specified values for cast shadows
n1 = np.array([[-0.139038,0.990287]])
n2 = np.array([[0.0167039,0.99986]])
n3 = np.array([[-.110262,0.99393]])
n4 = np.array([[0.0443534,0.999016]])
n5 = np.array([[-0.0900761, 0.995935]])
n6 = np.array([[.00868626, -0.999962]])
n7 = np.array([[-0.118596, 0.992943]])
n8 = np.array([[0.185023, 0.982734]])

p1 = np.array([[4.43,0.679]])
p2 = np.array([[6.039,0.8079]])
p3 = np.array([[9.264,0.73686]])
p4 = np.array([[7.1304347,2.7913044]])
```

[ ]:
```
# define manually specified values for attached shadows
n9 = np.array([[-0.026519,0.999648]])
n10 = np.array([[-0.0816932, 0.996658]])
n11 =  np.array([[-0.0995431, 0.995033]])

p5 = np.array([[8.658104,3.7110704]])
p6 = np.array([[5.667369,3.5347379]])
p7 = np.array([[0.4301, -1.3138]])
```

[ ]:
```
# because the numpy utility is not useful
def dot_product(a, b):
    total = 0
    for i,j in zip(a,b):
        total += i*j
    return np.sum(total)
```

[ ]:
```
# define matrices
N_real= np.concatenate((n1, n2, n3, n4, n5, n6, n9, n10), axis=0)
```

```
N_fake = np.concatenate((n1, n2, n3, n4, n5, n6, n7, n8, n9, n10, n11), axis=0)

P_real = np.array([
    [dot_product(n1, p1)],
    [dot_product(n2, p1)],
    [dot_product(n3, p2)],
    [dot_product(n4, p2)],
    [dot_product(n5, p3)],
    [dot_product(n6, p3)],
    [dot_product(n9, p5)],
    [dot_product(n10, p6)]
])
P_fake = np.array([
    [dot_product(n1, p1)],
    [dot_product(n2, p1)],
    [dot_product(n3, p2)],
    [dot_product(n4, p2)],
    [dot_product(n5, p3)],
    [dot_product(n6, p3)],
    [dot_product(n7, p4)],
    [dot_product(n8, p4)],
    [dot_product(n9, p5)],
    [dot_product(n10, p6)],
    [dot_product(n11, p7)]
])
```

[ ]:
```
# make P shape (x,) instead of (x,1)
P_real = P_real.reshape(8,)
P_fake = P_fake.reshape(11,)
```

[ ]:
```
# real case
t = cp.Variable(8)
x = cp.Variable(2)
objective = cp.Minimize(cp.sum(t))
constraints = [-t<=N_real@x-P_real, N_real@x-P_real<=t]
prob = cp.Problem(objective, constraints)
optimal_value = prob.solve()
print("x=",x.value)
print("s=",optimal_value)
```

```
x= [4.55047994 0.69591558]
s= 6.276697040643973
```

[ ]:
```
# real case - reversed
t = cp.Variable(8)
x = cp.Variable(2)
objective = cp.Minimize(cp.sum(t))
```

```
constraints = [-t<=P_real-N_real@x, P_real-N_real@x<=t]
prob = cp.Problem(objective, constraints)
optimal_value = prob.solve()
print("x=",x.value)
print("s=",optimal_value)
```

```
x= [4.55047994 0.69591558]
s= 6.276697040643964
```

[ ]:
```
# fake case
t = cp.Variable(11)
x = cp.Variable(2)
objective = cp.Minimize(cp.sum(t))
constraints = [-t<=N_fake@x-P_fake, N_fake@x-P_fake<=t]
prob = cp.Problem(objective, constraints)
optimal_value = prob.solve()
print("x=",x.value)
print("s=",optimal_value)
```

```
x= [6.03900017 0.80790002]
s= 11.83679820698165
```

[ ]:
```
# fake case - reversed
t = cp.Variable(11)
x = cp.Variable(2)
objective = cp.Minimize(cp.sum(t))
constraints = [-t<=P_fake-N_fake@x, P_fake-N_fake@x<=t]
prob = cp.Problem(objective, constraints)
optimal_value = prob.solve()
print("x=",x.value)
print("s=",optimal_value)
```

```
x= [6.03900017 0.80790002]
s= 11.836798206981651
```

11.2 Solving for Shading Constraints Using SVD

Table of Contents

..... 1
3D normals Box 1
Object Normal Matrices 2
Box Calculations 2
Constraints Box 2
Normal Vectors Ball 3
Ball Constraints 4
Normal Vectors Book 4
Constraints for book 5
Normal vecs for heel 5


```
clear
clc
close all
```

3D normals Box

```
f = 24;
x1 = [-85.46; 161.10];
x1 = x1/norm(x1);
n1 = [.20867 .977986 (1/f)*([.20867 .977986]*x1)];

x2 = [-72;137];
x2 = x2/norm(x2);
n2 = [.172534 .985003 (1/f)*([.172534 .985003]*x2)];

x3 = [-59.09;111.39];
x3 = x3/norm(x3);
n3 = [.388549 .921428 (1/f)*([.388549 .921428]*x3)];

x4 = [-66.92;68];
x4 = x4/norm(x4);
n4 = [.185988 .982552 (1/f)*([.185988 .982552]*x4)];

x5 = [-80; 28];
x5 = x5/norm(x5);
n5 = [.180088 .983651 (1/f)*([.180088 .983651]*x5)];

x6 = [-90;2];
x6 = x6/norm(x6);
n6 = [.1113 .993787 (1/f)*([.1113 .993787]*x6)];

x7 = [-90;-38];
x7 = x7/norm(x7);
n7 = [.239325 -.970939 (1/f)*([.239325 -.970939]*x7)];
x8 = [-90; -83];
```

```
x8 = x8/norm(x8);
n8 = [.935695 .35281 (1/f)*([.935695 .35281]*x8)];
```

Object Normal Matrices

```
n1 = n1/norm(n1);

n2 = n2/norm(n2);

n3 = n3/norm(n3);

n4 = n4/norm(n4);

n5 = n5/norm(n5);

n6 = n6/norm(n6);

n7 = n7/norm(n7);

n8 = n8/norm(n8);
oneVec = ones(8,1);

n_box = [n1;n2;n3;n4;n5;n6;n7;n8];
N_box = [n_box oneVec];


%r = [0.2745098039 0.6745098039 0.8745098039 0.4745098039 0.1745098039
0.0745098039 0.9745098039 0.7745098039]';
r_box = [0.2745098039 0.2745098039 0.2745098039 0.2745098039 0.2745098039
0.2745098039 0.2745098039 0.2745098039]';
```

Box Calculations

```
[U_box, S_box, VT_box] = svd(N_box);

Nplus_box = pinv(N_box);
e0_box = Nplus_box*r_box;
s0_box = e0_box(1:3,1);

V_box = VT_box';
W_box = V_box(1:3,:);

w4_box = W_box(:,4);
```

Constraints Box

```
%Intrinsic Camera Matrix
f = 24; %focal length in mm ~ adjustable parameter
cx = 0; %image center x coord mm
cy = 0; %image center y coord mm
K = [f 0 -cx;0 f -cy;0 0 -1];
```

```
% Box
stilde_box = K*s0_box


stilde_box =

   1.0e-14 *

    0.2665
    0.3039
   -0.0444
```

Normal Vectors Ball

```
x1 = [711; -30];
x1 = x1/norm(x1);
n1 = [.09292 .995673 (1/f)*([.09292 .995673]*x1)];

x2 = [730.30;-43.3];
x2 = x2/norm(x2);
n2 = [.121927 .992539 (1/f)*([.121927 .992539]*x2)];

x3 = [737;-54];
x3 = x3/norm(x3);
n3 = [.137655 0.99048 (1/f)*([.137655 0.99048]*x3)];

x4 = [745;-75];
x4 = x4/norm(x4);
n4 = [0.181355 0.983418 (1/f)*([0.181355 0.983418]*x4)];

x5 = [746; -92];
x5 = x5/norm(x5);
n5 = [.177949 0.98404 (1/f)*([.177949 0.98404]*x5)];

x6 = [741;-109];
x6 = x6/norm(x6);
n6 = [0.207478 0.97824 (1/f)*([0.207478 0.97824]*x6)];

x7 = [735;-119];
x7 = x7/norm(x7);
n7 = [.806337 0.591456 (1/f)*([.806337 0.591456]*x7)];

x8 = [727; -128];
x8 = x8/norm(x8);
n8 = [0.130778 -0.991412 (1/f)*([0.130778 -0.991412]*x8)];

n_ball = [n1;n2;n3;n4;n5;n6;n7;n8];
N_ball = [n_ball oneVec];

r_ball =
[0.4666666667;0.4666666667;0.4666666667;0.4666666667;0.4666666667;0.4666666667
;0.4666666667;0.4666666667];
```

Ball Constraints

```
[U_ball, S_ball, VT_ball] = svd(N_ball);

Nplus_ball = pinv(N_ball);

e0_ball = Nplus_ball*r_ball;
s0_ball = e0_ball(1:3,1);

V_ball = VT_ball';

stilde_ball = K*s0_ball


stilde_ball =

   1.0e-13 *

    0.2015
    0.0266
   -0.0266
```

Normal Vectors Book

```
x1 = [514; 175];
x1 = x1/norm(x1);
n1 = [0.999157 -0.042555 (1/f)*([0.999157 -0.042555]*x1)];

x2 = [511;132];
x2 = x2/norm(x2);
n2 = [0.999157 -0.042555 (1/f)*([0.999157 -0.042555]*x2)];

x3 = [511;87];
x3 = x3/norm(x3);
n3 = [0.999157 -0.042555 (1/f)*([0.999157 -0.042555]*x3)];

x4 = [511;42];
x4 = x4/norm(x4);
n4 = [0.999157 -0.042555 (1/f)*([0.999157 -0.042555]*x4)];

x5 = [503; 10];
x5 = x5/norm(x5);
n5 = [0.999157 -0.042555 (1/f)*([0.999157 -0.042555]*x5)];

x6 = [500;-4];
x6 = x6/norm(x6);
n6 = [0.999157 -0.042555 (1/f)*([0.999157 -0.042555]*x6)];

x7 = [506;-56];
x7 = x7/norm(x7);
n7 = [0.999157 -0.042555 (1/f)*([0.999157 -0.042555]*x7)];
```

```
x8 = [507; -94];
x8 = x8/norm(x8);
n8 = [0.999157 -0.042555 (1/f)*([0.999157 -0.042555]*x8)];

n_book = [n1;n2;n3;n4;n5;n6;n7;n8];
N_book = [n_book oneVec];

r_book =
[0.231372549;0.231372549;0.231372549;0.231372549;0.231372549;0.231372549;0.231
372549;0.231372549];
```

Constraints for book

```
[U_book, S_book, VT_book] = svd(N_book);

% Two degen singular values

V_book = VT_book';

w4_book = V_book(1:3,4);
w3_book = V_book(1:3,3);

Nplus_book = pinv(N_book);
e0_book = Nplus_book*r_book;
s0_book = e0_book(1:3,1);

p_book = K*w4_book;

htilde = cross(K*w4_book,K*w3_book);

temp = htilde' * s0_book;

d_book = htilde;

%Half plane inequality: s (dot) d >= d (dot) p

LHS = d_book' * p_book;

% Our plane is 16.9537s_x + 0.4898s_y - 407.2807s_z
```

Normal vecs for heel

```
x1 = [186; 40];
x1 = x1/norm(x1);
n1 = [0.821522  0.570177 (1/f)*([0.821522  0.570177]*x1)];

x2 = [180;18];
x2 = x2/norm(x2);
n2 = [0.998265  -0.058889 (1/f)*([0.998265  -0.058889]*x2)];

x3 = [183;-7];
x3 = x3/norm(x3);
n3 = [0.363304  0.931671 (1/f)*([0.363304  0.931671]*x3)];
```

```
x4 = [181;-41];
x4 = x4/norm(x4);
n4 = [0.303697  0.952769 (1/f)*([0.303697  0.952769]*x4)];

x5 = [168; -72];
x5 = x5/norm(x5);
n5 = [0.70921  0.704998 (1/f)*([0.70921  0.704998]*x5)];

x6 = [160;-93];
x6 = x6/norm(x6);
n6 = [0.936248  -0.351339 (1/f)*([0.936248  -0.351339]*x6)];

x7 = [161;-105];
x7 = x7/norm(x7);
n7 = [0.721686  0.692221 (1/f)*([0.721686  0.692221]*x7)];

x8 = [166; -122];
x8 = x8/norm(x8);
n8 = [0.993827  -0.110939 (1/f)*([0.993827  -0.110939]*x8)];

n_heel = [n1;n2;n3;n4;n5;n6;n7;n8];
N_heel = [n_heel oneVec];

r_heel =
[0.1843137255;0.1843137255;0.1843137255;0.1843137255;0.1843137255;0.1843137255
;0.1843137255;0.1843137255];

[U_heel,S_heel,VT_heel] = svd(N_heel);

% No DSV

% Heel Constraints
Nplus_heel = pinv(N_heel);

e0_heel = Nplus_heel*r_heel;
s0_heel = e0_heel(1:3,1);

stilde_heel = K*s0_heel


stilde_heel =

   1.0e-14 *

    0.4330
    0.1998
   -0.0444
```

Published with MATLAB® R2023b

11.3 Results for the Combined System

combined_constraints

July 22, 2024

[ ]:
```
# req. packages
import numpy as np
import cvxpy as cp
```

[ ]:
```
# define manually specified values for cast shadows
n1 = np.array([[-0.139038,0.990287]])
n2 = np.array([[0.0167039,0.99986]])
n3 = np.array([[-.110262,0.99393]])
n4 = np.array([[0.0443534,0.999016]])
n5 = np.array([[-0.0900761, 0.995935]])
n6 = np.array([[.00868626, -0.999962]])
n7 = np.array([[-0.118596, 0.992943]])
n8 = np.array([[0.185023, 0.982734]])

p1 = np.array([[4.43,0.679]])
p2 = np.array([[6.039,0.8079]])
p3 = np.array([[9.264,0.73686]])
p4 = np.array([[7.1304347,2.7913044]])
```

[ ]:
```
# define manually specified values for attached shadows
n9 = np.array([[-0.026519,0.999648]])
n10 = np.array([[-0.0816932, 0.996658]])
n11 =  np.array([[-0.0995431, 0.995033]])

p5 = np.array([[8.658104,3.7110704]])
p6 = np.array([[5.667369,3.5347379]])
p7 = np.array([[0.4301, -1.3138]])
```

[ ]:
```
# define manulaly specified values for shading constraint

n12 = np.array([[16.9537,0.4898]])

p8 = np.array([[-16.9630, -0.4901]])
```

[ ]:
```
# because the numpy utility is not useful
def dot_product(a, b):
    total = 0
```

```
    for i,j in zip(a,b):
        total += i*j
    return np.sum(total)
```

[ ]:
```
# define matrices
N_real= np.concatenate((n1, n2, n3, n4, n5, n6, n9, n10, n12), axis=0)
N_fake = np.concatenate((n1, n2, n3, n4, n5, n6, n7, n8, n9, n10, n11, n12),␣
 ↪axis=0)

P_real = np.array([
    [dot_product(n1, p1)],
    [dot_product(n2, p1)],
    [dot_product(n3, p2)],
    [dot_product(n4, p2)],
    [dot_product(n5, p3)],
    [dot_product(n6, p3)],
    [dot_product(n9, p5)],
    [dot_product(n10, p6)],
    [dot_product(n12, p8)]
])
P_fake = np.array([
    [dot_product(n1, p1)],
    [dot_product(n2, p1)],
    [dot_product(n3, p2)],
    [dot_product(n4, p2)],
    [dot_product(n5, p3)],
    [dot_product(n6, p3)],
    [dot_product(n7, p4)],
    [dot_product(n8, p4)],
    [dot_product(n9, p5)],
    [dot_product(n10, p6)],
    [dot_product(n11, p7)],
    [dot_product(n12, p8)]
])
```

[ ]:
```
# make P shape (x,) instead of (x,1)
P_real = P_real.reshape(9,)
P_fake = P_fake.reshape(12,)
```

[ ]:
```
# real case
t = cp.Variable(9)
x = cp.Variable(2)
objective = cp.Minimize(cp.sum(t))
constraints = [-t<=N_real@x-P_real, N_real@x-P_real<=t]
prob = cp.Problem(objective, constraints)
optimal_value = prob.solve()
print("x=",x.value)
```

```
print("s=",optimal_value)
```

```
x= [-17.00712246   1.03713351]
s= 12.720547507398
```

[ ]:
```
# real case - reversed
t = cp.Variable(9)
x = cp.Variable(2)
objective = cp.Minimize(cp.sum(t))
constraints = [-t<=P_real-N_real@x, P_real-N_real@x<=t]
prob = cp.Problem(objective, constraints)
optimal_value = prob.solve()
print("x=",x.value)
print("s=",optimal_value)
```

```
x= [-17.00712246   1.03713351]
s= 12.720547507397962
```

[ ]:
```
# fake case
t = cp.Variable(12)
x = cp.Variable(2)
objective = cp.Minimize(cp.sum(t))
constraints = [-t<=N_fake@x-P_fake, N_fake@x-P_fake<=t]
prob = cp.Problem(objective, constraints)
optimal_value = prob.solve()
print("x=",x.value)
print("s=",optimal_value)
```

```
x= [-16.99185827   0.50878614]
s= 23.572368793023745
```

[ ]:
```
# fake case - reversed
t = cp.Variable(12)
x = cp.Variable(2)
objective = cp.Minimize(cp.sum(t))
constraints = [-t<=P_fake-N_fake@x, P_fake-N_fake@x<=t]
prob = cp.Problem(objective, constraints)
optimal_value = prob.solve()
print("x=",x.value)
print("s=",optimal_value)
```

```
x= [-16.99185827   0.50878614]
s= 23.57236879302376
```