Image composition for photographic lighting

Tjaart van der Walt

An implementation based on the paper
User-assisted Image Compositing for Photographic Lighting
by Ivaylo Boyadzhiev et al.

2015-04-22

The basic idea

Imagine that we want to take a studio photo of a static scene.

The basic idea

Imagine that we want to take a studio photo of a static scene.

Traditional workflow

Imagine that we want to take a studio photo of a static scene.

Traditional workflow

- Carefully configure the lighting setup

Imagine that we want to take a studio photo of a static scene.

Traditional workflow

- Carefully configure the lighting setup
- Take the photo

Imagine that we want to take a studio photo of a static scene.

Traditional workflow

- Carefully configure the lighting setup
- Take the photo
- Do post processing (mostly non lighting oriented)

Imagine that we want to take a studio photo of a static scene.

Traditional workflow

- Carefully configure the lighting setup
- Take the photo
- Do post processing (mostly non lighting oriented)

The proposed worklow

The basic idea

Imagine that we want to take a studio photo of a static scene.

Traditional workflow

- Carefully configure the lighting setup
- Take the photo
- Do post processing (mostly non lighting oriented)

The proposed worklow

- Take multiple photos from a fixed vantage point

The basic idea

Imagine that we want to take a studio photo of a static scene.

Traditional workflow

- Carefully configure the lighting setup
- Take the photo
- Do post processing (mostly non lighting oriented)

The proposed worklow

- Take multiple photos from a fixed vantage point
- Move the light source between each shot

Imagine that we want to take a studio photo of a static scene.

Traditional workflow

- Carefully configure the lighting setup
- Take the photo
- Do post processing (mostly non lighting oriented)

The proposed worklow

- Take multiple photos from a fixed vantage point
- Move the light source between each shot
- Process the resulting images to form one well lit photo

How do we achieve this?

We define 3 Basis Lights and 3 Light Modifiers to create our well lit image

How do we achieve this?

We define 3 Basis Lights and 3 Light Modifiers to create our well lit image

Basis lights

How do we achieve this?

We define 3 Basis Lights and 3 Light Modifiers to create our well lit image

Basis lights

- Fill light ensures even illumination in the image

How do we achieve this?

We define 3 Basis Lights and 3 Light Modifiers to create our well lit image

Basis lights

- Fill light ensures even illumination in the image
- Edge light emphasizes edges in the image

How do we achieve this?

We define 3 Basis Lights and 3 Light Modifiers to create our well lit image

Basis lights

- Fill light ensures even illumination in the image
- Edge light emphasizes edges in the image
- Diffuse color light emphasizes the underlying colors

How do we achieve this?

We define 3 Basis Lights and 3 Light Modifiers to create our well lit image

Basis lights

- Fill light ensures even illumination in the image
- Edge light emphasizes edges in the image
- Diffuse color light emphasizes the underlying colors

Modifiers

How do we achieve this?

We define 3 Basis Lights and 3 Light Modifiers to create our well lit image

Basis lights

- Fill light ensures even illumination in the image
- Edge light emphasizes edges in the image
- Diffuse color light emphasizes the underlying colors

Modifiers

- Per-object modifier emphasize a particular object

How do we achieve this?

We define 3 Basis Lights and 3 Light Modifiers to create our well lit image

Basis lights

- Fill light ensures even illumination in the image
- Edge light emphasizes edges in the image
- Diffuse color light emphasizes the underlying colors

Modifiers

- Per-object modifier emphasize a particular object
- Soft lighting modifier simulate area lights that produce soft shadows and highlights

How do we achieve this?

We define 3 Basis Lights and 3 Light Modifiers to create our well lit image

Basis lights

- Fill light ensures even illumination in the image
- Edge light emphasizes edges in the image
- Diffuse color light emphasizes the underlying colors

Modifiers

- Per-object modifier emphasize a particular object
- Soft lighting modifier simulate area lights that produce soft shadows and highlights
- Regional lighting modifier balance the lighting across the image

Input images

Bowl of fruit

000.png

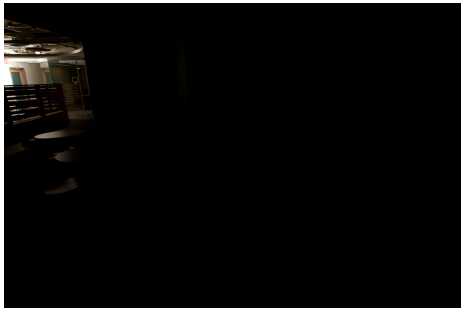
001.png

. . .

127.png

Café

000.png



001.png

. . .

112.png

Our implementation is done in Python.

```
import cv2
import numpy as np

from scipy.optimize import minimize

# This will possibly give us better performance
# import pyipopt
```

Fill light (idea)

Fill light tries to give even illumination everywhere in the input image.

Fill light (idea)

Fill light tries to give even illumination everywhere in the input image.

Assumption

We assume that our input lights will be approximately uniformly distributed across the input images .

Fill light (model)

$$I_{\text{fill}}(p) = \frac{\sum_i w_i(p) I_i(p)}{\sum_i w_i(p)}$$

$$I_{\text{fill}}(p) = \frac{\sum_i w_i(p) I_i(p)}{\sum_i w_i(p)}$$

Where

$$w_i(p) = \frac{\overline{I}_i(p)}{\overline{I}_i(p) + \epsilon}$$

$$I_{\text{fill}}(p) = \frac{\sum_i w_i(p) I_i(p)}{\sum_i w_i(p)}$$

Where

$$w_i(p) = \frac{\overline{I}_i(p)}{\overline{I}_i(p) + \epsilon}$$

$$\overline{I}_i(p) = I_i(p) \cdot (0.2990, 0.5870, 0.1140)$$

$$I_{\text{fill}}(p) = \frac{\sum_i w_i(p) I_i(p)}{\sum_i w_i(p)}$$

Where

$$w_i(p) = \frac{\overline{I}_i(p)}{\overline{I}_i(p) + \epsilon}$$

$$\overline{I}_i(p) = I_i(p) \cdot (0.2990, 0.5870, 0.1140)$$

$$\epsilon = 0.01$$

Fill light (implementation)

```
# Same shape as input images, but only 1 channel
sum_weights = np.zeros((rows, cols, 1))
i_bar = np.zeros_like(sum_weights)

for i, image in enumerate(self.image_list):
    weights = np.zeros_like(sum_weights)
    rgb_vector = np.array([0.114, 0.587, 0.299])
    i_bar = np.dot(image, rgb_vector)
    i_bar = np.reshape(i_bar, (rows, cols, 1))
    weights = i_bar/(i_bar + epsilon)

    sum_weights += weights
    sum_weights_image += (image * weights)

return sum_weights_image/sum_weights
```

Fill light

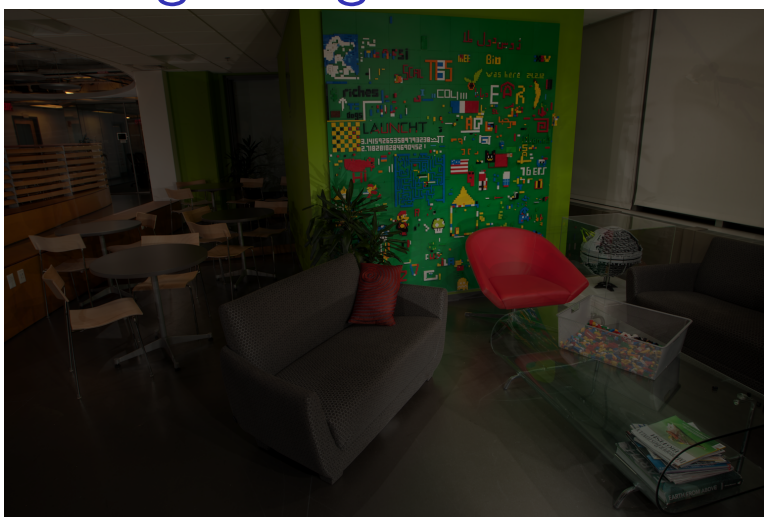
Fill light (results)

Average image

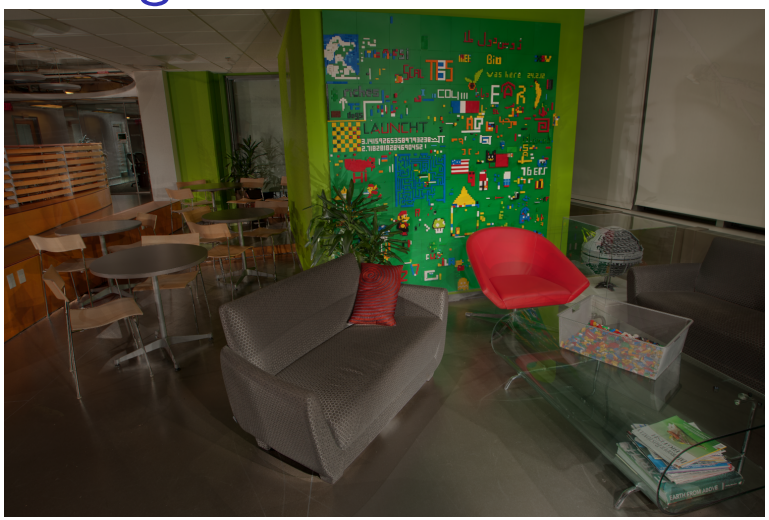
Fill light

Fill light (results)

Average image



Fill light



Edge light (idea)

Edge light tries to emphasize the main edges in a scene.

Edge light (idea)

Edge light tries to emphasize the main edges in a scene.

Key insight

We leverage the fact that the main edges of the scene will always be located in the same place in the image, whereas occasional shadows and highlights will be sporadically distributed over the input images.

Edge light (model)

$$I_{\text{edge}} = \sum_i \lambda_i I_i$$

$$I_{\text{edge}} = \sum_i \lambda_i I_i$$

Where

$$\text{arg min}_{\{\lambda_i\}} \sum_p h(p) ||\nabla \left(\sum_i \lambda_i I_i(p) \right) - G(p)||^2$$

$$I_{\text{edge}} = \sum_i \lambda_i I_i$$

Where

$$\text{arg min}_{\{\lambda_i\}} \sum_p h(p) ||\nabla \left(\sum_i \lambda_i I_i(p) \right) - G(p)||^2$$

G is the gradient map of the combined input images.

$$I_{\text{edge}} = \sum_i \lambda_i I_i$$

Where

$$\text{arg min}_{\{\lambda_i\}} \sum_p h(p) || \nabla \left(\sum_i \lambda_i I_i(p) \right) - G(p) ||^2$$

$h(p)$ is the per pixel weight designed to give more weight to pixels with a well defined orientation.

Gradient Orientation

We use the Sobel mask to get the gradient orientations.

Gradient Orientation

We use the Sobel mask to get the gradient orientations.

Code

```
sx = cv2.Sobel(image, cv2.CV_32F, 1, 0, ksize=-1)
sy = cv2.Sobel(image, cv2.CV_32F, 0, 1, ksize=-1)

mag = cv2.magnitude(sx, sy)
phase = cv2.phase(sx, sy, angleInDegrees=True)
```

Gradient Orientation

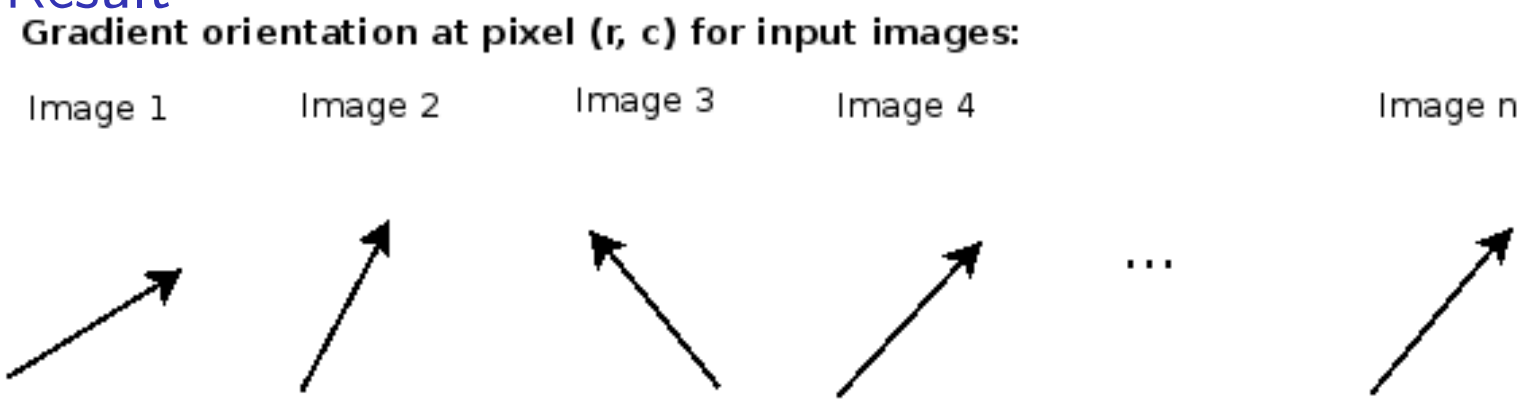
We use the Sobel mask to get the gradient orientations.

Code

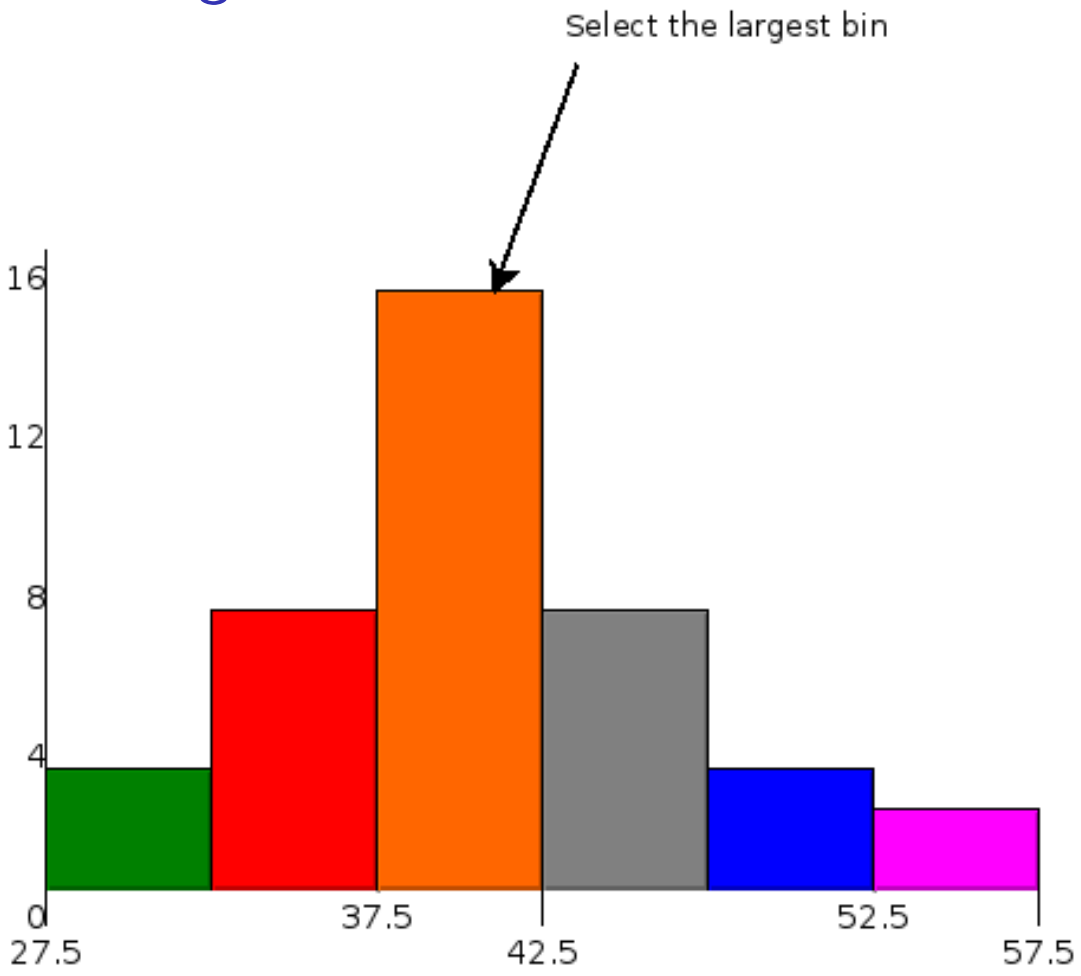
```
sx = cv2.Sobel(image, cv2.CV_32F, 1, 0, ksize=-1)
sy = cv2.Sobel(image, cv2.CV_32F, 0, 1, ksize=-1)

mag = cv2.magnitude(sx, sy)
phase = cv2.phase(sx, sy, angleInDegrees=True)
```

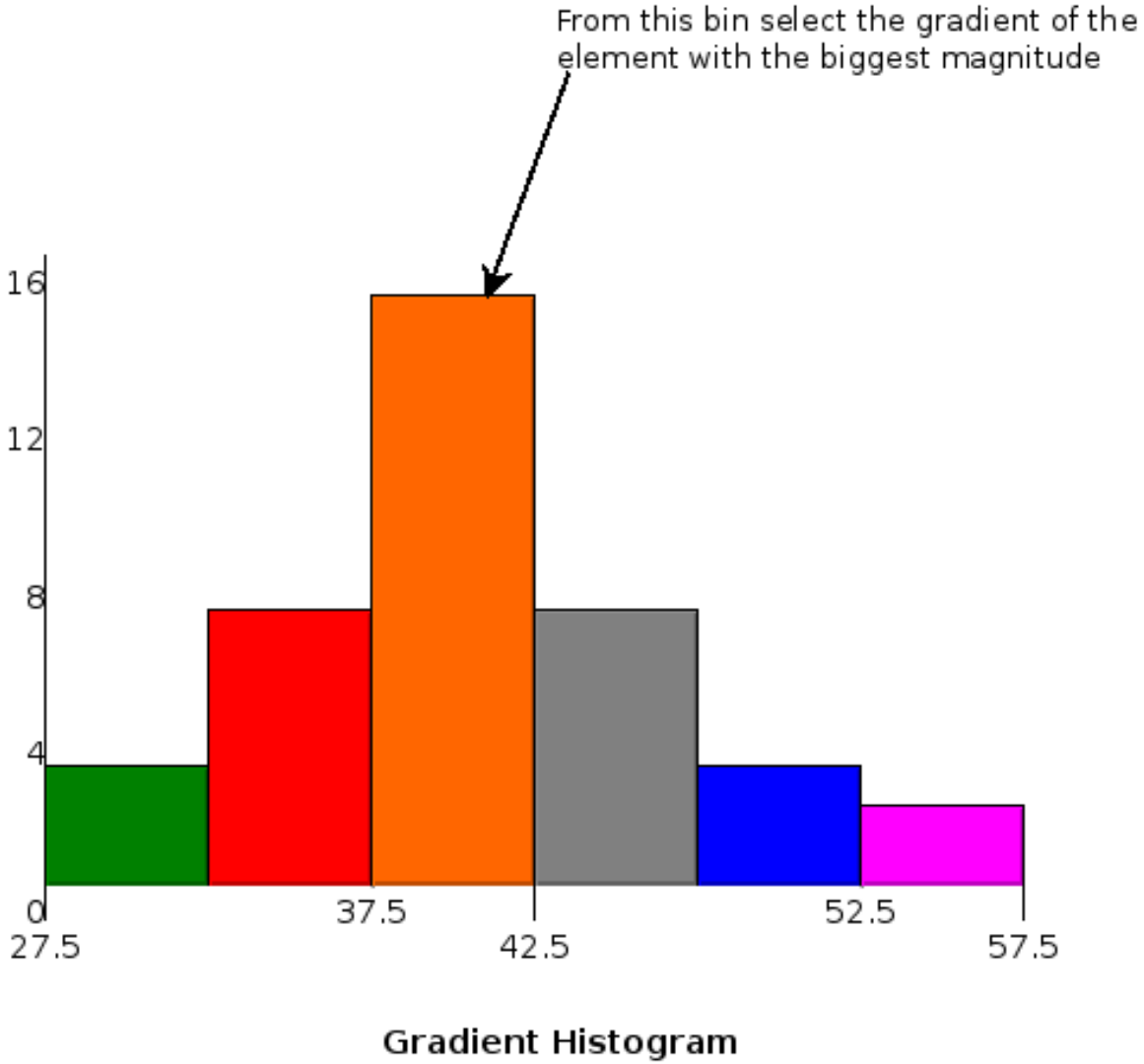
Result



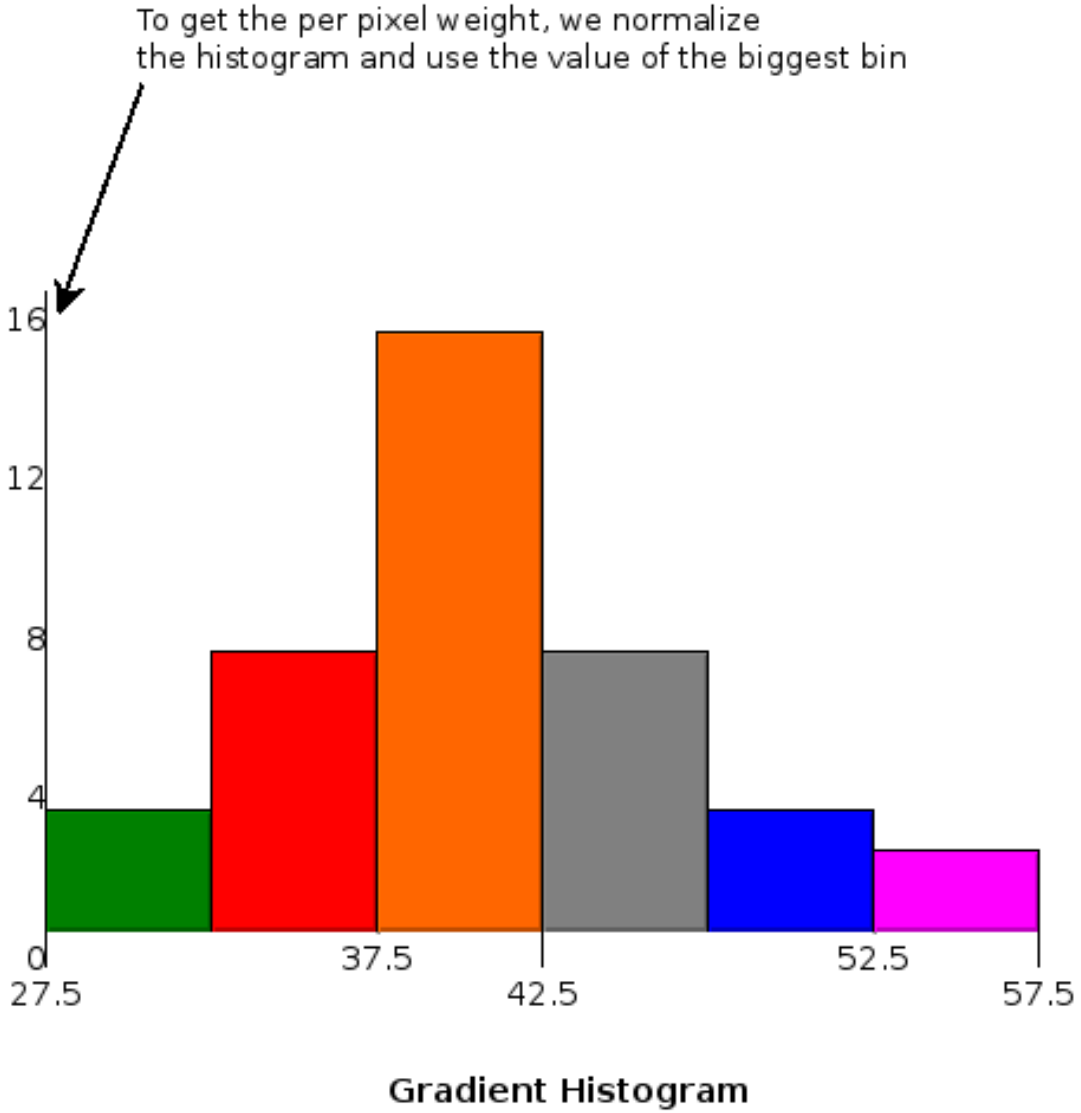
Gradient Histogram



Gradient map entry



Per pixel weights



Edge light (implementation)

Optimizer

Optimizer

Currently we use the *scipy.optimize* framework to find the λ values. It supports multiple optimization methods, we have chosen *Nelder-Mead*.

Optimizer

Currently we use the *scipy.optimize* framework to find the λ values. It supports multiple optimization methods, we have chosen *Nelder-Mead*.

Code

```
x0 = np.full(N, 1.0/N)
lambdas = minimize(edge_color, x0, method='Nelder-Mead')
```

Optimizer

Currently we use the *scipy.optimize* framework to find the λ values. It supports multiple optimization methods, we have chosen *Nelder-Mead*.

Code

```
x0 = np.full(N, 1.0/N)
lambdas = minimize(edge_color, x0, method='Nelder-Mead')
```

Advantage: Implementation simplicity.

Optimizer

Currently we use the *scipy.optimize* framework to find the λ values. It supports multiple optimization methods, we have chosen *Nelder-Mead*.

Code

```
x0 = np.full(N, 1.0/N)
lambdas = minimize(edge_color, x0, method='Nelder-Mead')
```

Advantage: Implementation simplicity.
Disadvantage: Takes long to converge.

Improving the Optimizer

Improving the Optimizer

To improve our current situation we intend to implement an Interior Point optimizer. One candidate framework we are investigating is *Pylpopt*.

Improving the Optimizer

To improve our current situation we intend to implement an Interior Point optimizer. One candidate framework we are investigating is *Pylpopt*.

Code

```
def fprime():
    pass

x0 = np.full(N, 1.0/N)
lambdas = pyipopt.fmin_unconstrained(edge_color,
                                     x0, fprime)
```

Improving the Optimizer

To improve our current situation we intend to implement an Interior Point optimizer. One candidate framework we are investigating is *Pylpopt*.

Code

```
def fprime():
    pass

x0 = np.full(N, 1.0/N)
lambdas = pyipopt.fmin_unconstrained(edge_color,
                                     x0, fprime)
```

Outstanding issues:

Improving the Optimizer

To improve our current situation we intend to implement an Interior Point optimizer. One candidate framework we are investigating is *Pylpopt*.

Code

```
def fprime():
    pass

x0 = np.full(N, 1.0/N)
lambdas = pyipopt.fmin_unconstrained(edge_color,
                                     x0, fprime)
```

Outstanding issues:

- How to calculate the gradient?

Improving the Optimizer

To improve our current situation we intend to implement an Interior Point optimizer. One candidate framework we are investigating is *Pylpopt*.

Code

```
def fprime():
    pass

x0 = np.full(N, 1.0/N)
lambdas = pyipopt.fmin_unconstrained(edge_color,
                                     x0, fprime)
```

Outstanding issues:

- How to calculate the gradient?
- Could we define some constraints?

Edge light (results)

Fill light

† Result from paper

Edge light

Diffuse color light (idea)

Diffuse color light tries to emphasize the base color of objects.

Diffuse color light (model)

We handle colorful objects and neutral colored objects seperately

We handle colorful objects and neutral colored objects seperately

Colorful objects

We define a function that favors diffuse reflection instead of the specular reflection.

We handle colorful objects and neutral colored objects seperately

Colorful objects

We define a function that favors diffuse reflection instead of the specular reflection.

Wikipedia definition

> *Diffuse reflection is the reflection of light from a surface such that an incident ray is reflected at many angles rather than at just one angle as in the case of specular reflection.*

We handle colorful objects and neutral colored objects seperately

Colorful objects

We define a function that favors diffuse reflection instead of the specular reflection.

Wikipedia definition

> *Diffuse reflection is the reflection of light from a surface such that an incident ray is reflected at many angles rather than at just one angle as in the case of specular reflection.*

Insight

The specular component is white. So the stronger it is the less colorful our object appears.

Pixel color

$$\mathbf{I} = d\mathbf{D} + s\mathbf{W}$$

Pixel color

$$\mathbf{I} = d\mathbf{D} + s\mathbf{W}$$

Saturation angle

The saturation angle is given by $\angle(I, W)$

Pixel color

$\mathbf{I} = d\mathbf{D} + s\mathbf{W}$

Saturation angle

The saturation angle is given by $\angle(I, W)$

Where

$\angle(C1, C2) = \arccos\left(\frac{C_1}{||C_1||} \cdot \frac{C_2}{||C_2||}\right)$ and $W = (1, 1, 1)^T$

Pixel color

$\mathbf{I} = d\mathbf{D} + s\mathbf{W}$

Saturation angle

The saturation angle is given by $\angle(I, W)$

Where

$\angle(C1, C2) = \arccos\left(\frac{C_1}{||C_1||} \cdot \frac{C_2}{||C_2||}\right)$ and $W = (1, 1, 1)^T$

Notice

For a fixed value d the angle decreases as s increases

Pixel color

$\mathbf{I} = d\mathbf{D} + s\mathbf{W}$

Saturation angle

The saturation angle is given by $\angle(I, W)$

Where

$\angle(C1, C2) = \arccos\left(\frac{C_1}{||C_1||} \cdot \frac{C_2}{||C_2||}\right)$ and $W = (1, 1, 1)^T$

Notice

For a fixed value d the angle decreases as s increases

Energy function

arg max$_{\{\lambda_i\}} \sum_p \hat{w}(p) \angle \left(\sum_i \lambda_i I_i(p), W\right)$

Pixel color

$\mathbf{I} = d\mathbf{D} + s\mathbf{W}$

Saturation angle

The saturation angle is given by $\angle(I, W)$

Where

$\angle(C1, C2) = \arccos\left(\frac{C_1}{||C_1||} \cdot \frac{C_2}{||C_2||}\right)$ and $W = (1, 1, 1)^T$

Notice

For a fixed value d the angle decreases as s increases

Energy function

arg max$_{\{\lambda_i\}} \sum_p \hat{w}(p) \angle \left(\sum_i \lambda_i I_i(p), W\right)$

Where

$\hat{w}(p) = \frac{\sum_i \lambda_i I_i(p)}{\sum_i \lambda_i I_i(p) + \epsilon}$

Neutral colored objects

We define a function that encourages similarity between the average image and our result.

Neutral colored objects

We define a function that encourages similarity between the average image and our result.

Energy function

$\arg\min_{\{\lambda_i\}} \sum_p \angle\left(\sum_i \lambda_i I_i, \frac{1}{N}\sum_i I_i\right)$

Neutral colored objects

We define a function that encourages similarity between the average image and our result.

Energy function

$\arg\min_{\{\lambda_i\}} \sum_p \angle\left(\sum_i \lambda_i I_i, \frac{1}{N}\sum_i I_i\right)$

To make sure we apply this only to neutral colors, we define a balancing term:

Neutral colored objects

We define a function that encourages similarity between the average image and our result.

Energy function

$\arg\min_{\{\lambda_i\}} \sum_p \angle\left(\sum_i \lambda_i I_i, \frac{1}{N}\sum_i I_i\right)$

To make sure we apply this only to neutral colors, we define a balancing term:

$$\alpha(p) = \exp\left(\frac{-\angle\left(\frac{1}{N}\sum_i I_i(p), W\right)^2}{2\sigma^2}\right)$$

Putting it all together

$I_{\text{diffuse}} = \sum_i \lambda_i I_i$

Putting it all together

$I_{\text{diffuse}} = \sum_i \lambda_i I_i$

Where

$$\arg\min_{\{\lambda_i\}} \sum_p \left[\alpha(p) \angle \left(\sum_i \lambda_i I_i(p), \frac{1}{N} \sum_i I_i(p) \right) - (1 - \alpha(p)) \hat{w}(p) \angle \left(\sum_i \lambda_i I_i(p), W \right) \right.$$

Diffuse color light (results)

Fill light

† Result from paper

Diffuse color light

Per-Object modifiers (idea)

The per-object modifier controls the spread of light. It is inspired by photographic equipment like a snoot.

Method

Method

- We compute the basis lights as discussed previously

Method

- We compute the basis lights as discussed previously
- We apply them to pixels within a selected object

Method

- We compute the basis lights as discussed previously
- We apply them to pixels within a selected object
- Users can mix these colors interactively

Method

- We compute the basis lights as discussed previously
- We apply them to pixels within a selected object
- Users can mix these colors interactively
- To ensure smooth blending with the rest of the image we apply a cross bilateral filter

Per-Object modifiers (results)

Soft light modifiers (idea)

The soft light modifier produces areas with soft shadows and highlights. It is inspired by photographic equipment like umbrellas and soft boxes

Soft light modifiers (model)

$$I_{\text{soft}} = \sum_i \lambda_i I_i$$

$$I_{\text{soft}} = \sum_i \lambda_i I_i$$

Where

$$\text{soft}_{\sigma_S}(\Lambda) = \frac{||\Lambda||}{||S\Lambda||} S\Lambda$$

$$I_{\text{soft}} = \sum_i \lambda_i I_i$$

Where

$$\text{soft}_{\sigma_S}(\Lambda) = \frac{||\Lambda||}{||S\Lambda||} S\Lambda$$

and

$$\Lambda = (\lambda_1, \ldots, \lambda_N)^T$$

$$I_{\text{soft}} = \sum_i \lambda_i I_i$$

Where

$$\text{soft}_{\sigma_S}(\Lambda) = \frac{||\Lambda||}{||S\Lambda||} S\Lambda$$

and

$$\Lambda = (\lambda_1, \ldots, \lambda_N)^T$$

$$S_{ij} = \exp\left(\frac{-||I_i - I_j||^2}{2\sigma_s^2}\right)$$

Soft light modifiers (implementation)

```
# The soft light modifier does not maintain the image
# intensity. We deal with this by converting to HSV
hsv_img = cv2.cvtColor(img, cv2.cv.CV_BGR2HSV)

for i in range(hsv_image.shape[0]):
    hsv_img[i][:, 2] = hsv_img[i][:, 2] + RESCALE

sl_img = cv2.cvtColor(hsv_image, cv2.cv.CV_HSV2BGR)
```

Soft light modifiers (results)

Fill light image

Soft light image

Soft light modifiers (results)

Fill light image

Soft light image

Regional lighting modifiers (idea)

The regional lighting modifier balances the light intensities in a scene to emphasize (or de-emphasize) a specific region in a scene. It provides a way to adjust the light balance across the scene at a coarse level.

Method

Method

- We are only interested in balancing the lighting, so we work with the intensity images $\bar{I}_i$

Method

- We are only interested in balancing the lighting, so we work with the intensity images $\bar{I}_i$
- We calculate the first component using Principle Component Analysis. This captures the different areas of illumination

Method

- We are only interested in balancing the lighting, so we work with the intensity images $\bar{I}_i$
- We calculate the first component using Principle Component Analysis. This captures the different areas of illumination
- We translate our image to the log domain $P = \ln(\bar{I}_i)$

Method

- We are only interested in balancing the lighting, so we work with the intensity images $\bar{I}_i$
- We calculate the first component using Principle Component Analysis. This captures the different areas of illumination
- We translate our image to the log domain $P = \ln(\bar{I}_i)$
- We ensure the overall image intensity remains unchanged by enforcing a zero mean on P: $\hat{P} = P - \frac{1}{N}\sum_p P(p)$

Method

- We are only interested in balancing the lighting, so we work with the intensity images $\bar{I}_i$
- We calculate the first component using Principle Component Analysis. This captures the different areas of illumination
- We translate our image to the log domain $P = \ln(\bar{I}_i)$
- We ensure the overall image intensity remains unchanged by enforcing a zero mean on P: $\hat{P} = P - \frac{1}{N}\sum_p P(p)$
- To remove undesirable boundaries from $\hat{P}$ we apply a bilateral filter

Method

- We are only interested in balancing the lighting, so we work with the intensity images $\bar{I}_i$
- We calculate the first component using Principle Component Analysis. This captures the different areas of illumination
- We translate our image to the log domain $P = \ln(\bar{I}_i)$
- We ensure the overall image intensity remains unchanged by enforcing a zero mean on P: $\hat{P} = P - \frac{1}{N}\sum_p P(p)$
- To remove undesirable boundaries from $\hat{P}$ we apply a bilateral filter
- We create a map $M = e^{(\beta \hat{P})}$

Method

- We are only interested in balancing the lighting, so we work with the intensity images $\bar{I}_i$
- We calculate the first component using Principle Component Analysis. This captures the different areas of illumination
- We translate our image to the log domain $P = \ln(\bar{I}_i)$
- We ensure the overall image intensity remains unchanged by enforcing a zero mean on P: $\hat{P} = P - \frac{1}{N}\sum_p P(p)$
- To remove undesirable boundaries from $\hat{P}$ we apply a bilateral filter
- We create a map $M = e^{(\beta \hat{P})}$
- Our modifier is defined by: $\mathsf{I}_{\text{regional}} = \frac{1}{N}\sum_i M I_i$

Method (cont.)

We observe that if:

Method (cont.)

We observe that if:

- $\beta = 0$ the result remains unchanged

Method (cont.)

We observe that if:

- $\beta = 0$ the result remains unchanged
- $\beta > 0$ emphasizes regions where $\hat{P} > 0$

Method (cont.)

We observe that if:

- $\beta = 0$ the result remains unchanged
- $\beta > 0$ emphasizes regions where $\hat{P} > 0$
- $\beta < 0$ emphasizes regions where $\hat{P} < 0$

Regional lighting modifiers (implementation)

```
# We use the HSV image because we only want to
# work with intensities, not colors
hsv_image = cv2.cvtColor(img, cv2.cv.CV_BGR2HSV)
...
# We calculate the mean image for the first principle
# component of our logarithmic intensity image
mean, e = cv2.PCACompute(log_row, maxComponents=1)
...
# We apply a bilateral filter to remove undesirable
# shadow boundaries. The parameter values are the
# suggested values from the OpenCV documentation
pca_hat = cv2.bilateralFilter(pca_hat, 5, 50, 50)
...
```

Regional lighting modifiers (implementation)

```
# We calculate our map
for r in range(pca_hat.shape[0]):
    for c in range(pca_hat.shape[1]):
        my_map.itemset(r, c, np.exp(beta *
                                    pca_hat.item(r, c)))
...
# We multiply our map with the intensity image, merge
# it back to the HSV image, and convert it back to RGB
new_intensity_img = np.multiply(my_map, intensity_image)
```

Regional lighting modifiers (results)

$\beta > 0$

$\beta < 0$

Improvements to our current implementation

Optimization algorithm

As discussed earlier in this talk, we could substantially improve our optimization algorithm by:

Optimization algorithm

As discussed earlier in this talk, we could substantially improve our optimization algorithm by:

- Using an interior point optimization method

Optimization algorithm

As discussed earlier in this talk, we could substantially improve our optimization algorithm by:

- Using an interior point optimization method
- Using GPU calculations to improve NumPy performance

Optimization algorithm

As discussed earlier in this talk, we could substantially improve our optimization algorithm by:

- Using an interior point optimization method
- Using GPU calculations to improve NumPy performance

Graphical user interface

We could implement a GUI were a user could interactively mix the different components discussed. This will require that at least the light modifiers can be calculated in real time.

Thank you

Thank you for your attention!