CS 445: Computational Photography

Programming Project #3: Gradient Domain Fusion

In [1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import os
from random import random
import time
import scipy
import scipy.sparse.linalg
import utils
```

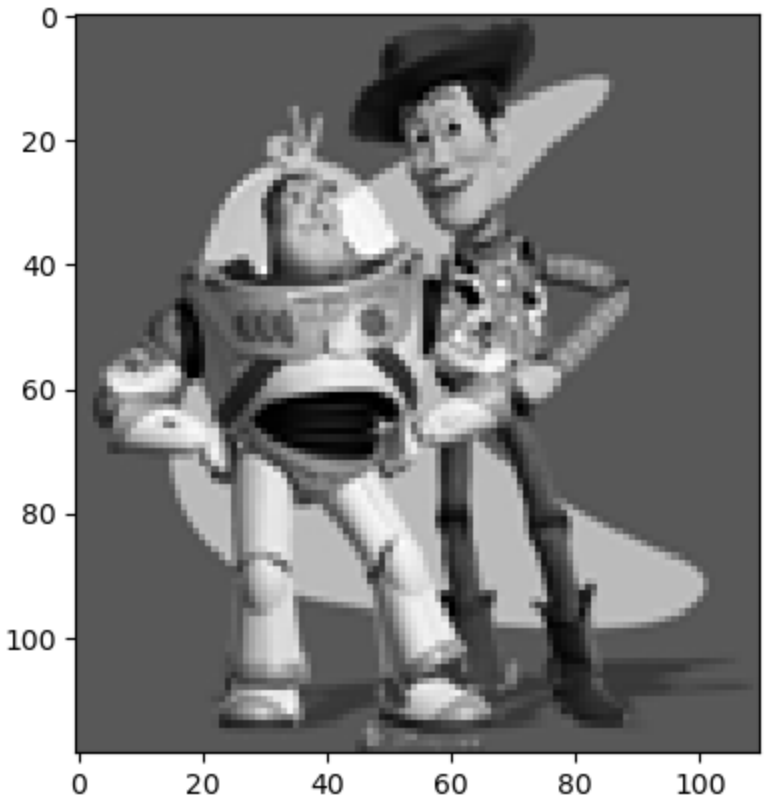
Part 1 Toy Problem (20 pts)

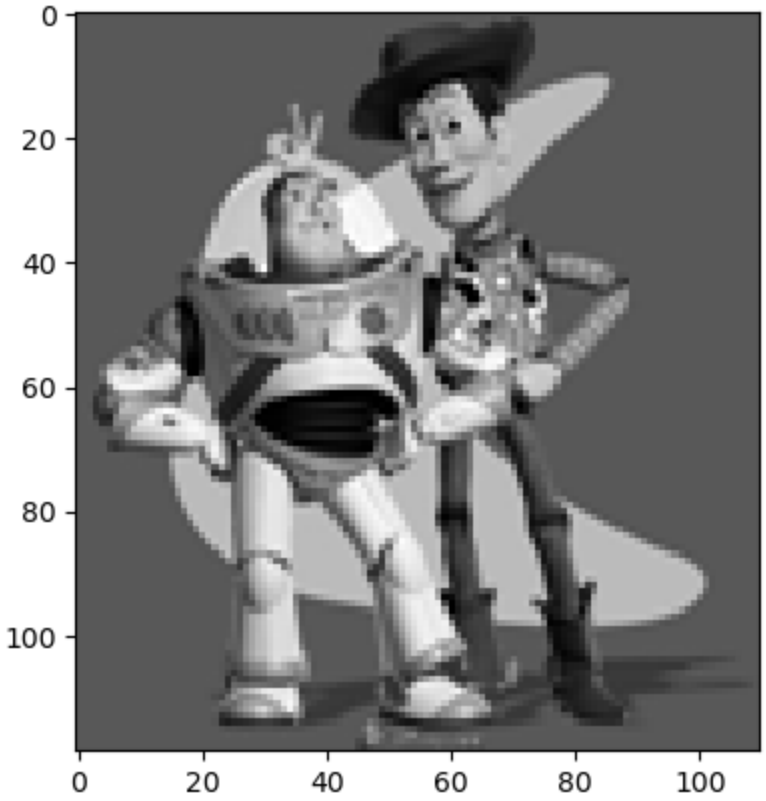
In [15]:
```
def toy_reconstruct(img):
    """
    The implementation for gradient domain processing is not complicated, but it is easy
    Reconstruct this image from its gradient values, plus one pixel intensity.
    Denote the intensity of the source image at (x, y) as s(x,y) and the value to solve
    1. minimize (v(x+1,y)-v(x,y) - (s(x+1,y)-s(x,y)))^2
    2. minimize (v(x,y+1)-v(x,y) - (s(x,y+1)-s(x,y)))^2
    Note that these could be solved while adding any constant value to v, so we will add
    3. minimize (v(1,1)-s(1,1))^2

    :param toy_img: numpy.ndarray
    """
    im_h, im_w = img.shape
    im2var = np.arange(im_h*im_w).reshape(im_h, im_w) # map each pixel to a variable num

    m = im_h*(im_w-1) + (im_h-1)*im_w + 1 # number of equations
    n = im_h*im_w # number of variables
    A = scipy.sparse.lil_matrix((m, n), dtype='double')
    b = np.zeros((m, 1), dtype='double')

    e = 0 # equation index

    # objective 1
    for y in range(im_h):
        for x in range(im_w-1):
            A[e, im2var[y, x+1]] = 1
            A[e, im2var[y, x]] = -1
            b[e] = img[y, x+1] - img[y, x]
            e += 1

    # objective 2
    for y in range(im_h-1):
        for x in range(im_w):
            A[e, im2var[y+1, x]] = 1
            A[e, im2var[y, x]] = -1
            b[e] = img[y+1, x] - img[y, x]
            e += 1

    # objective 3
    A[e, im2var[0, 0]] = 1
    b[e] = img[0, 0]
```

```
    v = scipy.sparse.linalg.lsqr(A.tocsr(), b, atol=10**-15, btol=10**-15)[0]
    v = v.reshape(im_h, im_w)

    return v
```

In [16]:

```
toy_img = cv2.cvtColor(cv2.imread('samples/toy_problem.png'), cv2.COLOR_BGR2GRAY).astype
plt.imshow(toy_img, cmap="gray")
plt.show()

im_out = toy_reconstruct(toy_img)
plt.imshow(im_out, cmap="gray")
plt.show()
print("Max error is: ", np.sqrt(((im_out - toy_img)**2).max()))
```





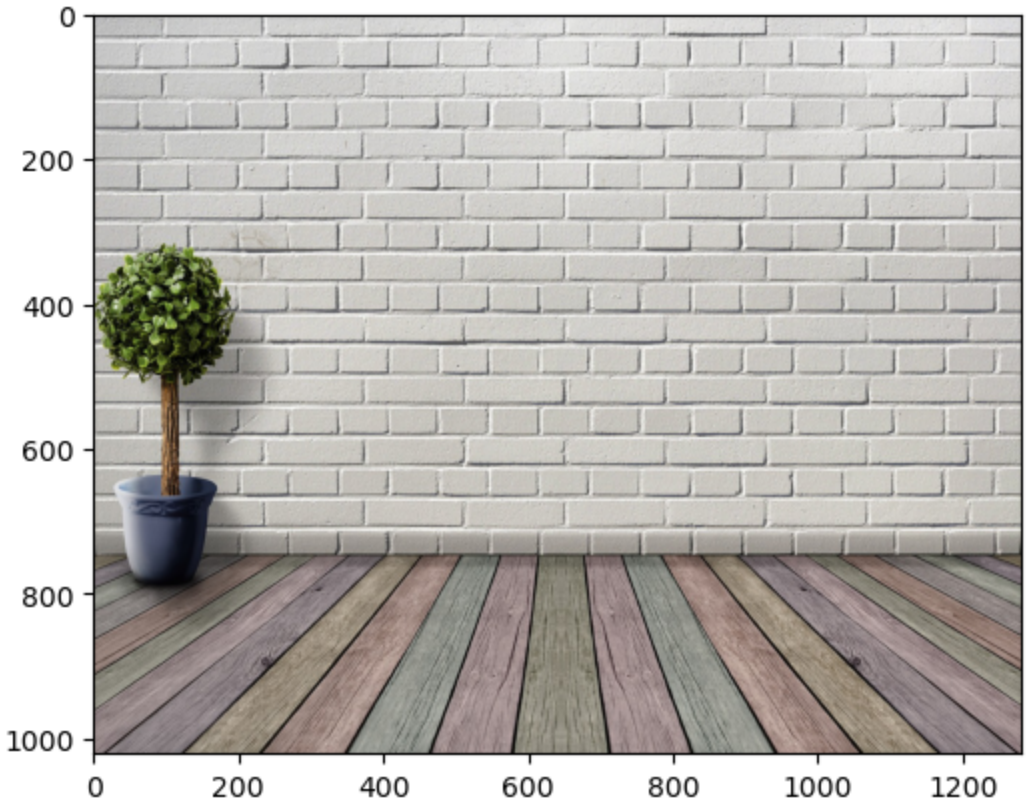

```
Max error is:  4.454214774796128e-13
```

Preparation

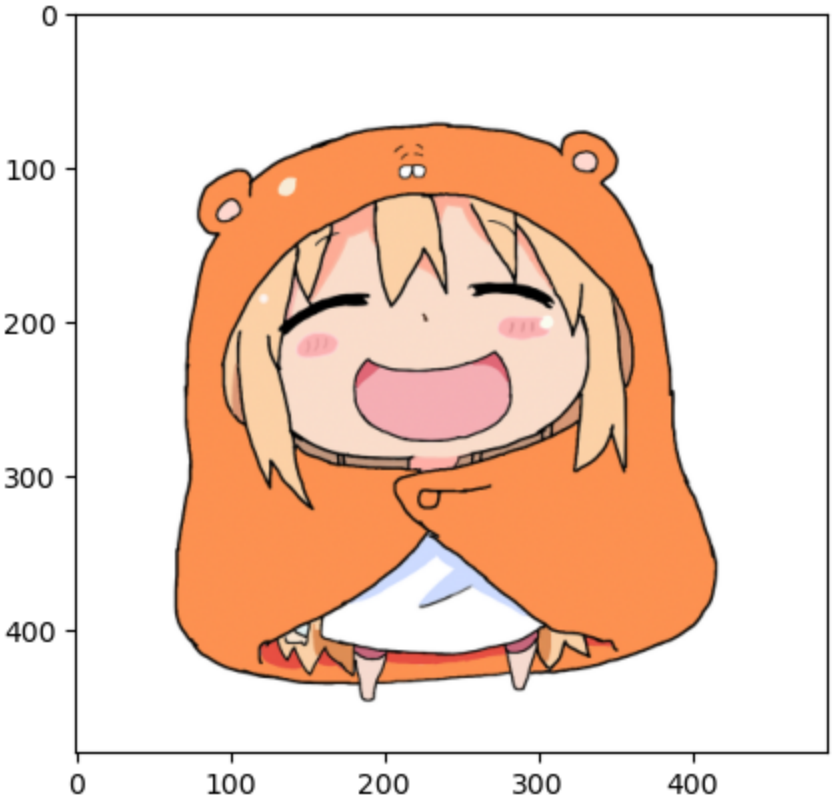
In [88]:

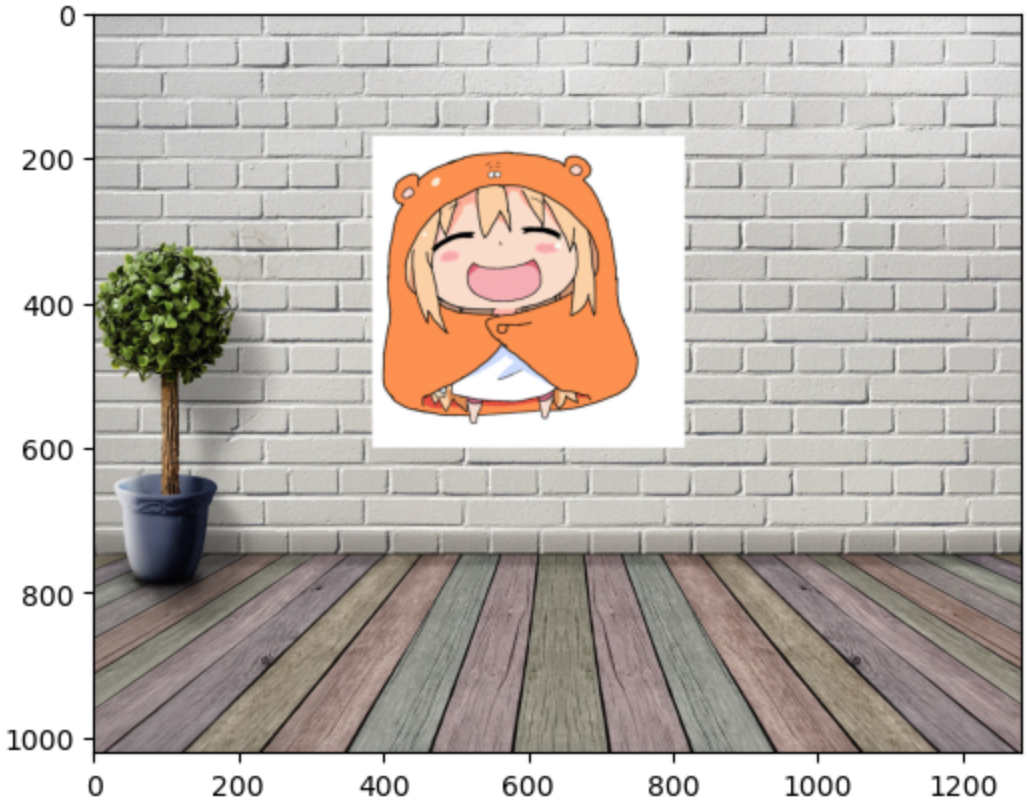
```
# background_img = cv2.cvtColor(cv2.imread('samples/im2.JPG'), cv2.COLOR_BGR2RGB).astype
# plt.figure()
# plt.imshow(background_img)
# plt.show()
# object_img = cv2.cvtColor(cv2.imread('samples/penguin-chick.jpeg'), cv2.COLOR_BGR2RGB)
# plt.imshow(object_img)
# plt.show()

# use_interface = False  # set to true if you want to use the interface to choose points
# if not use_interface:
#   xs = (65, 359, 359, 65)
#   ys = (24, 24, 457, 457)
#   object_mask = utils.get_mask(ys, xs, object_img)
#   bottom_center = (500, 2500) # (x,y)

#   object_img, object_mask = utils.crop_object_img(object_img, object_mask)
#   bg_ul = utils.upper_left_background_rc(object_mask, bottom_center)
#   plt.imshow(utils.get_combined_img(background_img, object_img, object_mask, bg_ul))

background_img = cv2.cvtColor(cv2.imread('imgs/bg_1.jpeg'), cv2.COLOR_BGR2RGB).astype('d
plt.figure()
plt.imshow(background_img)
plt.show()
object_img = cv2.cvtColor(cv2.imread('imgs/mai.png'), cv2.COLOR_BGR2RGB).astype('double'
object_img = cv2.resize(object_img, (0,0), fx=0.5, fy=0.5)
print(object_img.shape)
plt.imshow(object_img)
plt.show()

use_interface = False  # set to true if you want to use the interface to choose points (
if not use_interface:
  # xs = (200, 800, 800, 200)
  # ys = (400, 400, 100, 100)
  xs = (50, 480, 480, 50)
```

```
ys = (480, 480, 50, 50)
object_mask = utils.get_mask(ys, xs, object_img)
bottom_center = (600, 600) # (x,y)

object_img, object_mask = utils.crop_object_img(object_img, object_mask)
bg_ul = utils.upper_left_background_rc(object_mask, bottom_center)
plt.imshow(utils.get_combined_img(background_img, object_img, object_mask, bg_ul))
```




```
(480, 488, 3)
```





In [54]:
```
if use_interface:
  import matplotlib.pyplot as plt
  %matplotlib notebook
  mask_coords = utils.specify_mask(object_img)
```

In [55]:
```
if use_interface:
  xs = mask_coords[0]
  ys = mask_coords[1]
  %matplotlib inline
  import matplotlib.pyplot as plt
  plt.figure()
  object_mask = utils.get_mask(ys, xs, object_img)
```

In [56]:
```
if use_interface:
  %matplotlib notebook
  import matplotlib.pyplot as plt
  bottom_center = utils.specify_bottom_center(background_img)
#   print(bottom_center)
```

In [58]:
```
if use_interface:
  %matplotlib inline
  import matplotlib.pyplot as plt
  object_img, object_mask = utils.crop_object_img(object_img, object_mask)
  bg_ul = utils.upper_left_background_rc(object_mask, bottom_center)
  plt.imshow(utils.get_combined_img(background_img, object_img, object_mask, bg_ul))
```

Part 2 Poisson Blending (50 pts)

In [78]:
```
def poisson_blend(object_img, object_mask, bg_img, bg_ul):
    """
    Returns a Poisson blended image with masked object_img over the bg_img at position s
    Can be implemented to operate on a single channel or multiple channels
    :param object_img: the image containing the foreground object
    :param object_mask: the mask of the foreground object in object_img
    :param background_img: the background image
```

```
    :param bg_ul: position (row, col) in background image corresponding to (0,0) of obje
    """
    def neighbour(x, y):
        return [(x+1, y), (x-1, y), (x, y+1), (x, y-1)]

    idx_x, idx_y = bg_ul

    # creating im2var
    nnz = (object_mask>0).sum()
    im2var = -np.ones(object_img.shape[0:2], dtype='int32')
    im2var[object_mask>0] = np.arange(nnz)

    # construct sparse matrix A and b
    im_h, im_w = object_img.shape
    neq = 0
    for i in range(im_h):
        for j in range(im_w):
            if object_mask[i, j] > 0:
                for x, y in neighbour(i, j):
                    if 0 <= x < im_h and 0 <= y < im_w:
                        neq += 1

    A = scipy.sparse.lil_matrix((neq, nnz), dtype='double') # init lil
    b = np.zeros((neq,1), dtype='double')

    # possion blending
    e = 0
    for i in range(im_h):
        for j in range(im_w):
            if object_mask[i, j] > 0:
                for x, y in neighbour(i, j):
                    if 0 <= x < im_h and 0 <= y < im_w:
                        A[e, im2var[i, j]] = 1
                        b[e] = object_img[i, j] - object_img[x, y]
                        if object_mask[x, y] > 0:
                            A[e, im2var[x, y]] = -1
                        else:
                            b[e] = b[e] + bg_img[idx_x + x, idx_y + y]
                        e += 1

    v = scipy.sparse.linalg.lsqr(A.tocsr(), b, atol=10**-15, btol=10**-15)[0]

    # create blended image
    result = bg_img.copy()
    for i in range(im_h):
        for j in range(im_w):
            if object_mask[i, j] > 0:
                result[idx_x + i, idx_y + j] = v[im2var[i, j]]

    return result
```

In [79]:

```
im_blend = np.zeros(background_img.shape)
for b in np.arange(3):
  im_blend[:,:,b] = poisson_blend(object_img[:,:,b], object_mask, background_img[:,:,b].

plt.figure(figsize=(15,15))
plt.imshow(im_blend)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

Out[79]:

```
<matplotlib.image.AxesImage at 0x14c9a8670>
```

Part 3 Mixed Gradients (20 pts)

In [85]:
```
def mixed_blend(object_img, object_mask, bg_img, bg_ul):
    """
    Returns a mixed gradient blended image with masked object_img over the bg_img at pos
    Can be implemented to operate on a single channel or multiple channels
    :param object_img: the image containing the foreground object
    :param object_mask: the mask of the foreground object in object_img
    :param background_img: the background image
    :param bg_ul: position (row, col) in background image corresponding to (0,0) of obje
    """
    def neighbour(x, y):
        return [(x+1, y), (x-1, y), (x, y+1), (x, y-1)]

    idx_x, idx_y = bg_ul

    # creating im2var
    nnz = (object_mask>0).sum()
    im2var = -np.ones(object_img.shape[0:2], dtype='int32')
    im2var[object_mask>0] = np.arange(nnz)

    # construct sparse matrix A and b
    im_h, im_w = object_img.shape
    neq = 0
    for i in range(im_h):
        for j in range(im_w):
            if object_mask[i, j] > 0:
                for x, y in neighbour(i, j):
                    if 0 <= x < im_h and 0 <= y < im_w:
                        neq += 1

    A = scipy.sparse.lil_matrix((neq, nnz), dtype='double') # init lil
    b = np.zeros((neq,1), dtype='double')

    # mixed blending
    e = 0
    for i in range(im_h):
        for j in range(im_w):
```

```
            if object_mask[i, j] > 0:
                for x, y in neighbour(i, j):
                    if 0 <= x < im_h and 0 <= y < im_w:
                        A[e, im2var[i, j]] = 1
                        # determine the fg or bg gradient
                        grad_obj = object_img[i, j] - object_img[x, y]
                        grad_bg = bg_img[idx_x + i, idx_y + j] - bg_img[idx_x + x, idx_y
                        if abs(grad_obj) > grad_bg:
                            b[e] = grad_obj
                        else:
                            b[e] = grad_bg
                        if object_mask[x, y] > 0:
                            A[e, im2var[x, y]] = -1
                        else:
                            b[e] = b[e] + bg_img[idx_x + x, idx_y + y]
                        e += 1

    v = scipy.sparse.linalg.lsqr(A.tocsr(), b, atol=10**-15, btol=10**-15)[0]

    # create blended image
    result = bg_img.copy()
    for i in range(im_h):
        for j in range(im_w):
            if object_mask[i, j] > 0:
                result[idx_x + i, idx_y + j] = v[im2var[i, j]]

    return result
```

In [89]:

```
im_mix = np.zeros(background_img.shape)
for b in np.arange(3):
  im_mix[:,:,b] = mixed_blend(object_img[:,:,b], object_mask, background_img[:,:,b].copy

plt.figure(figsize=(15,15))
plt.imshow(im_mix)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

Out[89]:

```
<matplotlib.image.AxesImage at 0x14da2bcd0>
```

Bells & Whistles (Extra Points)

Color2Gray (20 pts)

In [ ]:
```
def color2gray(img):
    pass
```

Laplacian pyramid blending (20 pts)

In [ ]:
```
def laplacian_blend(object_img, object_mask, bg_img, bg_ul):
  # feel free to change input parameters
    pass
```

More gradient domain processing (up to 20 pts)

In [ ]:
```

```