**Cover**

**Push left**

for D in range(0, img\_shape[0] + 1, 2):

result[:,0:img\_shape[0]-D, :] = img1[:,D:,:]

result[:,img\_shape[0]-D:,:] = img2[:,0:D,:]

cv2.imshow('Push left', result)

cv2.waitKey(1)

for D in range(1, img\_shape[0] + 1, 2):

result[:,img\_shape[0]-D:, :] = img2[:, 0:D, :]

cv2.imshow('Cover', result)

cv2.waitKey(1)

**Go left and down**

for percent in range(1, 1000, 1):

percent = percent/1000

result[0:int(img\_shape[1] \* percent), int(img\_shape[0] \* (1 - percent)):,:] = img1[img\_shape[1] - int(img\_shape[1] \* percent):,0:img\_shape[0] - int(img\_shape[0] \* (1 - percent)),:]

result[int(img\_shape[1] \* percent):,0:int(img\_shape[0] \* (1 - percent)), :] = img2[int(img\_shape[1] \* percent):,0:int(img\_shape[0] \* (1 - percent)),:]

cv2.imshow('Go left and down', result)

cv2.waitKey(1)

**Seam carving:**

# Hàm tính ma trận năng lượng

def calc\_energy(img):

filter\_du = np.array([[1.0, 2.0, 1.0], [0.0, 0.0, 0.0], [-1.0, -2.0, -1.0],])

filter\_du = np.stack([filter\_du] \* 3, axis=2)

filter\_dv = np.array([[1.0, 0.0, -1.0], [2.0, 0.0, -2.0], [1.0, 0.0, -1.0],])

filter\_dv = np.stack([filter\_dv] \* 3, axis=2)

img = img.astype(int)

convolved = np.absolute(convolve(img, filter\_du)) + np.absolute(convolve(img, filter\_dv))

energy\_map = convolved.sum(axis=2)

return energy\_map

**Phát hiện biên cạnh với Hough Transform**

# Bước 1: Tính ảnh biên cạnh: Edge (dùng ảnh năng lượng)

edges = cv2.Canny(gray, 300, 400, apertureSize = 3, L2gradient = True)

# Bước 2: Khởi tạo H là ma trận 0

theta\_range = np.arange(-3.14, 3.14, 0.01)

H = np.zeros((500, len(theta\_range)), dtype=np.uint8) # 500 là độ dài đường chéo

# Bước 3:

list\_pro\_theta = []

def accumulate(point):

for theta in theta\_range:

pro = point[0]\*np.cos(theta) + point[1]\*np.sin(theta)

if pro >= 0 and pro < 500:

H[int(pro), int((theta + 3.14)/0.01)] += 1

list\_pro\_theta.append([int(pro), int((theta + 3.14)/0.01)])

return H

# Bước 4: Chọn ra những điểm pro và theta "đủ lớn"

def find\_the\_line():

list\_H = []

for list in list\_pro\_theta:

if H[list[0], list[1]] > 140:

list\_H.append([list[0], list[1]])

for i in range(len(list\_H)):

list\_H[i][1] = (list\_H[i][1]) \* 0.01 - 3.14

return list\_H

# Hàm tìm đường tốn ít năng lượng nhất

def minimum\_seam(img):

row, col, \_ = img.shape

energy\_map = calc\_energy(img)

Sum\_e = energy\_map.copy()

location\_pre = np.zeros\_like(Sum\_e, dtype=np.int)

for i in range(1, row):

for j in range(0, col):

if j == 0:

value = np.argmin(Sum\_e[i - 1, j:j + 1])

location\_pre[i, j] = value + j

min\_energy = Sum\_e[i - 1, value + j]

else:

value = np.argmin(Sum\_e[i - 1, j - 1:j + 1])

location\_pre[i, j] = value + j - 1

min\_energy = Sum\_e[i - 1, value + j - 1]

Sum\_e[i, j] += min\_energy

return Sum\_e, location\_pre

# Xóa các pixel khỏi đường nối nhỏ nhất

def del\_seam(img):

row, col, \_ = img.shape

Sum\_e, location\_pre = minimum\_seam(img)

mask = np.ones((row, col), dtype=np.bool)

j = np.argmin(Sum\_e[-1])

for i in reversed(range(row)):

mask[i, j] = False

j = location\_pre[i, j]

mask = np.stack([mask] \* 3, axis=2)

img = img[mask].reshape((row, col - 1, 3))

return img

# Xóa theo chiều dọc

# Xóa theo chiều ngang

def crop\_row(img, scale\_r):

img = np.rot90(img, 1, (0, 1))

img = crop\_col(img, scale\_r)

img = np.rot90(img, 3, (0, 1))

return img

def crop\_col(img, scale\_col):

row, col, \_ = img.shape

new\_col = int(scale\_col \* col)

for i in trange(col - new\_col):

img = del\_seam(img)

return img