DIP Lab 2: Image Warping

2017011620　计73　李家昊

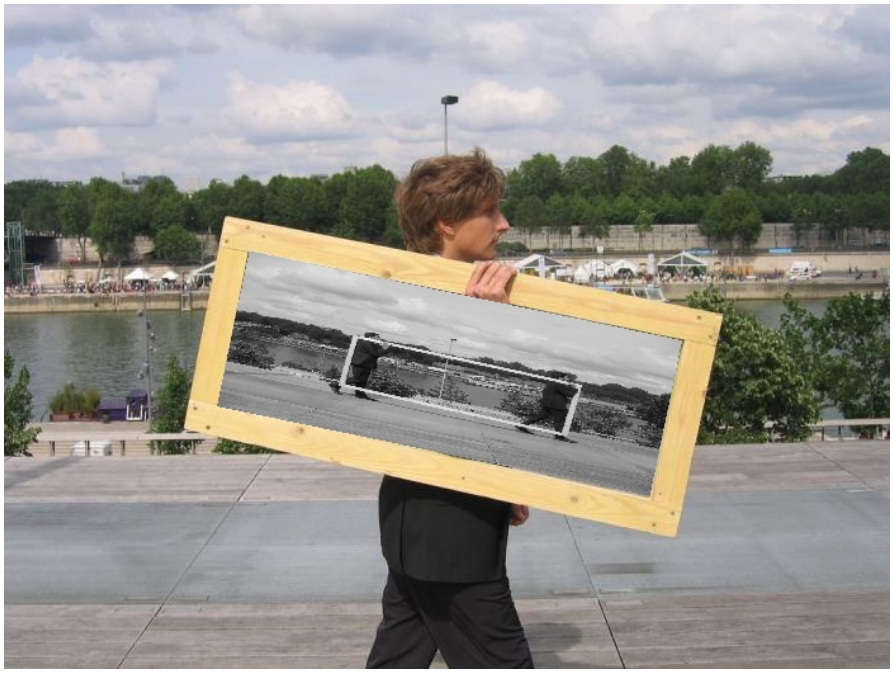
2020 年 3 月 13 日

1 Perspective Warping

首先展示生成效果，如图 1所示。

(a) Source

(b) Target



(c) Output

图 1: Perspective Warping

核心算法如下，首先需要得到变换矩阵，设其为，

$$\boldsymbol{A} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \tag{1}$$

在原图中找到四点的坐标为$\{(x_k, y_k)\}_{k=1}^4$，在目标图像中找到它们对应的四点的坐标$\{(u_k, v_k)\}_{k=1}^4$，则在齐次坐标下有，

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ 1 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} u_1 & u_2 & u_3 & u_4 \\ v_1 & v_2 & v_3 & v_4 \\ 1 & 1 & 1 & 1 \end{pmatrix} \tag{2}$$

其中两边取转置，即，

$$\begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \\ x_4 & y_4 & 1 \end{pmatrix} \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}^T = \begin{pmatrix} u_1 & v_1 & 1 \\ u_2 & v_2 & 1 \\ u_3 & v_3 & 1 \\ u_4 & v_4 & 1 \end{pmatrix} \tag{3}$$

通过解线性方程组即可求出变换矩阵$\boldsymbol{A}$，这样就可以对图像进行变换了。首先求出其逆矩阵$\boldsymbol{A}^{-1}$，然后扫描目标图像的每一个像素位置(r_d, c_d)，根据下式，

$$(r_s, c_s, 1)^T = s \cdot \boldsymbol{A}^{-1}(r_d, c_d, 1)^T \tag{4}$$

求出原图中对应的像素位置(r_s, c_s)，其中s为齐次坐标的归一系数，得出坐标可能不是整数，则四舍五入到最近的整数格点$(\text{round}(r_s), \text{round}(c_s))$，即最邻近插值，然后直接将原图此坐标下的值拷贝到目标图的对应位置。核心代码如下，

```
def get_perspective_transform(src_rect, dst_rect):
    assert src_rect.shape == (4, 2) and dst_rect.shape == (4, 2)
    src_rect = np.concatenate([src_rect.astype(np.float32), np.ones
        ((4, 1), dtype=np.float32)], axis=-1)
    dst_rect = np.concatenate([dst_rect.astype(np.float32), np.ones
        ((4, 1), dtype=np.float32)], axis=-1)
    matrix, _, _, _ = np.linalg.lstsq(src_rect, dst_rect)
    return matrix.T

def warp_perspective(src, matrix, dst_size):
    assert src.ndim == 2 or src.ndim == 3
    src_h, src_w = src.shape[:2]
    dst_w, dst_h = dst_size
    dst_shape = (dst_h, dst_w, src.shape[2]) if src.ndim == 3 else (
        dst_h, dst_w)
    dst = np.zeros(dst_shape, dtype=np.uint8)
```

```
    inv = np.linalg.inv(matrix)
    for dst_y in range(dst_h):
        for dst_x in range(dst_w):
            src_x, src_y, src_scale = np.matmul(inv, [dst_x, dst_y,
                1])
            src_x = int(round(src_x / src_scale))
            src_y = int(round(src_y / src_scale))
            if 0 <= src_x < src_w and 0 <= src_y < src_h:
                dst[dst_y, dst_x] = src[src_y, src_x]
    return dst
```

2 Sphere Warping

首先展示结果，如图 2所示。

(a) Source (b) Output

图 2: Sphere Warping

核心算法如下，将原图均匀铺在一个水晶球上面，剖面图如图 3所示，

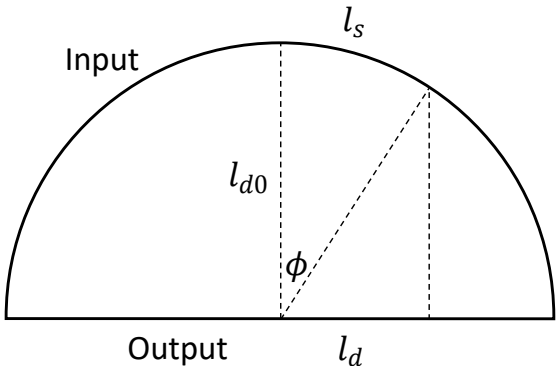


图 3: Sphere Warping Algorithm

设原图的中心为(r_{s0}, c_{s0})，目标图的中心为(r_{d0}, c_{d0})，水晶球的半径为$l_{d0} = \max(r_{d0}, c_{d0})/(\pi/2)$，对于目标图的每个像素$(r_d, c_d)$，首先计算出它到图像中心的距离$l_d = \sqrt{(r_d - r_{d0})^2 + (c_d - c_{d0})^2}$，由几何关系可得，原图对应点距离原图得中心点的距离为，

$$l_s = l_{d0} \cdot \phi = l_{d0} \cdot \arcsin\left(\frac{l_d}{l_{d0}}\right) \tag{5}$$

由相似三角形关系，可知原图对应点(r_s, c_s)满足，

$$(r_s - r_{s0}, c_s - c_{s0}) = \frac{l_s}{l_d}(r_d - r_{d0}, c_d - c_{d0}) \tag{6}$$

这样一来，就可以求出目标图像的每一点对应的原图坐标(r_s, c_s)了，同样，如果坐标不为整数，则四舍五入到邻近的整数格点。

核心代码如下，

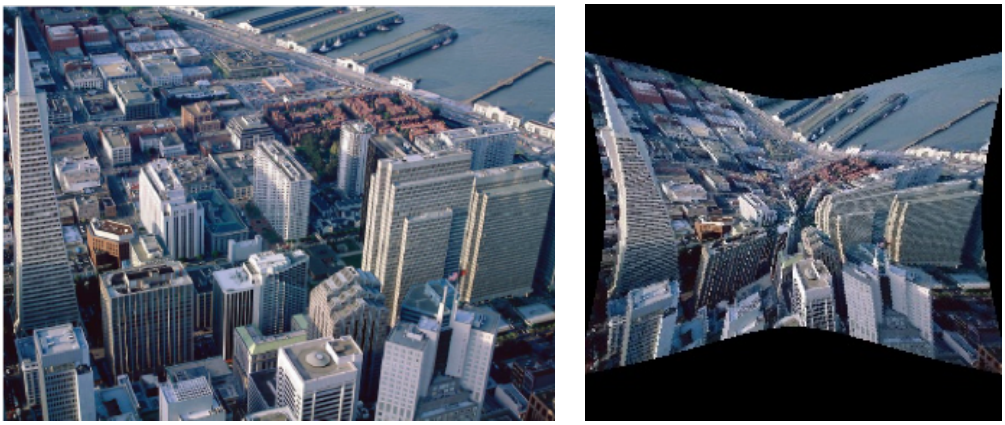
```
def warp_sphere(src, dst_size):
    assert src.ndim == 2 or src.ndim == 3
    src_h, src_w = src.shape[:2]
    dst_w, dst_h = dst_size
    dst_shape = (dst_h, dst_w, src.shape[2]) if src.ndim == 3 else (
        dst_h, dst_w)
    dst = np.zeros(dst_shape, dtype=np.uint8)

    max_dst_radius = min(dst_w, dst_h) // 2
    dst_center_y, dst_center_x = max_dst_radius, max_dst_radius
    src_center_y, src_center_x = src_h // 2, src_w // 2

    for dst_y in range(dst_h):
        for dst_x in range(dst_w):
            dst_radius = math.sqrt((dst_y - dst_center_y) ** 2 + (
                dst_x - dst_center_x) ** 2)
            if dst_radius > max_dst_radius:
                continue
            src_radius = max_dst_radius * math.asin(dst_radius /
                max_dst_radius)
            scale_factor = src_radius / dst_radius if dst_radius else
                1
            src_y = int(round(scale_factor * (dst_y - dst_center_y) +
                src_center_y))
            src_x = int(round(scale_factor * (dst_x - dst_center_x) +
                src_center_x))
            if 0 <= src_y < src_h and 0 <= src_x < src_w:
                dst[dst_y, dst_x] = src[src_y, src_x]
    return dst
```

3 Extra Task

这里做了一个拓展实验，前面的球面形变产生了凸透镜效果，于是我在这里尝试了一下凹透镜效果，如图 4所示，



(a) Source (b) Output

图 4: Inverse Sphere Warping

原理是把原图均匀铺在一个锥面上，剖面图如图 5所示，

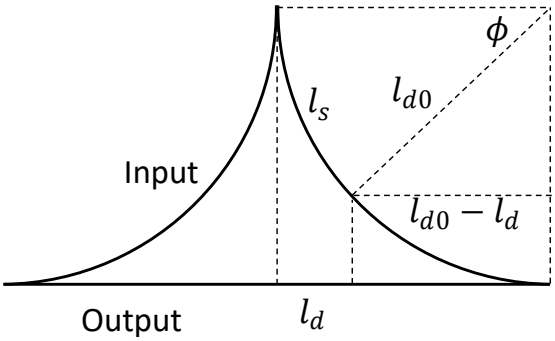


图 5: Inverse Sphere Warping Algorithm

同样设目标图一点(r_d, c_d)到中心距离为l_d，则原图对应点到中心距离为，

$$l_s = l_{d0} \cdot \phi = l_{d0} \cdot \arccos\left(\frac{l_{d0} - l_d}{l_{d0}}\right) \tag{7}$$

其余操作同凸透镜效果。核心代码类似，不再赘述，可以参考附录的完整代码。

4 Appendix: Codes

附上了完整代码，这里导入的opencv仅仅用来读取和保存图片。

```
import cv2
import numpy as np
import math
import argparse


def get_perspective_transform(src_rect, dst_rect):
    assert src_rect.shape == (4, 2) and dst_rect.shape == (4, 2)
    src_rect = np.concatenate([src_rect.astype(np.float32), np.ones
        ((4, 1), dtype=np.float32)], axis=-1)
    dst_rect = np.concatenate([dst_rect.astype(np.float32), np.ones
        ((4, 1), dtype=np.float32)], axis=-1)
    matrix, _, _, _ = np.linalg.lstsq(src_rect, dst_rect)
    return matrix.T


def warp_perspective(src, matrix, dst_size):
    assert src.ndim == 2 or src.ndim == 3
    src_h, src_w = src.shape[:2]
    dst_w, dst_h = dst_size
    dst_shape = (dst_h, dst_w, src.shape[2]) if src.ndim == 3 else (
        dst_h, dst_w)
    dst = np.zeros(dst_shape, dtype=np.uint8)

    inv = np.linalg.inv(matrix)
    for dst_y in range(dst_h):
        for dst_x in range(dst_w):
            src_x, src_y, src_scale = np.matmul(inv, [dst_x, dst_y,
                1])
            src_x = int(round(src_x / src_scale))
            src_y = int(round(src_y / src_scale))
            if 0 <= src_x < src_w and 0 <= src_y < src_h:
                dst[dst_y, dst_x] = src[src_y, src_x]
    return dst


def warp_sphere(src, dst_size):
    assert src.ndim == 2 or src.ndim == 3
    src_h, src_w = src.shape[:2]
    dst_w, dst_h = dst_size
    dst_shape = (dst_h, dst_w, src.shape[2]) if src.ndim == 3 else (
        dst_h, dst_w)
    dst = np.zeros(dst_shape, dtype=np.uint8)
```

```
    max_dst_radius = min(dst_w, dst_h) // 2
    dst_center_y, dst_center_x = max_dst_radius, max_dst_radius
    src_center_y, src_center_x = src_h // 2, src_w // 2

    for dst_y in range(dst_h):
        for dst_x in range(dst_w):
            dst_radius = math.sqrt((dst_y - dst_center_y) ** 2 + (
                dst_x - dst_center_x) ** 2)
            if dst_radius > max_dst_radius:
                continue
            src_radius = max_dst_radius * math.asin(dst_radius /
                max_dst_radius)
            scale_factor = src_radius / dst_radius if dst_radius else
                1
            src_y = int(round(scale_factor * (dst_y - dst_center_y) +
                src_center_y))
            src_x = int(round(scale_factor * (dst_x - dst_center_x) +
                src_center_x))
            if 0 <= src_y < src_h and 0 <= src_x < src_w:
                dst[dst_y, dst_x] = src[src_y, src_x]
    return dst


def warp_sphere_inv(src, dst_size):
    assert src.ndim == 2 or src.ndim == 3
    src_h, src_w = src.shape[:2]
    dst_w, dst_h = dst_size
    dst_shape = (dst_h, dst_w, src.shape[2]) if src.ndim == 3 else (
        dst_h, dst_w)
    dst = np.zeros(dst_shape, dtype=np.uint8)

    max_src_radius = min(src_h, src_w) // 2
    dst_center_y, dst_center_x = dst_h // 2, dst_w // 2
    src_center_y, src_center_x = src_h // 2, src_w // 2

    for dst_y in range(dst_h):
        for dst_x in range(dst_w):
            dst_radius = math.sqrt((dst_y - dst_center_y) ** 2 + (
                dst_x - dst_center_x) ** 2)
            if abs(1 - dst_radius > max_src_radius) > 1:
                continue
            src_radius = max_src_radius * math.acos(1 - dst_radius /
                max_src_radius)
            scale_factor = src_radius / dst_radius if dst_radius else
                1
```

```
            src_y = int(round(scale_factor * (dst_y - dst_center_y) +
                src_center_y))
            src_x = int(round(scale_factor * (dst_x - dst_center_x) +
                src_center_x))
            if 0 <= src_y < src_h and 0 <= src_x < src_w:
                dst[dst_y, dst_x] = src[src_y, src_x]
    return dst


def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('--mode', type=str, choices=['perspective', '
        sphere', 'sphere-inv'], default='perspective')
    args = parser.parse_args()

    if args.mode == 'perspective':
        src = cv2.imread('image/source.jpg')
        dst = cv2.imread('image/target.jpg')
        src_h, src_w, _ = src.shape
        dst_h, dst_w, _ = dst.shape
        src_rect = np.array([[0, 0], [src_w, 0], [src_w, src_h], [0,
            src_h]], dtype=np.float32)
        dst_rect = np.array([(192, 195), (536, 264), (508, 388),
            (169, 316)], dtype=np.float32)
        matrix = get_perspective_transform(src_rect, dst_rect)
        mask = np.ones((src_h, src_w), dtype=np.float32)
        out = warp_perspective(src, matrix, (dst_w, dst_h))
        mask = warp_perspective(mask, matrix, (dst_w, dst_h))
        out = (mask[:, :, None] < 0.5) * dst + out
        cv2.imwrite('output/warp_perspective.jpg', out)
    elif args.mode == 'sphere':
        src = cv2.imread('image/warping.png')
        dst_len = int(max(src.shape[:2]) / math.pi * 2)
        dst = warp_sphere(src, (dst_len, dst_len))
        cv2.imwrite('output/warp_sphere.jpg', dst)
    else:
        src = cv2.imread('image/warping.png')
        dst_len = int(max(src.shape[:2]) / math.pi * 2)
        dst = warp_sphere_inv(src, (dst_len, dst_len))
        cv2.imwrite('output/warp_sphere_inv.jpg', dst)


if __name__ == "__main__":
    main()
```