

Image Warping*

石大川 <sdc17@mails.tsinghua.edu.cn>

2020 年 3 月 14 日

Affine Warping

因为在实验中发现 Affine Warping 不能完美贴合边界，所以实际做的是 Projective Warping.

Procedure

Algorithm 1 Projective Warping

- 1: 选定投影结果的四个顶点
- 2: **for** 每个选中的顶点 **do**
- 3: 根据如下矩阵得到关于 x' 和 y' 的两个方程
$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$
- 4: **end for**
- 5: $f\text{solve}$ 解方程组得到八个系数的值
- 6: **for all** $pixel \in source.jpg$ **do**
- 7: 根据如下方程算出对应 $target.jpg$ 中的位置并做相应的投影

$$x' = \frac{ax + by + cw}{gx + hy + w}$$

$$y' = \frac{dx + ey + fw}{gx + hy + w}$$

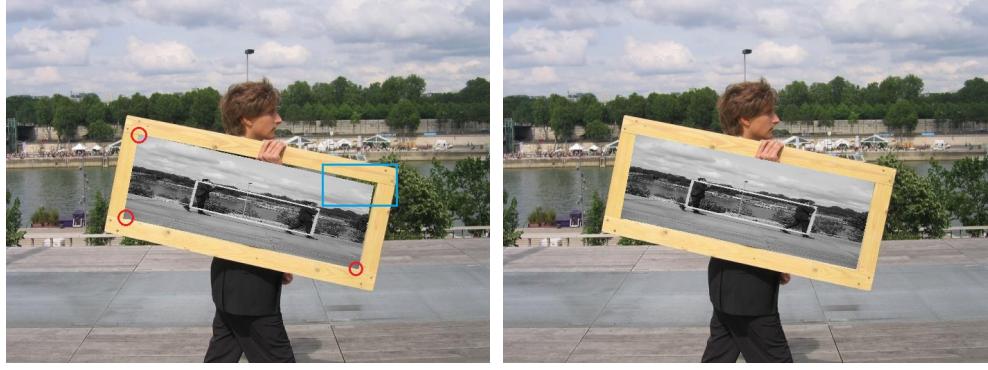
- 8: **end for**
-

Affine Warping 相比 Projective Warping 的流程只是将系数 g 和 h 简化为了 0

Result

Figure 1展示了实验结果，其中Figure a是以红圈标出的三个顶点为目标做的 Affine Warping，可见在蓝框标出的第四个顶点附近贴合程度并不好。相对地，Figure b是选定四个顶点做 Projective Warping 的结果，可见贴合程度是比较理想的。

*<https://github.com/sdc17/NaiveIW>



(a) Affine Warping

(b) Projective Warping

Figure 1: Affine Warping and Projective Warping

Sphere Warping

因为给的原图比较糊，所以自己在网上找了同一张图的高清版本作为替代并且保留了原图 4:3 的长宽比。

Procedure

Algorithm 2 Sphere Warping

- 1: 分别计算输入输出的最大中心距离 d_0 和 ρ_0
- 2: **for all** $pixel \in sphere.jpg$ **do**
- 3: 对于输出图像中的每个像素点计算

$$\rho = \sqrt{r_{out}^2 + c_{out}^2}$$

$$\theta = \tan^{-1}\left(\frac{r_{out}}{c_{out}}\right)$$

$$\phi = \sin^{-1}\left(\frac{\rho}{\rho_0}\right)$$
- 4: **if** $pixel$ 到圆心的距离小于 ρ **then**
- 5: 根据如下公式得到原图中相应 $pixel$ 偏离中心的距离

$$d = \frac{2d_0\phi}{\pi}$$

$$r_{in} = d\sin(\theta)$$

$$c_{in} = d\cos(\theta)$$

- 6: **if** 对应原图的 $pixel$ 超出了原图尺寸范围 **then**
- 7: 输出的 $pixel$ 置为 [127, 127, 127]
- 8: **else**
- 9: 根据行偏移 r_{in} 和列偏移 c_{in} 得到原像素地址并做 Bilinear

$$f(x, y) = \frac{1}{(x_2 - x_1)(y_2 - y_1)} \begin{bmatrix} x_2 - x & x - x_1 \end{bmatrix} \begin{bmatrix} f(Q_{11}) & f(Q_{12}) \\ f(Q_{21}) & f(Q_{22}) \end{bmatrix} \begin{bmatrix} y_2 - y \\ y - y_1 \end{bmatrix}$$
- 10: 其中 $Q_{11}, Q_{21}, Q_{22}, Q_{12}$ 分别是从左下角开始沿逆时针方向的四个相邻整数坐标点的像素值

```

11:      end if
12:  end if
13: end for

```

Result

Figure 2展示了实验结果，其中Figure a的插值方法是直接使用距离最近的一个像素，Figure b中用的是Bilinear插值。放大后对比可以发现在图像细节上，第二种插值方式是优于第一种插值方式的。例如对比左图红框标出的位置和右图中对应的位置可见，右图此处垂直方向的白色直线错位程度更低。

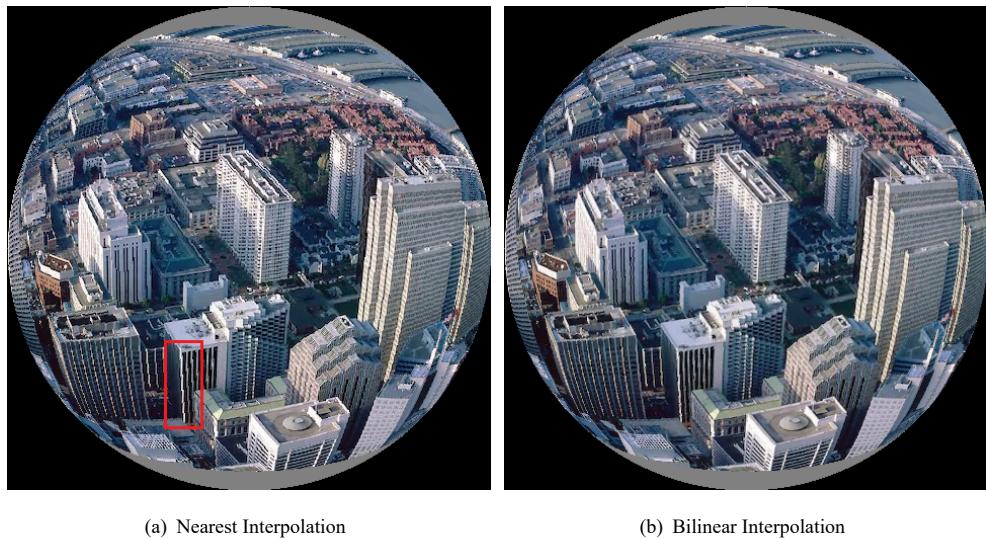


Figure 2: Sphere Warping

Water Wave Warping

效果类似于在平静的湖面上扔一颗石子，然后以石子落点为中心湖面会泛开涟漪。

Procedure

Algorithm 3 Water Wave Warping

- 1: 选定一个在原图尺寸范围内的波动中心 $center$
- 2: **for all** $pixel \in ww.jpg$ **do**
- 3: 计算每个 $pixel$ 到波动中心的相对位置

$$x_{relative} = x_{origin} - center_{col}$$

$$y_{relative} = center_{row} - y_{origin}$$

- 4: 根据相对位置计算 $pixel$ 到波动中心的夹角

$$\theta = \arctan \frac{y_{relative}}{x_{relative}}$$

5: 计算波动前 $pixel$ 到波动中心的距离

$$r_0 = \sqrt{x_{relative}^2 + y_{relative}^2}$$

6: 计算波动后 $pixel$ 到波动中心的距离, 其中 $decay$ 和 $velocity$ 分别是控制衰减和传播速度的超参数

$$r = r_0 + decay \times shape_{col} \times \sin(velocity \times r_0)$$

7: **end for**

8: 根据波动后的距离转换到相应的 $pixel$ 位置

$$x_{out} = r \times \cos \theta + center_{col}$$

$$y_{out} = center_{row} - r \times \sin \theta$$

9: **for all** $pixel \in result.jpg$ **do**

10: **if** 按照前述流程得到对应到原图中的像素位置没有超出原图尺寸 **then**

11: 按照已经计算出的映射关系, 做 Bilinear 插值得到结果 $pixel$

12: **else**

13: 第一行根据左边的 $pixel$ 插值, 第一列根据上边的 $pixel$ 插值, 其余的根据左上的 $pixel$ 插值

14: **end if**

15: **end for**

Result

Figure 3展示了实验结果, 其中 Figure a 是原图, Figure b 是做 Water Wave Warping 后的结果。在观感上确实产生了水面涟漪的效果, 在结果图中也能较容易地识别出原图的轮廓, 这符合在实际生活中向水面投掷石子的结果。

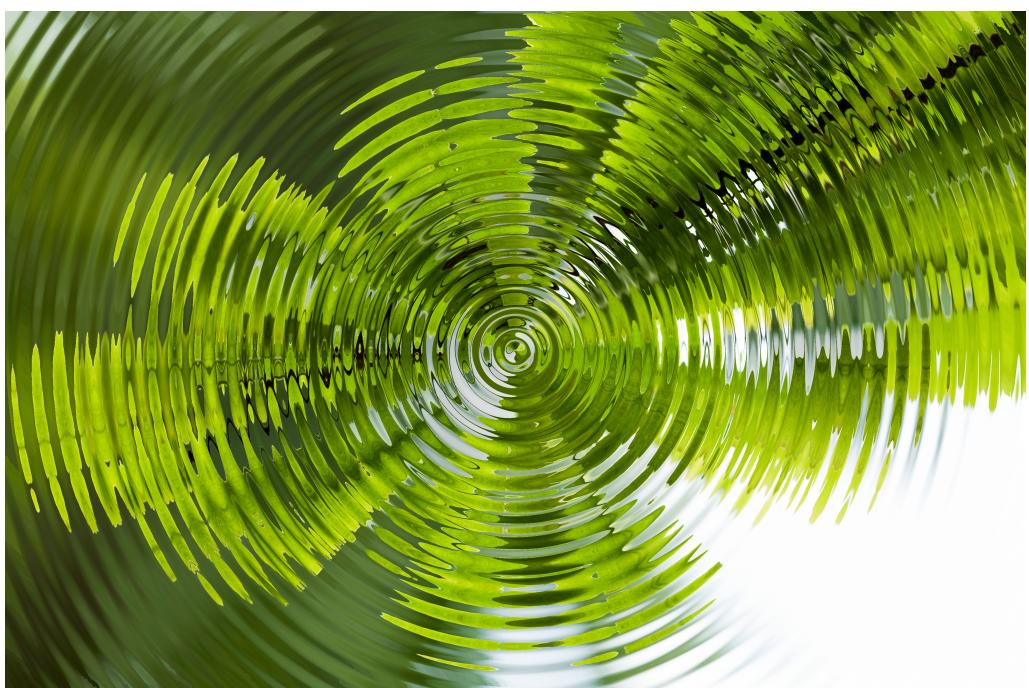
此外, 更进一步还可以做多个波动的干涉结果, 直观上类似于向水面不同位置投掷多枚石子。因为波动的干涉有线性叠加性, 所以在每个像素点将来自不同地方的多个波叠加起来即可。

Conclusion

这次作业加深了我对于 Image Warping 和 Bilinear 等插值方式的理解, 动手实现三种 Image Warping 和 Bilinear 插值方式的过程则让我对于 numpy 和 scipy 的使用变得更加熟练。



(a) Origin Photo



(b) Result Photo

Figure 3: Sphere Warping