

数字图象处理

综合作业三

2019010485 自 91 刘祖炎*

2022 年 1 月 16 日

1 原理及代码分析

1.1 效果一: *Shopping Arcade*

1.1.1 问题描述

给定图片1, 将其变换至 Shopping Arcade 的广告牌中, 如图2所示¹。



图 1: 待处理图像



图 2: 示例处理结果

分析该问题, 可知需要较好地复现该效果, 需要达成以下几点要求:

- 模板图像中存在灯光的遮挡, 需要抠图区分前景和背景部分。
- 由于光线因素, 模板色调与原图色调有明显不同, 模板色调偏黄, 需要进行色调风格变换。
- 模板图像中存在灯光产生的圆形打光效果, 需要对其进行模拟。

*liuzuyan19@mails.tsinghua.edu.cn

¹https://photofunia.com/categories/all_effects/shopping-arcade

1.1.2 基于 Reinhard 算法的自适应风格迁移

Reinhard 算法由 Reinhard 等人于 2001 年提出²，能够较好地实现对模板图像的色彩空间模拟，并将待变换图像转换至模板图像的色彩空间。

Reinhard 算法的实现原理基于 *Lab* 色彩空间，其将颜色用 L, a, b 三个值表示，其中 L 代表感知的亮度， a 和 b 表示人类视觉敏感的四种颜色（红色、绿色、蓝色、黄色）。*Lab* 色彩空间下，三原色之间的相关性减弱，且色调与亮度分量相互分离，这使得在模拟色彩空间时，不会受到模板图像本身颜色的影响，能够较好地对色调风格进行模拟。

Reinhard 算法首先将模板图像和待变换图像转换到 *Lab* 空间下，计算模板图像和待变换图像在 *Lab* 空间下的均值和标准差：

$$\begin{aligned} & MEAN[(L, a, b)_{ref}, (L, a, b)_{img}] \\ & STD[(L, a, b)_{ref}, (L, a, b)_{img}] \end{aligned} \quad (1)$$

分别处理 L, a, b 通道，对目标图片的每一个像素值 θ ，按照公式2进行计算，得到变换后的像素值 θ' ：

$$\theta' = (\theta - MEAN(img)) \cdot \frac{STD(ref)}{STD(img)} + MEAN(ref) \quad (2)$$

最后将变换后图像转换回 *RGB* 空间下即可。

笔者基于 Matlab 提供的 *Lab* 与 *RGB* 转换函数，编写实现了 *Reinhard* 算法，代码如下所示，输入待变换图像与参考图像即可。

```
1 function result = Reinhard (img, ref)
2     img = rgb2lab(img);
3     ref = rgb2lab(ref);
4     img_mean = squeeze(mean(mean(img)));
5     ref_mean = squeeze(mean(mean(ref)));
6     img_std = squeeze(std(std(img)));
7     ref_std = squeeze(std(std(ref)));
8     [H, W, C] = size(img);
9     for i = 1:H
10        for j = 1:W
11            for k = 1:C
12                t = img(i,j,k);
13                t = (t - img_mean(k)) * (ref_std(k) / img_std(k)) + ref_mean(k);
14                img(i,j,k) = t;
15            end
16        end
17    end
18    result = lab2rgb(img);
19 end
```

²Reinhard E, Adhikhmin M, Gooch B, et al. Color transfer between images[J]. IEEE Computer graphics and applications, 2001, 21(5): 34-41.

具体实现时，需要从模板图像中裁剪出有效部分，此处直接通过选取 ROI 矩阵端点，并利用矩阵索引的方式取出模板图像即可。

经测试，Reinhard 算法能够实现极佳的风格模拟效果，且该算法较为鲁棒，对多种色彩风格均可进行模拟，这在很大程度上减轻了本次数图作业中风格变换任务的调色压力，能够将繁琐复杂的调色任务交由程序自主进行。以本题为例，图3、4、5分别为变换前图像、风格参考图像、变换后图像，可以看到，利用 Reinhard 算法进行风格变换能够避免 RGB 颜色本身的影响，实现较好的风格模拟效果。



图 3: 待变换图像



图 4: 模板风格图像



图 5: 变换后图像

1.1.3 基于局部图割法的前景分割

为准确分割出前景区域，笔者采用两种方法进行分割。

方法一为基本的 ROI 分割方法，设定 ROI 矩形四个端点的坐标，产生矩形 Mask 矩阵 M_1 。

方法二利用 Matlab 提供的 Image Segmenter 工具箱实现，其基本原理为超像素分割方法，并通过用户交互产生前景、背景区域，产生 Mask 矩阵 M_2 。其使用方法较为简单，载入待分割图像，绘制 ROI 后，调整超像素分割子区域密度，并标记前景、背景即可。超像素分割结果及前景分割结果如图6所示。可以看到，超像素分割取得了较好的分割效果，成功分割了图像下部不规则区域。

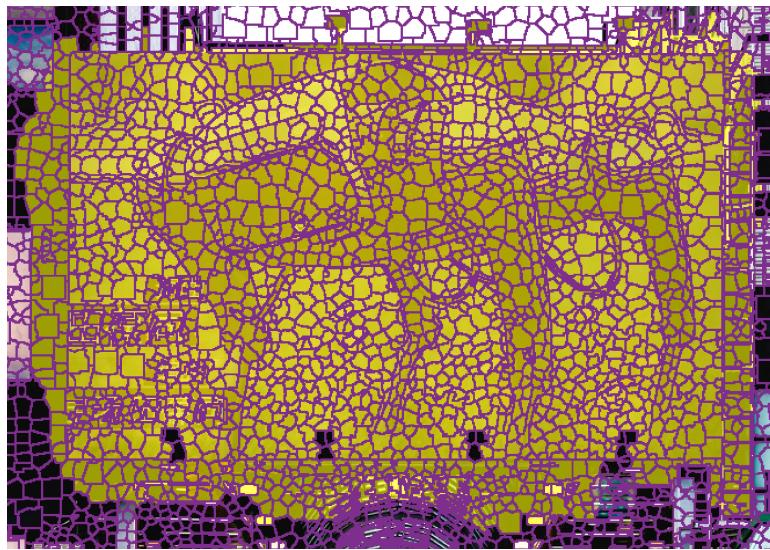


图 6: 超像素分割结果 (黄色部分为前景)

将相关分割结果保存至 *mask2.mat* 文件中，后续运行代码时直接读取 M_2 即可。相关代码如下所示：

```

1 mask_thres = [296, 888, 86, 453];
2 mask1 = zeros(height, width);
3 mask1(mask_thres(3):mask_thres(4), mask_thres(1):mask_thres(2)) = 1;
4 mask2 = load('mask2.mat').BW2;
```

```
5 mask = mask1 .* mask2;
```

产生的 M_1, M_2, M 分别如图7、8、9所示。可以看到，最终分割结果较为整齐、准确。



图 7: ROI 分割结果



图 8: 超像素分割结果

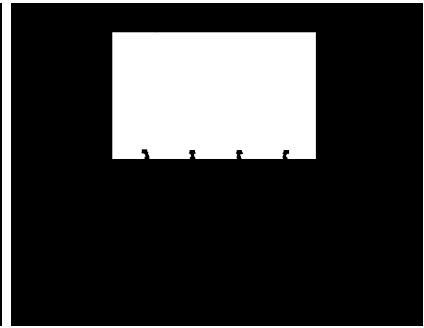


图 9: 最终分割结果

1.1.4 灯光效果模拟

由于 Matlab 没有与灯光效果模拟相关的函数，因此需要自行实现灯光效果。笔者考虑到高斯均值滤波器归一化后的图像与灯光有相似之处，考虑到便捷性，直接利用高斯滤波器模拟灯光效果，如图10所示。调节适当的灯光大小、中心光强后，选取原图中 8 个灯光中心点，并将高斯滤波器的值按照一定比例叠加在原像素上即可实现对灯光效果的模拟。经过试验，笔者采用如下公式进行亮度叠加，可实现较好的模拟效果：

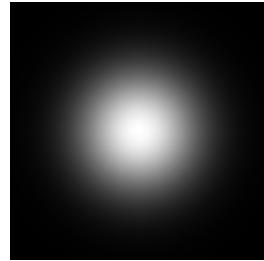


图 10: 高斯滤波器归一化结果

$$\Theta' = \Theta \times \left(1 + \frac{3Gauss}{2}\right) \quad (3)$$

相关实现代码如下所示，高斯滤波器相关参数取 $length = 125, \sigma = 20$:

```
1 img = imresize(img, [mask_thres(4)-mask_thres(3)+1 mask_thres(2)-mask_thres(1)+1]);
2 light = Normalize(fspecial('gaussian', [125, 125], 20));
3 light_pos = [94, 232, 397, 505, 35, 331];
4 for i = 1:4
5     img(1:light_pos(5)+62, light_pos(i)-62:light_pos(i)+62, :) =
6         img(1:35+62, light_pos(i)-62:light_pos(i)+62, :)
7         .* (1 + light(29:125, :) / 1.5);
8     img(light_pos(6)-62:368, light_pos(i)-62:light_pos(i)+62, :) =
9         img(light_pos(6)-62:368, light_pos(i)-62:light_pos(i)+62, :)
10        .* (1 + light(1:100, :) / 1.5);
11 end
```

灯光效果模拟结果如图11、图12所示。



图 11: 灯光模拟前图像



图 12: 灯光模拟后图像

1.1.5 图像融合

最终融合方式较为简单，利用前背景分割 M ，将目标图像作为前景，参考图像作为背景即可，如下式所示：

$$\Theta = M \times Image + (1 - M) \times Refer \quad (4)$$

最终图像变换结果见2.1节。

1.2 效果二：The Queen's Theatre

1.2.1 问题描述

给定图片13，将其变换至 The Queen's Theatre 的广告牌中，如图14所示³。

分析该问题，可知事实上该图像相当于对两张图片进行变换，需要较好地复现该效果，需要达成以下几点要求：

- 对左图，需要简洁、准确地分割出多边形前景区域。
- 对左图，需要对目标图像进行刚体变换。
- 对右图，需要简洁、准确地分割出多边形前景区域。
- 对右图，需要对目标图像进行弹性变换。
- 对右图，需要调整目标图像不同区域的亮度，并产生对应的线条图案以模拟海报效果。
- 需要对原图进行色调等调整以符合图像风格。

³https://photofunia.com/categories/all_effects/queens_theatre

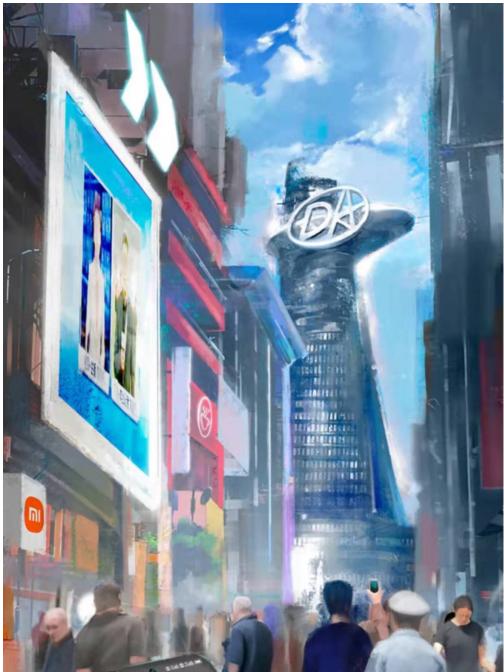


图 13: 待处理图像



图 14: 示例处理结果

1.2.2 基于几何区域限定法的前景分割

由于左图并不存在复杂的曲线，笔者考虑利用多边形分割方法分割出大致的轮廓，并利用超像素分割方法对细节进行处理。为实现多边形分割方法，需要产生多边形顶点，并利用 `inpolygon` 函数逐像素判断是否位于顶点围成的多边形区域内，由此产生对应的 M_1 。超像素分割相关操作方式与此前相同，此处不再赘述，由此产生对应的 M_2 。计算 $M = M_1 \cdot * M_2$ 即可。相关代码如下所示，其中，`poly_mask` 函数根据传入的目标图像和多边形顶点序列产生对应的 Mask 图像。

```

1 mask_point_left = [189,362;333,292;330,416;361,426;364,425;363,480;360,597;181,626];
2 mask1 = poly_mask( ref , mask_point_left );
3 mask2 = load( 'mask2.mat' ).BW;
4 mask = mask1 .* mask2;
5
6 function result = poly_mask(img,area)
7     [height , width , ~] = size(img);
8     result = zeros(height , width );
9     for i = 1:height
10         for j = 1:width
11             if inpolygon(j,i,area(:,1),area(:,2))
12                 result(i,j) = 1;

```

```

13         end
14     end
15 end
16 end

```

产生的 M_1, M_2, M 分别如图15、16、17所示。可以看到，最终的 M 同时考虑到了大轮廓和小细节，具有较好的分割效果。

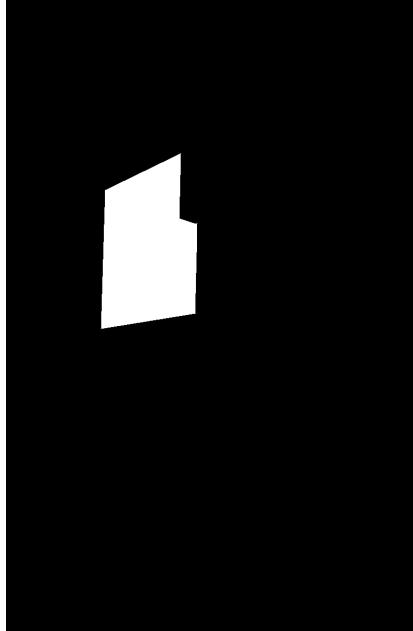


图 15: 多边形分割结果

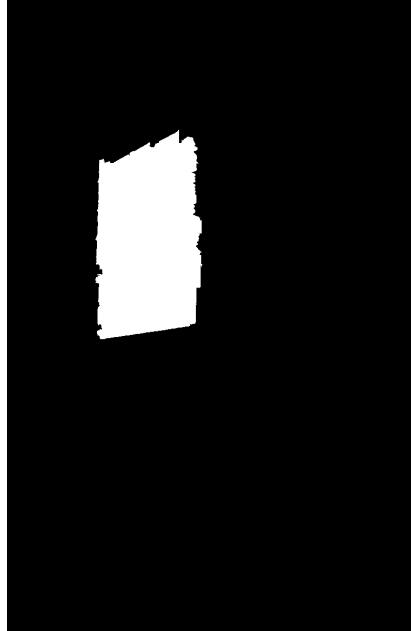


图 16: 超像素分割结果

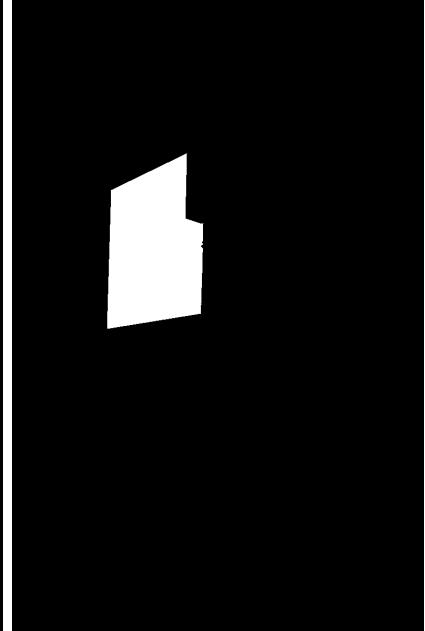


图 17: 最终分割结果

1.2.3 基于仿射变换的刚体变换效果

刚体变换基于仿射变换矩阵实现。在 Matlab 中，可以直接利用 *Computer Vision Toolbox* 中的 *estimateGeometricTransform* 函数计算 $[3 \times 3]$ 大小的仿射矩阵 T_{affine} 。上述函数需要传入模板图像、目标图像矩形四角的对应点坐标，生成仿射变换矩阵后，利用 *imwarp* 函数将其应用于目标图像中即可。相关代码如下所示。

```

1 affine_point = [189,362;369,273;360,597;181,626];
2 [height_img, width_img, ~] = size(img);
3 [height_ref, width_ref, ~] = size(ref);
4 area_src = [1,1;width_img,1;width_img,height_img;1,height_img];
5 affine_mat = estimateGeometricTransform(area_src, affine_point, 'projective');
6 img_left = imwarp(img, affine_mat, 'OutputView', imref2d([height_ref, width_ref]));

```

该部分较为简单，且效果较好，仿射变换结果如图18所示，将仿射变换与前景分割结果 M 融合后，结果如图19所示。

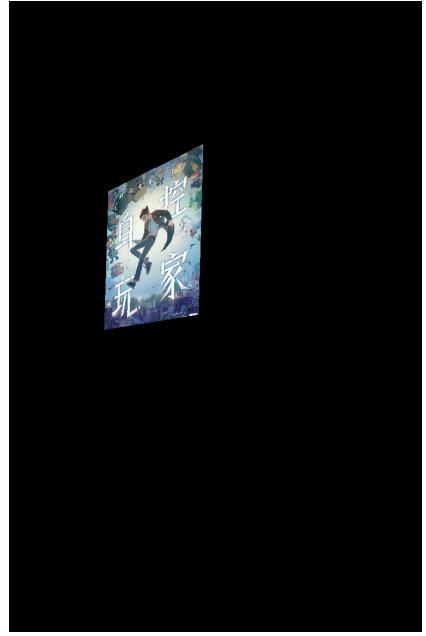


图 18: 仿射变换结果结果



图 19: 前景部分

最终，按照4相同方式将仿射变换后图片与背景图片融合即可。

1.2.4 图像油画风格模拟

为使图像更接近原图(油画手绘风格)，笔者尝试实现图像油画风格模拟。其基本思想为利用随机数进行控制，对图像中每个像素，随机利用其邻域内的像素进行替换，以此模拟油画风格中的颗粒感。相关代码如下所示。

```
1 img_style = zeros(height_img, width_img, 3);
2 for i=1:height_img
3     for j=1:width_img
4         temp=uint8(rand()*(3));
5         m=temp / 2;
6         n=mod(temp, 2);
7         h=mod(double(i - 1) + double(m), double(height_img));
8         w=mod(double(j - 1) + double(n), double(width_img));
9         if w==0
10             w = width_img;
11         end
12         if h==0
13             h = height_img;
14         end
15         img_style(i,j,:)=img(h,w,:);
16     end
17 end
18 img = img_style;
```

油画变换前、变换后相关细节如图20、21所示，可以看到，上述代码较好地实现了图像的颗粒化。



图 20: 油画风格变换前



图 21: 油画风格变换后

1.2.5 HSV 空间下饱和度调整

该部分原理较为简单，为降低图像饱和度，利用 HSV 色彩空间下 S 分量的值实现。将原图变换至 HSV 色彩空间，对 S 分量乘以系数后再变换回 RGB 色彩空间即可。相关代码如下所示。

```
1 img = rgb2hsv(img);  
2 img(:, :, 2) = img(:, :, 2) .* 0.7;  
3 img = hsv2rgb(img);
```

饱和度调整前、后图像如图22、23所示，可以看到，变换后图像饱和度明显降低。



图 22: 饱和度调整前



图 23: 饱和度调整后

1.2.6 线条模拟

观察到原图中存在纸质海报张贴时造成的线条，为复原线条效果，笔者直接产生等距的网格状图像，并将其按照一定比例叠加至原图像中。此处采用的叠加方式为直接叠加，为线条像素乘以 0.2 的系数以避免过于明显，此外，利用高斯滤波器对线条边缘进行柔化。相关代码如下所示，其中，*keyPoint* 为人工产生的等分关键点。

```

1 keyPoint_width = [1, 2, 3] * ceil(width_img / 4);
2 keyPoint_height = [1, 2, 3, 4, 5] * ceil(height_img / 6);
3 lines = zeros(height_img, width_img);
4 lineWidth = 2;
5 for i = 1:3
6     lines(:, keyPoint_width(i) - lineWidth:keyPoint_width(i) + lineWidth) = 1;
7 end
8 for i = 1:5
9     lines(keyPoint_height(i) - lineWidth:keyPoint_height(i) + lineWidth, :) = 1;
10 end
11 fs = fspecial('gaussian', [5, 5], 5);
12 lines = imfilter(lines, fs, 'replicate', 'same');
13 img = img + lines * 0.2;

```

模拟出的线条如图24所示，添加模拟线条后的图片如图25所示。可以看到，实现了较好的网格状线条效果。

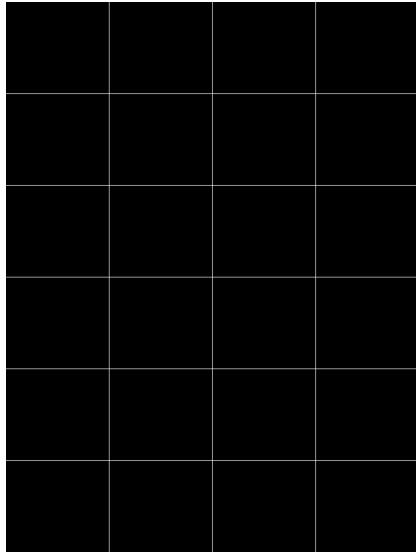


图 24: 线条图像



图 25: 添加线条后效果

1.2.7 图像分块 Gamma 校正

按照原图样式，将图像四等分，对原图亮度进行 Gamma 校正，对每一分块赋予不同的 Gamma 值以模拟光线产生的明暗效果。相关代码如下所示。

```

1 img(:, keyPoint_width(2):keyPoint_width(3), :) =
2     img(:, keyPoint_width(2):keyPoint_width(3), :) .^ 1.3;
3 img(:, keyPoint_width(3):width_img, :) =
4     img(:, keyPoint_width(3):width_img, :) .^ 1.6;
5 img(:, keyPoint_width(1):keyPoint_width(2), :) =
6     img(:, keyPoint_width(1):keyPoint_width(2), :) .^ 0.8;

```

Gamma 校正后的图像如图26所示。可以看到，图像沿不同纵坐标产生了不同的亮度。在后续效果展示部分，可见分块 Gamma 校正与线条模拟均在最终图像中取得了较好的效果。



图 26: 分块 Gamma 变换效果

1.2.8 基于对应点对的弹性变换

利用 Matlab 中的 *cpselect* 函数可以较为便捷地实现弹性变换。其采用交互式的实现方法，令用户手动标注目标图和参考图中的对应点，并由此产生由目标图到参考图的弹性变换结果，标注过程如图27所示。完成对应点标注后，可得到 *movingPoints*、*fixedPoints* 两组点对，利用 *fitgeotrans* 产生弹性变换矩阵 T_{elas} ，利用类似的方式将其作用于原图中即可。相关代码如下所示。

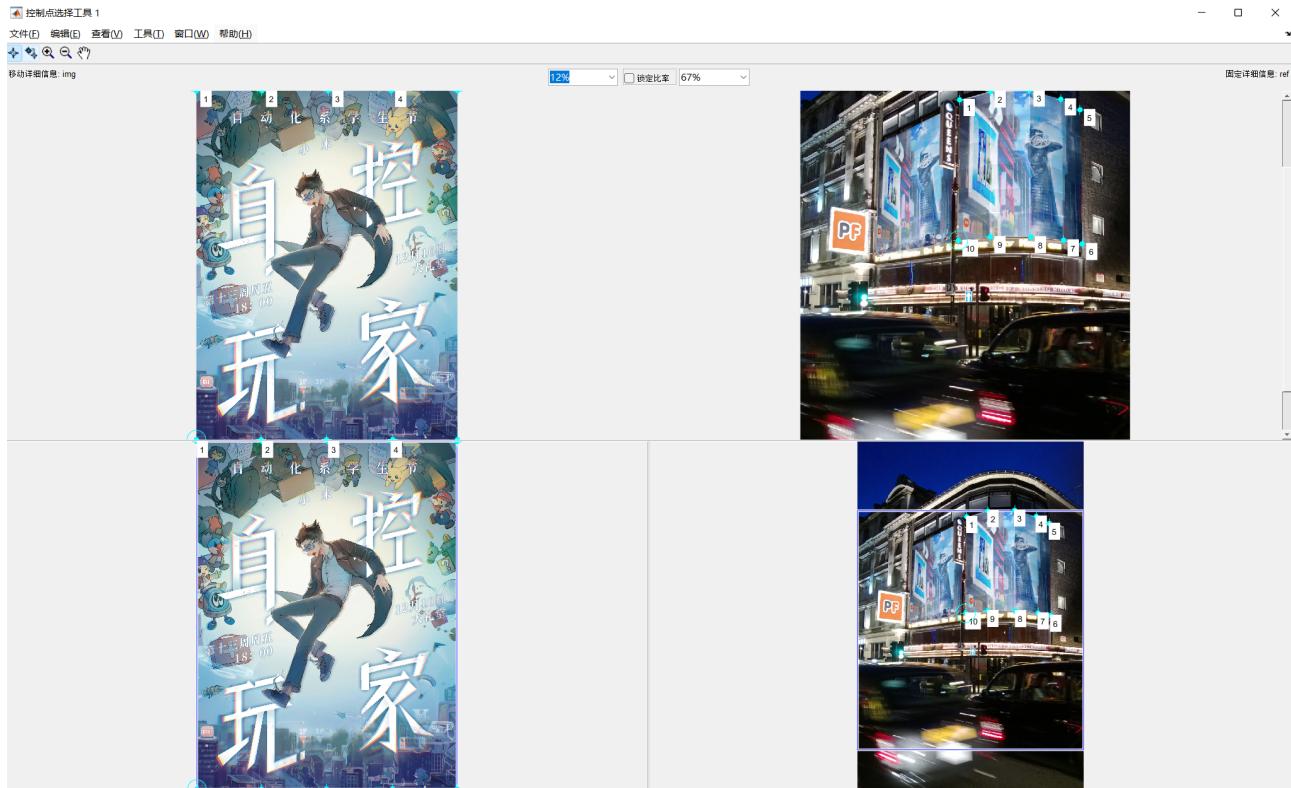


图 27: 弹性变换过程



图 28: 弹性变换结果

1.2.9 综合前景分割

为对右图进行前景分割，笔者利用多种方法提升分割效果。利用弹性变换自身有效部分产生 M_3 ，利用多边形分割产生 M_4 ，利用超像素分割产生 M_5 ，最终得到 $M = M_3.*M_4.*M_5$ 。

产生的 M_3 、 M_4 、 M_5 如图29、30、31所示。可以看到，多边形分割对弹性变换自身有效部分产生了进一步的约束，超像素分割为分割增加了细节，具有较好的分割效果。

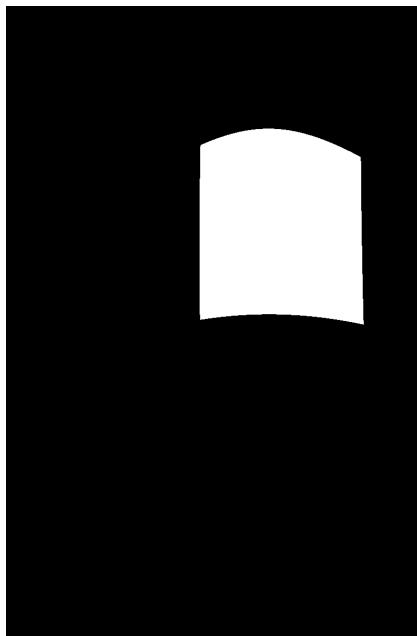


图 29: 弹性变换分割结果

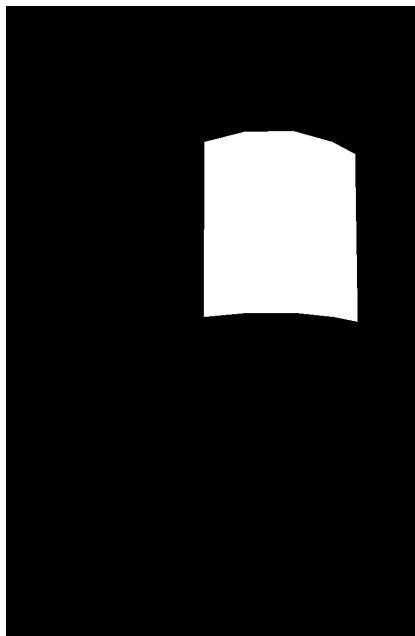


图 30: 多边形分割结果

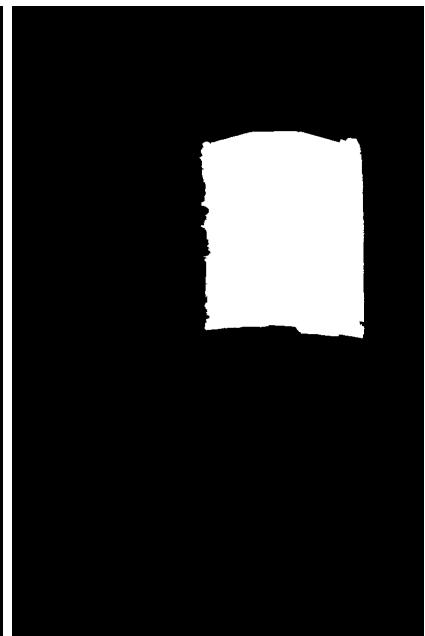


图 31: 超像素分割结果

1.2.10 图像融合

利用公式4，根据前景和背景完成图像融合过程即可。

最终图像变换结果见2.2节。

1.3 效果三：Snow Globe

1.3.1 问题描述

给定图片32，将其变换至 Snow Globe 的水晶球中，如图33所示⁴。

分析该问题，可知需要较好地复现该效果，需要达成以下几点要求：

- 对有较多复杂遮挡的前景进行准确分割。
- 处理图像边缘，与水晶球相关光效较好融合。
- 模拟图片中的雪花效果。



图 32: 待处理图像



图 33: 示例处理结果

1.3.2 综合前景分割

前景分割是本图效果实现的关键之一，由于相关区域不规则，笔者此处主要利用细致的超像素分割结合形态学运算实现。由于利用超像素分割会将原图中的白色雪花漏去，故利用 `bwareaopen` 函数去除较小的雪花点，使前景部分为完整的区域。相关代码如下所示。

```
1 [height, width, ~] = size(ref);  
2 ROI = [160, 823, 93, 627];
```

⁴https://photofunia.com/categories/all_effects/snow-globe

```

3 ROI_mask = zeros(height, width);
4 ROI_mask(ROI(3):ROI(4), ROI(1):ROI(2)) = 1;
5 BW_mask = load('BW.mat').BW;
6 mask = BW_mask .* ROI_mask;
7 mask = ~(bwareaopen(~mask, 64, 8));

```

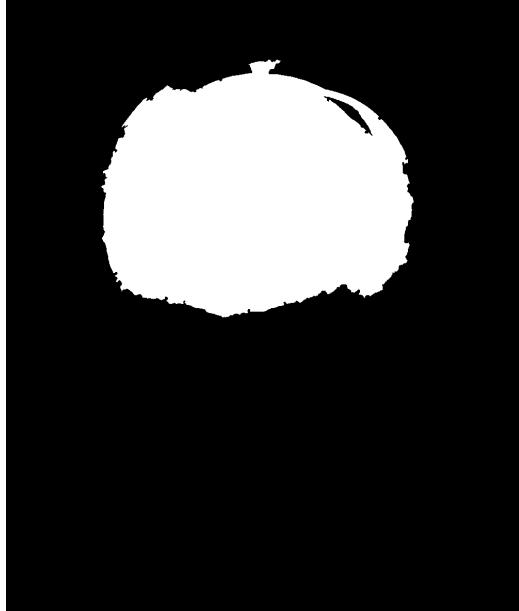


图 34: 前景分割结果

1.3.3 边缘提取与羽化

若直接将所得前景按照公式4进行融合，前景与背景的过渡将非常生硬。因此，笔者考虑对边缘进行提取与锐化操作。设定一较大的结构元素，提取较粗的边缘后，利用高斯滤波器对边缘进行羽化处理，将羽化后的边缘与形态学腐蚀运算后的原前景区域进行相加，归一化后即可得边缘羽化后的前景区域。相关代码如下所示。

```

1 se = strel('disk', 60);
2 mask_edge = mask - imerode(mask, se);
3 fs = fspecial('gaussian', [30, 30], 30);
4 mask_edge = Normalize(imfilter(mask_edge, fs));
5 se = strel('disk', 30);
6 mask = imerode(mask, se) + mask_edge;
7 mask(mask > 1) = 1;

```

原前景图如图35所示，提取出的边缘如图36所示，羽化后的边缘如图37所示，最终前景图如图38所示。

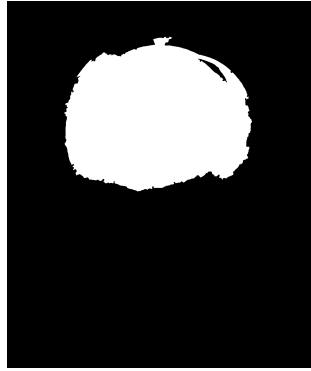


图 35: 原前景图

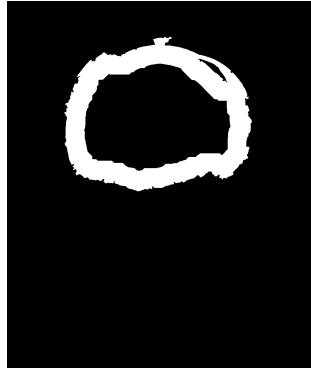


图 36: 边缘图



图 37: 羽化边缘图



图 38: 羽化前景图

如图39、40所示，对比羽化前、后的图片融合效果，明显可见，羽化使得图像效果有大幅提高。



图 39: 羽化处理前



图 40: 羽化处理后

1.3.4 雪花效果模拟

笔者利用随机点模拟雪花效果。新建与原图大小相同的黑色矩阵，按照 $p = 0.002$ 的概率产生随机白色点，利用形态学膨胀操作对白色点进行扩充，完成扩充后利用高斯滤波器柔化边缘即可模拟雪花效果。需要注意的是，此处需要利用前景部分限定雪花范围。相关代码如下所示。

```
1 mask_snow = zeros(height, width);
2 amount = fix(height * width * 0.002);
3 for j=1:amount
4     x = randi(height, 1, 1);
5     y = randi(width, 1, 1);
6     mask_snow(x, y)=1;
7 end
8 mask_snow = mask_snow .* mask_init;
9 se = strel('disk', 2);
10 mask_snow = imdilate(mask_snow, se);
```

```

11 fs = fspecial('gaussian', [3,3], 5);
12 mask_snow = imfilter(mask_snow, fs);

```

产生雪花效果如图41、42所示。

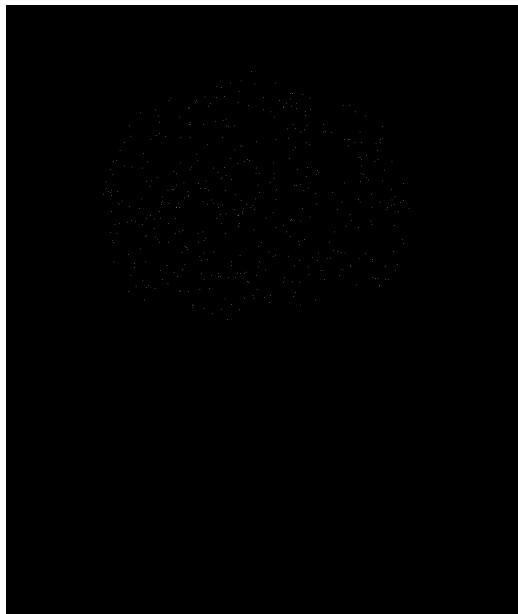


图 41: 随机白色点

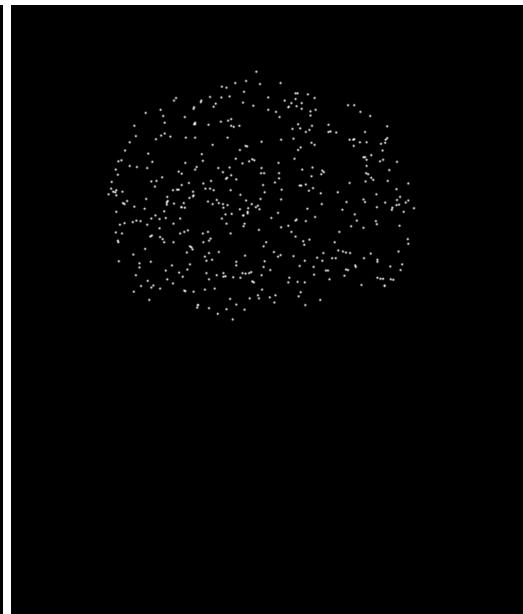


图 42: 雪花效果模拟

1.3.5 图像融合

对原图进行插值缩放到合适大小，利用羽化后的前景分割矩阵切分前景和背景，分别赋值，最后叠加雪花矩阵即可。相关代码如下所示。

```

1 pic_region = [163,823,93,646];
2 img = imresize(img, [pic_region(4)-pic_region(3)+1
3   pic_region(2)-pic_region(1)+1], 'bicubic');
4 result = zeros(height, width, 3);
5 result(pic_region(3):pic_region(4), pic_region(1):pic_region(2), :) = img;
6 result = result .* mask;
7 result = result + (1 - mask) .* black;
8 result = result + mask_snow * 0.5;
9 imshow(result);

```

最终图像变换结果见2.3节。

2 结果展示

2.1 效果一: Shopping Arcade

目标图像



图 43: 目标图像

模板图像



图 44: 模板图像

变换结果

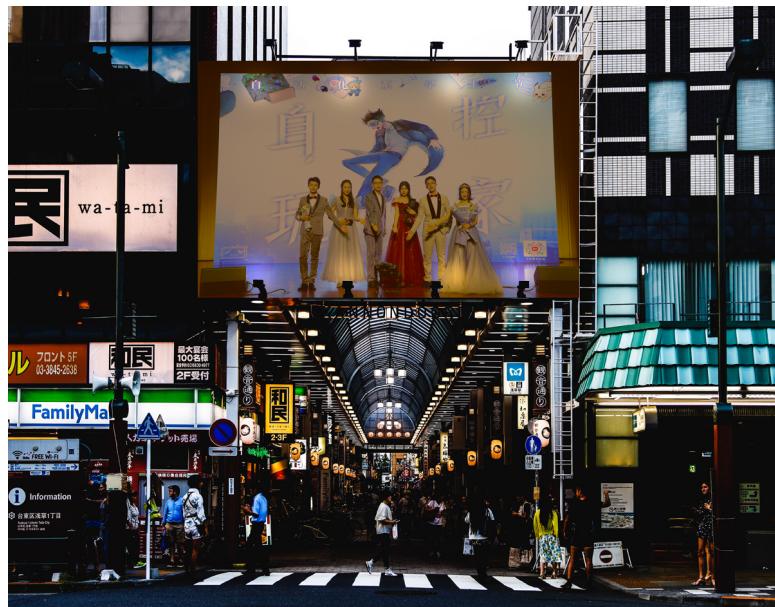


图 45: 变换结果

2.2 效果二：The Queen's Theatre

目标图像



图 46: 目标图像

模板图像



图 47: 模板图像

变换结果



图 48: 变换结果

2.3 效果三: *Snow Globe*

目标图像



图 49: 目标图像

模板图像

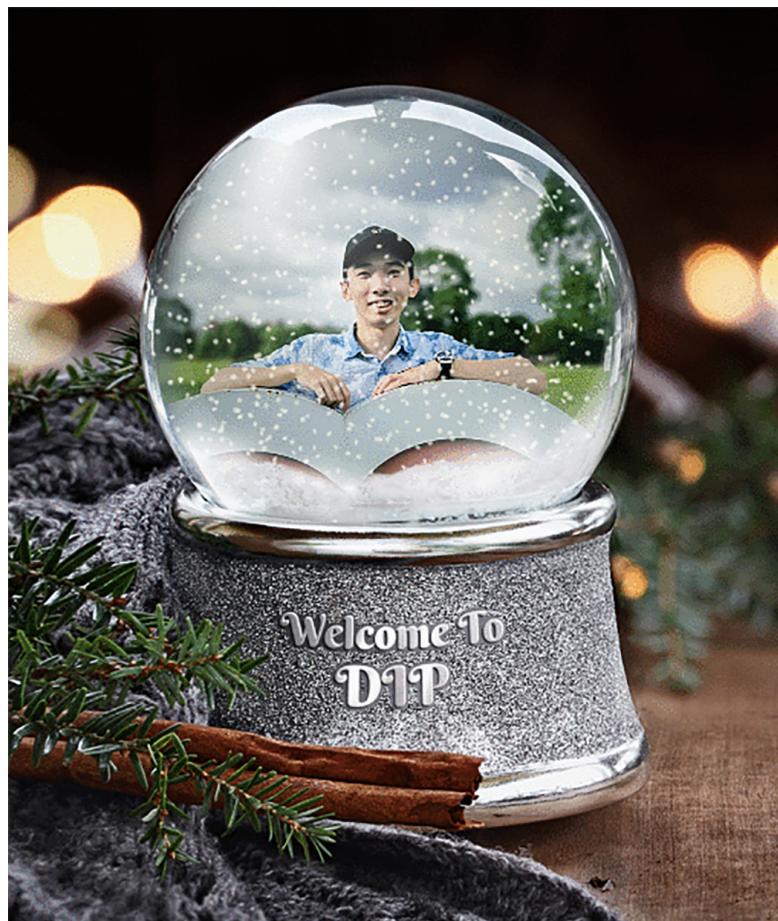


图 50: 模板图像

变换结果



图 51: 变换结果

3 收获与体会

- 对数字图象处理课程知识的整体梳理

得益于数图期末考试，笔者把整个学期学到的数图知识融会贯通地温习了一遍，在本次综合大作业中，笔者得以把一学期所有学到的数图知识在各种图片效果中进行应用，笔者在此次大作业中几乎用到了本学期所有章节的知识，包括但不限于滤波、校正、噪声与去噪、色彩空间、形态学运算、分割、配准等，也对很多图像效果与经典图像算法进行了探究。在做本次大作业时，笔者感受到了一种“信手拈来”的乐趣，这也是学习数字图象处理课程的意义所在。

- 工程能力与复现模仿能力

笔者通过此次大作业很大程度锻炼了自己的工程能力、代码能力以及对结果的逆向工程能力。我们所拥有的只是图像的结果，需要从结果出发，结合所学知识思考什么操作能够达成这样的结果。在反复思考的过程中，笔者的复现能力与对知识的掌握水平有了很大程度的提高。

- 精益求精的要求

尽管笔者仍然未能做到百分百复现模板图像，笔者仍然可以说，自己花费了大量时间尝试分析模板图像的特点，尝试靠近模板图像的内容。在这个过程中，笔者体会到了工程过程中的精益求精的思维，也进一步锻炼了自己的能力和心智。

- 对 Matlab 软件的正确运用

笔者在完成此次大作业的过程中，体会到了 Matlab 软件的强大之处。很多复杂的算法在 Matlab 中只是一句简单的语句，甚至能够通过可视化界面简单地实现。这启示笔者开发环境对开发速度和效率有很大的影响作用，我们一方面要学会用各类开发环境，另一方面也要学会制造开发环境。

4 后记

尽管数字图象处理课程可以说是本学期任务量最大的课程，但笔者仍然觉得非常值得。所学的数字图象处理算法让笔者对图像领域世界有了全新角度的认识，完成的各类小作业、大作业也让笔者充分锻炼了自己的代码能力和工程能力。在最后一次大作业的过程中，笔者真正体会到了数字图象处理带来的乐趣以及它巨大的作用。这门课程通过较大的任务量让同学们能够真正地将知识消化吸收，这一点本身就非常难得。希望在数字图象处理课程中学到的内容能够为自己的未来学习、科研生活有更多的帮助。

最后，感谢老师和助教一个学期以来的辛勤付出！