数字图象处理

课程作业一

2019010485　自 91　刘祖炎*

2021 年 10 月 3 日

1　算法原理与代码分析

1.1　灰度变换类

1.1.1　基本淡入淡出

基本淡入淡出用于实现从变换前图片淡入淡出至变换后图片。其实现原理较为简单，对于变换前图片 Θ_1 与变换后图片 Θ_2，利用公式即可求出当前图片帧：

$$\Theta = \Theta_1 \cdot \lambda + \Theta_2 \cdot (1 - \lambda) \tag{1}$$

其中 $\lambda \in [0, 1]$ 为从 1 到 0 逐渐变化的浮点数。

实现代码如下所示：

```
1  image_start = image_collect{image_idx};
2  image_end = image_collect{image_idx + 1};
3  for factor = 0:0.02:1
4      current_frame = image_start * (1 - factor) + image_end * factor;
5      imshow(current_frame);
6      pause(0.001);
7      imwrite(current_frame, [new_main_dir, num2str(save_idx,'%05d'), file_type]);
8      save_idx = save_idx + 1;
9  end
10 image_idx = image_idx + 1;
```

第 1、2 行代码用于从 *image_collect* 中读取图片，并在循环中产生当前视频帧。第 7 行代码用于将当前视频帧按照顺序存储至目标文件夹中，并在完成存储后将 *index* 增加用于下一帧存储。上述代码在每类变换中均以相同方法使用，在后续部分不再赘述。

核心代码为第 4 行，与公式1相同。

*liuzuyan19@mails.tsinghua.edu.cn

1.1.2 棋盘格淡入淡出

棋盘格淡入淡出用于实现以棋盘格的效果从变换前图片淡入淡出至变换后图片。对于变换前图片 Θ_1 与变换后图片 Θ_2，生成棋盘格遮罩 M。在第一阶段，利用 M 固定 Θ_1 的部分图像不变，另一部分仿照上例应用淡入淡出；第二阶段，利用 $\sim M$ 固定 Θ_2 已完成淡入淡出的部分图像不变，另一部分仿照上例应用淡入淡出，即可实现棋盘格淡入淡出的效果。

两阶段利用公式可分别表示为：

$$\Theta = M \cdot \Theta_1 + (\sim M) \cdot (\Theta_1 \cdot \lambda + \Theta_2 \cdot (1 - \lambda)) \tag{2}$$

$$\Theta = (\sim M) \cdot \Theta_2 + M \cdot (\Theta_1 \cdot \lambda + \Theta_2 \cdot (1 - \lambda)) \tag{3}$$

其中 $\lambda \in [0, 1]$ 为从 1 到 0 逐渐变化的浮点数，M 为棋盘格遮罩， M 为 M 的逻辑反。

实现代码如下所示：

```
1  unit_checker = 100;
2  x_checker = ceil(600 / unit_checker / 2);
3  y_checker = ceil(max_Y / unit_checker / 2);
4  checker = checkerboard(unit_checker, x_checker, y_checker) > 0.5;
5  checker = checker(1:600, 1:max_Y);
6  image_start = image_collect{image_idx};
7  image_end = image_collect{image_idx + 1};
8  for factor = 0:0.04:1
9      current_frame = image_start .* checker + image_end .* (~checker) * factor +
10     image_start .* (~checker) * (1 - factor);
11     imshow(current_frame);
12     pause(0.001);
13     imwrite(current_frame, [new_main_dir, num2str(save_idx,'%05d'), file_type]);
14     save_idx = save_idx + 1;
15 end
16 for factor = 0:0.04:1
17     current_frame = image_end .* (~checker) + image_start .* checker * (1 - factor) +
18     image_end .* checker * factor;
19     imshow(current_frame);
20     pause(0.001);
21     imwrite(current_frame, [new_main_dir, num2str(save_idx,'%05d'), file_type]);
22     save_idx = save_idx + 1;
23 end
24 image_idx = image_idx + 1;
```

首先定义棋盘格单位宽度 *unit_checker* 为 100 像素，并由图像长宽计算 x、y 方向棋盘数量 *x_checker*，*y_checker*。利用系统函数 *checkerboard* 产生棋盘遮罩 *checker* = M(由于函数会产生灰度图像，故此处需要对 M 进行归一化处理)，并将 M 处理为与图片大小相同的遮罩。

此后在第 9、10 行运用公式2，在第 17、18 行运用公式3进行计算即可。

1.1.3 按方向淡入淡出

按方向淡入淡出用于实现沿某一方向的淡入淡出效果。其实现原理为根据横坐标的大小赋予不同的增益值，使得图片右侧比图片左侧的淡入淡出速度更快。假设某像素横坐标位置值为 Γ，对于变换前图片 Θ_1 与变换后图片 Θ_2，最终图片可用公式表示为：

$$\Theta = \Gamma \cdot (\Theta_1 \cdot \lambda + \Theta_2 \cdot (1 - \lambda)) \tag{4}$$

实现代码如下所示：

```
1  image_start = image_collect{image_idx};
2  image_end = image_collect{image_idx + 1};
3  speed = 5;
4  adjust = (1:1:max_Y);
5  adjust = adjust / max_Y * speed;
6  for factor = 0:0.02:1
7      grey_num = ones(1, max_Y) * factor;
8      grey_num = min(grey_num .* (1 + adjust), 1);
9      grey_num = repmat(grey_num, 600, 1, 3);
10     current_frame = image_end .* grey_num + image_start .* (1 - grey_num);
11     imshow(current_frame);
12     pause(0.001);
13     imwrite(current_frame, [new_main_dir, num2str(save_idx,'%05d'), file_type]);
14     save_idx = save_idx + 1;
15 end
16 image_idx = image_idx + 1;
```

根据公式4可知，公式实现的关键在于求出各个像素值的增益值 Γ。在代码 4、5 行中，首先生成长度为图片宽度的数组 *adjust*，其值为 0-1 的均匀递增数组，则该数组即可作为增益值作用于灰度值数组 *grey_num* 上，当 *grey_num* 的基准值随 *factor* 变化时，其变化速度即会发生变化。获取了 *grey_num* 数组后的操作与基本淡入淡出类似，依照公式4进行计算即可。

1.2 几何变化类

1.2.1 3D 飞入飞出

3D 飞入飞出用于实现图片沿空间进行几何变化的效果。此处算法结合了图片的平移、旋转、仿射变换，对于变换前图片 Θ_1 与变换后图片 Θ_2，构造几何变换矩阵：

$$T_{scale} = \begin{bmatrix} \theta & 0 & 0 \\ 0 & \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T_{affine} = \begin{bmatrix} 1 & \theta & 0 \\ \theta & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T_{rotate} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

并利用公式得到当前帧图片：

$$\Theta = \omega(T_{rotate}, \omega(T_{affine}, \omega(T_{scale}, \Theta_1))) \tag{5}$$

求得经过三次变换后的图像。其中，ω 表示将矩阵作用于原图上。

实现代码如下所示：

```
image_start = image_collect{image_idx};
image_end = image_collect{image_idx + 1};
for factor = 0.04:0.04:0.96
    angle = factor * 2;
    T_scale = affine2d([(1 - factor) 0 0; 0 (1 - factor) 0; 0 0 1]);
    T_aff = affine2d([1 factor 0; factor 1 0; 0 0 1]);
    T_rotate = affine2d([cos(angle) -sin(angle) 0; sin(angle), cos(angle) 0; 0 0 1]);
    current_frame = imwarp(image_start, T_scale);
    current_frame = imwarp(current_frame, T_rotate);
    current_frame = imwarp(current_frame, T_aff,
    'OutputView', imref2d(size(image_start)));
    imshow(current_frame);
    pause(0.001);
    imwrite(current_frame, [new_main_dir, num2str(save_idx,'%05d'), file_type]);
    save_idx = save_idx + 1;
end
for factor = 0.04:0.04:0.96
    angle = (1 - factor) * 2;
    T_scale = affine2d([factor 0 0; 0 factor 0; 0 0 1]);
    T_aff = affine2d([1 (1 - factor) 0; (1 - factor) 1 0; 0 0 1]);
    T_rotate = affine2d([cos(angle) -sin(angle) 0; sin(angle), cos(angle) 0; 0 0 1]);
    current_frame = imwarp(image_end, T_scale);
    current_frame = imwarp(current_frame, T_rotate);
    current_frame = imwarp(current_frame, T_aff,
    'OutputView', imref2d(size(image_start)));
    imshow(current_frame);
    pause(0.001);
    imwrite(current_frame, [new_main_dir, num2str(save_idx,'%05d'), file_type]);
    save_idx = save_idx + 1;
end
```

```
image_idx = image_idx + 1;
```

效果利用两次循环实现，分别用于变换前图片 Θ_1 的消失与变换后图片 Θ_2 的出现。依照上述矩阵构造方式分别构造 T_{scale}、T_{affine}、T_{rotate}，其中参数 θ 随着循环次数而发生变化，以产生运动效果。此后，利用 *imwarp* 函数将变换矩阵依次作用于图片中。需要注意的是，此处需要利用 $'OutputView'$ 参数规定生成图片的长宽，否则产生的图片尺寸将会发生变化。

1.2.2 投影翻页

投影翻页用于实现图片在空间中投影转场的效果，利用了图像的投影变换特性。对于待变换图片 Θ_1，通过计算获取其起始点与目标点 P_1，P_2，并生成投影矩阵 T_{proj}。利用公式：

$$\Theta = \omega(T_{proj}, \Theta_1) \tag{6}$$

即可求出当前图片帧。

实现代码如下所示：

```
image_start = image_collect{image_idx};
image_end = image_collect{image_idx + 1};
init = [1 1; max_Y 1; max_Y 600; 1 600];
point1 = [200 250; 400 200; 400 400; 200 350];
point2 = [880 200; 1080 250; 1080 350; 880 400];
for factor = 0:0.04:1
    current_point = factor * point1 + (1 - factor) * init;
    T_proj = fitgeotrans(init, current_point, 'Projective');
    current_frame = imwarp(image_start, T_proj,
    'OutputView', imref2d(size(image_start)));
    imshow(current_frame);
    pause(0.001);
    imwrite(current_frame, [new_main_dir, num2str(save_idx,'%05d'), file_type]);
    save_idx = save_idx + 1;
end
for factor = 0:0.04:1
    current_point = (1 - factor) * point2 + factor * init;
    T_proj = fitgeotrans(init, current_point, 'Projective');
    current_frame = imwarp(image_end, T_proj, 'OutputView', imref2d(size(image_start)));
    imshow(current_frame);
    pause(0.001);
    imwrite(current_frame, [new_main_dir, num2str(save_idx,'%05d'), file_type]);
    save_idx = save_idx + 1;
end
image_idx = image_idx + 1;
```

效果利用两次循环实现，分别用于变换前图片 Θ_1 的消失与变换后图片 Θ_2 的出现。由于图片在开始变换前或完成变换后其坐标均为 $(1,1),(1,max_Y),(max_X,1),(max_X,max_Y)$，故只需定义图片的终点或起

点状态时四角位置，并利用线性插值的方法计算当前状态下图片应当处于的位置。利用函数 $fitgeotrans$ 即可求出两点之间对应的投影变换矩阵 T_{proj}，同样利用 $imwrap$ 将变换矩阵作用于图片中即可。

1.2.3 单页翻转

单页翻转用于实现图片在空间中翻转转场的效果。对于待变换图片 Θ_1，对其 Y 轴方向进行压缩，压缩比例随着循环进行不断增大或减小即可实现。

实现代码如下所示：

```
1  image_start = image_collect{image_idx};
2  image_end = image_collect{image_idx + 1};
3  for factor = 0:0.04:0.96
4      image_resize = imresize(image_start, [600, ceil(max_Y * (1 - factor))], 'bicubic');
5      current_frame = zeros(600, max_Y, 3);
6      start_pos = ceil((max_Y - ceil(max_Y * (1 - factor))) / 2) + 1;
7      end_pos = ceil((max_Y + ceil(max_Y * (1 - factor))) / 2);
8      current_frame(:, start_pos:end_pos, :) = image_resize;
9      imshow(current_frame);
10     pause(0.001);
11     imwrite(current_frame, [new_main_dir, num2str(save_idx,'%05d'), file_type]);
12     save_idx = save_idx + 1;
13 end
14 for factor = 0.04:0.04:1
15     image_resize = imresize(image_end, [600, ceil(max_Y * factor)], 'bicubic');
16     current_frame = zeros(600, max_Y, 3);
17     start_pos = ceil((max_Y - ceil(max_Y * factor)) / 2) + 1;
18     end_pos = ceil((max_Y + ceil(max_Y * factor)) / 2);
19     current_frame(:, start_pos:end_pos, :) = image_resize;
20     imshow(current_frame);
21     pause(0.001);
22     imwrite(current_frame, [new_main_dir, num2str(save_idx,'%05d'), file_type]);
23     save_idx = save_idx + 1;
24 end
25 image_idx = image_idx + 1;
```

效果利用两次循环实现，分别用于变换前图片 Θ_1 的消失与变换后图片 Θ_2 的出现。利用 $image\_resize$ 函数即可求出当前经过尺度压缩的图片。为确保图片尺寸不变，需要将经过压缩的图片与黑色背景相加。第 6、7 行利用简单的公式计算图片应当处于的位置，并在第 8 行对黑色背景进行赋值即可。

1.3 其他操作

1.3.1 图像预处理

图像预处理代码如下：

```
1  for i = 1:len
2      image_name{i} = image_files(i).name;
```

```
    image = imread([main_dir, image_name{i}]);
    image = im2double(image);
    [height, width, channel] = size(image);
    width = width / height * 600;
    scale = 600 / height;
    image = imresize(image, scale, 'bicubic');
    if width > max_Y
        max_Y = width;
    end
    image_collect{i} = image;
end

for i = 1:len
    back_bg = zeros(600, max_Y, 3);
    [height, width, channel] = size(image_collect{i});
    start_pos = (max_Y - width) / 2 + 1;
    end_pos = (max_Y + width) / 2;
    back_bg(:, start_pos:end_pos, :) = image_collect{i};
    image_collect{i} = back_bg;
end
```

利用 *imread* 函数读取图像，利用 *im2double* 函数将图像转化为浮点数，利用 *imresize* 函数对图像大小进行归一化处理。将完成处理后的图像放入 *image_collect* 中。此后，循环读取 *image_collect* 中的图像，并将其置于背景图中，使得各图像的长、宽完全相同，便于后续效果处理。

1.3.2 视频生成

视频生成代码如下：

```
animation = VideoWriter('photo_album','MPEG-4');、
animation.FrameRate = 25;
open(animation);
image_files = dir([new_main_dir,'*',file_type]);
len = length(image_files);
for i=1:len
    image_name{i} = image_files(i).name;
    current_frame = imread([new_main_dir, image_name{i}]);
    writeVideo(animation, current_frame);
end
close(animation);
```

2 实验结果与分析

经过实验验证与调试，各算法均达到了预期效果，部分实验结果截图参见下例。为进一步观察算法效果，可参考随报告一同提交的视频。

2.1 灰度变换类

2.1.1 基本淡入淡出

图 1: 基本淡入淡出 (1)

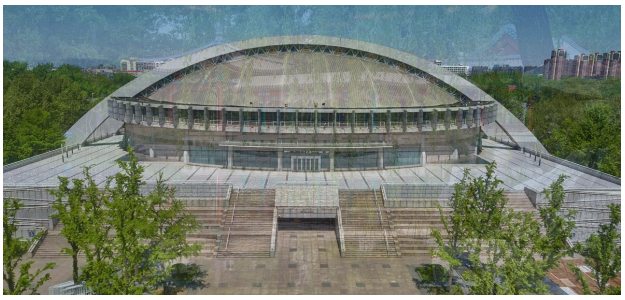

图 2: 基本淡入淡出 (2)

2.1.2 棋盘格淡入淡出

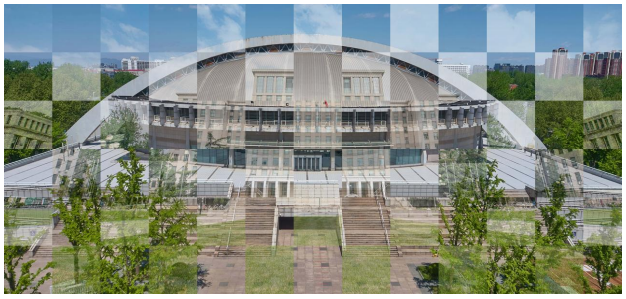

图 3: 棋盘格淡入淡出 (1)

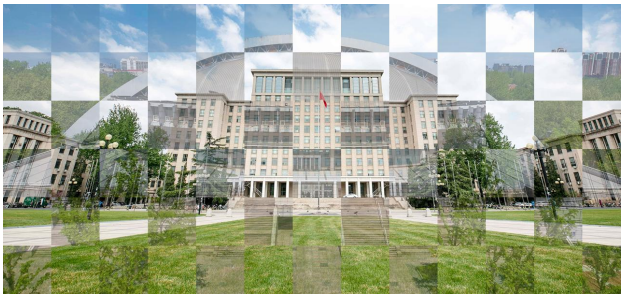

图 4: 棋盘格淡入淡出 (2)

2.1.3 按方向淡入淡出

图 5: 按方向淡入淡出 (1)

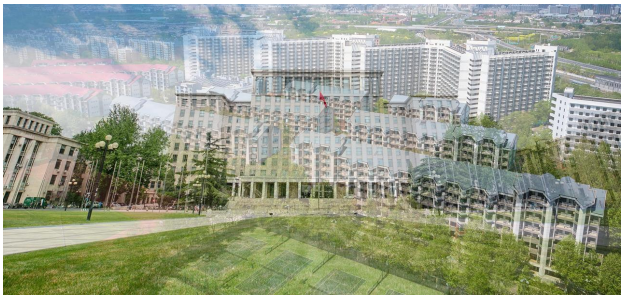

图 6: 按方向淡入淡出 (2)

图 7: 按方向淡入淡出 (3)

图 8: 按方向淡入淡出 (4)

2.2 几何变化类

2.2.1 3D 飞入飞出

图 9: 3D 飞入飞出 (1)

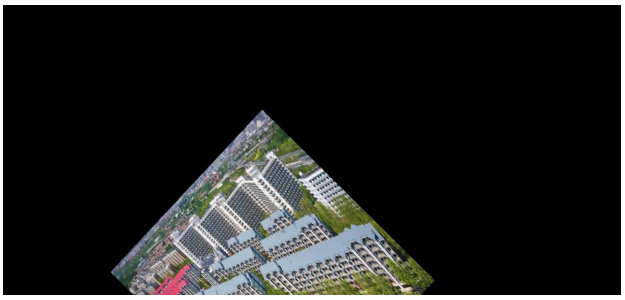

图 10: 3D 飞入飞出 (2)

图 11: 3D 飞入飞出 (3)

图 12: 3D 飞入飞出 (4)

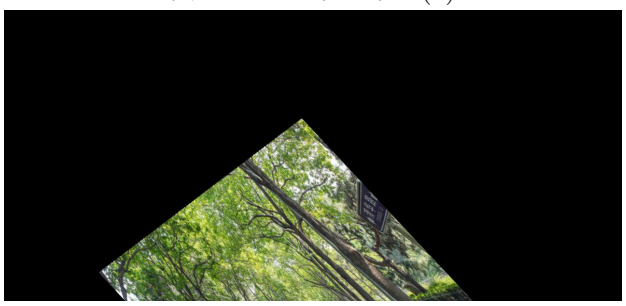

图 13: 3D 飞入飞出 (5)

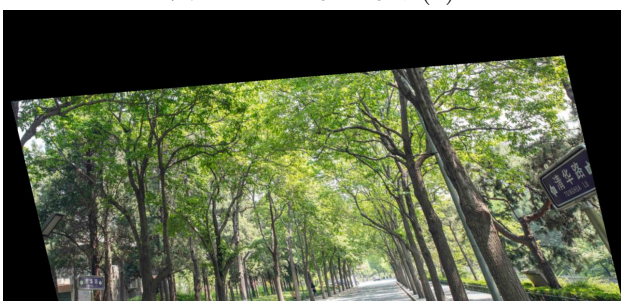

图 14: 3D 飞入飞出 (6)

2.2.2 投影翻页

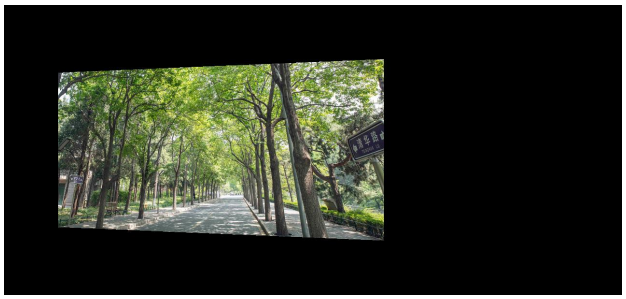

图 15: 投影翻页 (1)

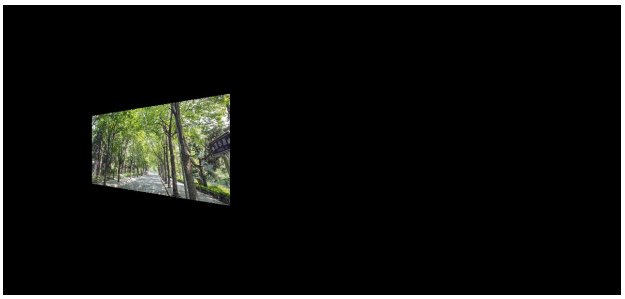

图 16: 投影翻页 (2)

图 17: 投影翻页 (3)

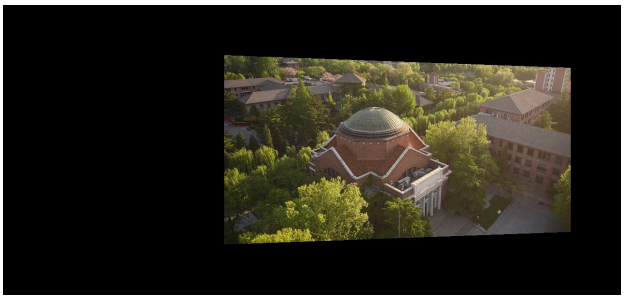

图 18: 投影翻页 (4)

2.2.3 单页翻转

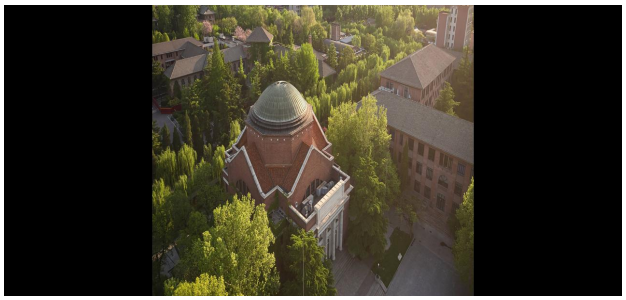

图 19: 单页翻转 (1)

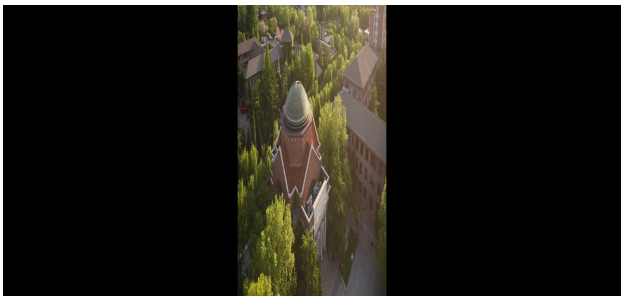

图 20: 单页翻转 (2)

图 21: 单页翻转 (3)

图 22: 单页翻转 (4)

3 遇到的困难和解决办法

由于此次作业逻辑上较为简单，故没有遇到较多困难。在熟悉了 Matlab 语言规范以及相关操作后，其他问题的解决均较为顺利。具体而言，主要的困难在于几何变换效果的实现。由于 $imrotate$、$imresize$ 等函数提供的效果过于简单，我参照 Matlab 官方文档学习了 $imwarp$、$affine2d$ 等函数的用法，较为完

美地实现了几何变换的效果。

-

4 收获

- Matlab 读取图片、操作图片、保存图片、视频生成的基本操作。
- Matlab 矩阵操作的方法。
- $imresize$ 等图像处理基本函数的使用方法。
- 图像灰度变换效果的实现方法。
- 图像几何变换矩阵的原理以及生成方法，$imwarp$、$affine2d$、$fitgeotrans$ 等几何变换函数的使用。

5 可能的改进方向

- 更加复杂的效果实现。(例如 PowerPoint 软件中折纸、翻页、幕布、涟漪等效果的实现方法。)
- 非线性动画的效果实现。(相比于本次作业使用的线性动画，非线性动画会使切换过程更加自然，也是目前各大软件中广泛采用的方法。)