数字图像处理

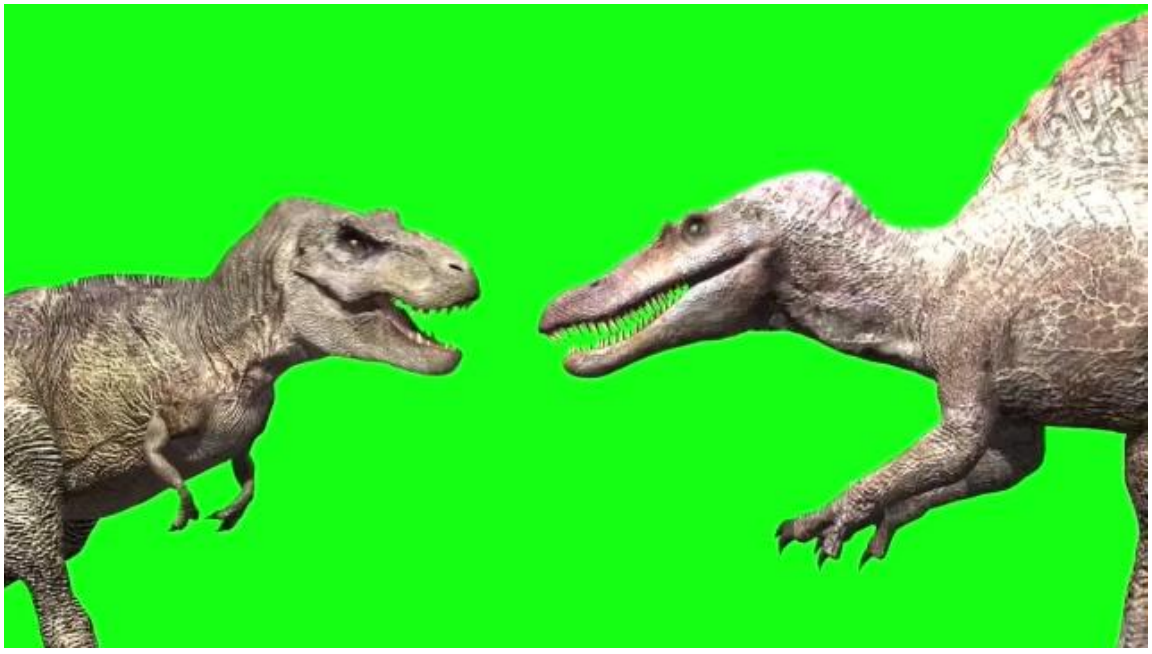
作业 1：视频合成

一. 项目简介

利用现有的背景视频素材和前景视频素材，使用代数运算实现简单的视频合成，达到背景替换等效果。

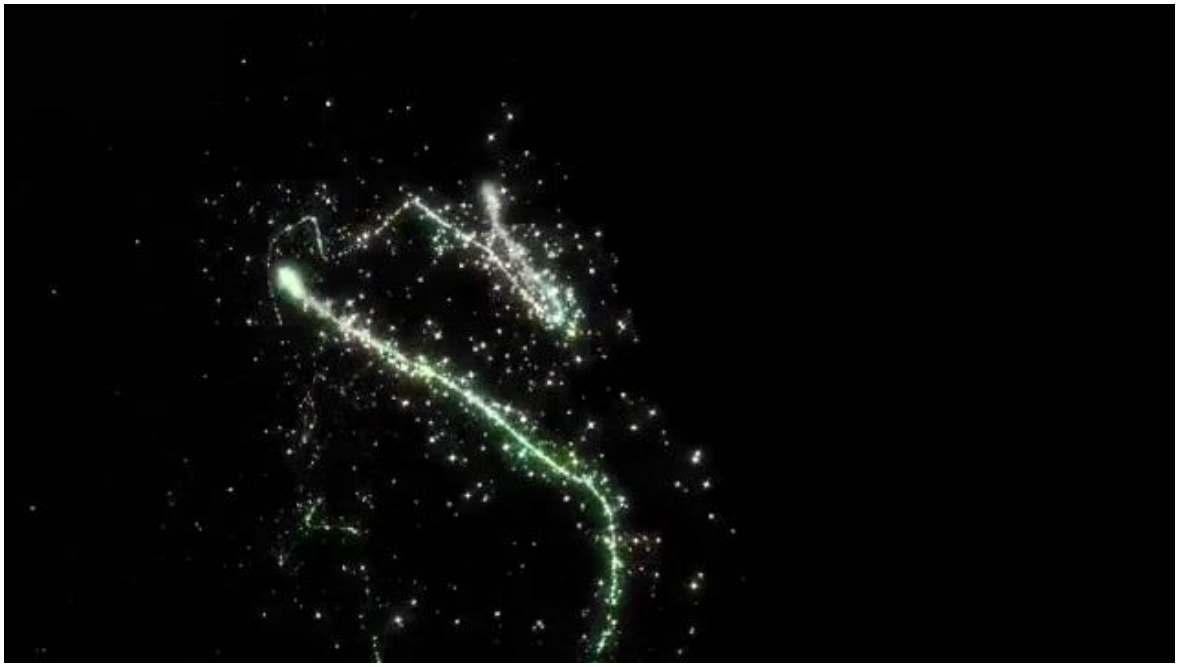
二. 设计文档

本案例采用的素材都在本文件夹中，分别是 origin.mp4，sky.mp4，light.mp4，分别作为前景视频、背景视频和装饰视频来使用。截图如下：



前景视频截图

背景视频截图



装饰视频截图

合成视频截图

下面考虑以四种方式实现视频合成。

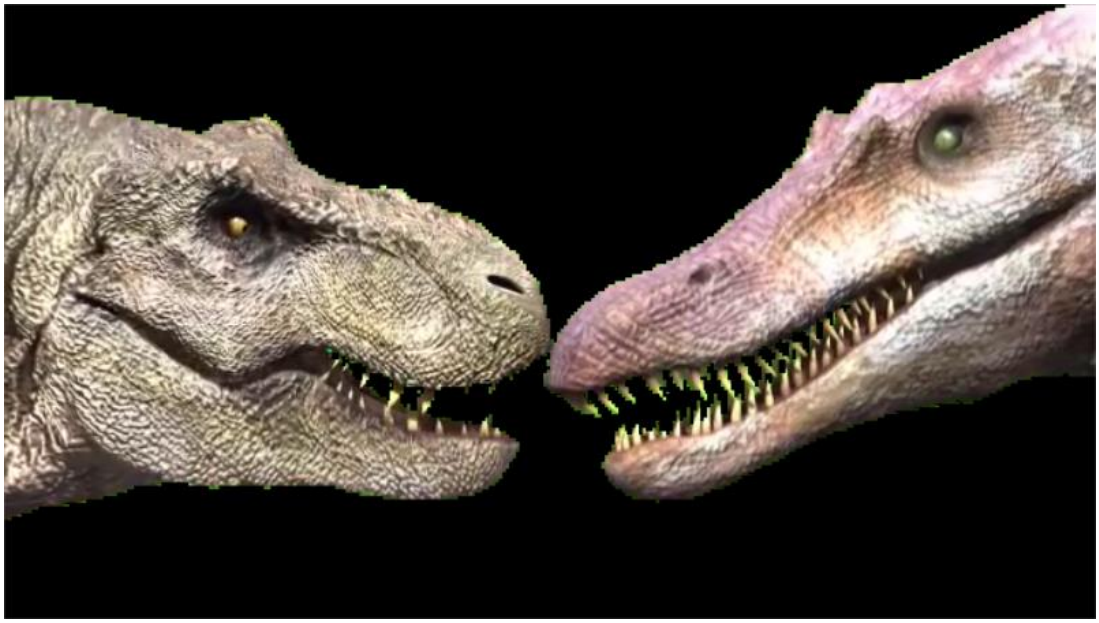
方式一：阈值掩模法

步骤：

1. **转换颜色空间。**提取原前景视频的每一帧，转换到 HSV 色彩空间。
2. **设置背景色阈值并构建掩模。**由于原前景视频的背景是绿色，查询绿色的 HSV 范围，并根据该范围构造原视频的掩模：原视频中属于该范围的置为 0 (黑色)，不属于该范围的置为 1 (白色)。

原视频某一帧的掩模

3. **原视频帧和掩模进行位运算。**掩模中数值为 1 的白色部分与原视频相与后，原视频的相同部分被原样保留。掩模中数值为 0 的黑色部分与原视频相与后，原视频的相同部分被置为黑色，从而达到了保留前景并去除绿色背景的效果。



原视频帧和掩模位运算后得到的去背景的图像

4. **去除背景中的前景区域。**为使前景覆盖背景而不是叠加在背景上，先将背景和前景的重合区域除去。具体方法是将背景画面减去掩模画面，得到的就是去除了前景区域的背景画面。

去除了前景区域的背景画面

5. **代数加。**将步骤 3 的输出画面、步骤 4 的输出画面和装饰用视频的画面进行代数相加，得到的就是最终的合成视频。

合成视频帧

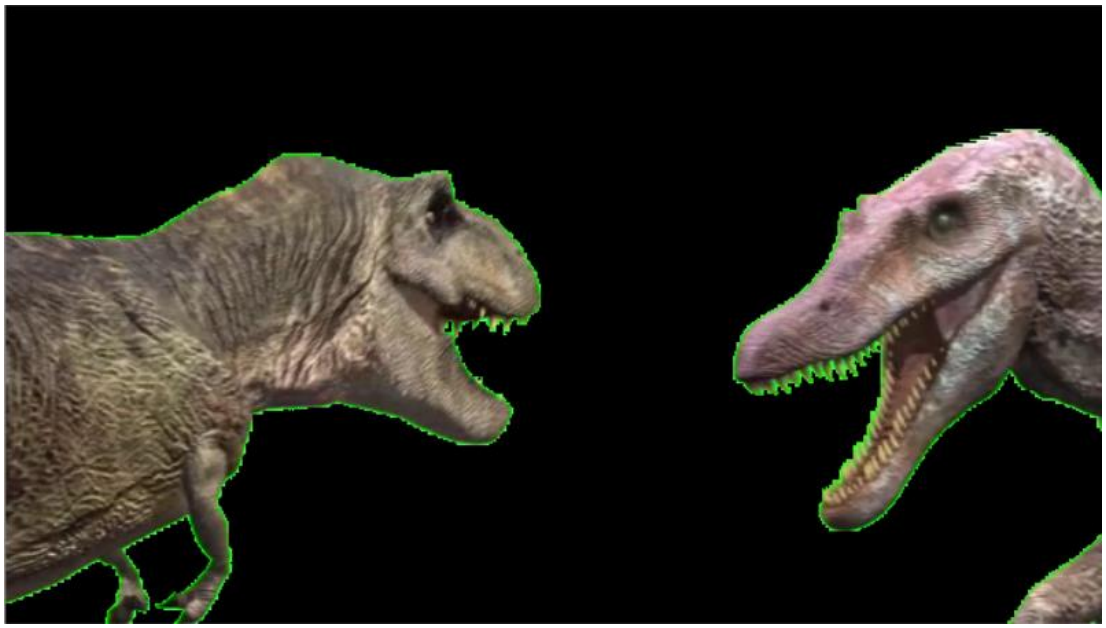
源代码：

```
import cv2
import numpy as np
# 加载视频合成素材
origin = cv2.VideoCapture('origin.mp4')
sky = cv2.VideoCapture('sky.mp4')
light = cv2.VideoCapture('light.mp4')
while origin.isOpened() & sky.isOpened() & light.isOpened():
# 获取每一帧
    ret1, origin_frame = origin.read()
    ret2, sky_frame = sky.read()
    ret3, light_frame = light.read()
# 原视频画面转换到 HSV
    hsv = cv2.cvtColor(origin_frame,cv2.COLOR_BGR2HSV)
# 设定绿色的阈值
    lower_green = np.array([50, 43, 46])
    upper_green = np.array([70, 255, 255])
# 根据阈值构建掩模
    mask = cv2.inRange(hsv,lower_green,upper_green)
# 对原图像和掩模进行位运算,去除绿色背景
    res = cv2.bitwise_and(origin_frame,origin_frame,mask=~mask)
# 为使前景覆盖背景而不是叠加在背景上，先将背景和前景的重合区域除去（置为黑色）
    dst = cv2.addWeighted(sky_frame, 1, res, -20, 0)
# 视频合成
```

```
        dst1 = cv2.addWeighted(dst, 1, res, 1, 0)
        dst2 = cv2.addWeighted(dst1, 1, light_frame, 1, 0)
    # 显示视频每一帧（包括掩模和原视频）
        cv2.imshow('frame',dst)
        cv2.imshow('res',res)
        cv2.imshow('mask',~mask)
        cv2.imshow('output',dst2)
    # ESC 键退出
        if cv2.waitKey(10)&0xFF == 27:
            break
    # 关闭窗口
    cv2.destroyAllWindows()
```

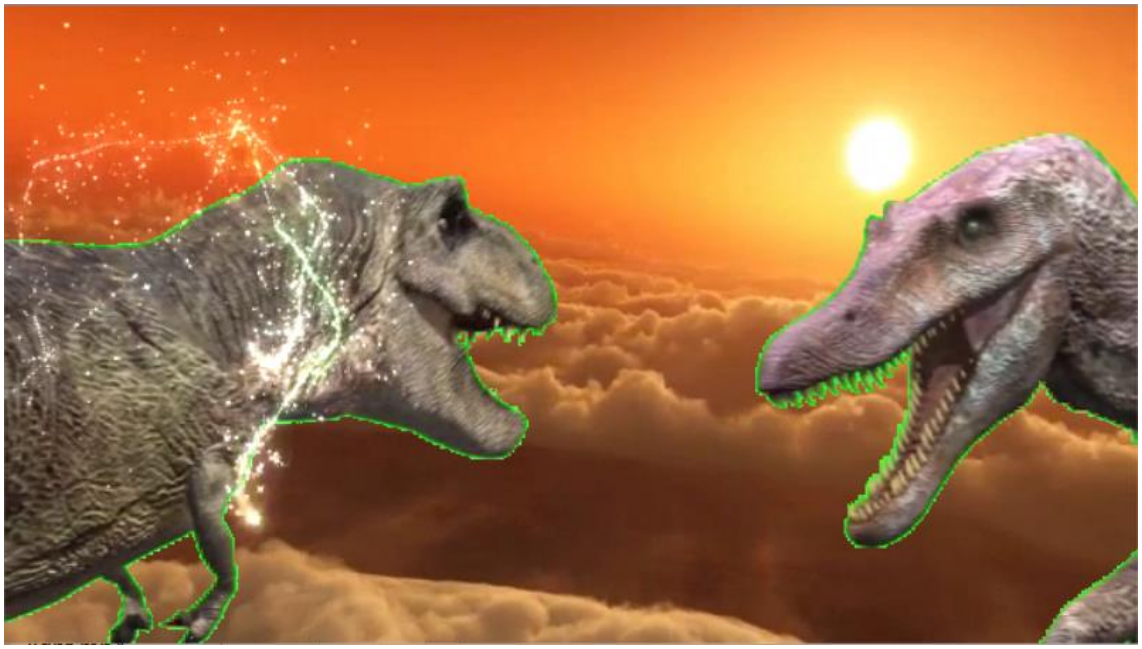
方式二：像素比较法

思想：对于视频的每一帧，对每个像素和背景颜色比较，若相同则置为 0，不同则保留。



去背景

去前景



合成效果

源代码：

```
import cv2
origin = cv2.VideoCapture('origin.mp4')
```

```
sky = cv2.VideoCapture('sky.mp4')
light = cv2.VideoCapture('light.mp4')
# 获取背景像素 G 值
ret,first_frame = origin.read()
g = first_frame[1, 1, 1]
while origin.isOpened() & sky.isOpened() & light.isOpened():
    # 获取每一帧
    ret1, origin_frame = origin.read()
    ret2, sky_frame = sky.read()
    ret3, light_frame = light.read()
    # 遍历原视频的每一帧中的每个像素，将其中与背景像素相等的像素置为黑色
    for i in range(360):
        for j in range(640):
            if  (abs(origin_frame[i, j, 1] - g) < 5):
                origin_frame[i, j, 0] = 0
                origin_frame[i, j, 1] = 0
                origin_frame[i, j, 2] = 0
    # 为使前景覆盖背景而不是叠加在背景上，先将背景和前景的重合区域除去（置为黑色）
    dst = cv2.addWeighted(sky_frame, 1, origin_frame, -100, 0)
    # 视频合成
    dst1 = cv2.addWeighted(dst, 1, origin_frame, 1, 0)
    dst2 = cv2.addWeighted(dst1, 1, light_frame, 1, 0)
    # 输出合成后的帧
    cv2.imshow('frame', origin_frame)
    cv2.imshow('dst', dst)
    cv2.imshow('output', dst2)
    # ESC 键退出
    if cv2.waitKey(1) & 0xFF == 27:
        break
# 关闭窗口
cv2.destroyAllWindows()
```

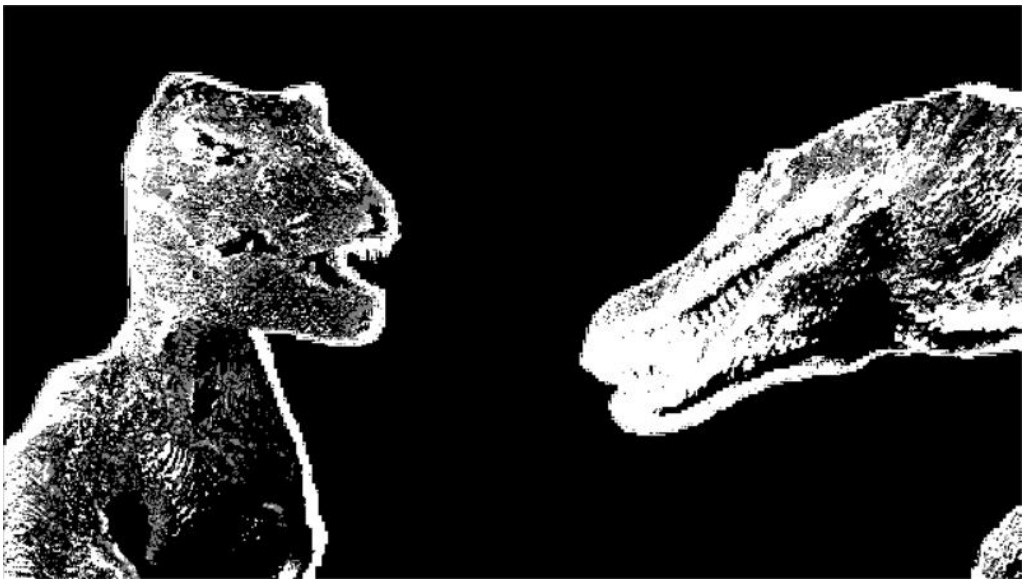
方式三：KNN 算法构建掩模

邻近算法，或者说 K 最近邻(kNN，k-NearestNeighbor)分类算法。所谓 K 最近邻，就是 k 个最近的邻居的意思，说的是每个样本都可以用它最接近的 k 个邻居来代表。

kNN 算法的核心思想是如果一个样本在特征空间中的 k 个最相邻的样本中的大多数属于某一个类别，则该样本也属于这个类别，并具有这个类别上样本的特

性。该方法在确定分类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。 kNN 方法在类别决策时，只与极少量的相邻样本有关。由于 kNN 方法主要靠周围有限的邻近的样本，而不是靠判别类域的方法来确定所属类别的，因此对于类域的交叉或重叠较多的待分样本集来说，kNN 方法较其他方法更为适合。

KNN 算法得到的掩模：



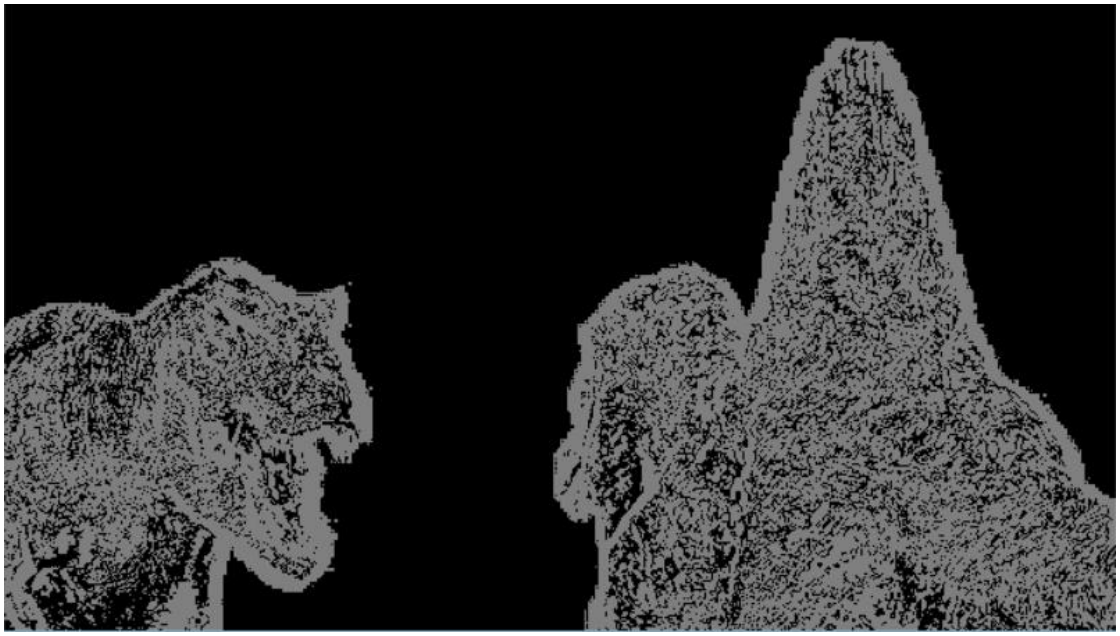
膨胀处理后的掩模：

之后的步骤与方式一的步骤 3 开始相同，最后得到的合成视频截图：

方式四：高斯混合模型背景建模法构建掩模

MOG2 算法，也是高斯混合模型分离算法，是 MOG 的改进算法。该算法的一个重要特征是： 它为每个像素选择适当数量的高斯分布，它可以更好地适应不同场景的照明变化等。

MOG2 算法得到的掩模：



膨胀处理后的掩模：

之后的步骤与方式一的步骤 3 开始相同，最后得到的合成视频截图：

综上，本案例中，方式一的处理效果最好。