重 庆 大 学

学 生 实 验 报 告

实验课程名称 多媒体技术

开课实验室 DS1501

学 院 软件学院 年级 2025 专业班 软件工程 1 班

学 生 姓 名 邓永思 学 号 20231265

开 课 时 间 2024 至 2025 学年第 二 学期

总 成 绩	
教师签名	桑军

软件学院制

《多媒体技术》实验报告

开课实验室：DS1501 **2025 年 10 月 18 日**

学院	软件学院	年级、专业、班	2323 级软件工程 1 班	姓名	邓永思	成绩	
课程名称	多媒体技术	实验项目名　称	图像变换动画	指导教师	桑军		
教师评语	教师签名：桑军 年　月　日						

一、实验目的

通过两幅图像的变换，实现动画效果。

二、实验原理

变形动画将一幅图像动态变换为另一幅大小相同、结构相似的图像。其中需要在两幅图像的主体结构中标注对应的变换点，在变换过程中按照对应点进行形状结构的变换。

如果不考虑图像形状结构的变换，则可简单地实现将一幅图像动态变换为另一幅大小相同的图像。其对应变换点就以对应的像素点确定。本实验即按照该方式实现。

对于大小不一样的图像需要更复杂的处理方式。需要一定的插值算法生成图像变换所产生的额外的像素值。最终效果因算法的不同而不同。本实验不考虑该情形。

变换结果先以图像文件方式存储，然后可采用两种方式实现动画展示：使用构造 GIF 动画的软件将变换过程的图像连接成 GIF 动画；自行编制软件按照一定的速度打开并展示变换过程的图像，形成动画展示效果。

理论上对于不同格式的图像文件均可以实现图像变换动画。这里为了简化操作，统一使用 BMP 格式的图像文件。

对于 24 位真彩色 BMP 图像构造图像变换动画，主要就是对于图像数据阵列中的各对应像素点的 RGB 值进行插值变换，实现将一幅图像中的像素点的 RGB 值变换为另一幅图像中对应像素点的 RGB 值。

对于使用调色板的 BMP 索引图像，生成图像变换动画的过程稍稍复杂。主要采取以下方式：保持调色板不变，对于像素点进行颜色变换。将一个像素点变换为其对应的像素点时，根据该像素点索引值所指向的调色板表项的 RGB 和其对应的像素点索引值所指向的调色板表项的 RGB，计算

其变换图像的 RGB 值，然后在调色板中查找与变换过程图像的 RGB 值最接近的表项，将其对应的索引值作为变换过程图像的像素索引值。该方式不改变原始调色板，但需要将变换过程 RGB 值映射为调色板中最接近的表项；

三、实验内容

打开两幅大小相同的 BMP 图像，分别指定为起始帧、终止帧，指定变换帧数，实现将图像从起始帧逐步变换到终止帧，将一副图像动态地变换为另一幅图像。

。

四、实验工具

Python

五、实验步骤

1.找到分辨率相同的两张 BMP 图像作为测试图像

2.编写代码（利用 python 库支持的 GUI）

```
import tkinter as tk
from tkinter import filedialog, messagebox, ttk
from PIL import Image, ImageTk
import os
import numpy as np
import math


class BMPAnimationGenerator:
    def __init__(self, root):
        self.root = root
        self.root.title("BMP 图像动态转换工具")
        self.root.geometry("800x700")

        # 图像数据
        self.start_image = None
        self.end_image = None
        self.start_image_path = ""
        self.end_image_path = ""

        # 动画参数
        self.frame_count = 20
        self.duration = 2000  # 毫秒

        self.setup_ui()

    def setup_ui(self):
        # 主框架
        main_frame = ttk.Frame(self.root, padding="10")
        main_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        # 标题
        title_label = ttk.Label(main_frame, text="BMP 图像动态转换工具",
                                font=("Arial", 16, "bold"))
        title_label.grid(row=0, column=0, columnspan=3, pady=(0, 20))

        # 图像选择区域
        select_frame = ttk.LabelFrame(main_frame, text="图像选择", padding="10")
```

```
        select_frame.grid(row=1, column=0, columnspan=3, sticky=(tk.W, tk.E), pady=(0,
10))

        # 开始图像
        ttk.Label(select_frame, text="开始图像:").grid(row=0, column=0, sticky=tk.W,
padx=(0, 10))
        self.start_label = ttk.Label(select_frame, text="未选择", foreground="gray")
        self.start_label.grid(row=0, column=1, sticky=tk.W)
        ttk.Button(select_frame, text="选择开始图像",
                   command=self.select_start_image).grid(row=0, column=2, padx=(10,
0))

        # 结束图像
        ttk.Label(select_frame, text="结束图像:").grid(row=1, column=0, sticky=tk.W,
padx=(0, 10), pady=(10, 0))
        self.end_label = ttk.Label(select_frame, text="未选择", foreground="gray")
        self.end_label.grid(row=1, column=1, sticky=tk.W, pady=(10, 0))
        ttk.Button(select_frame, text="选择结束图像",
                   command=self.select_end_image).grid(row=1, column=2, padx=(10, 0),
pady=(10, 0))

        # 参数设置区域
        param_frame = ttk.LabelFrame(main_frame, text="动画参数", padding="10")
        param_frame.grid(row=2, column=0, columnspan=3, sticky=(tk.W, tk.E), pady=(0,
10))

        # 帧数设置
        ttk.Label(param_frame, text="中间帧数量:").grid(row=0, column=0, sticky=tk.W)
        self.frame_var = tk.StringVar(value=str(self.frame_count))
        frame_spinbox = ttk.Spinbox(param_frame, from_=2, to=100,
textvariable=self.frame_var, width=10)
        frame_spinbox.grid(row=0, column=1, sticky=tk.W, padx=(10, 0))

        # 动画时长
        ttk.Label(param_frame, text="动画时长(ms):").grid(row=0, column=2,
sticky=tk.W, padx=(20, 0))
        self.duration_var = tk.StringVar(value=str(self.duration))
        duration_spinbox = ttk.Spinbox(param_frame, from_=100, to=5000,
textvariable=self.duration_var, width=10)
        duration_spinbox.grid(row=0, column=3, sticky=tk.W, padx=(10, 0))

        # 预览区域
        preview_frame = ttk.LabelFrame(main_frame, text="图像预览", padding="10")
        preview_frame.grid(row=3, column=0, columnspan=3, sticky=(tk.W, tk.E),
pady=(0, 10))
```

```
        # 创建画布用于显示图像
        self.canvas_start = tk.Canvas(preview_frame, width=200, height=200,
bg="white")
        self.canvas_start.grid(row=0, column=0, padx=(0, 10))
        ttk.Label(preview_frame, text="开始图像").grid(row=1, column=0)

        self.canvas_end = tk.Canvas(preview_frame, width=200, height=200, bg="white")
        self.canvas_end.grid(row=0, column=1, padx=(0, 10))
        ttk.Label(preview_frame, text="结束图像").grid(row=1, column=1)

        # 控制按钮区域
        button_frame = ttk.Frame(main_frame)
        button_frame.grid(row=4, column=0, columnspan=3, pady=20)

        ttk.Button(button_frame, text="生成中间帧",
                   command=self.generate_animation).pack(side=tk.LEFT, padx=(0, 10))
        ttk.Button(button_frame, text="预览动画",
                   command=self.preview_animation).pack(side=tk.LEFT, padx=(0, 10))
        ttk.Button(button_frame, text="保存中间帧",
                   command=self.save_frames).pack(side=tk.LEFT)

        # 进度条
        self.progress = ttk.Progressbar(main_frame, mode='determinate')
        self.progress.grid(row=5, column=0, columnspan=3, sticky=(tk.W, tk.E),
pady=(10, 0))

        # 状态栏
        self.status_var = tk.StringVar(value="就绪")
        status_label = ttk.Label(main_frame, textvariable=self.status_var,
relief=tk.SUNKEN)
        status_label.grid(row=6, column=0, columnspan=3, sticky=(tk.W, tk.E),
pady=(10, 0))

    def select_start_image(self):
        filename = filedialog.askopenfilename(
            title="选择开始图像",
            filetypes=[("BMP 文件", "*.bmp"), ("所有文件", "*.*")]
        )
        if filename:
            self.start_image_path = filename
            self.start_label.config(text=os.path.basename(filename))
            self.load_and_display_image(filename, self.canvas_start, "start")

    def select_end_image(self):
        filename = filedialog.askopenfilename(
            title="选择结束图像",
```

```
            filetypes=[("BMP 文件", "*.bmp"), ("所有文件", "*.*")]
        )
        if filename:
            self.end_image_path = filename
            self.end_label.config(text=os.path.basename(filename))
            self.load_and_display_image(filename, self.canvas_end, "end")

    def load_and_display_image(self, filename, canvas, image_type):
        try:
            image = Image.open(filename)
            # 调整图像大小以适应画布
            image.thumbnail((180, 180), Image.Resampling.LANCZOS)
            photo = ImageTk.PhotoImage(image)

            if image_type == "start":
                self.start_image = Image.open(filename)
                self.start_preview = photo
            else:
                self.end_image = Image.open(filename)
                self.end_preview = photo

            # 清除画布并显示新图像
            canvas.delete("all")
            canvas.create_image(100, 100, image=photo)
            canvas.image = photo  # 保持引用

            self.status_var.set(f"已加载: {os.path.basename(filename)}")

        except Exception as e:
            messagebox.showerror("错误", f"加载图像失败: {str(e)}")

    def validate_images(self):
        if self.start_image is None or self.end_image is None:
            messagebox.showwarning("警告", "请先选择开始图像和结束图像")
            return False

        if self.start_image.size != self.end_image.size:
            messagebox.showerror("错误", "开始图像和结束图像的分辨率必须相同")
            return False

        try:
            self.frame_count = int(self.frame_var.get())
            self.duration = int(self.duration_var.get())
            if self.frame_count < 2 or self.duration < 100:
                messagebox.showwarning("警告", "帧数必须大于等于 2，动画时长必须大于
等于 100ms")
```

```
                return False
        except ValueError:
            messagebox.showerror("错误", "请输入有效的数字")
            return False

        return True

    def generate_intermediate_frames(self):
        """生成中间帧"""
        if not self.validate_images():
            return None

        self.status_var.set("正在生成中间帧...")
        self.progress['value'] = 0
        self.root.update_idletasks()

        try:
            # 获取图像数据
            start_array = np.array(self.start_image)
            end_array = np.array(self.end_image)

            frames = []
            total_frames = self.frame_count + 2  # 包括开始和结束帧

            for i in range(total_frames):
                # 计算插值比例 (0 到 1)
                t = i / (total_frames - 1)

                # 生成中间帧
                if self.start_image.mode == 'P':
                    # 索引图像处理
                    intermediate_frame  =  self.generate_indexed_frame(start_array,
end_array, t)
                else:
                    # 24 位真彩色图像处理
                    intermediate_frame = self.generate_truecolor_frame(start_array,
end_array, t)

                frames.append(intermediate_frame)

                # 更新进度
                progress_value = (i + 1) / total_frames * 100
                self.progress['value'] = progress_value
                self.root.update_idletasks()

            self.status_var.set("中间帧生成完成")
```

```
            return frames

        except Exception as e:
            messagebox.showerror("错误", f"生成中间帧失败: {str(e)}")
            return None

    def generate_truecolor_frame(self, start_array, end_array, t):
        """生成24位真彩色中间帧"""
        # 线性插值
        intermediate_array = (1 - t) * start_array.astype(float) + t * end_array.astype(float)
        intermediate_array = np.clip(intermediate_array, 0, 255).astype(np.uint8)

        return Image.fromarray(intermediate_array, mode=self.start_image.mode)

    def generate_indexed_frame(self, start_array, end_array, t):
        """生成索引图像中间帧"""
        # 获取调色板
        start_palette = self.start_image.getpalette()
        end_palette = self.end_image.getpalette()

        if start_palette is None or end_palette is None:
            raise ValueError("无法获取调色板信息")

        # 将调色板转换为RGB列表
        start_rgb_palette = []
        end_rgb_palette = []

        for i in range(0, len(start_palette), 3):
            start_rgb_palette.append((start_palette[i], start_palette[i + 1], start_palette[i + 2]))
            end_rgb_palette.append((end_palette[i], end_palette[i + 1], end_palette[i + 2]))

        # 创建中间帧数组
        height, width = start_array.shape
        intermediate_array = np.zeros((height, width), dtype=np.uint8)

        # 对每个像素进行处理
        for y in range(height):
            for x in range(width):
                start_idx = start_array[y, x]
                end_idx = end_array[y, x]

                # 获取对应的RGB值
                start_rgb = start_rgb_palette[start_idx]
```

```
                end_rgb = end_rgb_palette[end_idx]

                # 计算中间 RGB 值
                intermediate_rgb = (
                    int((1 - t) * start_rgb[0] + t * end_rgb[0]),
                    int((1 - t) * start_rgb[1] + t * end_rgb[1]),
                    int((1 - t) * start_rgb[2] + t * end_rgb[2])
                )

                # 在调色板中查找最接近的颜色
                closest_idx       =       self.find_closest_color(intermediate_rgb,
start_rgb_palette)
                intermediate_array[y, x] = closest_idx

        # 创建新图像，使用开始图像的调色板
        intermediate_image = Image.fromarray(intermediate_array, mode='P')
        intermediate_image.putpalette(start_palette)

        return intermediate_image

    def find_closest_color(self, target_rgb, palette):
        """在调色板中查找与目标 RGB 最接近的颜色索引"""
        min_distance = float('inf')
        closest_index = 0

        for i, color in enumerate(palette):
            # 计算欧几里得距离
            distance = math.sqrt(
                (target_rgb[0] - color[0]) ** 2 +
                (target_rgb[1] - color[1]) ** 2 +
                (target_rgb[2] - color[2]) ** 2
            )

            if distance < min_distance:
                min_distance = distance
                closest_index = i

        return closest_index

    def generate_animation(self):
        """生成中间帧"""
        frames = self.generate_intermediate_frames()
        if frames:
            self.animation_frames = frames
            messagebox.showinfo("成功", f"已生成 {len(frames)} 帧图像")
```

```
    def preview_animation(self):
        """预览动画"""
        if not hasattr(self, 'animation_frames') or not self.animation_frames:
            messagebox.showwarning("警告", "请先生成中间帧")
            return

        # 创建预览窗口
        preview_window = tk.Toplevel(self.root)
        preview_window.title("动画预览")
        preview_window.geometry("400x450")

        canvas = tk.Canvas(preview_window, width=300, height=300, bg="white")
        canvas.pack(pady=10)

        frame_index = 0
        photo_references = []  # 保持对 PhotoImage 的引用

        def update_frame():
            nonlocal frame_index
            if frame_index < len(self.animation_frames):
                frame = self.animation_frames[frame_index]
                frame.thumbnail((280, 280), Image.Resampling.LANCZOS)
                photo = ImageTk.PhotoImage(frame)
                photo_references.append(photo)  # 保持引用

                canvas.delete("all")
                canvas.create_image(150, 150, image=photo)
                canvas.image = photo

                frame_index = (frame_index + 1) % len(self.animation_frames)
                preview_window.after(self.duration  //  len(self.animation_frames),
update_frame)

        # 控制按钮
        control_frame = ttk.Frame(preview_window)
        control_frame.pack(pady=10)

        ttk.Button(control_frame, text="开始预览",
                   command=update_frame).pack(side=tk.LEFT, padx=5)
        ttk.Button(control_frame, text="关闭",
                   command=preview_window.destroy).pack(side=tk.LEFT, padx=5)

    def save_frames(self):
        """保存中间帧为单独的 BMP 文件"""
        if not hasattr(self, 'animation_frames') or not self.animation_frames:
            messagebox.showwarning("警告", "请先生成中间帧")
```

```
            return

        folder = filedialog.askdirectory(title="选择保存中间帧的文件夹")

        if folder:
            try:
                self.status_var.set("正在保存中间帧...")
                self.progress['value'] = 0
                self.root.update_idletasks()

                # 获取基础文件名（不含扩展名）
                start_name                                                    =
os.path.splitext(os.path.basename(self.start_image_path))[0]
                end_name = os.path.splitext(os.path.basename(self.end_image_path))[0]

                total_frames = len(self.animation_frames)

                for i, frame in enumerate(self.animation_frames):
                    # 生成文件名：基础名_帧号.bmp
                    filename = f"{start_name}_to_{end_name}_frame_{i:03d}.bmp"
                    filepath = os.path.join(folder, filename)

                    # 保存为BMP格式
                    frame.save(filepath, format="BMP")

                    # 更新进度
                    progress_value = (i + 1) / total_frames * 100
                    self.progress['value'] = progress_value
                    self.status_var.set(f"正在保存帧 {i + 1}/{total_frames}")
                    self.root.update_idletasks()

                self.status_var.set(f"中间帧保存完成，共 {total_frames} 帧")
                messagebox.showinfo("成功", f"中间帧已保存到: {folder}\n 共保存了
{total_frames} 帧图像")

            except Exception as e:
                messagebox.showerror("错误", f"保存中间帧失败: {str(e)}")


def main():
    root = tk.Tk()
    app = BMPAnimationGenerator(root)
    root.mainloop()


if __name__ == "__main__":
```

```
main()
```

六、实验结果及分析

效果图：按照提示选取关键帧，选取完毕后设置中间帧的数量和动画时长，可生成动画并预览，预览完毕后可选择保存关键帧