实验五 混合

2151094 宋正非

1. 实验目标

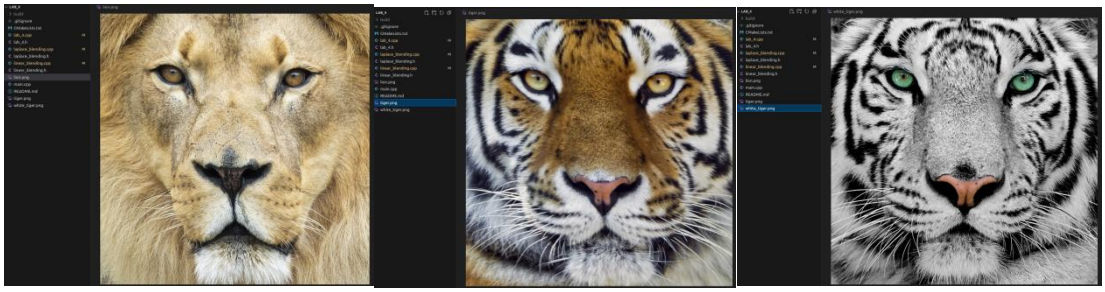
实现并测试 blending 混合算法。

2. 实验流程

1) 下载实验代码模板

在 Ubuntu 终端中使用 git 下载实验代码模板，有如下的三张照片 lion.png/tiger.png/ white_tiger.png。

```
git clone https://git.tongji.edu.cn/ipmv/examples/lab_4.git
```



2) 读取并转换图像

使用 cv::imread() 函数读取图像文件，然后将图像转换为 CV_32F 格式，并缩放图像像素，使其值介于 [0,1] 之间。

3) 创建混合权重的图像

使用渐变创建简单的掩膜，需要满足：与输入图像大小相同，左半列为黑色 (0.0)，右半列为白色 (1.0)。如下图所示。

```
void lab4()
{
 // Load images.
 // TODO: Load the images using cv::imread() and convert to 32-bit floating point images.
 // Using relative filenames such as "../tiger.png" should work.
 // Remember to rescale so that they have values in the interval [0, 1].
 // Hint: convertTo().
  // Load images.
 cv::Mat img_1 = cv::imread("../tiger.png");
 cv::Mat img_2 = cv::imread("../lion.png");
```

```
if (img_1.empty()) {
  std::cout << "Error loading image 1!" << std::endl;
  return;
}
if (img_2.empty()) {
  std::cout << "Error loading image 2!" << std::endl;
  return;
}
if (img_1.size() != img_2.size()) {
  cv::resize(img_2, img_2, img_1.size());  // Resize img_2 to match img_1 size
}
// Convert to 32-bit floating point images and rescale to [0, 1].
img  1.convertTo(img  1, CV  32F, 1.0 / 255);
img_2.convertTo(img_2, CV_32F, 1.0 / 255);

showResult("Lab 4 - Image 1 original", img_1);
showResult("Lab 4 - Image 2 original", img  2);

// Construct weights.
// TODO: Create a 32-bit, 3 channel floating point weight image.
// The first half of the columns should be black (0.0f).
// The last half of the columns should be white (1.0f).
// Then make a ramp between these two halves.
// Hint: Use cv::blur() to make the ramp.
// Construct weights.
const int ramp_width = 200;//todo
cv::Mat weights = cv::Mat::zeros(img  1.size(), CV  32FC3);
int cols = weights.cols;
int ramp_start = cols / 2 - ramp_width / 2;
int ramp_end = ramp_start + ramp_width;

for (int i = 0; i < cols; ++i)
{
  float alpha = 0.0f;
  if (i < ramp_start)
    alpha = 0.0f;
  else if (i >= ramp  end)
    alpha = 1.0f;
  else
    alpha = static_cast<float>(i - ramp_start) / ramp_width;

  weights.col(i).setTo(cv::Scalar(alpha, alpha, alpha));
}

// Apply blur to smooth out the transition.
cv::blur(weights, weights, cv::Size(15, 15));  // Use a larger blur for smoother transition

showResult("Lab 4 - Weights", weights);

// Perform blending.
// TODO: Finish linear  blending.cpp.
cv::Mat lin_blend = linearBlending(img_1, img_2, weights);
showResult("Lab 4 - Linear blend", lin_blend);

// TODO: Finish laplace_blending.cpp.
cv::Mat lap  blend = laplaceBlending(img  1, img  2, weights);
showResult("Lab 4 - Laplace blend", lap_blend);
```

```
 // Show all results.
 // Press a key when finished.
 // If you close the windows, the program won't stop!
 cv::waitKey();
}
```

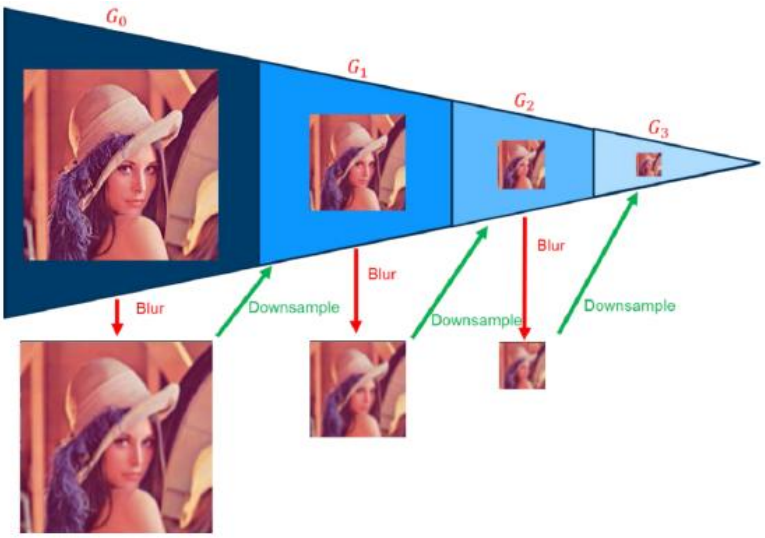
4) 简单线性混合

使用刚刚创建的权重掩膜实现两幅图像的简单线性混合。

```
cv::Mat linearBlending(const cv::Mat& img_1, const cv::Mat& img_2, const cv::Mat& weights)
{
 // TODO: Blend the two images according to the weights: result = weights*img_1 + (1-weights)*img_2
 // No need to loop through all pixels!
 // Hint: https://docs.opencv.org/3.3.1/d1/d10/classcv_1_1MatExpr.html

 // Ensure the input images have the same size and type
 CV_Assert(img_1.size() == img_2.size() && img_1.type() == img_2.type());

 // Ensure weights are the same size as the images
 CV_Assert(weights.size() == img_1.size() && weights.type() == img_1.type());

 // Perform the linear blending: result = weights * img_1 + (1 - weights) * img_2
 cv::Mat result = weights.mul(img_1) + (cv::Scalar(1.0, 1.0, 1.0) - weights).mul(img_2);

 return result;
}
```

5) 拉普拉斯混合

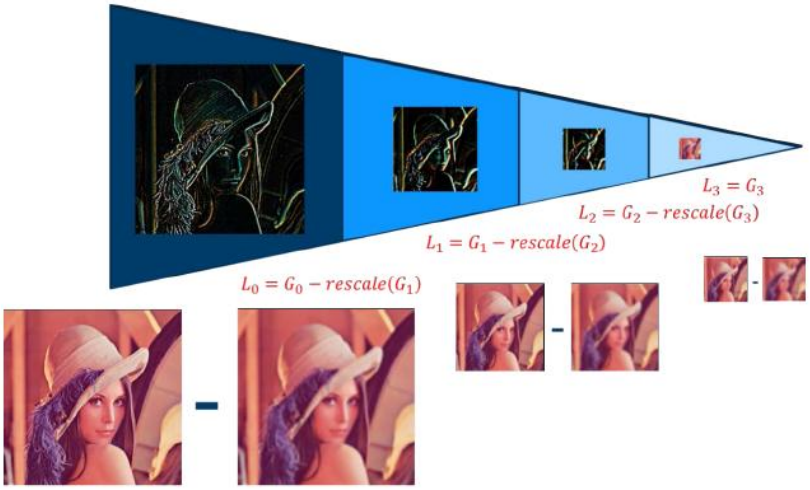
构建高斯金字塔。




```
std::vector<cv::Mat, std::allocator<cv::Mat>> constructGaussianPyramid(const cv::Mat& img)
{
 // Construct the pyramid starting with the original image.
 std::vector<cv::Mat> pyr;
 pyr.push_back(img.clone());

 // Add new downscaled images to the pyramid
```

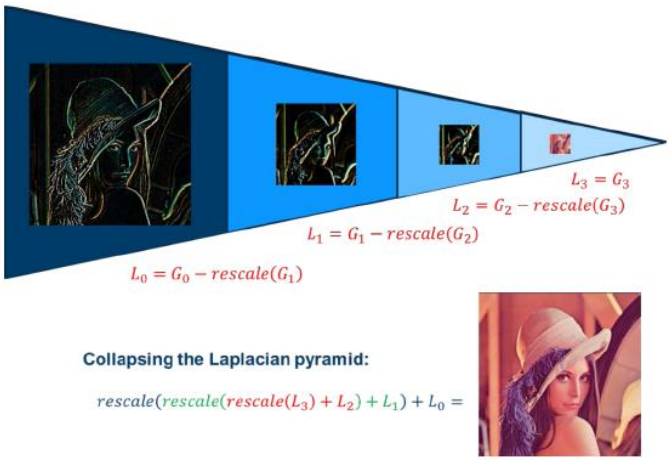
```
 // until image width is <= 16 pixels
 while(pyr.back().cols > 16)
 {
  // TODO: Add the next level in the pyramid.
  // Hint cv::pyrDown(...)
  cv::Mat downscaled;
  cv::pyrDown(pyr.back(), downscaled);
  pyr.push  back(downscaled);
 }

 return pyr;
}
```

构建拉普拉斯金字塔。



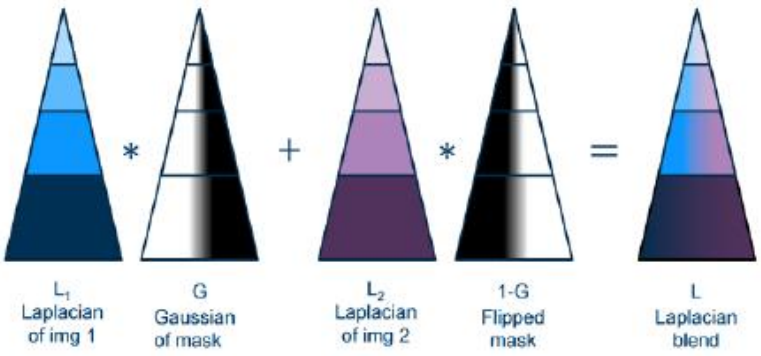

```
std::vector<cv::Mat> constructLaplacianPyramid(const cv::Mat& img)
{
 // TODO: Use constructGaussianPyramid() to construct a laplacian pyramid.
 // Hint: cv::pyrUp(...)
 // First, create the Gaussian pyramid
 std::vector<cv::Mat> gaussian_pyr = constructGaussianPyramid(img);

 // Construct the Laplacian pyramid
 std::vector<cv::Mat> laplacian  pyr;

 // For each level in the Gaussian pyramid, subtract the upscaled next level
 for (size  t i = 0; i < gaussian  pyr.size() - 1; ++i)
 {
  cv::Mat upscaled;
  cv::pyrUp(gaussian_pyr[i + 1], upscaled, gaussian_pyr[i].size());
  cv::Mat laplacian = gaussian_pyr[i] - upscaled;
  laplacian_pyr.push_back(laplacian);
 }

 // The last level of the pyramid is just the last image in the Gaussian pyramid
 laplacian_pyr.push_back(gaussian_pyr.back());

 return laplacian  pyr;
}
```

通过折叠拉普拉斯金字塔重建图像。




```
cv::Mat collapsePyramid(const std::vector<cv::Mat>& pyr)
{
 // TODO: Collapse the pyramid.
 cv::Mat result = pyr.back();

 // Collapse the pyramid by upscaling each level and adding it to the result
 for (int i = pyr.size() - 2; i >= 0; --i)
 {
  cv::Mat upscaled;
  cv::pyrUp(result, upscaled, pyr[i].size());
  result = upscaled + pyr[i];
 }

 return result;
}
```

实现拉普拉斯混合。

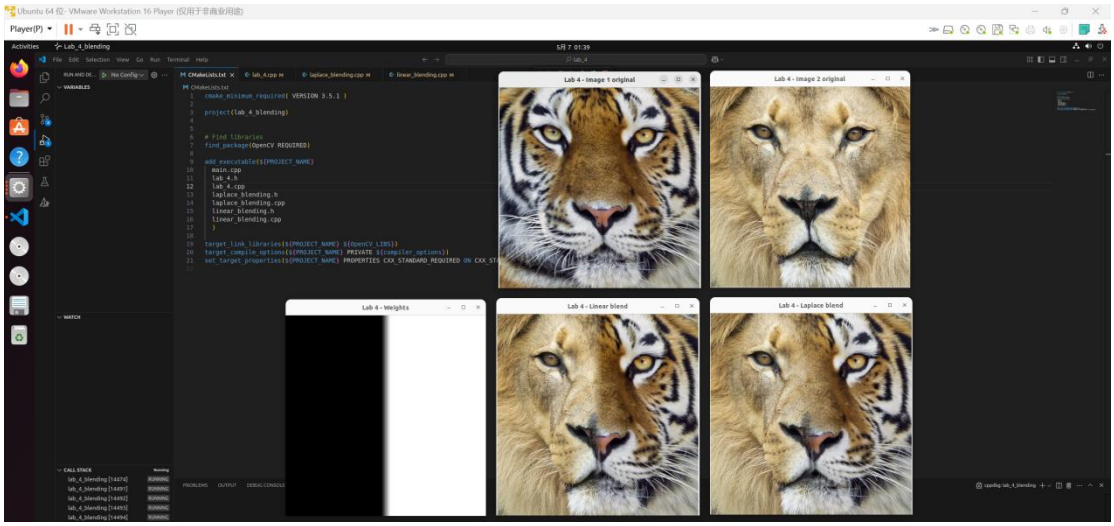



```
cv::Mat laplaceBlending(const cv::Mat& img_1, const cv::Mat& img_2, const cv::Mat& weights)
{
 // Construct a gaussian pyramid of the weight image.
 // TODO: Finish constructGaussianPyramid().
 std::vector<cv::Mat> weights_pyr = constructGaussianPyramid(weights);

 // Construct a laplacian pyramid of each of the images.
 // TODO: Finish constructLaplacianPyramid().
 std::vector<cv::Mat> img_1_pyr = constructLaplacianPyramid(img_1);
 std::vector<cv::Mat> img_2_pyr = constructLaplacianPyramid(img_2);
```

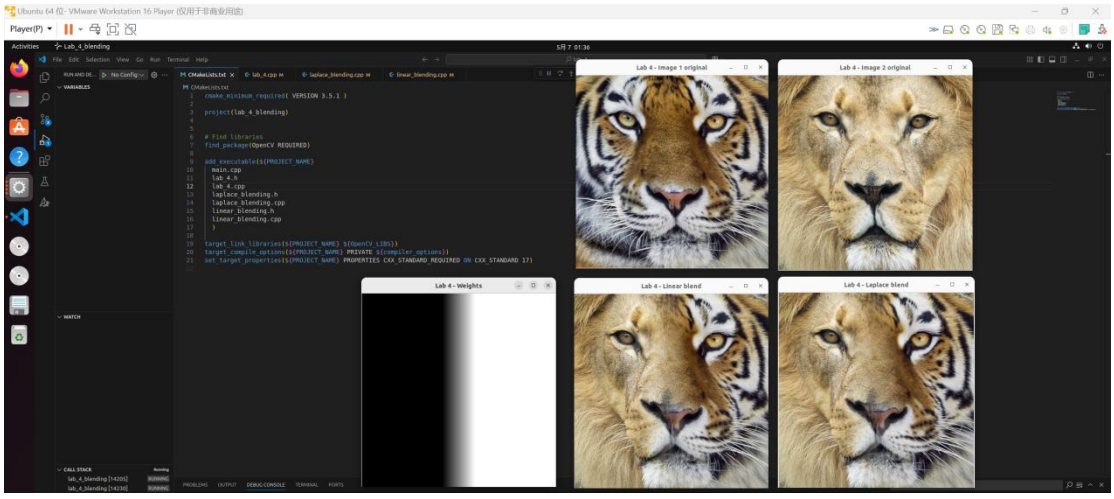
```
// Blend the laplacian pyramids according to the corresponding weight pyramid.
std::vector<cv::Mat> blend_pyr(img_1_pyr.size());
for (size_t i = 0; i < img_1_pyr.size(); ++i)
{
  // TODO: Blend the images using linearBlending() on each pyramid level.
  blend_pyr[i] = linearBlending(img_1_pyr[i], img_2_pyr[i], weights_pyr[i]);
}

// Collapse the blended laplacian pyramid.
// TODO: Finish collapsePyramid().
return collapsePyramid(blend_pyr);
}
```

6) 实现结果

ramp_width = 50

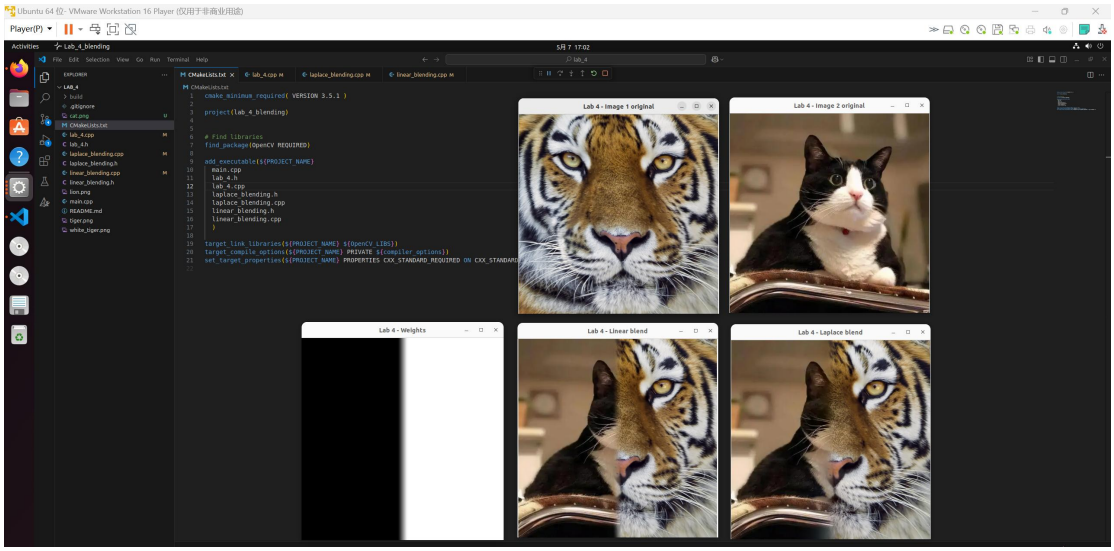


ramp_width = 200



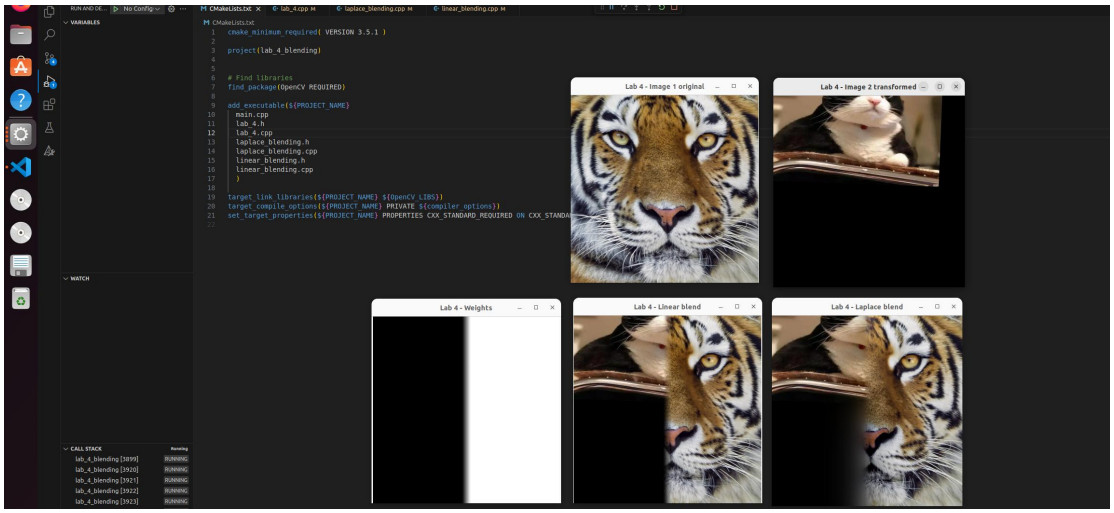
3. 额外任务

从网上下载图像，混合得到图像。



对 img2 采用如下代码进行仿射变换。

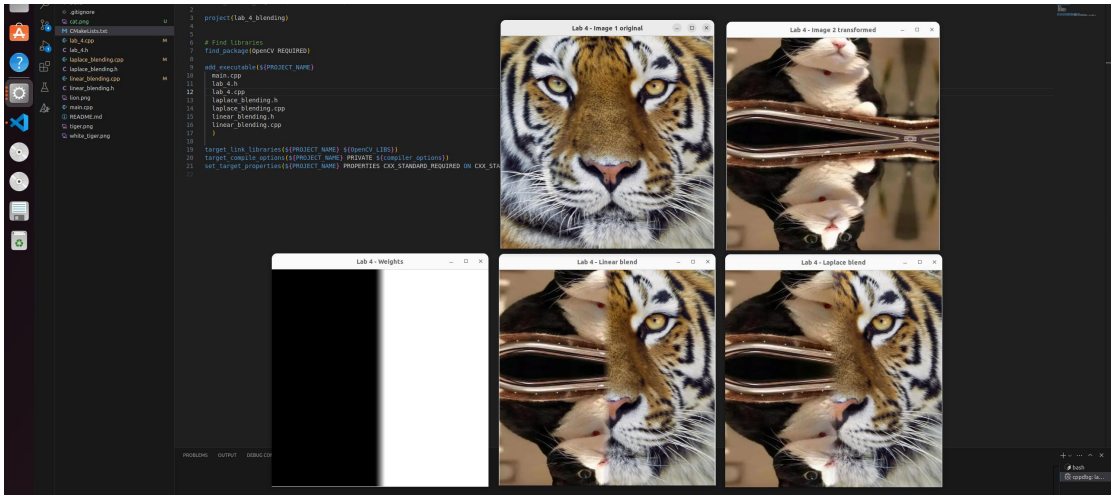
```
cv::Point2f pts_1[] = {{321, 200}, {647, 200}, {476, 509}};
cv::Point2f pts_2[] = {{441, 726}, {780, 711}, {615, 1142}};
cv::Mat trans_mat = cv::getAffineTransform(pts_2, pts_1);
cv::warpAffine(img_2, img_2, trans_mat, img_1.size());
```



在 cv::warpAffine() 中使用默认参数 borderMode=cv::BORDER_CONSTANT，仿射变换后的图像某些区域超出了原图的边界，OpenCV 默认会将超出边界的像素填充为黑色。如上图所示。

```
 cv::Point2f pts_1[] = {{321, 200}, {647, 200}, {476, 509}};
 cv::Point2f pts_2[] = {{441, 726}, {780, 711}, {615, 1142}};
 cv::Mat trans  mat = cv::getAffineTransform(pts  2, pts  1);
 cv::warpAffine(img  2, img  2, trans  mat, img  1.size(), cv::INTER  LINEAR,
cv::BORDER_REFLECT);
```

而在 cv::warpAffine() 中显式设置 cv::BORDER_REFLECT，就会看到类似翻折的效果，即边缘像素以镜像方式延伸。如下图所示。



使用圆形掩膜。

```
// Construct circular mask weights.
 // The mask will be a circle where the center is black (0.0f) and the edges are white (1.0f).
 const int radius = img_1.cols / 2;  // Using the width/height of the image to determine the radius
 cv::Mat weights = cv::Mat::zeros(img_1.size(), CV_32FC3);
 cv::Point center(weights.cols / 2, weights.rows / 2);  // Center of the image

 for (int y = 0; y < weights.rows; ++y) {
  for (int x = 0; x < weights.cols; ++x) {
    float dist = cv::norm(cv::Point(x, y) - center);  // Calculate the distance from the center
    float alpha = std::min(1.0f, dist / radius);  // Gradual transition from center (0) to edges (1)
    weights.at<cv::Vec3f>(y, x) = cv::Vec3f(alpha, alpha, alpha);  // Set the same alpha for all
channels
   }
 }
```