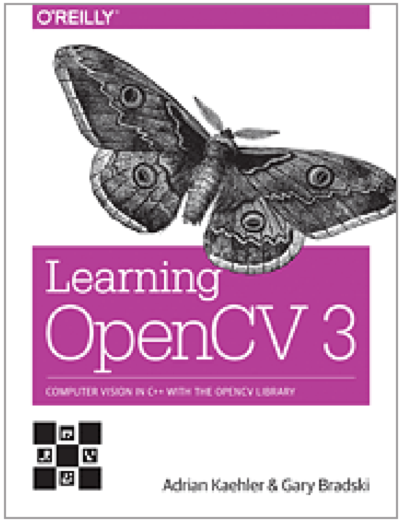
Lab 2 – Image blending with OpenCV

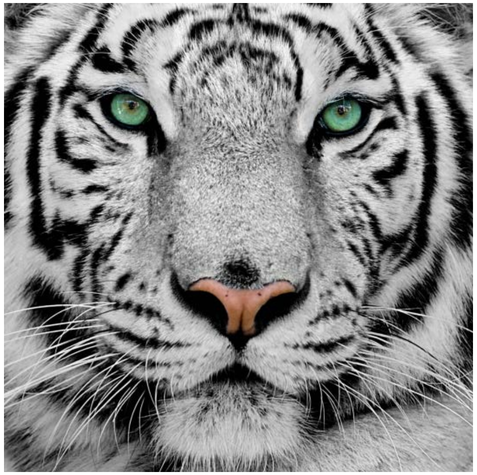
01.02.2018

OpenCV resources

- Documentation
  - http://docs.opencv.org/3.3.1/
- Tutorials
  - https://docs.opencv.org/3.3.1/d9/df8/tutorial_root.html
- Learning OpenCV 3, 1st Edition
  - Gary Bradski, Adrian Kaehler


O'REILLY
Learning
OpenCV 3
COMPUTER VISION IN C++ WITH THE OPENCV LIBRARY
Adrian Kaehler & Gary Bradski

Laplace blending



Step 1: Read and convert the images

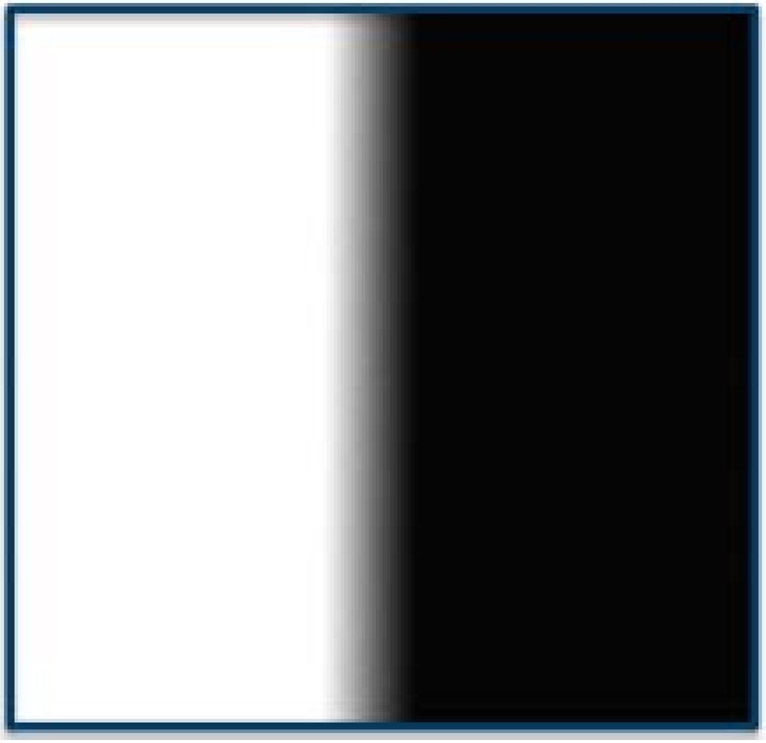
- Open `lab_2_blending`

- Read two images
  - `cv::imread(...)`

- Convert the images to `CV_32F`
  - Scale the image pixels so that they get values in the interval [0, 1]

Step 2: Create an image of blend weights

- Create simple mask with ramp
  - Same size as the input images
  - Left half of the columns are black (0.0)
  - Right half of the columns are white (1.0)
  - How can we make the ramp?



Step 3: Simple linear blending

- Implement simple blending of two images using the 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
cv::Mat linearBlending(const cv::Mat& img_1, const cv::Mat& img_2, const cv::Mat& weights)
{
  return cv::Mat();
}
```

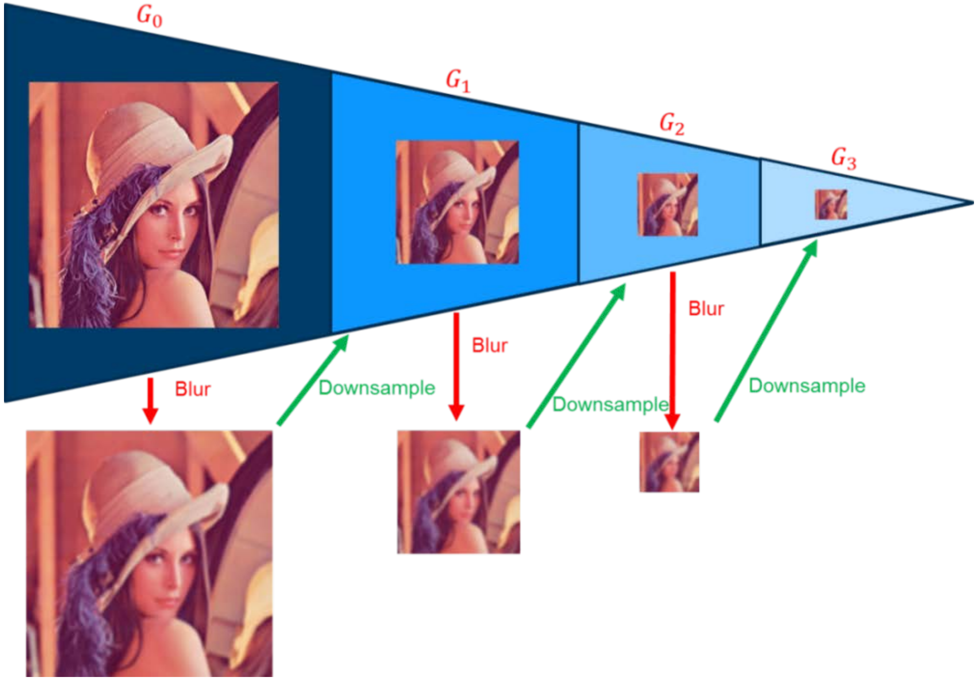
- See https://docs.opencv.org/3.3.1/d1/d10/classcv_1_1MatExpr.html

- Run the code and look at the results
  - Try changing ramp size

Step 4: Laplace blending

- Construct a gaussian pyramid

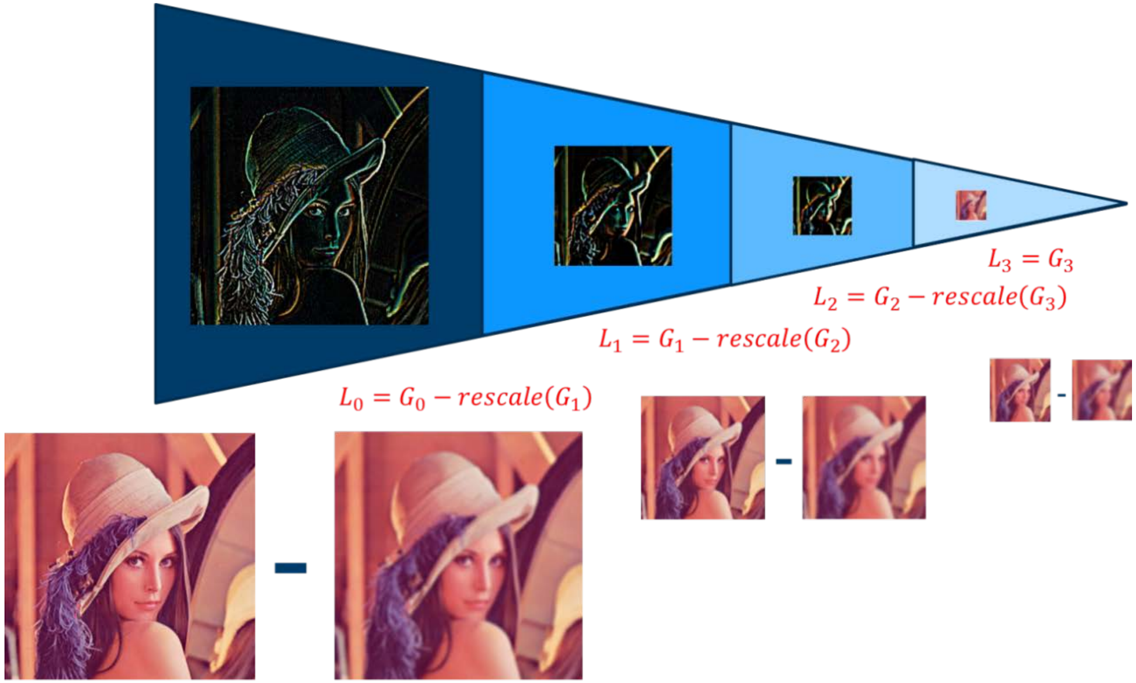
```
std::vector<cv::Mat, std::allocator<cv::Mat>> constructGaussianPyramid(const cv::Mat& img)
{
  // Construct the pyramid starting with the original image.
  std::vector<cv::Mat> pyr;
  pyr.push_back(img.clone());

  // 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(...)

    break; // TODO: Remove this break!
  }

  return pyr;
}
```



Step 4: Laplace blending

- Construct a laplacian pyramid

```
std::vector<cv::Mat> constructLaplacianPyramid(const cv::Mat& img)
{
  // TODO: Use constructGaussianPyramid() to construct a laplacian pyramid.
  // Hint: cv::pyrUp(...)
  std::vector<cv::Mat> pyr;
  return pyr;
}
```



$L_3 = G_3$

$L_2 = G_2 - rescale(G_3)$

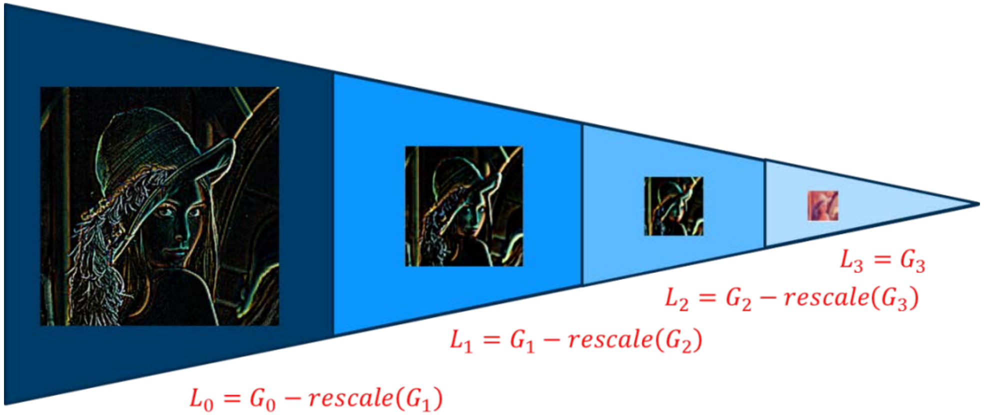
$L_1 = G_1 - rescale(G_2)$

$L_0 = G_0 - rescale(G_1)$

Step 4: Laplace blending

- Reconstruct an image by collapsing a laplacian pyramid

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



$L_3 = G_3$

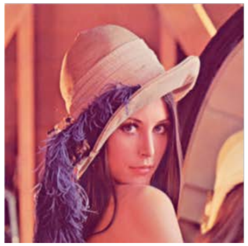
$L_2 = G_2 - rescale(G_3)$

$L_1 = G_1 - rescale(G_2)$

$L_0 = G_0 - rescale(G_1)$

Collapsing the Laplacian pyramid:

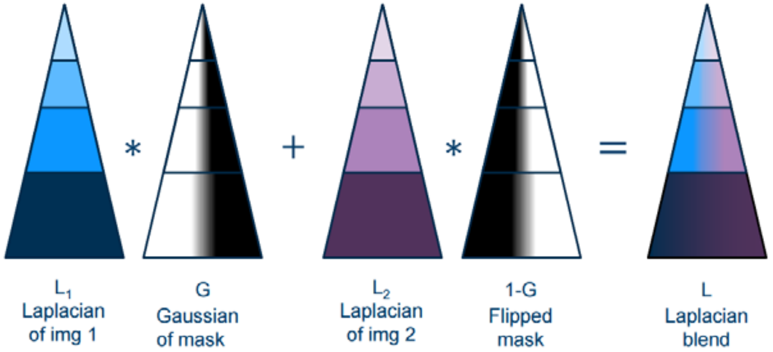
$rescale(rescale(rescale(L_3) + L_2) + L_1) + L_0 =$



Step 4: Laplace blending

- Perform the laplace blending

```
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().
  }

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



Step 4: Laplace blending

- Compare the result with linear blending
  - Try different ramp widths

Step 5: Extra fun!

- Try other images
  - Take images using the camera
  - Download images from the net
  - Co-register the images:

```
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());
```

- Try other masks
  - Circles
  - Download GIMP and draw some masks