# 参考1

# 使用 OpenCV 进行高动态范围（HDR）成像

在本教程中，我们将学习如何使用由不同曝光设置拍摄的多张图像创建（HDR）图像。 我们将以 C++ 和 Python 两种形式分享代码。

**什么是高动态范围成像？**

大多数数码相机和显示器都是按照 24 位矩阵捕获或者显示彩色图像。 每个颜色通道有 8 位，因此每个通道的像素值在 0-255 范围内。 换句话说，普通的相机或者显示器的动态范围是有限的。

但是，我们周围世界动态范围极大。 在车库内关灯就会变黑，直接看着太阳就会变得非常亮。 即使不考虑这些极端，在日常情况下，8 位的通道勉强可以捕捉到现场场景。 因此，相机会尝试去评估光照并且自动设置曝光，这样图像的最关注区域就会有良好的动态范围，并且太暗和太亮的部分会被相应截取为 0 和 255。

在下图中，左侧的图像是正常曝光的图像。 请注意，由于相机决定使用拍摄主体（我的儿子）的设置，所以背景中的天空已经完全流失了，但是明亮的天空也因此被刷掉了。 右侧的图像是由 iPhone 生成的HDR图像。



*High Dynamic Range (HDR)*

iPhone 是如何拍摄 HDR 图像的呢？ 它实际上采用三种不同的曝光度拍摄了 3 张图像，3 张图像拍摄非常迅速，在 3 张图像之间几乎没有产生位移。然后组合三幅图像来产生 HDR 图像。 我们将在下一节看到一些细节。

将在不同曝光设置下获取的相同场景的不同图像组合的过程称为高动态范围（HDR）成像。

**高动态范围（HDR）成像是如何工作的？**

在本节中，我们来看下使用 OpenCV 创建 HDR 图像的步骤。

要想轻松学习本教程，请点击[此处下载](http://www.learnopencv.com/wp-content/uploads/2017/10/hdr.zip) C++ 和 Python 代码还有图像。 如果您有兴趣了解更多关于人工智能，计算机视觉和机器学习的信息，请[订阅](https://bigvisionllc.leadpages.net/leadbox/143948b73f72a2%3A173c9390c346dc/5649050225344512/)我们的电子杂志。

**第 1 步：捕获不同曝光度的多张图像**

当我们使用相机拍照时，每个通道只有 8 位来表示场景的动态范围（亮度范围）。 但是，通过改变快门速度，我们可以在不同的曝光条件下拍摄多个场景图像。 大多数单反相机（SLR）有一个功能称为（AEB），只需按一下按钮，我们就可以在不同的曝光下拍摄多张照片。 如果你正在使用 iPhone，你可以使用这个[自动包围式 HDR 应用程序](https://itunes.apple.com/us/app/autobracket-hdr/id923626339?mt=8&amp;amp;ign-mpt=uo%3D8)，如果你是一个 Android 用户，你可以尝试一个[更好的相机应用程序](https://play.google.com/store/apps/details?id=com.almalence.opencam&amp;amp;hl=en)。

场景没有变化时，在相机上使用自动包围式曝光或在手机上使用自动包围式应用程序，我们可以一张接一张地快速拍摄多张照片。 当我们在 iPhone 中使用 HDR 模式时，会拍摄三张照片。

1. 曝光不足的图像：该图像比正确曝光的图像更暗。 目标是捕捉非常明亮的图像部分。
2. 正确曝光的图像：这是相机将根据其估计的照明拍摄的常规图像。
3. 曝光过度的图像：该图像比正确曝光的图像更亮。 目标是拍摄非常黑暗的图像部分。

但是，如果场景的动态范围很大，我们可以拍摄三张以上的图片来合成 HDR 图像。 在本教程中，我们将使用曝光时间为1/30 秒，0.25 秒，2.5 秒和 15 秒的 4 张图像。 缩略图如下所示。



*Auto Exposure Bracketed HDR image sequence*

单反相机或手机的曝光时间和其他设置的信息通常存储在 JPEG 文件的 EXIF 元数据中。 查看此[链接](https://www.howtogeek.com/289712/how-to-see-an-images-exif-data-in-windows-and-macos)可在 Windows 和 Mac 中查看存储在 JPEG 文件中的 EXIF 元数据。 或者，您可以使用我最喜欢的名为 [EXIFTOOL](https://www.sno.phy.queensu.ca/%7Ephil/exiftool) 的查看 EXIF 的命令行工具。

我们先从读取分配到不同曝光时间的图像开始。

**C++**

1. void readImagesAndTimes(vector<Mat> &images, vector<float> &times)
2. {
3. int numImages = 4;
4. *// 曝光时间列表*
5. static const float timesArray[] = {1/30.0f,0.25,2.5,15.0};
6. times.assign(timesArray, timesArray + numImages);
7. *// 图像文件名称列表*
8. static const char\* filenames[] = {"img\_0.033.jpg", "img\_0.25.jpg", "img\_2.5.jpg", "img\_15.jpg"};
9. for(int i=0; i < numImages; i++)
10. {
11. Mat im = imread(filenames[i]);
12. images.push\_back(im);
13. }
14. }

**Python**

1. def readImagesAndTimes():
2. *#* 曝光时间列表
3. times = np.array([ 1/30.0, 0.25, 2.5, 15.0 ], dtype=np.float32)
4. *#* 图像文件名称列表
5. filenames = ["img\_0.033.jpg", "img\_0.25.jpg", "img\_2.5.jpg", "img\_15.jpg"]
6. images = []
7. for filename in filenames:
8. im = cv2.imread(filename)
9. images.append(im)
10. return images, times

**第 2 步：对齐图像**

合成 HDR 图像时使用的图像如果未对齐可能会导致严重的伪影。 在下图中，左侧的图像是使用未对齐的图像组成的 HDR 图像，右侧的图像是使用对齐的图像的图像。 通过放大图像的一部分（使用红色圆圈显示的）我们会在左侧图像中看到严重的鬼影。



*Misalignment problem in HDR*

在拍摄照片制作 HDR 图像时，专业摄影师自然是将相机安装在三脚架上。 他们还使用称为[镜像锁定](https://www.slrlounge.com/workshop/using-mirror-up-mode-mirror-lockup)功能来减少额外的振动。 即使如此，图像可能仍然没有完美对齐，因为没有办法保证无振动的环境。 使用手持相机或手机拍摄图像时，对齐问题会变得更糟。

幸运的是，OpenCV 提供了一种简单的方法，使用 AlignMTB 对齐这些图像。 该算法将所有图像转换为（MTB）。 图像的 MTB 生成方式为将比中值亮度的更亮的分配为 1，其余为 0。 MTB 不随曝光时间的改变而改变。 因此不需要我们指定曝光时间就可以对齐 MTB。

基于 MTB 的对齐方式的代码如下。

**C++**

1. *// 对齐输入图像*
2. Ptr<AlignMTB> alignMTB = createAlignMTB();
3. alignMTB->process(images, images);

**Python**

1. *#* 对齐输入图像
2. alignMTB = cv2.createAlignMTB()
3. alignMTB.process(images, images)

**第 3 步：提取相机响应函数**

典型相机的响应与场景亮度不成线性关系。 那是什么意思呢？ 假设有两个物体由同一个相机拍摄，在现实世界中其中一个物体是另一个物体亮度的两倍。 当您测量照片中两个物体的像素亮度时，较亮物体的像素值将不会是较暗物体的两倍。 在不估计（CRF）的情况下，我们将无法将图像合并到一个HDR图像中。

将多个曝光图像合并为 HDR 图像意味着什么？

只考虑图像的某个位置 (x,y) 一个像素。 如果 CRF 是线性的，则像素值将直接与曝光时间成比例，除非像素在特定图像中太暗（即接近 0）或太亮（即接近 255）。 我们可以过滤出这些不好的像素（太暗或太亮），并且将像素值除以曝光时间来估计像素的亮度，然后在像素不差的（太暗或太亮）所有图像上对亮度值取平均。我们可以对所有像素进行这样的处理，并通过对“好”像素进行平均来获得所有像素的单张图像。

但是 CRF 不是线性的， 我们需要评估 CRF 把图像强度变成线性，然后才能合并或者平均它们。

好消息是，如果我们知道每个图像的曝光时间，则可以从图像估计 CRF。 与计算机视觉中的许多问题一样，找到 CRF 的问题本质是一个最优解问题，其目标是使由数据项和平滑项组成的目标函数最小化。 这些问题通常会降维到线性最小二乘问题，这些问题可以使用（SVD）来解决，奇异值分解是所有线性代数包的一部分。 CRF 提取算法的细节在[从照片提取高动态范围辐射图](http://www.pauldebevec.com/Research/HDR/debevec-siggraph97.pdf)这篇论文中可以找到。

使用 OpenCV 的 CalibrateDebevec 或者 CalibrateRobertson 就可以用 2 行代码找到 CRF。本篇教程中我们使用 CalibrateDebevec

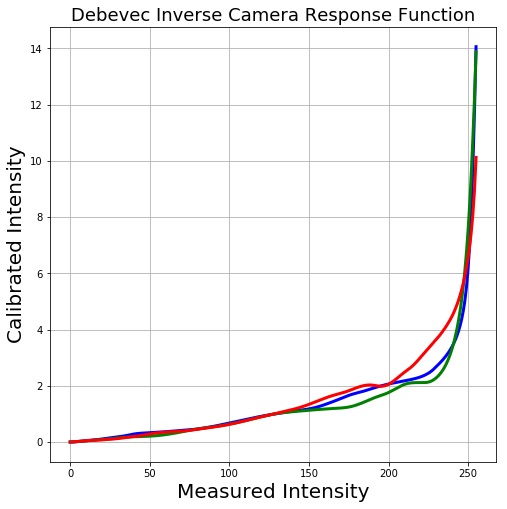
**C++**

1. *// 获取图像响应函数 (CRF)*
2. Mat responseDebevec;
3. Ptr<CalibrateDebevec> calibrateDebevec = createCalibrateDebevec();
4. calibrateDebevec->process(images, responseDebevec, times);

**Python**

1. *#* 获取图像响应函数 (CRF)
2. calibrateDebevec = cv2.createCalibrateDebevec()
3. responseDebevec = calibrateDebevec.process(images, times)

下图显示了使用红绿蓝通道的图像提取的 CRF。



*Camera Response Function*

**第 4 步：合并图像**

一旦 CRF 评估结束，我们可以使用 MergeDebevec 将曝光图像合并成一个HDR图像。 C++ 和 Python 代码如下所示。

**C++**

1. *// 将图像合并为HDR线性图像*
2. Mat hdrDebevec;
3. Ptr<MergeDebevec> mergeDebevec = createMergeDebevec();
4. mergeDebevec->process(images, hdrDebevec, times, responseDebevec);
5. *// 保存图像*
6. imwrite("hdrDebevec.hdr", hdrDebevec);

**Python**

1. *#* 将图像合并为HDR线性图像
2. mergeDebevec = cv2.createMergeDebevec()
3. hdrDebevec = mergeDebevec.process(images, times, responseDebevec)
4. *#* 保存图像
5. cv2.imwrite("hdrDebevec.hdr", hdrDebevec)

上面保存的 HDR 图像可以在 Photoshop 中加载并进行色调映射。示例图像如下所示。



*HDR Photoshop 色调映射*

**第 5 步：色调映射**

现在我们已经将我们的曝光图像合并到一个 HDR 图像中。 你能猜出这个图像的最小和最大像素值吗？ 对于黑色条件，最小值显然为 0。 理论最大值是什么？ 无限大！ 在实践中，不同情况下的最大值是不同的。 如果场景包含非常明亮的光源，那么最大值就会非常大。

尽管我们已经使用多个图像恢复了相对亮度信息，但是我们现在又面临了新的挑战：将这些信息保存为 24 位图像用于显示。

将高动态范围（HDR）图像转换为 8 位单通道图像的过程称为色调映射。这个过程的同时还需要保留尽可能多的细节。

有几种色调映射算法。 OpenCV 实现了其中的四个。 要记住的是没有一个绝对正确的方法来做色调映射。 通常，我们希望在色调映射图像中看到比任何一个曝光图像更多的细节。 有时色调映射的目标是产生逼真的图像，而且往往是产生超现实图像的目标。 在 OpenCV 中实现的算法倾向于产生现实的并不那么生动的结果。

我们来看看各种选项。 以下列出了不同色调映射算法的一些常见参数。

1. ：该参数通过应用伽马校正来压缩动态范围。 当伽马等于 1 时，不应用修正。 小于 1 的伽玛会使图像变暗，而大于 1 的伽马会使图像变亮。
2. ：该参数用于增加或减少饱和度。 饱和度高时，色彩更丰富，更浓。 饱和度值接近零，使颜色逐渐消失为灰度。
3. ：控制输出图像的对比度（即 log(maxPixelValue/minPixelValue)）。

让我们来探索 OpenCV 中可用的四种色调映射算法。

Drago 色调映射

Drago 色调映射的参数如下所示：

1. createTonemapDrago
2. (
3. float gamma = 1.0f,
4. float saturation = 1.0f,
5. float bias = 0.85f
6. )

这里，bias 是 [0, 1] 范围内偏差函数的值。 从 0.7 到 0.9 的值通常效果较好。 默认值是 0.85。 有关更多技术细节，请参阅这篇[论文](http://resources.mpi-inf.mpg.de/tmo/logmap/logmap.pdf)。

C++ 和 Python 代码如下所示。 参数是通过反复试验获得的。 最后的结果乘以 3 只是因为它给出了最令人满意的结果。

**C++**

1. *// 使用Drago色调映射算法获得24位彩色图像*
2. Mat ldrDrago;
3. Ptr<TonemapDrago> tonemapDrago = createTonemapDrago(1.0, 0.7);
4. tonemapDrago->process(hdrDebevec, ldrDrago);
5. ldrDrago = 3 \* ldrDrago;
6. imwrite("ldr-Drago.jpg", ldrDrago \* 255);

**Python**

1. *#* 使用Drago色调映射算法获得24位彩色图像
2. tonemapDrago = cv2.createTonemapDrago(1.0, 0.7)
3. ldrDrago = tonemapDrago.process(hdrDebevec)
4. ldrDrago = 3 \* ldrDrago
5. cv2.imwrite("ldr-Drago.jpg", ldrDrago \* 255)

结果如下：



*使用Drago算法的HDR色调映射*

Durand 色调映射

Durand 色调映射的参数如下所示：

1. createTonemapDurand
2. (
3. float gamma = 1.0f,
4. float contrast = 4.0f,
5. float saturation = 1.0f,
6. float sigma\_space = 2.0f,
7. float sigma\_color = 2.0f
8. );

该算法基于将图像分解为基础层和细节层。 使用称为双边滤波器的边缘保留滤波器来获得基本层。 sigma\_space 和sigma\_color 是双边滤波器的参数，分别控制空间域和彩色域中的平滑量。

有关更多详细信息，请查看这篇[论文](https://people.csail.mit.edu/fredo/PUBLI/Siggraph2002/DurandBilateral.pdf)。

**C++**

1. *// 使用Durand色调映射算法获得24位彩色图像*
2. Mat ldrDurand;
3. Ptr<TonemapDurand> tonemapDurand = createTonemapDurand(1.5,4,1.0,1,1);
4. tonemapDurand->process(hdrDebevec, ldrDurand);
5. ldrDurand = 3 \* ldrDurand;
6. imwrite("ldr-Durand.jpg", ldrDurand \* 255);

**Python**

1. *#* 使用Durand色调映射算法获得24位彩色图像
2. tonemapDurand = cv2.createTonemapDurand(1.5,4,1.0,1,1)
3. ldrDurand = tonemapDurand.process(hdrDebevec)
4. ldrDurand = 3 \* ldrDurand
5. cv2.imwrite("ldr-Durand.jpg", ldrDurand \* 255)

结果如下：



*使用Durand算法的HDR色调映射*

Reinhard 色调映射

1. createTonemapReinhard
2. (
3. float gamma = 1.0f,
4. float intensity = 0.0f,
5. float light\_adapt = 1.0f,
6. float color\_adapt = 0.0f
7. )

intensity 参数应在 [-8, 8] 范围内。 更高的亮度值会产生更明亮的结果。 light\_adapt 控制灯光，范围为 [0, 1]。 值 1 表示仅基于像素值的自适应，而值 0 表示全局自适应。 中间值可以用于两者的加权组合。 参数 color\_adapt 控制色彩，范围为 [0, 1]。 如果值被设置为 1，则通道被独立处理，如果该值被设置为 0，则每个通道的适应级别相同。中间值可以用于两者的加权组合。

有关更多详细信息，请查看这篇[论文](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.106.8100&amp;amp;rep=rep1&amp;amp;type=pdf)。

**C++**

1. *// 使用Reinhard色调映射算法获得24位彩色图像*
2. Mat ldrReinhard;
3. Ptr<TonemapReinhard> tonemapReinhard = createTonemapReinhard(1.5, 0,0,0);
4. tonemapReinhard->process(hdrDebevec, ldrReinhard);
5. imwrite("ldr-Reinhard.jpg", ldrReinhard \* 255);

**Python**

1. *#* 使用Reinhard色调映射算法获得24位彩色图像
2. tonemapReinhard = cv2.createTonemapReinhard(1.5, 0,0,0)
3. ldrReinhard = tonemapReinhard.process(hdrDebevec)
4. cv2.imwrite("ldr-Reinhard.jpg", ldrReinhard \* 255)

结果如下：



*使用Reinhard算法的HDR色调映射*

Mantiuk 色调映射

1. createTonemapMantiuk
2. (
3. float gamma = 1.0f,
4. float scale = 0.7f,
5. float saturation = 1.0f
6. )

参数 scale 是对比度比例因子。 从 0.7 到 0.9 的值通常效果较好

有关更多详细信息，请查看这篇[论文](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.60.4077&amp;amp;rep=rep1&amp;amp;type=pdf)。

**C++**

1. *// 使用Mantiuk色调映射算法获得24位彩色图像*
2. Mat ldrMantiuk;
3. Ptr<TonemapMantiuk> tonemapMantiuk = createTonemapMantiuk(2.2,0.85, 1.2);
4. tonemapMantiuk->process(hdrDebevec, ldrMantiuk);
5. ldrMantiuk = 3 \* ldrMantiuk;
6. imwrite("ldr-Mantiuk.jpg", ldrMantiuk \* 255);

**Python**

1. *#* 使用Mantiuk色调映射算法获得24位彩色图像
2. tonemapMantiuk = cv2.createTonemapMantiuk(2.2,0.85, 1.2)
3. ldrMantiuk = tonemapMantiuk.process(hdrDebevec)
4. ldrMantiuk = 3 \* ldrMantiuk
5. cv2.imwrite("ldr-Mantiuk.jpg", ldrMantiuk \* 255)

结果如下：



*使用Mantiuk算法的HDR色调映射*

**订阅然后下载代码**

如果你喜欢这篇文章，并希望下载本文中使用的代码（C++ 和 Python）和示例图片，请[订阅](https://bigvisionllc.leadpages.net/leadbox/143948b73f72a2%3A173c9390c346dc/5649050225344512/)我们的电子杂志。 您还将获得免费的[计算机视觉资源](https://bigvisionllc.leadpages.net/leadbox/143948b73f72a2%3A173c9390c346dc/5649050225344512/)指南。 在我们的电子杂志中，我们分享了用 C++ 还有 Python 编写的 OpenCV 教程和例子，以及计算机视觉和机器学习的算法和新闻。

[点此订阅](https://bigvisionllc.leadpages.net/leadbox/143948b73f72a2%3A173c9390c346dc/5649050225344512/)

图片致谢

本文中使用的四个曝光图像获得 [CC BY-SA 3.0](https://creativecommons.org/licenses/by-sa/3.0/) 许可，并从[维基百科的 HDR 页面](https://en.wikipedia.org/wiki/High-dynamic-range_imaging)下载。 图像由 Kevin McCoy拍摄。

via: <http://www.learnopencv.com/high-dynamic-range-hdr-imaging-using-opencv-cpp-python/>