# Creating a Bitmap Graphics Context

位图图形上下文接受指向包含位图存储空间的内存缓冲区的指针。当您绘制位图图形上下文时，缓冲区会更新。在您释放图形上下文之后，您可以使用指定的像素格式更新位图。

注：位图图形上下文有时用于离屏渲染。如因离屏渲染而使用位图图形上下文，请参阅[Core Graphics Layer Drawing](https://developer.apple.com/library/content/documentation/GraphicsImaging/Conceptual/drawingwithquartz2d/dq_layers/dq_layers.html#//apple_ref/doc/uid/TP30001066-CH219-TPXREF101)。CGLayer对象（CGLayerRef）针对离屏渲染进行了优化，因为只要可能，Quartz会在视频帧上缓存图层。

iOS注意：iOS应用程序应该使用函数UIGraphicsBeginImageContextWithOptions，而不是使用这里描述的低级Quartz函数。如果应用程序使用Quartz创建离屏位图，则位图图形上下文使用的坐标系是默认的Quartz坐标系。相比之下，如果您的应用程序通过调用函数UIGraphicsBeginImageContextWithOptions来创建图像上下文，则UIKit将相应的转换应用于上下文的坐标系统，就像它对UIView对象的图形上下文所做的那样。这使您的应用程序可以使用相同的绘图代码，而无需担心不同的坐标系。虽然您的应用程序可以手动调整坐标转换矩阵以获得正确的结果，但在实践中，这样做没有性能优势。

您可以使用函数CGBitmapContextCreate创建位图图形上下文。该功能采用以下参数：

* 数据。提供一个指向内存中您希望绘制渲染的目的地的指针。此内存块的大小应至少为（bytesPerRow \* height）个字节。
* 宽度。指定位图的宽度（以像素为单位）。
* 高度。指定位图的高度（以像素为单位）。
* bitsPerComponent。指定内存中每个像素组件使用的位数。例如，对于32位像素格式和RGB颜色空间，您可以指定每个组件8位的值。请参阅支持的像素格式。
* bytesPerRow。指定每行位图使用的内存字节数。

提示：创建位图图形上下文时，如果确保数据和bytesPerRow是16字节对齐的，则会获得最佳性能。

* 色彩空间。用于位图上下文的颜色空间。创建位图图形上下文时，可以提供灰色，RGB，CMYK或NULL色彩空间。有关色彩空间和色彩管理原理的详细信息，请参阅色彩管理概述。有关在Quartz中创建和使用颜色空间的信息，请参阅颜色和颜色空间。有关支持的颜色空间的信息，请参阅位图图像和图像蒙版章节中的颜色空间和位图布局。
* 位图布局信息。 以CGBitmapInfo常量表示，该信息指定位图是否应包含alpha分量，像素中alpha分量的相对位置（如果存在），alpha分量是否为预乘，以及颜色分量是整数或浮点值。有关这些常量是什么的详细信息，以及Quartz支持的用于位图图形上下文和图像的像素格式，请参阅位图图像和图像蒙版章节中的颜色空间和位图布局。

清单2-5显示了如何创建一个位图图形上下文。当您绘制所得到的位图图形上下文时，Quartz会将您的图形作为位图数据记录在指定的内存块中。列表中的每一行代码都有详细的解释。

清单2-5创建一个位图图形上下文

|  |
| --- |
| CGContextRef MyCreateBitmapContext (int pixelsWide, |
| int pixelsHigh) |
| { |
| CGContextRef context = NULL; |
| CGColorSpaceRef colorSpace; |
| void \* bitmapData; |
| int bitmapByteCount; |
| int bitmapBytesPerRow; |
|  |
| bitmapBytesPerRow = (pixelsWide \* 4);// 1 |
| bitmapByteCount = (bitmapBytesPerRow \* pixelsHigh); |
|  |
| colorSpace = CGColorSpaceCreateWithName(kCGColorSpaceGenericRGB);// 2 |
| bitmapData = calloc( bitmapByteCount, sizeof(uint8\_t) );// 3 |
| if (bitmapData == NULL) |
| { |
| fprintf (stderr, "Memory not allocated!"); |
| return NULL; |
| } |
| context = CGBitmapContextCreate (bitmapData,// 4 |
| pixelsWide, |
| pixelsHigh, |
| 8, // bits per component |
| bitmapBytesPerRow, |
| colorSpace, |
| kCGImageAlphaPremultipliedLast); |
| if (context== NULL) |
| { |
| free (bitmapData);// 5 |
| fprintf (stderr, "Context not created!"); |
| return NULL; |
| } |
| CGColorSpaceRelease( colorSpace );// 6 |
|  |
| return context;// 7 |
| } |

以下是代码解读：

1.声明一个变量来表示每行的字节数。本例中位图中的每个像素由4个字节表示;红色，绿色，蓝色和alpha各8位(字节)。

2.创建一个通用的RGB颜色空间。您也可以创建一个CMYK颜色空间。请参阅颜色和颜色空间以获取更多信息，以及讨论通用色彩空间与依赖设备的色彩空间。

3.调用calloc函数以创建并清除存储位图数据的内存块。本示例创建一个32位RGBA位图（即，每像素32位的数组，每个像素包含8位红色，绿色，蓝色和alpha信息）。位图中的每个像素占用4个字节的内存。在Mac OS X 10.6和iOS 4中，这一步可以省略 - 如果您将NULL作为位图数据传递，则Quartz自动为位图分配空间。

4.创建位图图形上下文，提供位图数据，位图的宽度和高度，每个组件的位数，每行的字节数，颜色空间以及指定位图是否应包含Alpha通道及其像素中的相对位置。常数kCGImageAlphaPremultipliedLast指示alpha分量存储在每个像素的最后一个字节中，并且颜色分量已经乘以该alpha值。有关预乘alpha的更多信息，请参阅Alpha值。

5.如果由于某种原因未创建上下文，则释放为位图数据分配的内存。

6.释放颜色空间。

7.返回位图图形上下文。调用者必须在不再需要时释放图形上下文。

清单2-6显示了调用MyCreateBitmapContext来创建位图图形上下文，使用位图图形上下文创建CGImage对象，然后将结果图像绘制到窗口图形上下文的代码。图2-3显示了绘制到窗口的图像。列表中的每一行代码都有详细的解释。

清单2-6绘制到位图图形上下文

|  |
| --- |
| CGRect myBoundingBox;// 1 |
|  |
| myBoundingBox = CGRectMake (0, 0, myWidth, myHeight);// 2 |
| myBitmapContext = MyCreateBitmapContext (400, 300);// 3 |
| // \*\*\*\*\*\*\*\*\*\* Your drawing code here \*\*\*\*\*\*\*\*\*\* // 4 |
| CGContextSetRGBFillColor (myBitmapContext, 1, 0, 0, 1); |
| CGContextFillRect (myBitmapContext, CGRectMake (0, 0, 200, 100 )); |
| CGContextSetRGBFillColor (myBitmapContext, 0, 0, 1, .5); |
| CGContextFillRect (myBitmapContext, CGRectMake (0, 0, 100, 200 )); |
| myImage = CGBitmapContextCreateImage (myBitmapContext);// 5 |
| CGContextDrawImage(myContext, myBoundingBox, myImage);// 6 |
| char \*bitmapData = CGBitmapContextGetData(myBitmapContext); // 7 |
| CGContextRelease (myBitmapContext);// 8 |
| if (bitmapData) free(bitmapData); // 9 |
| CGImageRelease(myImage); |

以下是代码步骤解读：

1.声明一个变量来存储边界框的原点和尺寸，Quartz将绘制从位图图形上下文创建的图像。

2.将边界框的原点设置为（0,0），并将宽度和高度设置为先前声明的变量，但其声明未在此代码中显示。

3.调用应用程序提供的函数MyCreateBitmapContext（参见清单2-5）创建一个400像素宽，300像素高的位图上下文。您可以使用适合您应用程序的任何尺寸创建位图图形上下文。

4.调用Quartz 2D函数以绘制位图图形上下文。您可以用适合您应用程序的绘图代码替换此代码和接下来的四行代码。

5.从位图图形上下文创建一个Quartz 2D图像（CGImageRef）。

6.将图像绘制到由边界框指定的窗口图形上下文中的位置。边界框指定绘制图像的用户空间中的位置和尺寸。

此示例不显示创建窗口图形上下文。有关如何创建窗口图形上下文的信息，请参阅在Mac OS X中创建窗口图形上下文。

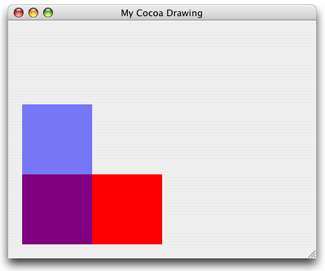
7.获取与位图图形上下文关联的位图数据。

8.不再需要时释放位图图形上下文。

9.如果存在，则释放位图数据。

10.不再需要时释放图像。

图2-3从位图图形上下文创建并绘制到窗口图形上下文的图像



支持的像素格式

表2-1总结了位图图形上下文支持的像素格式，关联的颜色空间（cs）以及格式第一次可用的Mac OS X版本。 像素格式被指定为每像素位数（bpp）和每个分量位数（bpc）。 该表还包括与该像素格式相关的位图信息常量。 有关每个位图信息格式常量代表的详细信息，请参阅CGImage Reference。

表2-1位图图形上下文支持的像素格式

|  |  |  |
| --- | --- | --- |
| **CS** | **Pixel format and bitmap information constant** | **Availability** |
| Null | 8 bpp, 8 bpc, [kCGImageAlphaOnly](https://developer.apple.com/documentation/coregraphics/cgimagealphainfo/kcgimagealphaonly" \t "_self) | Mac OS X, iOS |
| Gray | 8 bpp, 8 bpc,[kCGImageAlphaNone](https://developer.apple.com/documentation/coregraphics/cgimagealphainfo/1455164-none" \t "_self) | Mac OS X, iOS |
| Gray | 8 bpp, 8 bpc,[kCGImageAlphaOnly](https://developer.apple.com/documentation/coregraphics/cgimagealphainfo/kcgimagealphaonly" \t "_self) | Mac OS X, iOS |
| Gray | 16 bpp, 16 bpc, [kCGImageAlphaNone](https://developer.apple.com/documentation/coregraphics/cgimagealphainfo/1455164-none" \t "_self) | Mac OS X |
| Gray | 32 bpp, 32 bpc, [kCGImageAlphaNone](https://developer.apple.com/documentation/coregraphics/cgimagealphainfo/1455164-none" \t "_self)|[kCGBitmapFloatComponents](https://developer.apple.com/documentation/coregraphics/cgbitmapinfo/kcgbitmapfloatcomponents" \t "_self) | Mac OS X |
| RGB | 16 bpp, 5 bpc, [kCGImageAlphaNoneSkipFirst](https://developer.apple.com/documentation/coregraphics/cgimagealphainfo/kcgimagealphanoneskipfirst" \t "_self) | Mac OS X, iOS |
| RGB | 32 bpp, 8 bpc, [kCGImageAlphaNoneSkipFirst](https://developer.apple.com/documentation/coregraphics/cgimagealphainfo/kcgimagealphanoneskipfirst" \t "_self) | Mac OS X, iOS |
| RGB | 32 bpp, 8 bpc, [kCGImageAlphaNoneSkipLast](https://developer.apple.com/documentation/coregraphics/cgimagealphainfo/kcgimagealphanoneskiplast" \t "_self) | Mac OS X, iOS |
| RGB | 32 bpp, 8 bpc, [kCGImageAlphaPremultipliedFirst](https://developer.apple.com/documentation/coregraphics/cgimagealphainfo/1456607-premultipliedfirst" \t "_self) | Mac OS X, iOS |
| RGB | 32 bpp, 8 bpc, [kCGImageAlphaPremultipliedLast](https://developer.apple.com/documentation/coregraphics/cgimagealphainfo/1454112-premultipliedlast" \t "_self) | Mac OS X, iOS |
| RGB | 64 bpp, 16 bpc, [kCGImageAlphaPremultipliedLast](https://developer.apple.com/documentation/coregraphics/cgimagealphainfo/1454112-premultipliedlast" \t "_self) | Mac OS X |
| RGB | 64 bpp, 16 bpc, [kCGImageAlphaNoneSkipLast](https://developer.apple.com/documentation/coregraphics/cgimagealphainfo/kcgimagealphanoneskiplast" \t "_self) | Mac OS X |
| RGB | 128 bpp, 32 bpc, [kCGImageAlphaNoneSkipLast](https://developer.apple.com/documentation/coregraphics/cgimagealphainfo/kcgimagealphanoneskiplast" \t "_self) |[kCGBitmapFloatComponents](https://developer.apple.com/documentation/coregraphics/cgbitmapinfo/kcgbitmapfloatcomponents" \t "_self) | Mac OS X |
| RGB | 128 bpp, 32 bpc, [kCGImageAlphaPremultipliedLast](https://developer.apple.com/documentation/coregraphics/cgimagealphainfo/1454112-premultipliedlast" \t "_self) |[kCGBitmapFloatComponents](https://developer.apple.com/documentation/coregraphics/cgbitmapinfo/kcgbitmapfloatcomponents" \t "_self) | Mac OS X |
| CMYK | 32 bpp, 8 bpc, [kCGImageAlphaNone](https://developer.apple.com/documentation/coregraphics/cgimagealphainfo/1455164-none" \t "_self) | Mac OS X |
| CMYK | 64 bpp, 16 bpc, [kCGImageAlphaNone](https://developer.apple.com/documentation/coregraphics/cgimagealphainfo/1455164-none" \t "_self) | Mac OS X |
| CMYK | 128 bpp, 32 bpc, [kCGImageAlphaNone](https://developer.apple.com/documentation/coregraphics/cgimagealphainfo/1455164-none" \t "_self) |[kCGBitmapFloatComponents](https://developer.apple.com/documentation/coregraphics/cgbitmapinfo/kcgbitmapfloatcomponents" \t "_self) | Mac OS X |

抗锯齿

位图图形上下文支持消除锯齿，这是人为地纠正绘制文本或形状时有时在位图图像中看到的锯齿状（或锯齿形）边缘的过程。当位图的分辨率明显低于眼睛的分辨率时，会出现这些锯齿状边缘。为了使对象在位图中看起来平滑，Quartz对形状轮廓周围的像素使用不同的颜色。通过以这种方式混合颜色，形状显得平滑。您可以在图2-4中看到使用抗锯齿的效果。您可以通过调用函数CGContextSetShouldAntialias来关闭特定位图图形上下文的消除锯齿。抗锯齿设置是图形状态的一部分。

您可以通过使用函数CGContextSetAllowsAntialiasing来控制是否允许对特定图形上下文进行消除锯齿。通过这个函数来实现抗锯齿;假不允许它。此设置不是图形状态的一部分。当上下文和图形状态设置设置为true时，Quartz执行消除锯齿。

图2-4混叠和抗锯齿绘图的比较

