

Bitmap Shapes

This chapter describes bitmap shapes and the functions you use to manipulate them. It also discusses functions described in other chapters and shows how you can apply them to bitmap shapes.

Before you read this chapter, you should be familiar with the information in the chapter “Shape Objects” of *Inside Macintosh: QuickDraw GX Objects*, and you will probably want to be familiar with much of the information discussed in the chapters “Color and Color-Related Objects,” “Transform Objects,” and “View-Related Objects,” also in that book.

This chapter introduces bitmap shapes, describes bitmap geometries, and then shows how to

- define bitmap geometries
- create bitmap shapes
- draw bitmap shapes
- manipulate the pixel image stored in a bitmap shape
- apply transfer modes and transformations to bitmap shapes
- draw other QuickDraw GX objects into a bitmap shape
- create bitmap shapes with disk-based pixel images
- replace a part of a bitmap shape’s pixel image

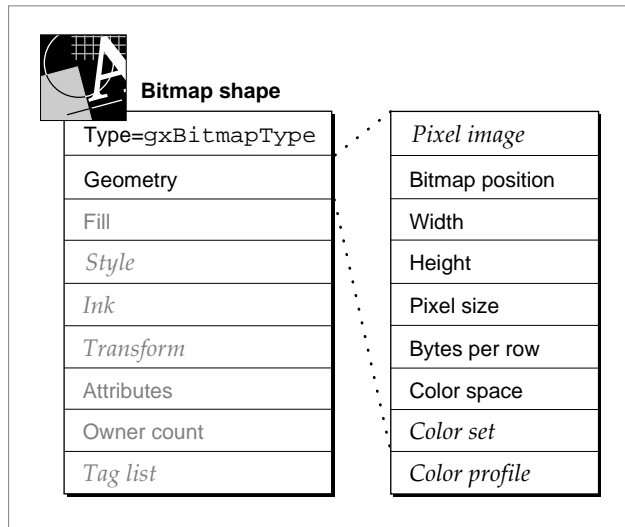
About Bitmap Shapes

Like all shapes, a **bitmap shape** is represented in memory by a shape object, a style object, an ink object, and a transform object. A shape object representing a bitmap shape contains the same properties as a shape object representing a geometric or typographic shape: owner count, tag list, shape type, shape fill, geometry, and so on.

Bitmap Shapes

Figure 5-1 shows a graphic representation of a bitmap shape and a bitmap geometry.

Figure 5-1 A bitmap shape



Bitmap shapes make extensive use of their geometry property. In fact, most of the information useful to bitmap shapes is stored in their geometry—the values of the bitmap’s pixels, the dimensions of the bitmap, and the color information used by the bitmap.

Bitmap shapes don’t make much use of their shape fill property, and they use very little of their associated style object. In fact, the only information in a style object used by bitmap shapes are the style attributes that determine whether the upper-left corner of the bitmap should be constrained to an integer grid position.

Bitmap shapes don’t use the color property of their ink objects because they store their own color information in their geometries. However, they do use the transfer mode property of their ink objects.

Bitmap shapes do make full use of their transform objects. For example, you can scale, skew, rotate, and clip bitmap shapes. You can also hit-test bitmap shapes, but you cannot hit-test parts of a bitmap shape, as you can for other types of shapes. For more information about transform objects and hit-testing, see the chapter “Transform Objects” of *Inside Macintosh: QuickDraw GX Objects*.

The next few sections discuss bitmap geometries, bitmap styles, bitmap inks, and bitmap transforms.

Bitmap Geometries

The geometry of a bitmap contains eight fields:

- The **pixel image**—a pointer to a two-dimensional array of **pixel values**. Each pixel value represents the color of one pixel of the bitmap.
- The **bitmap width**—the number of pixels in each row of the bitmap.
- The **bitmap height**—the number of pixels in each column of the bitmap.
- The **pixel size**—the number of bits required to represent the color information for each pixel of the bitmap.
- The **bytes per row**—the number of bytes of the pixel image that correspond to each row of the bitmap.
- The **bitmap color space**—or the color space that determines how QuickDraw GX translates the bitmap's pixel values into colors. If the bitmap has any color space except indexed space, each pixel value in the pixel image represents a color specification in this color space. If the bitmap has an indexed color space, each pixel value is interpreted using the bitmap color set.
- The **bitmap color set**—the optional array of color values associated with the bitmap. If the bitmap uses a color set (also called an *indexed color space*), each pixel value in the bitmap's pixel image represents an index into this color set.
- The **bitmap color profile**—the color-matching information that you can specify for the device on which the bitmap was created.
- The **bitmap position**—the position of the upper-left corner of the bitmap. The actual position of the bitmap when drawn may differ depending on the information in the bitmap's transform object.

QuickDraw GX provides the bitmap data type which you can use to create bitmap geometries. The bitmap data type has a field corresponding to every field of a bitmap geometry except the bitmap position. You must set and determine the bitmap position programmatically.

QuickDraw GX enforces a few restrictions on the values of these geometry fields. For example, the pixel size of the bitmap must be a power of 2 (from 1 to 32), and it must correspond to the pixel size implicit in the bitmap color space.

The bytes per row of the bitmap must be a multiple of 2. This requirement allows for faster bitmap manipulation.

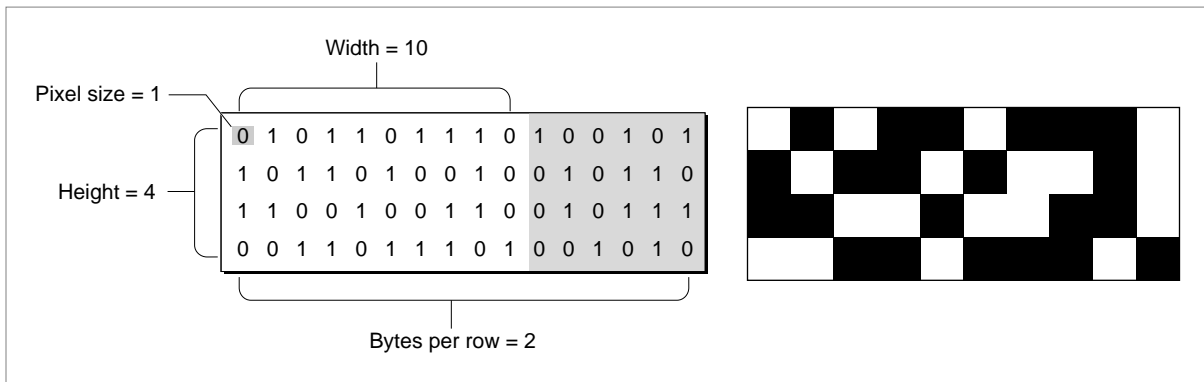
Note

Although the Macintosh platform accepts any even number of bytes per row, you might want to use a multiple of 4 bytes per row in your bitmaps to promote cross-platform compatibility. ♦

Bitmap Shapes

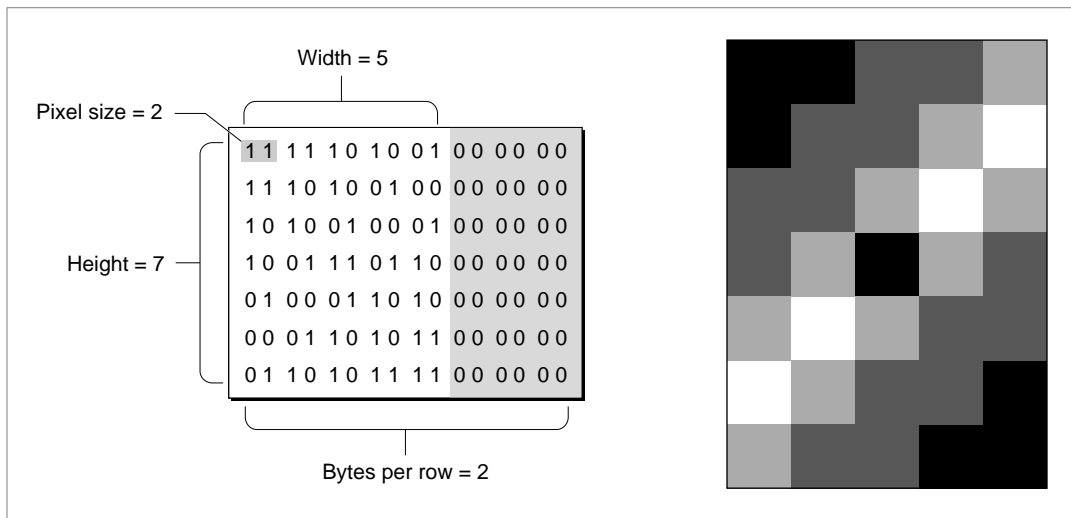
Sometimes you must pad a pixel image with extra bits to get an even number of bytes per row. For example, Figure 5-2 shows a small, black-and-white (1 bit per pixel) bitmap with a bitmap width of 10. The smallest number of bytes per row into which 10 bits fit is 2, so the number of bytes per row for this bitmap is 2. Although the six extra bits at the end of each row of the pixel image have values, they do not appear as pixels in the bitmap when it is drawn.

Figure 5-2 A black-and-white bitmap geometry



All QuickDraw GX bitmaps are actually color bitmaps. A black-and-white bitmap is simply a color bitmap with a color set containing only two colors—black and white.

Figure 5-3 shows another bitmap. In this example, the pixel size is 2; each pixel is represented by 2 bits. Since no color space has an implicit pixel size of 2, this bitmap uses a color set instead of a color space. In this example, the color set contains four shades of gray (although it could contain any four colors); each pixel value in the pixel image is an index into this color set. Since the bitmap width is 5 pixels and the pixel size is 2 bits per pixel, at least 10 bits are required to represent each row of the pixel image. The smallest number of bytes per row that contains 10 bits is 2 bytes, so each row of the bitmap has 16 bits total. The last 6 bits are ignored by QuickDraw GX.

Figure 5-3 A grayscale bitmap geometry

QuickDraw GX allows you to store a bitmap's pixel image in one of three locations:

- You can allocate memory for the pixel image yourself. In this case, you provide QuickDraw GX with a pointer to the memory containing your pixel image. The section “Creating and Drawing Bitmaps” beginning on page 5-15 gives examples of allocating memory for a pixel image yourself and incorporating the pixel image into a bitmap shape.
- You can request that QuickDraw GX allocate the memory for you. In this case, you must use QuickDraw GX functions to draw into the bitmap and to edit it. If you want to edit the pixel image directly, you can use other QuickDraw GX functions to lock the image in memory and to request a pointer to it. However, if QuickDraw GX is storing the pixel image on an accelerator card, your application might not be able to edit the pixel image directly. The section “Creating Bitmaps Offscreen” beginning on page 5-45 gives an example of requesting that QuickDraw GX allocate memory for your pixel image.
- You can associate the bitmap with a **disk-based pixel image**—a pixel image stored in a disk file. In this case, you can use QuickDraw GX functions to read and draw the bitmap, but you cannot use QuickDraw GX functions to edit the bitmap. You must edit the bitmap directly using file-manipulation functions. The section “Creating Bitmaps With Disk-Based Pixel Images” beginning on page 5-44 shows how you can create a bitmap that uses a disk-based pixel image.

Bitmap Styles and Inks

Although bitmap shapes have style objects and ink objects, they do not make full use of the properties of these objects. Of the many properties of the style object, only the `styleAttributes` property affects bitmap shapes. In fact, only the `gxSourceGridStyle` and `gxDeviceGridStyle` style attributes affect bitmap shapes.

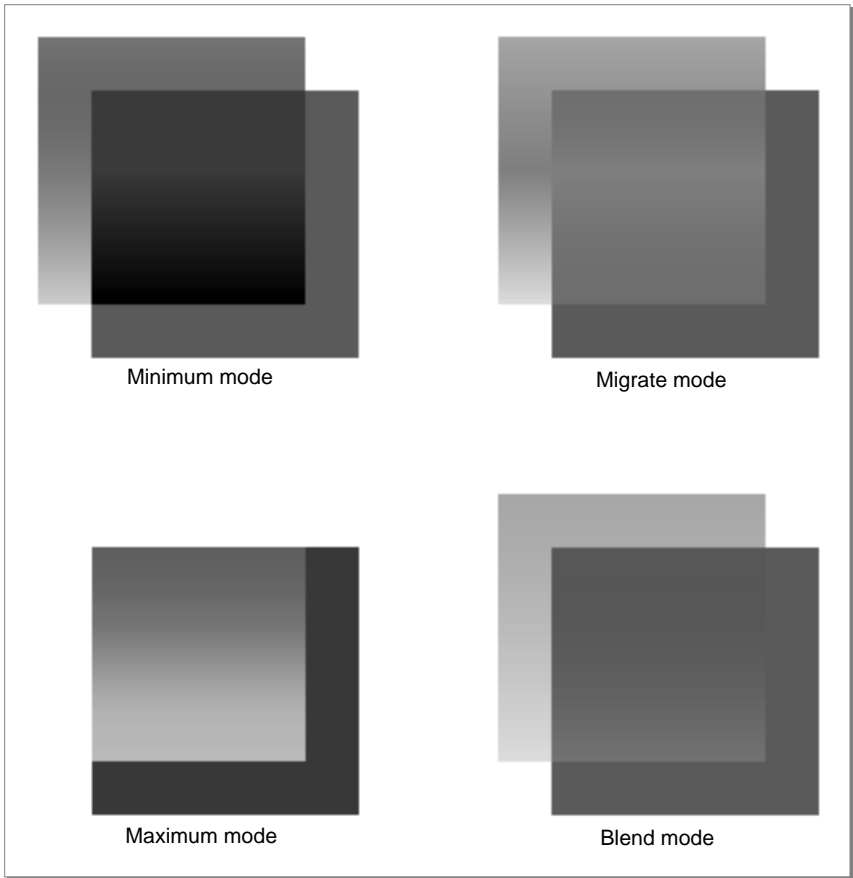
QuickDraw GX ignores the other style attributes and the other style properties when drawing a bitmap shape. You can set the values of these properties and determine the values you have set them to, but they do not affect how the bitmap is drawn.

See Chapter 3, “Geometric Styles,” for a description of how the `gxSourceGridStyle` and `gxDeviceGridStyle` style attributes affect shapes, including bitmap shapes.

Of the ink object properties, bitmaps use the `transferMode` property and ignore the `color` property. Since bitmap shapes have color information stored in their geometries, they do not need the color information stored in their ink objects. You can set the color of a bitmap shape’s ink object, but it does not affect how the bitmap is drawn.

The `transferMode` property, on the other hand, does affect the drawing of the bitmap. QuickDraw GX applies the `transferMode` as it draws each pixel of the bitmap, as shown in Figure 5-4. You can find a color version of this figure in Plate 1 at the front of this book.

Figure 5-4 The effect of transfer modes on bitmap shapes

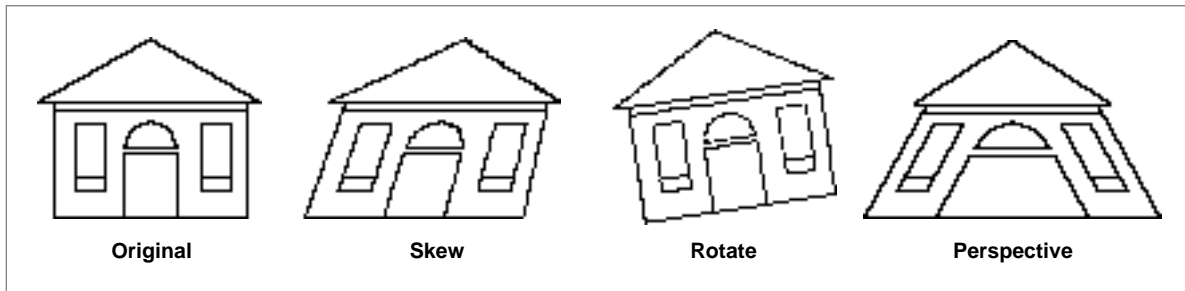


The section “Applying Transfer Modes to Bitmaps” beginning on page 5-32 shows you how to apply a transfer mode to a bitmap shape.

Bitmap Transforms

Although bitmap shapes make limited use of their style and ink objects, they make full use of their transform objects. Using the transform object, you can clip bitmap shapes and apply mapping transformations to them. Some examples are shown in Figure 5-5.

Figure 5-5 The effect of mappings on bitmap shapes

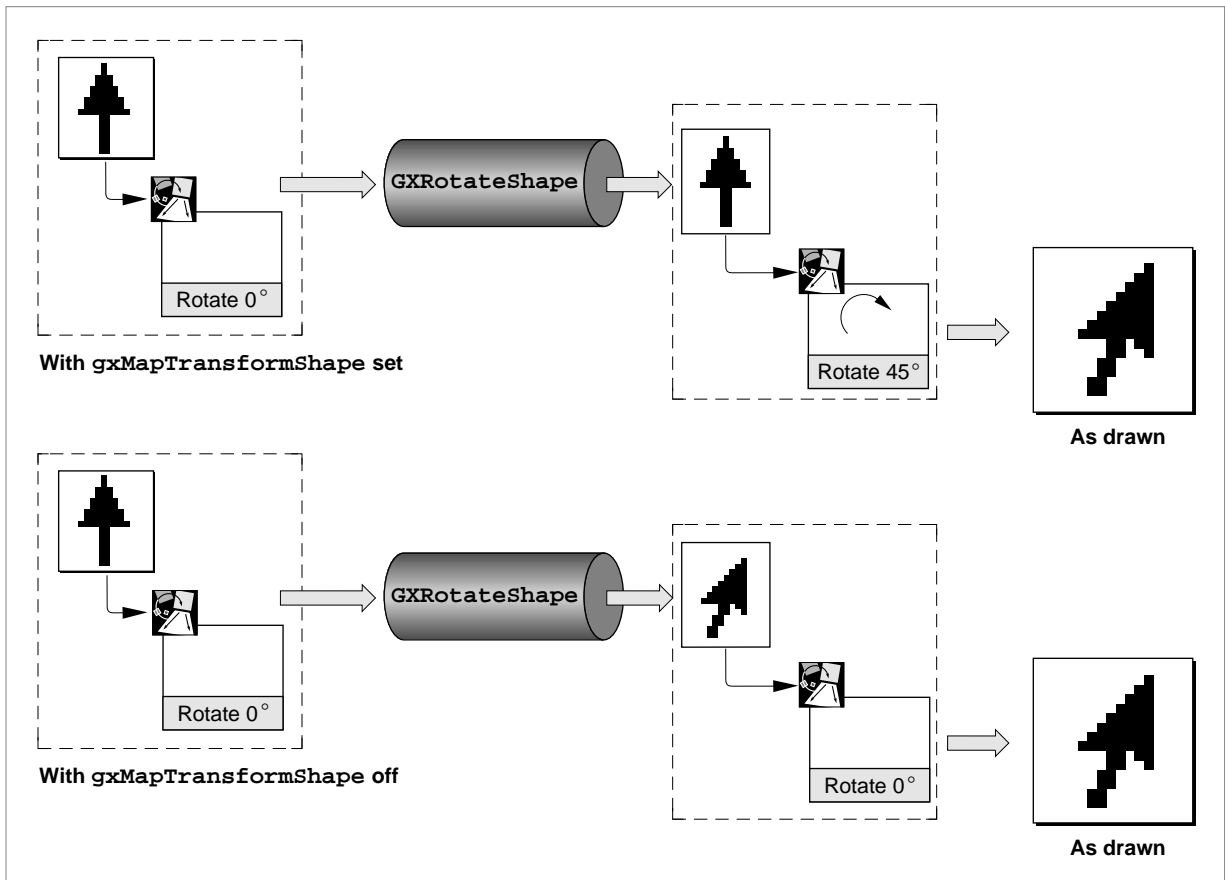


You can find examples of how to clip and map bitmap shapes in “Applying Transformations to Bitmaps,” which begins on page 5-38, and you can find more information about clipping and mapping in the chapter “Transform Objects” in *Inside Macintosh: QuickDraw GX Objects* as well as the chapter “Mathematical Functions” in *Inside Macintosh: QuickDraw GX Environment and Utilities*.

Bitmap shapes, like other types of shapes, use the `gxMapTransformShape` shape attribute to determine how mappings should be applied to the shape. If you set this shape attribute, applying a mapping to a bitmap shape changes the mapping matrix stored in the transform object of the bitmap shape. However, if you do not set this shape attribute, applying a mapping to a bitmap shape changes the geometry of the bitmap directly—that is, QuickDraw GX creates a completely new pixel image to represent the transformed bitmap.

Figure 5-6 compares the results of rotating a bitmap shape with and without the `gxMapTransformShape` shape attribute set.

Figure 5-6 The effect of the `gxMapTransformShape` shape attribute on bitmap mappings



Each mapping that you apply to a bitmap shape that does not have its `gxMapTransformShape` shape attribute set results in quality degradation of the bitmap's pixel image. If you apply multiple mappings to a bitmap shape that does not have this shape attribute set, error can arise rapidly. The section "Applying Transformations to Bitmaps," which begins on page 5-38, gives an example of this phenomenon.

You can find information about the `GXRotateShape` function in the chapter "Transform Objects" in *Inside Macintosh: QuickDraw GX Objects*.

Bitmaps and View Devices

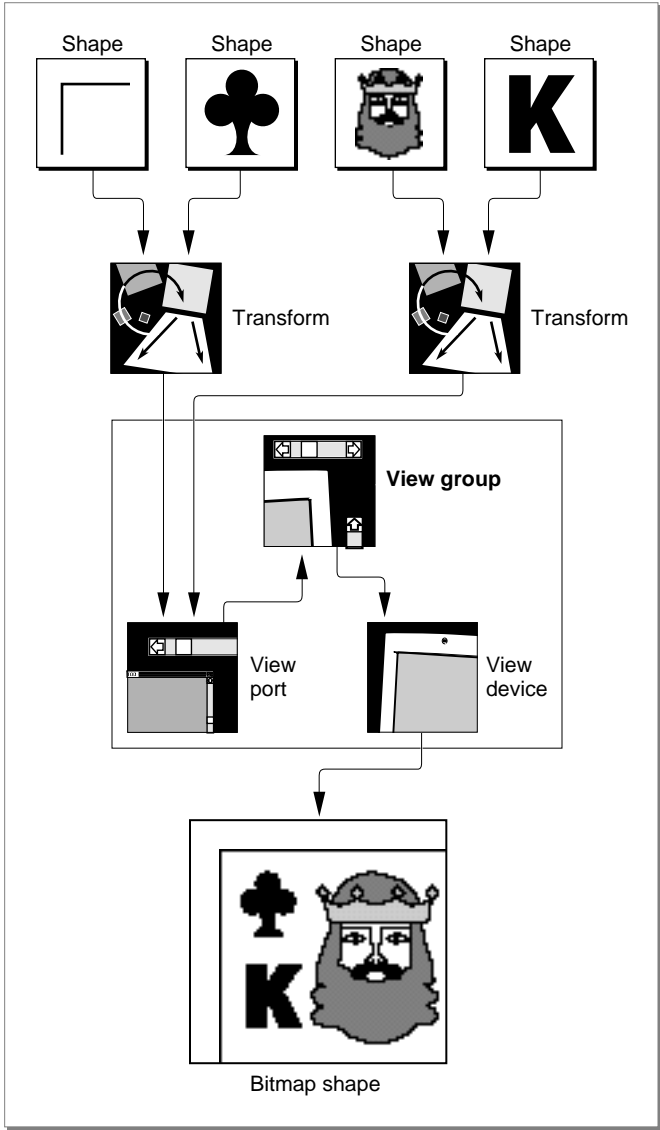
When you draw a shape, QuickDraw GX applies the information in the shape's style, ink, and transform objects to the shape's geometry and then renders the shape to the display devices that correspond to the view information contained in the shape's transform object.

The transform object of a shape contains a list of view ports to which QuickDraw GX should draw the shape. Each view port exists in the coordinate space of a specific view group, and each view group contains view devices that share the same coordinate space. QuickDraw GX determines where the shape appears in the coordinate space of each view group. If the area of the shape when drawn overlaps the area covered by any view device in that view group, QuickDraw GX renders the shape into the bitmap attached to that view device.

Figure 5-7 depicts how this drawing mechanism works with four shapes: a polygon shape, a path shape, a bitmap shape, and a text shape. The two path shapes share one transform object and the bitmap shape and the text shape share a second transform object.

Both of the transform objects contain one view port in their view port list. That view port exists in a view group that also contains a view device. The view device has a bitmap shape associated with it to hold the renderings of shapes drawn to it.

Figure 5-7 Bitmaps and view devices



Bitmap Shapes

Whenever you draw a QuickDraw GX shape, you are using this view architecture to render the shape to a display device. You can also use this view architecture to draw shapes into an **offscreen bitmap**—a bitmap that is not associated with a physical display device.

The section “Creating Bitmaps Offscreen,” which begins on page 5-45, shows how you can create an offscreen bitmap, draw shapes into it, and then draw it to the screen.

You can find more information about the QuickDraw GX view architecture in the chapter “View-Related Objects” in *Inside Macintosh: QuickDraw GX Objects*.

Using Bitmap Shapes

This section shows you how to create, edit, and draw bitmap shapes. In particular, this section shows you how to

- create and draw black-and-white bitmaps
- create and draw color bitmaps
- dither and halftone bitmaps
- apply transfer modes to bitmaps
- convert other types of shapes to bitmap shapes
- apply transformations to bitmaps
- create offscreen bitmaps
- edit sections of bitmaps

Bitmap shape geometries use a `gxPoint` structure to indicate the initial position of the bitmap. Since a `gxPoint` structure contains two fixed-point values (type `Fixed`), the sample functions in this section must convert integer constants to fixed-point constants when specifying bitmap positions. QuickDraw GX provides the `GXIntToFixed` macro to perform this conversion:

```
#define GXIntToFixed(a) ((Fixed)(a) << 16)
```

QuickDraw GX also provides the `ff` macro as a convenient alias:

```
#define ff(a) GXIntToFixed(a)
```

Creating and Drawing Bitmaps

QuickDraw GX provides a number of methods to create and draw bitmaps. For example, you can

- define a bitmap geometry and draw it without creating a bitmap shape
- define a bitmap geometry, encapsulate it in a bitmap shape, and draw the bitmap shape
- create another type of shape, convert it to a bitmap shape, perform any desired bitmap editing, and draw the bitmap shape
- create an offscreen bitmap, draw shapes to it, and then copy the offscreen bitmap to the screen
- unflatten a bitmap shape that was created earlier and stored to disk or that was created by another application
- convert a QuickDraw bitmap to a QuickDraw GX bitmap shape

The next section, “Creating Black-and-White Bitmaps,” and “Creating Color Bitmaps,” which begins on page 5-21, show you how to create bitmaps by specifying the bitmap geometry yourself.

The section “Converting Other Types of Shapes to Bitmaps,” which begins on page 5-34, shows you how you can create a bitmap shape containing a bitmap representation of other types of QuickDraw GX shapes.

The section “Creating Bitmaps Offscreen,” which begins on page 5-45, shows you how you can draw other shapes into the pixel image of a bitmap shape. You can use this method to create a bitmap representation of multiple QuickDraw GX shapes.

For information about flattening and unflattening bitmap shapes, see the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

Creating Black-and-White Bitmaps

You create a black-and-white bitmap by creating a bitmap shape with a pixel size of 1. To do this, you can define a pixel image, fill the fields of a bitmap geometry structure, and create a bitmap shape using the `GXNewBitmap` function.

Listing 5-1 shows a complete sample function that defines a black-and-white bitmap geometry, creates a bitmap shape, draws the shape, and disposes of it.

Listing 5-1 Creating a black-and-white bitmap

```
void CreateBlackAndWhiteBitmap(void)
{
    gxShape  aBitmapShape;

    gxBitmap aBitmapGeometry;
    gxPoint  initialPosition = {ff(20), ff(40)};
```

Bitmap Shapes

```

const char envelopeImage[] = {0x7F, 0xFF, 0xFF, 0xFE,
                               0xC0, 0x00, 0x00, 0x03,
                               0xB0, 0x00, 0x00, 0x0D,
                               0x8C, 0x00, 0x00, 0x31,
                               0x83, 0x00, 0x00, 0xC1,
                               0x80, 0xC0, 0x03, 0x01,
                               0x80, 0x30, 0x0C, 0x01,
                               0x80, 0x0C, 0x30, 0x01,
                               0x80, 0x33, 0xCC, 0x01,
                               0x80, 0xC0, 0x03, 0x01,
                               0x83, 0x00, 0x00, 0xC1,
                               0x8C, 0x00, 0x00, 0x31,
                               0xB0, 0x00, 0x00, 0x0D,
                               0x7F, 0xFF, 0xFF, 0xFE};

aBitmapGeometry.image = (char *) aSmallBitmapImage;

aBitmapGeometry.width = 32;      /* width in pixels */
aBitmapGeometry.height = 14;     /* height in pixels */
aBitmapGeometry.rowBytes = 4;    /* bytes per row */
aBitmapGeometry.pixelSize = 1;   /* bits per pixel */

/* QuickDraw GX creates a black-and-white color set for you */
aBitmapGeometry.space = gxNoSpace;
aBitmapGeometry.set = nil;
aBitmapGeometry.profile = nil;

aBitmapShape = GXNewBitmap(&aBitmapGeometry, &initialPosition);

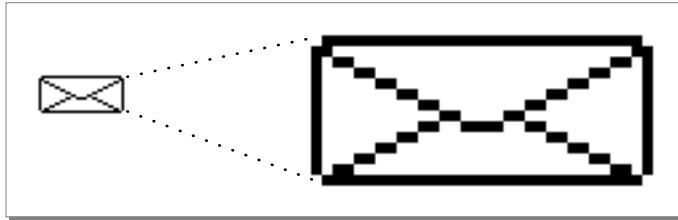
GXDrawShape(aBitmapShape);
GXDisposeShape(aBitmapShape);
}

```

Bitmap Shapes

The result of this function is shown in Figure 5-8.

Figure 5-8 A black-and-white bitmap—32 bits wide



The sample function from Listing 5-1 first defines a variable to hold the reference to the bitmap shape:

```
gxShape aBitmapShape;
```

Then the sample function defines two local variables to specify the bitmap geometry:

```
gxBitmap aBitmapGeometry;

gxPoint initialPosition = {ff(20), ff(40)};
```

The `initialPosition` variable, which is type `gxPoint`, contains the initial bitmap position, and the `aBitmapGeometry` variable, which is type `gxBitmap`, contains the rest of the information about the bitmap.

The sample function then defines the bitmap's pixel image:

```
const char envelopeImage[] = {0x7F, 0xFF, 0xFF, 0xFE,
                              0xC0, 0x00, 0x00, 0x03,
                              0xB0, 0x00, 0x00, 0x0D,
                              0x8C, 0x00, 0x00, 0x31,
                              0x83, 0x00, 0x00, 0xC1,
                              0x80, 0xC0, 0x03, 0x01,
                              0x80, 0x30, 0x0C, 0x01,
                              0x80, 0x0C, 0x30, 0x01,
                              0x80, 0x33, 0xCC, 0x01,
                              0x80, 0xC0, 0x03, 0x01,
                              0x83, 0x00, 0x00, 0xC1,
                              0x8C, 0x00, 0x00, 0x31,
                              0xB0, 0x00, 0x00, 0x0D,
                              0x7F, 0xFF, 0xFF, 0xFE};
```

Bitmap Shapes

The `envelopeImage` variable, which is defined as an array of bytes, contains a pixel image depicting a small envelope, as shown in Figure 5-8.

To create a bitmap shape encapsulating this envelope image, the sample function fills in the eight fields of the `aBitmapGeometry` variable. First, it sets the `image` field by casting the `envelopeImage` variable to the correct type:

```
aBitmapGeometry.image = (char *) envelopeImage;
```

Then the sample function fills in the bitmap dimensions. The bitmap is 32 pixels wide by 14 pixels high, and there are 4 bytes of information in each row of the pixel image:

```
aBitmapGeometry.width = 32;          /* width in pixels */
aBitmapGeometry.height = 14;         /* height in pixels */
aBitmapGeometry.rowBytes = 4;        /* bytes per row */
```

The sample function specifies the pixel size next. Since this bitmap is black-and-white, only one bit is needed to represent each pixel of the bitmap:

```
aBitmapGeometry.pixelSize = 1;       /* bits per pixel */
```

Finally, the sample function specifies color information. Since QuickDraw GX does not provide a black-and-white color space, this bitmap needs a black-and-white color set in which pixel values of 0 represent white pixels and pixel values of 1 represent black pixels. Setting the `pixelSize` field to 1 and the `space` field to `gxNoSpace` indicates that QuickDraw GX should create this black-and-white color set for you.

```
aBitmapGeometry.space = gxNoSpace;
aBitmapGeometry.set = nil;
aBitmapGeometry.profile = nil;
```

Setting the `space` field to the value `gxNoSpace` always indicates that QuickDraw GX should choose a color space for you. If the pixel size were large—for example, 16 or 32—QuickDraw GX would choose an RGB color space. However, since the pixel size is 1, no appropriate color space exists, so QuickDraw GX creates a grayscale color set. The pixel size determines the size of the color set created. In this case, a pixel size of 1 dictates that the color set have two entries—an white entry for a pixel value of 0 and a black entry for a pixel value of 1.

After you define a bitmap geometry, you could use the `GXDrawBitmap` function to cause QuickDraw GX to

- create a temporary bitmap shape (using the style, ink, and transform objects of the default bitmap shape)
- draw the bitmap
- dispose of the temporary bitmap shape

with this line of code:

```
GXDrawBitmap(&aBitmapGeometry, &initialPosition);
```


Bitmap Shapes

You should use the `GXDrawBitmap` function, however, only when you know in advance that you want to draw a bitmap only one time.

If you want to draw a bitmap more than once, you should encapsulate the bitmap geometry in a bitmap shape and then draw the bitmap shape. The sample function in Listing 5-1 uses this method:

```
aBitmapShape = GXNewBitmap(&aBitmapGeometry, &initialPosition);
GXDrawShape(aBitmapShape);
```

As with any type of QuickDraw GX shape, if you create a bitmap shape, you are responsible for disposing of it when you no longer need it. Listing 5-1 does this by calling

```
GXDisposeShape(aBitmapShape);
```

Notice that the envelope bitmap requires 4 bytes—an even number—to represent each row of the pixel image. However, to draw a similar envelope bitmap that includes two more rows of bits, as shown in Figure 5-10, the required number of bytes might seem to be 5 since 5 bytes contain 40 bits, more than enough needed to store the 34 bits per row in this image.

However, if you set the `rowBytes` field to 5:

```
aBitmapGeometry.rowBytes = 5;
```

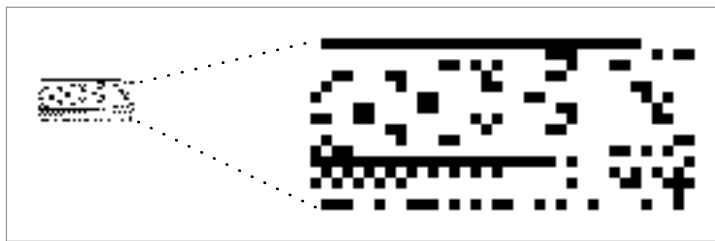
both the `GXDrawBitmap` function and the `GXNewBitmap` function post the error `bitmap_rowBytes_not_aligned`, because the value of the `rowBytes` field must be an even number.

Therefore, the value of the `rowBytes` field must be at least 6 for the bitmap of the envelope with a shadow. However, simply setting the `rowBytes` field to the value 6 with the assignment

```
aBitmapGeometry.rowBytes = 6;
```

results in the bitmap shown in Figure 5-9.

Figure 5-9 An example of unaligned bytes per row



Bitmap Shapes

Clearly, the value of the bitmap's `rowBytes` field is not aligned with the data in the bitmap's pixel image. If you set the value of the `rowBytes` field to 6, you must be sure to pad the pixel image so that each row actually contains 6 bytes of information. Listing 5-2 shows a new definition of the pixel image. In this definition, each row contains one extra byte so that the total number of bytes per row is even.

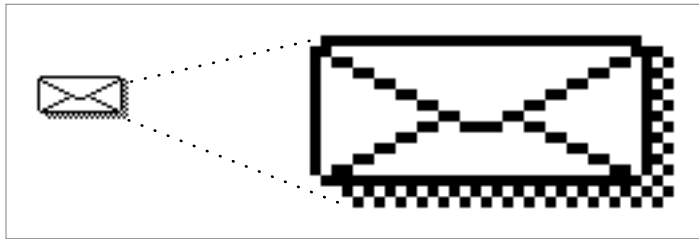
In this example, the extra bytes are initialized to the value 0x00. However, since these bytes are just padding, you can specify any values for them. As indicated by the bitmap width, QuickDraw GX ignores these extra bytes when drawing, hit-testing, or otherwise manipulating the bitmap.

Listing 5-2 A bit image with an even number of bytes per row

```
static char envelopeImage[] = {0x7F, 0xFF, 0xFF, 0xFE, 0x00, 0x00,
                               0xC0, 0x00, 0x00, 0x03, 0x50, 0x00,
                               0xB0, 0x00, 0x00, 0x0D, 0xA0, 0x00,
                               0x8C, 0x00, 0x00, 0x31, 0x50, 0x00,
                               0x83, 0x00, 0x00, 0xC1, 0xA0, 0x00,
                               0x80, 0xC0, 0x03, 0x01, 0x50, 0x00,
                               0x80, 0x30, 0x0C, 0x01, 0xA0, 0x00,
                               0x80, 0x0C, 0x30, 0x01, 0x50, 0x00,
                               0x80, 0x33, 0xCC, 0x01, 0xA0, 0x00,
                               0x80, 0xC0, 0x03, 0x01, 0x50, 0x00,
                               0x83, 0x00, 0x00, 0xC1, 0xA0, 0x00,
                               0x8C, 0x00, 0x00, 0x31, 0x50, 0x00,
                               0xB0, 0x00, 0x00, 0x0D, 0xA0, 0x00,
                               0x7F, 0xFF, 0xFF, 0xFE, 0x50, 0x00,
                               0x15, 0x55, 0x55, 0x55, 0xA0, 0x00,
                               0x0A, 0xAA, 0xAA, 0xAA, 0x80, 0x00};
```

With this new, padded definition of the pixel image, you can set `rowBytes` field to 6 so that the resulting bitmap appears as shown in Figure 5-10.

Figure 5-10 An envelope with a shadow



Bitmap Shapes

For a discussion of pixel images and bitmap geometries, see “Bitmap Geometries” beginning on page 5-5.

For more information about the `GXNewBitmap` function, see its description on page 5-66. For more information about the `GXDrawBitmap` function, see its description on page 5-79.

The next section shows you how you can create a bitmap with color information.

Creating Color Bitmaps

All QuickDraw GX bitmaps are actually color bitmaps. A black-and-white bitmap is simply a color bitmap with a color set containing only two colors—black and white.

The sample function in Listing 5-1 on page 5-15 creates a black-and-white bitmap geometry by

- specifying the pixel size to be 1
- specifying the color space to be the `gxNoSpace` color space

The sample function encapsulates the geometry into a bitmap shape with this call to the `GXNewBitmap` function:

```
aBitmapShape = GXNewBitmap(&aBitmapGeometry, &initialPosition);
```

Because the `space` field of the bitmap geometry specifies the `gxNoSpace` color space, the `GXNewBitmap` function chooses a color space for you, based on the pixel size specified in the `pixelSize` field. QuickDraw GX does not provide any color spaces appropriate for a pixel size of 1, so the `GXNewBitmap` function creates a grayscale color set with two entries—white and black.

If you specify the `gxNoSpace` color space with a pixel size of 2, the `GXNewBitmap` function creates a grayscale color set with four entries—white, light gray, dark gray, and black. After you change the pixel size to 2, you must reflect that change in the pixel image, the bitmap width, and the number of bytes per row.

Typically, if you wanted to make a 1 bit-per-pixel bitmap into a 2 bit-per-pixel bitmap, you would do the following

- Maintain the bitmap width, as it represents the number of pixel values—not the number of bits—per row of the bitmap
- Double the number of bytes per row to accommodate the extra bits
- Double the size of the pixel image, replacing 1-bit pixel values with 2-bit pixel values

This method allows you to maintain the size of the bitmap while allowing you to specify more possible values (colors) for each pixel.

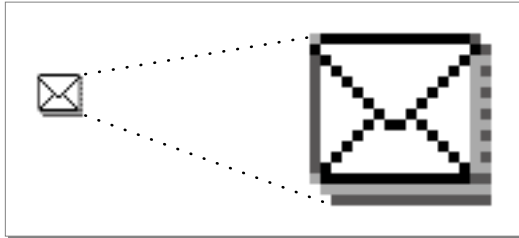
Bitmap Shapes

However, an easier (if somewhat less useful) way to make a 1 bit-per-pixel bitmap into a 2 bit-per-pixel bitmap is as follows:

- Divide the bitmap width in half
- Maintain the same number of bytes per row
- Maintain the same pixel image

Let's see what happens when you apply this simpler method for doubling the pixel size of a bitmap. Doubling the pixel size and halving the bitmap width of the envelope bitmap shown in Figure 5-10 on page 5-20 indicates that QuickDraw GX should interpret every pair of bits in the pixel image as a single pixel. Since each pixel can have one of four possible values (00, 01, 10, 11), the resulting bitmap contains four shades of gray, as shown in Figure 5-11.

Figure 5-11 A bitmap with a grayscale color set (four shades)



You can double the pixel size and halve the bitmap width again, with the following assignments:

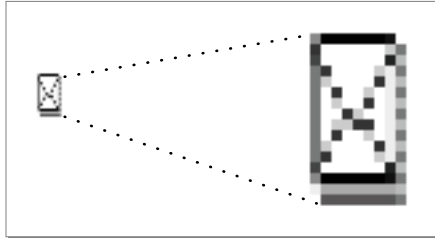
```
aBitmapGeometry.width = 9;
aBitmapGeometry.height = 16;
aBitmapGeometry.rowBytes = 6;
aBitmapGeometry.pixelSize = 4;
```

QuickDraw GX interprets each set of 4 bits in the pixel image as representing a single pixel of the bitmap, which means each pixel can now be represented by 16 different values (0000, 0001, 0010, and so on). Since QuickDraw GX has no predefined color space that uses a pixel size of 4, it creates for this bitmap a color set with sixteen shades of gray.

Bitmap Shapes

Figure 5-12 shows the resulting bitmap.

Figure 5-12 A bitmap with a grayscale color set (sixteen shades)



As the previous examples have shown, setting the `space` field of a bitmap geometry to the `gxNoSpace` constant indicates that you want QuickDraw GX to choose a color space for you. In these examples, which had 1, 2, or 4 bits per pixel, QuickDraw GX chose the `gxIndexedSpace` color space and created a grayscale color set with the appropriate number of color entries.

You are not limited to these grayscale color sets, however. You can create your own color set, by choosing your own set of colors for the color entries. Listing 5-3 shows how to define a simple color set with eight colors—black and white, the three primary RGB colors, and the three secondary RGB colors.

Listing 5-3 Defining a color set

```
gxColorSet aColorSet;

gxSetColor newColorList[] = {
    {0xFFFF, 0xFFFF, 0xFFFF, 0},    /* white */
    {0xFFFF, 0,      0,      0},    /* red */
    {0,      0xFFFF, 0,      0},    /* green */
    {0,      0,      0xFFFF, 0},    /* blue */
    {0,      0xFFFF, 0xFFFF, 0},    /* cyan */
    {0xFFFF, 0,      0xFFFF, 0},    /* magenta */
    {0xFFFF, 0xFFFF, 0,      0},    /* yellow */
    {0,      0,      0,      0},    /* black */
};
```

Bitmap Shapes

The colors in this color set are specified in the RGB color space, and each color contains four components—the red component, the green component, the blue component, and a fourth component, which QuickDraw GX ignores for the RGB color space.

QuickDraw GX allows you to specify colors in other color spaces and with different numbers of components. For complete color-specifying information, see the chapter “Colors and Color-Related Objects” in *Inside Macintosh: QuickDraw GX Objects*.

Once you’ve defined the color list for a color set, you create the actual color set object by using the `GXNewColorSet` function, which requires you to specify the color space in which you’ve specified the colors, the total number of colors, and the list of colors:

```
aColorSet = GXNewColorSet(gxRGBSpace, 8, newColorList);
```

Note

Remember, you are responsible for disposing of QuickDraw GX objects when you no longer need them, so you are responsible for disposing of this new color set. ♦

To use the new color set in your bitmap, you need to set the `space` and `set` fields of the bitmap geometry:

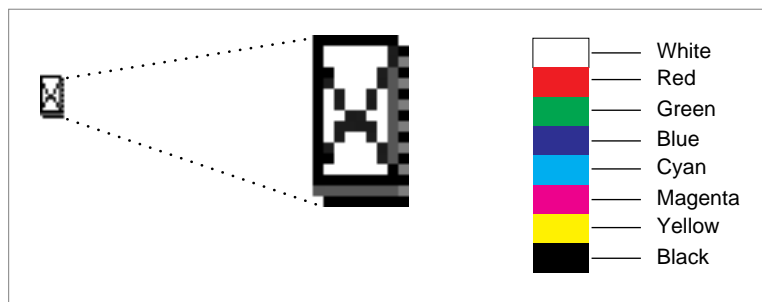
```
aBitmapGeometry.space = gxIndexedSpace;  
aBitmapGeometry.set = aColorSet;
```

Setting the `space` field to `gxIndexedSpace` indicates that you are supplying the color set, rather than having QuickDraw GX create one for you.

Figure 5-13 shows the result of applying this new color set to the 4-bits-per-pixel version of the envelope bitmap. Notice that pixel values in the pixel image greater than 7 are out of the range of the color set, so QuickDraw GX maps those pixel values to the color black.

For a color version of Figure 5-13, see Plate 3 at the front of this book.

Figure 5-13 A bitmap with an eight-color color set



Bitmap Shapes

Each of the previous examples in this chapter creates a bitmap that uses a color set. QuickDraw GX interprets each pixel of these bitmaps as an index into a set of colors. For example, in the black-and-white bitmap that results from Listing 5-1 on page 5-15, each pixel value (single bit) of the pixel image is an index into a color set with two colors—the index of the color white is 0 and the index of the color black is 1. In the 2 bits-per-pixel example on page 5-6, each pixel value (pair of bits) in the pixel image is an index into a color set with four colors—the index of white is 0 (bits 00), the index of light gray is 1 (bits 01), the index of dark gray is 2 (bits 10), and the index of black is 3 (bits 11).

QuickDraw GX also allows you to create bitmaps that use color spaces other than indexed color spaces (that is, other than color sets). In these bitmaps, each pixel value is an actual color value instead of an index into a list of colors. The chapter “Colors and Color-Related Objects” in *Inside Macintosh: QuickDraw GX Objects* explains color spaces, color values, color sets, and color indexes.

One example of a bitmap for which you might want to use a color space instead of a color set is a color ramp. A **color ramp** is a shape that blends from one color into another. Since bitmaps are the only type of QuickDraw GX shape (except picture shapes) that allows multiple colors in one shape, you must implement color ramps as bitmap shapes.

The sample function in Listing 5-4 on page 5-26 shows how to create a simple color ramp. This function declares a bitmap shape reference and a bitmap geometry structure using the declarations

```
gxShape  aBitmapShape;
gxBitmap aBitmapGeometry;
```

It then fills in the fields of the bitmap geometry structure. First, it fills in the dimensions:

```
aBitmapGeometry.width = 1;
aBitmapGeometry.height = 256;
aBitmapGeometry.rowBytes = 1;
aBitmapGeometry.pixelSize = 32;
```

Notice that the sample function defines the bitmap width to be 1. Later, the sample function uses the `GXSetShapeBounds` function later to widen the bitmap.

Next, the sample function sets the `image` field to `nil` to indicate that QuickDraw GX should allocate memory for the pixel image of the bitmap. The value of the `rowBytes` field is ignored because QuickDraw GX sets this field when allocating the pixel image.

The sample function then sets the color-related fields of the bitmap geometry structure:

```
aBitmapGeometry.space = gxRGB32Space;
aBitmapGeometry.set = nil;
aBitmapGeometry.profile = nil;
```

Notice that the pixel size implied by the color space (which is the `gxRGB32Space` color space) is the same as the pixel size indicated in the `pixelSize` field of the bitmap geometry structure (which is 32).

Bitmap Shapes

Next, the sample function creates the bitmap shape:

```
aBitmapShape = GXNewBitmap(&aBitmapGeometry, &initialPosition);
```

The sample function sets the color values of each pixel in the bitmap shape. To do this, it creates a color structure with the declaration

```
gxColor current;
```

Then it fills in the values of the fields of the color structure:

```
current.space = gxRGBSpace;
current.profile = nil;
current.element.rgb.red = 0xFFFF;
current.element.rgb.green = 0;
current.element.rgb.blue = 0;
current.element.rgb.alpha = 0;
```

For a complete discussion of these fields, see the chapter “Colors and Color-Related Objects” in *Inside Macintosh: QuickDraw GX Objects*.

The sample function then uses the `GXSetShapePixel` function to set each pixel value in the pixel image of the bitmap shape. Each time the sample function sets the value of a pixel, it changes the color value of the `current` variable slightly, decreasing the amount of green and increasing the amount of red:

```
for (count = 0; count < 256; count++) {
    current.element.component[0] -= 0x0101; /* decrease red */
    current.element.component[1] += 0x0101; /* increase green */

    GXSetShapePixel(aBitmapShape, 0, count, &current, 0);
}
```

Finally, the sample function resizes the bitmap, widening it to be a square, and draws the resulting bitmap color ramp. The complete sample function definition is shown in Listing 5-4.

Listing 5-4 Creating a color ramp

```
void CreateColorRamp(void)
{
    gxShape aBitmapShape;
    gxBitmap aBitmapGeometry;
    const gxPoint initialLocation = {ff(50), ff(50)};
    const gxRectangle theBounds = {ff(50), ff(50),
                                   ff(150), ff(150)};
```


Bitmap Shapes

```

gxColor current;
int count;

/* create a one-pixel-wide bitmap */
aBitmapGeometry.width = 1;
aBitmapGeometry.height = 256;
aBitmapGeometry.rowBytes = 1;
aBitmapGeometry.pixelSize = 32;

aBitmapGeometry.image = nil; /* have QuickDraw GX allocate */

aBitmapGeometry.space = gxRGB32Space;
aBitmapGeometry.set = nil;
aBitmapGeometry.profile = nil;

aBitmapShape = GXNewBitmap(&aBitmapGeometry, &initialLocation);

/* create a red color */
current.space = gxRGBSpace;
current.profile = nil;
current.element.component[0] = 0xFFFF; /* red */
current.element.component[1] = 0;      /* green */
current.element.component[2] = 0;      /* blue */
current.element.component[3] = 0;      /* alpha */

/* fill in the colors of the bitmap pixel by pixel */
for (count = 0; count < 256; count++) {
    current.element.rgb.red -= 0x0101; /* decrease red */
    current.element.rgb.green += 0x0101; /* increase green */

    GXSetShapePixel(aBitmapShape, 0, count, &current, 0);
}

/* resize the bitmap to give it more width */
GXSetShapeBounds(aBitmapShape, &theBounds);

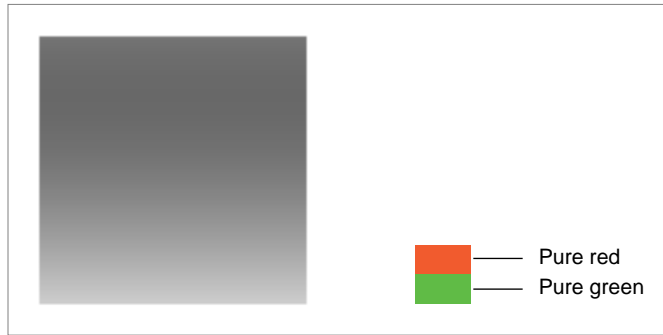
GXDrawShape(aBitmapShape);
GXDisposeShape(aBitmapShape);
}

```

Bitmap Shapes

The resulting color ramp is shown in Figure 5-14. For a color version of this figure, see Plate 4 at the front of this book.

Figure 5-14 A color ramp from red to green



QuickDraw GX provides the ramp library to assist you in creating color ramps. The `NewRamp` library function requires you to provide a start color, an end color, an integer indicating the number of different colors to calculate in between the start color and end color, and a bounding rectangle for the final color ramp. Listing 5-5 shows how to use this function to create the same color ramp shown in Figure 5-14.

Listing 5-5 Creating a color ramp using the ramp library

```
void CreateColorRamp(void)
{
    gxShape aBitmapShape;
    gxColor start, end;
    const gxRectangle theBounds = {ff(50), ff(50),
                                    ff(150), ff(150)};

    start.space = gxRGBSpace;
    start.profile = nil;
    start.element.rgb.red = 0xFFFF;
    start.element.rgb.green = 0;
    start.element.rgb.blue = 0;
    start.element.rgb.alpha = 0;

    end.space = gxRGBSpace;
    end.profile = nil;
    end.element.rgb.red = 0;
    end.element.rgb.green = 0xFFFF;
```

Bitmap Shapes

```

    end.element.rgb.blue = 0;
    end.element.rgb.alpha = 0;

    aBitmapShape = NewRamp(&start, &end, 256, &theBounds);
    GXDrawShape(aBitmapShape);
    GXDisposeShape(aBitmapShape);
}

```

As a further convenience, QuickDraw GX provides the color library, which allows you to use predefined constants to specify frequently used colors. You provide the `SetCommonColor` library function with a pointer to a color structure, and a predefined constant specifying the color you want.

This function then initializes the color structure with the appropriate values to represent the color you specify.

For example, the following call sets the fields of the `start` color structure with the values that represent the color red:

```
SetCommonColor(&start, red);
```

Listing 5-6 shows you how to create the color ramp in Figure 5-14 by using functions from both the ramp and color libraries.

Listing 5-6 Creating a color ramp using both the ramp and color libraries

```

void CreateColorRamp(void)
{
    gxShape  aBitmapShape;
    gxColor start, end;
    const gxRectangle theBounds = {ff(50), ff(50),
                                   ff(150), ff(150)};

    SetCommonColor(&start, red);
    SetCommonColor(&end, green);

    aBitmapShape = NewRamp(&start, &end, 0, &theBounds);
    GXDrawShape(aBitmapShape);
    GXDisposeShape(aBitmapShape);
}

```

For a discussion of pixel images and bitmap geometries, see “Bitmap Geometries” beginning on page 5-5.

You can find more information about colors, color structures, color values, color sets, and color spaces in the chapter “Colors and Color-Related Objects” in *Inside Macintosh: QuickDraw GX Objects*.

Dithering and Halftoning Bitmaps

The color ramp created in the previous section uses the `gxRGB32Space` color space, but not all display devices can display 32 bits of color. To optimize the appearance of color on displays with limited numbers of colors, QuickDraw GX allows you to **dither** shapes—that is, approximate colors that a display device cannot draw, with patterns of similar colors that the display device can draw.

The chapter “View-Related Objects” in *Inside Macintosh: QuickDraw GX Objects* describes dithering in detail.

This section shows how you can use dithering to draw the color ramp shown in Figure 5-14 on page 5-28.

Since dithering is a function of view port objects, you must first determine the view port to which the color ramp is drawn. Since this color ramp is only being drawn to one view port, you can declare an array to hold a single view port reference:

```
gxViewPort aViewPortList[1];
```

Then you can use the `GXGetShapeGlobalViewPorts` function to copy the view port list from the transform object of the color ramp bitmap shape into the view port array:

```
GXGetShapeGlobalViewPorts(aColorRampBitmapShape, aViewPortList);
```

If the color ramp were being drawn to multiple view ports, you would call this function once specifying `nil` for the view port array to determine the number of view ports, then allocate space to hold the view port references, and then call the function a second time to determine the actual view port references.

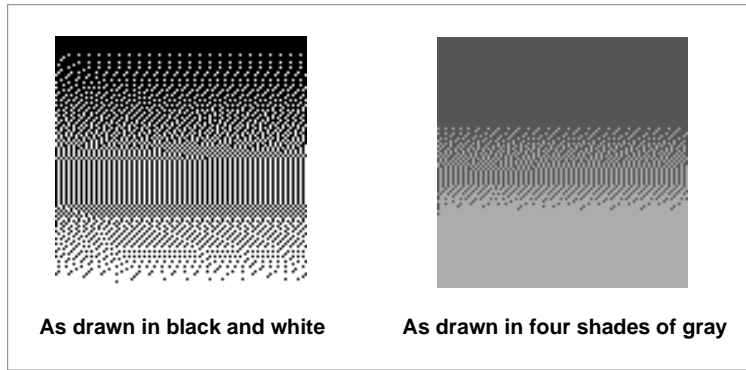
In the color ramp example, you can use the `GXSetViewPortDither` function to indicate that shapes drawn to this view port should be dithered. This function takes two parameters: a reference to the view port and a dither level, which is described in detail in the chapter “View-Related Objects” in *Inside Macintosh: QuickDraw GX Objects*. If a view port has a dither level of 2 or greater, QuickDraw GX dithers bitmaps drawn to that view port:

```
GXSetViewPortDither(aViewPortList[0], 4); /* Dither bitmaps */
```

Bitmap Shapes

Figure 5-15 shows how QuickDraw GX draws the dithered color ramp to display devices at two different pixel depths.

Figure 5-15 Dithered bitmaps



Halftoning, which is also described in the chapter “View-Related Objects” in *Inside Macintosh: QuickDraw GX Objects*, is similar to dithering. To specify halftoning for a view port, you need to create a `gxHalftone` structure. This structure specifies information about how QuickDraw GX should halftone shapes drawn to the view port. Listing 5-7 shows how to create a sample `gxHalftone` structure and set the halftone characteristics for the view port of the color ramp bitmap.

Listing 5-7 Halftoning a bitmap

```
gxHalftone aHalftone;

SetCommonColor(&halftoneDots, gxBlack);
SetCommonColor(&halftoneBackground, gxWhite);

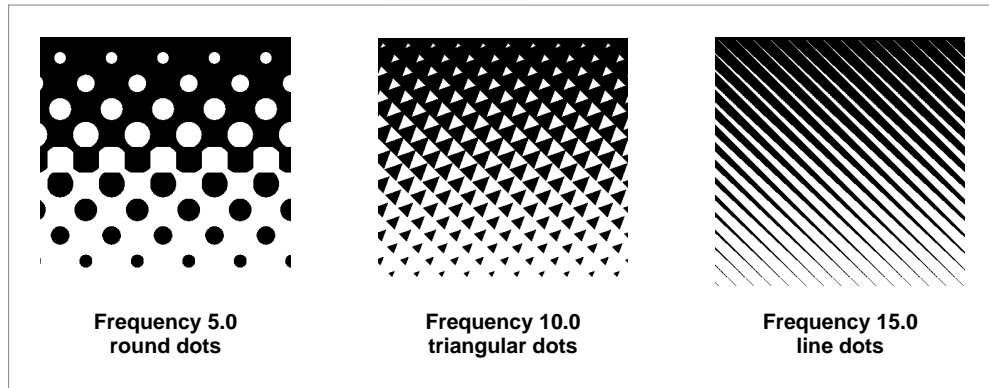
aHalftone.angle = ff(45);
aHalftone.frequency = ff(5);
aHalftone.method = gxRoundDot;
aHalftone.tinting = gxComponent1Tint;
aHalftone.dotColor = halftoneDots;
aHalftone.backgroundColor = halftoneBackground;
aHalftone.tintSpace = gxRGBSpace;

GXGetShapeGlobalViewPorts(aBitmapShape, aViewPort);
GXSetViewPortHalftone(aViewPort[0], &aHalftone);
```

Bitmap Shapes

Figure 5-16 shows three possible results of halftoning the color ramp bitmap. The first example is the result of Listing 5-7—round dots and a dot frequency of 5. The other two examples show the result of halftoning the color ramp bitmap with other dot frequencies and dot shapes.

Figure 5-16 Halftoned bitmaps



Applying Transfer Modes to Bitmaps

When drawing a bitmap, QuickDraw GX uses the color information stored in the geometry of the bitmap shape; it ignores the color information stored in the ink object associated with the bitmap shape.

However, QuickDraw GX does consider the transfer mode information specified in a bitmap shape's ink object. QuickDraw GX uses the transfer mode when drawing each pixel of a bitmap.

Bitmap Shapes

As an example, the sample function in Listing 5-8 creates a rectangle shape containing a purple rectangle and a bitmap shape containing a color ramp from red to green (as defined in Listing 5-6 on page 5-29).

Listing 5-8 Applying a transfer mode to a bitmap

```
void ApplyTransferModeToBitmap(void)
{
    gxShape aRectangleShape, aBitmapShape;
    const gxRectangle theRectangleBounds = {ff(100), ff(100),
                                            ff(200), ff(200)};

    const gxRectangle theBitmapBounds = {ff(50), ff(50),
                                         ff(150), ff(150)};

    gxColor start, end;

    aRectangleShape = GXNewRectangle(&theRectangleBounds);
    SetShapeCommonColor(aRectangleShape, purple);

    SetCommonColor(&start, red);
    SetCommonColor(&end, green);
    aBitmapShape = NewRamp(&start, &end, 0, &theBitmapBounds);
    SetShapeCommonTransfer(aBitmapShape, gxBlendMode);

    GXDrawShape(aRectangleShape);
    GXDrawShape(aBitmapShape);

    GXDisposeShape(aRectangleShape);
    GXDisposeShape(aBitmapShape);
}
```

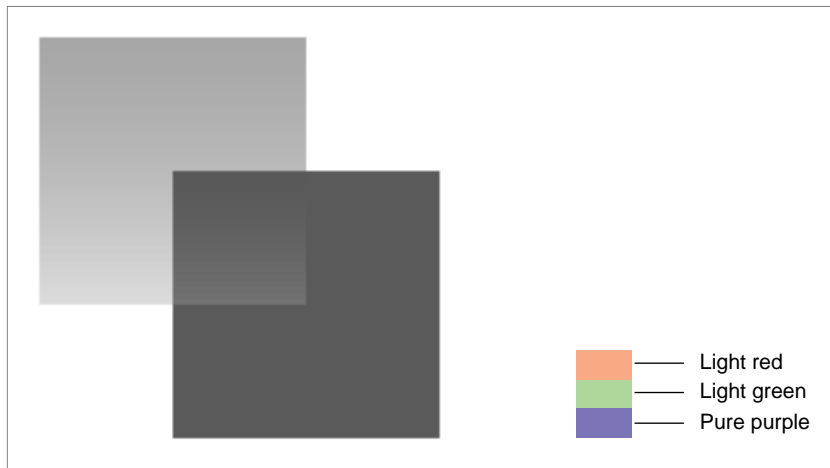
Bitmap Shapes

The sample function then uses the transfer mode library function `SetShapeCommonTransfer` to set the transfer mode of the bitmap shape to `gxBlendMode`.

Finally, the sample function draws the purple rectangle and the bitmap. Since the ink object associated with the bitmap specifies the `gxBlendMode` transfer mode, QuickDraw GX applies this transfer mode when drawing each pixel of the bitmap. Pixels that fall over the white background are blended with white, and pixels that fall over the purple rectangle are blended with purple.

Figure 5-17 shows the result of this sample function. For a color version of this figure, see Plate 2 at the front of this book.

Figure 5-17 A blended color ramp



You can find more information about transfer modes in the chapter “Ink Objects” in *Inside Macintosh: QuickDraw GX Objects*.

Converting Other Types of Shapes to Bitmaps

The examples in the previous sections show you how to create a bitmap shape by specifying the value of every pixel in the pixel image yourself. You can also create bitmaps using one of a number of simpler methods. For example, you can convert any QuickDraw GX shape to a bitmap shape. The pixel image of the resulting bitmap shape contains a bitmap representation of the original shape. (In a similar way, when you draw a shape to a display device, the display device displays a bitmap representation of the original shape.)

Bitmap Shapes

To convert another type of shape into a bitmap shape, you use the `GXSetShapeType` function, which is described in detail in the chapter “Shape Objects” of *Inside Macintosh: QuickDraw GX Objects*.

Listing 5-9 shows a sample function that defines a figure-eight geometry, encapsulates the geometry in a path shape, sets the pen width of that path to 10, and skews the path around its center by 10% along both the horizontal and vertical axes. Then the sample function converts the path shape into a bitmap shape and draws the bitmap.

Listing 5-9 Converting a path to a bitmap

```
void ConvertPathToBitmap(void)
{
    gxShape pathToBitmapShape;

    gxRectangle theBounds;

    const long figureEightGeometry[] = {1, /* number of contours */
                                         4, /* number of points */
                                         0xF0000000, /* 1111 ... */
                                         ff(20), ff(20), /* off */
                                         ff(100), ff(100), /* off */
                                         ff(20), ff(100), /* off */
                                         ff(100), ff(20)}; /* off */

    pathToBitmapShape = GXNewPaths((gxPaths *) figureEightGeometry);
    GXSetShapeFill(pathToBitmapShape, gxClosedFrameFill);
    GXSetShapePen(pathToBitmapShape, ff(10));
    GXSkewShape(pathToBitmapShape, fl(.1), fl(.1), ff(60), ff(60));

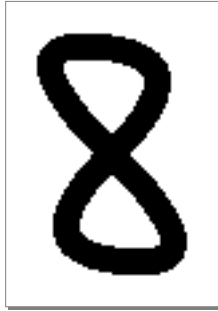
    GXSetShapeType(pathToBitmapShape, gxBitmapType);
    GXDrawShape(pathToBitmapShape);
    GXDisposeShape(pathToBitmapShape);
}
```

Listing 5-9 uses the `GXSkewShape` function, which is described fully in the chapter “Transform Objects” in *Inside Macintosh: QuickDraw GX Objects*.

Bitmap Shapes

Figure 5-18 shows the result of this function.

Figure 5-18 A bitmap representation of a path shape



Notice that QuickDraw GX draws the bitmap at 72 pixels per inch.

When converting shapes to bitmap shapes, QuickDraw GX creates a bitmap shape with the smallest pixel image possible to contain the bitmap representation of the original shape. To illustrate, you can draw the bounding rectangle of the skewed figure-eight bitmap by adding to Listing 5-9 the declaration

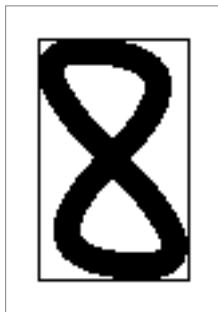
```
gxRectangle theBounds;
```

and these two lines of code:

```
GXGetShapeBounds(pathToBitmapShape, 0, &theBounds);  
GXDrawRectangle(&theBounds, gxClosedFrameFill);
```

The resulting bitmap and bounding rectangle are shown in Figure 5-19.

Figure 5-19 A bitmap and its bounding rectangle



Bitmap Shapes

When QuickDraw GX converts other types of shapes into a bitmap shape, it creates a new bitmap geometry and draws the original shape into the bitmap's pixel image. If the original shape does not cover all of the pixels in the bitmap's pixel image, QuickDraw GX sets the color value of the extra pixels to white. These white pixels may produce unexpected results if you draw the bitmap over a background that includes colors other than white.

For example, the following code adds a background shape to the sample function in Listing 5-9:

```
gxShape backgroundShape;
const gxRectangle backgroundBounds = {ff(20), ff(10),
                                       ff(100), ff(110)};

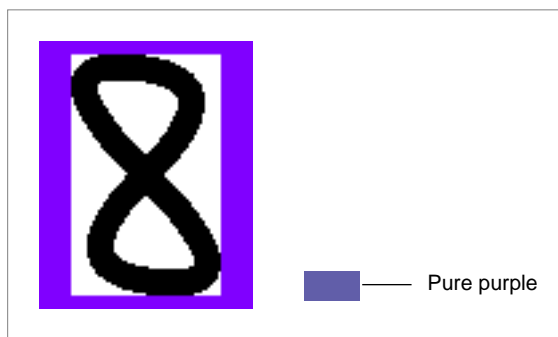
backgroundShape = GXNewRectangle(&backgroundBounds);
SetShapeCommonColor(backgroundShape, purple);
```

If you draw the background before the bitmap, the white pixels of the bitmap cover the corresponding area of the purple rectangle:

```
GXDrawShape(backgroundShape);
GXDrawShape(pathToBitmapShape);
```

The result appears as shown in Figure 5-20. For a color version of this figure, see Plate 6 at the front of this book.

Figure 5-20 A bitmap drawn over a background



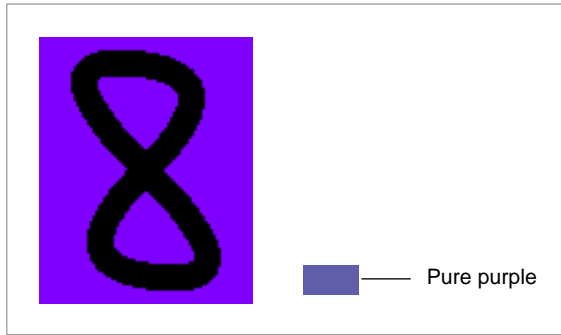
You can set the transfer mode of the bitmap shape to allow the purple to show through the white pixels. For example, you can set the transfer mode of the bitmap to the `gxMinimumMode` transfer mode using this code:

```
SetShapeCommonTransfer(pathToBitmapShape, gxMinimumMode);
GXDrawShape(pathToBitmapShape);
```

Bitmap Shapes

The result is shown in Figure 5-21. For a color version of this figure, see Plate 6 at the front of this book.

Figure 5-21 A bitmap with a transfer mode drawn over a background



Another way to allow the purple rectangle to show through the white areas of this bitmap is to set the clip shape of the bitmap. The next section, “Applying Transformations to Bitmaps,” shows an example of clipping a bitmap.

The examples in this section use colors and the `SetCommonColor` library function, which are available in the color library, and transfer modes and the `SetShapeCommonTransfer` library function, which are available in the transfer mode library.

For more information about the `GXSetShapeType` function, see the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

For information about combining multiple QuickDraw GX shapes into a single bitmap shape, see “Creating Bitmaps Offscreen,” which begins on page 5-45.

Applying Transformations to Bitmaps

Although bitmap shapes make limited use of their style and ink objects, they make full use of their transform objects. The examples in this section show how you can use the transform object of a bitmap to affect the drawing of that bitmap. The first few sample functions illustrate mapping transformations, and the last sample function illustrates clipping.

Mapping Bitmap Shapes

Since a bitmap geometry contains a pixel image rather than a geometric description, applying mapping transformations to bitmap shapes does not produce the same quality results as applying mapping transformations to geometric shapes. To use as an example, Figure 5-22 shows the path shape converted to a bitmap in Listing 5-9 on page 5-35.

Figure 5-22 A path shape converted to a bitmap shape



You can call the `GXSkewShape` function to undo the skewing of the figure-eight shape:

```
GXSkewShape(pathToBitmapShape, -fl(.1), -fl(.1), ff(60), ff(60));
```

Figure 5-23 shows the results of performing this transformation on the figure-eight bitmap shape.

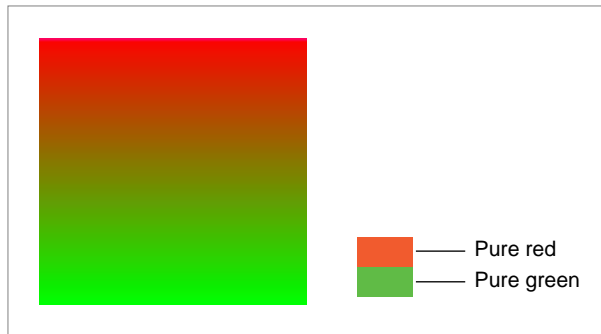
Figure 5-23 A path shape converted to a bitmap shape and then skewed



Bitmap Shapes

As Figure 5-23 shows, the quality of the transformed bitmap has degraded due to the skewing. If the `gxMapTransformShape` shape attribute of the bitmap shape is not set, this degradation of quality becomes more pronounced with multiple transformations. For example, consider the color ramp depicted in Figure 5-24. For a color version of this figure, see Plate 4 at the front of this book.

Figure 5-24 A color ramp bitmap



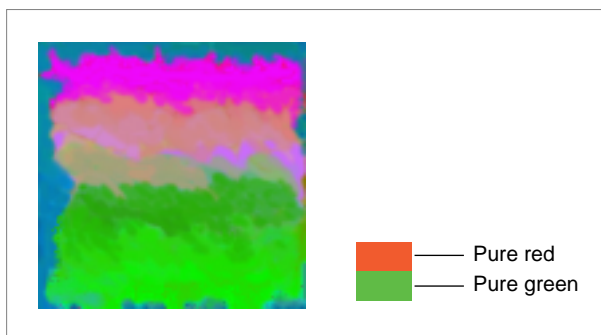
The following lines of code clear the `gxMapTransformShape` shape attribute for this bitmap shape and then rotate the shape 360 times by 1 degree each time:

```
GXSetShapeAttributes(aColorRampBitmapShape,
                    GXGetShapeAttributes(aColorRampBitmapShape)
                    & ~gxMapTransformShape);

for (count = 1; count <= 360; count++)
    GXRotateShape(aColorRampBitmapShape, ff(1), ff(100), ff(100));
```

Enough error is introduced to create an interesting new bitmap, as shown in Figure 5-25. For a color version of this figure, see Plate 5 at the front of this book.

Figure 5-25 A bitmap after multiple transformations



Bitmap Shapes

However, if you leave the `gxMapTransformShape` shape attribute set, you can apply the same 360 transformations, and the resulting bitmap is identical to the original bitmap. In this case, all of the transformations affect the mapping matrix stored in the bitmap's transform object and not the pixel values of the bitmap's pixel image.

Scaling text provides another example of transformations degrading the quality with which QuickDraw GX draws a shape. As an example, the sample function in Listing 5-10 creates a text shape, draws it, scales it up, and then draws the scaled version. This sample function uses the `GXScaleShape` function, which is described in the chapter "Transform Objects" of *Inside Macintosh: QuickDraw GX Objects*.

Listing 5-10 Scaling text

```
void ScaleText(void)
{
    gxShape  aTextShape;
    const gxPoint initialLocation = {ff(50), ff(50)};

    aTextShape = GXNewText(9, (unsigned char *) "123456789",
                          &initialLocation) ;
    GXDrawShape(aTextShape);

    GXScaleShape(aTextShape, ff(3), ff(3), ff(0), ff(50));

    GXDrawShape(aTextShape);
    GXDisposeShape(aTextShape);
}
```

The result is shown in Figure 5-26.

Figure 5-26 Scaled text


Bitmap Shapes

If you convert the text shape to a bitmap shape before scaling it, as in the sample function in Listing 5-11, the result is quite different.

Listing 5-11 Scaling a bitmap

```
void ScalingABitmap(void)
{
    gxShape aBitmapShape;
    gxPoint initialLocation = {ff(50), ff(50)};

    aBitmapShape = GXNewText(9, (unsigned char *) "123456789",
                             &initialLocation) ;

    GXSetShapeType(aBitmapShape, gxBitmapType);

    GXDrawShape(aBitmapShape);

    GXScaleShape(aBitmapShape, ff(3), ff(3), ff(0), ff(50));

    GXDrawShape(aBitmapShape);
    GXDisposeShape(aBitmapShape);
}
```

Figure 5-27 compares the result of scaling the text shape with the result of scaling the bitmap shape.

Figure 5-27 Scaled text and a scaled bitmap


Bitmap Shapes

When scaling the text, QuickDraw GX uses the outline information in the font to draw the best representation of the text at the appropriate size. When scaling the bitmap representation of the text, QuickDraw GX simply scales the bits used to represent the smaller version of the text.

For more information about text shapes, see *Inside Macintosh: QuickDraw GX Typography*.

Clipping Bitmap Shapes

You can use the transform object of a bitmap shape to clip the bitmap—that is, restrict the area where QuickDraw GX draws the bitmap.

As an example, to apply a circular clip to the color ramp from Figure 5-24 on page 5-40, you start by defining the circular geometry and encapsulating it in a path shape:

```
long theClipGeometry[] = {1, 4, 0xF0000000,
                          ff(50), ff(50),
                          ff(150), ff(50),
                          ff(150), ff(150),
                          ff(50), ff(150)};

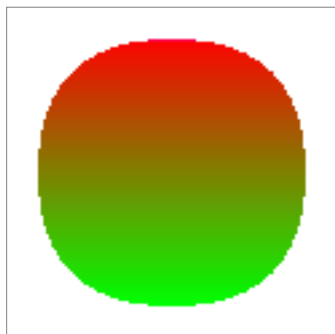
aClipShape = GXNewPaths((gxPaths *) theClipGeometry);
```

Then set the clip property of the bitmap's transform object by using this call to the `GXSetShapeClip` function:

```
GXSetShapeClip(aColorRampBitmapShape, aClipShape);
```

QuickDraw GX draws the resulting bitmap shape as shown in Figure 5-28. For a color version of this figure, see Plate 5 at the front of this book.

Figure 5-28 A clipped bitmap



For more information about transform objects, mapping transformations, clip shapes, and the `GXSetShapeClip` function, see the chapter “Transform Objects” in *Inside Macintosh: QuickDraw GX Objects*.

Creating Bitmaps With Disk-Based Pixel Images

QuickDraw GX allows you to store the pixel image of a bitmap shapes in a disk file. To create this type of bitmap, you specify a predefined constant for the image field of the bitmap's geometry:

```
aBitmapGeometry.image = gxBitmapFileAliasImageValue;
```

The other fields of the geometry you can initialize as you would for other bitmaps:

```
aBitmapGeometry.width = widthOfDiskBasedImage;
aBitmapGeometry.height = heightOfDiskBasedImage;
aBitmapGeometry.rowBytes = rowBytesOfDiskBasedImage;
aBitmapGeometry.pixelSize = pixelSizeOfDiskBasedImage;

aBitmapGeometry.space = colorSpaceOfDiskBasedImage;
aBitmapGeometry.set = colorSetOfDiskBasedImage;
aBitmapGeometry.profile = colorProfileOfDiskBasedImage;
```

You still create the bitmap using the `GXNewBitmap` function:

```
aBitmapShape = GXNewBitmap(&aBitmapGeometry, &initialLocation);
```

You specify the file that contains the pixel image using the bitmap data source alias structure, which is defined by the `gxBitmapDataSourceAlias` data type:

```
struct gxBitmapDataSourceAlias {
    unsigned long fileOffset; /* offset (in bytes) to image */
    unsigned long aliasRecordSize; /* size of alias record */
    unsigned char aliasRecord[gxAnyNumber]; /* alias record */
};
```

To use this data type, you need to declare a variable to hold the structure:

```
gxBitmapDataSourceAlias anAlias;
```

Then, you need to set the three fields of the structure:

- the `aliasRecord` field should contain a Macintosh Alias Manager alias record specifying the file containing the pixel image
- the `aliasRecordSize` field should specify the size in bytes of the alias record
- the `fileOffset` field should specify the offset in bytes from the beginning of the data fork of the file to the first pixel value of the pixel image

Bitmap Shapes

Once you've created the bitmap data source alias structure, you create a tag object to encapsulate the structure, using the call

```
anAliasTag = GXNewTag(gxBitmapFileAliasTagType, sizeof(anAlias)
                    &anAlias);
```

Then you associate the tag object with the bitmap shape using the call

```
GXSetShapeTags(aBitmapShape, gxBitmapFileAliasTagType,
               1, /* first tag */
               -1, /* replace all tags of same type */
               1, /* insert one new tag */
               &anAliasTag); /* tag to insert */
```

Now the disk-based bitmap is completely initialized. You can use most bitmap-related functions with this bitmap, but there are bitmap-related functions you cannot use. In particular, you cannot call the `GXSetBitmapParts`, `GXSetShapePixel`, `GXNewViewDevice`, or `GXSetViewDeviceBitmap` functions, as these functions would require QuickDraw GX to write to the file.

For more information about alias records, see the chapter “Alias Manager” of *Inside Macintosh: Files*.

For more information about tags and the `GXNewTag` function, see the chapter “Tag Objects” of *Inside Macintosh: QuickDraw GX Objects*. For information about the `GXSetShapeTags` function, see the chapter “Shape Objects” in that book.

Creating Bitmaps Offscreen

The section “Converting Other Types of Shapes to Bitmaps” beginning on page 5-34 describes how you can convert a single QuickDraw GX shape to a bitmap shape. This section shows you how to draw multiple QuickDraw GX shapes to a single bitmap shape.

When you draw a shape, QuickDraw GX does the following:

- examines the shape's transform object, which contains a view port list
- examines the view ports in this list, each of which belongs to a view group
- examines these view groups, which contain view devices
- decides which view devices the shape actually intersects
- examines these view devices, each of which contains a bitmap
- renders the shape into these bitmaps

Bitmap Shapes

Therefore, to draw shapes into an offscreen bitmap, you need to

- create a bitmap shape to contain the rendered shapes
- create a view group to contain a view device
- create a view device to contain the bitmap shape
- create a view port that belongs to the view group
- create a transform to reference the view port
- associate the transform with the shapes you want to draw offscreen
- clear the offscreen bitmap
- draw the shapes

You can find complete information about transforms, view devices, view groups, and view ports in the chapters “Transform Objects” and “View-Related Objects” in *Inside Macintosh: QuickDraw GX Objects*.

To create the offscreen bitmap, you must define a shape reference for the bitmap shape and create a bitmap shape of the appropriate size:

```
gxShape  aBitmapShape;

aBitmapShape = CreateABitmap(200, 200);
```

Listing 5-12 shows a possible definition for the `CreateABitmap` function. This function creates a black-and-white bitmap of a specified height and width.

Listing 5-12 Creating a black-and-white bitmap

```
static gxShape CreateABitmap(long height, long width)
{
    gxShape aBitmapShape;
    gxBitmap aBitmapGeometry;
    const gxPoint initialLocation = {ff(0), ff(0)};

    aBitmapGeometry.image = nil;

    aBitmapGeometry.width = width;
    aBitmapGeometry.height = height;
    aBitmapGeometry.rowBytes = 0;
    aBitmapGeometry.pixelSize = 1;

    aBitmapGeometry.space = gxNoSpace;
    aBitmapGeometry.set = nil;
    aBitmapGeometry.profile = nil;
```

Bitmap Shapes

```

    aShape = GXNewBitmap(&aBitmapGeometry, &initialLocation);

    return(aBitmapShape);
}

```

To create the offscreen view device, view group, and view port objects, you must declare references to them:

```

gxViewGroup offscreenViewGroup;
gxViewDevice offscreenViewDevice;
gxViewPort offscreenViewPort;

```

You create the view group object first:

```
offscreenViewGroup = GXNewViewGroup();
```

Then you can create the view device and view port objects. To create a view device, you must specify both the view group it belongs to and the bitmap it uses when rendering shapes:

```

offscreenViewDevice = GXNewViewDevice(offscreenViewGroup,
                                       aBitmapShape);

```

To create a view port, you need only specify the view group to which it belongs:

```
offscreenViewPort = GXNewViewPort(offscreenViewGroup);
```

To draw shapes to this offscreen view port, you need to create a new transform object. First, you must declare a reference to a transform object:

```
gxTransform offscreenTransform;
```

Then you can create it and set its view port list to contain the offscreen view port:

```

offscreenTransform = GXNewTransform();
GXSetTransformViewPorts(offscreenTransform, 1,
                        &offscreenViewPort);

```

Now you're ready to draw shapes offscreen. The first shape that you draw is a simple white rectangle, and drawing it initializes the pixels in the offscreen bitmap:

```

gxShape aRectangleShape;
gxRectangle boundsRectangle = {ff(0), ff(0), ff(200), ff(200)};

aRectangleShape = GXNewRectangle(&boundsRectangle);

SetShapeCommonColor(aRectangleShape, gxWhite);

```

Bitmap Shapes

To draw this white rectangle to the offscreen bitmap, you must set its transform object to be the offscreen transform object:

```
GXSetShapeTransform(aRectangleShape, offscreenTransform);
```

Then you draw and dispose of the shape:

```
GXDrawShape(aRectangleShape);
GXDisposeShape(aRectangleShape);
```

Since the rectangle shape references the offscreen transform object, QuickDraw GX draws the white rectangle into the offscreen bitmap.

Because the offscreen bitmap is now initialized, you can draw other shapes to it. The following code demonstrates how to create a line shape and draw it to the offscreen bitmap:

```
gxShape aLineShape;
gxLine lineGeometry = {ff(40), ff(40), ff(160), ff(160)};

aLineShape = GXNewLine(&lineGeometry);
GXSetShapePen(aLineShape, ff(50));
GXSetShapeTransform(aLineShape, offscreenTransform);
GXDrawShape(aLineShape);
GXDisposeShape(aLineShape);
```

As another example, the following code demonstrates how to create a text shape and draw it to the offscreen bitmap:

```
gxShape aTextShape;
gxPoint textLocation = {ff(70), ff(100)};
gxPoint textCenter;

aTextShape = GXNewText(9, (unsigned char *) "123456789",
                        &textLocation);
GXGetShapeCenter(aTextShape, 0, &textCenter);
GXScaleShape(aTextShape, ff(3), ff(3),
             textCenter.x, textCenter.y);
SetShapeCommonTransfer(aTextShape, gxXorMode);

GXSetShapeTransform(aTextShape, offscreenTransform);
GXDrawShape(aTextShape);
GXDisposeShape(aTextShape);
```

This code segment uses the `SetShapeCommonTransfer` library function, which is available in the transfer mode library.

Bitmap Shapes

Finally, to transfer the offscreen bitmap to the screen, you need only draw the bitmap:

```
GXDrawShape(aBitmapShape);
```

When drawing the offscreen bitmap, QuickDraw GX uses the information in the transform object of the offscreen bitmap shape. This example uses the `GXNewBitmap` function to create the offscreen bitmap, and so it references the same transform object as the default bitmap shape. The transform of the default bitmap references the default view port, as described in the chapter “View-Related Objects” in *Inside Macintosh: QuickDraw GX Objects*. Since the default view port is typically on screen, drawing the offscreen bitmap effectively transfers it to the screen.

Listing 5-13 shows the complete sample function to create an offscreen bitmap, draw shapes to it, and copy it to the screen.

Listing 5-13 Creating an offscreen bitmap

```
void CreateOffscreenBitmap(void)
{
    gxShape  aBitmapShape, aRectangleShape, aLineShape, aTextShape;

    gxRectangle boundsRectangle = {ff(0), ff(0), ff(200), ff(200)};
    gxLine lineGeometry = {ff(40), ff(40), ff(160), ff(160)};
    gxPoint textLocation = {ff(70), ff(100)};
    gxPoint textCenter;

    /* declare view group, and so forth. */
    aBitmapShape = CreateABitmap(200, 200);

    offscreenViewGroup = GXNewViewGroup();
    offscreenViewDevice = GXNewViewDevice(offscreenViewGroup,
                                          aBitmapShape);
    offscreenViewPort = GXNewViewPort(offscreenViewGroup);
    offscreenTransform = GXNewTransform();
    GXSetTransformViewPorts(offscreenTransform, 1,
                           &offscreenViewPort);

    /* draw white rectangle to clear bitmap */
    aRectangleShape = GXNewRectangle(&boundsRectangle);
    GXSetShapeTransform(aRectangleShape, offscreenTransform);
    SetShapeCommonColor(aRectangleShape, gxWhite);
    GXDrawShape(aRectangleShape);
    GXDisposeShape(aRectangleShape);
}
```

Bitmap Shapes

```

/* draw thick diagonal line offscreen */
aLineShape = GXNewLine(&lineGeometry);
GXSetShapePen(aLineShape, ff(50));
GXSetShapeTransform(aLineShape, offscreenTransform);
GXDrawShape(aLineShape);
GXDisposeShape(aLineShape);

/* draw text offscreen */
aTextShape = GXNewText(9, (unsigned char *) "123456789",
                        &textLocation) ;
GXGetShapeCenter(aTextShape, 0, &textCenter);
GXScaleShape(aTextShape, ff(3), ff(3), textCenter.x,
            textCenter.y);
SetShapeCommonTransfer(aTextShape, gxXorMode);

GXSetShapeTransform(aTextShape, offscreenTransform);
GXDrawShape(aTextShape);
GXDisposeShape(aTextShape);

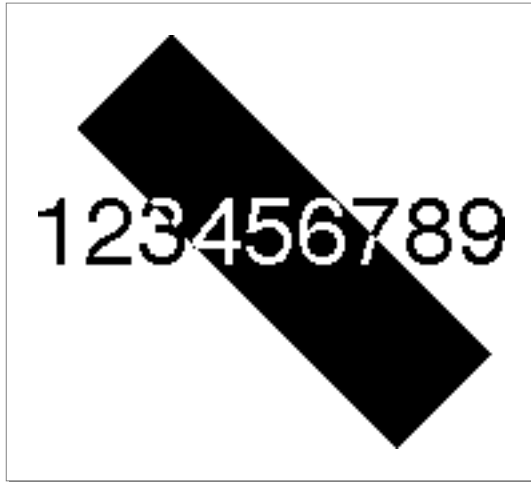
/* transfer bitmap to screen */
GXDrawShape(aBitmapShape);
GXDisposeShape(aBitmapShape);

GXDisposeTransform(offscreenTransform);
GXDisposeViewGroup(offscreenViewGroup);
}

```


Figure 5-29 shows the result of this function.

Figure 5-29 Multiple shapes drawn to a bitmap



The offscreen library provided with QuickDraw GX contains some utilities that simplify the creation of offscreen bitmaps. This library defines the `offscreen` structure, which contains a reference to a transform, view port, view device, and view group. Listing 5-14 shows how to use the offscreen library to create the bitmap shown in Figure 5-29.

Listing 5-14 Creating an offscreen bitmap using the offscreen library

```
void CreateOffscreenBitmap(void)
{
    shape  aBitmapShape, aRectangleShape, aLineShape, aTextShape;

    offscreen anOffscreen;

    const gxRectangle boundsRectangle = {ff(0), ff(0),
                                         ff(200), ff(200)};
    const gxLine lineGeometry = {ff(40), ff(40),
                                 ff(160), ff(160)};
    const gxPoint textLocation = {ff(70), ff(100)};
    gxPoint textCenter;

    aBitmapShape = CreateABitmap(200, 200);
```

Bitmap Shapes

```

/* create all offscreen-related objects */
CreateOffscreen(&anOffscreen, aBitmapShape);

aRectangleShape = GXNewRectangle(&boundsRectangle);
GXSetShapeTransform(aRectangleShape, anOffscreen.xform);
SetShapeCommonColor(aRectangleShape, gxWhite);
GXDrawShape(aRectangleShape);
GXDisposeShape(aRectangleShape);

aLineShape = GXNewLine(&lineGeometry);
GXSetShapePen(aLineShape, ff(50));
GXSetShapeTransform(aLineShape, anOffscreen.xform);
GXDrawShape(aLineShape);
GXDisposeShape(aLineShape);

aTextShape = GXNewText(9, (unsigned char *) "123456789",
                        &textLocation) ;
GXGetShapeCenter(aTextShape, 0, &textCenter);
GXScaleShape(aTextShape, ff(3), ff(3), textCenter.x,
             textCenter.y);
SetShapeCommonTransfer(aTextShape, gxXorMode);

GXSetShapeTransform(aTextShape, anOffscreen.xform);
GXDrawShape(aTextShape);
GXDisposeShape(aTextShape);

GXDrawShape(aBitmapShape);
GXDisposeShape(aBitmapShape);

/* dispose of all offscreen-related objects */
DisposeOffscreen(&anOffscreen);

GXDrawShape(aBitmapShape);
GXDisposeShape(aBitmapShape);
}

```

Editing Part of a Bitmap

QuickDraw GX provides two functions that allow you to manipulate part of a bitmap. The `GXGetBitmapParts` function copies a rectangular subsection from one bitmap to a new bitmap, and the `GXSetBitmapParts` function replaces a rectangular subsection of one bitmap with another bitmap.

To extract part of a bitmap shape, you need to declare a reference to a new bitmap shape to hold the extracted part:

```
gxShape extractedBitmap;
```

You also need to specify what part of the bitmap to extract. QuickDraw GX provides the `gxLongRectangle` structure for this purpose:

```
gxLongRectangle extractedBounds = {70, 70, 125, 125};
```

You can then use the `GXGetBitmapParts` function to extract the specified section. For example, the following call extracts from the bitmap referenced by the `aBitmapShape` variable the section starting at 70 pixels over and 70 pixels down and ending at 125 pixels over and 125 pixels down.

```
extractedBitmap = GXGetBitmapParts(aBitmapShape,
                                   &extractedBounds);
```

Applying this function call to the bitmap shown in Figure 5-29 results in the bitmap shown in Figure 5-30.

Figure 5-30 An extracted bitmap



You can use the `GXSetBitmapParts` function to replace a section of one bitmap with the contents of another bitmap.

For example, you might create a small, square bitmap containing all black pixels:

```
gxShape insertionBitmap;
gxRectangle insertionGeometry = {ff(0), ff(0), ff(100), ff(100)};

insertionBitmap = GXNewRectangle(&insertionGeometry);
GXSetShapeType(insertionBitmap, gxBitmapType);
```

Bitmap Shapes

Then you can insert that bitmap into the bitmap from Figure 5-29 by specifying where it should be inserted with the declaration

```
gxLongRectangle whereToInsert = {70, 70, 125, 125};
```

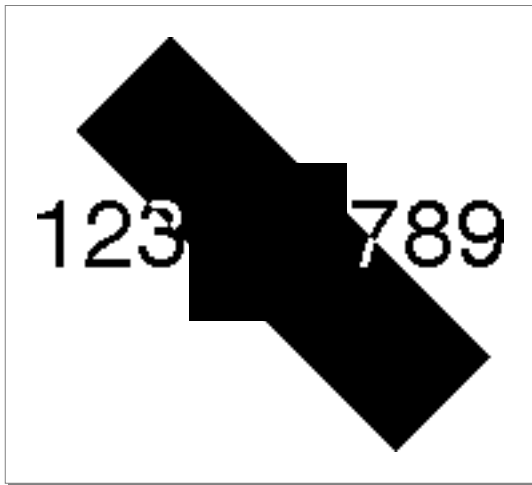
and then inserting it with this call to the `GXSetBitmapParts` function:

```
GXSetBitmapParts(aBitmapShape, &whereToInsert, insertionBitmap);
```

Notice that the `insertionBitmap` shape is larger than the `whereToInsert` rectangle. QuickDraw GX only inserts as much of the `insertionBitmap` shape as fits in the `whereToInsert` rectangle, starting with the upper-left corner of the `insertionBitmap` shape.

The resulting bitmap is shown in Figure 5-31.

Figure 5-31 An edited bitmap



For more information about the `GXGetBitmapParts` and the `GXSetBitmapParts` functions, see page 5-74 and page 5-75, respectively.

Applying Functions Described Elsewhere to Bitmap Shapes

QuickDraw GX provides only a small number of functions that apply exclusively to bitmaps. However, most of the QuickDraw GX functions that apply to other types of shapes can also be applied to bitmap shapes.

Bitmap Shapes

The next seven sections discuss how functions described elsewhere operate on bitmaps. These sections are as follows:

- “Functions That Post Errors or Warnings When Applied to Bitmap Shapes” on page 5-55, which lists functions that you can apply to other types of shapes but not to bitmap shapes
- “Shape-Related Functions Applicable to Bitmap Shapes” on page 5-56, which lists functions that operate on bitmap shape objects
- “Geometric Operations Applicable to Bitmap Shapes” on page 5-58, which lists the few geometric operation functions that you can apply to bitmap geometries
- “Style-Related Functions Applicable to Bitmap Shapes” on page 5-59, which lists the few style-related functions that affect the drawing of bitmaps
- “Ink-Related Functions Applicable to Bitmap Shapes” on page 5-59, which lists the functions that manipulate on the transfer mode of a bitmap shape’s ink object
- “Transform-Related Functions Applicable to Bitmap Shapes” on page 5-59, which discusses the functions that allow you to map and clip a bitmap as well as set its hit-test parameters and its view port list
- “View-Related Functions Applicable to Bitmap Shapes” on page 5-61, which lists the functions that allow you to associate a bitmap shape with a view device object

Functions That Post Errors or Warnings When Applied to Bitmap Shapes

Some QuickDraw GX functions that operate on other types of shapes only post an error or a warning if you try to apply them to a bitmap shape.

For example, the shape-editing functions listed in Table 5-1 operate on the geometric shape types, but not on bitmap shapes. These functions are described in Chapter 2, “Geometric Shapes,” in this book.

Table 5-1 Shape-editing functions that post errors or warnings when applied to bitmaps

Function name	Error or warning posted
GXGetShapeParts	shape_operator_may_not_be_a_bitmap
GXSetShapeParts	shape_operator_may_not_be_a_bitmap

Although you cannot apply the functions listed in Table 5-1 to a bitmap shape, you can use the `GXGetBitmapParts` and `GXSetBitmapParts` functions to edit sections of a bitmap. These functions are described in “Editing Bitmaps” beginning on page 5-71.

Bitmap Shapes

There are also a number of geometric operations that you cannot apply to bitmap shapes. Table 5-2 lists these functions, which are described in Chapter 4, “Geometric Operations,” in this book.

Table 5-2 Geometric operations that post errors or warnings when applied to bitmaps

Function name	Error or warning posted
GXBreakShape	graphic_type_does_not_contain_points
GXContainsShape	shape_operator_may_not_be_a_bitmap
GXDifferenceShape	shape_operator_may_not_be_a_bitmap
GXExcludeShape	shape_operator_may_not_be_a_bitmap
GXGetShapeCenter	illegal_type_for_shape
GXGetShapeDirection	graphic_type_does_not_have_multiple_contours
GXGetShapeLength	shape_does_not_have_length
GXInsetShape	graphic_type_cannot_be_inset
GXIntersectShape	shape_operator_may_not_be_a_bitmap
GXInvertShape	shape_cannot_be_inverted
GXReduceShape	graphic_type_cannot_be_reduced
GXReverseDifferenceShape	shape_operator_may_not_be_a_bitmap
GXReverseShape	contour_out_of_range
GXShapeLengthToPoint	shape_does_not_have_length
GXTouchesShape	shape_operator_may_not_be_a_bitmap
GXUnionShape	shape_operator_may_not_be_a_bitmap

Most of these geometric operations do not apply to bitmap shapes because the geometry of a bitmap is substantially different from the geometry of a geometric shape.

You can apply a few of the geometric operations to bitmaps, however. These functions are discussed in “Geometric Operations Applicable to Bitmap Shapes” beginning on page 5-58.

Shape-Related Functions Applicable to Bitmap Shapes

You can apply all of the functions described in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects* to bitmap shapes. These functions allow you to

- manipulate the shape object that represents the bitmap shape—for example, you can copy, clone, cache, compare, and dispose of the bitmap shape
- set the geometry, shape type, shape fill, and shape attributes of the bitmap shape

Bitmap Shapes

- change the style, ink, and transform objects that are associated with the bitmap shape
- manipulate the tags and owner count of the bitmap shape

Table 5-3 gives important bitmap-related information for a subset of the functions from the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*. Functions described in that chapter that do not appear in this list exhibit the same behavior when applied to bitmap shapes as they do when applied to other types of shapes.

Table 5-3 Shape-related functions that exhibit special behavior when applied to bitmaps

Function name	Action taken
GXChangedShape	Notifies QuickDraw GX that you have directly edited the geometry of the bitmap (using the GXGetShapeStructure and GXLockShape functions). You should call this function when you directly edit any field of a bitmap geometry structure or when you edit the pixel values of a bitmap’s pixel image.
GXCopyToShape	Makes a copy of the bitmap shape, but does not copy the pixel image. Instead, the new bitmap shape references the same pixel image.
GXCopyDeepToShape	Makes a copy of the bitmap shape, including a complete copy of the bitmap’s pixel image. The copy of the pixel image is allocated in QuickDraw GX memory, regardless of where the original image was allocated.
GXEqualShape	Determines if two bitmap shapes are equal—that is, their bitmap position, height, width, color space, color set, and color profile fields are equal, and their pixel images contain the same pixel values.
GXGetShapeSize	Determines the amount of memory currently used by the bitmap shape, including the amount of QuickDraw GX memory currently used by the pixel image of the bitmap.
GXGetDefaultShape	Returns a reference to the default bitmap shape. The default bitmap shape has 1 bit per pixel, 0 width, and 0 height.
GXGetShapeFill	Returns the shape fill of the shape. The shape fill for bitmap shapes is always even-odd fill or no fill.
GXGetShapeStructure	Returns a pointer to the geometry of the bitmap shape. You can use this function to determine the address of the pixel image, even if it is allocated in QuickDraw GX memory.
GXLockShape	Loads the bitmap shape into memory and locks its geometry into a fixed memory location. If the pixel image is allocated in QuickDraw GX memory, it is loaded and locked as well.
GXNewShape	Creates a bitmap shape with 0 width, 0 height, 32 bits per pixel and rgb32space color space.
GXSetShapeFill	Sets the shape fill of the shape. You must always set the shape fill of a bitmap shape to even-odd fill or no fill.
GXSetShapeType	Changes the shape type of the bitmap shape and converts the shape fill and geometry as appropriate.

Geometric Operations Applicable to Bitmap Shapes

Most geometric operations post errors or warnings when applied to bitmap shapes, as described in “Functions That Post Errors or Warnings When Applied to Bitmap Shapes” on page 5-55.

You can, however, apply the remainder of the functions described in Chapter 4, “Geometric Operations,” to bitmap shapes. Table 5-4 gives important bitmap-related information for a subset of these functions. The remainder of the geometric operations exhibit the same behavior when applied to bitmap shapes as they do when applied to other types of shapes.

Table 5-4 Geometric operations that exhibit special behavior when applied to bitmaps

Function name	Action taken
GXGetShapeArea	Returns bitmap width multiplied by bitmap height.
GXPrimitiveShape	Applies sourceGridStyle attribute to bitmap position.
GXSimplifyShape	Reduces the pixel size of the bitmap if the bitmap uses a limited number of colors.
GXSetShapeBounds	If the gxMapTransformShape shape attribute is set, this function changes the transform mapping of the bitmap; otherwise, it changes the bitmap height, width, and location, and creates a new, scaled version of the bit image to fit in the new bounding rectangle.

Style-Related Functions Applicable to Bitmap Shapes

As discussed in “Bitmap Styles and Inks” on page 5-8, bitmap shapes make limited use of their style objects. Although you can apply to a bitmap shape any of the functions described in Chapter 3, “Geometric Styles,” only the `GXSetShapeStyleAttributes` function affects the drawing of the bitmap. While you can use this function to set or clear any of a bitmap’s style attributes, QuickDraw GX considers only the `gxSourceGridStyle` style attribute and the `gxDeviceGridStyle` style attribute when drawing bitmaps; other attributes are ignored.

You may use the other style-related functions (such as `GXSetShapePen`, `GXSetShapeDash`, and so on) to set the other properties of a bitmap’s style object, and you may use the corresponding functions (`GXGetShapePen`, `GXGetShapeDash`, and so on) to examine these properties. However, QuickDraw GX ignores these properties when drawing a bitmap.

Ink-Related Functions Applicable to Bitmap Shapes

Since bitmap shapes contain their own color information in their geometries, QuickDraw GX does not use the color property of the ink object when drawing a bitmap. However, QuickDraw GX does consider the transfer mode of the ink object and applies it to each pixel when drawing a bitmap. You can use the `GXSetShapeTransfer` function, which is described in the chapter “Ink Objects” in *Inside Macintosh: QuickDraw GX Objects*, to assign a transfer mode to a bitmap shape.

You may also use the `GXSetShapeColor` function to set the color property of a bitmap’s ink object and use the `GXGetShapeColor` function to examine this property. However, QuickDraw GX ignores this property when drawing a bitmap.

Transform-Related Functions Applicable to Bitmap Shapes

Although bitmap shapes do not make full use of their style and ink objects, they do make full use of their transform objects. You can apply all of the shape-related functions that are described in the chapter “Transform Objects” in *Inside Macintosh: QuickDraw GX Objects* to bitmap shapes.

Bitmap Shapes

Table 5-5 gives important bitmap-related information for a subset of the functions from that chapter. Functions described in that chapter that do not appear in this list exhibit the same behavior when applied to bitmap shapes as they do when applied to other types of shapes.

Table 5-5 Transform-related functions that exhibit special behavior when applied to bitmaps

Function name	Action taken
<code>GXGetShapeHitTest</code>	Returns the hit-test parameters associated with the bitmap shape's transform. QuickDraw GX hit-tests bitmaps using only the <code>boundsPart</code> shape part.
<code>GXMapShape</code>	Multiplies the mapping associated with the transform object of the bitmap shape (if the <code>gxMapTransformShape</code> shape attribute of the bitmap shape is set) by a mapping matrix, or applies the mapping directly to the geometry of the bitmap (if the <code>gxMapTransformShape</code> attribute is not set). Depending on the mapping, this function may also change the clip shape of the bitmap.
<code>GXMoveShape</code>	Moves the bitmap by a specified distance. This function can affect the mapping of the bitmap's transform or the geometry of the bitmap itself, depending on the value of the <code>gxMapTransformShape</code> shape attribute of the bitmap shape.
<code>GXMoveShapeTo</code>	Moves the bitmap to a specified position. This function can affect the mapping of the bitmap's transform or the geometry of the bitmap itself, depending on the value of the <code>gxMapTransformShape</code> shape attribute of the bitmap shape.
<code>GXRotateShape</code>	Rotates the bitmap. This function can affect the mapping of the bitmap's transform or the geometry of the bitmap itself, depending on the value of the <code>gxMapTransformShape</code> shape attribute of the bitmap shape. This function can also affect the clip shape of the bitmap.
<code>GXScaleShape</code>	Scales the bitmap. This function can affect the mapping of the bitmap's transform or the geometry of the bitmap itself, depending on the value of the <code>gxMapTransformShape</code> shape attribute of the bitmap shape. This function can also affect the clip shape of the bitmap.
<code>GXSkewShape</code>	Skews the bitmap. This function can affect the mapping of the bitmap's transform or the geometry of the bitmap itself, depending on the value of the <code>gxMapTransformShape</code> shape attribute of the bitmap shape. This function can also affect the clip shape of the bitmap.
<code>GXSetShapeClip</code>	Assigns a clip shape to the transform object associated with the bitmap shape.
<code>GXSetShapeHitTest</code>	Assigns hit-test parameters to the transform object associated with the bitmap shape. QuickDraw GX only hit-tests bitmaps using the <code>boundsPart</code> shape part.
<code>GXSetShapeMapping</code>	Changes the mapping associated with the transform object of the bitmap shape (if the <code>gxMapTransformShape</code> shape attribute of the bitmap shape is set) or applies the mapping directly to the geometry of the bitmap (if the <code>gxMapTransformShape</code> attribute is not set). Depending on the mapping, this function may also change the clip shape of the bitmap.

View-Related Functions Applicable to Bitmap Shapes

As described in “Bitmaps and View Devices” beginning on page 5-12, view device objects use bitmaps to store rendered shape images. Table 5-6 lists the function that allows you to determine the bitmap assigned to a view device and the function that allows you to change the bitmap of a view device. Both of these functions are described in detail in the chapter “View-Related Objects” in *Inside Macintosh: QuickDraw GX Objects*.

Table 5-6 View-related functions that can be applied to bitmaps

Function name	Action taken
GXGetViewDeviceBitmap	Returns the bitmap shape associated with a view device object.
GXSetViewDeviceBitmap	Assigns a bitmap shape to a view device. You can use this function to create offscreen bitmaps, which are discussed in “Bitmaps and View Devices” beginning on page 5-12.

Bitmap Shapes Reference

This section describes the data types and functions you use to create and manipulate bitmap shapes. The first subsection, “Constants and Data Types,” shows the definitions of the data types related to bitmap shapes. The section “Functions,” beginning on page 5-65, gives a complete reference for the functions specific to bitmaps. In addition to the functions described in this section, you can apply many functions described elsewhere to bitmap shapes. See the section “Applying Functions Described Elsewhere to Bitmap Shapes,” which begins on page 5-54, for more details.

Constants and Data Types

This section describes the data structures you use to create and manipulate bitmaps.

You use the `gxBitmap` structure to specify information about a bitmap geometry and information about how QuickDraw GX should create a bitmap shape. You also use this data structure to specify color information for bitmaps. A complete discussion of the QuickDraw GX color architecture appears in the chapter “Color and Color-Related Objects” in *Inside Macintosh: QuickDraw GX Objects*.

You use the `gxLongRectangle` structure to specify a rectangular subsection of a bitmap pixel image when editing bitmap parts.

This section also discusses the constants and data types you use when creating disk-based bitmaps.

The Bitmap Geometry Structure

The `gxBitmap` structure specifies the geometry of a bitmap shape. You can use this data structure when creating bitmap shapes with the `GXNewBitmap` function, when altering bitmap shapes with the `GXGetBitmap` and `GXSetBitmap` functions, and when directly editing bitmap shapes with the `GXGetShapeStructure` function.

The `gxBitmap` structure is defined as follows:

```
typedef struct {
    char        *image;
    long         width;
    long         height;
    long         rowBytes;
    long         pixelSize;
    gxColorSpace space;
    gxColorSet   set;
    gxColorProfile profile;
} gxBitmap;
```

Field descriptions

<code>image</code>	A pointer to the pixel image. When creating a bitmap, you can specify <code>nil</code> for this field to indicate that QuickDraw GX should allocate memory for the pixel image of the bitmap.
<code>width</code>	The width of the bitmap in pixels.
<code>height</code>	The height of the bitmap in pixels.
<code>rowBytes</code>	The number of bytes of the pixel image corresponding to each row of the bitmap. This value must be a positive even number.
<code>pixelSize</code>	The number of bits representing a single pixel in the pixel image. This value must be 1, 2, 4, 8, 16, or 32.
<code>space</code>	The color space that QuickDraw GX uses when interpreting the pixel values in the pixel image. When creating a bitmap, you may specify the <code>gxNoSpace</code> constant for this field to indicate that QuickDraw GX should choose a color space for you. If the value of the <code>pixelSize</code> field is 32, QuickDraw GX uses the value <code>gxRGB32Space</code> ; if the pixel size is 16, QuickDraw GX uses <code>gxRGB16Space</code> ; if the pixel size is 8 or less, QuickDraw GX uses <code>gxIndexedSpace</code> and creates the default color set for the pixel size, which is usually a grayscale color set.

Bitmap Shapes

set	The color set that QuickDraw GX uses when interpreting the pixel values of the pixel image. If the <code>space</code> field contains the value <code>gxIndexedSpace</code> , QuickDraw GX interprets the pixel values in the pixel image as indexes to this color set. If the bitmap's color space is not <code>gxIndexedSpace</code> , this field should be <code>nil</code> .
profile	The color matching information about the device on which the bitmap was created. You may provide a reference to a color profile object, or you may set the value of this field to <code>nil</code> .

Implementation Note

Version 1.0 of QuickDraw GX limits the bitmap width and the bitmap height to 32,767. ♦

When creating a bitmap, you can allocate the memory for the pixel image of the bitmap yourself and store a pointer to it in the `image` field, or you can set the `image` field to `nil`, which indicates that QuickDraw GX should allocate the pixel image memory.

If you create the pixel image for a bitmap, you must pad the end of each row of the pixel image so that each row contains an even number of bytes. You must store the number of bytes per row in the `rowBytes` field, unless you are creating a bitmap with a pixel image in QuickDraw GX memory, in which case you want to set this field to 0.

If you set the `image` field to `nil` when creating a bitmap, QuickDraw GX does two things:

- creates an uninitialized pixel image based on the bitmap width and height you specify in the `width` and `height` fields and the pixel size you specify in the `pixelSize` field
- determines an appropriate `rowBytes` value

If you want to create a bitmap with a disk-based pixel image, you should specify the `gxBitmapFileAliasImageValue` constant for the `image` field.

If you specify the `gxNoSpace` constant for the `space` field, QuickDraw GX chooses an appropriate color space for you, based on the value of the `pixelSize` field.

If you specify the color space yourself, you must be sure the pixel size of that color space matches the value you indicate in the `pixelSize` field.

For a discussion of pixel images, bitmap width, bitmap height, and pixel size, see “Bitmap Geometries” beginning on page 5-5. For a detailed discussion of color spaces, color sets, and color profiles, see the chapter “Color and Color-Related Objects” in *Inside Macintosh: QuickDraw GX Objects*.

For examples of creating `gxBitmap` structures and bitmap shapes, see “Creating and Drawing Bitmaps” beginning on page 5-15.

The Long Rectangle Structure

The `gxLongRectangle` structure allows you to specify a rectangular subsection of the pixel image of a bitmap shape. It differs from the `gxRectangle` structure, described in the chapter “Geometric Shapes” in this book, in that the coordinates of a `gxLongRectangle` structure have no fractional part.

```
struct gxLongRectangle {
    long    left;
    long    top;
    long    right;
    long    bottom;
};
```

Field descriptions

<code>left</code>	The left side of the rectangle in number of pixels.
<code>top</code>	The top of the rectangle in number of pixels.
<code>right</code>	The right side of the rectangle in number of pixels.
<code>bottom</code>	The bottom of the rectangle in number of pixels.

You use the `gxLongRectangle` structure when editing parts of a bitmap, as discussed in “Editing Part of a Bitmap” beginning on page 5-53.

Constants For Bitmaps With Disk-Based Pixel Images

QuickDraw GX provides two constants for you to use when creating bitmaps with disk-based pixel images.

```
#define  gxBitmapFileAliasImageValue    0x00000001

#define  gxBitmapFileAliasTagType       'bfil'
```

You indicate that a bitmap uses a disk-based pixel image by setting the bitmap geometry’s `image` field to the `gxBitmapFileAliasImageValue` constant. You specify which file contains the pixel image in a bitmap data source alias structure, which you attach to the bitmap using a tag with the `gxBitmapFileAliasTagType` tag type.

For an example of a bitmap with a disk-based pixel image, see “Creating Bitmaps With Disk-Based Pixel Images” beginning on page 5-44.

Bitmap Data Source Alias Structure

QuickDraw GX provides the bitmap data source alias structure to allow you to specify file information for disk-based pixel images.

```
struct gxBitmapDataSourceAlias {
    unsigned long fileOffset;
    unsigned long aliasRecordSize;
    unsigned char aliasRecord[gxAnyNumber];
};
```

Field descriptions

<code>fileOffset</code>	The offset in bytes from the beginning of the file to the first pixel value of the pixel image.
<code>aliasRecordSize</code>	The size in bytes of the alias record.
<code>aliasRecord</code>	A Macintosh Alias Manager alias record specifying the file containing the pixel image.

For an example of a bitmap with a disk-based pixel image, see “Creating Bitmaps With Disk-Based Pixel Images” beginning on page 5-44.

Functions

This section describes the functions provided by QuickDraw GX specifically for creating and manipulating bitmap shapes. With the functions described in this section, you can

- create a new bitmap shape
- determine and replace the geometry of a bitmap shape
- edit a single pixel of a bitmap
- examine or replace a rectangular subsection of a bitmap

The section “Applying Functions Described Elsewhere to Bitmap Shapes,” which begins on page 5-54, contains information about other QuickDraw GX functions that you can apply to bitmap shapes.

Creating Bitmaps

This section describes the function you use to create new bitmap shapes.

The `GXNewBitmap` function requires that you specify information about the bitmap in a `gxBitmap` structure, and the function encapsulates that information in a new bitmap shape.

GXNewBitmap

You can use the `GXNewBitmap` function to create a new bitmap shape.

```
gxShape GXNewBitmap(const gxBitmap *data,
                    const gxPoint *position);
```

<code>data</code>	A pointer to a <code>gxBitmap</code> bitmap structure that specifies information about the bitmap shape you want to create.
<code>position</code>	A pointer to a <code>gxPoint</code> structure that indicates the initial position of the upper-right corner of the bitmap. You may set this parameter to <code>nil</code> to indicate (0.0, 0.0).

function result A reference to the newly created bitmap shape.

DESCRIPTION

The `GXNewBitmap` function creates a new bitmap shape and returns a reference to that shape as its function result.

You specify the initial position of the new bitmap in the `position` parameter, and you specify the rest of the bitmap geometry by creating a `gxBitmap` structure and passing a pointer to it in the `data` parameter.

You must provide values for the `width`, `height`, and `pixelSize` fields of the `gxBitmap` structure.

Implementation Note

Version 1.0 of QuickDraw GX limits the bitmap width and the bitmap height to 32,767. ♦

You may specify the pixel image in the `image` field of the bitmap geometry structure, or you may set this field to `nil`, in which case QuickDraw GX allocates memory for the pixel image based on the requested width, height, and pixel size. If you supply the pixel image, you must also supply an appropriate value in the `rowBytes` field of the bitmap geometry structure. If QuickDraw GX allocates the pixel image, you should initialize the `rowBytes` field to 0.

You may indicate a color space for the bitmap in the `space` field of the bitmap geometry structure, but the pixel size of that color space must match the pixel size you specify in the `pixelSize` field. If you specify the `gxNoSpace` constant for the `space` field, QuickDraw GX chooses a color space for you:

- If you indicate in the `pixelSize` field a pixel size of 16 or 32, QuickDraw GX chooses the `gxRGB16Space` color space or the `gxRGB32Space` color space, respectively.
- If you indicate in the `pixelSize` field a pixel size of 1, 2, 4, or 8, QuickDraw GX chooses the `gxIndexedSpace` color space, and creates a default color set of the appropriate size.

Bitmap Shapes

If you indicate the `gxIndexedSpace` color space for the `space` field, you must provide a color set in the `set` field.

In the `profile` field, you may provide a reference to a color profile describing the color matching information for the device on which the bitmap was created, or you may set this field to `nil`.

SPECIAL CONSIDERATIONS

If no error results, the `GXNewBitmap` function creates a bitmap shape; you are responsible for disposing of this shape when you no longer need it. See *Inside Macintosh: QuickDraw GX Objects* for information about creating and disposing of shapes.

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>size_of_bitmap_exceeds_implementation_limit</code>	
<code>parameter_is_nil</code>	(debugging version)
<code>invalid_pixelSize</code>	(debugging version)
<code>bitmap_height_negative</code>	(debugging version)
<code>bitmap_width_negative</code>	(debugging version)
<code>bitmap_height_negative</code>	(debugging version)
<code>bitmap_rowBytes_negative</code>	(debugging version)
<code>bitmap_rowBytes_too_small</code>	(debugging version)
<code>bitmap_rowBytes_not_aligned</code>	(debugging version)
<code>bitmap_ptr_not_aligned</code>	(debugging version)
<code>bitmap_rowBytes_must_be_specified_for_user_image_buffer</code>	(debugging version)
<code>colorSpace_out_of_range</code>	(debugging version)

Warnings

<code>shape_access_not_allowed</code>	(debugging version)
---------------------------------------	---------------------

SEE ALSO

For examples using this function, see “Creating and Drawing Bitmaps” beginning on page 5-15.

For information about the `gxBitmap` structure, see “The Bitmap Geometry Structure” beginning on page 5-62.

For information about bitmap width, height, and pixel size, see “Bitmap Geometries” beginning on page 5-5.

For information about disposing of bitmap shapes, see the description of the `GXDisposeShape` function, which is in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

For a complete discussion of the QuickDraw GX color architecture, see the chapter “Color and Color-Related Objects” in *Inside Macintosh: QuickDraw GX Objects*.

Getting and Setting Bitmap Geometries

This section describes the functions you can use to examine or replace the entire geometry of a bitmap shape.

The `GXGetBitmap` function copies the information from the geometry of a bitmap shape into a `gxBitmap` data structure.

The `GXSetBitmap` function replaces the geometry of a bitmap shape with information you provide in a `gxBitmap` structure.

GXGetBitmap

You can use the `GXGetBitmap` function to obtain a copy of the information in a bitmap shape's geometry.

```
gxBitmap *GXGetBitmap(gxShape source, gxBitmap *data,
                      gxPoint *position);
```

<code>source</code>	A reference to the bitmap shape whose geometry you want to copy.
<code>data</code>	A pointer to a <code>gxBitmap</code> structure. On return, this structure contains information copied from the geometry of the bitmap shape.
<code>position</code>	A pointer to a <code>gxPoint</code> structure. On return, this structure indicates the position of the upper-left corner of the bitmap shape.

function result A pointer to a `gxBitmap` structure containing information from the the geometry of the bitmap shape. This value is the same as the value you provided in the `data` parameter.

DESCRIPTION

The `GXGetBitmap` function copies the information from the geometry of the bitmap shape indicated by the `source` parameter to the `gxBitmap` structure pointed to by the `data` parameter and returns a pointer to this information as its function result. This function also copies the bitmap position from the bitmap geometry to the `gxPoint` structure pointed to by the `position` parameter.

You may specify `nil` for the `data` or `position` parameters. If you do, this function does return the corresponding information.

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>illegal_type_for_shape</code>	(debugging version)

SEE ALSO

For information about the `gxBitmap` structure, see “The Bitmap Geometry Structure” beginning on page 5-62.

For information about pixel images, see “Bitmap Geometries” beginning on page 5-5.

To create a bitmap shape, use the `GXNewBitmap` function, which is described on page 5-66.

To change the geometry of a bitmap shape, use the `GXSetBitmap` function, which is described in the next section.

GXSetBitmap

You can use the `GXSetBitmap` function to change the information in the geometry of a bitmap shape.

```
void GXSetBitmap(gxShape target, const gxBitmap *data,
                 const gxPoint *position);
```

<code>target</code>	A reference to the bitmap shape whose geometry you want to change.
<code>data</code>	A pointer to a <code>gxBitmap</code> structure containing new information for the geometry of the target bitmap shape.
<code>position</code>	A pointer to a <code>gxPoint</code> structure indicating the new bitmap position for the target bitmap shape.

DESCRIPTION

The `GXSetBitmap` function uses information you provide both in the `gxBitmap` structure pointed to by the `data` parameter and the `gxPoint` structure pointed to by the `position` parameter to change the information in the geometry of the bitmap shape referenced by the `target` parameter. If the target shape is not a bitmap shape, this function converts the target shape to a bitmap shape before setting the geometry of the shape.

You can change only the bitmap position by creating a `gxPoint` structure, setting its fields to reflect the new position, passing a pointer to it in the `position` parameter, and setting the `data` parameter to `nil`.

You can change other information in the geometry of the target bitmap shape by providing new information in a `gxBitmap` structure and passing a pointer to this structure in the `data` parameter.

If the pixel image of the target bitmap shape was not allocated by QuickDraw GX (for example, if you allocated the pixel image yourself before calling the `GXNewBitmap` function), then the `GXSetBitmap` function simply replaces the information in the geometry of the target bitmap shape with information from the fields of the `gxBitmap` structure pointed to by the `data` parameter.

Bitmap Shapes

However, if QuickDraw GX allocated the pixel image of the target bitmap shape, you can use this function to change the dimensions of the existing pixel image.

You can change the bitmap height by providing a new height in the `height` field of the `gxBitmap` structure. You can change the bitmap width by setting the `rowBytes` field to 0 and provide a new bitmap width in the `width` field of the bitmap geometry structure. In this case, QuickDraw GX calculates an appropriate number of bytes per row.

In either case, this function does not scale the original pixel image; instead, it changes the amount of memory allocated to hold the pixel image. If you decrease the dimensions of the pixel image, QuickDraw GX fits the pixels in the original pixel image into a smaller space in memory, thereby losing some of the original pixel values. If you increase the dimensions of the pixel image, QuickDraw GX allocates more memory (possibly moving the original pixel image), thereby adding uninitialized pixels to the pixel image.

If QuickDraw GX allocated the original pixel image, you can also change the pixel size of the bitmap shape. You provide the new pixel size in the `pixelSize` field of the `gxBitmap` structure and the `GXSetBitmap` function expands or compresses the image to fit in the new pixel size. If you specify a smaller pixel size than the original, this function redistributes the colors in the color space of the bitmap shape.

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>size_of_bitmap_exceeds_implementation_limit</code>	
<code>parameter_is_nil</code>	(debugging version)
<code>invalid_pixelSize</code>	(debugging version)
<code>bitmap_height_negative</code>	(debugging version)
<code>bitmap_width_negative</code>	(debugging version)
<code>bitmap_height_negative</code>	(debugging version)
<code>bitmap_rowBytes_negative</code>	(debugging version)
<code>bitmap_rowBytes_too_small</code>	(debugging version)
<code>bitmap_rowBytes_not_aligned</code>	(debugging version)
<code>bitmap_ptr_not_aligned</code>	(debugging version)
<code>bitmap_rowBytes_must_be_specified_for_user_image_buffer</code>	(debugging version)
<code>colorSpace_out_of_range</code>	(debugging version)

Warnings

<code>shape_access_not_allowed</code>	(debugging version)
---------------------------------------	---------------------

SEE ALSO

For information about the `gxBitmap` structure, see “The Bitmap Geometry Structure” beginning on page 5-62.

For information about pixel images, bitmap height, bitmap width, pixel size, and number of rows per byte, see “Bitmap Geometries” beginning on page 5-5.

Bitmap Shapes

For a complete discussion of the QuickDraw GX color architecture, see the chapter “Color and Color-Related Objects” in *Inside Macintosh: QuickDraw GX Objects*.

To create a bitmap shape, use the `GXNewBitmap` function, which is described on page 5-66.

To obtain a copy of the information from the geometry of a bitmap shape, use the `GXGetBitmap` function, which is described on page 5-68.

Editing Bitmaps

This section describes the functions you can use to examine and change information in the pixel image of a bitmap shape.

The `GXGetShapePixel` function allows you to examine the value of a single pixel. The `GXSetShapePixel` function allows you to change the value of a single pixel.

The `GXGetBitmapParts` function allows you to extract a rectangular section of one bitmap shape and encapsulate it in another bitmap shape. The `GXSetBitmapParts` function allows you to replace a rectangular section of one bitmap shape with the pixel image of another bitmap shape.

GXGetShapePixel

You can use the `GXGetShapePixel` function to determine the pixel value and the pixel offset of a specific pixel in a bitmap shape.

```
long GXGetShapePixel(gxShape source, long x, long y,
                    gxColor *data, long *index);
```

<code>source</code>	A reference to the bitmap shape containing the pixel to examine.
<code>x</code>	The index of the column in which the pixel lies.
<code>y</code>	The index of the row in which the pixel lies.
<code>data</code>	A pointer to a <code>gxColor</code> structure. On return, this structure contains the color value of the specified pixel.
<code>index</code>	A pointer to a <code>long</code> value. On return, this value contains the color value of the specified pixel (if the pixel size of the bitmap is 16 or 32) or the specified pixel's index into the bitmap's color set (if the pixel size of the bitmap is 1, 2, 4, or 8).

function result The index of the byte containing the specified pixel in the source bitmap's pixel image.

Bitmap Shapes

DESCRIPTION

The `GXGetShapePixel` function copies the pixel value of the pixel determined by the `x` and `y` parameters from the source bitmap shape into the `gxColor` structure pointed to by the `data` parameter.

If the source bitmap shape has the `gxKeepShapeDirect` shape attribute set, this function also determines the pixel offset of the specified pixel and returns it in the `long` value pointed to by the `index` parameter. This function also returns a pointer to this value as the function result.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`shape_is_nil`

Warnings

`shape_does_not_contain_a_bitmap` (debugging only)

SEE ALSO

For information about the `gxBitmap` structure, see “The Bitmap Geometry Structure” beginning on page 5-62.

For information about pixels, pixel values, and pixel offsets, see “Bitmap Geometries” beginning on page 5-5.

To examine more than a single pixel of a bitmap, use the `GXGetBitmapParts` function, which is described on page 5-74.

To change the value of a pixel, use the `GXSetShapePixel` function, which is described in the next section.

GXSetShapePixel

You can use the `GXSetShapePixel` function to change the pixel value of a specific pixel in a bitmap shape.

```
void GXSetShapePixel(gxShape target, long x, long y,
                    const gxColor *newColor, long newIndex);
```

<code>target</code>	A reference to the bitmap shape containing the pixel to change.
<code>x</code>	The index of the column in which the pixel lies.
<code>y</code>	The index of the row in which the pixel lies.

Bitmap Shapes

<code>newColor</code>	A pointer to a <code>gxColor</code> structure indicating the new pixel value of the specified pixel. You may specify <code>nil</code> for this parameter if the target bitmap shape has the <code>gxIndexedSpace</code> color space.
<code>newIndex</code>	An index into a color set. You may use this parameter to set the pixel value if the target bitmap shape has the <code>gxIndexedSpace</code> color space.

DESCRIPTION

The `GXSetShapePixel` function sets the pixel value of a specific pixel in the target bitmap. The pixel is determined by the values you provide in the `x` and `y` parameters. The new pixel value is determined by the `newColor` or `newIndex` parameter:

- If you provide a color value in the `newColor` parameter, this function sets the pixel value of the specified pixel to be the closest color available in the color space of the target bitmap shape—even if the target bitmap shape has the `gxIndexedSpace` color space.
- Alternatively, and only if the target bitmap shape has the `gxIndexedSpace` color space, you may provide `nil` for the `newColor` parameter and provide in the `newIndex` parameter a new index into the color set of the bitmap shape.

This function posts a `functionality_unimplemented` error for disk-based bitmaps.

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>point_does_not_intersect_bitmap</code>	(debugging only)
<code>functionality_unimplemented</code>	(debugging only)

Warnings

<code>shape_does_not_contain_a_bitmap</code>	(debugging only)
----------------------------------------------	------------------

SEE ALSO

For information about the `gxBitmap` structure, see “The Bitmap Geometry Structure” beginning on page 5-62.

For information about pixels, pixel values, and pixel offsets, see “Bitmap Geometries” beginning on page 5-5.

To change more than a single pixel of a bitmap, use the `GXSetBitmapParts` function, which is described on page 5-75.

To examine the value of a pixel, use the `GXGetShapePixel` function, which is described on page 5-71.

GXGetBitmapParts

You can use the `GXGetBitmapParts` function to extract a rectangular section of pixels from a bitmap.

```
gxShape GXGetBitmapParts(gxShape source,
                        const gxLongRectangle *bounds);
```

`source` A reference to the bitmap shape containing the pixels to extract.

`bounds` A pointer to a `gxRectangle` indicating which part of the bitmap to extract.

function result A reference to a new bitmap shape containing only the extracted section of the source bitmap shape.

DESCRIPTION

The `GXGetBitmapParts` function extracts the pixels whose row number and column number fall within the boundaries of the rectangle pointed to by the `bounds` parameter, encapsulates the extracted pixel image in a new bitmap shape, and returns a reference to the new bitmap shape as the function result.

The returned bitmap shape has the same pixel size as the source bitmap shape. The returned bitmap shape also shares the same color space, color set, and color profile as the source bitmap shape.

The pixel image of the returned bitmap is allocated in QuickDraw GX memory.

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>parameter_is_nil</code>	
<code>parameter_out_of_range</code>	(debugging only)
<code>point_does_not_intersect_bitmap</code>	(debugging only)

Warnings

<code>shape_does_not_contain_a_bitmap</code>	(debugging only)
----------------------------------------------	------------------

SEE ALSO

For examples using this function, see “Editing Part of a Bitmap” beginning on page 5-53.

For information about the `gxBitmap` structure, see “The Bitmap Geometry Structure” beginning on page 5-62.

For information about the `gxLongRectangle` structure, see “The Long Rectangle Structure” on page 5-64.

For information about pixels and pixel images, see “Bitmap Geometries” beginning on page 5-5.

To examine a single pixel of a bitmap, use the `GXGetShapePixel` function, which is described on page 5-71.

To change a section of a bitmap, use the `GXSetBitmapParts` function, which is described in the next section.

GXSetBitmapParts

You can use the `GXSetBitmapParts` function to replace the pixel values in a rectangular subsection of a bitmap’s pixel image.

```
void GXSetBitmapParts(gxShape target, const gxRectangle *bounds,
                     gxShape bitmapShape);
```

target A reference to the bitmap shape containing the pixels to replace.

bounds A pointer to a `gxRectangle` structure indicating which part of the target bitmap to replace.

bitmapShape A reference to a bitmap shape containing the pixel values to use when replacing the specified pixels in the target bitmap shape.

DESCRIPTION

The `GXSetBitmapParts` function copies the pixel values (starting at the upper-left corner of the pixel image) of the source bitmap shape (which is indicated by the `bitmapShape` parameter) to the pixel image of the target bitmap shape. The `bounds` parameter determines how many rows and columns this function copies and where in the target bitmap the function places the copied pixel values.

The pixel image of the source bitmap may not be smaller than the size indicated by the `bounds` parameter; that is, the number of rows and columns in the pixel image of the source bitmap shape may not be less than the height and width of the specified rectangle, respectively.

Bitmap Shapes

The source and target bitmap shapes must have the same pixel size, color space, and color set.

This function posts a `functionality_unimplemented` error for disk-based bitmaps.

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>parameter_is_nil</code>	
<code>parameter_out_of_range</code>	(debugging only)
<code>point_does_not_intersect_bitmap</code>	(debugging only)
<code>functionality_unimplemented</code>	(debugging only)

Warnings

<code>shape_does_not_contain_a_bitmap</code>	(debugging only)
----------------------------------------------	------------------

SEE ALSO

For examples using this function, see “Editing Part of a Bitmap” beginning on page 5-53.

For information about the `gxBitmap` structure, see “The Bitmap Geometry Structure” beginning on page 5-62.

For information about the `gxLongRectangle` structure, see “The Long Rectangle Structure” on page 5-64.

For information about pixels and pixel images, see “Bitmap Geometries” beginning on page 5-5.

To change the pixel value of a single pixel, use the `GXSetShapePixel` function, which is described on page 5-72.

To extract a rectangular subsection of a bitmap, use the `GXGetBitmapParts` function, which is described on page 5-74.

Drawing Bitmaps

QuickDraw GX provides two methods of drawing a bitmap:

- You can create a bitmap shape (by calling the `GXNewBitmap` function, by copying an existing bitmap shape, and so on) and use the `GXDrawShape` function to draw the bitmap.
- You can create a `gxBitmap` structure and use the `GXDrawBitmap` function to draw the bitmap.

Bitmap Shapes

In general, you should use the `GXDrawShape` function to draw any QuickDraw GX graphic, including bitmap shapes. In fact, the `GXDrawBitmap` function creates a temporary bitmap shape, uses the `GXDrawShape` function to draw it, and then disposes of it. The `GXDrawShape` function is described in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

You would typically use the `GXDrawBitmap` function only in simple situations—for example, if you knew you wanted to draw a particular bitmap only once.

GXDrawBitmap

You can use the `GXDrawBitmap` function to draw a bitmap without encapsulating the bitmap geometry in a bitmap shape.

```
void GXDrawBitmap(const gxBitmap *data, const gxPoint *position);
```

<code>data</code>	A pointer to a <code>gxBitmap</code> structure that specifies information about the bitmap you want to draw.
<code>position</code>	A pointer to a <code>gxPoint</code> structure which indicates the position to draw the bitmap.

DESCRIPTION

The `GXDrawBitmap` function allows you to draw a bitmap without having to create a bitmap shape yourself. Instead, you create a `gxBitmap` structure specifying the bitmap you want to draw and a `gxPoint` structure indicating the position of the bitmap, and then you pass a pointer to these structures in the `data` and `position` parameters, respectively.

The `GXDrawBitmap` function calls the `GXNewBitmap` function to create a temporary bitmap shape using the values specified in these structures and the style, ink and transform of the default bitmap shape. Then the `GXDrawBitmap` function draws the bitmap shape using the `GXDrawShape` function.

For information about how QuickDraw GX creates bitmap shapes using the values you provide in the fields of the `gxBitmap` structure, see the description of the `GXNewBitmap` function on page 5-66.

Bitmap Shapes

ERRORS, WARNINGS, AND NOTICES

Errors

out_of_memory	
shape_is_nil	
parameter_is_nil	
size_of_bitmap_exceeds_implementation_limit	
parameter_is_nil	(debugging version)
invalid_pixelSize	(debugging version)
bitmap_height_negative	(debugging version)
bitmap_width_negative	(debugging version)
bitmap_height_negative	(debugging version)
bitmap_rowBytes_negative	(debugging version)
bitmap_rowBytes_too_small	(debugging version)
bitmap_rowBytes_not_aligned	(debugging version)
bitmap_ptr_not_aligned	(debugging version)
bitmap_rowBytes_must_be_specified_for_user_image_buffer	(debugging version)
colorSpace_out_of_range	(debugging version)

Warnings

shape_access_not_allowed	(debugging version)
--------------------------	---------------------

SEE ALSO

For examples of this function, see “Creating Black-and-White Bitmaps” beginning on page 5-15.

For information about the `gxBitmap` structure, see “The Bitmap Geometry Structure” beginning on page 5-62.

To encapsulate a bitmap geometry in a bitmap shape, use the `GXNewBitmap` function, which is described on page 5-66.

To draw a bitmap once you’ve encapsulated it in a bitmap shape, use the `GXDrawShape` function, which is described in the “Shape Objects” chapter of *Inside Macintosh: QuickDraw GX Objects*.

Checking Bitmap Colors

QuickDraw GX provides the `GXCheckBitmapColor` function to allow you to determine which pixels in a bitmap are in the gamut of a specified color space or exactly match a color in a color set.

GXCheckBitmapColor

You can use the `GXCheckBitmapColor` function to determine whether the color values in a bitmap's pixel image are in the gamut of a given color space or exactly match colors in a given color set.

```
gxShape GXCheckBitmapColor(gxShape source,
                           const gxLongRectangle *area,
                           gxColorSpace space, gxColorSet aSet,
                           gxColorProfile profile);
```

source A reference to the bitmap shape whose pixels you want to check.

area A pointer to a long rectangle specifying the area of the bitmap to check. You can specify a value of `nil` for this parameter to check the entire bitmap.

space The color space to check the pixel values of the source bitmap against. You can specify the `gxIndexedSpace` color space to indicate that you want to test the pixel values against a color set.

aSet A reference to the color set to check the pixel values of the source bitmap against.

profile A pointer to the color profile to use when checking the pixel values of the source bitmap.

function result A new bitmap shape with a pixel size of 1 bit per pixel. The value of each pixel in this bitmap indicates whether the color of the corresponding pixel in the source bitmap lies in the gamut of the specified color space (or, if you specified a color set, whether the color of the corresponding pixel exactly matches a color in that color set). If the corresponding source pixel does lie in the gamut of the color space (or match a color in the color set), the pixel value of this bitmap is set to 0, otherwise it is 1.

Bitmap Shapes

DESCRIPTION

The `GXCheckBitmapColor` function performs one of two tests on the pixels of the source bitmap:

- If you specify an indexed color space in the `space` parameter, the function determines whether the color of the pixel exactly matches any color within the color set you provide in the `aSet` parameter.
- If you specify any other color space in the `space` parameter, the function converts the pixel color to the indicated color space, using the color profile in the `profile` parameter, to determine whether the color is in the gamut represented by the color space and color profile.

If you specify `nil` as the `area` parameter, this function tests every pixel in the source bitmap's pixel image. If you provide a pointer to a long rectangle in this parameter, the function only tests the pixels that fall within the corresponding rectangular subsection of the source bitmap.

This function returns as the function result a 1 bit-per-pixel bitmap shape with a bitmap height and bitmap width corresponding to the dimensions of the `area` parameter. Each pixel in the returned bitmap is set to a value of 0 if the corresponding pixel in the source bitmap passed the test. The pixel value is 1 otherwise.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`color_is_nil`
`colorSpace_out_of_range` (debugging version)

Warnings

`colorSet_index_out_of_range` (debugging version)

SEE ALSO

For information about the `gxLongRectangle` structure, see page 5-64.

For information about colors, color spaces, color sets, color profiles, and the `GXCheckColor` function, see chapter “Colors and Color-Related Objects” in *Inside Macintosh: QuickDraw GX Objects*.

Summary of Bitmap Shapes

Constants and Data Types

The Bitmap Geometry Structure

```
typedef struct {
    char        *image;    /* pointer to the pixel image */
    long         width;     /* bitmap width */
    long         height;    /* bitmap height */
    long         rowBytes;  /* number of bytes per row */
    long         pixelSize; /* number of bits per pixel */
    gxColorSpace space;    /* color space used to interpret pixel values */
    gxColorSet   set;      /* color set to use to interpret pixel values */
    gxColorProfile profile; /* color matching information */
} gxBitmap;
```

The Long Rectangle Structure

```
struct gxLongRectangle {
    long    left;
    long    top;
    long    right;
    long    bottom;
};
```

Constants For Bitmaps With Disk-Based Pixel Images

```
#define  gxBitmapFileAliasImageValue    0x00000001

#define  gxBitmapFileAliasTagType      'bfil'
```

Bitmap Data Source Alias Structure

```
struct gxBitmapDataSourceAlias {
    unsigned long fileOffset;          /* file offset (in bytes) */
    unsigned long aliasRecordSize;     /* size of alias record */
    unsigned char aliasRecord[gxAnyNumber]; /* alias record */
};
```

Functions

Creating Bitmaps

```
gxShape GXNewBitmap          (const gxBitmap *data, const gxPoint *position);
```

Getting and Setting Bitmap Geometries

```
gxBitmap *GXGetBitmap        (gxShape source, const gxBitmap *data,  
                               const gxPoint *position);
```

```
void GXSetBitmap              (gxShape target, const gxBitmap *data,  
                               const gxPoint *position);
```

Editing Bitmaps

```
long GXGetShapePixel          (gxShape source, long x, long y, gxColor *data,  
                               long *index);
```

```
void GXSetShapePixel          (gxShape target, long x, long y,  
                               const gxColor *newColor, long newIndex);
```

```
gxShape GXGetBitmapParts      (gxShape source, const gxLongRectangle *bounds);
```

```
void GXSetBitmapParts         (gxShape target, const gxLongRectangle *bounds,  
                               gxShape bitmapShape);
```

Drawing Bitmaps

```
void GXDrawBitmap             (const gxBitmap *data, const gxPoint *position);
```

Checking Bitmap Colors

```
gxShape GXCheckBitmapColor    (gxShape source,  
                               const gxLongRectangle *area,  
                               gxColorSpace space, gxColorSet aSet,  
                               gxColorProfile profile);
```