

Geometric Operations

This chapter describes the functions that allow you to perform geometric operations on shapes. Some of the geometric operations described in this chapter work on all types of shapes. Read this chapter if you perform any kind of geometric manipulation on the shapes you create.

Before reading this chapter, you should be familiar with the QuickDraw GX object architecture as described in *Inside Macintosh: QuickDraw GX Objects*. You should also be familiar with the information in the chapters “Geometric Shapes” and “Geometric Styles” in this book.

For more information about geometric manipulation of shapes, you might want to read the chapter “Transform Objects” in *Inside Macintosh: QuickDraw GX Objects* and the chapter “QuickDraw GX Mathematics” in *Inside Macintosh: QuickDraw GX Environment and Utilities*.

This chapter introduces the basic categories of geometric operations and shows how to use these operations to

- determine and reverse the contour direction of a shape’s contours
- simplify the geometric description of a shape
- incorporate style information into a shape’s geometry
- obtain geometric information about a shape’s geometry, such as contour length and area
- determine and alter the bounding rectangle of a shape
- inset a shape’s geometry
- determine if two shapes touch
- determine if one shape contains another
- perform geometric arithmetic, such as intersection and union, on shapes

Finally, this chapter contains a complete reference for the geometric operations.

About Geometric Operations

The geometric operations allow you to obtain geometric information about geometric shapes and perform geometric calculations on them without having to manipulate shape geometries directly.

The geometric operations fall into five main categories:

- operations that affect contours and contour direction
- operations that simplify the drawing of shapes
- operations that determine and alter basic geometric information about shapes
- operations that test for intersection and inclusion
- operations that perform geometric arithmetic on shapes

The next five sections discuss these categories.

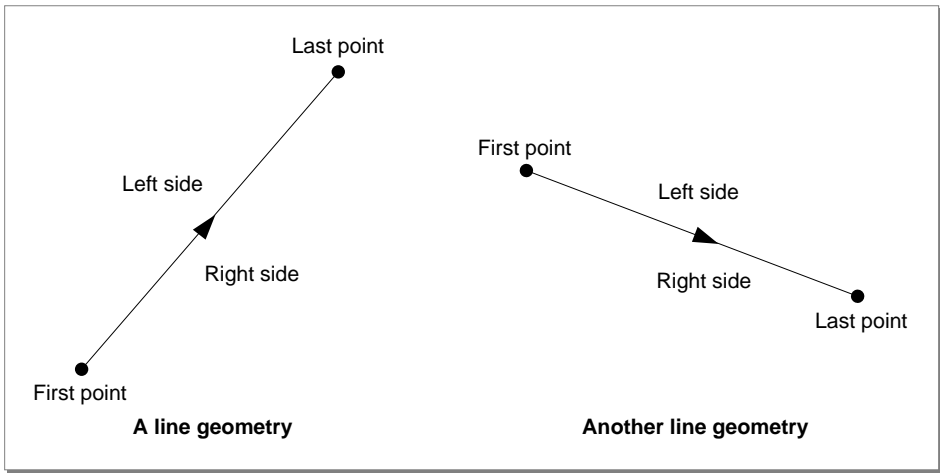
Contours and Contour Direction

With the exception of empty, full, and point shapes, geometric shapes are made up of contours. Line, curve, and rectangle shapes have a single contour, while polygon and path shapes can have zero, one, or more contours. Every contour is defined by an ordered series of on-curve or off-curve geometric points, or a combination of both.

Geometric Operations

For example, the geometry of a line shape contains two (on-curve) geometric points—a first point and a last point. The contour of a line shape is the line segment connecting these two points. Since the line has a first point and a last point, it also has a direction, a right side, and a left side, as shown in Figure 4-1.

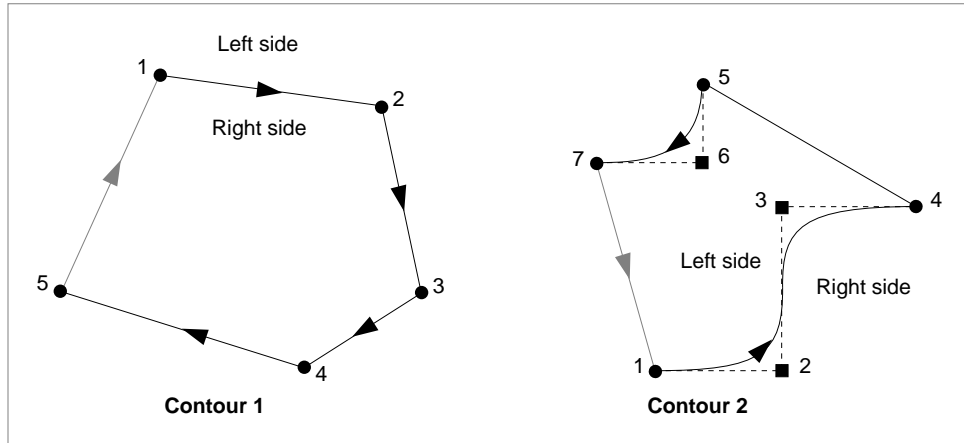
Figure 4-1 Line contours



Geometric Operations

As another example, a path shape can have multiple contours; each path contour is defined by a series of on-curve and off-curve points. As with line contours, each path contour has a direction, a right side, and a left side. Notice that the order of the geometric points decides which side is the left side and which side is the right side, as shown in Figure 4-2.

Figure 4-2 A path shape with two contours



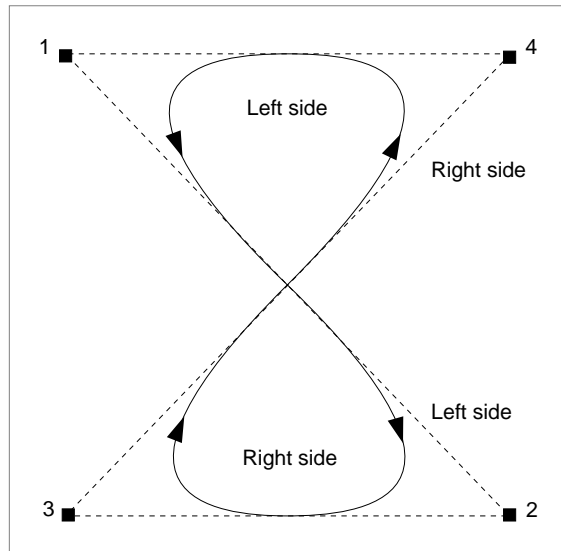
Each contour of a polygon or path shape has an implied line (or curve) connecting the last geometric point of the contour to the first geometric point of the contour. QuickDraw GX uses this implied line (or curve) when the shape fill of the polygon or path shape is the closed-frame shape fill or any of the solid shape fills. These implied lines are shown in gray in Figure 4-2.

Notice that the right side of the first contour falls inside the area enclosed by the contour and the right side of the second contour falls outside the area enclosed by the contour.

Geometric Operations

All contours have either a clockwise or a counterclockwise **contour direction**. Sometimes the contour direction of a contour is obvious, such as the contour directions of the contours in Figure 4-2. In this figure, the first contour has a clockwise contour direction and the second contour has a counterclockwise contour direction. However, sometimes the contour direction is not so obvious. Figure 4-3 gives an example.

Figure 4-3 A path whose contour direction is not immediately obvious

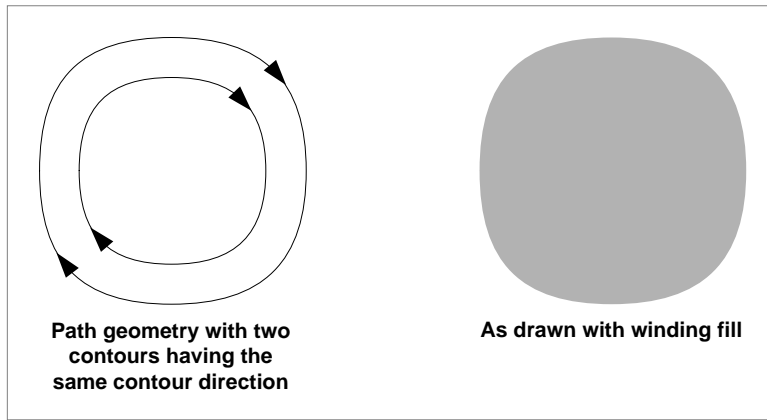


The upper half of the contour shown in Figure 4-3 seems to have a counterclockwise direction while the lower half of the contour seems to have a clockwise direction. In cases like this one, QuickDraw GX assigns an arbitrary contour direction to the entire contour. You can use the `GXGetShapeDirection` function, described on page 4-68, to find the contour direction that QuickDraw GX has assigned to a particular contour.

Geometric Operations

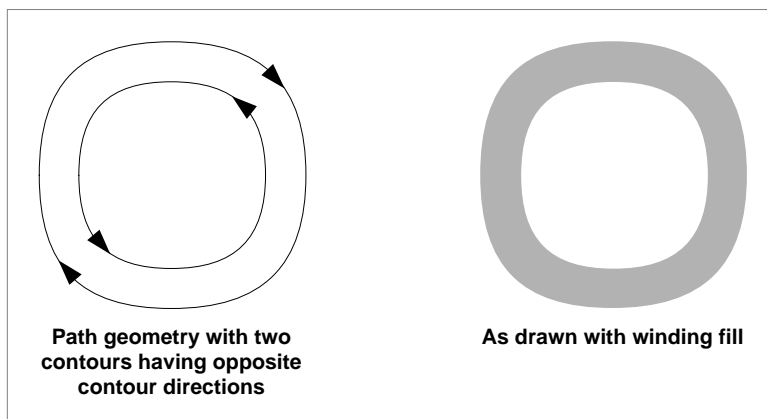
QuickDraw GX uses contour direction for a number of purposes—for example, when filling shapes that have a winding shape fill. The path shape shown in Figure 4-4 contains an inner contour with the same contour direction as the contour that surrounds it. When drawing this path using a winding fill, QuickDraw GX ignores the inner contour.

Figure 4-4 A path whose inner contour has the same contour direction as its outer contour



To indicate that QuickDraw GX should not ignore the inner contour, you could change the shape fill to even-odd fill, or you could reverse the contour direction of the inner contour, as shown in Figure 4-5.

Figure 4-5 A path shape whose inner and outer contours have different contour directions



Geometric Operations

QuickDraw GX lets you reverse a contour's direction by reversing the order of the geometric points in the contour.

Note

QuickDraw GX always considers line shapes to have a clockwise contour direction, regardless of the order of the geometric points in the line's geometry. Therefore, you cannot change the contour direction of line shapes. However, a line contour in a polygon or a path does have a clockwise or a counterclockwise direction (which QuickDraw GX assigns to it depending on the other contours in the shape); therefore, you can change the contour direction of line contours in polygons and paths. ♦

In certain situations, QuickDraw GX needs to know which side of a contour is the inside and which is the outside—for example, when drawing a geometric shape that has the inside-frame style attribute set. The default assumption is that the right side of a contour is the inside—which works well for clockwise contours but can produce surprising results with counterclockwise contours. The auto-inset style attribute indicates that QuickDraw GX should find the true inside for each contour of a shape, rather than assuming the right side is the inside. The **true inside** of a contour is defined to be the right side of the contour if the contour direction is clockwise and the left side of a contour if the contour direction is counterclockwise.

You can find more information about the inside-frame style attribute and the auto-inset style attribute in Chapter 3, “Geometric Styles,” in this book.

The section “Determining and Reversing Contour Direction” beginning on page 4-23 contains programming examples relating to contour direction.

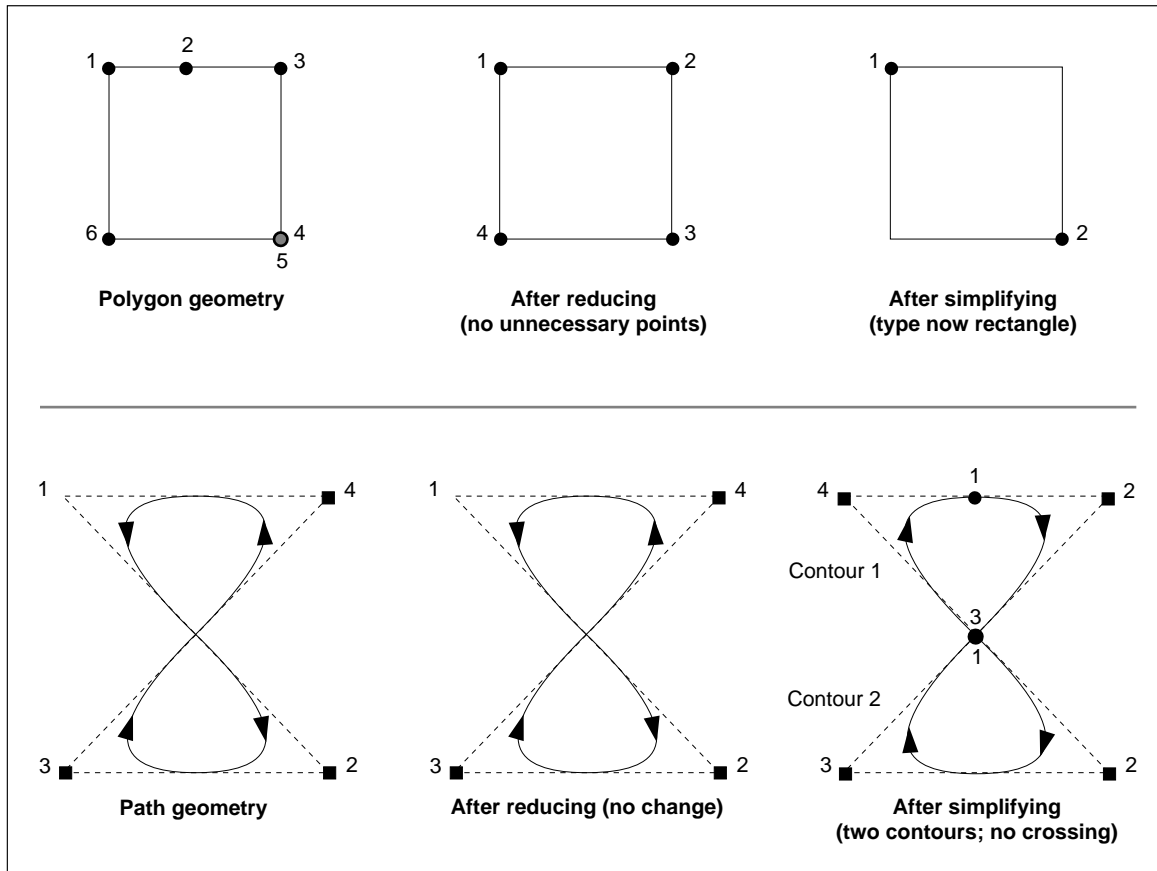
Reducing and Simplifying Shape Geometries

QuickDraw GX allows you to change shape geometries to simpler forms. You can reduce the number of geometric points in a shape by removing unnecessary ones. You can also simplify a shape's geometry by removing unnecessary contour breaks, eliminating crossed and overlapping contours, and even simplifying the shape's shape type, if possible.

Geometric Operations

Figure 4-6 shows the difference between reducing a shape's geometry and simplifying a shape's geometry.

Figure 4-6 Effects of reducing and simplifying shape geometries



Geometric Operations

In this figure, the polygon geometry has two unnecessary geometric points, which are removed in the reduced polygon. Since the polygon is actually a square, simplifying this polygon converts the polygon geometry to the simplest type of geometry necessary—which in this case is a rectangle geometry.

The path geometry in the lower part of Figure 4-6 has a crossed contour, but no unnecessary geometric points. Reducing this path results in the same path geometry, whereas simplifying this path reorders the geometric points and breaks the geometry into two path contours so that no contour crossing occurs. Also notice that, because the original path geometry starts with an off-curve control point, simplifying the path adds an initial on-curve geometric point. The new initial geometric point is halfway between the point that was originally at the end of the contour and the point that was originally at the beginning of the contour. Although adding this new complexity might not seem like a simplification, removing crossed contours does result in more predictable drawing results, as shown in Figure 4-7.

Figure 4-7 How simplifying a shape can produce more predictable results when drawing

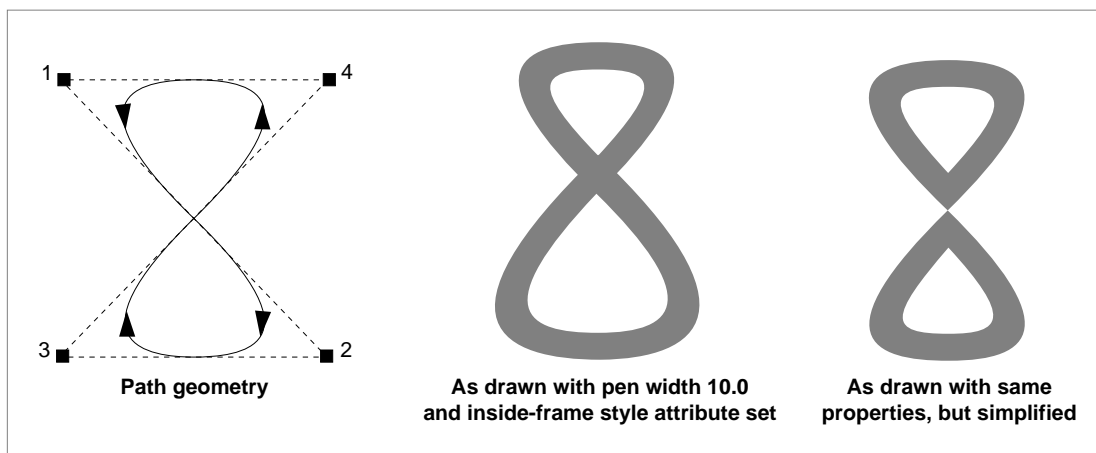


Figure 4-7 shows the path geometry from Figure 4-6. When this path is drawn with a pen width of 10.0 and the inside-frame style attribute set, the upper half of the path is inset, but the lower half of the path is outset, because of the crossed contour. Simplifying the shape uncrosses the contour, which results in both halves of the path shape being inset when drawn.

For more examples of the effect of simplifying shapes on drawing, see the section “Simplifying Shapes” beginning on page 4-33, as well as in the pen placement examples in Chapter 3, “Geometric Styles,” in this book.

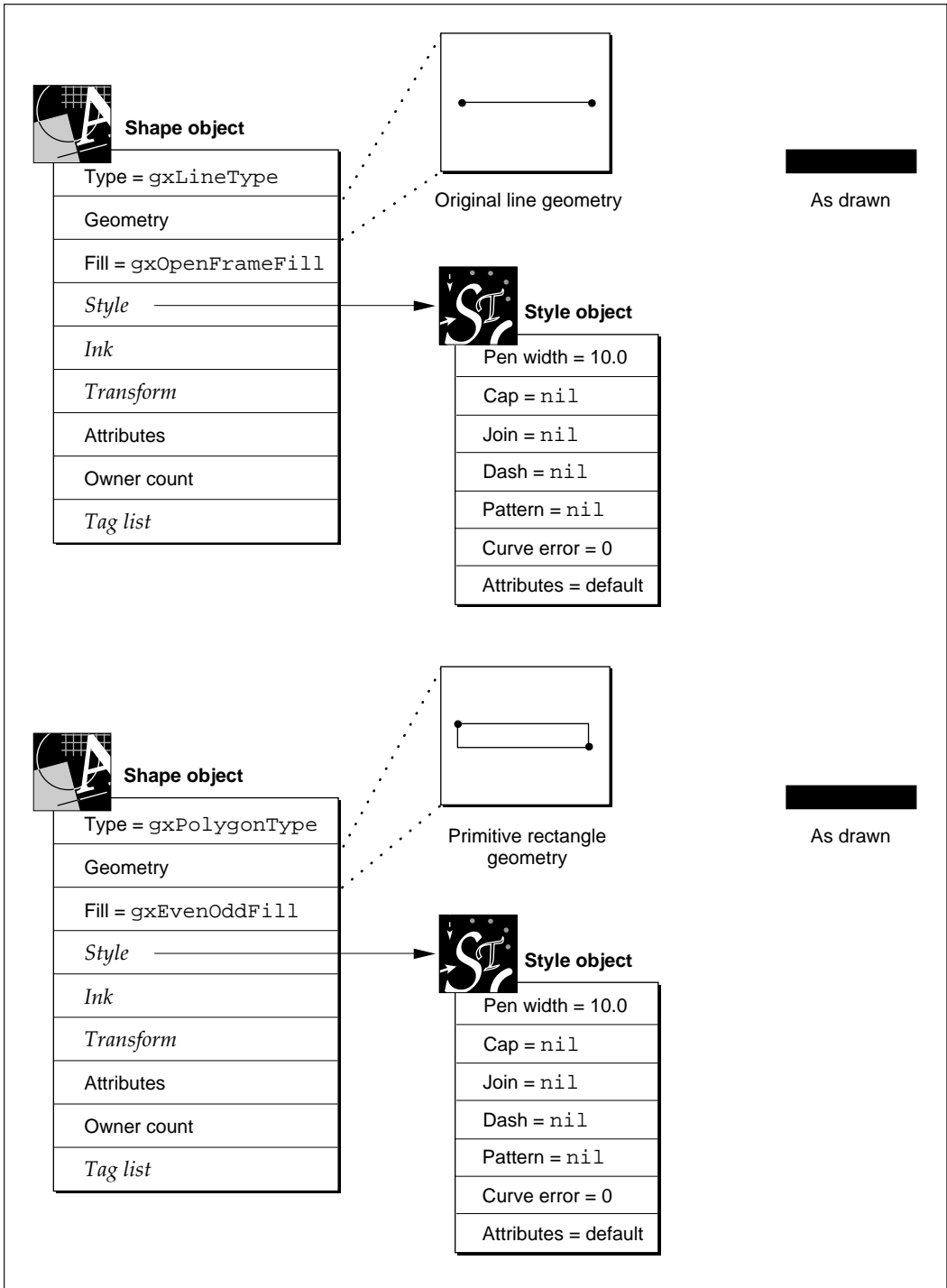
The Primitive Form of Shape Geometries

QuickDraw GX provides a mechanism for incorporating the stylistic variations contained in a style object directly into the geometry of a shape object. This mechanism is the `GXPrimitiveShape` function. When the geometry of a shape has its stylistic variations incorporated into it, it is said to be in **primitive form**. Shapes in primitive form include

- empty shapes and full shapes, which are described in Chapter 2, “Geometric Shapes”
- filled rectangle, polygon, and path shapes, which are also described in Chapter 2, “Geometric Shapes”
- hairline framed shapes, which are described in Chapter 3, “Geometric Styles”
- glyph shapes, which are described in *Inside Macintosh: QuickDraw GX Typography*

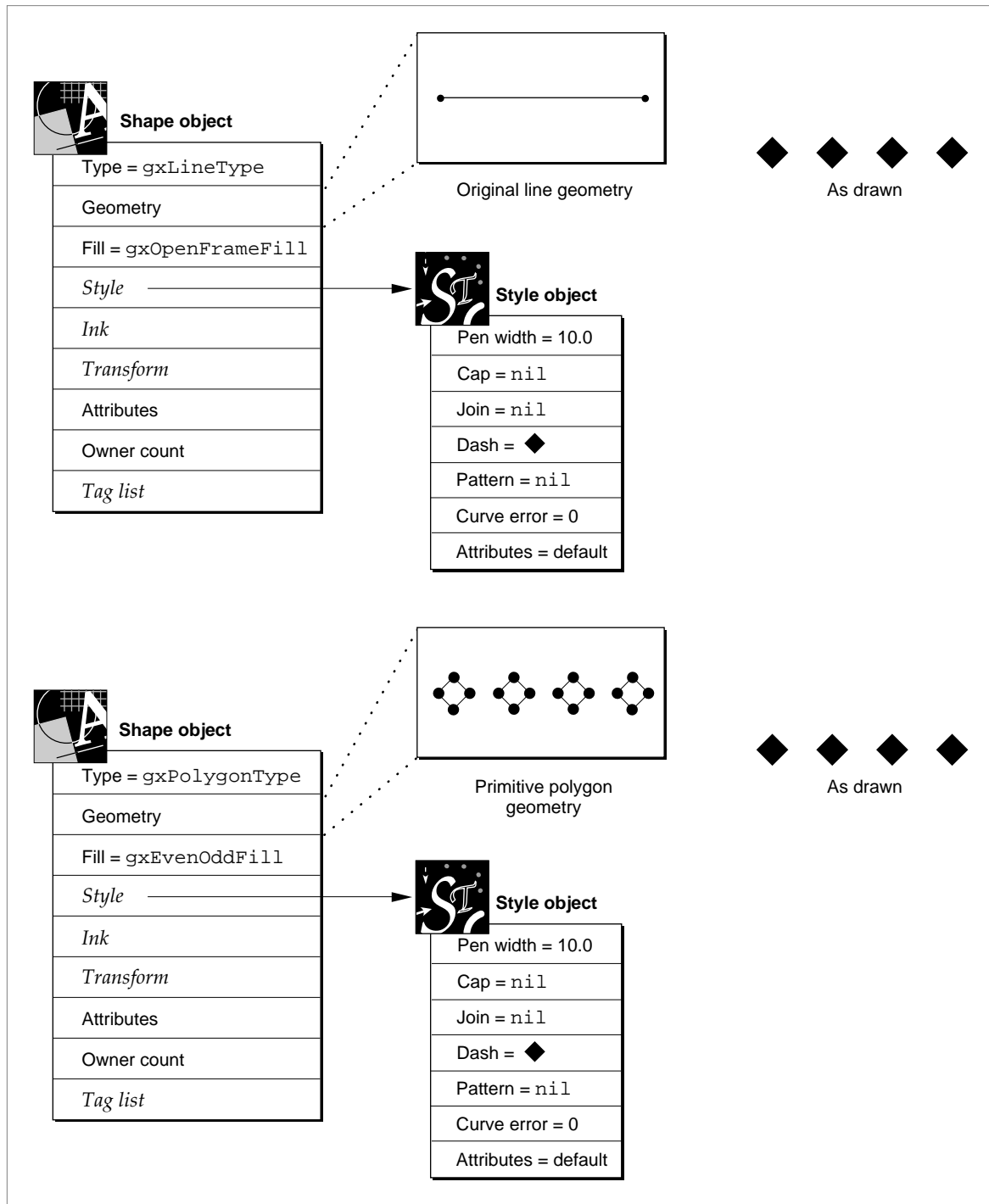
Figure 4-8 shows a simple example of the `GXPrimitiveShape` function. This figure shows a line geometry as drawn with a pen width of 10.0. Converting this line shape to its primitive form results in a rectangle shape with an even-odd fill; the pen width has been incorporated into the geometry of the shape.

Figure 4-8 Simple example of the `GXPrimitiveShape` function



Geometric Operations

Figure 4-9 shows a more involved example—a line shape dashed with diamond-shaped polygons. Converting this line shape to its primitive form results in a polygon shape with multiple contours—one contour for each dash.

Figure 4-9 More involved example of the GXPrimitiveShape function

Geometric Operations

Notice that, even though the geometry of the shape has changed significantly, the shape appears the same when drawn. Also notice that the `GXPrimitiveShape` function affects only the shape type, shape geometry, and shape fill of a shape—it does not affect the shape’s associated style object. In the example in Figure 4-9, the result of the `GXPrimitiveShape` function has a pen width of 10.0 and dash shape. However, since the shape fill was changed to even-odd fill, these aspects of the style are ignored when the shape is drawn.

For a complete description of the primitive forms of shapes, see the reference description of the `GXPrimitiveShape` function, which is on page 4-79. For some examples that demonstrate when it is necessary to use primitive shapes, see the descriptions of caps, joins, dashes, and patterns in Chapter 3, “Geometric Styles,” in this book, and the description of clip shapes in *Inside Macintosh: QuickDraw GX Objects*.

For programming examples illustrating shapes in their primitive form, see “Converting a Shape to Primitive Form” beginning on page 4-38.

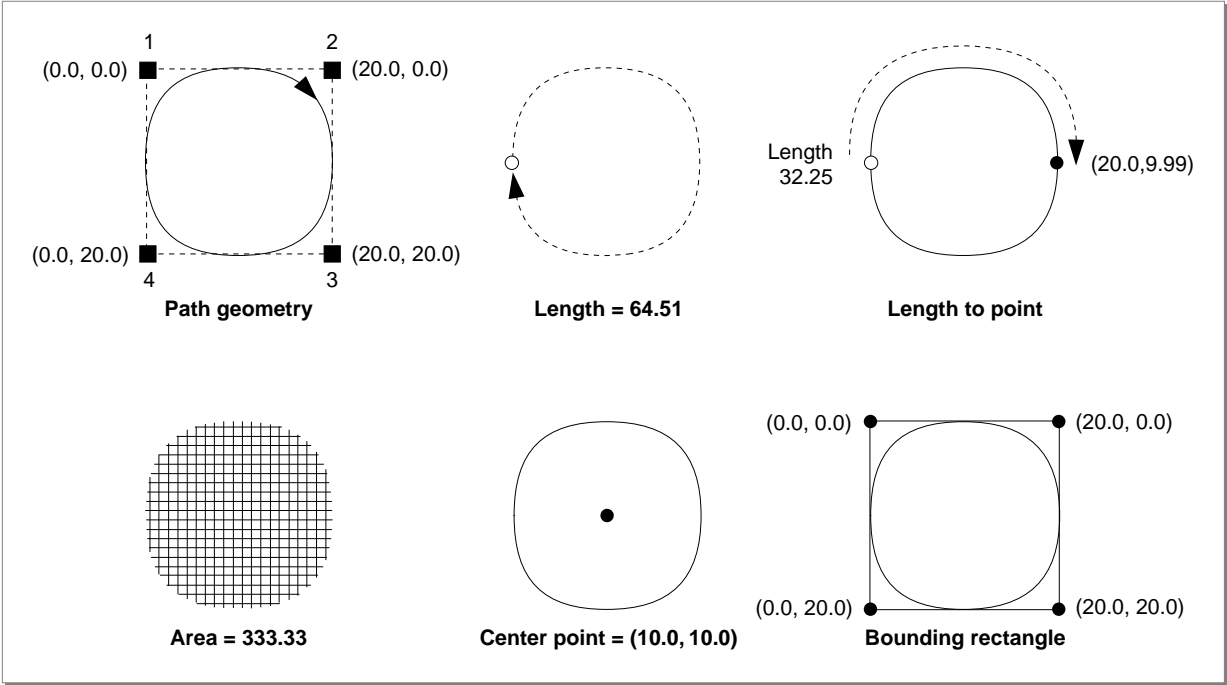
Geometric Information

QuickDraw GX lets you calculate specific geometric information about a shape, or about the contour of a shape. You can

- find the length of all of a shape’s contours or of a particular contour of a shape
- locates the point that falls at a given distance along a particular contour of a shape
- calculates the area contained by the contours of a shape’s geometry or by a particular contour of a shape’s geometry
- find the center point of a shape or of a particular contour of a shape
- find the bounding rectangle of a shape

Figure 4-10 illustrates the geometric information you can obtain about a shape.

Figure 4-10 Geometric information available about a path shape

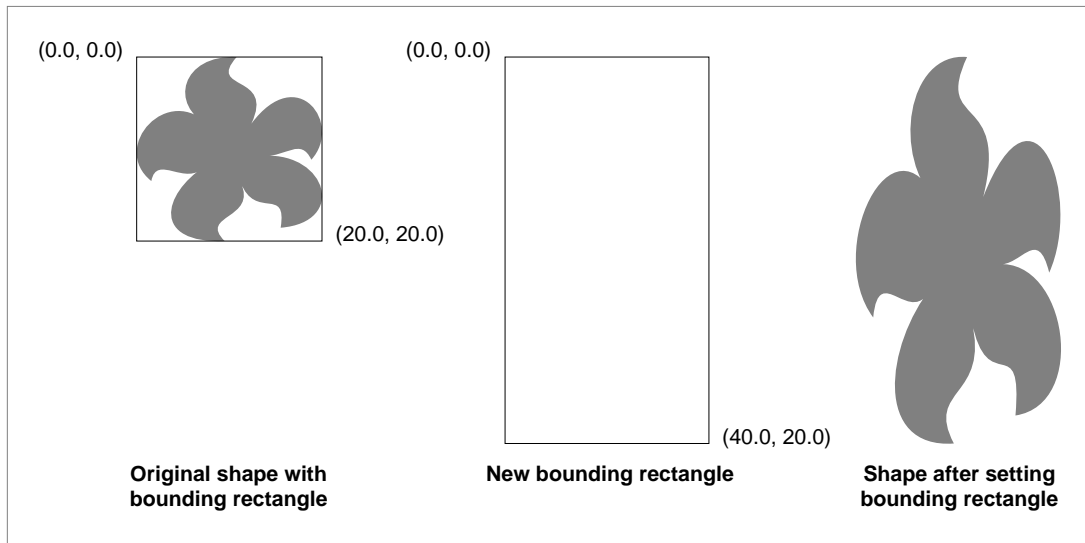


Notice in Figure 4-10 that, because the first point of the path shape is an off-curve control point, the length-to-point operation starts its calculation at an initial on-curve point, halfway between the original first and last points of the contour.

Geometric Operations

QuickDraw GX also allows you to set the bounding rectangle of a shape, and therefore move and scale the shape, as shown in Figure 4-11.

Figure 4-11 A path shape resized by changing its bounding rectangle



For programming examples of obtaining geometric information about shapes, see “Finding Geometric Information About a Shape” beginning on page 4-41.

For programming examples of setting the bounding rectangle of a shape, see “Setting a Shape’s Bounding Rectangle” beginning on page 4-47.

Touching and Containing

QuickDraw GX allows you to determine if the area enclosed by the contours of one shape touch the area enclosed by the contours of another shape. You can also determine if one shape’s area contains the area of another shape.

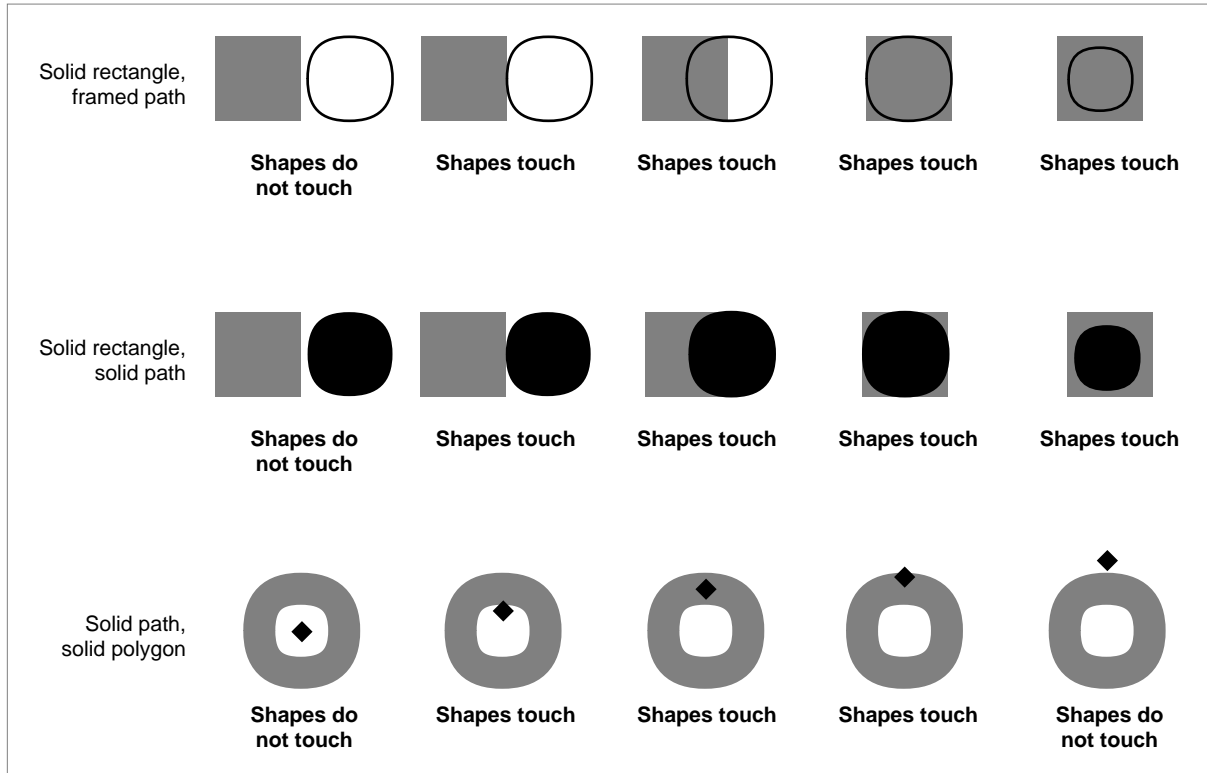
In particular, you can

- determine if a point touches the area enclosed by a rectangle
- determine if the area enclosed by the contours of a shape touches the area enclosed by a rectangle
- determine if the areas of two shapes touch

Geometric Operations

Figure 4-12 shows the results of testing to see whether pairs of different geometric shapes touch. In this figure, a solid rectangle shape is tested for touching with both a framed path and a solid path, and a solid path is tested for touching with a solid polygon.

Figure 4-12 Testing whether one shape touches another



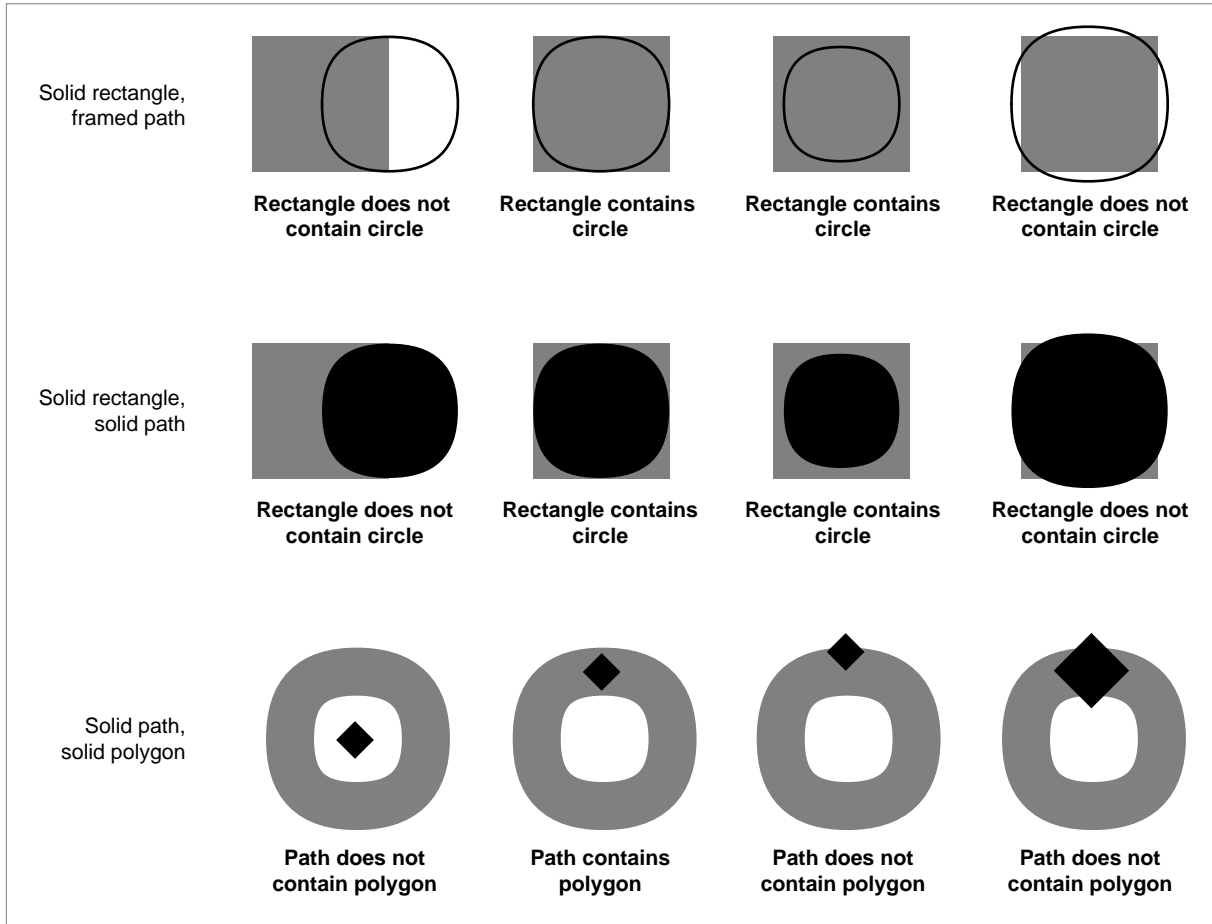
QuickDraw GX also allows you to determine whether or not

- one rectangle contains another
- a rectangle contains the area covered by a shape
- the area covered by one shape contains the area covered by another shape.

Geometric Operations

Figure 4-13 shows the results of testing pairs of shapes to see if one shape contains another.

Figure 4-13 Testing whether one shape contains another



Notice the first diagram in the third row of Figure 4-13. A shape does not contain another shape if it merely surrounds the other shape; the area covered by the first shape as drawn must contain the area of the second shape as drawn.

Note

QuickDraw GX defines empty shapes as touching no shapes and full shapes as touching any shape except an empty shape. QuickDraw GX also defines full shapes as containing any shape and empty shapes as being contained by any shape except other empty shapes. ♦

Geometric Operations

For programming examples of testing shapes for intersection, see “Determining Whether Two Shapes Touch” beginning on page 4-53.

For programming examples of testing shapes for inclusion, see “Determining Whether One Shape Contains Another” beginning on page 4-58.

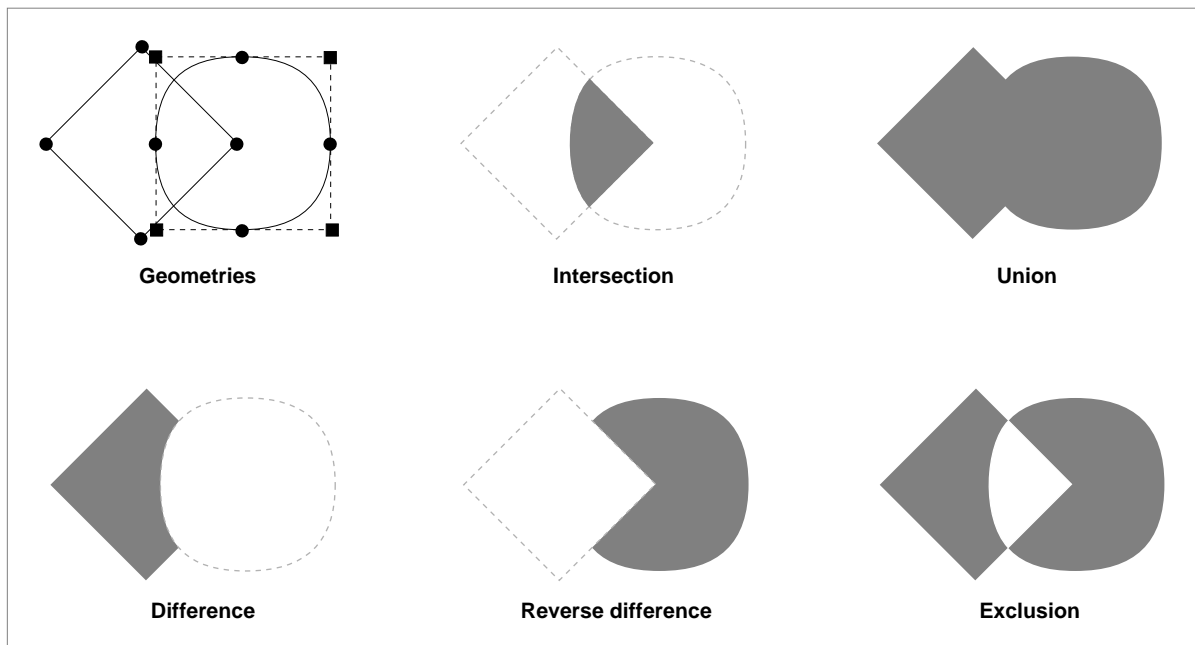
Geometric Arithmetic

QuickDraw GX provides six different arithmetic operations that you can perform on geometric shapes. These operations are: intersection, union, difference, reverse difference, exclusion, and inversion. With these operations, you can

- find the intersection of two rectangles
- find the union of two rectangles
- find the area common to two shapes
- find the combined area of by two shapes
- find the area covered by one shape that is not also covered by another
- find the area covered by one shape or another, but not both
- find the area not covered by a shape

Figure 4-14 illustrates the first five of these arithmetic operations.

Figure 4-14 Geometric arithmetic with two solid shapes



Geometric Operations

Figure 4-14 shows geometric arithmetic with two solid shapes. You can also perform some geometric arithmetic on a filled shape and a solid shape, as shown in Figure 4-15.

Figure 4-15 Geometric arithmetic with a framed shape and a solid shape

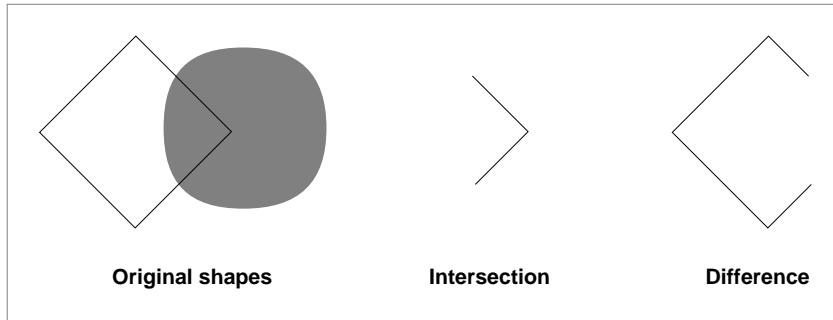
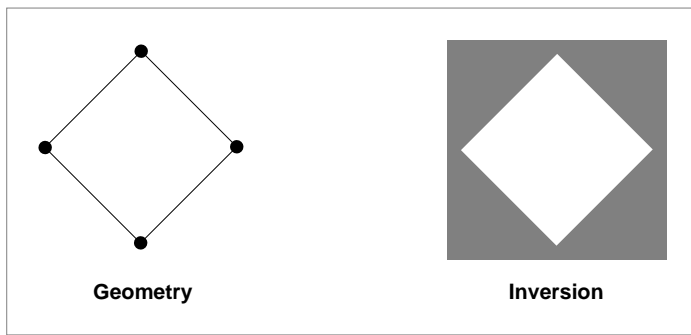


Figure 4-16 illustrates the geometric inversion—the area not covered by a shape. The inverted shape extends to the limits of its clip shape or the limits of the view port to which it is drawn.

Figure 4-16 Geometric inversion



For programming examples of geometric arithmetic, see “Performing Geometric Arithmetic With Shapes” beginning on page 4-60.

Using Geometric Operations

This section shows you how to apply geometric operations to shapes. In particular, this section shows you how to

- determine and reverse the contour direction of a shape's contours
- break a contour into multiple contours
- reduce and simplify the geometric description of a shape
- incorporate style information into a shape's geometry
- obtain geometric information about a shape's geometry, such as contour length and area
- determine and alter the bounding rectangle of a shape
- inset a shape's geometry
- determine if two shapes touch
- determine if one shape contains another
- perform geometric arithmetic, such as intersection and union, on shapes

Many of the sample functions in this section create geometric shapes, and to do so, they specify geometric points for the shapes' geometries. Since a geometric point contains two fixed-point values, the sample functions in this section must convert integer constants to fixed-point constants when specifying a geometric point. QuickDraw GX provides the `GXIntToFixed` macro, which performs this conversion by shifting the integer value 16 bits to the left:

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

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

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

The `ff` macro is used throughout this section.

Determining and Reversing Contour Direction

The contours of geometric shapes have contour direction: either clockwise or counterclockwise, as described in "Contours and Contour Direction" beginning on page 4-4. QuickDraw GX allows you to determine the contour direction of a specific contour of a shape and also allows you to change the direction of a shape's contour by reversing the order of the geometric points in the geometry defining the contour.

Geometric Operations

The sample function in Listing 4-1 creates a polygon shape with two contours—one having a clockwise contour direction and the other having a counterclockwise contour direction.

Listing 4-1 Creating a polygon shape with two contours having opposite contour directions

```
void CreateConcentricTriangles(void)
{
    gxShape twoTriangles;

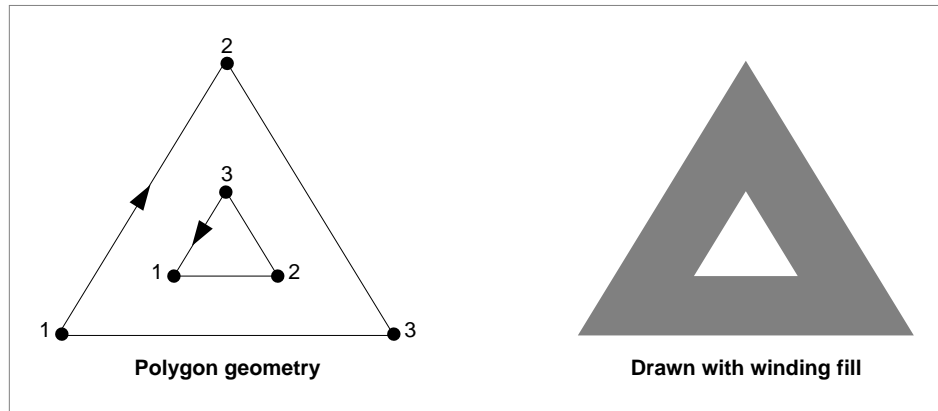
    long twoTrianglesGeometry[] = {2, /* number of contours */
                                    3, /* number of points */
                                    ff(50), ff(200),
                                    ff(110), ff(100),
                                    ff(170), ff(200),
                                    3, /* number of points */
                                    ff(90), ff(178),
                                    ff(130), ff(178),
                                    ff(110), ff(145)};

    twoTriangles = GXNewPolygons((gxPolygons *)
                                twoTrianglesGeometry);
    GXSetShapeFill(twoTriangles, gxWindingFill);

    GXDrawShape(twoTriangles);
    GXDisposeShape(twoTriangles);
}
```

The result of this sample function is shown in Figure 4-17.

Figure 4-17 A polygon shape whose two contours have opposite contour directions



QuickDraw GX provides the `GXGetShapeDirection` function to allow you to determine the contour direction of a specific contour in a shape. This function takes two parameters: the first parameter is a reference to the shape and the second parameter is the index of the contour whose contour direction you want to find. In the example from Listing 4-1, the first contour (the outer contour) has a clockwise contour direction. Calling the function

```
GXGetShapeDirection(twoTriangles, 1);
```

returns the constant `gxClockwiseDirection`.

The second contour (the inner contour) has a counterclockwise direction. Calling the function

```
GXGetShapeDirection(twoTriangles, 2);
```

returns the constant `gxCounterclockwiseDirection`.

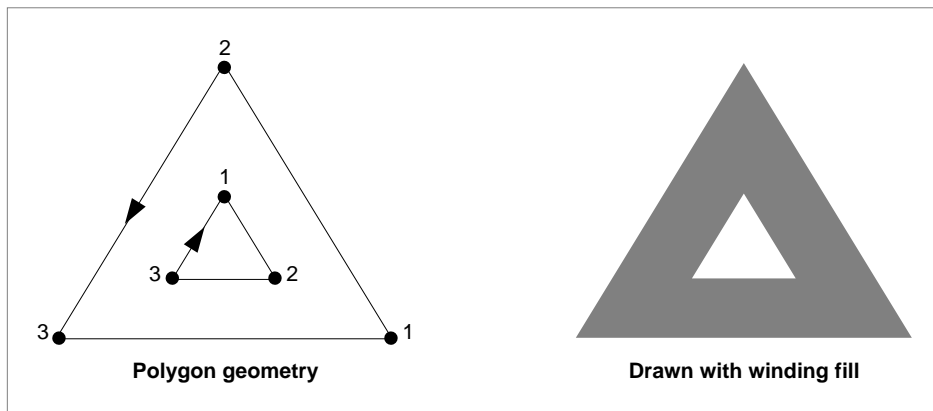
Geometric Operations

You can reverse the direction of a contour by reversing the order of the contour's geometric points. For this purpose, QuickDraw GX provides the `GXReverseShape` function. This function also takes two parameters: a reference to the shape and the index of the contour to reverse. Specifying 0 as the number of the contour to reverse causes the `GXReverseShape` function to reverse all the contours of a shape. For example, you can add the following function call to the sample function in Listing 4-1:

```
GXReverseShape(twoTriangles, 0);
```

The result is shown in Figure 4-18.

Figure 4-18 A polygon shape with the direction of both contours reversed



Geometric Operations

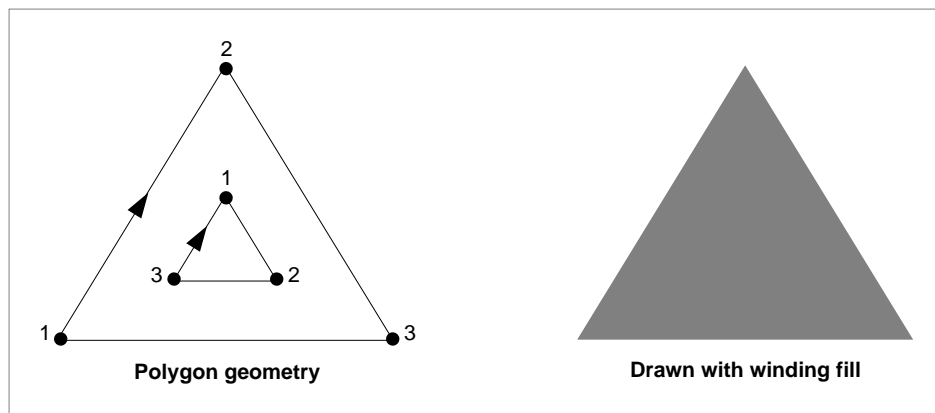
Since both contours are reversed in this example, the shape appears the same when drawn as it did before the contours were reversed.

However, reversing only the inner contour of this polygon by calling

```
GXReverseShape(twoTriangles, 2);
```

results in the polygon shown in Figure 4-19.

Figure 4-19 A polygon shape with the direction of the inner contour reversed



Reversing the contour of a shape by calling the `GXReverseShape` function almost always changes the result of the `GXGetShapeDirection` function. One important exception, however, is that line shapes always have a clockwise direction. The order of a line shape's geometric points does not affect the result of the `GXGetShapeDirection` function.

For a discussion of contour direction, see “Contours and Contour Direction” beginning on page 4-4.

For more information about the `GXGetShapeDirection` function, see page 4-68. For more information about the `GXReverseShape` function, see page 4-70.

Breaking Shape Contours

Polygon and path shapes can contain many contours. Each contour of a polygon shape can be made up of many lines and each contour of a path shape can be made up of many lines and curves.

QuickDraw GX provides a method for breaking a single contour of a polygon or path shape into two contours at a specified geometric point in the original contour.

As an example, the sample function in Listing 4-2 creates a path shape with a single contour. This contour contains six geometric points and is made up of a curve, a line, and another curve.

Listing 4-2 Creating a path shape with a single contour

```
void CreateSingleContourPath(void)
{
    gxShape  aPathShape;

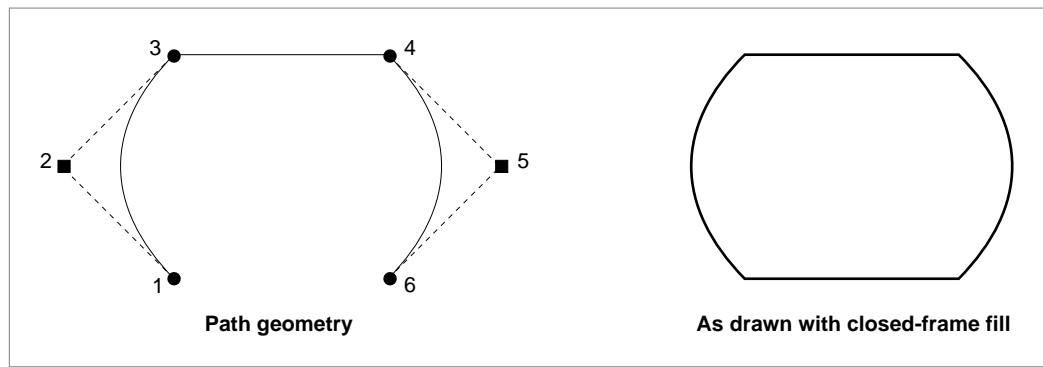
    static long oneContourGeometry[] = {1, /* number of contours */
                                         6, /* number of points */
                                         0x48000000, /* 0100 1000 */
                                         ff(100), ff(150), /* on */
                                         ff(50), ff(100), /* off */
                                         ff(100), ff(50), /* on */
                                         ff(200), ff(50), /* on */
                                         ff(250), ff(100), /* off */
                                         ff(200), ff(150)}; /* on */

    aPathShape = GXNewPaths((gxPaths *) oneContourGeometry);
    GXSetShapeFill(aPathShape, gxClosedFrameFill);

    GXDrawShape(aPathShape);
    GXDisposeShape(aPathShape);
}
```

The result of this function is shown in Figure 4-20.

Figure 4-20 A path shape with a single contour

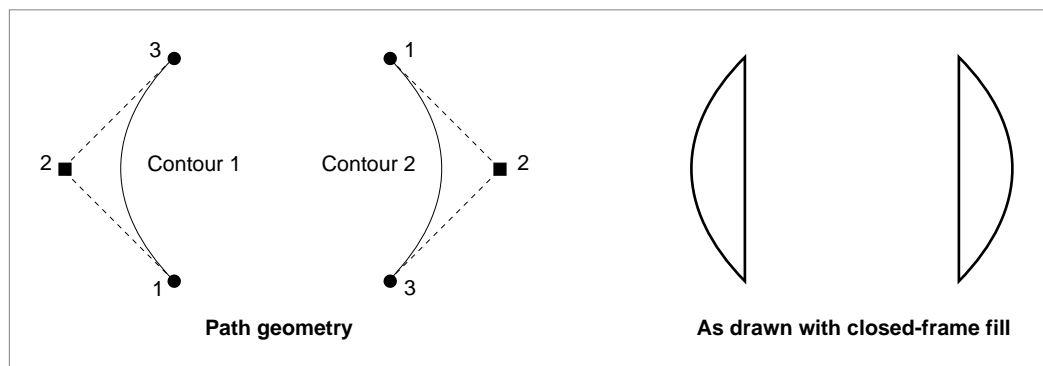


The `GXBreakShape` function allows you to break a single contour into two contours at a specified geometric point. Adding the function call

```
GXBreakShape(aPathShape, 4);
```

to the sample function in Listing 4-2 breaks the single contour of the path shape into two contours at the fourth geometric point, as shown in Figure 4-21.

Figure 4-21 A path shape broken into two contours



Geometric Operations

After the call to the `GXBreakShape` function, the path shape has two contours, each with three geometric points. Calling the function

```
GXCountShapeContours(aPathShape);
```

returns the value 2.

In addition to breaking the contours of polygon and path shapes, you can also use the `GXBreakShape` function to break line shapes and curve shapes. For example, if the variable `aLine` references a line shape, the function call

```
GXBreakShape(aLine, 1);
```

converts the line shape to a polygon shape with two contours. The first contour is empty (that is, it has no geometric points) and the second contour is the original line. Calling the function

```
GXCountShapeContours(aLine);
```

returns the value 2.

For a discussion of contours, geometric points, and the `GXCountShapeContours` function, see Chapter 2, “Geometric Shapes,” in this book.

You can also use the `GXSetPolygonParts`, `GXSetPathParts`, and `GXSetShapeParts` functions to break a shape’s contours. These functions are also described in Chapter 2, “Geometric Shapes,” in this book.

For more information about the `GXBreakShape` function, see page 4-72.

Eliminating Unnecessary Geometric Points

There are many ways in which polygon and path shapes can contain more geometric points than necessary to describe their underlying geometry. Two common examples are

- duplicate points—sequential points with the same coordinates
- colinear points—points that lie in a straight line between a preceding point and subsequent point

Geometric Operations

The sample function in Listing 4-3 creates a polygon shape with a single contour that has six geometric points, two of which are unnecessary.

Listing 4-3 Creating a polygon with redundant geometric points

```
void ReduceUnnecessaryPoints(void)
{
    gxShape squareShape;

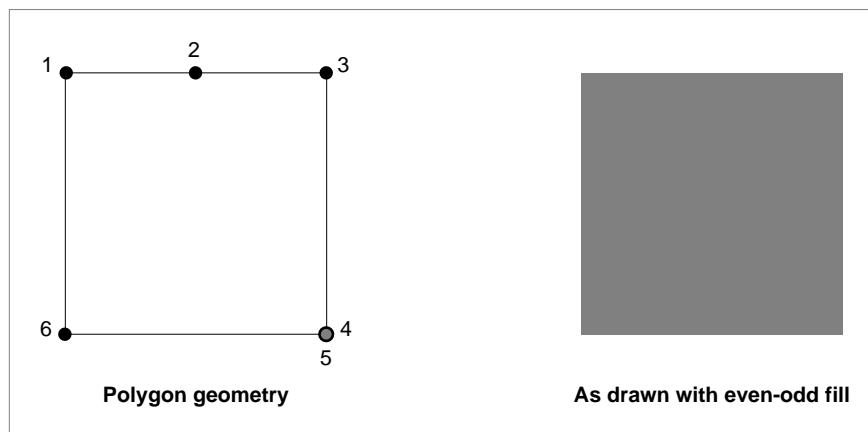
    static long paddedSquareGeometry[] = {1, /* # of contours */
                                           6, /* # of contours */
                                           ff(50), ff(50),
                                           ff(100), ff(50),
                                           ff(150), ff(50),
                                           ff(150), ff(150),
                                           ff(150), ff(150),
                                           ff(50), ff(150)};

    squareShape = GXNewPolygons((gxPolygons *)
                                &paddedSquareGeometry);
    GXSetShapeFill(squareShape, gxEvenOddFill);

    GXDrawShape(squareShape);
    GXDisposeShape(squareShape);
}
```

The resulting polygon shape is shown in Figure 4-22.

Figure 4-22 A polygon shape with unnecessary geometric points



Geometric Operations

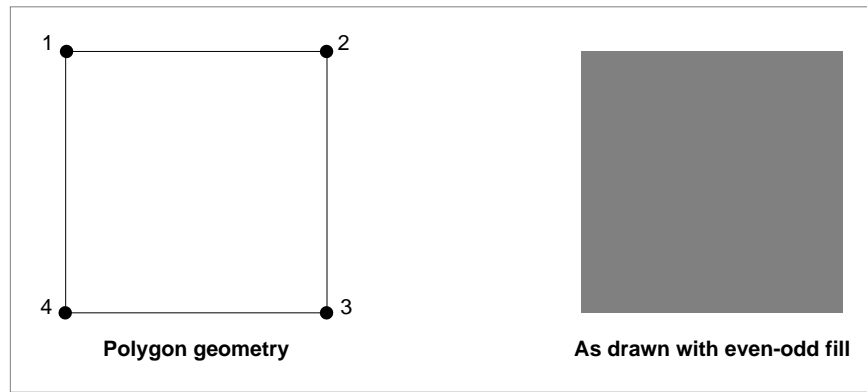
QuickDraw GX provides the `GXReduceShape` function so you can eliminate unnecessary duplicate and colinear points. The `GXReduceShape` function takes two parameters: a reference to the shape containing the contour whose unnecessary geometric points you want to eliminate and an index specifying the contour itself. If you supply the value 0 for the second parameter, the `GXReduceShape` function eliminates unnecessary geometric points from all the contours of a shape.

As an example, adding the function call

```
GXReduceShape(squareShape, 0);
```

to the sample function in Listing 4-3 results in the polygon shape shown in Figure 4-23.

Figure 4-23 A polygon shape with the unnecessary geometric points removed



The unnecessary duplicate geometric point and the unnecessary colinear geometric point are gone, but the polygon still appears the same when drawn. Although the resulting geometry could be described by a rectangle shape, the shape in this example remains a polygon shape. The `GXReduceShape` function does not convert the shape type of the original shape. (However, the `GXSimplifyShape` function, shown in the next section, does convert shape type, when possible.)

The `GXReduceShape` function considers two points to be duplicate points if they are within the distance from each other specified by the curve error property of the shape's style object. See Chapter 3, "Geometric Styles," in this book for a discussion of curve error.

For a discussion of geometric points, see Chapter 2, "Geometric Shapes," in this book.

For more information about the `GXReduceShape` function, see page 4-74.

Simplifying Shapes

In addition to unnecessary geometric points, there are other aspects of shape geometries that complicate the definition and drawing of a shape. Some examples are:

- an unnecessary contour break, where an open-framed contour ends on the same point where the subsequent contour begins
- a crossed contour, where a contour crosses over itself or another contour of the same shape
- overlapping contours, where inner contour loops have the same contour direction as the contour that contains them

The sample function in Listing 4-4 creates a polygon shape with a single contour that crosses over itself.

Listing 4-4 Creating a polygon shape with a crossed contour

```
void CreateHourglassPolygon(void)
{
    gxShape aPolygonShape;

    static long hourglassGeometry[] = {1, /* number of contours */
                                       4, /* number of points */
                                       ff(50), ff(50),
                                       ff(150), ff(50),
                                       ff(50), ff(150),
                                       ff(150), ff(150)};

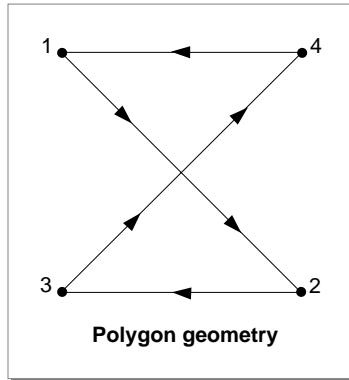
    aPolygonShape = GXNewPolygons((gxPolygons *)
                                   hourglassGeometry);
    GXSetShapeFill(aPolygonShape, gxClosedFrameFill);

    GXDrawShape(aPolygonShape);
    GXDisposeShape(aPolygonShape);
}
```

Geometric Operations

The resulting polygon shape is shown in Figure 4-24.

Figure 4-24 A polygon shape with a crossed contour



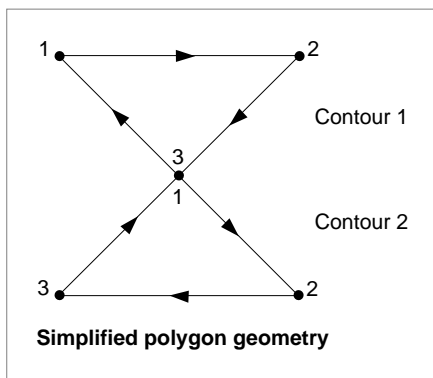
QuickDraw GX provides the `GXSimplifyShape` function so you can eliminate unnecessary contour breaks, crossed contours, and overlapping contours. This function takes one parameter: a reference to the shape you want to simplify.

As an example, adding the function call

```
GXSimplifyShape(aPolygonShape);
```

to the sample function in Listing 4-4 creates the polygon shown in Figure 4-25.

Figure 4-25 A polygon shape with no crossed contours



Geometric Operations

Notice that although this polygon shape is simplified, it contains more geometric points and more contours than the original polygon. However, the crossed contour is eliminated.

As another example, the sample function in Listing 4-5 creates a path shape with two concentric contours: an outer contour and an inner contour, both of which have a clockwise contour direction.

Listing 4-5 Creating a path shape with two clockwise contours

```
void CreateConcentricPaths(void)
{
    gxShape aPathShape;

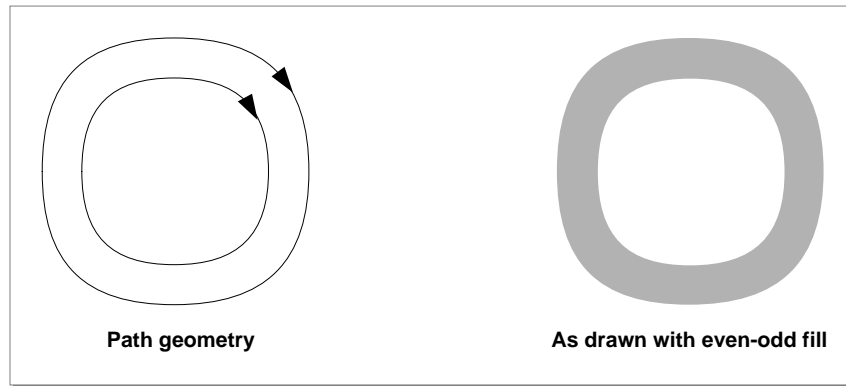
    static long twoCircleGeometry[] = {2, /* # of contours */
                                        4, /* # of points */
                                        0xF0000000, /* 1111 ... */
                                        ff(50), ff(50), /* off */
                                        ff(150), ff(50), /* off */
                                        ff(150), ff(150), /* off */
                                        ff(50), ff(150), /* off */
                                        4, /* # of points */
                                        0xF0000000, /* 1111 ... */
                                        ff(65), ff(65), /* off */
                                        ff(135), ff(65), /* off */
                                        ff(135), ff(135), /* off */
                                        ff(65), ff(135)}; /* off */

    aPathShape = GXNewPaths((gxPaths *) twoCircleGeometry);
    GXSetShapeFill(aPathShape, gxEvenOddFill);

    GXDrawShape(aPathShape);
    GXDisposeShape(aPathShape);
}
```

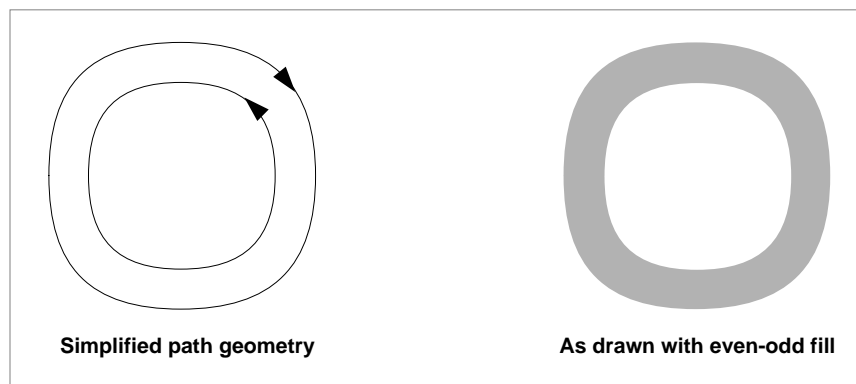
Figure 4-26 shows the result of this sample function.

Figure 4-26 A path shape with two concentric clockwise contours and even-odd shape fill



Applying the `GXSimplifyShape` function to the path shape in Figure 4-26 reverses the contour direction of the inner contour, so that it is no longer an overlapping contour with the same contour direction. The result is shown in Figure 4-27.

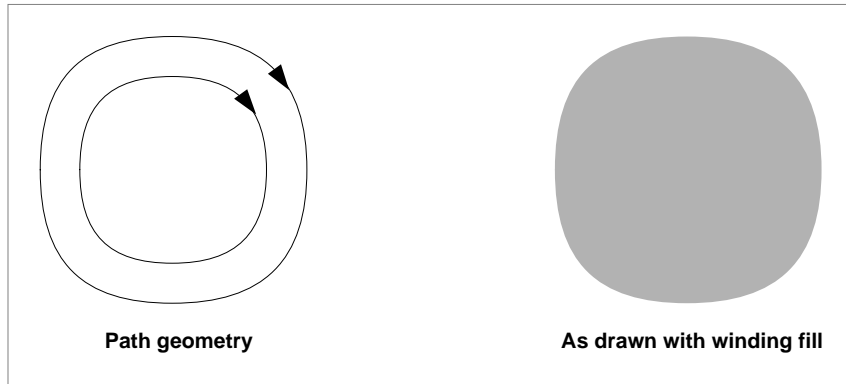
Figure 4-27 A path shape with two concentric contours with opposite contour direction



Geometric Operations

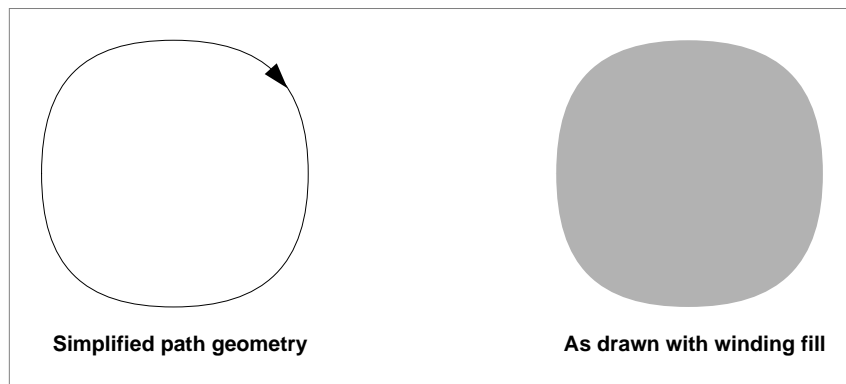
However, imagine that the path shape defined in Listing 4-5 originally had a winding fill, as shown in Figure 4-28.

Figure 4-28 A path shape with two concentric clockwise contours and winding shape fill



In this case, the `GXSimplifyShape` function removes the inner contour entirely, as it is not necessary to describe the shape as drawn. The result is shown in Figure 4-29.

Figure 4-29 A path shape simplified to a single clockwise contour



The `GXSimplifyShape` function can change the shape type of a shape, as well the geometry of shape, if the shape can be expressed by a simpler shape type. For example, a polygon shape is converted to a rectangle shape or a line shape, if possible. Similarly, a path shape is converted to a polygon shape if it has no off-curve control points.

For a discussion of shape fills and contour direction, see Chapter 2, “Geometric Shapes,” in this book.

For more information about the `GXSimplifyShape` function, see page 4-76.

Converting a Shape to Primitive Form

QuickDraw GX requires that certain shapes (such as cap shapes, join shapes, dash shapes, pattern shapes, and clip shapes) be in primitive form—that is, they must have all of their style modifications incorporated into their geometries. Before you set a cap shape, join shape, dash shape, and so on, you must ensure that the shape is in primitive form.

As an example of converting a shape to primitive form, the sample function in Listing 4-6 creates a polygon shape that is not in primitive form. The polygon has one contour, a closed-frame shape fill, and a pen width of 15.0.

Listing 4-6 Creating an hourglass polygon shape with a thick pen width

```
void CreateHourglassPolygon(void)
{
    gxShape aPolygonShape;
    gxJoinRecord theJoinRecord;

    static long hourglassGeometry[] = {1, /* number of contours */
                                        4, /* number of points */
                                        ff(50), ff(50),
                                        ff(150), ff(50),
                                        ff(50), ff(150),
                                        ff(150), ff(150)};

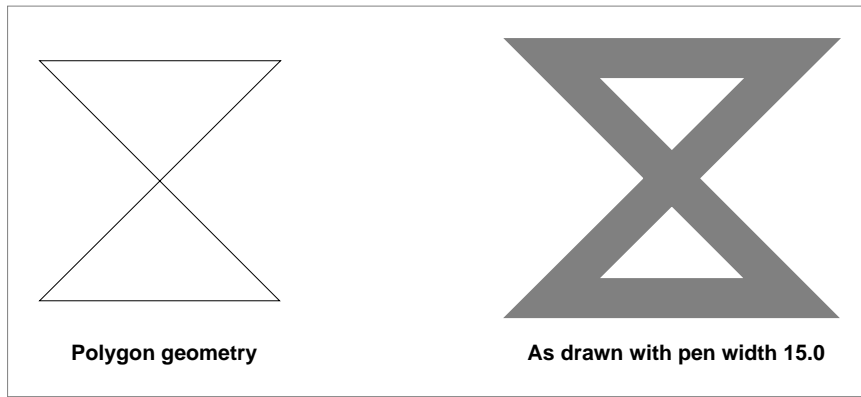
    aPolygonShape = GXNewPolygons((gxPolygons *)
                                  hourglassGeometry);
    GXSetShapeFill(aPolygonShape, gxClosedFrameFill);
    GXSetShapePen(aPolygonShape, ff(15));

    theJoinRecord.attributes = gxSharpJoin;
    theJoinRecord.join = nil;
    theJoinRecord.miter = gxPositiveInfinity;
    GXSetShapeJoin(aPolygonShape, &theJoinRecord);

    GXDrawShape(aPolygonShape);
    GXDisposeShape(aPolygonShape);
}
```

The result of this sample function is shown in Figure 4-30.

Figure 4-30 A hourglass-shaped polygon with a thick border



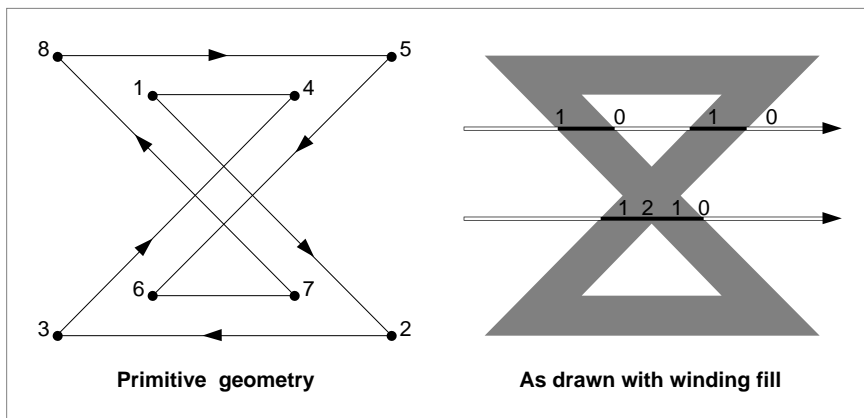
The polygon shape defined in Listing 4-6 is not in primitive form because an element of the polygon's style (the pen width) is not incorporated into the polygon's geometry.

QuickDraw GX provides the `GXPrimitiveShape` function so you can incorporate the elements of a shape's style into the shape's geometry. For example, adding the function call

```
GXPrimitiveShape(aPolygonShape);
```

to the sample function in Listing 4-6 creates the polygon shown in Figure 4-31.

Figure 4-31 A polygon shape with style information incorporated into its geometry



Geometric Operations

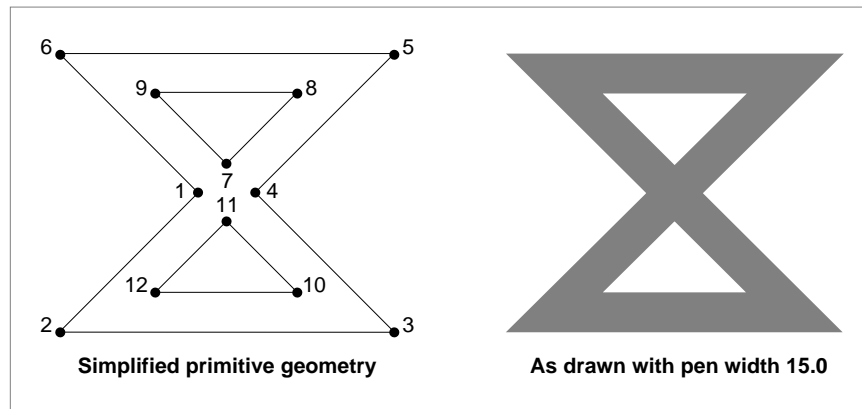
As shown in Figure 4-31, the `GXPrimitiveShape` function has incorporated the pen width into the geometry of the polygon; the resulting polygon has two contours whereas the original had one, and the resulting polygon has a winding fill instead of a closed-frame fill.

Notice that the primitive form of this polygon is not simplified because the `GXPrimitiveShape` function does not simplify its result. You can simplify the result of this function by calling

```
GXSimplifyShape(aPolygonShape);
```

Figure 4-32 shows the resulting shape—the primitive form of the polygon with no crossed or overlapping contours.

Figure 4-32 The primitive form of the polygon shape after simplification



The polygon shape now has three contours—which do not cross or overlap—and yet it appears the same as the original polygon shape when drawn.

For a discussion of style modifications and more examples of the `GXPrimitiveShape` function, see Chapter 3, “Geometric Styles,” in this book.

For more information about the `GXPrimitiveShape` function, see page 4-79.

Finding Geometric Information About a Shape

QuickDraw GX provides a number of functions that allow you to determine geometric information about a shape, such as the length of a contour or the area covered by a shape.

The sample function in Listing 4-7 creates a path shape with two concentric contours, the outer contour having a clockwise contour direction and the inner contour having a counterclockwise contour direction. This path shape is used in subsequent sections to illustrate the geometric information functions.

Listing 4-7 Creating a path shape with two contours having opposite contour directions

```
void CreateConcentricCircles(void)
{
    gxShape aPathShape;

    static long twoCircleGeometry[] = {2, /* number of contours */
                                        4, /* number of points */
                                        0xF0000000, /* 1111 ... */
                                        ff(50), ff(50), /* off */
                                        ff(150), ff(50), /* off */
                                        ff(150), ff(150), /* off */
                                        ff(50), ff(150), /* off */
                                        4, /* number of points */
                                        0xF0000000, /* 1111 ... */
                                        ff(65), ff(135), /* off */
                                        ff(135), ff(135), /* off */
                                        ff(135), ff(65), /* off */
                                        ff(65), ff(65)}; /* off */

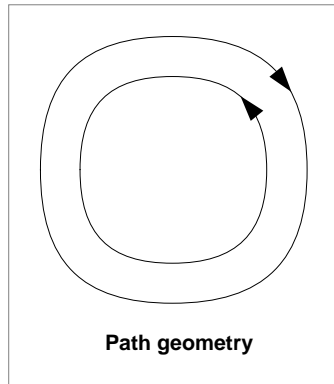
    aPathShape = GXNewPaths((gxPaths *) twoCircleGeometry);

    GXDrawShape(aPathShape);
    GXDisposeShape(aPathShape);
}
```

Geometric Operations

The resulting shape geometry is shown in Figure 4-33.

Figure 4-33 A path with an outer clockwise contour and an inner counterclockwise contour



Finding the Length of a Contour

QuickDraw GX provides the `GXGetShapeLength` function so you can measure the length of a contour. This function takes three parameters: a reference to the shape containing the contour you want to measure, an index indicating which contour you want to measure, and a pointer to a variable of type `gxWide` to store the result.

For example, if you add the declaration

```
gxWide length;
```

and the function call

```
GXGetShapeLength(aPathShape, 1, &length);
```

to the sample function in Listing 4-7, the value returned in the `length` parameter is approximately 322.543, which is the length (the circumference) of the outer contour.

For more information about the `GXGetShapeLength` function, see page 4-83.

Finding the Point at a Certain Distance Along a Contour

QuickDraw GX provides the `GXShapeLengthToPoint` function that allows you to calculate the position of the point that falls at a specified distance along a contour. This function also calculates the tangent of the contour at that point.

Geometric Operations

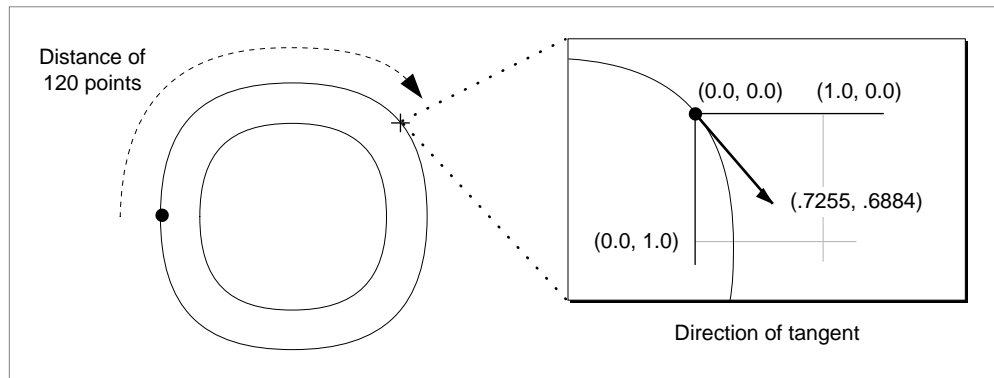
As an example, adding the function call

```
GXShapeLengthToPoint(aPathShape, 1, ff(120),
                    &thePoint, &theDirection);
```

to the sample function in Listing 4-7 determines the point that falls along the first contour at a distance of 120.0 points from the start of the contour, and stores the resulting point in the `thePoint` parameter. Also, in the `theDirection` parameter, this function stores a tangent vector indicating the direction of the contour at that point.

The result of this function is shown in Figure 4-34.

Figure 4-34 Finding a specified point on a path contour



For more information about the `GXShapeLengthToPoint` function, see page 4-85.

Finding the Bounding Rectangle and Center Point of a Shape

QuickDraw GX provides functions for finding the bounding rectangle of a shape and the center point of a shape. The bounding rectangle is the smallest rectangle that contains the shape. The center point of a shape is not the center of the shape's bounding rectangle; rather it is the "center of gravity" of a shape. QuickDraw GX guarantees that the center point of a shape remains the same even if the shape is rotated.

You can use the `GXGetShapeBounds` function to find the bounding rectangle of a shape. As an example, if you apply the function

```
GXGetShapeBounds(aPathShape, 0, &theBounds);
```

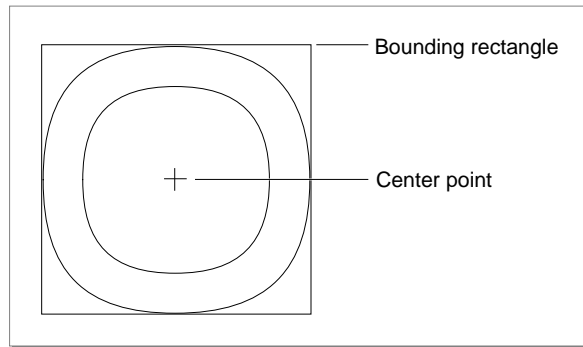
to the path shape from Listing 4-7, the result is a rectangle with the coordinates (50.0, 50.0, 150.0, 150.0). Similarly, if you apply the function

```
GXGetShapeCenter(aPathShape, 0, &thePoint);
```

to the same path shape, the result is the point: (100.0, 100.0).

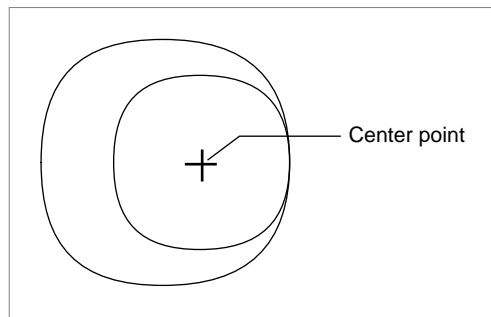
The results of these functions are depicted in Figure 4-35.

Figure 4-35 Finding the bounding rectangle and the center point of a path



If you move the inner contour of the path shape to right, the center point moves to the right as well, effectively moving with the combined “center of gravity” of the two contours, as shown in Figure 4-36.

Figure 4-36 Finding the center point of two contours



Notice that the center point lies somewhere between the center of the outer contour and the center of the inner contour.

For more information about the `GXGetShapeBounds` function, see page 4-90. For more information about the `GXGetShapeCenter` function, see page 4-87.

Finding the Area of a Shape

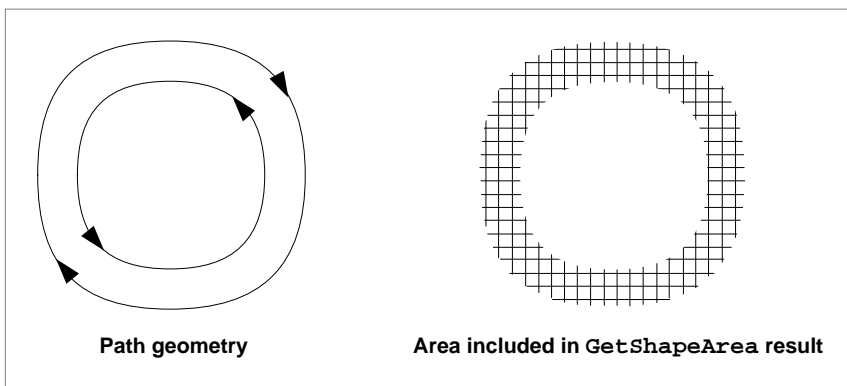
QuickDraw GX provides the `GXGetShapeArea` function so you can determine the area covered by a shape.

With the path shape from Listing 4-7 (on page 4-41), applying the function

```
GXGetShapeArea(aPathShape, 1, &theArea);
```

results in the value 4250.0. This value represents the area of the outer contour minus the area of the inner contour, as shown in Figure 4-37.

Figure 4-37 Finding the area of a path, two contours with same contour direction



In effect, the function finds the area covered by the shape as if it were filled with the winding shape fill.

Therefore, if you reverse the direction of the inner contour of this path with the function call

```
GXReverseShape(aPathShape, 2);
```

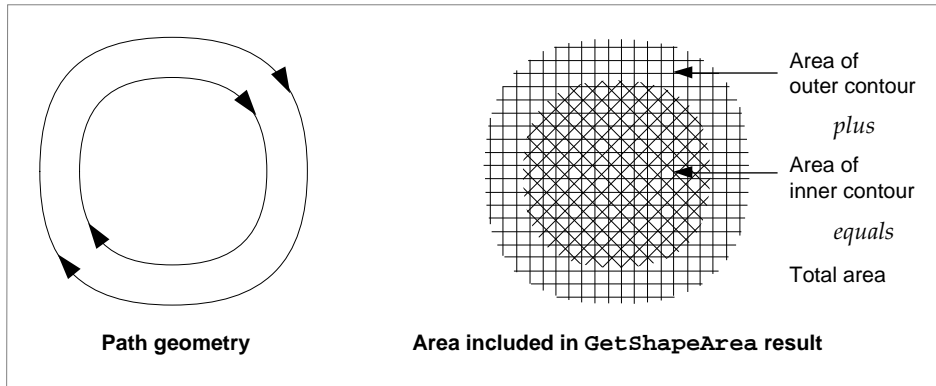
then the function call

```
GXGetShapeArea(aPathShape, 1, &theArea);
```

results in the value 12416.6666. This value represents the area of the outer contour *plus the area of the inner contour*—the area covered by the inner contour is counted twice.

The area included in this calculation is depicted in Figure 4-38.

Figure 4-38 Finding the area of a path, two contours with opposite contour direction



Note that the `GXGetShapeArea` function does not consider the shape fill when calculating area—it includes this overlapping area twice whether the shape fill is winding fill, even-odd fill, open-frame fill, or closed-frame fill.

You can correct this calculation by calling the `GXSimplifyShape` function first. For example, if you set the shape fill to winding fill with the function call

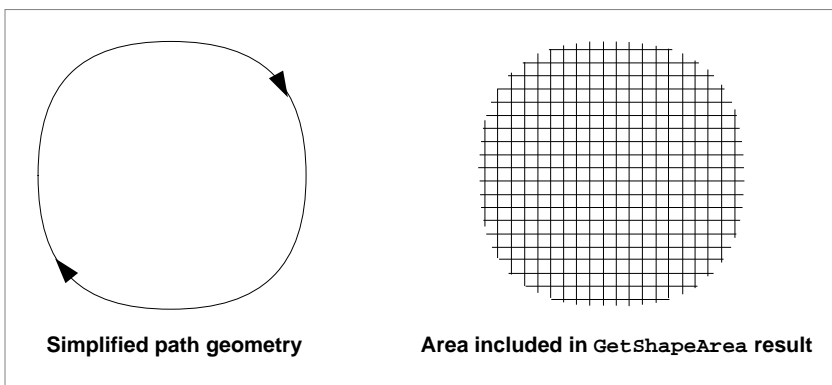
```
GXSetShapeFill(aPathShape, gxWindingFill);
```

and then call the `GXSimplifyShape` function:

```
GXSimplifyShape(aPathShape);
```

the `GXSimplifyShape` function removes the inner contour, as shown in Figure 4-39.

Figure 4-39 Finding the area of a simplified path



Geometric Operations

Once this inner contour is removed, you can call the `GXGetShapeArea` function, and the area of the original outer contour (8333.3333) is returned.

For more information about the `GXGetShapeArea` function, see page 4-88.

Setting a Shape's Bounding Rectangle

The `GXGetShapeBounds` function, illustrated on page 4-44, allows you to determine the bounding rectangle of a shape. Similarly, the `GXSetShapeBounds` function allows you to alter the bounding rectangle of a shape, thereby scaling the shape to a new size and moving it to a new location.

As an example, the sample function in Listing 4-8 creates a path with a single circular contour.

Listing 4-8 Creating a circular path

```
void CreateCircularPath(void)
{
    gxShape      aPathShape;

    static long  circularGeometry[] = {1, /* # of contours */
                                        4, /* # of points */
                                        0xF0000000, /* 1111 ... */
                                        ff(50), ff(50), /* off */
                                        ff(150), ff(50), /* off */
                                        ff(150), ff(150), /* off */
                                        ff(50), ff(150)}; /* off */

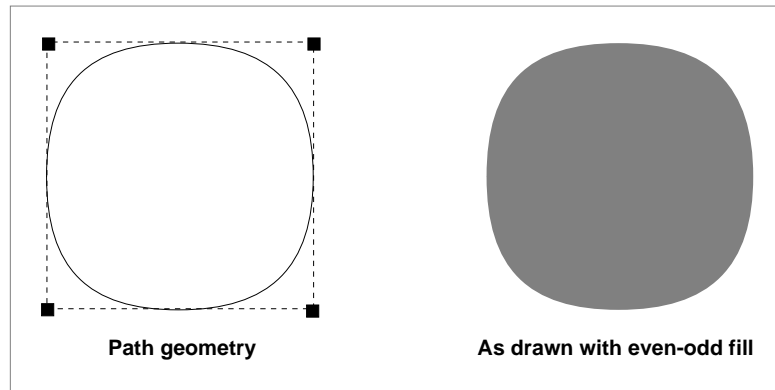
    aPathShape = GXNewPaths((gxPaths *) circularGeometry);

    GXDrawShape(aPathShape);
    GXDisposeShape(aPathShape);
}
```

Geometric Operations

The result of this function is shown in Figure 4-40.

Figure 4-40 A circular path



The bounding rectangle of this shape, which you can determine by calling

```
GXGetShapeBounds(aPathShape, 0, &theBounds);
```

is (50.0, 50.0, 150.0, 150.0). You can move and resize this shape by declaring a new bounding rectangle

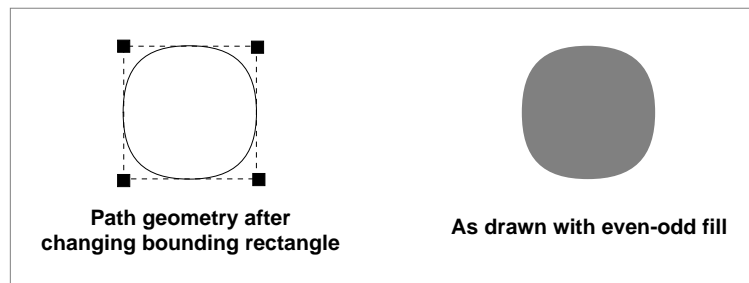
```
gxRectangle newBounds = {ff(60), ff(60), ff(110), ff(110)};
```

and then calling the function

```
GXSetShapeBounds(aPathShape, &newBounds);
```

The geometry of the altered shape is centered around the point (85.0, 85.0) and is smaller than the original shape, as shown in Figure 4-41.

Figure 4-41 A circular path after bounding rectangle changed



Geometric Operations

In this example, the `GXSetShapeBounds` function actually alters the geometry of the original shape. If you call

```
GXGetShapeArea(aPathShape, 0, &theArea);
```

the area returned in the `theArea` parameter reflects the area of the new, smaller, geometry.

However, if you set the `gxMapTransformShape` shape attribute of the path shape before setting the shape bounds, QuickDraw GX moves and resizes the shape by changing the information in the shape's transform—not by changing the geometric points of the shape's geometry. In this case, calling the `GXGetShapeArea` function, which examines only a shape's geometry and ignore its transform mapping, results in the area of the original geometry. The result of declaring a new bounding rectangle and then calling

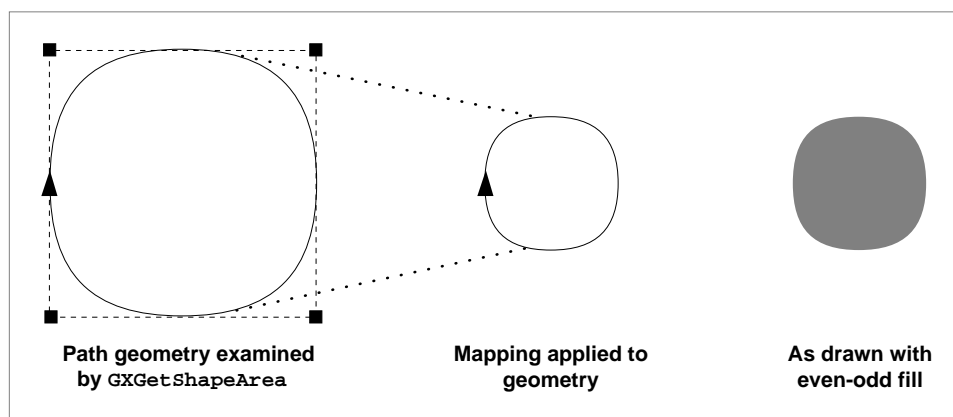
```
GXSetShapeAttributes(aPathShape,
                    GXGetShapeAttributes(aPathShape) |
                    gxMapTransformShape);
```

```
GXSetShapeBounds(aPathShape, &newBounds);
```

```
GXGetShapeArea(aPathShape, 0, &theArea);
```

is shown in Figure 4-42.

Figure 4-42 A path shape with a transform mapping



For more information about the `GXSetShapeBounds` function, see page 4-92. For more information about the `GXGetShapeBounds` function, see page 4-90.

For more information about the `gxMapTransformShape` shape attribute, see the chapter “Shape Objects” and the chapter “Transform Objects” of *Inside Macintosh: QuickDraw GX Objects*.

Insetting Shapes

Whereas the `GXSetShapeBounds` function, illustrated in the previous section, provides a way to scale a shape, the `GXInsetShape` function provides a way to resize a shape relative to its original contours.

The sample function in Listing 4-9 creates a curve shape to use as an example.

Listing 4-9 Creating a tight curve shape

```
void CreateATightCurve(void)
{
    gxShape curveShape;

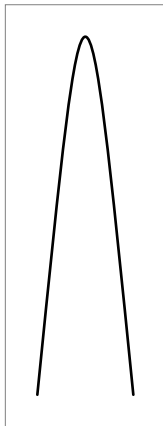
    const gxCurve tightCurveGeometry = {ff(90), ff(200),
                                         ff(110), ff(0),
                                         ff(120), ff(200)};

    curveShape = GXNewCurve(&tightCurveGeometry);
    GXSetShapeFill(curveShape, gxOpenFrameFill);

    GXDrawShape(curveShape);
    GXDisposeShape(curveShape);
}
```


The result of this function is shown in Figure 4-43.

Figure 4-43 A tight curve

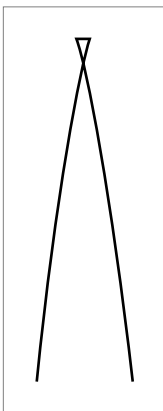


If you apply

```
GXInsetShape(curveShape, ff(10));
```

to this curve, the function insets the curve a distance of 10.0 points from the original geometry, as shown in Figure 4-44. The resulting shape is a path shape with 16 geometric points.

Figure 4-44 An inset curve shape



Geometric Operations

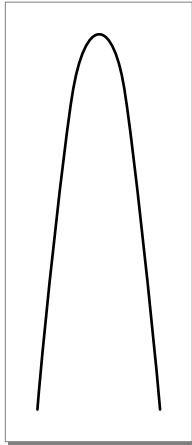
You can use a shape's curve error to control the number of geometric points in the shape resulting from the inset operation. The result of the `GXInsetShape` function has no two consecutive points closer than the distance indicated by the shape's curve error.

The `GXInsetShape` function considers the contour direction when calculating an inset contour. For example, if you use

```
GXReverseShape ( curveShape , 1 ) ;
```

to reverse the direction of the curve from Listing 4-9 before you inset the curve, the resulting path shape is actually outset from the original curve, as shown in Figure 4-45.

Figure 4-45 An outset curve



The contours created by the `GXInsetShape` function lie to the right of the original contours if you specify a positive distance and to the left of the original contours if you specify a negative distance. You can alter this behavior by setting the auto-inset style attribute, as described in Chapter 3, "Geometric Styles," in this book.

For more information about the `GXInsetShape` function, see page 4-94.

Determining Whether Two Shapes Touch

QuickDraw GX provides three functions to help you determine whether the areas of two shapes touch—that is, whether they intersect, even at a single point.

- The `GXTouchesRectanglePoint` function determines whether a point lies within the boundaries of a rectangle.
- The `GXTouchesBoundsShape` function determines whether the area covered by a shape touches the area covered a rectangle.
- The `GXTouchesShape` function determines whether one shape touches another shape

This section shows examples of using the `GXTouchesShape` function. The sample function in Listing 4-10 defines a rectangle and a circular path shape to use for these examples.

Listing 4-10 Creating a rectangle and a circular path shape

```
void CreateBoxedCircle(void)
{
    gxShape aLargeCircle;

    static gxRectangle largeBounds = {ff(50), ff(50),
                                      ff(150), ff(150)};

    static long largeCircleGeometry[] = {1, /* number of contours */
                                         4, /* number of points */
                                         0xF0000000, /* 1111 ... */
                                         ff(50), ff(50), /* off */
                                         ff(150), ff(50), /* off */
                                         ff(150), ff(150), /* off */
                                         ff(50), ff(150)}; /* off */

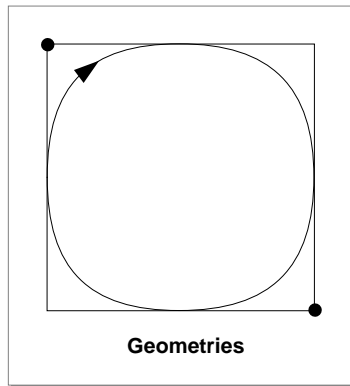
    aLargeCircle = GXNewPaths((gxPaths *) largeCircleGeometry);
    GXSetShapeFill(aLargeCircle, gxClosedFrameFill);

    GXDrawRectangle(&largeBounds, gxClosedFrameFill);
    GXDrawShape(aLargeCircle);
    GXDisposeShape(aLargeCircle);
}
```

Geometric Operations

The result of this function is shown in Figure 4-46.

Figure 4-46 A rectangle containing a circular path



You can call the `GXTouchesBoundsShape` function to test whether the rectangle and the path shape defined in Listing 4-10 touch:

```
GXTouchesBoundsShape(&largeBounds, aLargeCircle)
```

This function call returns `true`; the area of the rectangle does intersect the area of the path.

Geometric Operations

When calculating whether a rectangle and a shape intersect, the `GXTouchesBoundsShape` function assumes that the rectangle has an even-odd shape fill. The following code defines another, smaller, path shape to test for intersection with the rectangle defined in Listing 4-10:

```
gxShape aSmallCircle;

static long smallCircleGeometry[] = {1, /* number of contours */
                                     4, /* number of points */
                                     0xF0000000,
                                     ff(65), ff(65), /* off */
                                     ff(135), ff(65), /* off */
                                     ff(135), ff(135), /* off */
                                     ff(65), ff(135)}; /* off */

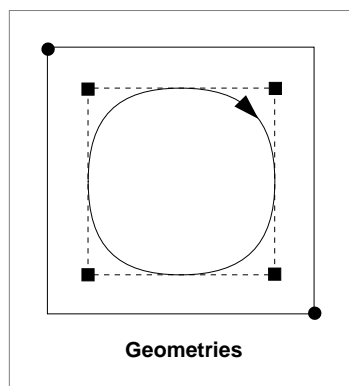
aSmallCircle = GXNewPaths((gxPaths *) smallCircleGeometry);
GXSetShapeFill(aSmallCircle, gxClosedFrameFill);
```

The call

```
GXTouchesBoundsShape(&largeBounds, aSmallCircle);
```

returns true because the smaller path shape touches the area contained by the rectangle, as shown in Figure 4-47.

Figure 4-47 A rectangle that touches a circular path shape



Geometric Operations

The `GXTouchesBoundsShape` function returns `true` even if the rectangle and the path shape share only an edge or even a single point. For example, if you move the small circle to the right by a distance of 85.0 points by calling

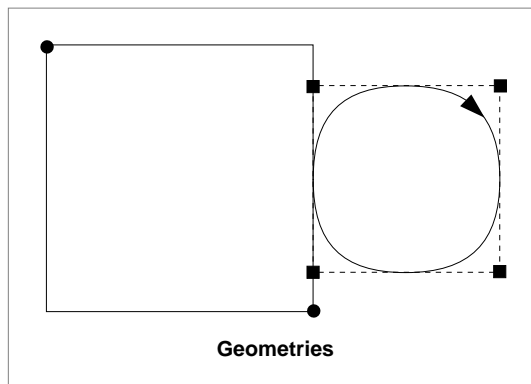
```
GXMoveShape(aSmallCircle, ff(85), 0);
```

as depicted in Figure 4-48, then the call

```
GXTouchesBoundsShape(&largeBounds, aSmallCircle)
```

still returns `true`.

Figure 4-48 A rectangle and a circular path touching at a single point



The `GXTouchesShape` function works similarly to the `GXTouchesBoundsShape` function, but it determines whether any two shapes intersect.

Geometric Operations

As an example, if you give the path shapes defined earlier in this section the even-odd shape fill by calling

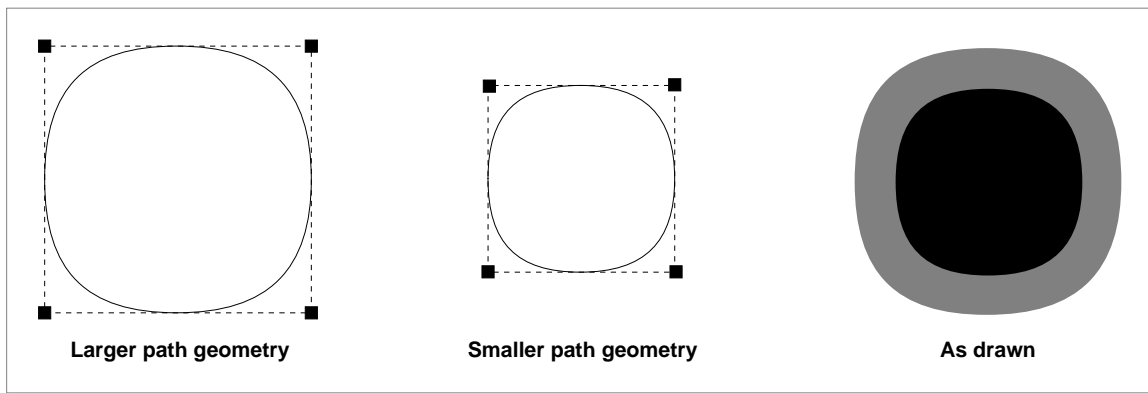
```
GXSetShapeFill(aLargeCircle, gxEvenOddFill);
GXSetShapeFill(aSmallCircle, gxEvenOddFill);
```

then the call

```
GXContainsShape(aLargeCircle, aSmallCircle)
```

returns `true`; the small path intersects the area contained in the large path, as shown in Figure 4-49.

Figure 4-49 A large circular path shape touching a smaller circular path shape



For information about the `GXTouchesRectanglePoint` function, see page 4-96. For more information about the `GXTouchesBoundsShape` function and the `GXTouchesShape` function, see page 4-97 and page 4-98, respectively.

Determining Whether One Shape Contains Another

QuickDraw GX also provides three functions to help you determine whether one area contains another:

- The `GXContainsRectangle` function determines whether one rectangle contains another.
- The `GXContainsBoundsShape` function determines whether the area covered by a rectangle contains the area covered by a shape.
- The `GXContainsShape` function determines whether the area covered by one shape contains the area covered by another shape.

The sample function in Listing 4-11 creates a small circular path and a larger, donut-shaped path to test for containment.

Listing 4-11 Creating a path shape with two contours and a smaller concentric rectangle shape

```
void CreateMultiplePaths(void)
{
    gxShape twoCircleShape, smallSquareShape;

    static rectangle smallSquareGeometry[] = {ff(90), ff(90),
                                              ff(110), ff(110)};

    static long twoCircleGeometry[] = {2, /* # of contours */
                                       4, /* # of points */
                                       0xF0000000, /* 1111 ... */
                                       ff(50), ff(50),
                                       ff(150), ff(50),
                                       ff(150), ff(150),
                                       ff(50), ff(150),
                                       4, /* # of points */
                                       0xF0000000, /* 1111 ... */
                                       ff(65), ff(65),
                                       ff(65), ff(135),
                                       ff(135), ff(135),
                                       ff(135), ff(65)};

    twoCircleShape = GXNewPaths((gxPaths *) twoCircleGeometry);
    GXSetShapeFill(twoCircleShape, gxEvenOddFill);
}
```


Geometric Operations

```

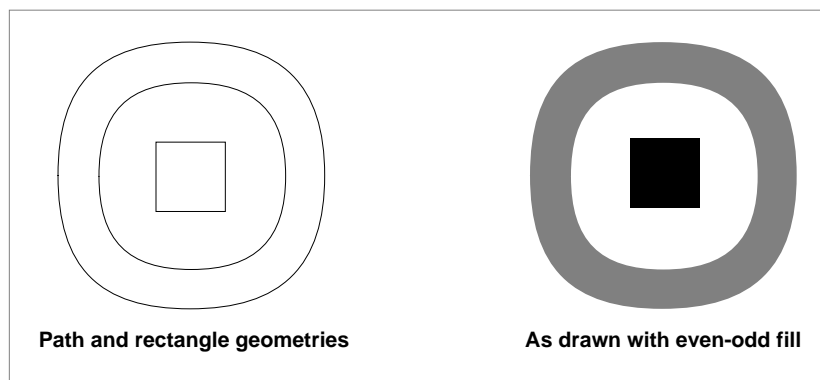
    smallSquareShape = GXNewRectangle(&smallSquareGeometry);
    GXSetShapeFill(smallSquareShape, gxEvenOddFill);

    GXDrawShape(twoCircleShape);
    GXDisposeShape(twoCircleShape);
    GXDrawShape(smallSquareShape);
    GXDisposeShape(smallSquareShape);
}

```

The results of this sample function are shown in Figure 4-50.

Figure 4-50 A path shape with two contours and a smaller concentric rectangle shape



Since the `GXContainsShape` function considers shape fill when calculating whether one shape contains another, the following function call:

```
GXContainsShape(twoCircleShape, smallerCircleShape);
```

returns `false`; the area covered by the larger path does not contain the area covered by the smaller path.

For more information about the `GXContainsShape` function, see page 4-103.

The functions `GXContainsRectangle` and `GXContainsBoundsShape` work similarly to the `GXContainsShape` function, except the input parameters to these functions are rectangle geometries, rather than shapes. The `GXContainsRectangle` function compares two rectangle geometries and the `GXContainsBoundsShape` compares a rectangle geometry to a shape.

For more information about the `GXContainsRectangle` function and the `GXContainsBoundsShape` function, see page 4-100 and page 4-101, respectively.

Performing Geometric Arithmetic With Shapes

QuickDraw GX provides six arithmetic operations you can apply to geometric shapes: union, intersection, difference, reverse difference, exclusion, and inversion.

To illustrate these operations, the sample function in Listing 4-12 creates two shapes: a diamond-shaped polygon and a circular path.

Listing 4-12 Creating a diamond-shaped polygon and a circular path that intersect

```
void CreateDiamondAndCircle(void)
{
    gxShape  diamondShape, circleShape;

    static long circleGeometry[] = {1, /* number of contours */
                                    4, /* number of points */
                                    0xF0000000, /* 1111 ... */
                                    ff(100), ff(100), /* off */
                                    ff(200), ff(100), /* off */
                                    ff(200), ff(200), /* off */
                                    ff(100), ff(200)}; /* off */

    static long diamondGeometry[] = {1, /* number of contours */
                                     4, /* number of points */
                                     ff(50), ff(150),
                                     ff(100), ff(100),
                                     ff(150), ff(150),
                                     ff(100), ff(200)};

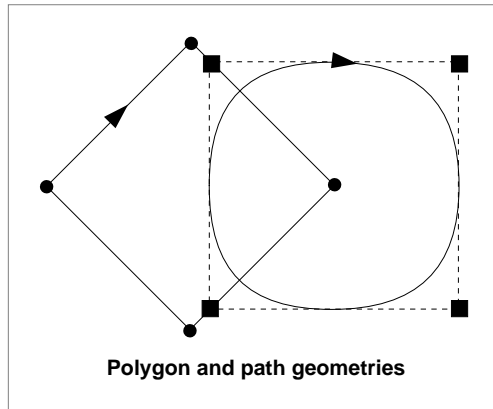
    diamondShape = GXNewPolygons((gxPolygons *) diamondGeometry);

    circleShape = GXNewPaths((gxPaths *) circleGeometry);

    GXDrawShape(diamondShape);
    GXDisposeShape(diamondShape);
    GXDrawShape(circleShape);
    GXDisposeShape(circleShape);
}
```

The resulting shapes are shown in Figure 4-51.

Figure 4-51 A diamond-shaped polygon geometry and a circular path geometry



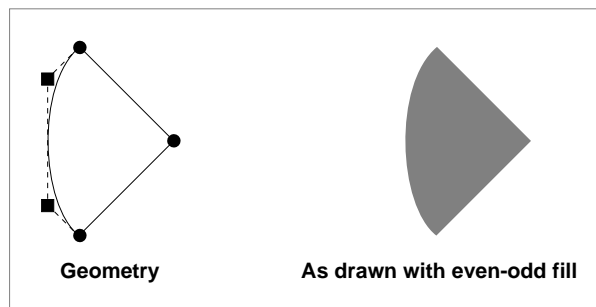
The `GXIntersectShape` function finds the area common to two shapes. This function takes two parameters—the shapes to intersect—and stores the result in the first parameter.

If you apply the `GXIntersectShape` function to the diamond-shaped polygon and the circular path from Listing 4-12 by calling

```
GXIntersectShape(diamondShape, circleShape);
GXDrawShape(diamondShape);
```

you get the resulting shape shown in Figure 4-52.

Figure 4-52 The intersection of a diamond-shaped polygon and a circular path



Geometric Operations

Implementation Note

Due to an implementation limit with QuickDraw GX version 1.0, you can find the intersection of two framed shapes only if the shapes are points, lines, or curves. You can, however, find the intersection of a framed shape and a filled shape; the intersection is the part of the framed shape contained in the filled shape. In this case, the target shape must be the framed shape and the operand shape must be the filled shape. ♦

You can find the intersection of two rectangle geometries—without having to encapsulate those geometries into shapes—using the `GXIntersectRectangle` function, which is described on page 4-105. Similarly, you can find the union of two rectangle geometries (which is considered to be the smallest rectangle that contains them both) using the `GXUnionRectangle` function, which is described on page 4-106.

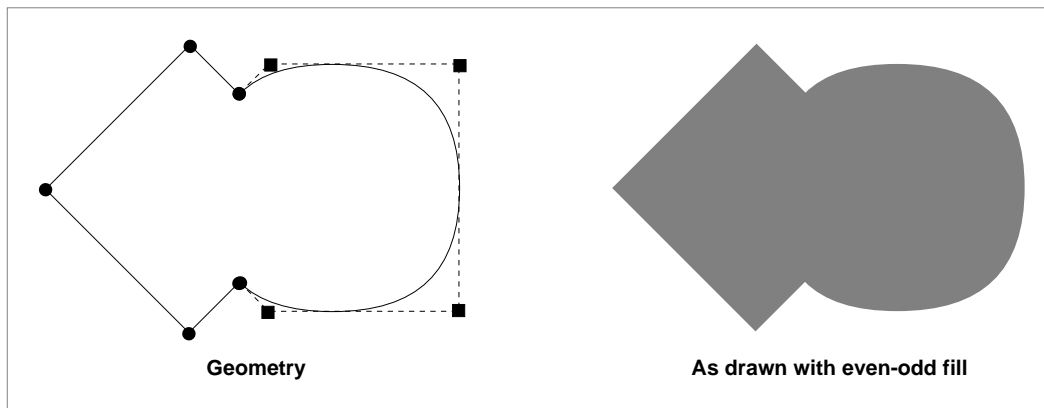
The `GXUnionShape` function combines the areas covered by two shapes. This function also takes two parameters—the shapes to combine—and stores the result in the first parameter.

If you apply the `GXUnionShape` function to the diamond-shaped polygon and the circular path from Listing 4-12 by calling

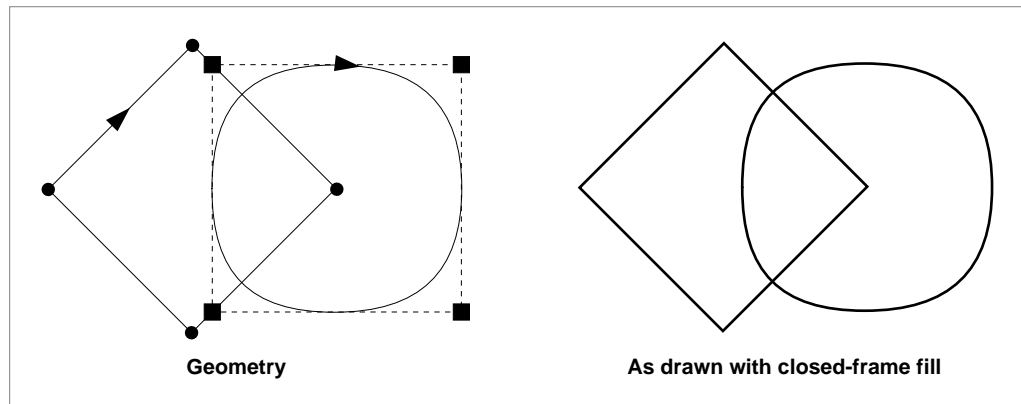
```
GXUnionShape(diamondShape, circleShape);
GXDrawShape(diamondShape);
```

you get the resulting shape shown in Figure 4-53.

Figure 4-53 The union of a diamond-shaped polygon and a circular path



Although you cannot find the union of a framed shape and a solid shape, you can find the union of two framed shapes. If the diamond-shaped polygon and the circular path from Listing 4-12 had closed-frame fills, the resulting union would appear as shown in Figure 4-54.

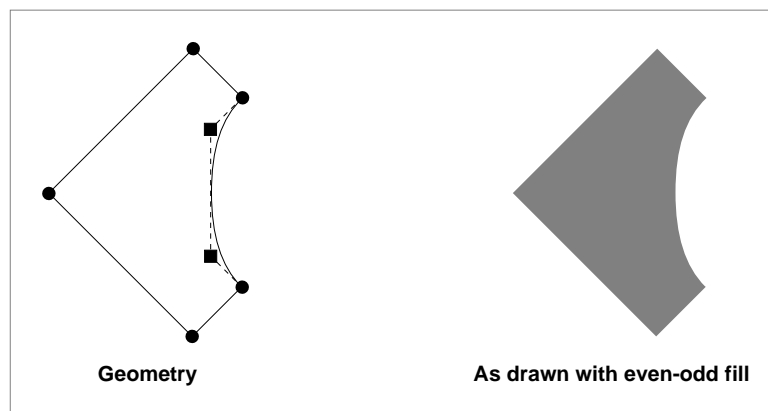
Figure 4-54 The union of a framed diamond-shaped polygon and a circular path

The `GXDifferenceShape` function subtracts the area of one shape from the area of another. This function takes two parameters—the shape to subtract from and the shape to subtract—and stores the result in the first parameter.

If you apply the `GXDifferenceShape` function to the diamond-shaped polygon and the circular path from Listing 4-12 by calling

```
GXDifferenceShape(diamondShape, circleShape);
GXDrawShape(diamondShape);
```

you get the resulting shape shown in Figure 4-55.

Figure 4-55 The result of subtracting a circular path from a diamond-shaped polygon

Geometric Operations

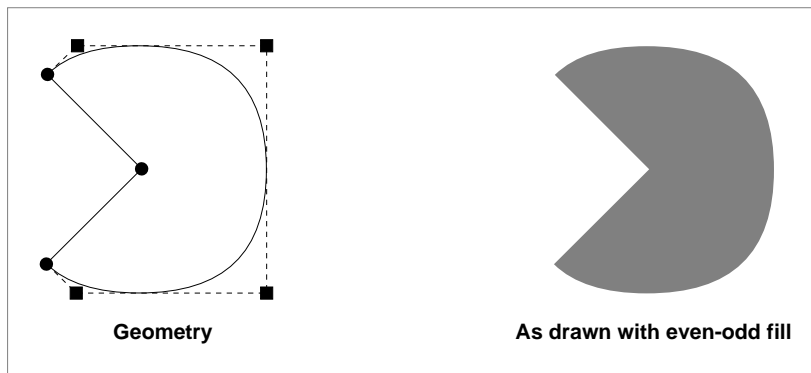
The `GXReverseDifferenceShape` function is similar to the `GXDifferenceShape` function, except the `GXReverseDifferenceShape` function subtracts the first parameter from the second parameter. Like `GXDifferenceShape`, it also stores the result in the first parameter.

If you apply the `GXReverseDifferenceShape` function to the diamond-shaped polygon and the circular path from Listing 4-12 by calling

```
GXReverseDifferenceShape(diamondShape, circleShape);
GXDrawShape(diamondShape);
```

you get the resulting shape shown in Figure 4-56.

Figure 4-56 The result of subtracting a diamond-shaped polygon from a circular path



The `GXExcludeShape` function performs the exclusive-OR operation on the areas of two shapes—that is, it finds the area that is covered by one shape or the other, but not by both shapes.

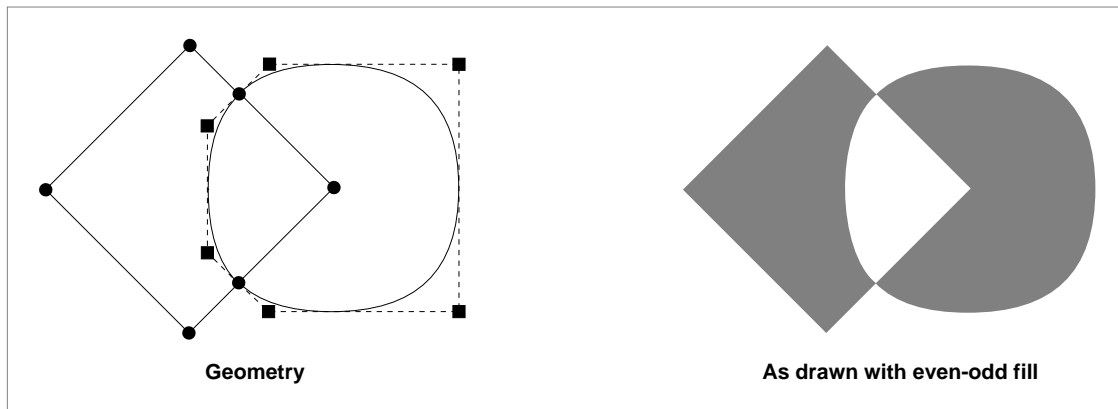
Geometric Operations

If you apply the `GXExcludeShape` function to the diamond-shaped polygon and the circular path from Listing 4-12 by calling

```
GXExcludeShape(diamondShape, circleShape);
GXDrawShape(diamondShape);
```

you get the resulting shape shown in Figure 4-57.

Figure 4-57 The result of the exclusive-OR operation on a polygon and a path



Finally, QuickDraw GX provides the `GXInvertShape` function which inverts the area covered by a shape—that is, the resulting shape covers all of the area not covered by the original shape. This function takes one parameter—the shape to invert—and stores the result in this parameter.

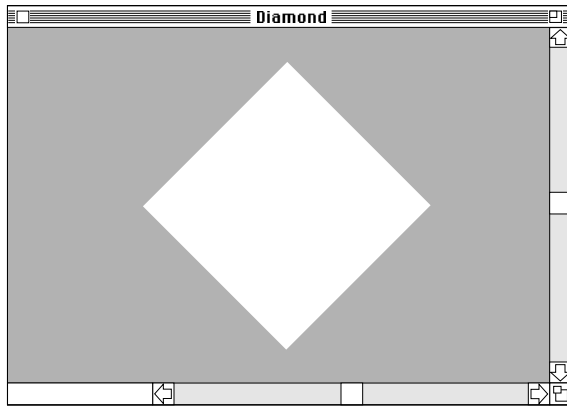
Geometric Operations

If you apply the `GXInvertShape` function to the diamond-shaped polygon from Listing 4-12 by calling

```
GXInvertShape(diamondShape);
```

you get the resulting shape shown in Figure 4-58. Notice that this shape extends to the full extent of its view port.

Figure 4-58 An inverted diamond



For more information about these arithmetic operations, see the function descriptions in “Performing Geometric Arithmetic With Shapes” beginning on page 4-104.

Geometric Operations Reference

QuickDraw GX allows you to determine geometric information about and perform geometric manipulations on shapes. This section describes the data types and functions that are related to these operations.

Constants and Data Types

This section describes the enumeration QuickDraw GX uses to specify information about contour direction.

Contour Directions

QuickDraw GX assigns a contour direction to every contour in a shape: contours are either clockwise or counterclockwise.

Contour directions are specified by the `gxContourDirections` enumeration, which is defined as follows:

```
enum gxContourDirections {
    gxCounterclockwiseDirection,
    gxClockwiseDirection
};

typedef long gxContourDirection;
```

Constant descriptions

`gxCounterclockwiseDirection`
A counterclockwise contour direction.

`gxClockwiseDirection`
A clockwise contour direction.

For more information about the contours of the various geometric shapes, see Chapter 2, “Geometric Shapes,” in this book.

For more information about contour direction, see “Contours and Contour Direction” beginning on page 4-4.

For more information about how QuickDraw GX uses contour direction, see the description of the `GXReverseShape` function on page 4-70.

Functions

QuickDraw GX provides functions you can call to perform geometric operations on geometric shapes and their geometries. This section includes descriptions of the functions that allow you to

- determine and alter the contour direction of a shape's contours
- reduce and simplify a shape's geometry
- incorporate style information into a shape's geometry
- obtain geometric information about a shape's geometry
- determine and alter the bounding rectangle of a shape
- inset a shape's geometry
- perform geometric arithmetic on shapes

Determining and Reversing Contour Direction

The contours of geometric shapes have a contour direction: clockwise or counterclockwise. The following factors determine the contour direction of a contour:

- the order and position of the geometric points that make up the contour
- the contour's relative position to other contours in the shape if the shape has multiple contours

For a discussion of geometric points, see Chapter 2, "Geometric Shapes," in this book.

The `GXGetShapeDirection` function allows you to determine the contour direction of a specified contour of a shape.

The `GXReverseShape` function allows you to reverse the order of the geometric points that define a contour and therefore reverse the contour's direction.

GXGetShapeDirection

You can use the `GXGetShapeDirection` function to determine whether a contour has a clockwise or counterclockwise direction.

```
gxContourDirection GXGetShapeDirection(gxShape source,
                                       long contour);
```

`source` A reference to the shape containing the contour.

`contour` The contour index of the contour whose direction you want to determine.

Geometric Operations

function result The direction of the contour, either `gxClockwiseDirection` or `gxCounterclockwiseDirection`.

DESCRIPTION

The `GXGetShapeDirection` function indicates whether QuickDraw GX considers the contour indicated by the `contour` parameter of the shape indicated by the `source` parameter to be clockwise or counterclockwise. You can use this information to determine how QuickDraw GX will draw a shape that has the `gxInsideFrameStyle` or `gxOutsideFrameStyle` style attribute set, or how the `GXInsetShape` function affects a shape.

For empty and full shapes, this function posts the error `graphic_type_does_not_have_multiple_contours`.

For point shapes and line shapes, this function always returns `gxClockwiseDirection`. Although the order of the geometric points in a line shape's geometry does not affect the result of this function, it may affect how QuickDraw GX draws the line. For example, if the line is dashed, the order of the geometric points determines the end of the line at which dashing begins. Also, if the line has a pen width and the `gxInsideFrameStyle` or `gxOutsideFrameStyle` style attribute is set, the order of the geometric points determines the side of the geometry on which QuickDraw GX draws the line.

For rectangle shapes, the result of this function is determined by examining which corners are specified by the geometric points. For a rectangle whose geometry includes the upper-left point and the lower-right point, this function returns `gxClockwiseDirection`, regardless of the order of the two points in the geometry. For a rectangle whose geometry includes the upper-right point and the lower-left point, this function returns `gxCounterclockwiseDirection`, again regardless of the order of the two points.

For curve shapes, polygon shapes, and path shapes, reversing the order of any contour's geometric points reverses the result of this function.

If you provide a source shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Posts the error <code>graphic_type_does_not_have_multiple_contours</code>
picture	Posts the error <code>graphic_type_does_not_have_multiple_contours</code>
text	Posts the error <code>illegal_type_for_shape</code>
glyph	Posts the error <code>illegal_type_for_shape</code>
layout	Posts the error <code>illegal_type_for_shape</code>

Geometric Operations

ERRORS, WARNINGS, AND NOTICES

Errors

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

Warnings

<code>contour_out_of_range</code>

SEE ALSO

For examples using this function, see “Determining and Reversing Contour Direction” beginning on page 4-23.

To reverse the order of geometric points in a shape’s geometry, use the `GXReverseShape` function, described in the next section.

For a description of the `GXInsetShape` function, see page 4-94. For a description of the `gxInsideFrameStyle` and `gxOutsideFrameStyle` style attributes, see Chapter 3, “Geometric Styles,” in this book.

GXReverseShape

You can use the `GXReverseShape` function to reverse the order of the geometric points in a shape’s contour.

```
void GXReverseShape(gxShape target, long contour);
```

<code>target</code>	A reference to the shape containing the contour.
<code>contour</code>	The number of the contour you want to reverse. You may specify a value of 0 for this parameter to indicate all contours.

DESCRIPTION

The `GXReverseShape` function reverses the order of the geometric points of the contour specified by the `contour` parameter in the shape specified by the `target` parameter.

If you specify a value of 0 for the `contour` parameter, this function reverses the order of the geometric points in each contour of the target shape, but does not affect the order of the contours themselves. If the target shape is a rectangle shape, this function converts it to a polygon shape before reversing the direction.

Geometric Operations

You can use this function to control how QuickDraw GX

- draws shapes with a winding shape fill
- draws shapes with the `gxInsideFrameStyle` or `gxOutsideFrameStyle` style attribute set
- places dashes on a dashed contour
- insets shape geometries

If you provide a target shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Posts the error <code>illegal_type_for_shape</code>
picture	Posts the error <code>illegal_type_for_shape</code>
text	Posts the error <code>illegal_type_for_shape</code>
glyph	Posts the error <code>illegal_type_for_shape</code>
layout	Posts the error <code>illegal_type_for_shape</code>

ERRORS, WARNINGS, AND NOTICES

Errors

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

Warnings

<code>contour_out_of_range</code>
<code>shape_access_not_allowed</code>

SEE ALSO

For examples using this function, see “Determining and Reversing Contour Direction” beginning on page 4-23.

To determine the direction of a contour, use the `GXGetShapeDirection` function, described on page 4-68.

For a discussion of geometric points, see the Chapter 2, “Geometric Shapes.”

For a discussion of contour direction, see “Contours and Contour Direction” beginning on page 4-4.

For a discussion of dashes and the `gxInsideFrameStyle` or `gxOutsideFrameStyle` style attributes, see the Chapter 3, “Geometric Styles,” in this book.

For a description of the `GXInsetShape` function, see page 4-94.

Breaking Shape Contours

Each contour of a polygon or path shape can be made up of many parts: each polygon contour can contain many lines and each path contour can contain many lines and curves.

The `GXBreakShape` function allows you to specify a geometric point at which to break a single contour into two contours.

GXBreakShape

You can use the `GXBreakShape` function to break a single contour into two contours.

```
void GXBreakShape(gxShape target, long index);
```

`target` A reference to the shape containing the contour to break.

`index` The geometry index of the point at which to break the contour.

DESCRIPTION

The `GXBreakShape` function breaks an existing contour into two contours at a specified geometric point.

This function can convert the shape type of the target shape. For example, you can break line shapes at their first point by specifying a geometry index of 1 for the `index` parameter. The result is a polygon shape with two contours: the first contour is empty and the second contour is the original line. If you specify a value of 2 for the `index` parameter, this function posts the notice `shape_already_broken`, and the original line is unaffected.

Similarly, you can break curve shapes at their first point; the result is a path shape with two contours. You can also break curve shapes at the off-curve control point by specifying a value of 2 for the `index` parameter. The resulting path shape has two contours: the first contour ends at the off-curve control point, and the second contour begins at the off-curve control point. You must add on-curve geometric points at the end of the first contour and the beginning of the second contour before drawing this path shape.

The function affects polygon and path shapes in the following ways:

- If the geometry index you specify corresponds to the first point of a contour, this function inserts an empty contour into the shape before the specified point.
- If the geometry index you specify corresponds to the last point of a contour, this function posts a `shape_already_broken` notice, and the original shape is unaffected.
- If the geometric index you specify corresponds to a point between the first and last points of a contour, this function breaks the existing contour into two contours. The specified point becomes the first point of the new second contour.

Geometric Operations

For empty, full, and point shapes, this function posts the error `graphic_type_does_not_contain_points`. For rectangle shapes, this function posts a `rectangles_cannot_be_inserted_into` notice.

If you provide a target shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Posts the error <code>graphic_type_does_not_contain_points</code>
picture	Posts the error <code>graphic_type_does_not_contain_points</code>
text	Posts the error <code>illegal_type_for_shape</code>
glyph	Posts the error <code>illegal_type_for_shape</code>
layout	Posts the error <code>illegal_type_for_shape</code>

ERRORS, WARNINGS, AND NOTICES

Errors	
<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>index_is_less_than_one</code>	(debugging version)
<code>shape_access_not_allowed</code>	(debugging version)
<code>rectangles_cannot_be_inserted_into</code>	(debugging version)
<code>graphic_type_does_not_contain_points</code>	(debugging version)
<code>illegal_type_for_shape</code>	(debugging version)
Warnings	
<code>shape_access_not_allowed</code>	
<code>index_out_of_range</code>	
<code>contour_out_of_range</code>	
Notices (debugging version)	
<code>shape_already_broken</code>	

SEE ALSO

For an example using this function, see “Breaking Shape Contours” beginning on page 4-28.

For a discussion of geometric points, see Chapter 2, “Geometric Shapes,” in this book. For other methods of breaking contours, see the shape-editing functions also described in that chapter.

To learn how this function works for the typographic shapes, see *Inside Macintosh: QuickDraw GX Typography*.

Reducing and Simplifying Shapes

The geometries of QuickDraw GX shapes can contain many unnecessary or complicating elements:

- duplicate geometric points
- unnecessary colinear geometric points
- crossed contours
- overlapping contours
- unnecessary contour breaks
- a more complex shape type than necessary

The `GXReduceShape` function eliminates unnecessary geometric points.

The `GXSimplifyShape` function eliminates crossed contours, overlapping contours, unnecessary contour breaks, and also sets a shape's shape type to the simplest type necessary to describe the shape's geometry.

GXReduceShape

You can use the `GXReduceShape` function to remove unnecessary geometric points from a polygon or path contour.

```
void GXReduceShape(gxShape target, long contour);
```

<code>target</code>	A reference to the polygon or path shape containing the contour whose unnecessary geometric points you want to eliminate.
<code>contour</code>	The index of the contour you want to reduce. You may specify a value of 0 for this parameter to indicate all contours.

DESCRIPTION

The `GXReduceShape` function removes unnecessary geometric points from the contour indicated by the `contour` parameter of the shape indicated by the `target` parameter. The geometric points removed by this function include both duplicate and colinear geometric points. Duplicate geometric points are sequential geometric points in the same contour with the same (x, y) coordinate pair. Colinear geometric points are sequential geometric points that fall on the same line as the preceding and the subsequent geometric point. Although this function may affect the geometry of a shape, the resulting shape appears the same as the original shape when drawn, unless the curve error of the target shape is nonzero.

Note

Under certain circumstances, the `GXReduceShape` function actually increases the number of geometric points used to define a shape. For path shapes, the number of geometric points in the resulting shape can be up to one third more than the number of points in the original shape. Even in this case, the resulting shape appears the same as the original shape when drawn. ♦

The `GXReduceShape` function does consider the curve error of the target shape when selecting which geometric points to remove. If the distance between a point and a neighboring point is less than that indicated by the curve error, the `GXReduceShape` function considers them to be duplicate points. If you specify a target shape with a nonzero curve error, the resulting shape may draw differently than the original shape—the greater the curve error, the more drastic the difference may be. For shapes with many points within a distance of less than that indicated by the curve error, the resulting shape can sometimes degenerate to a surprising result.

The shape fill of the target shape can also affect the results of this function. For example, if the first point and the last point of a contour are the same geometric point, this function removes the last point if the target shape has a closed-frame fill or any of the solid fills. However, if the target shape has an open-frame fill, this function does not remove the last point.

Similarly, if one or more of the points at the end of a contour is colinear with one or more points at the beginning of that contour, this function considers them all to lie on the same line if the target shape has a closed-frame fill or any of the solid fills and the unnecessary points are removed—even the first and last point of the original contour can be removed. However, if the target shape has an open-frame fill, the first and the last points of a contour are never removed.

This function operates only within individual contours; it never combines contours or compares points from different contours. Also, this function does not convert between shape types. The resulting shape always has the same shape type as the original shape.

If you specify a source shape that is not a polygon or a path shape, this function posts the error `graphic_type_cannot_be_reduced`.

If you provide a target shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Posts the error <code>graphic_type_cannot_be_reduced</code>
picture	Posts the error <code>graphic_type_cannot_be_reduced</code>
text	Posts the error <code>graphic_type_cannot_be_reduced</code>
glyph	Posts the error <code>graphic_type_cannot_be_reduced</code>
layout	Posts the error <code>graphic_type_cannot_be_reduced</code>

Geometric Operations

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`shape_is_nil`
`size_of_path_exceeds_implementation_limit`
`number_of_points_exceeds_implementation_limit`
`graphic_type_cannot_be_reduced` (debugging version)

Warnings

`unable_to_traverse_open_contour_that_starts_or_ends_off_the_curve` (debugging version)
`contour_out_of_range`
`shape_access_not_allowed`

SEE ALSO

For examples using this function, see “Eliminating Unnecessary Geometric Points” beginning on page 4-30.

For more information about reduced and simplified shapes, see “Reducing and Simplifying Shape Geometries” beginning on page 4-9.

For a discussion of geometric points and contours, see Chapter 2, “Geometric Shapes,” in this book.

GXSimplifyShape

You can use the `GXSimplifyShape` function to eliminate from a shape any unnecessary contour breaks, contour crossings, and internal contour loops.

```
void GXSimplifyShape(gxShape target);
```

`target` A reference to the shape you want to simplify.

DESCRIPTION

The `GXSimplifyShape` function performs operations on the geometry of the shape specified by the `target` parameter and simplifies the description of the shape, sometimes changing the shape type, without affecting how the shape is drawn.

Most importantly, the resulting shape has no crossed contours. The `GXSimplifyShape` function adds geometric points and changes contour directions to redefine the shape's geometry so that no contour crosses over itself or any other contour.

This function also removes unnecessary contour breaks. If the last point of one contour is identical to the first point of the next contour, this function combines the two contours into a single contour.

Note

Under certain circumstances, the `GXSimplifyShape` function actually increases the number of geometric points and the number of contours used to define a shape. However, the simplified shape still appears the same as the original shape when drawn. ♦

If the geometry of the original shape can be expressed as a geometry of a simpler shape type, this function converts the shape to the simpler type. For example, if the shape referenced by the `target` parameter is a polygon, but the geometry of that polygon defines a simple square, the `GXSimplifyShape` function converts the shape to a rectangle type and redefines the geometry as appropriate. As another example, a path shape with no curved contours is converted to a polygon shape type.

The shape fill of the target shape also affects the simplifications. For example, if the target shape has two circular, concentric contours (an inner contour and an outer contour) and both contours have the same contour direction, the following occurs:

- If the shape has a winding shape fill, the inner contour does not affect how the shape is drawn. In this case, the `GXSimplifyShape` function removes the inner contour.
- If the shape has an even-odd shape fill, the inner contour does affect how the shape is drawn. In this case, the `GXSimplifyShape` function maintains the inner contour, but it reverses the direction of that contour.

As a result of these simplifications, changing the shape fill of a simplified shape from winding fill to even-odd fill or from even-odd fill to winding fill does not affect the appearance of the shape when drawn.

Geometric Operations

If you provide a target shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Chooses smaller color set if possible, if the pixel size is 1, 2, 4, or 8; posts the notice <code>shape_already_in_simple_form</code> if the pixel size is 16 or 32; converts to a rectangle shape if every pixel in the bitmap has the same color
picture	Posts the notice <code>shape_already_in_simple_form</code>
text	Simplifies to the empty shape if appropriate; posts the notice <code>shape_already_in_simple_form</code> otherwise
glyph	Simplifies to the empty shape, a text shape, or a simpler glyph shape as appropriate; posts the notice <code>shape_already_in_simple_form</code> if no simplification possible
layout	Simplifies to the empty shape if appropriate; posts the notice <code>shape_already_in_simple_form</code> otherwise

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`shape_is_nil`
`number_of_contours_exceeds_implementation_limit`
`number_of_points_exceeds_implementation_limit`
`size_of_path_exceeds_implementation_limit`
`size_of_polygon_exceeds_implementation_limit`
`shape_access_not_allowed`
`functionality_unimplemented` (debugging version)

Warnings

`unable_to_traverse_open_contour_that_starts_or_ends_off_the_curve`
 (debugging version)

Notices (debugging version)

`shape_already_in_simple_form`

SEE ALSO

For examples using this function, see “Simplifying Shapes” beginning on page 4-33.

For more information about simplified shapes, see “Reducing and Simplifying Shape Geometries” beginning on page 4-9.

For a discussion of geometric points and contours, see Chapter 2, “Geometric Shapes,” in this book.

To remove unnecessary geometric points but not perform other simplifications, use the `GXReduceShape` function, described on page 4-74.

Incorporating Style Information Into Shape Geometries

QuickDraw GX requires that shapes used for certain purposes (caps, joins, dashes, patterns, and clips) be in primitive form—that is, their style modifications must be incorporated into their geometries. For example, the `GXSetShapeDash` function requires that the shape used for dashing be in primitive form; the `GXSetShapeCap`, `GXSetShapeJoin`, and `GXSetShapePattern` functions are similar.

For more information about the primitive form of shapes and for examples of functions that use shapes in their primitive form, see Chapter 3, “Geometric Styles,” in this book.

The `GXPrimitiveShape` function converts a shape to its primitive form, incorporating the modifications made by the shape’s style into the shape’s geometry.

GXPrimitiveShape

You can use the `GXPrimitiveShape` function to convert a shape to its primitive form.

```
void GXPrimitiveShape(gxShape target);
```

`target` A reference to the shape to convert to primitive form.

DESCRIPTION

The `GXPrimitiveShape` function converts the shape referenced by the `target` parameter to its primitive form—that is, it changes the geometry, shape fill, and shape type of the target shape to incorporate the information from the original shape’s style (including pen width, dashes, joins, and so on).

For example, a horizontal line shape with a greater-than-zero pen width becomes a filled rectangle shape. A diagonal line shape with a greater-than-zero pen width becomes a filled polygon shape. A curve shape with a greater-than-zero pen width becomes a filled path shape. A framed shape dashed with rectangles becomes a polygon shape with multiple contours—each contour representing one of the original dashes.

For the geometric shapes, the shape resulting from this function can be a hairline shape or a solid-filled shape. In either case, the information from the style object is no longer necessary because it has been incorporated into the shape object itself.

Implementation Note

In version 1.0 of QuickDraw GX, this function posts an error of `functionality_unimplemented` for picture shapes. ♦

The result of the `GXPrimitiveShape` function is not simplified, nor are its unnecessary geometric points removed. You may want to simplify or reduce the resulting shape by calling the `GXSimplifyShape` function or the `GXReduceShape` function.

Geometric Operations

The following table gives information about this function for each type of geometric shape:

Shape type	Action taken
empty	If the shape fill is the <code>noFill</code> shape fill, this function posts the notice <code>shape_already_in_primitive_form</code> . If the shape fill is either of the inverse shape fills, this function returns a full shape (unless the shape has a pattern, in which case the function returns the shape described by the pattern).
full	If the shape fill is the <code>noFill</code> shape fill or any of the inverse fills, this function returns an empty shape. Otherwise, the function posts the notice <code>shape_already_in_primitive_form</code> (unless the shape has a pattern, in which case the function returns the shape described by the pattern).
point	<p>If the shape fill is the <code>noFill</code> shape fill, this function returns an empty shape. If the pen width is greater than zero and the shape has a start cap, the function returns the start cap. If the pen width is zero or the shape has no start cap, the function returns an empty shape.</p> <p>If the shape has a pen width of zero, no start cap, and a solid pattern, the function returns a point as indicated by the pattern. If the shape has a framed pattern, the function posts the <code>clip_to_frame_shape_unimplemented</code> error. Bitmap patterns are ignored.</p> <p>If one of the grid-constraining attributes is set, this function constrains the point geometry to the grid.</p>
line	<p>If the shape fill is the <code>noFill</code> shape fill, this function returns an empty shape. If the pen width is greater than zero, the function returns a polygon shape (or a path shape depending on the start and end caps).</p> <p>If the pen width is zero and the shape has a solid pattern, the function returns a point or a line as indicated by the pattern. If the shape has a framed pattern, the function posts the <code>clip_to_frame_shape_unimplemented</code> error. Bitmap patterns are ignored.</p> <p>If one of the grid-constraining attributes is set, this function constrains the geometry to the grid.</p>

Geometric Operations

Shape type	Action taken
curve	<p>If the shape fill is the <code>noFill</code> shape fill, this function returns an empty shape. If the pen width is greater than zero, the function returns a path shape.</p> <p>If the pen width is zero and the shape has a solid pattern, the function returns a point, line, curve, or path line as indicated by the pattern. If the shape has a framed pattern, the function posts the <code>clip_to_frame_shape_unimplemented</code> error. Bitmap patterns are ignored.</p> <p>If one of the grid-constraining attributes is set, this function constrains the geometry to the grid.</p>
rectangle	<p>If the shape fill is the <code>noFill</code> shape fill, this function returns an empty shape. If the shape fill is one of the framed fills and the pen width is greater than zero, the function returns a polygon shape (or a path shape depending on the join shape).</p> <p>If the rectangle has a solid pattern, the function returns a point, line, or polygon as indicated by the pattern. If the rectangle is framed, has a pen width of zero, and has a framed pattern, the function posts the <code>clip_to_frame_shape_unimplemented</code> error. Bitmap patterns are ignored.</p> <p>If one of the grid-constraining attributes is set, this function constrains the geometry to the grid.</p>
polygon	<p>If the shape fill is the <code>noFill</code> shape fill or the shape has no contours, this function returns an empty shape. If the shape fill is one of the framed fills and the pen width is greater than zero, the function returns a polygon shape (or a path shape depending on the caps and join).</p> <p>If the shape has a solid pattern, the function returns a point, line, or polygon as indicated by the pattern. If the polygon is framed, has a pen width of zero, and has a framed pattern, the function posts the <code>clip_to_frame_shape_unimplemented</code> error. Bitmap patterns are ignored.</p> <p>If one of the grid-constraining attributes is set, this function constrains the geometry to the grid.</p>
path	<p>If the shape fill is the <code>noFill</code> shape fill or the shape has no contours, this function returns an empty shape. If the shape fill is one of the framed fills and the pen width is greater than zero, the function returns a path shape.</p> <p>If the shape has a solid pattern, the function returns a point, line, or polygon as indicated by the pattern. If the path is framed, has a pen width of zero, and shape has a framed pattern, the function posts the <code>clip_to_frame_shape_unimplemented</code> error. Bitmap patterns are ignored.</p> <p>If one of the grid-constraining attributes is set, this function constrains the geometry to the grid.</p>

Geometric Operations

If you provide a target shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Constrains the bitmap position to an integer grid position if one of the grid-constraining attributes is set
picture	Posts the notice <code>shape_already_in_primitive_form</code>
text	Converts to glyph shape; constrains the text position to an integer grid position if one of the grid-constraining attributes is set; converts to an empty shape if appropriate
glyph	Constrains the glyph positions to integer grid positions if one of the grid-constraining attributes is set; eliminates any nil styles and complex styles; converts to an empty shape if appropriate
layout	Converts to a glyph shape; converts to an empty shape if appropriate

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>shape_access_not_allowed</code>	
<code>functionality_unimplemented</code>	(debugging version)
<code>clip_to_frame_shape_unimplemented</code>	(debugging version)

Warnings

<code>unable_to_traverse_open_contour_that_starts_or_ends_off_the_curve</code>	(debugging version)
<code>face_override_style_font_must_match_style</code>	

Notices (debugging version)

<code>shape_already_in_primitive_form</code>
--

SEE ALSO

For examples using this function, see “Converting a Shape to Primitive Form” beginning on page 4-38.

For more information about the primitive form of shapes and for examples of functions that use shapes in their primitive form, see Chapter 3, “Geometric Styles,” in this book.

To eliminate unnecessary geometric points, use the `GXReduceShape` function, described on page 4-74. To simplify a shape’s contours, use the `GXSimplifyShape` function, described on page 4-76.

Finding Geometric Information About Shapes

The functions described in this section calculate geometric information about a shape.

The `GXGetShapeLength` function calculates the length of a particular contour or of all contours in a shape.

The `GXShapeLengthToPoint` function determines the point that falls at a specified distance along a particular contour or along all combined contours of a shape.

The `GXGetShapeCenter` function determines the center point of a particular contour or of all combined contours of a shape's geometry.

The `GXGetShapeArea` function calculates the area covered by a particular contour or by all combined contours of a shape's geometry.

You can also use the `GXGetShapeBounds` function, described on page 4-90, to find geometric information about a shape—in this case, the shape's bounding rectangle.

GXGetShapeLength

You can use the `GXGetShapeLength` function to determine the length of a particular contour or of all contours of a shape.

```
gxWide *GXGetShapeLength(gxShape source, long index,
                          gxWide *length);
```

source A reference to the shape containing the contour.

index The index of the contour you want to measure. You may specify a value of 0 for this parameter to measure all contours.

length A pointer to a `gxWide` value. On return, this value indicates the length of the indicated contour.

function result The length of the indicated contour.

DESCRIPTION

The `GXGetShapeLength` function returns as the function result the length of the perimeter of a particular contour of a shape. This function calculates the length of the contour as defined in the shape's geometry; it does not consider transformations to the shape made by the shape's transform.

Geometric Operations

For empty and full shapes, this function posts the warning `shape_does_not_have_length`. For point shapes, it returns zero. For solid polygon and path shapes, this function calculates the length as if the shape had the closed-frame shape fill.

If you provide a target shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
<code>bitmap</code>	Posts the warning <code>shape_does_not_have_length</code>
<code>picture</code>	Returns the length of the specified picture item or the sum of the length of all picture items
<code>text</code>	Posts the warning <code>shape_does_not_have_length</code>
<code>glyph</code>	Posts the warning <code>shape_does_not_have_length</code>
<code>layout</code>	Posts the warning <code>shape_does_not_have_length</code>

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>parameter_is_nil</code>	(debugging version)
<code>contour_is_less_than_zero</code>	(debugging version)

Warnings

<code>contour_out_of_range</code>	
<code>shape_does_not_have_length</code>	(debugging version)
<code>unable_to_traverse_open_contour_that_starts_or_ends_off_the_curve</code>	(debugging version)

SEE ALSO

For an example using this function, see “Finding the Length of a Contour” beginning on page 4-42.

For information about the contours of the various geometric shapes, see Chapter 2, “Geometric Shapes,” in this book.

GXShapeLengthToPoint

You can use the `GXShapeLengthToPoint` function to determine the point that falls at a certain distance along a contour of a shape.

```
gxPoint *GXShapeLengthToPoint(gxShape target, long index,
                               Fixed length, gxPoint *location,
                               gxPoint *tangent);
```

<code>target</code>	A reference to the shape containing the contour you want to examine.
<code>index</code>	The number of the contour within the shape. You may specify a value of 0 for this parameter to indicate that the function should start measuring at the beginning of the first contour and continue through all contours.
<code>length</code>	The distance along the specified contour.
<code>location</code>	A pointer to a <code>gxPoint</code> structure. On return, this structure contains the point that lies along the contour at the specified distance.
<code>tangent</code>	A pointer to a <code>gxPoint</code> structure. On return, this structure contains a point that specifies a tangent vector representing the slope of the contour at the specified distance.

function result The point that lies along the contour at the specified distance. (This value is the same as the value returned in the `location` parameter.)

DESCRIPTION

The `GXShapeLengthToPoint` function returns the location of the point that lies at the distance specified by the `length` parameter along the contour specified by the `index` parameter of the target shape.

If you provide a pointer for the `tangent` parameter that is not `nil`, this function returns the slope of the specified contour at that point, in the form of a tangent vector. (The tangent vector implicitly starts at point (0.0, 0.0) and ends at the point indicated by the returned `tangent` parameter.)

This function measures the contour length as defined in the shape's geometry; it does not consider transformations to the shape made by the shape's transform.

Geometric Operations

If the target shape has the `noFill` shape fill, this function posts the error `shape_fill_not_allowed`.

If the target shape is an empty shape or a full shape, this function posts the warning `shape_does_not_have_length`.

If you provide a target shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
<code>bitmap</code>	Posts the warning <code>shape_does_not_have_length</code>
<code>picture</code>	Posts the warning <code>shape_does_not_have_length</code>
<code>text</code>	Posts the warning <code>shape_does_not_have_length</code>
<code>glyph</code>	Posts the warning <code>shape_does_not_have_length</code>
<code>layout</code>	Posts the warning <code>shape_does_not_have_length</code>

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>length_is_less_than_zero</code>	(debugging version)
<code>parameter_is_nil</code>	(debugging version)
<code>shape_fill_not_allowed</code>	(debugging version)

Warnings

<code>contour_out_of_range</code>	
<code>length_out_of_range</code>	
<code>shape_does_not_have_length</code>	(debugging version)
<code>unable_to_traverse_open_contour_that_starts_or_ends_off_the_curve</code>	(debugging version)

SEE ALSO

For an example using this function, see “Finding the Point at a Certain Distance Along a Contour” beginning on page 4-42.

For information about the contours of the various geometric shapes, see Chapter 2, “Geometric Shapes,” in this book.

To measure the length of a contour, use the `GXGetShapeLength` function, described on page 4-83.

GXGetShapeCenter

You can use the `GXGetShapeCenter` function to determine the center of a specified contour of a shape.

```
gxPoint *GXGetShapeCenter(gxShape source, long index,
                          gxPoint *center);
```

<code>source</code>	A reference to the shape containing the contour whose center you want to find.
<code>index</code>	The number of the contour whose center you want to find. You may specify a value of 0 to indicate you want to find the center of the entire shape.
<code>center</code>	A pointer to a <code>gxPoint</code> structure. On return, this structure contains the point that falls at the center of the specified contour.

function result The point that falls at the center of the specified contour. (This value is the same as the value returned in the `center` parameter.)

DESCRIPTION

The `GXGetShapeCenter` function determines the point that falls at the center of the contour specified by the `index` parameter of the shape specified by the `source` parameter. If you specify a value of 0 for the `index` parameter, this function finds the center point of the entire source shape.

The center point of a shape is not merely the center of the shape's bounding rectangle; rather it is the "center of gravity" of a shape. QuickDraw GX guarantees the center point of a shape does not change even if the shape is rotated.

This function finds the center of a shape (or of a particular contour) as defined by the source shape's geometry; it does not consider shape fill or transformations to the shape made by the shape's transform. For point shapes, this function returns a copy of the point's geometry. For line and rectangle shapes, this function returns the midpoint of the geometry. For empty and full shapes, this function posts the error `shape_does_not_have_length`.

Geometric Operations

If you provide a target shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Returns midpoint of bounding rectangle
picture	Posts the error <code>illegal_type_for_shape</code>
text	Returns midpoint of the bounding rectangle of the specified glyph
glyph	Returns midpoint of the bounding rectangle of the specified glyph
layout	Converts to glyph shape and returns midpoint of the bounding rectangle of the specified glyph

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>parameter_is_nil</code>	(debugging version)
<code>illegal_type_for_shape</code>	(debugging version)
<code>contour_less_than_zero</code>	(debugging version)
<code>graphic_type_does_not_contain_points</code>	(debugging version)

Warnings

<code>contour_out_of_range</code>	
<code>unable_to_traverse_open_contour_that_starts_or_ends_off_the_curve</code>	(debugging version)

SEE ALSO

For examples using this function, see “Finding the Bounding Rectangle and Center Point of a Shape” beginning on page 4-43.

To find the bounding rectangle of a shape or a contour of a shape, use the `GXGetShapeBounds` function described on page 4-90.

GXGetShapeArea

You can use the `GXGetShapeArea` function to determine the area covered by a specific contour of a shape’s geometry.

```
gxWide *GXGetShapeArea(gxShape source, long index, gxWide *area);
```

source A reference to the shape containing the contour whose area you want to determine.

Geometric Operations

index	The number of the contour whose area you want to determine. You may specify a value of 0 for this parameter to indicate you want to determine the area of the entire shape.
area	A pointer to a <code>gxWide</code> value. On return, this value indicates the area covered by the contour.
<i>function result</i>	The area covered by the contour. (This value is the same as the value returned in the <code>area</code> parameter.)

DESCRIPTION

The `GXGetShapeArea` function returns the area covered by the contour specified by the `index` parameter of the shape indicated by the `source` parameter. This function considers only the geometry of the source shape—it does not consider the shape fill of the shape. The same geometry returns the same area whether the shape has one of the framed fills, an even-odd fill, a winding fill, or one of the inverse fills.

Some shapes have overlapping contours with the same contour direction. (When drawing these shapes, QuickDraw GX fills these overlapping areas if the shape has a winding fill and does not fill these areas if the shape has an even-odd fill.) The `GXGetShapeArea` function counts these overlapping areas twice. To correct this calculation, call the `GXSimplifyShape` function before calling the `GXGetShapeArea` function:

- For shapes with a winding shape fill, the `GXSimplifyShape` function eliminates the inner contour and, therefore, the `GXGetShapeArea` function counts the overlapping area only once.
- For shapes with an even-odd shape fill, the `GXSimplifyShape` function reverses the contour direction of the internal contour, and therefore the `GXGetShapeArea` function does not count the overlapping area at all.

This function measures the shape area as defined in the shape's geometry; it does not consider transformations to the shape made by the shape's transform.

For empty shapes, point shapes, and line shapes, this function posts the error `shape_does_not_have_area`. For full shapes, it posts the error `illegal_type_for_shape`.

If you provide a target shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Returns the bitmap height multiplied by the bitmap width
picture	Returns the sum of the areas of the picture items
text	Converts to path shape and finds area
glyph	Converts to path shape and finds area
layout	Converts to path shape and finds area

Geometric Operations

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`shape_is_nil`
`parameter_is_nil`
`shape_does_not_have_area` (debugging version)

Warnings

`index_out_of_range`
`contour_out_of_range`

SEE ALSO

For examples using this function, see “Finding the Area of a Shape” beginning on page 4-45.

For a discussion of shape fills, see Chapter 2, “Geometric Shapes,” in this book

To simplify a shape before measuring its area, use the `GXSimplifyShape` function, described on page 4-76.

Getting and Setting Shape Bounds

Every shape has a bounding rectangle—the smallest rectangle that contains the shape. The functions in this section allow you to determine and alter a shape’s bounding rectangle.

The `GXGetShapeBounds` function finds the bounding rectangle of a shape, or of a specified contour of a shape.

The `GXSetShapeBounds` function allows you to alter a shape’s bounding rectangle (and thereby move and resize the shape).

GXGetShapeBounds

You can use the `GXGetShapeBounds` function to determine the bounding rectangle of a shape or of a specified contour of a shape.

```
gxRectangle *GXGetShapeBounds(gxShape source, long index,
                              gxRectangle *bounds);
```

<code>source</code>	A reference to the shape containing the contour whose bounding rectangle you want to find.
<code>index</code>	The number of the contour whose bounding rectangle you want to find. You may specify a value of 0 to indicate you want to find the bounding rectangle of the entire shape.

Geometric Operations

bounds A pointer to a `gxRectangle` structure. On return, this structure contains the bounding rectangle of the specified contour.

function result The bounding rectangle of the specified contour. (This value is the same as the value returned in the `bounds` parameter.)

DESCRIPTION

The `GXGetShapeBounds` function determines the bounding rectangle of the contour specified by the `index` parameter of the shape specified by the `source` parameter. If you specify a value of 0 for the `index` parameter, this function finds the bounding rectangle of the entire source shape.

The bounding rectangle of a shape (or of a contour of a shape) is the smallest rectangle that contains the geometry of the shape (or of the contour).

This function finds the bounding rectangle of the source shape (or a contour of the source shape) as defined by the source shape's geometry; it does not consider shape fill or transformations to the shape made by the shape's transform.

For empty shapes and full shapes, this function posts the warning `shape_passed_has_no_bounds`. For full shapes, it returns an infinitely large rectangle; for empty shapes, it returns the inverse rectangle.

If you provide a target shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Returns the bounding rectangle
picture	Returns the bounding rectangle for the entire picture
text	Returns bounding rectangle of specified glyphs
glyph	Returns bounding rectangle of specified glyphs
layout	Posts the error <code>functionality_unimplemented</code> if the <code>index</code> parameter is not zero; returns bounding rectangle of glyphs otherwise

ERRORS, WARNINGS, AND NOTICES

Errors	
<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>contour_is_less_than_zero</code>	
<code>parameter_is_nil</code>	(debugging version)
<code>functionality_unimplemented</code>	(debugging version)
Warnings	
<code>contour_out_of_range</code>	
<code>shape_passed_has_no_bounds</code>	(debugging version)
<code>unable_to_traverse_open_contour_that_starts_or_ends_off_the_curve</code>	(debugging version)

Geometric Operations

SEE ALSO

For a discussion of rectangles and bounding rectangles, see Chapter 2, “Geometric Shapes,” in this book.

To find the center of a shape or a contour of a shape, use the `GXGetShapeCenter` function, which is described on page 4-87.

To change the bounding rectangle of a shape, use the `GXSetShapeBounds` function, described in the next section.

GXSetShapeBounds

You can use the `GXSetShapeBounds` function to change a shape’s bounding rectangle, thereby moving and resizing the shape.

```
void GXSetShapeBounds(gxShape target,
                     const gxRectangle *newBounds);
```

source A reference to the shape whose bounding rectangle you want to change.

newBounds The new bounding rectangle.

DESCRIPTION

The `GXSetShapeBounds` function changes the bounding rectangle of the shape specified by the `source` parameter to be the rectangle specified by the `newBounds` parameter.

How this function changes the bounding rectangle is determined by the source shape’s `gxMapTransformShape` shape attribute:

- If the `gxMapTransformShape` shape attribute is not set, the function changes the geometry of the source shape to fit the new bounding rectangle.
- If the `gxMapTransformShape` shape attribute is set, the function does not alter the shape’s geometry directly; instead, it changes the mapping of the shape’s transform object to scale the shape to fit in the new bounding rectangle.

By changing a shape’s bounding rectangle, you can move the shape as well as scale it in the horizontal and vertical dimensions.

For empty and full shapes, this function does nothing.

If you provide a point shape as the target shape and a new bounding rectangle that has height or width, this function posts the warning `scale_shape_out_of_range`.

Geometric Operations

If you provide a target shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Calls the GXMapShape function
picture	Calls the GXMapShape function
text	Converts to a path shape if necessary (when the ratio between the height of the new bounding rectangle and the height of the original bounding rectangle is not the same as the ratio between the width of the new bounding rectangle and the width of the original bounding rectangle)
glyph	Converts to a path shape if necessary
layout	Converts to a path shape if necessary

ERRORS, WARNINGS, AND NOTICES

Errors	
out_of_memory	
shape_is_nil	
shape_access_not_allowed	(debugging version)
functionality_unimplemented	(debugging version)
Warnings	
scale_shape_out_of_range	
character_substitution_took_place	
font_substitution_took_place	
unable_to_traverse_open_contour_that_starts_or_ends_off_the_curve	(debugging version)
shape_passed_has_no_bounds	(debugging version)

SEE ALSO

For examples of this function, see “Setting a Shape’s Bounding Rectangle” beginning on page 4-47.

For a discussion of rectangles and bounding rectangles, see Chapter 2, “Geometric Shapes,” in this book.

For a discussion of the gxMapTransformShape shape attribute, see the chapters “Shape Objects” and “Transform Objects” in *Inside Macintosh: QuickDraw GX Objects*.

To determine the bounding rectangle of a shape, use the GXGetShapeBounds function, described on page 4-90.

Insetting Shapes

The `GXInsetShape` function, described in this section, provides a way to inset or outset the contours of a shape a specified distance from the original contours.

`GXInsetShape`

You can use the `GXInsetShape` procedure to inset a shape's geometry.

```
void GXInsetShape(gxShape target, Fixed inset);
```

`source` A reference to the shape whose geometry you want to inset.

`inset` The distance to inset the geometry of the shape.

DESCRIPTION

The `GXInsetShape` function insets the geometry of the shape specified by the `target` parameter by the distance specified in the `inset` parameter. The on-curve geometric points of the resulting geometry are the specified distance inside the contour of the original geometry.

You can specify a positive or negative value for the `inset` parameter: positive values move the geometry to the inside of the original geometry; negative values move it outside the original geometry.

QuickDraw GX uses the direction of a contour to define which side is the inside of a contour: the inside is the side to the right of the contour. As a result, insetting clockwise contours by a positive amount makes them smaller while insetting counterclockwise contours by a positive amount makes them larger.

You can override this behavior by setting the `gxAutoInsetStyle` style attribute. If you set this style attribute for a shape, QuickDraw GX finds the true inside of the contour, regardless of its contour direction. With this attribute set, insetting a contour by a positive amount makes it smaller, whether it has a clockwise direction or a counterclockwise direction.

If the target shape has the `noFill` shape fill, this function posts the error `shape_fill_not_allowed`.

For empty, full, and point shapes, this function posts the error `graphic_type_cannot_be_inset`. Line shapes and rectangle shapes are converted to polygon shapes; curve shapes are converted to path shapes.

If you provide a target shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Posts the error <code>graphic_type_cannot_be_inset</code>
picture	Posts the error <code>graphic_type_cannot_be_inset</code>
text	Posts the error <code>graphic_type_cannot_be_inset</code>
glyph	Posts the error <code>graphic_type_cannot_be_inset</code>
layout	Posts the error <code>graphic_type_cannot_be_inset</code>

ERRORS, WARNINGS, AND NOTICES

Errors	
<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>graphic_type_cannot_be_inset</code>	(debugging version)
<code>shape_fill_not_allowed</code>	(debugging version)
<code>shape_access_not_allowed</code>	(debugging version)
Warnings	
<code>unable_to_traverse_open_contour_that_starts_or_ends_off_the_curve</code>	(debugging version)
Notices (debugging version)	
<code>geometry_unaffected</code>	(debugging version)

SEE ALSO

For examples using this function, see “Insetting Shapes” beginning on page 4-50.

For a discussion of contours and contour direction, see Chapter 2, “Geometric Shapes,” in this book.

For a discussion of the `gxAutoInsetStyle` style attribute, see Chapter 3, “Geometric Styles,” in this book.

To change the bounding rectangle of a shape, use the `GXSetShapeBounds` function, described on page 4-92.

Determining Whether Two Areas Touch

The functions described in this section determine if two areas touch.

The `GXTouchesRectanglePoint` function determines whether the area covered by a rectangle touches a point.

The `GXTouchesBoundsShape` function determines whether the area covered by a rectangle touches a shape.

Geometric Operations

The `GXTouchesShape` function determines whether the area covered by one shape touches the area covered by another.

The `GXIntersectShape` function, which is described on page 4-107 in the section “Performing Geometric Arithmetic With Shapes,” determines not only if two shapes intersect but also what their intersection is.

GXTouchesRectanglePoint

You can use the `GXTouchesRectanglePoint` function to determine if a point lies within or on the edge of a rectangle.

```
gxBoolean GXTouchesRectanglePoint(const gxRectangle *target,
                                   const gxPoint *test);
```

`target` A pointer to the rectangle to test as the container.

`test` A pointer to the point to test for inclusion.

function result A Boolean value indicating whether the point touches the rectangle.

DESCRIPTION

The `GXTouchesRectanglePoint` function returns `true` as its function result if the point specified by the `test` parameter lies within or on the edge of the rectangle specified by the `target` parameter, and returns `false` otherwise.

Notice that the parameters to this function are not shapes; they are pointers to a `gxPoint` or to a `gxRectangle` structure.

ERRORS, WARNINGS, AND NOTICES

Errors

`parameter_is_nil`

SEE ALSO

For a discussion of the `gxPoint` and `gxRectangle` data structures, see Chapter 2, “Geometric Shapes,” in this book.

To determine if a rectangle touches a shape, use the `GXTouchesBoundsShape` function, described in the next section.

To determine if a rectangle contains a shape, use the `GXContainsBoundsShape` function, described on page 4-101.

GXTouchesBoundsShape

You can use the `GXTouchesBoundsShape` function to determine if a rectangle and a shape touch.

```
gxBoolean GXTouchesBoundsShape(const gxRectangle *target,
                                gxShape test);
```

`target` A pointer to the rectangle to test to determine if it touches a shape.

`test` A reference to the shape to test to determine if it touches the rectangle.

function result A Boolean value indicating whether the shape touches the rectangle.

DESCRIPTION

The `GXTouchesBoundsShape` function returns `true` as its function result if the rectangle specified by the `target` parameter touches the shape specified by the `test` parameter—even if they share only an edge or a point—and returns `false` otherwise.

This function considers the shape fill, the style modifications, and the transform mapping of the test shape. Only areas that are drawn are considered when determining touching.

If you provide a test shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Compares bounding rectangle of bitmap
picture	Posts the error <code>illegal_type_for_shape</code>
text	Converts to path shape
glyph	Converts to path shape
layout	Converts to path shape

Geometric Operations

ERRORS, WARNINGS, AND NOTICES

Errors

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

SEE ALSO

For an example using this function, see “Determining Whether Two Shapes Touch” beginning on page 4-53.

For a discussion of shape fills, see Chapter 2, “Geometric Shapes,” in this book.

For a discussion of style modifications, see Chapter 3, “Geometric Styles,” in this book.

For a discussion of transform mappings, see the chapter “Transform Objects” in *Inside Macintosh: QuickDraw GX Objects*.

To determine if a rectangle touches a point, use the `GXTouchesRectanglePoint` function, described on page 4-96.

To determine if a rectangle contains a shape, use the `GXContainsBoundsShape` function, described on page 4-101.

To determine if two shapes touch, use the `GXTouchesShape` function, described in the next section.

GXTouchesShape

You can use the `GXTouchesShape` function to determine if two shapes touch.

```
gxBoolean GXTouchesShape(gxShape target, gxShape test);
```

`target` A reference to one shape to test to determine if it touches another.

`test` A reference to the other shape to test.

function result A Boolean value indicating whether the shapes intersect.

DESCRIPTION

The `GXTouchesShape` function returns `true` as its function result if the shape specified by the `target` parameter touches the shape specified by the `test` parameter—even if they share only an edge or a point—and returns `false` otherwise.

This function considers the shape fill, the style modifications, and the transform mapping of the target and test shapes. Only areas that are drawn are considered when determining touching.

For example, if the target shape has an even-odd fill and contains an overlapping contour, then the shape has an internal area that is not drawn. If the test shape lies entirely within this area, the `GXTouchesShape` function returns `false`.

As another example, if the test shape lies entirely within the target shape, but the target shape has an inverse shape fill, the `GXTouchesShape` function returns `false`.

If you provide a target or test shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Compares bounding rectangle of bitmap
picture	Posts the error <code>illegal_type_for_shape</code>
text	Converts to path shape
glyph	Converts to path shape
layout	Converts to path shape

ERRORS, WARNINGS, AND NOTICES**Errors**

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

SEE ALSO

For examples using this function, see “Determining Whether Two Shapes Touch” beginning on page 4-53.

For a discussion of shape fills, see Chapter 2, “Geometric Shapes,” in this book.

For a discussion of style modifications, see Chapter 3, “Geometric Styles,” in this book.

For a discussion of transform mappings, see the chapter “Transform Objects” in *Inside Macintosh: QuickDraw GX Objects*.

To determine if a rectangle touches a shape, use the `GXTouchesBoundsShape` function, described on page 4-97.

To determine if a shape contains another shape, use the `GXContainsShape` function, described on page 4-103.

Determining Whether One Shape Contains Another

The functions described in this section determine if one area contains another.

The `GXContainsRectangle` function determines whether the area covered by one rectangle contains the area covered by another.

The `GXContainsBoundsShape` function determines whether the area covered by a rectangle contains the area covered by a shape.

The `GXContainsShape` function determines whether the area covered by one shape contains the area covered by another.

GXContainsRectangle

You can use the `GXContainsRectangle` function to determine if one rectangle contains another.

```
gxBoolean GXContainsRectangle(const gxRectangle *container,
                             const gxRectangle *test);
```

`container` A pointer to the rectangle to test as the container.

`test` A pointer to the rectangle to test for inclusion.

function result A Boolean value indicating whether the container rectangle contains the test rectangle.

DESCRIPTION

The `GXContainsRectangle` function returns `true` as its function result if the rectangle specified by the `test` parameter lies within the rectangle specified by the `target` parameter, and `false` otherwise.

This function may return `true` even if the container and test rectangles share one or more edges. This function returns `true` when the container and test rectangles are defined by the same coordinates.

Notice that the parameters to this function are not shapes; they are pointers to `gxRectangle` structures.

ERRORS, WARNINGS, AND NOTICES

Errors`parameter_is_nil`

SEE ALSO

For a discussion of the `gxRectangle` data structure, see Chapter 2, “Geometric Shapes,” in this book.

To determine if a rectangle touches a point, use the `GXTouchesRectanglePoint` function, described on page 4-96.

To determine if a rectangle contains a shape, use the `GXContainsBoundsShape` function, described in the next section.

GXContainsBoundsShape

You can use the `GXContainsBoundsShape` function to determine if a rectangle contains a shape or a particular contour of a shape.

```
gxBoolean GXContainsBoundsShape(const gxRectangle *container,
                                gxShape test, long index);
```

`container` A pointer to the rectangle to test as the container.

`test` A reference to the shape containing the contour to test for inclusion.

`index` The number of the contour to test for inclusion. You may specify a value of 0 to indicate you want to test the entire shape for inclusion.

function result A Boolean value indicating whether the container rectangle contains the specified contour of the test shape.

DESCRIPTION

The `GXContainsBoundsShape` function returns `true` as its function result if the rectangle specified by the `container` parameter contains the contour indicated by the `index` parameter of the shape specified by the `test` parameter and returns `false` otherwise.

This function may return `true` even if the container rectangle and the indicated contour of the test shape share one or more edges.

Geometric Operations

This function considers the shape fill, the style modifications, and the transform mapping of the test shape. Only areas that are drawn are considered when determining whether the container rectangle contains the specified contour.

If you provide a test shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Compares bounding rectangle of bitmap
picture	Compares bounding rectangle of entire picture
text	Converts to path shape
glyph	Converts to path shape
layout	Converts to path shape

ERRORS, WARNINGS, AND NOTICES

Errors

out_of_memory	
shape_is_nil	
parameter_is_nil	(debugging version)
illegal_type_for_shape	(debugging version)
shape_operator_may_not_be_a_picture	(debugging version)

SEE ALSO

For a discussion of shape fills, see Chapter 2, “Geometric Shapes,” in this book.

For a discussion of style modifications, see Chapter 3, “Geometric Styles,” in this book.

For a discussion of transform mappings, see the chapter “Transform Objects” of *Inside Macintosh: QuickDraw GX Objects*.

To determine if a rectangle touches a shape, use the `GXTouchesBoundsShape` function, described on page 4-97.

To determine if a shape contains another shape, use the `GXContainsShape` function, described in the next section.

GXContainsShape

You can use the `GXContainsShape` function to determine if the area covered by a shape contains the area covered by another shape.

```
gxBoolean GXContainsShape(gxShape container, gxShape test);
```

`container` A reference to the shape to test as the container.

`test` A reference to the shape to test for inclusion.

function result A Boolean value indicating whether the container shape contains the test shape.

DESCRIPTION

The `GXContainsShape` function returns `true` as its function result if the shape specified by the `container` parameter contains the shape specified by the `test` parameter, and returns `false` otherwise.

This function may return `true` even if the container shape and the test shape share one or more edges; it returns `true` if they are the same shape.

This function considers the shape fill, the style modifications, and the transform mapping of the container and test shapes. Only areas that are drawn are considered when determining whether the container shape contains the test shape.

The container shape must have one of the solid shape fills (even-odd, winding, inverse even-odd, or inverse winding). The test shape may have any shape fill.

If the test shape has a framed shape fill, this function returns `true` if the frame lies entirely within the area of the container shape, or along the edges of the container shape. As a result, a solid shape contains its own frame.

If you provide a test shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Compares bounding rectangle of bitmap
picture	Posts the error <code>illegal_type_for_shape</code>
text	Converts to path shape
glyph	Converts to path shape
layout	Converts to path shape

Geometric Operations

ERRORS, WARNINGS, AND NOTICES

Errors

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

SEE ALSO

For examples using this function, see “Determining Whether One Shape Contains Another” beginning on page 4-58.

For a discussion of shape fills, see Chapter 2, “Geometric Shapes,” in this book.

For a discussion of style modifications, see Chapter 3, “Geometric Styles,” in this book.

For a discussion of transform mappings, see the chapter “Transform Objects” in *Inside Macintosh: QuickDraw GX Objects*.

To determine if a rectangle contains a shape, use the `GXContainsBoundsShape` function, described on page 4-101.

To determine if one shape touches another, use the `GXTouchesShape` function, described on page 4-98.

Performing Geometric Arithmetic With Shapes

QuickDraw GX provides six arithmetic operations you can apply to geometric shapes: intersection, union, difference, reverse difference, exclusion, and inversion.

The `GXIntersectRectangle` and `GXUnionRectangle` perform the intersection and union operations on rectangle structures.

The other functions described in this section perform the arithmetic operations on shapes:

- The `GXIntersectShape` function finds the area common to the shapes.
- The `GXUnionShape` function finds the smallest area that contains both the shapes.
- The `GXDifferenceShape` function finds the area covered by the first shape that is not covered by the second shape.
- The `GXReverseDifferenceShape` function finds the area covered by the second shape that is not covered by the first shape.
- The `GXExcludeShape` function finds the area covered by one shape or the other, but not by both.
- The `GXInvertShape` function finds the area not covered by a shape.

GXIntersectRectangle

You can use the `GXIntersectRectangle` function to find the intersection of two rectangles.

```
gxBoolean GXIntersectRectangle(gxRectangle *target,
                               const gxRectangle *source,
                               const gxRectangle *operand);
```

<code>target</code>	A pointer to a <code>gxRectangle</code> structure. On return, the intersection of the source and operand rectangles. You may specify the value <code>nil</code> for this parameter if you do not want the intersection to be calculated. Depending on the result of the intersection operation, this pointer may point to the source or operand rectangle.
<code>source</code>	A pointer to one of the rectangles to intersect.
<code>operand</code>	A pointer to the other rectangle to intersect.

function result A Boolean value indicating whether the rectangles intersect.

DESCRIPTION

The `GXIntersectRectangle` function returns `true` as its function result if the source rectangle and the operand rectangle intersect, and returns `false` otherwise.

If you provide a pointer for the `target` parameter that is not `nil`, this function returns the intersection of the source and operand rectangles in the `gxRectangle` structure pointed to by the `target` parameter.

If the source rectangle and the operand rectangle do not intersect or share only one edge, this function returns `false` and does not affect the `target` rectangle.

You may specify the source rectangle or the operand rectangle as the `target` rectangle. In this case, the function calculates the intersection of the original rectangles and then places the calculated intersection into the source or operand rectangle, as specified.

Notice that the parameters to this function are not shapes; they are pointers to `gxRectangle` data structures.

ERRORS, WARNINGS, AND NOTICES

Errors

`parameter_is_nil`

Geometric Operations

SEE ALSO

For a discussion of the `gxRectangle` data structure, see Chapter 2, “Geometric Shapes.”

For a discussion of geometric arithmetic, see “Geometric Arithmetic” beginning on page 4-21.

To find the intersection of two shapes, use the `GXIntersectShape` function, described on page 4-107.

To find the union of two rectangles, use the `GXUnionRectangle` function, described in the next section.

GXUnionRectangle

You can use the `GXUnionRectangle` function to find the smallest rectangle that contains two other rectangles.

```
gxRectangle *GXUnionRectangle(gxRectangle *target,
                              const gxRectangle *source,
                              const gxRectangle *operand);
```

target A pointer to a `gxRectangle` structure. On return, the smallest rectangle containing both the source and operand rectangles.

source A pointer to one of the rectangles to combine.

operand A pointer to the other rectangle to combine.

function result The smallest rectangle containing both the source and operand rectangles. (This rectangle is the same as the rectangle returned in the `target` parameter.)

DESCRIPTION

The `GXUnionRectangle` function calculates the smallest rectangle containing both the source rectangle and the operand rectangle and stores the results in `target` parameter. This function also returns the calculated rectangle as its function result.

You may specify the source rectangle or the operand rectangle as the target rectangle. In this case, the function calculates the smallest rectangle containing both of the original rectangles and then places the calculated rectangle into the source or operand rectangle, as specified.

Notice that the parameters to this function are not shapes, but pointers to the `gxRectangle` data structures.

ERRORS, WARNINGS, AND NOTICES

Errors`parameter_is_nil`

SEE ALSO

For a discussion of the `gxRectangle` data structure, see Chapter 2, “Geometric Shapes,” in this book.

For a discussion of geometric arithmetic, see “Geometric Arithmetic” beginning on page 4-21.

To find the intersection of two rectangles, use the `GXIntersectRectangle` function, described in this previous section.

To find the union of two shapes, use the `GXUnionShape` function, described on page 4-109.

GXIntersectShape

You can use the `GXIntersectShape` function to find the intersection of two shapes.

```
void GXIntersectShape(gxShape target, gxShape operand);
```

target On input, a reference to one of the shapes to intersect. On output, a reference to the intersection of the input target shape and the operand shape.

operand A reference to the other shape to intersect.

DESCRIPTION

The `GXIntersectShape` function finds the intersection of the target shape and the operand shape, reduces and simplifies the result, and stores it in the target shape. If the original target shape and the operand shape do not intersect, the resulting target shape is an empty shape.

If the target shape and the operand shape share only an edge, and if both have a solid fill, the resulting target shape is an empty shape. However, you can provide a framed target shape and a solid operand shape—the result being a framed shape.

This function considers the shape fill, the style modifications, and the transform mapping of the target and operand shapes. Only areas that are drawn are considered when determining intersection.

Implementation Note

Due to an implementation limit with QuickDraw GX version 1.0, you can find the intersection of two framed shapes only if the shapes are points, lines, or curves. ♦

Geometric Operations

If you provide a target or operand shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Posts the error <code>shape_operator_may_not_be_a_bitmap</code>
picture	Posts the error <code>shape_operator_may_not_be_a_picture</code>
text	Converts to path shape if other parameter is not an empty shape or a full shape
glyph	Converts to path shape if other parameter is not an empty shape or a full shape
layout	Converts to path shape if other parameter is not an empty shape or a full shape

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>number_of_contours_exceeds_implementation_limit</code>	
<code>number_of_points_exceeds_implementation_limit</code>	
<code>size_of_path_exceeds_implementation_limit</code>	
<code>size_of_polygon_exceeds_implementation_limit</code>	
<code>fill_type_not_allowed</code>	(debugging version)
<code>shape_access_not_allowed</code>	(debugging version)
<code>clip_to_frame_shape_unimplemented</code>	(debugging version)
<code>shape_operator_may_not_be_a_bitmap</code>	(debugging version)
<code>shape_operator_may_not_be_a_picture</code>	(debugging version)

Warnings

<code>character_substitution_took_place</code>	
<code>font_substitution_took_place</code>	
<code>unable_to_traverse_open_contour_that_starts_or_ends_off_the_curve</code>	(debugging version)

SEE ALSO

For an example using this function, see “Performing Geometric Arithmetic With Shapes” beginning on page 4-60.

For a discussion of geometric arithmetic, see “Geometric Arithmetic” beginning on page 4-21.

For a discussion of shape fills, see Chapter 2, “Geometric Shapes,” in this book.

For a discussion of style modifications, see Chapter 3, “Geometric Styles,” in this book.

Geometric Operations

For a discussion of transform mappings, see the chapter “Transform Objects” in *Inside Macintosh: QuickDraw GX Objects*.

To determine if two shapes touch, use the `GXTouchesShape` function, described on page 4-98.

To find the union of two shapes, use the `GXUnionShape` function, described in the next section.

GXUnionShape

You can use the `GXUnionShape` function to find the union of two shapes.

```
void GXUnionShape(gxShape target, gxShape operand);
```

<code>target</code>	On input, a reference to one of the shapes to combine. On output, a reference to the union of the input target shape and the operand shape.
<code>operand</code>	A reference to the other shape to combine.

DESCRIPTION

The `GXUnionShape` function finds the union of the target shape and the operand shape, reduces and simplifies the result, and stores it in the target shape.

This function considers the shape fill, the style modifications, and the transform mapping of the target and operand shape. Only areas that are drawn are considered when calculating the union.

The target shape and the operand shape must both have solid fills (even-odd, winding, inverse even-odd, or inverse winding) or both have framed fills (open-frame or closed-frame); one of each type of fill is not allowed.

If you provide a target or operand shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Posts the error <code>shape_operator_may_not_be_a_bitmap</code>
picture	Posts the error <code>shape_operator_may_not_be_a_picture</code>
text	Converts to path shape if other parameter is not an empty shape or a full shape
glyph	Converts to path shape if other parameter is not an empty shape or a full shape
layout	Converts to path shape if other parameter is not an empty shape or a full shape

Geometric Operations

ERRORS, WARNINGS, AND NOTICES

Errors

out_of_memory	
shape_is_nil	
number_of_contours_exceeds_implementation_limit	
number_of_points_exceeds_implementation_limit	
size_of_path_exceeds_implementation_limit	
size_of_polygon_exceeds_implementation_limit	
fill_type_not_allowed	(debugging version)
shape_access_not_allowed	(debugging version)
clip_to_frame_shape_unimplemented	(debugging version)
shape_operator_may_not_be_a_bitmap	(debugging version)
shape_operator_may_not_be_a_picture	(debugging version)

Warnings

character_substitution_took_place	
font_substitution_took_place	
unable_to_traverse_open_contour_that_starts_or_ends_off_the_curve	(debugging version)

SEE ALSO

For examples using this function, see “Performing Geometric Arithmetic With Shapes” beginning on page 4-60.

For a discussion of geometric arithmetic, see “Geometric Arithmetic” beginning on page 4-21.

For a discussion of shape fills, see Chapter 2, “Geometric Shapes,” in this book.

For a discussion of style modifications, see Chapter 3, “Geometric Styles,” in this book.

For a discussion of transform mappings, see *Inside Macintosh: QuickDraw GX Objects*.

To find the intersection of two shapes, use the `GXIntersectShape` function, described on page 4-107.

GXDifferenceShape

You can use the `GXDifferenceShape` function to find the geometric difference between two shapes.

```
void GXDifferenceShape(gxShape target, gxShape operand);
```

target	On input, a reference to the shape to subtract from. On output, a reference to a shape describing the difference between the two shapes.
--------	--

operand	A reference to the shape to subtract.
---------	---------------------------------------

DESCRIPTION

The `GXDifferenceShape` function subtracts the operand shape from the target shape, reduces and simplifies the result, and stores it in the target shape.

The initial target shape does not have to contain the operand shape; the result of this function is the intersection of the target and operand shapes subtracted from the target shape.

This function considers the shape fill, the style modifications, and the transform mapping of the target and operand shapes: only areas that are drawn are considered when calculating the difference.

The operand shape cannot have one of the framed shape fills (open-frame or closed-frame). The target shape can have one of the framed fills; in this case, the resulting shape is the part of the frame that does not lie within the operand shape.

If you provide a target or operand shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Posts the error <code>shape_operator_may_not_be_a_bitmap</code>
picture	Posts the error <code>shape_operator_may_not_be_a_picture</code>
text	Converts to path shape if other parameter is not an empty shape or a full shape
glyph	Converts to path shape if other parameter is not an empty shape or a full shape
layout	Converts to path shape if other parameter is not an empty shape or a full shape

ERRORS, WARNINGS, AND NOTICES

Errors	
<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>number_of_contours_exceeds_implementation_limit</code>	
<code>number_of_points_exceeds_implementation_limit</code>	
<code>size_of_path_exceeds_implementation_limit</code>	
<code>size_of_polygon_exceeds_implementation_limit</code>	
<code>fill_type_not_allowed</code>	(debugging version)
<code>shape_access_not_allowed</code>	(debugging version)
<code>clip_to_frame_shape_unimplemented</code>	(debugging version)
<code>shape_operator_may_not_be_a_bitmap</code>	(debugging version)
<code>shape_operator_may_not_be_a_picture</code>	(debugging version)
Warnings	
<code>character_substitution_took_place</code>	
<code>font_substitution_took_place</code>	
<code>unable_to_traverse_open_contour_that_starts_or_ends_off_the_curve</code>	(debugging version)

Geometric Operations

SEE ALSO

For examples using this function, see “Performing Geometric Arithmetic With Shapes” beginning on page 4-60.

For a discussion of geometric arithmetic, see “Geometric Arithmetic” beginning on page 4-21.

For a discussion of shape fills, see Chapter 2, “Geometric Shapes,” in this book.

For a discussion of style modifications, see Chapter 3, “Geometric Styles,” in this book.

For a discussion of transform mappings, see *Inside Macintosh: QuickDraw GX Objects*.

For information about related routines, see the description of the `GXIntersectShape` function on page 4-107, the `GXUnionShape` function on page 4-109, and the `GXReverseDifferenceShape` function in the next section.

GXReverseDifferenceShape

You can use the `GXReverseDifferenceShape` function to find the geometric difference between two shapes.

```
void GXReverseDifferenceShape(gxShape target, gxShape operand);
```

target On input, a reference to the shape to subtract. On output, a reference to a shape describing the difference between the two shapes.

operand A reference to the shape to subtract from.

DESCRIPTION

The `GXReverseDifferenceShape` function subtracts the target shape from the operand shape and stores the result in the target shape.

The initial operand shape does not have to contain the target shape; the result of this function is the intersection of the target and operand shapes subtracted from the operand shape.

This function considers the shape fill, the style modifications, and the transform mapping of the target and operand shapes. Only areas that are drawn are considered when calculating the difference.

Neither the target shape nor the operand shape have one of the framed fills (open-frame or closed-frame).

Geometric Operations

If you provide a target or operand shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Posts the error <code>shape_operator_may_not_be_a_bitmap</code>
picture	Posts the error <code>shape_operator_may_not_be_a_picture</code>
text	Converts to path shape if other parameter is not an empty shape or a full shape
glyph	Converts to path shape if other parameter is not an empty shape or a full shape
layout	Converts to path shape if other parameter is not an empty shape or a full shape

ERRORS, WARNINGS, AND NOTICES

Errors	
<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>number_of_contours_exceeds_implementation_limit</code>	
<code>number_of_points_exceeds_implementation_limit</code>	
<code>size_of_path_exceeds_implementation_limit</code>	
<code>size_of_polygon_exceeds_implementation_limit</code>	
<code>fill_type_not_allowed</code>	(debugging version)
<code>shape_access_not_allowed</code>	(debugging version)
<code>clip_to_frame_shape_unimplemented</code>	(debugging version)
<code>shape_operator_may_not_be_a_bitmap</code>	(debugging version)
<code>shape_operator_may_not_be_a_picture</code>	(debugging version)
Warnings	
<code>character_substitution_took_place</code>	
<code>font_substitution_took_place</code>	
<code>unable_to_traverse_open_contour_that_starts_or_ends_off_the_curve</code>	(debugging version)

SEE ALSO

For examples using this function, see “Performing Geometric Arithmetic With Shapes” beginning on page 4-60.

For a discussion of geometric arithmetic, see “Geometric Arithmetic” beginning on page 4-21.

For a discussion of shape fills, see Chapter 2, “Geometric Shapes,” in this book.

For a discussion of style modifications, see Chapter 3, “Geometric Styles,” in this book.

Geometric Operations

For a discussion of transform mappings, see the chapter “Transform Objects” in *Inside Macintosh: QuickDraw GX Objects*.

For information about related functions, see the description of the `GXIntersectShape` function on page 4-107, the `GXUnionShape` function on page 4-109, and the `GXDifferenceShape` function on page 4-110.

GXExcludeShape

You can use the `GXExcludeShape` function to find the result of performing the exclusive-OR operation on two shapes.

```
void GXExcludeShape(gxShape target, gxShape operand);
```

<code>target</code>	On input, a reference to one of the shapes on which to perform the exclusive-OR operation. On output, a reference to a shape describing the exclusive-OR of the two shapes.
<code>operand</code>	A reference to the other shape on which to perform the exclusive-OR operation.

DESCRIPTION

The `GXExcludeShape` function performs an exclusive-OR operation on the `target` and `operand` shapes, and stores the result in the `target` shape.

The exclusion of two shapes (the result of the exclusive-OR operation) is the area contained by the union of the two shapes less the area contained by the intersection of the two shapes.

This function considers the shape fill, the style modifications, and the transform mapping of the `target` and `test` shapes. Only areas that are drawn are considered when calculating the difference.

Neither the `target` shape nor the `operand` shape may have one of the framed fills (open-frame or closed-frame).

If you provide a `target` or `operand` shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Posts the error <code>shape_operator_may_not_be_a_bitmap</code>
picture	Posts the error <code>shape_operator_may_not_be_a_picture</code>
text	Converts to path shape if other parameter is not an empty shape or a full shape
glyph	Converts to path shape if other parameter is not an empty shape or a full shape
layout	Converts to path shape if other parameter is not an empty shape or a full shape

Geometric Operations

ERRORS, WARNINGS, AND NOTICES

Errors

out_of_memory	
shape_is_nil	
number_of_contours_exceeds_implementation_limit	
number_of_points_exceeds_implementation_limit	
size_of_path_exceeds_implementation_limit	
size_of_polygon_exceeds_implementation_limit	
fill_type_not_allowed	(debugging version)
shape_access_not_allowed	(debugging version)
clip_to_frame_shape_unimplemented	(debugging version)
shape_operator_may_not_be_a_bitmap	(debugging version)
shape_operator_may_not_be_a_picture	(debugging version)

Warnings

character_substitution_took_place	
font_substitution_took_place	
unable_to_traverse_open_contour_that_starts_or_ends_off_the_curve	(debugging version)

SEE ALSO

For examples using this function, see “Performing Geometric Arithmetic With Shapes” beginning on page 4-60.

For a discussion of geometric arithmetic, see “Geometric Arithmetic” beginning on page 4-21.

For a discussion of shape fills, see Chapter 2, “Geometric Shapes,” in this book.

For a discussion of style modifications, see Chapter 3, “Geometric Styles,” in this book.

For a discussion of transform mappings, see the chapter “Transform Objects” in *Inside Macintosh: QuickDraw GX Objects*.

For information about related functions, see the description of the `GXIntersectShape` function on page 4-107, the `GXUnionShape` function on page 4-109, the `GXDifferenceShape` function on page 4-110, and the `GXReverseDifferenceShape` function on page 4-112.

GXInvertShape

You can use the `GXInvertShape` function to invert a shape.

```
void GXInvertShape(gxShape target);
```

`target` A reference to the shape to invert.

DESCRIPTION

The `GXInvertShape` function inverts the target shape and stores the resulting shape in the target shape. Typically, this function changes the shape fill of the target shape. It also converts empty shapes to full shapes and full shapes to empty shapes.

If the target shape has one of the framed shape fills (open-frame or closed-frame), this function posts the error `shape_cannot_be_inverted`.

For empty shapes, this function converts the shape to a full shape; for full shapes, it converts to empty shapes.

If you provide a target or operand shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Posts the error <code>shape_cannot_be_inverted</code>
picture	Posts the error <code>shape_cannot_be_inverted</code>
text	Posts the error <code>shape_cannot_be_inverted</code>
glyph	Posts the error <code>shape_cannot_be_inverted</code>
layout	Posts the error <code>shape_cannot_be_inverted</code>

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>shape_cannot_be_inverted</code>	(debugging version)
<code>shape_access_not_allowed</code>	(debugging version)

SEE ALSO

For an example using this function, see “Performing Geometric Arithmetic With Shapes” beginning on page 4-60.

For a discussion of geometric arithmetic, see “Geometric Arithmetic” beginning on page 4-21.

For a discussion of shape fills, see Chapter 2, “Geometric Shapes,” in this book.

Summary of Geometric Operations

Constants and Data Types

Contour Directions

```
enum gxContourDirections {
    gxCounterclockwiseDirection,
    gxClockwiseDirection
};

typedef long gxContourDirection;
```

Functions

Determining and Reversing Contour Direction

```
gxContourDirection GXGetShapeDirection
    (gxShape source, long contour);
void GXReverseShape    (gxShape target, long contour);
```

Breaking Shape Contours

```
void GXBreakShape    (gxShape target, long index);
```

Reducing and Simplifying Shapes

```
void GXReduceShape    (gxShape target, long contour);
void GXSimplifyShape   (gxShape target);
```

Incorporating Style Information Into Shape Geometries

```
void GXPrimitiveShape    (gxShape target);
```

Finding Geometric Information About Shapes

```
gxWide *GXGetShapeLength    (gxShape source, long index, gxWide *length);
gxPoint *GXShapeLengthToPoint
    (gxShape target, long index, Fixed length,
     gxPoint *location, gxPoint *tangent);
gxPoint *GXGetShapeCenter    (gxShape source, long index, gxPoint *center);
gxWide *GXGetShapeArea    (gxShape source, long index, gxWide *area);
```

Getting and Setting Shape Bounds

```

gxRectangle *GXGetShapeBounds
                                (gxShape source, long index,
                                 gxRectangle *bounds);

void GXSetShapeBounds           (gxShape target, const gxRectangle *newBounds);

```

Insetting Shapes

```

void GXInsetShape               (gxShape target, Fixed inset);

```

Determining Whether Two Shapes Touch

```

gxBoolean GXTouchesRectanglePoint
                                (const gxRectangle *target,
                                 const gxPoint *test);

gxBoolean GXTouchesBoundsShape
                                (const gxRectangle *target, gxShape test);

gxBoolean GXTouchesShape       (gxShape target, gxShape test);

```

Determining Whether One Shape Contains Another

```

gxBoolean GXContainsRectangle
                                (const gxRectangle *container,
                                 const gxRectangle *test);

gxBoolean GXContainsBoundsShape
                                (const gxRectangle *container, gxShape test,
                                 long index);

gxBoolean GXContainsShape       (gxShape container, gxShape test);

```

Performing Geometric Arithmetic With Shapes

```

gxBoolean GXIntersectRectangle
                                (gxRectangle *target, const gxRectangle *source,
                                 const gxRectangle *operand);

gxRectangle *GXUnionRectangle
                                (gxRectangle *target,
                                 const gxRectangle *source,
                                 const gxRectangle *operand);

void GXIntersectShape           (gxShape target, gxShape operand);
void GXUnionShape               (gxShape target, gxShape operand);
void GXDifferenceShape          (gxShape target, gxShape operand);
void GXReverseDifferenceShape    (gxShape target, gxShape operand);
void GXExcludeShape             (gxShape target, gxShape operand);
void GXInvertShape              (gxShape target);

```