NOTE: This Technical Note has been retired. Please see the Technical Notes page for current documentation.

Technical Note TN1037

QuickDraw GX MappingLibrary.c: Its Uses and Limitations

CONTENTS

This Technote discusses MappingLibrary.c from the QuickDraw GX Libraries.

This Note is intended for Macintosh QuickDraw GX developers who are using MappingLibrary.c or who are considering using it for their QuickDraw GX graphics applications.

Important for all Apple Printing and Graphics Developers:

The information in this Technote is still relevant up to and including Mac OS 7.6 with QuickDraw GX 1.1.5. Beginning with the release of Mac OS 8.0, however, Apple plans to deliver a system which incorporates QuickDraw GX graphics and typography **only**. QuickDraw GX printer drivers and GX printing extensions will **not** be supported in Mac OS 8.0 or in future Mac OS releases. Apple's goal is to simplify the user experience of printing by unifying the Macintosh graphic and printing architectures and standardizing on the classic Printing Manager.

For details on Apple's official announcement, refer to </technotes/gxchange.html>

Updated: [Mar 1 1996]

About the GX Libraries

For better or worse, the development of QuickDraw GX took seven years from conception to initial release. During that time, there were many requests for feature enhancements and interface improvements that, if implemented, might have taken seven more years to complete. As it turns out, some of these enhancements could readily be built on existing services, but there was no time to test or document these services with the rigor required to make them fully part of the released system.

The GX Libraries fill this gap by providing services built on top of the rest of GX in source form. This Technote and others document these services. Since GX libraries are provided as source, it is reasonable for developers to modify them to meet their specific needs. Care was taken for the libraries not to depend on the implementation details of GX, so that future versions of GX should not invalidate them, in original or modified form.

The libraries are likely to evolve to take advantage of improved algorithms, new Macintosh or GX services; if you modify one for your application's specific needs, it's worth occasionally reviewing the GX library provided by Apple to stay synchronized with any improvements.

Back to top

What is in MappingLibrary.c?

The GX Library, MappingLibrary.c, generates mappings that project the coordinates of one polygon onto another. It was written by Robert Johnson, the resident mathematics whiz.

MappingLibrary.c has one multi-purpose function: `PolyToPolyMap`. Its declaration is:

```
void PolyToPolyMap(const gxPolygon *source, const gxPolygon *dest,
                   gxMapping *map)
```

This function takes a pair of polygons as parameters and returns a mapping. Both polygons should have the same number of points; the number can vary from 1 to 4. Thus, both polygons contain either a point, a line, a triangle, or a quadrilateral. The mapping returned by the function projects the first polygon onto the second.

> **Note:**
> `PolyToPolyMap` is roughly the inverse function of `MapPoints`, which computes how a mapping will transform a list of points. For more information on `MapPoints`, see p. 8-66 of *Inside Macintosh: QuickDraw GX Environment and Utilities.*

Back to top

Working with MappingLibrary.c

`PolyToPolyMap` can be used to compute the mapping that projects one shape into a specified area. Just as the QuickDraw routines `MapPt`, `MapRect`, `MapRgn` and `MapPoly` allow mapping geometries from a source rectangle to a destination rectangle, `PolyToPolyMap` can be used to do the same thing with any QuickDraw GX shape. Here's an example:

```
#include "GraphicsLibraries.h"

static void MapAnything(gxShape shape, gxRectangle source, gxRectangle destination)
{
    struct{
                long countOf2;
                gxPoint points[2];
             } sourcePoly, destPoly;
    gxMapping map;

    sourcePoly.countOf2 = 3;
    sourcePoly.points[0].x = source.left;
    sourcePoly.points[0].y = source.top;
    sourcePoly.points[1].x = source.right;
    sourcePoly.points[1].y = source.top;
    sourcePoly.points[2].x = source.right;
    sourcePoly.points[2].y = source.bottom;
    destPoly.countOf2 = 2;
    destPoly.points[0].x = destination.left;
    destPoly.points[0].y = destination.top;
    destPoly.points[1].x = destination.right;
    destPoly.points[1].y = destination.top;
    destPoly.points[2].x = destination.right;
    destPoly.points[2].y = destination.bottom;

    PolyToPolyMap((gxPolygon*) &sourcePoly, (gxPolygon*) &destPoly,
                        &map);
    GXMapShape(shape, &map);
}
```

This example describes the two rectangles as a pair of triangles from the upper left to the upper right to the lower right. The mapping returned by `PolyToPolyMap` maps the first triangle to the second. Similarly, it will map the shape so that its former scale and position relative to the first rectangle is the same as its subsequent scale and position relative to the second rectangle.

If the two polygons passed to `PolyToPolyMap` contain points, the resulting mapping will translate one point to the other. If they contain lines, the mapping translates and scales. If they contain triangles, the mapping is affine; that is, it translates, scales, skews and/or rotates from one to the other. If they contain quadrilaterals, then the perspective elements of the matrix can be used as well to project one to the other.

Back to top

PolyToPolyMap Limitations and Solutions

There are limitations that developers need to be aware of. For instance, if the quadrilateral points form a bow tie (they cross), then `PolyToPolyMap` can't succeed. It will likely generate an overflow in one of the math calculations, and pin the resulting value to `0x7FFFFFFF` or `0x80000000` (GX's equivalents to plus and minus infinity). You may also find that more pedestrian pairs of quadrilaterals can fail.

This is why the libraries are supplied in source form -- so developers can fix and personalize them.

A Sample Quadrilateral Limitation and Solutions

The limitation that causes some quadrilaterals to fail is in lines 107 through 115 of MappingLibrary.c; the calls to `FractDivide` can go out of range. One simple solution is to replace these calls with `FixedDivide` and replace the four

occurrences of FractMultiply in lines 117 through 121 with FixedMultiply. This may affect the accuracy of the results, however.

A more ambitious solution is to determine the number of significant bits in the relevant variables, then maximize the precision without overflowing.

Replace lines 106 - 109 with:

```
        Fract a1;
        Fract numerator, denominator;
        if ( x2 > 0 ? y2 > 0 ? x2 > y2 : x2 > -y2 : y2 > 0 ? -x2 > y2 : x2
                    < y2) {
             numerator = MultiplyDivide(x0 - x1, y2, x2) - y0 + y1;
            denominator = MultiplyDivide(x1, y2, x2) - y1;
    } else {
            numerator = x0 - x1 - MultiplyDivide(y0 - y1, x2, y2);
            denominator = x1 - MultiplyDivide(y1, x2, y2);
    }
    Fract absNum = numerator, absDenom = denominator;
    if (absNum < 0)
        absNum = -absNum;
    if (absDenom < 0)
        absDenom = -absDenom;
    absDenom += absDenom;
    int a1Shift = 0;
    while (absNum >= absDenom) {
        a1Shift++;
        absNum <<= 1;
    }
    a1 = FractDivide(numerator << a1Shift, denominator);
    scaleY <<= a1Shift;
```

(You'll have to twiddle the variable declarations if you're using C instead of C++.)

Repeat this substitution in lines 112 - 115 to compute a2, a2Shift and scaleX.

Then replace line 120 with:

```
    *destPtr++ = FixedDivide(FractMultiply(a1, source[1].x) +
        (source[1].x - source[0].x << a1Shift), scaleY);
```

Similarly, update lines 117, 118 and 121.

The complete integration of these changes is left to the reader.

With these changes, PolyToPolyMap will return the correct result for most visibly useful pairs of polygons. In the four point case, useful polygons are ones that are convex; that is, they shouldn't look like arrowheads, have consecutive line segments that form a straight line, or as mentioned before, cross over themselves. In the three point case, useful polygons enclose an area; that is, the triangles do not degenerate into lines; and in the two point case, the lines should not degenerate into points.

Back to top

Summary

GX Libraries have a wealth of information and show how to use QuickDraw GX to solve real problems. The Mapping Library shows how to use GX to construct mappings that project shapes into the desired coordinates so that one to four reference points match up. This makes creating perspective mappings easy without resorting to using QuickDraw 3D.

Back to top

References

MacOS SDK CD, Development Kits (Disc 1): QuickDraw GX: Programming Stuff: GX Libraries

Inside Macintosh: QuickDraw GX Objects

Inside Macintosh: QuickDraw GX Environment and Utilities

Back to top

Downloadables

 Acrobat version of this Note (K). Download

Back to top
