AMERICAN INTERACTIVE MEDIA

AIM Technical Note

TN#34: DYUV Panning Algorithms

Written by: Dave Townsend, Capitol Disc April 17, 1989

Dave Townsend, from Capitol Disc in Washington prepared the following document on DYUV Panning algorithms.

DYUV PANNING

I'll discuss first the general case of panning around a large drawmap. This is a fairly straightforward task. You just grab the proper range of addresses out of the drawmap's line address table to determine your vertical position, and the add in your horizontal offset. The resulting addresses are then written into the LCT. In code terms, the instructions are of the form:

```
-----
dc_wrli(vpath, lct, sline, 0, cp_dadr((int) dmap->dm_lnatbl1[vert] + horiz));
-----
```

You then accomplish panning by varying <vert> and <horiz> and re-writing the LCT.
For CLUT and RGB555 drawmaps, this technique works fine. However, DYUV drawmaps need some additional processing, since each pixel is a delta value rather than an absolute value. That is, each pixel is depending on the video decoder to have processed the previous pixel.
Vertical panning still works for DYUV drawmaps, since the starting YUV value is reset by the hardware on each LCT line. It's only the horizontal direction that causes problems.
What you must do is reset the starting DYUV value yourself each time that you move horizontally. You will need a separate start value for every line. The amounts to adjust the starting values are, of course, determined by the pixel data itself.
Although each of the three components (Y, U, and V) is delta-encoded, it's easier for example purposes to deal with only one dimension. Here's some sample pixel data for a single scan line of a drawmap:

```
-----
--------------------------------
| 10 | -3 | 20 | 11    8 |-12 | ...
--------------------------------
-----
```

Let's assume that the start value is 1. So, if you display the drawmap with a horizontal offset of 0, you'd use a starting value of 1. If you then move right one pixel, you'd need to add the value of the previous pixel (10 in this case) to the previous start value (1) to get the new start value (11). If you move right one more pixel, your start value would be 8 (== 1 + 10 - 3). And so on.

That's the basic idea of DYUV panning. Now, let's move from the general idea to the specific application.

The encoding method of DYUV drawmaps (see Green Book section 4.4.2.1, pg. V-41) adds some interesting wrinkles to the problem. Two pixels are encoded in 16 bits, but it is NOT the case that a single pixel is encoded in 8 bits! By Green Book rules, the smallest horizontal pan is two pixels.
Each of the three video components (Y, U, and V) need to be operated on like the single-dimensional example above. Within the sixteen bits are TWO deltas for Y and a single delta each for U and V. Both Y deltas are just added sequentially to the Y component; there's nothing tricky there.
The tricky part is that the 4-bit values U, V, Y0 and Y1 are not really the deltas themselves; they need to be translated into 8-bit values. See Green Book, Fig. V.16, p. V-25 for the translations. I use a 16-byte lookup table like the following:

```
-----
unsigned char c4to8[16] = {
                             0,   1,   4,   9,  16,  27,  44,  79,
                           128, 177, 212, 229, 240, 247, 252, 255
                           };
-----
```

I then use code like the following to compute a starting value for drawmap line <dline> at horizontal offset <h_offset>:

```
-----
unsigned char y, u, v;
unsigned char *cp;        /* current position in line */
y = START_Y;         /* default starting value for when at left edge of drawmap */
u = START_U;
v = START_V;
cp = dmap->dm_lnatbl1[dline];                    /* 1st byte in 1st pixel pair */
while (cp < (dmap->dm_lnatbl1[dline] + h_offset))
     {
     u += c4to8[*cp   >>  4];        /* pull out U component */
     y += c4to8[*cp++ & 0xf];        /* get Y0, point to 2nd byte in pair */
     v += c4to8[*cp   >>  4];        /* get V */
     y += c4to8[*cp++ & 0xf];        /* get Y1, go on to next pair */
     }
start_value = cp_yuv(plane_whatever, y, u, v);

-----
```

Since all the arithmetic is supposed to be modulo 256, unsigned chars fit the bill nicely. You could also treat the pixel pairs as a short int, but I think that this method is a little faster.

Speed is an important issue. Obviously, the larger your horizontal offset, the longer the above loop is going to take. It doesn't take very long at all for the speed to become unacceptable.

A solution is save some state information. Rather than starting from the beginning of the drawmap line for each starting YUV computation, use the previous starting value modified and just add in the pixels that "disappear". When you pan left, you subtract off the pixels that "reappear".

This method doesn't help for pans by large amounts, but has proven quite acceptable for medium-speed smooth pans.

For vertical pans, you can need only compute the starting values for newly-appearing row; most of the old starting values can probably be reused.

The encoding method of DYUV drawmaps (see Green Book section 4.4.2.1, pg. V-41) adds some interesting wrinkles to the problem. Two pixels are encoded in 16 bits, but it is NOT the case that a single pixel is encoded in 8 bits! By Green Book rules, the smallest horizontal pan is two pixels.
Each of the three video components (Y, U, and V) need to be operated on like the single-dimensional example above. Within the sixteen bits are TWO deltas for Y and a single delta each for U and V. Both Y deltas are just added sequentially to the Y component; there's nothing tricky there.
The tricky part is that the 4-bit values U, V, Y0 and Y1 are not really the deltas themselves; they need to be translated into 8-bit values. See Green Book, Fig. V.16, p. V-25 for the translations. I use a 16-byte lookup table like the following:

```
-----
unsigned char c4to8[16] = {
                                0,   1,   4,   9,  16,  27,  44,  79,
                              128, 177, 212, 229, 240, 247, 252, 255
                              };
-----
```

I then use code like the following to compute a starting value for drawmap line <dline> at horizontal offset <h_offset>:

```
-----
unsigned char y, u, v;
unsigned char *cp;          /* current position in line */
y = START_Y;          /* default starting value for when at left edge of drawmap */
u = START_U;
v = START_V;
cp = dmap->dm_lnatbl1[dline];                        /* 1st byte in 1st pixel pair */
while (cp < (dmap->dm_lnatbl1[dline] + h_offset))
    {
    u += c4to8[*cp   >>  4];          /* pull out U component */
    y += c4to8[*cp++ & 0xf];          /* get Y0, point to 2nd byte in pair */
    v += c4to8[*cp   >>  4];          /* get V */
    y += c4to8[*cp++ & 0xf];          /* get Y1, go on to next pair */
    }
start_value = cp_yuv(plane_whatever, y, u, v);

-----
```

Since all the arithmetic is supposed to be modulo 256, unsigned chars fit the bill nicely. You could also treat the pixel pairs as a short int, but I think that this method is a little faster.

Speed is an important issue. Obviously, the larger your horizontal offset, the longer the above loop is going to take. It doesn't take very long at all for the speed to become unacceptable.

A solution is save some state information. Rather than starting from the beginning of the drawmap line for each starting YUV computation, use the previous starting value modified and just add in the pixels that "disappear". When you pan left, you subtract off the pixels that "reappear".

This method doesn't help for pans by large amounts, but has proven quite acceptable for medium-speed smooth pans.

For vertical pans, you can need only compute the starting values for newly-appearing row; most of the old starting values can probably be reused.