Subject: Re: Inverse palette conversion algorithm...
Posted by mr_ped on Sun, 21 May 2006 13:40:12 GMT
View Forum Message <> Reply to Message

I was unable to look at it sooner. :/

Well, I think swapped those two versions of code when I was benchmarking it first time, so in reality your code was faster.
I don't understand how I could have done such mistake. Feels so embarassing, sorry.

Anyway, the improved version of my parallel flood fill is still 2 times slower, but it improved by 50% when compared the old version, so I will post it anyway just if anyone is curious, how the same algorithm can be speeded up by 50%.
It can be a tad faster (5%) if you inline the AddPoint function, but I tried to rather keep it more readable, than fast.

```
/*
  - new distance square "(d+x)^2" calculation (from old distance square "d^2" and old delta "d")
   delta will change by +-4 (blue), +-2 (red) or +-1 (green)
   +-1 : (d+1)^2 = d^2 + (2d + 1)  max 62->63: 125  (max distance delta for green)
   +-2 : (d+2)^2 = d^2 + (4d + 4)  nax 60->62: 244
   +-4 : (d+4)^2 = d^2 + (8d + 16)  max 56->60: 464
*/

#ifdef COMPILER_MSC
#pragma pack(push, 1)
#endif
struct sCubePoint3 : Moveable<sCubePoint3> {
 word   address;  //high-color 5:6:4 = address into cube space too
 byte   index;   //index into palette
 int8   delta[3];  //current deltas of this point from origin
}
#ifdef COMPILER_GCC
__attribute__((packed))
#endif
;
#ifdef COMPILER_MSC
#pragma pack(pop)
#endif

//conversion map
struct PaletteCv3 {
 enum {
  MAX_DISTANCE_DELTA  = (8*56+16),

  R_SHIFT    = 10,
  G_SHIFT    = 0,
```

```
  B_SHIFT    = 6,
  R_MASK    = ((RASTER_MAP_R-1)<<R_SHIFT),
  G_MASK    = ((RASTER_MAP_G-1)<<G_SHIFT),
  B_MASK    = ((RASTER_MAP_B-1)<<B_SHIFT),
  R_ADR_ADD   = (1<<R_SHIFT),
  G_ADR_ADD   = (1<<G_SHIFT),
  B_ADR_ADD   = (1<<B_SHIFT),
 };

 Buffer<byte> cv;
 static inline word GetIndex(const RGBA &c) {
  return (word(c.r >> RASTER_SHIFT_R) << R_SHIFT) +
         (word(c.g >> RASTER_SHIFT_G) << G_SHIFT) +
         (word(c.b >> RASTER_SHIFT_B) << B_SHIFT); }
 byte Get(const RGBA& c) const { return cv[GetIndex(c)]; }
 PaletteCv3() { cv.Alloc(RASTER_MAP_R * RASTER_MAP_G * RASTER_MAP_B); }
};

//generator of data for conversion maps
struct sPalCv3 {
 PaletteCv3&   cv_pal;
 const RGBA    *palette;
 int           ncolors;
 //FIFO queue for parallel flood fill, radix-sorted by distance of points from their origin
 //the radix sort works on dynamic subset of distances, as you never need the full range during fill
 Vector<sCubePoint3> feed_me[PaletteCv3::MAX_DISTANCE_DELTA+1];
 byte    filled[RASTER_MAP_R * RASTER_MAP_G * RASTER_MAP_B];

 void AddPoint(const sCubePoint3 & cubpt, int ii, word add2address, int move, int a, int8
add2delta);

 sPalCv3(const RGBA *palette, int ncolors, PaletteCv3& cv_pal);
};

struct sFillMovementData {
 word   mask;
 word   mask_delta;
 int8   delta;
 int    a, b;
};
static const sFillMovementData fmovedata[3] =
{
 {PaletteCv3::R_MASK, PaletteCv3::R_ADR_ADD, 2, 4, 4},
 {PaletteCv3::G_MASK, PaletteCv3::G_ADR_ADD, 1, 2, 1},
 {PaletteCv3::B_MASK, PaletteCv3::B_ADR_ADD, 4, 8, 16},
};

void sPalCv3::AddPoint(const sCubePoint3 & cubpt, int ii, word add2address, int move, int a, int8
```

```
add2delta)
{
 int newii;
 sCubePoint3 cubpt2;

 cubpt2.address = cubpt.address + add2address;
 if ( filled[cubpt2.address] ) return;

 newii = ii + a * cubpt.delta[move] + fmovedata[move].b;
 ASSERT( newii > ii );
 if ( newii > PaletteCv3::MAX_DISTANCE_DELTA ) {
  newii -= PaletteCv3::MAX_DISTANCE_DELTA+1;
  ASSERT( newii <= PaletteCv3::MAX_DISTANCE_DELTA );
  ASSERT( newii < ii );
 }
 cubpt2.delta[0] = cubpt.delta[0];
 cubpt2.delta[1] = cubpt.delta[1];
 cubpt2.delta[2] = cubpt.delta[2];
 cubpt2.index = cubpt.index;
 cubpt2.delta[move] += add2delta;
 feed_me[newii].Add(cubpt2);
}


sPalCv3::sPalCv3(const RGBA *palette, int ncolors, PaletteCv3& cv_pal)
: cv_pal(cv_pal), ncolors(ncolors), palette(palette)
{
 int   ii, jj = (RASTER_MAP_R * RASTER_MAP_G * RASTER_MAP_B), move;
 sCubePoint3 cubpt;

 ZeroArray(filled);
 feed_me[0].Reserve(ncolors);
 //Fill up the FIFO queue with colors from palette,
 //those will start the parallel flood fill in the color cube space
 ii = ncolors;
 cubpt.delta[0] = cubpt.delta[1] = cubpt.delta[2] = 0;
 while (ii--) {
  cubpt.index  = ii;
  cubpt.address = cv_pal.GetIndex(palette[ii]);
  feed_me[0].Add(cubpt);
 }
 //process the FIFO queue untill all points in color cube space are filled (jj == 0)
 ii = 0;
 while ( true ) {
  //if ( !feed_me[ii].IsEmpty() ) printf("%d\t(%d)\t", ii, feed_me[ii].GetCount());
  while ( !feed_me[ii].IsEmpty() ) {
   cubpt = feed_me[ii].Pop();
   if ( filled[cubpt.address] ) continue;
```

```
   cv_pal.cv[cubpt.address] = cubpt.index;
   if ( --jj == 0 ) return;
   filled[cubpt.address] = 1;
   //try all possible moves (6 possible directions)
   for ( move = 0; move < 3; ++move )
   {
    if ( ( cubpt.delta[move] >= 0 ) &&
      ( (cubpt.address & fmovedata[move].mask) < fmovedata[move].mask ) )
      AddPoint(cubpt, ii, fmovedata[move].mask_delta, move, fmovedata[move].a,
fmovedata[move].delta);
    if ( ( cubpt.delta[move] <= 0 ) &&
      ( (cubpt.address & fmovedata[move].mask) > 0 ) )
      AddPoint(cubpt, ii, -fmovedata[move].mask_delta, move, -fmovedata[move].a,
-fmovedata[move].delta);
   }
  }
  if ( ++ii > PaletteCv3::MAX_DISTANCE_DELTA ) ii = 0;
 }
 return;
}
```

File Attachments

1) test_upp_console.zip, downloaded 1956 times