Subject: Inverse palette conversion algorithm...
Posted by mirek on Wed, 05 Apr 2006 08:11:49 GMT
View Forum Message <> Reply to Message

I have spend a 3 days optimizing this little snippet (for new Image) and I think it is quite interesting implementation, so I am posting it here so that fruits of my efforts are a little bit more public:

```
enum {
 RASTER_MAP_R = 32,
 RASTER_SHIFT_R = 3,
 RASTER_MAP_G = 64,
 RASTER_SHIFT_G = 2,
 RASTER_MAP_B = 16,
 RASTER_SHIFT_B = 4,

 RASTER_MAP_MAX = 64
};

struct PaletteCv {
 Buffer<byte> cv;

 byte *At(int r, int b)          { return cv + (r << 10) + (b << 6); }
 byte  Get(const RGBA& b) const   { return cv[(int(b.r >> RASTER_SHIFT_R) << 10) +
                                     (int(b.g >> RASTER_SHIFT_G)) +
                                     (int(b.b >> RASTER_SHIFT_B) << 6)]; }
 PaletteCv()                     { cv.Alloc(RASTER_MAP_R * RASTER_MAP_G * RASTER_MAP_B); }
};

struct sPalCv {
 PaletteCv&  cv_pal;
 const RGBA *palette;
 int         ncolors;
 bool        done[RASTER_MAP_G];
 struct Ginfo {
  int   dist;
  byte  g;
  byte  ii;
 };

 enum { BINS = 16, BINSHIFT = 11 };

 Ginfo       line[BINS][256];
 Ginfo      *eline[BINS];
 byte       *gline;

 static int Sq(int a, int b) { return (a - b) * (a - b); }
```

```
 void SetLine(int r, int b);
 int  Get(int g);

 sPalCv(const RGBA *palette, int ncolors, PaletteCv& cv_pal);
};

void sPalCv::SetLine(int r, int b)
{
 gline = cv_pal.At(r, b);
 r = 255 * r / (RASTER_MAP_R - 1);
 b = 255 * b / (RASTER_MAP_B - 1);
 for(int i = 0; i < BINS; i++)
  eline[i] = line[i];
 for(int i = 0; i < ncolors; i++) {
  int dist = Sq(palette[i].r, r) + Sq(palette[i].b, b);
  int bini = dist >> BINSHIFT;
  Ginfo *t = eline[bini >= BINS ? BINS - 1 : bini]++;
  t->dist = dist;
  t->g = palette[i].g;
  t->ii = i;
 }
 ZeroArray(done);
}

int sPalCv::Get(int g)
{
 if(done[g])
  return gline[g];
 int gg = 255 * g / (RASTER_MAP_G - 1);
 int ii = 0;
 int dist = INT_MAX;
 for(int th = 0; th < BINS; th++) {
  Ginfo *s = line[th];
  Ginfo *e = eline[th];
  while(s < e) {
   int sdist = Sq(s->g, gg) + s->dist;
   if(sdist < dist) {
    ii = s->ii;
    dist = sdist;
   }
   s++;
  }
  if(th < BINS - 1 && dist < ((th + 1) << BINSHIFT))
   break;
 }
 done[g] = true;
 gline[g] = ii;
 return ii;
```

```
}

sPalCv::sPalCv(const RGBA *palette, int ncolors, PaletteCv& cv_pal)
: cv_pal(cv_pal), ncolors(ncolors), palette(palette)
{
 byte ender[256];
 for(int b = 0; b < RASTER_MAP_B; b++) {
  ZeroArray(ender);
  for(int r = 0; r < RASTER_MAP_R; r++) {
   SetLine(r, b);
   int g = 0;
   while(g < RASTER_MAP_G) {
    int ii = Get(g);
    int eg = max<int>(g, ender[ii]);
    if(Get(eg) == ii)
     while(eg < RASTER_MAP_G - 1 && Get(eg + 1) == ii)
      eg++;
    else
     while(Get(eg) != ii)
      eg--;
    ender[ii] = eg;
    g++;
    while(g <= eg - 1) {
     gline[g] = ii;
     done[g] = true;
     g++;
    }
   }
  }
 }
}
```

(BTW, level of optimization: This is almost 20 times faster than brute-force algorithm...)

Mirek

Subject: Re: Inverse palette conversion algorithm...
Posted by mr_ped on Fri, 07 Apr 2006 08:05:36 GMT

View Forum Message <> Reply to Message

I'm not sure what's this about and supposed to do,
I just catched up the "r = 255 * r / (RASTER_MAP_R - 1);" lines...

Why to do this with such expensive calculation? (1 multiply + 1 divide)

```
enum {
   ...
   RASTER_SHIFT2_R = ((4-RASTER_SHIFT_R)*2),
   RASTER_SHIFT2_G = ((4-RASTER_SHIFT_G)*2),
   RASTER_SHIFT2_B = ((4-RASTER_SHIFT_B)*2),
};

r = (r<<RASTER_SHIFT_R) + (r>>RASTER_SHIFT2_R);
g = (g<<RASTER_SHIFT_G) + (g>>RASTER_SHIFT2_G);
b = (b<<RASTER_SHIFT_B) + (b>>RASTER_SHIFT2_B);
```

... These are less expensive for older CPU (and at least of the same speed on modern ones), and provide a better distribution of converted values IMHO.

Than again I'm curious about the whole purpose of this piece of code.

Subject: Re: Inverse palette conversion algorithm...
Posted by mirek on Fri, 07 Apr 2006 09:16:00 GMT
View Forum Message <> Reply to Message

What a nice trick, I did not knew this one! Thanks!

As for the purpose of the code: You have truecolor raster image and you have some existing palette. You want to convert truecolor to palette (indexed).

Now in theory, you should for each color of image find closest color - by equaction (r1 - r2)^2 + (g1 - g2)^2 + (b1 - b2)^2 (where ^2 is not xor but square).

Of course, performing this calculation for each pixel would be too slow, therefore standard approach reduces original pixel depth (to 64K colors here) and provides "inverse transformation cube" - a table that can be used to perform quick lookup of palette index for each possible color.

This algorithm builds such cube. "Brute force approach" is to go through all cells of cube and find closest color from palette for it. Above implementation is almost 20x faster in the average case...

Mirek

Subject: Re: Inverse palette conversion algorithm...
Posted by mr_ped on Fri, 07 Apr 2006 11:16:43 GMT
View Forum Message <> Reply to Message

1) why to use indexed color? (ok, sometimes it's still really needed, but usually 32b ARGB is the way to go)
2) so you speeded up 20 times code which is run single time during init and takes under 100ms even with brute-force algorithm, and you needed 3 days to do that.

2 - related) in case you generate the conversion table more than one time, there's something fishy about that. (and looks like you can speed up the thing much more just by omiting multiple initialization of the same table)

While I like the actual trick and will probably check it more deeply to understand fully your approarch, I'm not sure it was worth of the effort in "economical" sense.
Than again, if you enjoyed it, it was probably worth of it anyway.

Ad my trick:
Beware it will stop to work if you get negative numbers in ((4-RASTER_SHIFT_X)*2) formula (i.e. RASTER_MAP_X is under 16).

(Also using signed "int" is not really safe when working with ARGB color channels, because the ">>" will be compiled as SAL, not SHL. As long as you are sure the MSb is zero it doesn't matter, but once you start to use alpha channel or "signed short" on 16bpp colors you are asking for trouble ... usually it's best to keep colors and channels in unsigned byte/word/dword)

Subject: Re: Inverse palette conversion algorithm...
Posted by mr_ped on Fri, 07 Apr 2006 11:23:46 GMT
View Forum Message <> Reply to Message

luzr wrote on Fri, 07 April 2006 11:16(to 64K colors here)

Just 32K in your code. (5bits for red, 6 for green, 4 for blue = 15 bits total)

Subject: Re: Inverse palette conversion algorithm...
Posted by mirek on Fri, 07 Apr 2006 11:33:19 GMT
View Forum Message <> Reply to Message

mr_ped wrote on Fri, 07 April 2006 07:161) why to use indexed color? (ok, sometimes it's still really needed, but usually 32b ARGB is the way to go)
2) so you speeded up 20 times code which is run single time during init and takes under 100ms even with brute-force algorithm, and you needed 3 days to do that.
2 - related) in case you generate the conversion table more than one time, there's something fishy about that. (and looks like you can speed up the thing much more just by omiting multiple initialization of the same table)

Add 1) Yes, of course, that is BTW the main motivation why I completely refactor U++ Images now. HOWEVER, for certain applications, palette images are still needed. E.g. webserver could need to generate .gifs based on some RGBA data and this algorithm has to be part of it (and would limit the speed of webserver significantly; "brute force" algorithm spend 100ms on 2.4Ghz AMD64 machine, that simply was unacceptable).

Also, believe it or not, another motivation is that at least two of our existing applications deal with images that simply do not fit into memory even when indexed - e.g. nice 256 color, 32768x32768 pixels - >1GB of data. If you want to deal with such beasts, you need to have some nice tools to do so (of course speed of inverse transformation does not matter here).

2) You have to generate such table for each used palette. And each time you are about to save RGBA image in indexed format, you need to generate such inverse transformation cube.

(And sure, there is always "because I can" factor

Mirek

Subject: Re: Inverse palette conversion algorithm...
Posted by mirek on Fri, 07 Apr 2006 11:34:08 GMT
View Forum Message <> Reply to Message

mr_ped wrote on Fri, 07 April 2006 07:23luzr wrote on Fri, 07 April 2006 11:16(to 64K colors here)

Just 32K in your code. (5bits for red, 6 for green, 4 for blue = 15 bits total)

Yes, correct (I was experimenting with various sizes for a while).

Mirek

Subject: Re: Inverse palette conversion algorithm...
Posted by mr_ped on Fri, 07 Apr 2006 15:30:47 GMT
View Forum Message <> Reply to Message

correction from my previous post
SAR vs SHR conflict, SAL vs SHL is no problem, as both instructions give the same result even with negative numbers.

Now ... as I really enjoyed the task you were trying to accomplish, and I love graphics effects and speed optimalisations, I choosed to spend this lovely day polishing my skills and to give you another code to think about...
(well, just half of the day spend and finally I did use something from U++ (containers) .. so far I always checked TheIDE only on some simple "stdio.h" console application, so now I tried to do something more complex with it)

Beat this :

enum {

```
 RASTER_MAP_R = 32,
 RASTER_SHIFT_R = 3,
 RASTER_MAP_G = 64,
 RASTER_SHIFT_G = 2,
 RASTER_MAP_B = 16,
 RASTER_SHIFT_B = 4,

 RASTER_MAP_MAX = 64,
 RASTER_MAP_MAX_SHIFT = 2,

 RASTER_MAP_R_ADD = (1<<RASTER_SHIFT_R),
 RASTER_MAP_G_ADD = (1<<RASTER_SHIFT_G),
 RASTER_MAP_B_ADD = (1<<RASTER_SHIFT_B),
 RASTER_MAP_MAX_DIST = 3*((RASTER_MAP_MAX-1)*(RASTER_MAP_MAX-1))
};

struct PaletteCv {
 Buffer<byte> cv;
 static inline int GetIndex(const RGBA &c) { return (int(c.r >> RASTER_SHIFT_R) << 10) +
                                    (int(c.g >> RASTER_SHIFT_G)) +
                                    (int(c.b >> RASTER_SHIFT_B) << 6); }
 byte  Get(const RGBA& c) const   { return cv[GetIndex(c)]; }
 PaletteCv()                      { cv.Alloc(RASTER_MAP_R * RASTER_MAP_G * RASTER_MAP_B); }
};

struct sCubePoint : Moveable<sCubePoint> {
 RGBA mycol;
 byte index;
};

struct sPalCv2 {
 PaletteCv&  cv_pal;
 const RGBA *palette;
 int         ncolors;

 void AddPoint(byte r, byte g, byte b, byte idx, Vector<sCubePoint> *feed);

 sPalCv2(const RGBA *palette, int ncolors, PaletteCv& cv_pal);
};

void sPalCv2::AddPoint(byte r, byte g, byte b, byte idx, Vector<sCubePoint> *feed)
{
 sCubePoint pt;
 int dr = (int(palette[idx].r)-int(r))>>RASTER_MAP_MAX_SHIFT;
 int dg = (int(palette[idx].g)-int(g))>>RASTER_MAP_MAX_SHIFT;
 int db = (int(palette[idx].b)-int(b))>>RASTER_MAP_MAX_SHIFT;
 int dist = dr*dr + dg*dg + db*db;
 pt.mycol.r = r;
```

```
 pt.mycol.g = g;
 pt.mycol.b = b;
 pt.index = idx;
 ASSERT(dist <= RASTER_MAP_MAX_DIST);
 feed[dist].Add(pt);
}

sPalCv2::sPalCv2(const RGBA *palette, int ncolors, PaletteCv& cv_pal)
: cv_pal(cv_pal), ncolors(ncolors), palette(palette)
{
 int ii, jj;
 sCubePoint cubpt;
 Vector<sCubePoint> feed_me[RASTER_MAP_MAX_DIST+1];
 byte filled[RASTER_MAP_R * RASTER_MAP_G * RASTER_MAP_B];
 ZeroArray(filled);

 ii = ncolors;
 while (ii--) {
  cubpt.index = ii;
  cubpt.mycol = palette[ii];
  feed_me[0].Add(cubpt);
 }
 for (ii = 0; ii <= RASTER_MAP_MAX_DIST; ++ii) {
  while ( !feed_me[ii].IsEmpty() ) {
   cubpt = feed_me[ii].Pop();
   jj = cv_pal.GetIndex(cubpt.mycol);
   if (filled[jj] != 0) continue;
   filled[jj] = 1;
   cv_pal.cv[jj] = cubpt.index;
   if ( int(cubpt.mycol.r)+RASTER_MAP_R_ADD <= 255 )
    AddPoint(cubpt.mycol.r+RASTER_MAP_R_ADD, cubpt.mycol.g, cubpt.mycol.b, cubpt.index,
feed_me);
   if ( int(cubpt.mycol.r)-RASTER_MAP_R_ADD >= 0 )
    AddPoint(cubpt.mycol.r-RASTER_MAP_R_ADD, cubpt.mycol.g, cubpt.mycol.b, cubpt.index,
feed_me);
   if ( int(cubpt.mycol.g)+RASTER_MAP_G_ADD <= 255 )
    AddPoint(cubpt.mycol.r, cubpt.mycol.g+RASTER_MAP_G_ADD, cubpt.mycol.b, cubpt.index,
feed_me);
   if ( int(cubpt.mycol.g)-RASTER_MAP_G_ADD >= 0 )
    AddPoint(cubpt.mycol.r, cubpt.mycol.g-RASTER_MAP_G_ADD, cubpt.mycol.b, cubpt.index,
feed_me);
   if ( int(cubpt.mycol.b)+RASTER_MAP_B_ADD <= 255 )
    AddPoint(cubpt.mycol.r, cubpt.mycol.g, cubpt.mycol.b+RASTER_MAP_B_ADD, cubpt.index,
feed_me);
   if ( int(cubpt.mycol.b)-RASTER_MAP_B_ADD >= 0 )
    AddPoint(cubpt.mycol.r, cubpt.mycol.g, cubpt.mycol.b-RASTER_MAP_B_ADD, cubpt.index,
feed_me);
  }
```

```
 }
}
```

My simple benchmark shows it's almost 2 times faster.
It's probably a bit more memory hungry than your original code, but I don't think it's too greedy. (I didn't benchmark that, I didn't do any serious profiling under Win32 for long time, so I'm not sure what tools to use for that)

It does probably not give absolutely same result, but only marginaly different. (can you test it? I did some quick test to see if it looks functional, but didn't have any real-life app to see check)

Also it does confuse Assist a bit.
(try "feed_me[23]." ... and there we go, Assist has no idea it's Vector, instead it does thing it's sCubePoint)

I'm running on self compiled (MS VCC toolkit) upp-src-604-dev1.zip

And debugger is not really helpul with his inability to show feed[ii] data. (he keeps displaying feed[0] no matter what ii contains)

(At least it forced me to think harder why it does crash, and took me just 2min to figure out my mistake)

Subject: Re: Inverse palette conversion algorithm...
Posted by mr_ped on Fri, 07 Apr 2006 15:46:21 GMT
View Forum Message <> Reply to Message

The idea behind my code is simple FIFO flood fill.
You put the palette colors into the color space (cube), and start parallel flood fills from each point, until whole space is filled.

The only problem is that you can't use FIFO approarch directly, as the (+3,+3,+0) point's "distance" in FIFO terms would be 6 (6 steps needed to reach it), while the space distance is sqrt(3*3+3*3+0*0).

So the "FIFO" has been improved by distance sorting.
("radix" sorted queue "Vector<sCubePoint> feed_me[RASTER_MAP_MAX_DIST+1];")

This is quite an overhead in terms of memory usage, and also I can just hope the Vector constructor/destructor is fast enough to keep this fast (probably this is the bottleneck of the code, which can be optimalised by someone who knows containers better).

But from the syntetic benchmark it's already almost 2 times faster than your original piece of code.

Subject: Re: Inverse palette conversion algorithm...
Posted by mirek on Sat, 08 Apr 2006 12:00:57 GMT
View Forum Message <> Reply to Message

What a lovely idea! Even if it is true that my original approach is using a sort of similar technique - distances are radix sorted for each G-line.

However, I have not reproduced your results. I have tried it with the real photo and your algorithm spend 14ms, while mine was done in 5ms.

It is quite possible that it has something to do with babysitting my kids - that means I am on notebook and that is Sempron with 128K level 2 cache - based on task manager, memory costs seem to be somewhere around 512KB (quite believable, you have 140KB in feed_me without any actual data stored....).

In any case, I am posting whole nImage package to play with... (with both PalCv algos present - BTW, there is also another "fast" variant that is supposed to be used when inverse palette cube is created with known histogram for the purpose of saving the image without dithering - in that case it is possible to omit colors not present in original image - zero histogram entries).

Mirek

File Attachments

1) nImage.zip, downloaded 2248 times

Subject: Re: Inverse palette conversion algorithm...
Posted by mr_ped on Mon, 10 Apr 2006 15:58:29 GMT
View Forum Message <> Reply to Message

I did some benchmark on "old" Athlon 1GHz, and your old code is almost 3 times faster than my "supposed to be faster" code!
(original benchmark with my code being 2 times faster was from Celeron 2GHz)

That's ridiculous, I think that can't be the data cache trashing anymore, but I didn't have time to work on it more and figure out what's happening.
But having such simple code so significantly faster/slower on different CPU did really surprise, I was expecting like 50% speed differences between AMD vs Intel and different cache sizes.

Anyway, I did have time to *think* about it, and I can assure you the code I posted is just a mere "proof of concept". I didn't had time to try all those optimisations yet if they are correct - but if they are, expect some surprises and some more coding tricks. (and IMHO really nice ones)

Especially the above mentioned bottleneck of large Vector[] table will be removed to the extend I didn't even dream it was possible.
I think I will be back with improved version in 2-3 days, when I will have some time to sit behind my home PC and try all those ideas.

So far I have a little programmer's quiz here:
How will I make the "feed_me" array about 100 times smaller without any really "dirty" trick or major algorithm change?
(the code will change considerably, but the basic idea of the whole code will remain same)

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 2007 times

Subject: Re: Inverse palette conversion algorithm...
Posted by mr_ped on Sun, 21 May 2006 13:46:10 GMT
View Forum Message <> Reply to Message

By the way, my approach scales with the color-cube space, and the number of colors used in palette doesn't matter too much.
So if you make the cube even smaller than 32k colors, it would be quite faster.

Your original code strongly depends on number of colors, so with smaller than 256 color palettes it starts to be lot more faster.

Maybe with some 512 or 1024 palettes my algorithm would start to be faster, but that's probably not interesting in real life.

Subject: Re: Inverse palette conversion algorithm...
Posted by mirek on Sun, 21 May 2006 14:01:53 GMT
View Forum Message <> Reply to Message

mr_ped wrote on Sun, 21 May 2006 09:46By the way, my approach scales with the color-cube space, and the number of colors used in palette doesn't matter too much.
So if you make the cube even smaller than 32k colors, it would be quite faster.

Your original code strongly depends on number of colors, so with smaller than 256 color palettes it starts to be lot more faster.

Maybe with some 512 or 1024 palettes my algorithm would start to be faster, but that's probably not interesting in real life.

Ok, when I will meet some 10-bits palette device, I will know where to find the right code (here)

Thanks for your efforts, you approach is interesting.

Mirek