Subject: RGBA and ImageBuffer classes improve
Posted by tojocky on Thu, 05 May 2011 19:02:22 GMT
View Forum Message <> Reply to Message

Hello,

I propose to improve class RGBA like:

```
template<typename T>
struct RGBA_T : Moveable<RGBA_T<T> > {
	T b, g, r, a;
};
typedef RGBA_T<byte> RGBA;
```

And classe ImageBuffer Like:
in .h file

```
template<class T>
class ImageBuffer_T : NoCopy {
 mutable int  kind;
 Size         size;
 Buffer<RGBA_T<T> > pixels;
 Point        hotspot;
 Point        spot2;
 Size         dots;

 void         Set(Image& img);
 void         DeepCopy(const ImageBuffer_T<byte>& img);

 RGBA_T<T>*   Line(int i) const;
 RGBA*        Line8(int i) const;
 friend void  DropPixels___(ImageBuffer_T<T>& b) { b.pixels.Clear(); }

 friend class Image;

public:
 void  SetKind(int k)              { kind = k; }
 int   GetKind() const             { return kind; }
 int   ScanKind() const;
 int   GetScanKind() const         { return kind == IMAGE_UNKNOWN ? ScanKind() : kind; }

 void  SetHotSpot(Point p)         { hotspot = p; }
 Point GetHotSpot() const          { return hotspot; }

 void  Set2ndSpot(Point p)         { spot2 = p; }
 Point Get2ndSpot() const          { return spot2; }
```

```
 void  SetHotSpots(const Image& src);

 void  SetDots(Size sz)            { dots = sz; }
 Size  GetDots() const             { return dots; }
 void  SetDPI(Size sz);
 Size  GetDPI();

 Size  GetSize() const             { return size; }
 int   GetWidth() const            { return size.cx; }
 int   GetHeight() const           { return size.cy; }
 int   GetLength() const           { return size.cx * size.cy; }

 RGBA *operator[](int i)           { return Line8(i); }
 const RGBA *operator[](int i) const { return Line8(i); }
 RGBA *operator~();
 operator RGBA*();
 operator RGBA_T<uint16>*();
 operator RGBA_T<uint32>*();
 const RGBA *operator~() const;
 operator const RGBA*() const;

 operator const RGBA_T<uint16>*() const;

 operator const RGBA_T<uint32>*() const;

 void  Create(int cx, int cy);
 void  Create(Size sz)             { Create(sz.cx, sz.cy); }
 bool  IsEmpty() const             { return (size.cx | size.cy) == 0; }
 void  Clear()                   { Create(0, 0); }

 void  operator=(Image& img);
 void  operator=(ImageBuffer_T<byte>& img);

 ImageBuffer_T()                   { Create(0, 0); }
 ImageBuffer_T(int cx, int cy)       { Create(cx, cy); }
 ImageBuffer_T(Size sz)            { Create(sz.cx, sz.cy); }
 ImageBuffer_T(Image& img);
 ImageBuffer_T(ImageBuffer_T<byte>& b);
// BW, defined in CtrlCore:
 ImageBuffer_T(ImageDraw& iw);
};

typedef ImageBuffer_T<byte> ImageBuffer;

template<class T>
void ImageBuffer_T<T>::SetHotSpots(const Image& src){
 SetHotSpot(src.GetHotSpot());
```

```
 Set2ndSpot(src.Get2ndSpot());
}

template<class T>
void ImageBuffer_T<T>::Create(int cx, int cy){
 ASSERT(cx >= 0 && cy >= 0);
 size.cx = cx;
 size.cy = cy;
 pixels.Alloc(GetLength());
#ifdef _DEBUG
 RGBA_T<T> *s = pixels;
 RGBA_T<T> *e = pixels + GetLength();
 byte  a = 0;
 while(s < e) {
  s->a = a;
  a = ~a;
  s->r = 255;
  s->g = s->b = 0;
  s++;
 }
#endif
 kind = IMAGE_UNKNOWN;
 spot2 = hotspot = Point(0, 0);
 dots = Size(0, 0);
}

template<class T>
void ImageBuffer_T<T>::DeepCopy(const ImageBuffer& img){
 Create(img.GetSize());
 SetHotSpot(img.GetHotSpot());
 Set2ndSpot(img.Get2ndSpot());
 SetDots(img.GetDots());
 memcpy(pixels, img.pixels, GetLength() * sizeof(RGBA));
}

template<class T>
void ImageBuffer_T<T>::Set(Image& img){
 if(img.data)
  if(img.data->refcount == 1) {
   size = img.GetSize();
   kind = IMAGE_UNKNOWN;
   hotspot = img.GetHotSpot();
   spot2 = img.Get2ndSpot();
   dots = img.GetDots();
   pixels = img.data->buffer.pixels;
   img.Clear();
  }
  else {
```

```
   DeepCopy(img.data->buffer);
   kind = IMAGE_UNKNOWN;
   img.Clear();
  }
 else
  Create(0, 0);
}

template<class T>
void ImageBuffer_T<T>::operator=(Image& img){
 Clear();
 Set(img);
}

template<class T>
void ImageBuffer_T<T>::operator=(ImageBuffer& img){
 Clear();
 Image m = img;
 Set(m);
}

template<class T>
ImageBuffer_T<T>::ImageBuffer_T(Image& img){
 Set(img);
}

template<class T>
ImageBuffer_T<T>::ImageBuffer_T(ImageBuffer& b){
 kind = b.kind;
 size = b.size;
 dots = b.dots;
 pixels = b.pixels;
 hotspot = b.hotspot;
 spot2 = b.spot2;
}

template<class T>
void ImageBuffer_T<T>::SetDPI(Size dpi){
 dots.cx = int(600.*size.cx/dpi.cx);
 dots.cy = int(600.*size.cy/dpi.cy);
}

template<class T>
Size ImageBuffer_T<T>::GetDPI(){
 return Size(dots.cx ? int(600.*size.cx/dots.cx) : 0, dots.cy ? int(600.*size.cy/dots.cy) : 0);
}
```

and .cpp

```
template<class T>
RGBA_T<T>* ImageBuffer_T<T>::Line(int i) const{
 ASSERT(i >= 0 && i < size.cy); return (RGBA_T<T> *)~pixels + i * size.cx;
}

template<>
RGBA* ImageBuffer_T<byte>::Line8(int i) const{
 ASSERT(i >= 0 && i < size.cy); return (RGBA *)~pixels + i * size.cx;
}

template<>
RGBA* ImageBuffer_T<byte>::operator~(){
 return pixels;
}

template<>
ImageBuffer_T<byte>::operator RGBA*(){
 return pixels;
}

template<>
ImageBuffer_T<uint16>::operator RGBA_T<uint16>*(){
 return pixels;
}

template<>
ImageBuffer_T<uint32>::operator RGBA_T<uint32>*(){
 return pixels;
}

template<>
const RGBA* ImageBuffer_T<byte>::operator~() const{
 return pixels;
}

template<>
ImageBuffer_T<byte>::operator const RGBA*() const{
 return pixels;
}

//uint16
template<class T>
ImageBuffer_T<T>::operator const RGBA_T<uint16>*() const{
 return pixels;
}
```

```
template<>
ImageBuffer_T<uint16>::operator const RGBA_T<uint16>*() const{
 return pixels;
}

//uint32
template<class T>
ImageBuffer_T<T>::operator const RGBA_T<uint32>*() const{
 return pixels;
}

template<>
ImageBuffer_T<uint32>::operator const RGBA_T<uint32>*() const{
 return pixels;
}

template<>
int ImageBuffer_T<byte>::ScanKind() const{
 bool a255 = false;
 bool a0 = false;
 const RGBA *s = pixels;
 const RGBA *e = s + GetLength();
 while(s < e) {
  if(s->a == 0)
   a0 = true;
  else
  if(s->a == 255)
   a255 = true;
  else
   return IMAGE_ALPHA;
  s++;
 }
 return a255 ? a0 ? IMAGE_MASK : IMAGE_OPAQUE : IMAGE_EMPTY;
}
```

The proposed realization is not finished yet, but the first worked step is done!

I use this classes to read image in 16/32 bits.
