GIL 2.0 to GIL 2.1 Changes

This document outlines the more significant changes in GIL introduced in version 2.1

Support for non-byte-aligned images:

Added support for non-byte-aligned images. Example of reading an image and writing it back transposed in RGB121 (4-bit per pixel) format:

```
bgr8_image_t img;
jpeg_read_image("test.jpg",img);

typedef bit_aligned_image3_type<1,2,1, rgb_layout_t>::type rgb121_image_t;
rgb121_image_t rgb121_img(img.height(),img.width());

copy_and_convert_pixels(const_view(img),transposed_view(view(rgb121_img)));

// Color convert to rgb8 upon save
// (i/o does not yet support non-byte-aligned)
jpeg_write_view("out.jpg",
                color_converted_view<rgb8_pixel_t>(const_view(rgb121_img)));
```

To support bit distance, we are using the same classes that were providing byte distance (`byte_addressible_step_iterator`, `byte_addressible_2d_locator`, etc.) except that now they operate on *memory units* instead of bytes. A memory unit can currently be either a byte or a bit. Thus all byte-related names are changed to memory-unit related:

FROM	TO
ByteAdvancableIteratorConcept	MemoryBasedIteratorConcept
byte_addressable_step_iterator	memory_based_step_iterator
byte_addressable_2d_locator	memory_based_2d_locator
byte_advance	memunit_advance
byte_advanced	memunit_advanced
byte_distance	memunit_distance
byte_step	memunit_step
	byte_to_memunit
Locator::row_bytes()	Locator::row_size()
Locator::pix_bytestep()	Locator::pixel_size()

Notice that there is a new metafunction required by the `MemoryBasedIteratorConcept`, `byte_to_memunit`, which specifies the number of bits per memory unit (either 1 or 8).

References and iterators over bit-aligned pixels are implemented using two new classes, `bit_aligned_pixel_reference` and `bit_aligned_pixel_iterator`. The memory unit of bit aligned pixel iterators is a bit, i.e. `byte_to_unit< bit_aligned_pixel_iterator<T> >::value == 8`.

The value_type of a bit-aligned image is a packed_pixel. (There is a strong analogy with the way interleaved and planar images are implemented, with `packed_pixel` corresponding to `pixel`, `bit_aligned_pixel_reference` corresponding to `planar_pixel_reference` and `bit_aligned_pixel_iterator` corresponding to `planar_pixel_iterator`)

Added new metafunctions:

1. For constructing a homogeneous pixel value from elements:

```
template <typename Channel, typename Layout>
struct pixel_value_type {
    typedef … type;
};
```

2. For constructing a homogeneous packed pixel from elements. A packed pixel is a pixel that is byte-aligned but whose channels may not be byte aligned:

```
template <typename BitField, typename ChannelBitSizeVector, typename Layout>
struct packed_pixel_type {
    typedef … type;
};
```

Where `ChannelBitSizeVector` is an MPL integral vector of bit sizes to all channels. Example:

```
typedef packed_pixel_type<uint16_t, mpl::vector3_c<unsigned,5,6,5>,
                                       rgb_layout_t>::type rgb565_pixel_t;
```

3. For constructing packed images. A packed image is an image whose value_type is a packed pixel:

```
template <typename BitField, typename ChannelBitSizeVector, typename Layout,
          typename Alloc=std::allocator<unsigned char> >
struct packed_image_type {
    typedef … type;
};
```

There are also helper metafunctions for constructing packed images of 1 through 5 channels by taking the channels directly. Example:

```
typedef packed_image3_type<uint16_t,7,7,2,bgr_layout_t>::type bgr772_image_t;
```

Metafunctions for constructing bit-aligned images. A bit-aligned image is an image whose pixels may not be byte-aligned (such as an RGB222 image):

```
template <typename ChannelBitSizeVector, typename Layout,
          typename Alloc=std::allocator<unsigned char> >
struct bit_aligned_image_type { typedef … type; };
```

There are also helper metafunctions for constructing packed images of 1 through 5 channels by taking the channels directly. Example:

```
typedef bit_aligned_image3_type<1,2,1, rgb_layout_t>::type rgb121_image_t;
```

Added support for getting the raw memory from image views:

It is now more convenient to get the raw pointer to the beginning of the memory associated with a homogeneous image view by using the following functions:

```
template <typename HomogeneousView>
typename detail::channel_pointer_type<HomogeneousView>::type
interleaved_view_get_raw_data(const HomogeneousView& view) {…}

template <typename HomogeneousView>
typename detail::channel_pointer_type<HomogeneousView>::type
planar_view_get_raw_data(const HomogeneousView& view, int plane_index) {…}
```

Example:

```
rgb8_image_t rgb8(100,100);
unsigned char* data=interleaved_view_get_raw_data(view(rgb8));
const unsigned char* cdata=interleaved_view_get_raw_data(const_view(rgb8));

rgb16s_planar_image_t rgb8(100,100);
short* second_plane=planar_view_get_raw_data(view(rgb8),1);
const short* csecond_plane=planar_view_get_raw_data(const_view(rgb8),1);
```

Other changes:

- Renamed `heterogeneous_packed_pixel` to `packed_pixel`.
- Fixed histogram regression tests
- Improved `channel_convert` (it is faster by switching to floating point math only if necessary). Also fixed a roundoff bug in the conversion.
- Simplified `packed_channel_reference` and `packed_dynamic_channel_reference` by removing the `BitField` parameter (it is now computed automatically).