

Advanced Programming Techniques in Computer Vision

Lecture 11

Image Libraries Adobe's Generic Image Library

July 19, 2007

Gyuri Dorko
Multimodal Interactive Systems

TU Darmstadt
dorko@mis.tu-darmstadt.de



Last few lectures

- Templates: syntax & rules (name resolution, lookup, etc)
- Function objects
 - STL binders & adaptors
 - Boost.lambda
- Traits
- Last lecture:
- Concept checking & concept covering
 - Example: Boost, gcc, ConceptC++
- Modern Design
 - Policy-based design
 - Design patterns example: Singleton
 - Typelists
- Compile time polymorphism (CRTP)
- In-class non-member functions (Barton-Nackman trick)

Advanced Programming Techniques in CV, Lecture 11

Gyuri Dorko - TU Darmstadt, 2007 2

This week

- Notes on const correctness
- Available Image Libraries
- Image Libraries & Generic Programming
 - An introduction to Adobe's GIL
- Future of C++
- Few notes on the exam

Advanced Programming Techniques in CV, Lecture 11

Gyuri Dorko - TU Darmstadt, 2007 3

Notes on const correctness

- The const modifiers have a *shallow* behavior on member pointers and references.
- A typical solution is to hide the member, and *constify* it for the user if the object is const. E.g.:

```
class Image{
    const float* getRow(unsigned int i) const
    { return imageData_ + rowsize_*i; }
    float* getRow(unsigned int i)
    { return imageData_ + rowsize_*i; }
private:
    unsigned int width_;
    unsigned int rowsize_;
    unsigned int height_;
    float* imageData_;
    bool subimg_;
};
```

Advanced Programming Techniques in CV, Lecture 11

Gyuri Dorko - TU Darmstadt, 2007 4

Notes on const correctness - cont'd

- Unfortunately, sometimes this simple solution does not work.
- consider our named constructor for subimage creation:

```
const Image Image::CreateSubImage(const Image& master,
    unsigned int xoffset, unsigned int yoffset,
    unsigned int xsize, unsigned int ysize){
    Image img;
    img.width_ = xsize;
    img.rowsize_ = master.rowsize_;
    img.height_ = ysize;
    img.imageData_ = master.getRow(yoffset)+xoffset;
    float*
    const float*
    img.subimg_ = true;
    return img;
}
```

Advanced Programming Techniques in CV, Lecture 11

Gyuri Dorko - TU Darmstadt, 2007 5

Notes on const correctness - cont'd

- One, yet a bit complicated, solution is to wrap member pointers into member classes.
- Another solution is to copy the non-const pointer to the new image, consequently allow the creation of non-const subimages from a const image. E.g.:

```
const Image Image::CreateSubImage(const Image& master,
    unsigned int xoffset, unsigned int yoffset,
    unsigned int xsize, unsigned int ysize){
    Image img;
    img.width_ = xsize;
    img.rowsize_ = master.rowsize_;
    img.height_ = ysize;
    img.imageData_ =
    master.imageData_ + master.rowsize_ * yoffset + xoffset;
    img.subimg_ = true;
    return img;
}
```

Advanced Programming Techniques in CV, Lecture 11

Gyuri Dorko - TU Darmstadt, 2007 6

Available Image Libraries

Image representation

- Color spaces:
 - red, green, and blue
 - red, green, blue, and alpha
 - cyan, magenta, yellow, and key (black)
- Order of channels (e.g. RGB vs. BGR)
- Depth of each channel (8, 16, 32, etc. bits, signed, unsigned)
- Row alignment (aligned to word boundaries, no alignment)
- Structure:
 - interleaved: RGBRGBRGB
 - non-interleaved (planar): RRRGGGBBB

Channel layout

Image representation

- Example: 3x6 image
 - channel layout: RGB
 - aligned
 - interleaved
- 2nd pixel on the 3rd row is marked blue
- Image libraries can usually
 - work only with a subset of these representation,
 - sometimes allow to work with different structures,
 - image structures are typically fixed at compile time,
 - many times provide access to raw data,
 - very hard to mix different libraries.
- Important concept: subimages, image views

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|-----------|
| R | G | B | R | G | B | R | G | B | alignment |
| R | G | B | R | G | B | R | G | B | alignment |
| R | G | B | R | G | B | R | G | B | alignment |
| R | G | B | R | G | B | R | G | B | alignment |
| R | G | B | R | G | B | R | G | B | alignment |

Available (C++) image libraries Insight Segmentation and Registration Toolkit

- ITK is in principle developed for medical imaging.
- built on VNL (part of VXL package).
- Very well documented.
- Lazy evaluation (with data processing pipelines) of filters.
- Many basic algorithms implemented.
- I/O supports.
- Visualization support (VTK, Qt support).
- Generic programming support, (e.g., image templated over pixel type, and image dimensions)
- Memory management by smart pointers.
- 1D navigation on images
- Hard to work with images of different color layout.
- <http://www.itk.org> (open source)

Available (C++) image libraries VXL

- Similar to ITK
- run-time image dimensions
- VXL consist of
 - VIL (image library): I/O, etc.
 - VNL (fast mathematics library): matrices, etc.
 - VGL (geometry): curves, objects, etc
 - VSL (streaming I/O), VBL (basic templates), VUL (utilities)
- weaknesses are similar to ITK
- reasonable set of algorithms are implemented
- pretty big, but very powerful, sometimes tricky to compile
- <http://vxl.sourceforge.net>

Available (C++) image libraries Cool Image

- Cimg is a simple "suits for all" image type templated on the pixel.
- Small, 1 header file (5 classes).
- Targeted people not specifically computer scientists, i.e., no need to understand template meta-programming, while they still can profit from a little genericity.
- Algorithms:
 - very basic transformations
 - drawing routines
- STL-like interface, but hard mix with real STL code.
- <http://cimg.sourceforge.net>

Available image libraries

OpenCV



- Intel development, now it is open source
- Many computer vision algorithms are implemented: object, recognition, tracking, camera calibration, 3D reconstruction.
- Compatible with IPP (Intel Performance Primitives)
- It is a C library
- No genericity at all. No iterators, no views.
- There is ROI (one per image) support.
- <http://sourceforge.net/projects/opencvlibrary/>

Available (C++) image libraries

Vision with Generic Algorithms (VIGRA)



- Generic Programming is used in many places. Image is templated on pixels, STL compatible iterator support, support for several image views.
- Support for 2D iterators.
- Basic image processing algorithms (STL-like implementation).
- Lacks of planar (non-interleaved) image support.
- <http://kogs-www.informatik.uni-hamburg.de/~koethe/vigra/>

Available (C++) image libraries

Olena



- Generic Programming
- Image dimension is a part of the image type (template).
 - makes the library more general (can work with more than just 2D or 3D images)
 - makes the genericity more complicated (code is harder to read).
- Image is templated on storage type
- Extension with new algorithms are done by extensive use of templates, but they are hidden by macros.

Available (C++) image libraries

Generic Image Library



- Good peer reviews
 - Part of Adobe Source Libraries
 - Part of Boost
- Primary aim is to provide a generic image representation.
- Templated on channel layout (with semantic match of channels)
- Support for planar and interleaved images.
- Extensive view support (subimage view, color convert view, channel view, etc)
- Views can be defined on existing raw data.
- 2D image iterators, locators
- No basic algorithms (yet).
- Very basic I/O support (jpeg, png, tiff).
- Support for run-time image structures (dynamic images).
- <http://opensource.adobe.com/gil/>

Generic Image Library - Tutorial

- <http://opensource.adobe.com/gil/>
- <http://opensource.adobe.com/gil/html/giltutorial.html>
- <http://opensource.adobe.com/gil/documentation.html>

GIL - Tutorial

Gradient computation - Direction X



- How to use GIL?
- Simple example (the "non-generic way"):

```
#include <boost/gil/gil_all.hpp>
using namespace boost::gil;

void x_gradient(const gray8c_view_t& src, const gray8s_view_t& dst){
    for(int y=0; y<src.height(); ++y)
        for(int x=1; x<src.width(); ++x)
            dst(x,y) = ( src(x+1,y) - src(x-1,y) ) / 2;
}

void ComputeXGradientGray8(int width, int height,
    const unsigned char* src_ptr, int src_rowbytes,
    signed char* dst_ptr, int dst_rowbytes){
    gray8c_view_t src = interleaved_view(width, height,
        (const gray8c_pixel_t*)src_ptr, src_rowbytes);
    gray8s_view_t dst = interleaved_view(width, height,
        (gray8s_pixel_t*)dst_ptr, dst_rowbytes);
    x_gradient(src, dst);
}
```


GIL - Tutorial Generalized gradient computation.

```
template <typename S_VI EW, typename D_VI EW>
void x_gradient(const S_VI EW& src, const D_VI EW& dst){
    gi_function_requires<ImageViewConcept<S_VI EW> >();
    gi_function_requires<MutableImageViewConcept<D_VI EW> >();
    gi_function_requires<ColorSpacesCompatibilityConcept<
        typename color_space_type<S_VI EW>::type,
        typename color_space_type<D_VI EW>::type > >();

    for(int y=0; y<src.height(); ++y){
        typename S_VI EW::x_iterator src_it = src.row_begin(y);
        typename D_VI EW::x_iterator dst_it = dst.row_begin(y);
        for(int x=1; x<src.width()-1; ++x){
            static_transform(src_it[x], src_it[x-1], dst_it[x], (1-2)/constant(2));
        }
    }
}
```

- `static_transform` is the channel equivalent of `std::transform`, and additionally
 - the loop on channels is unrolled (because number of channels known at compile time),
 - the channels are matched semantically.

GIL - Tutorial Example calls (instantiations)

```
void ComputeXGradientGray8(int width, int height,
    const unsigned char* src_pix, int src_rowbytes,
    signed char* dst_pix, int dst_rowbytes){
    gray8c_view_t src = interleaved_view(width, height,
        (const gray8c_pixel_t*)src_pix, src_rowbytes);
    gray8s_view_t dst = interleaved_view(width, height,
        (gray8s_pixel_t*)dst_pix, dst_rowbytes);
    x_gradient(src, dst);
}

void ComputeXGradientRGB8_BGR16(int width, int height,
    const unsigned char* src_pix, ptrdiff_t src_rowbytes,
    signed short* dst_pix, ptrdiff_t dst_rowbytes){
    rgb8c_view_t src = interleaved_view(width, height,
        (const rgb8c_pixel_t*)src_pix, src_rowbytes);
    bgr16s_view_t dst = interleaved_view(width, height,
        (bgr16s_pixel_t*)dst_pix, dst_rowbytes);
    x_gradient(src, dst);
}

void ComputeXGradientPlanarRGB16_RGB32(int width, int height,
    const unsigned short* src_r,
    const unsigned short* src_g,
    const unsigned short* src_b, ptrdiff_t src_rowbytes, signed int* dst_pix,
    ptrdiff_t dst_rowbytes){
    rgb16c_planar_view_t src = planar_rgb_view(width, height,
        src_r, src_g, src_b, src_rowbytes);
    rgb32s_view_t dst = interleaved_view(width, height,
        (rgb32s_pixel_t*)dst_pix, dst_rowbytes);
    x_gradient(src, dst);
}
```

GIL Image Views

- Constructing images on raw data (or visa-versa), e.g.:
 - `interleaved_view`
 - `planar_rgb_view`, `planar_rgba_view`, `planar_cmyk_view`
 - ...
- View transformations
 - `color_converted_view` (Color converted view of another view)
 - `flipped_up_down_view` (view of a view flipped up-to-down)
 - `flipped_left_right_view` (view of a view flipped left-to-right)
 - `transposed_view` (view of a view transposed)
 - `rotated90cw_view`, `rotated90ccw_view`, `rotated180_view` (rotated view of a view)
 - `subimage_view` (view of an axis-aligned rectangular area within a view)
 - `subsampling_view` (stepping over a number of channels in X and number of rows in Y)
 - `nth_channel_view` (single-channel, i.e. grayscale view of the N-th channel of a view)
 - `kth_channel_view` (same as `nth_channel_view`, but the channel index is a template argument)

GIL Example: Compute the Harris image

```
int main() {
    using namespace boost::gil;

    {
        gray8_image_t img; // create an image
        png_read_and_convert_image("test.png", img); // #include <boost/gil/extension/io/png_io.hpp>
        gray32f_image_t convolved(img, 0); // create another image, same size
        harris(const_view(img), 1.4, 1.0, 0.05, view(convolved)); // We implement this!

        gray8_image_t vconvolved(img, 0); // Create an image for visualization
        transform_to_srgb(const_view(convolved), view(vconvolved)); // Normalization

        png_write_view("out-convoluted.png", view(vconvolved)); // GIL I/O extension

        {
            // same, but compute the harris image on a colored image (channel by channel)
            rgb8_image_t img;
            png_read_and_convert_image("test.png", img);
            rgb32f_image_t convolved(img, 0);
            harris(const_view(img), 1.4, 1.0, 0.05, view(convolved)); // Notice: same syntax as above
            transform_to_srgb(const_view(convolved), view(vconvolved), 60.0);
            png_write_view("out-convoluted.png", nth_channel_view(view(vconvolved), 0));
            png_write_view("out-convoluted.png", nth_channel_view(view(vconvolved), 1));
            png_write_view("out-convoluted.png", nth_channel_view(view(vconvolved), 2));
        }
    }
}
```

GIL Example: Harris (cornerness) image

```
template <typename S_VI EW, typename D_VI EW>
void harris(const S_VI EW& src, double si_gma1, double si_gma2, double si_gma3, const D_VI EW& dst){
    using namespace boost::gil;

    const size_t kws = 7;
    kernel_1d<float> ksmooth(kws, kws/2);
    kernel_1d<float> kder1v(kws, kws/2);
    int kernel_Gauss1d(ksmooth, si_gma1);
    int kernel_Gauss1dDer1v1D(kder1v, si_gma2);

    typedef pixel<float, layout<row-major, channel<D_VI EW>::val ue>> comp_pixel_t;
    typedef image<comp_pixel_t, false> image_t;

    image_t tmp(src, 0);
    image_t dx2(src, 0);
    image_t dy2(src, 0);
    image_t dxy(src, 0);

    convolve_rows<typename image_t::val ue_type>(src, kder1v, view(tmp));
    convolve_cols<typename image_t::val ue_type>(const_view(tmp), ksmooth, view(dx2));
    convolve_rows<typename image_t::val ue_type>(src, kder1v, view(tmp));
    convolve_cols<typename image_t::val ue_type>(const_view(tmp), ksmooth, view(dy2));

    multiply(const_view(dx2), const_view(dy2), view(dxy));
    multiply(const_view(dx2), const_view(dx2), view(dx2));
    multiply(const_view(dy2), const_view(dy2), view(dy2));
}
```

GIL Example: Harris (cornerness) image

```
kernel_1d<float> k1nteg(5, 3);
int kernel_Gauss1d(k1nteg, si_gma1);

convolve_rows<typename image_t::val ue_type>(const_view(dx2), k1nteg, view(tmp));
convolve_cols<typename image_t::val ue_type>(const_view(tmp), k1nteg, view(dx2));
convolve_rows<typename image_t::val ue_type>(const_view(dy2), k1nteg, view(tmp));
convolve_cols<typename image_t::val ue_type>(const_view(tmp), k1nteg, view(dy2));
convolve_rows<typename image_t::val ue_type>(const_view(dx2), k1nteg, view(tmp));
convolve_cols<typename image_t::val ue_type>(const_view(tmp), k1nteg, view(dxy));

for (int y=0; y<dst.height(); ++y) {
    typename image_t::val ue_t::x_iterator har11 = view(dx2).row_begin(y);
    typename image_t::val ue_t::x_iterator har22 = view(dy2).row_begin(y);
    typename image_t::val ue_t::x_iterator har12 = view(dxy).row_begin(y);
    typename D_VI EW::x_iterator dst_it = dst.row_begin(y);

    for (int x=0; x<dst.width(); ++x) {
        for (int c=0; c<num_channels<D_VI EW>::val ue>> ++c) {
            float trace = har11[x][c]*har22[x][c];
            dst_it[x][c] = har11[x][c] * har22[x][c] - (har12[x][c] * har12[x][c])
                - alpha * trace*trace;
        }
    }
}
```


The future of C++ C++0x - TR2 - (C++09)



- Several pending proposals (TR2)
 - **Concept checking** (see earlier lecture)
 - Small language improvements, e.g.
 - `vector<vector<T>>` → `vector<vector<T>>`
 - **delegating constructors**
 - **extern templates**: suppress implicit template instantiation in a compilation unit. (→ improve compilation speed)
 - **constexpr** (solves the problem with dynamic initialization, see earlier lecture)
 - **template aliases**, basically templated typedefs
E.g., : `template<class T> using Vec = std::vector<T, MyAllocator<T>>;`
 - rvalue references for perfect forwarding: `void forward(T&& t)`
 - etc.
- According to ISO rules, C++09 should be finalized by the end of 2007. The new standard is very close...

Next steps



- Try to read the standard
- Go for newsgroups (`comp.lang.c++.moderated`)
- Look into libraries:
 - Boost
 - BGL (graph library)
 - MPL (for metaprograming)
 - GIL (for image processing)
 - etc...
 - The mentioned image libraries
 - Blitz, uBlas for matrix manipulation
 - etc...

Exam



- Date: 24.07.2006
- Time: <individually sent by email>
- Place: B107
- Content:
 - Explain source code
 - Might need to write a few lines of source code
 - Questions about the lecture content
- Structure:
 - 15-20 minutes preparation: you will have some questions + the some source code, and you will have time to think about it
 - 30 minutes: oral exam

Advanced Programming Techniques in Computer Vision
Summer Term, 2007

- That's all -