C++ wrappers for OpenVX-1.x C API

Core ideas:

- lightweight - minimal overhead vs standard C API
- automatic references counting
- exceptions instead of return codes
- object-oriented design
- (NYI) helpers for user-defined kernels & nodes
- C++ 11 friendly

Quick start sample

The following short sample gives basic knowledges on the wrappers usage:

```
#include "ivx.hpp"
#include "ivx_lib_debug.hpp" // ivx::debug::*

int main()
{
    vx_uint32 width = 640, height = 480;
    try
    {
        ivx::Context context = ivx::Context::create();
        ivx::Graph graph = ivx::Graph::create(context);
        ivx::Image
            gray = ivx::Image::create(context, width, height, VX_DF_IMAGE_U8),
            gb   = ivx::Image::createVirtual(graph),
            res  = ivx::Image::create(context, width, height, VX_DF_IMAGE_U8);

        context.loadKernels("openvx-debug");  // ivx::debug::*

        ivx::debug::fReadImage(context, inputPath, gray);

        ivx::Node::create(graph, VX_KERNEL_GAUSSIAN_3x3, gray, gb);
        ivx::Node::create(
            graph,
            VX_KERNEL_THRESHOLD,
            gb,
            ivx::Threshold::createBinary(context, VX_TYPE_UINT8, 50),
            res
        );

        graph.verify();
        graph.process();

        ivx::debug::fWriteImage(context, res, "ovx-res-cpp.pgm");
    }
    catch (const ivx::RuntimeError& e)
    {
        printf("ErrorRuntime: code = %d(%x), message = %s\n", e.status(), e.status(), e.what
        return e.status();
    }
    catch (const ivx::WrapperError& e)
    {
        printf("ErrorWrapper: message = %s\n", e.what());
        return -1;
    }
    catch(const std::exception& e)
    {
        printf("runtime_error: message = %s\n", e.what());
        return -1;
    }

    return 0;
}
```

C++ API overview

The wrappers have **header-only** implementation that simplifies their integration to projects. All the API is inside `ivx` namespace (E.g. `class ivx::Graph`).

While the C++ API is pretty much the same for underlying OpenVX version **1.0** and **1.1**, there are alternative code branches for some features implementation that are selected at **compile time** via `#ifdef` preprocessor directives. E.g. external ref-counting is implemented for 1.0 version and native OpenVX one is used (via `vxRetainReference()` and `vxReleaseXYZ()`) for version 1.1.

Also there are some **C++ 11** features are used (e.g. rvalue ref-s) when their availability is detected at ***compile time***.

C++ exceptions are used for errors indication instead of return codes. There are two types of exceptions are defined: `RuntimeError` is thrown when OpenVX C call returned unsuccessful result and `WrapperError` is thrown when a problem is occured in the wrappers code. Both exception calsses are derived from `std::exception` (actually from its inheritants).

The so called **OpenVX objects** (e.g. `vx_image`) are represented as C++ classes in wrappers. All these classes use automatic ref-counting that allows development of exception-safe code. All these classes have `create()` or `createXYZ()` `static` methods for instances creation. (E.g. `Image::create()`, `Image::createVirtual()` and `Image::createFromHandle()`) Most of the wrapped OpenVX functions are represented as methods of the corresponding

C++ classes, but in most cases they still accept C “object” types (e.g. `vx_image` or `vx_context`) that allows mixing of C and C++ OpenVX API use. E.g.:

```
class Image
{
    static Image create(vx_context context, vx_uint32 width, vx_uint32 height, vx_df_image f
    static Image createVirtual(vx_graph graph, vx_uint32 width = 0, vx_uint32 height = 0, vx
    // ...
}
```

All the classes instances can automatically be converted to the corresponding C “object” types.

For more details please refer to C++ wrappers reference manual or directly to their source code.