Picturenaut 3: Multithreaded Plugin-Environment for HDR Imaging

2009 Marc Mehl, Christian Bloch

Introduction

Picturenaut was developed to provide open access to HDR Imaging for everyone.

It's a freeware application that is here to fill the void left when HDRShop turned commercial. While HDR Imaging gains popularity, the market is becoming increasingly capitalized. We need a common ground, sufficiency with no price tag.

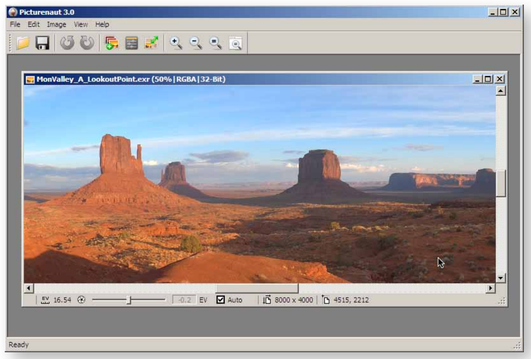
Program Architecture

Consequent multitasking and **extendibility with plugins** are the two main principles that make Picturenaut special. The program is built in layers, where the core program is just a hub for managing tasks and interfacing with the OS itself. All image-related operations, even seemingly built-in ones, are implemented via a plugin API.

The plugin SDK is **public and free**, making Picturenaut an ideal platform for rapid prototyping new algorithms. We believe this is the better alternative to making the entire program open-source. A plugin API allows us to keep the main code organized while still allowing interested coders to extend the program's functionality.

Core functions

Picturenaut has an **advanced HDR viewport** with zoom and pan capabilities. It performs realtime gamma and exposure correction, even with adaptive auto-exposure on the displayed region.

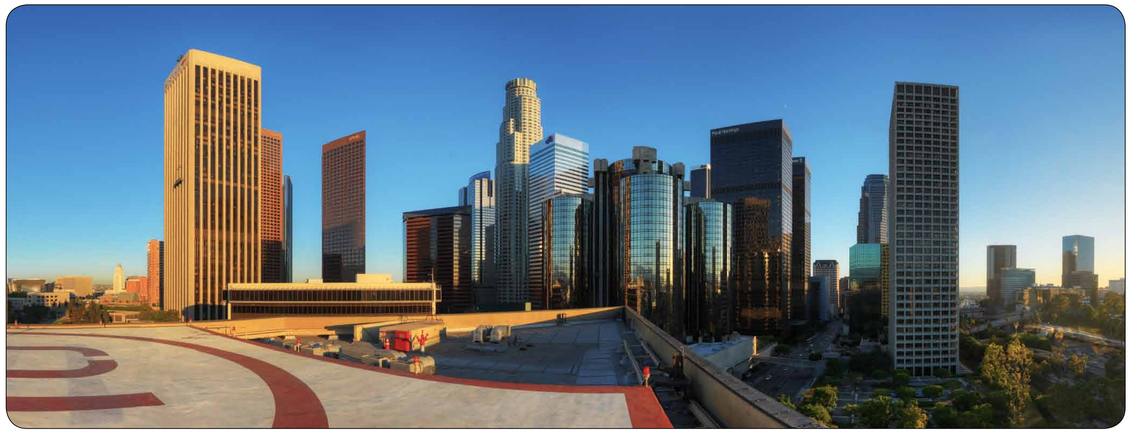


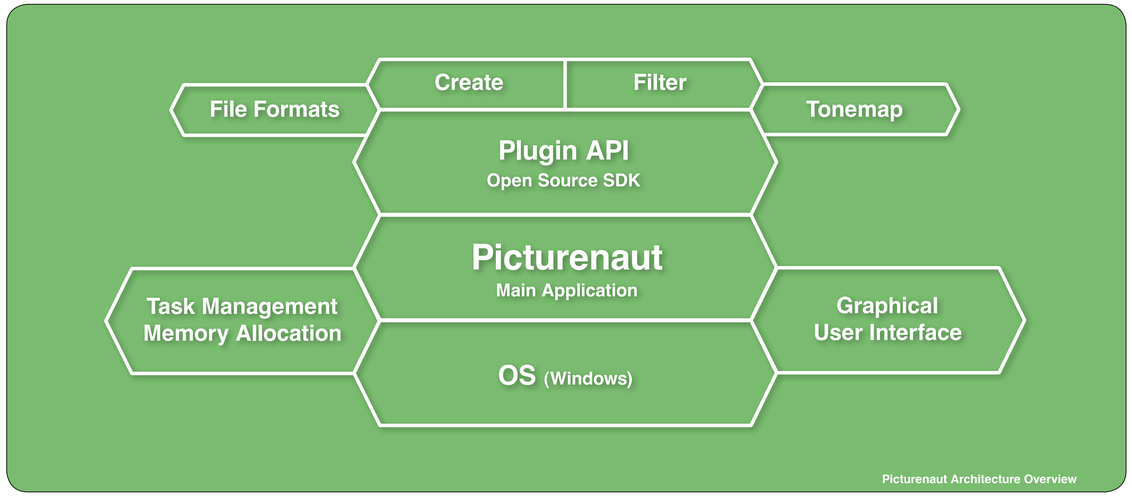
Picturenaut also auto-detects plugins and makes them available in the menu and to internal routines. But most importantly, the core program takes care of:

- Memory allocation
- Task management
- User interface
- Viewport updating
- Image buffer handling
- Undo and redo

Task management

Every operation is run in a **separate task** without blocking program execution. That even includes image loading and saving, which is unheard of in any other image editor. Operations that qualify for multitasking (like filters or tonemappers) are distributed across available CPUs. Every task can be cancelled by the user, and operations may specify chained tasks that are executed in succession.





Picturenaut Architecture Overview

Plugin Types

Plugins are written in C++ and compiled into DLL files. There are several types available, intended for different stages of the HDR imaging pipeline. The plugin type defines how Picturenaut will present this plugin to the user and internally to other plugins. Although all plugin types have access to all utility functions, some plugin types draw an extra benefit from beeing presented in the proper place.

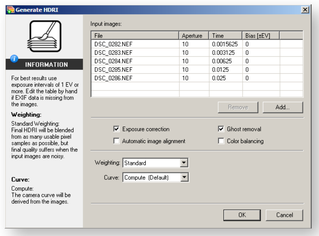
Format Plugins

Additional image file formats, both HDR and LDR, can be implemented as format plugins. Just like format plugins for Photoshop, this type provides funcions for loading and saving, and is automatically recognized in Picturenaut's file dialog. One example from the default installation is the RAW format plugin, that links in Dave Coffin's dcraw libraries.

Create Plugins

This type of plugin can create new 32-bit floating point image buffers of arbitrary size and number.

For example, Picturenaut's own HDR merging function is implemented as Create Plugin. Other candidates for potential Create Plugins could be a DSLR remote capturing interface or a physical sky generator. Such plugins would automatically appear in the "File" menu, and they may open their own interface window.

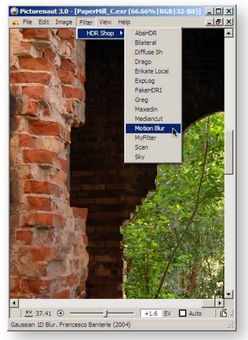


Filter Plugins

This is the most generic type of plugin, allowing the **highest level of freedom**. It has full read/write access to all image data and may create new image buffers, metadata, layers, or an entirely new image. Filter Plugins may open their own interface, optionally with realtime refresh of the image's main viewport.

One example, found in the standard installation, is a generic HDRShop emulator. It will detect existing HDRShop plugins, provide an interface for them, and execute them. While this is useful for legacy purpose, this bridge plugin uses only a fraction of Picturenaut's Plugin API capabilities.

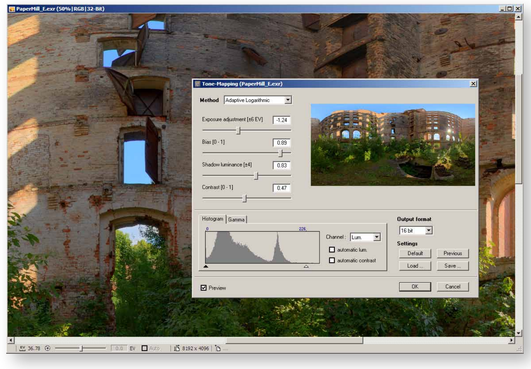
Possible Filter plugins yet to be written are a Panoramic Rewarp, Blur, Sharpen, even a paint engine or a layer manager.



Tonemapping Plugins

This is the **most convenient** plugin type. It will automatically integrate in Picturenaut's tonemapping dialog, completely equipped with realtime preview, interactive histogram, post-gamma correction and settings load/save functionality. Even the main viewport will stay responsive for panning and zooming, and with the Preview box enabled it will show the current TMO result applied to the full image. This allows a true WYSIWYG experience for the user.

All your plugin needs to do is provide the tonemapping algorithm and a parameter list (which will turn into live feedback sliders automatically).



Technically, the Tonemapping plugin is a **sub-class of the Filter plugins**. You're essentially adding a new method Picturenaut's open-ended "Tonemapping Filter" plugin. Of course, you may also create your very own interface by implementing the TMO as real Filter plugin.

Example: Basic Exposure TMO

The simplicity of Picturenaut's Plugin API is best illustrated with a code example. The C++ source code below is all you need to create a **fully functional tonemapping operator**.

```
/**
 *
 *   Exposure Tone-Mapping Plugin Example
 *   ====================================
 *
 *   Copyright (c) 2005-2009 Marc Mehl
 *
 */

#if _MSC_VER >= 1200
#pragma warning (disable : 4786)
#endif

#include "picturenaut/pnapp.h"
#include "mexptmo.h"
#include "resource.h"
#include <math.h>

/**
 * Creates an exposure tone-mapper.
 */
MExpTmo::MExpTmo() {
  paramList.pbNtries = &exposureEntry;
  paramList.pListName = pn_text("exposure");

  exposureEntry.paramType = pn::tmoparamSLIDER;
  exposureEntry.value._double = -2.64;
  exposureEntry.pInitValue = pn_text("-16.0|16.0");
  exposureEntry.pName = pn_text("exposure");
  exposureDisplayName = pn::PNApp::loadString(IDS_EXPOSURE);
  exposureEntry.pDisplayName = exposureDisplayName.c_str();
  exposureEntry.pNext = &ofSetEntry;

  ofSetEntry.paramType = pn::tmoparamSLIDER;
  ofSetEntry.value._double = 0.0;
  ofSetEntry.pInitValue = pn_text("-1.0|1.0");
  ofSetEntry.pName = pn_text("offset");
  ofSetDisplayName = pn::PNApp::loadString(IDS_OFFSET);
  ofSetEntry.pDisplayName = ofSetDisplayName.c_str();
  ofSetEntry.pNext = 0;
} // MExpTmo::MExpTmo

/**
 * Initializes this tone-mapper.
 *
 * @param inputBuffers list of input buffers
 * @return true on success; false otherwise
 */
bool
MExpTmo::init(const pn::PNImageBufferList &inputBuffers) {
  return true;
} // MExpTmo::init

/**
 * Returns a list of TMO parameters.
 *
 * @return a list of TMO parameters
 */
pn::PPNTMOPARAMLIST
MExpTmo::getParameterList() const {
  return (pn::PPNTMOPARAMLIST)&paramList;
} // MExpTmo::getParameterList
```

```
/**
 * Tonemaps an image.
 *
 * @param pDrawingInfo points to a drawing info
 */
void
MExpTmo::onTonemap(pn::PPNIMAGEDRAWINGINFO pDrawingInfo) {
  pn::PNIMAGEDESC *pImageIn = pDrawingInfo->image[0];
  pn::PNIMAGEDESC *pImageOut = pDrawingInfo->image[1];

  int width = pImageIn->size.cx;
  int height = pImageIn->size.cy;
  pn::PNBufferFormat &format = pImageIn->format;
  int numSamples = pn::PNImageBuffer::getNumberOfSamples(format);
  float *pSrcImage = (float *)pImageIn->pBuffer;
  float *pDstImage = (float *)pImageOut->pBuffer;
  double &exposure = this->exposureEntry.value._double;
  double &ofSet = this->ofSetEntry.value._double;

  applyExposureMultiplier(pSrcImage, pImageIn->pitch,
                          pDstImage, pImageOut->pitch,
                          width, height, numSamples,
                          (float)exposure, (float)ofSet);
} // MExpTmo::onTonemap

/**
 * Applies an exposure multiplier on a RGB image.
 *
 * @param pSrcImage source image data pointer
 * @param srcPitch a source pitch
 * @param pDstImage destination image data pointer
 * @param dstPitch a destination pitch
 * @param width a width
 * @param height a height
 * @param numSamples number of samples
 * @param exposure exposure multiplier
 * @param ofSet value ofSet
 */
void
MExpTmo::applyExposureMultiplier(float *pSrcImage, int srcPitch,
                                 float *pDstImage, int dstPitch,
                                 int width, int height,
                                 int numSamples,
                                 float exposure, float ofSet) {

  float *ps = pSrcImage;
  float *pd = pDstImage;

  srcPitch /= sizeof(float);
  dstPitch /= sizeof(float);
  exposure = (float)pow((double)2.0, (double)exposure);

  for (int j = 0; j < height; ++j) {
    for (int i = 0; i < width; ++i) {
      for (int c = 0; c < numSamples; ++c, ++ps, ++pd) {
        *pd = (*ps + ofSet) * exposure;
      } // for
    } // for
    ps += srcPitch;
    pd += dstPitch;
  } // for
} // MExpTmo::applyExposureMultiplier
```

Explanation and Notes

First, link in the required libraries from the plugin SDK.

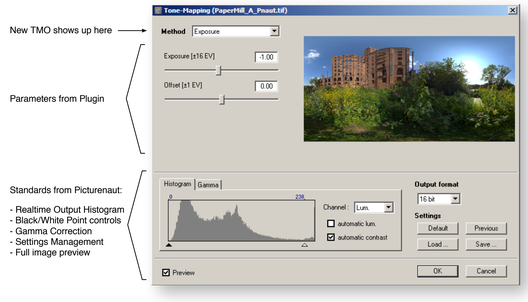
Then define the user parameters we needed and configure the slider defaults and ranges.

Next, the TMO needs to be initialized with Picturenaut's internal image buffers. Note that an image may internally contain more than RGB buffers, like additional channels and layers. You may also create temporary buffers for preprocessing.

The *onTonemap* procedure is where you start your TMO algorithm. Unless otherwise specified, the requested **image buffers are served in tiles**, enabling your algorithm to run multithreaded and on very big images. You can specify a tile margin to ensure large kernel operations are processed correctly. Picturenaut will automatically re-assemble tiles into the full-size image.

Result

Once the code above is compiled and saved in the picturenaut/plugins folder, this new TMO will be fully functional. It will simply show up in the drop-down menu of the tonemapping dialog.



Licensing and Availability

Picturenaut's Plugin SDK is released under the MIT license. That means you may do with it whatever you want (as long as we're not held responsible for bad things happening).

Although we appreciate and encourage open-source contributions, **this is not a requirement**. We welcome commercial plugins for Picturenaut, possibly adapted from existing Photoshop plugins. After all, we're providing a free alternative platform that will make your plugin more widely accessible.

More info and download at Picturenaut.com