Version 2.2 Release Notes

There are few changes and improvements in version 2.2. It was decided to release the new version due to some changes in the interfaces. They are not considerable, but still affect your code. From now on the names of the archives will be **agg22.zip** and and **agg22.tar.gz**, but inside the archives the directory name remains agg2.

- The scanline renderers are cleaned up. The former renderer_scanline_u renderer_scanline_u_solid, and renderer_scanline_p_solid were replaced with two ones: renderer_scanline_aa and renderer_scanline_aa_solid. Former classes were confusing, it wasn't clear, when to use "_u", and when to use "_p". Now they are renamed to "_aa" that refers to "Anti-Aliased". Still, there will be "_solid" version of the renderer because it works faster and requires less number of declarations (easier to use). However, the scanline containers, scanline_u and scanline_p remain. You just will be able to use both with the same renderer. I'd like to remind you that scanline_u refers to "unpacked", scanline_p to "packed". Unpacked means that the spans are stored "as is", that's if you have a span with "cover" values of 255 (that is, a solid one), it still keeps "len" number of bytes. Packed keeps only one byte and length. That is, in the packed version there's a simplest RLE compression is used. But for anti-aliased shapes the packed container generates about 3 times more spans (anti-aliased beginning, solid line, and anti-aliased end). So, for rendering small glyphs is better to use scanline_u, for large polygons scanline_p is more sutable. Besides, scanline_p takes less memory when being serialized through the scanline storage.
- Removed methods render() and render_ctrl() from rasterizer_scanline_aa and other classes. The new kind of interface is "licensed", it's called "**ScanlineSource**" and consists of two functions, rewind_scanlines(); and sweep_scanline();. The scanline source classes are:
  rasterizer_scanline_aa,
  scanline_storage_aa,
  serialized_scanlines_adaptor_aa,
  scanline_storage_bin, and
  serialized_scanlines_adaptor_bin.
- Removed file agg_color_rgba8_pre.h, that was a mistake to use different color type for plain and premultiplied color spaces. All necessary functionality of premultiplication is now in agg_color_rgba8.h. Using of plain and premultiplied colors is confusing. Below is a brief explanation. Format agg::pixfmt_rgba32 is the main and the fastest pixel format and it's supposed to be used in most cases. But it always uses plain colors as input and produces pre-multiplied result on the canvas. It has even less number of calculations than agg::pixfmt_rgba32_pre. Format agg::pixfmt_rgba32_plain is slow because of division operations. APIs allowing for alpha-blending require premultiplied colors. Besides, if you display RGBA with RGB API (that is, without alpha, like WinAPI BitBlt), the colors still must be premultiplied. Note that the formulas in agg::pixfmt_rgba32 and agg::pixfmt_rgb24 are exactly

the same! So, premultiplied colors are more natural and agg::pixfmt_rgba32_plain is rather useless.
Format agg::pixfmt_rgba32_pre is a bit slower than agg::pixfmt_rgba32 because of additional "cover" values, i.e. secondary alphas, that are to be mixed with the source premultiplied color. That spoils the beauty of the premultiplied colors idea. But the "cover" values are important because there can be other color spaces and color types that don't have any "alpha" at all, or the alpha is incompatible with integral types. So, the "cover" is a secondary, uniform alpha in range of 0…255, used specifically for anti-aliasing purposes.
One needs to consider this issue when transforming images. Actually, all RGBA images are supposed to be in the premultiplied color space and the result of filtering is also premultiplied. Since the resulting colors of the filtered images are the source for the renderers, one should use the premultiplied renderers, that is, agg::pixfmt_rgba32_pre, or the new one, agg::pixfmt_rgb24_pre. But it's important only if images are translucent, that is, have actual alpha channel.
For example, if you generate some pattern with **AGG** (premultiplied) and would like to use it for filling, you'll need to use agg::pixfmt_rgba32_pre. If you use agg::span_image_filter_rgb24_gamma_bilinear (that is, RGB for input) and draw it on the RGBA canvas, you still need to use agg::pixfmt_rgba32_pre as the destination canvas. The only thing you need is to premultiply the background color used out of bounds.

- Added files agg_render_scanlines.h and agg_pixfmt_rgb24_pre.h
- Please replace in your code:
  - `ras.render(sl,ren);` to **`agg::render_scanlines(ras,sl,ren);`**
  - `ras.render_ctrl(sl,ren,ctrl);` to **`agg::render_ctrl(ras,sl,ren,ctrl);`**
  - `agg::renderer_scanline_u` to **`agg::renderer_scanline_aa`**
  - `agg::renderer_scanline_u_solid` to **`agg::renderer_scanline_aa_solid`**
  - `agg::renderer_scanline_p_solid` to **`agg::renderer_scanline_aa_solid`**
- Added function `const char* full_file_name(const char* fname)` to agg::platform_support. It helps handle access to files in demo examples in some systems like BeOS.
- Added new functions and operators to agg::trans_affine:
  - `bool is_identity(double epsilon) const;`
  - `bool is_equal(const trans_affine& m, double epsilon) const;`
  - `double rotation() const;`
  - `void translation(double* dx, double* dy) const;`
  - `void scaling(double* sx, double* sy) const;`