Home/
News/

News Docs Download Mailing List CVS

Version 2.4 Release Notes

Include Files

Source Files

Renames

Span Generators and Scanline Renderers

Low Level (pixfmt) Renderers

Image and Pattern Transformers

Compound Shape Rasterizer

Path Storage

Custom Clippers in the Rasterizers

There are changes in the design, files, and interfaces are made, so that, you will have to spend some time for migratintg your applications. I tried to maintain is as painless as possible, but still some concepts have changed. There some files were removed from the package, but **not** functionality.

Include Files

- Removed Files
  - include/agg_render_scanlines.h
  - include/agg_span_generator.h
  - include/agg_span_image_resample.h
  - include/agg_span_image_resample_gray.h
  - include/agg_span_image_resample_rgb.h
  - include/agg_span_image_resample_rgba.h
  - include/agg_span_pattern.h
  - include/agg_span_pattern_filter_gray.h
  - include/agg_span_pattern_filter_rgb.h
  - include/agg_span_pattern_filter_rgba.h
  - include/agg_span_pattern_resample_gray.h
  - include/agg_span_pattern_resample_rgb.h

    - include/agg_span_pattern_resample_rgba.h

- New Files
    - include/agg_image_accessors.h
    - include/agg_path_length.h
    - include/agg_rasterizer_cells_aa.h
    - include/agg_rasterizer_compound_aa.h
    - include/agg_rasterizer_sl_clip.h
    - include/agg_span_pattern_gray.h

Source Files

- Removed Files
    - src/agg_path_storage.cpp
    - src/agg_rasterizer_scanline_aa.cpp

Renames

Old Name	New Name
rect	rect_i
pod_array	pod_vector
pod_deque	pod_bvector

Span Generators and Scanline Renderers

Removed files include/agg_render_scanlines.h and include/agg_span_generator.h. The major change is: In **AGG** v2.3 it was the responsibility of the span generator to allocate and provide an array of colors, which was conceptually wrong. That is, the function was:

```
color_type* generate(int x, int y, unsigned len);
```

Now it is:

```
void generate(color_type* span, int x, int y, unsigned len);
```

It means that the space for the color array is provided externally. This approach has more consistency and simplifies span generators and converters.

It also means that the span generators don't need to have span allocators inside, which simplified their creation. Span allocators are now passed to the scanline rendering functions and classes. That is, before (for example):

```
// AGG v2.3
typedef agg::span_allocator<color_type> span_alloc_type;
typedef agg::span_gouraud_rgba<color_type> span_gen_type;
typedef agg::renderer_scanline_aa<base_ren_type, span_gen_type> ren_type;

span_alloc_type span_alloc;
span_gen_type   span_gen(span_alloc);
ren_type        ren(ren_base, span_gen);
```

Now:

```
// AGG v2.4
typedef agg::span_allocator<color_type> span_alloc_type;
typedef agg::span_gouraud_rgba<color_type> span_gen_type;
typedef agg::renderer_scanline_aa<base_ren_type, span_alloc_type, span_gen_type> ren_type;

span_alloc_type span_alloc;
span_gen_type   span_gen;
ren_type        ren(ren_base, span_alloc, span_gen);
```

It may look more complex, but it isn't. Now you can simplify it and do without the scanline renderer class at all:

```
// AGG v2.4
typedef agg::span_allocator<color_type> span_alloc_type;
typedef agg::span_gouraud_rgba<color_type> span_gen_type;

span_alloc_type span_alloc;
span_gen_type   span_gen;
. . .
agg::render_scanlines_aa(ras, sl, ren_base, span_alloc, span_gen);
```

However, the renderer_scanline_aa class template is still useful, it's left for the sake of compatibility and for creating new function templates, like this:

```
template<class Ren>
void render_something(Ren& renderer)
{
   . . .
}
```

This function will work equally well with renderer_scanline_aa_solid and arbitrary renderer_scanline_aa.

Below is the summary of the scanline renderering function and classes in agg_renderer_scanline.h. The names refer to:

- **render_scanline_*** - a function that renders a single scanline.
- **render_scanlines_*** - a function that renders the content of a rasterizer or a scanline container, such as
  - rasterizer_scanline_aa,
  - scanline_storage_aa,
  - serialized_scanlines_adaptor_aa,
  - etc.
- **renderer_scanlines_*** - a class template with the scanline renderer interface, basically a

simple wrapper over the scanline rendering function.

```
// Render a single, anti-aliased, solid color scanline
//----------------------------------------------------------------------------
template<class Scanline, class BaseRenderer, class ColorT>
void render_scanline_aa_solid(const Scanline& sl,
                              BaseRenderer& ren,
                              const ColorT& color);



// Render all scanlines from Rasterizer, as solid color ones,
// with anti-aliasing
//----------------------------------------------------------------------------
template<class Rasterizer, class Scanline,
         class BaseRenderer, class ColorT>
void render_scanlines_aa_solid(Rasterizer& ras, Scanline& sl,
                               BaseRenderer& ren, const ColorT& color);



// Class template for solid color scanline rendering as it was before
//----------------------------------------------------------------------------
template<class BaseRenderer> class renderer_scanline_aa_solid
{
    . . .
};



// Render a single, color span scanline provided by SpanGenerator
//----------------------------------------------------------------------------
template<class Scanline, class BaseRenderer,
         class SpanAllocator, class SpanGenerator>
void render_scanline_aa(const Scanline& sl, BaseRenderer& ren,
                        SpanAllocator& alloc, SpanGenerator& span_gen);



// Render all scanlines from Rasterizer, as color span ones
//----------------------------------------------------------------------------
template<class Rasterizer, class Scanline, class BaseRenderer,
         class SpanAllocator, class SpanGenerator>
void render_scanlines_aa(Rasterizer& ras, Scanline& sl, BaseRenderer& ren,
                         SpanAllocator& alloc, SpanGenerator& span_gen)



// Class template for color span scanline rendering as it was before
//----------------------------------------------------------------------------
template<class BaseRenderer, class SpanAllocator, class SpanGenerator>
class renderer_scanline_aa
{
    . . .
}



// Render a single, aliased (binary) solid color scanline
//----------------------------------------------------------------------------
```

```
template<class Scanline, class BaseRenderer, class ColorT>
void render_scanline_bin_solid(const Scanline& sl,
                               BaseRenderer& ren,
                               const ColorT& color);



// Render all scanlines from Rasterizer, as solid color ones,
// without anti-aliasing (binary)
//--------------------------------------------------------------------
template<class Rasterizer, class Scanline,
         class BaseRenderer, class ColorT>
void render_scanlines_bin_solid(Rasterizer& ras, Scanline& sl,
                                BaseRenderer& ren, const ColorT& color);



// Class template for solid color scanline rendering as it was before
//--------------------------------------------------------------------
template<class BaseRenderer> class renderer_scanline_bin_solid
{
    . . .
};



// Render a single aliased (binary), color span scanline provided
// by SpanGenerator
//--------------------------------------------------------------------
template<class Scanline, class BaseRenderer,
         class SpanAllocator, class SpanGenerator>
void render_scanline_bin(const Scanline& sl, BaseRenderer& ren,
                         SpanAllocator& alloc, SpanGenerator& span_gen);



// Render all scanlines from Rasterizer, as color span ones,
// without anti-aliasing (binary)
//--------------------------------------------------------------------
template<class Rasterizer, class Scanline, class BaseRenderer,
         class SpanAllocator, class SpanGenerator>
void render_scanlines_bin(Rasterizer& ras, Scanline& sl, BaseRenderer& ren,
                          SpanAllocator& alloc, SpanGenerator& span_gen)



// Class template for color span scanline rendering as it was before
//--------------------------------------------------------------------
template<class BaseRenderer, class SpanAllocator, class SpanGenerator>
class renderer_scanline_bin
{
    . . .
}



// Render an abstract Rasterizer to an abstract Renderer. The Renderer
// can be of the following types:
// renderer_scanline_aa_solid
// renderer_scanline_aa
// renderer_scanline_bin_solid
```

```
    // renderer_scanline_bin
    //--------------------------------------------------------------------
    template<class Rasterizer, class Scanline, class Renderer>
    void render_scanlines(Rasterizer& ras, Scanline& sl, Renderer& ren);



    // A very simple function to render all paths with solid colors
    //--------------------------------------------------------------------
    template<class Rasterizer, class Scanline, class Renderer,
             class VertexSource, class ColorStorage, class PathId>
    void render_all_paths(Rasterizer& ras,
                          Scanline& sl,
                          Renderer& r,
                          VertexSource& vs,
                          const ColorStorage& as,
                          const PathId& path_id,
                          unsigned num_paths);



    // Render a compound shape from rasterizer_compound_aa
    //--------------------------------------------------------------------
    template<class Rasterizer,
             class ScanlineAA,
             class ScanlineBin,
             class BaseRenderer,
             class SpanAllocator,
             class StyleHandler>
    void render_scanlines_compound(Rasterizer& ras,
                                   ScanlineAA& sl_aa,
                                   ScanlineBin& sl_bin,
                                   BaseRenderer& ren,
                                   SpanAllocator& alloc,
                                   StyleHandler& sh);
```

Span clipping was removed from the scanline renderers, which simplified the design considerably and made it more flexible. It means that if the rasterizer produces scanlines from -10000 to 10000 and your window is only 800x600 it will work much slower than before, because the span generator will have to generate the whole 20000 pixels span. To prevent from this slowdown just use clipping in the rasterizer:

```
ras.clip_box(0, 0, 800, 600);
```

Low Level (pixfmt) Renderers

All low level renderers now can work with custom pixel accessors, instead of former hardcoded rendering_buffer. For the sake of compatibility there are pixfmt_rgba32 and all other are “typedefed”.

For example, you can use rendering_buffer_dynarow that allocates the pixel rows on demand as needed. You can also write your own pixel accessor provided the interface equivalent to rendering_buffer or rendering_buffer_dynarow.

Image and Pattern Transformers

The image and pattern transformers are now combined. For example, before it was declaration:

```
// AGG v2.3
template<class ColorT,
         class Order,
         class Interpolator,
         class WrapModeX,
         class WrapModeY,
         class Allocator = span_allocator<ColorT> >
class span_pattern_filter_rgba_bilinear :
public span_image_filter<ColorT, Interpolator, Allocator>
{
    . . .
};
```

Now it is:

```
// AGG v2.3
template<class Source, class Interpolator>
class span_image_filter_rgba_bilinear :
public span_image_filter<Source, Interpolator>
{
    . . .
};
```

The idea is to move the pixel wrapping functionality into a separate abstract image source. It allows us to get rid of a lot of almost duplicate code and extend the functionality. Template parameter Source can be one of the following types (agg_image_accessors.h):

- image_accessor_clip - image accessor with clipping, works as former span_image_filter_rgba_bilinear, etc.
- image_accessor_no_clip - no clipping, that is, it can be used only if you are really-really sure there will be no requests for pixels out of image bounds. If there are, there will be memory access violations. It's provided just in case if the performance is really very critical, but not recommended for general use because it's dangerous.
- image_accessor_clone - new functionality. If the requested pixel is out of bounds the boundary pixels are cloned. Very useful to reproduce functionality of some libraries like PDF.
- image_accessor_wrap - provides functionality of the former pattern transformers. Parameters WrapX, WrapY can be of the following types:
    - wrap_mode_repeat
    - wrap_mode_repeat_pow2
    - wrap_mode_repeat_auto_pow2
    - wrap_mode_reflect
    - wrap_mode_reflect_pow2
    - wrap_mode_reflect_auto_pow2

Below is an example if pattern transformer (texture-like) declarations and rendering:

```
// AGG v2.3
typedef agg::span_allocator<color_type> span_alloc_type;
span_alloc_type sa;
```

```
    agg::image_filter<agg::image_filter_hanning> filter;
    typedef agg::wrap_mode_reflect_auto_pow2 remainder_type;

    typedef agg::span_interpolator_linear<agg::trans_affine> interpolator_type;
    interpolator_type interpolator(mtx);

    typedef span_pattern_filter_2x2<color_type,
                                    component_order,
                                    interpolator_type,
                                    remainder_type,
                                    remainder_type> span_gen_type;
    typedef agg::renderer_scanline_aa<renderer_base_pre, span_gen_type> renderer_type;

    span_gen_type sg(sa,
                     rbuf_img(0),
                     interpolator,
                     filter);

    renderer_type ri(rb_pre, sg);
    agg::render_scanlines(g_rasterizer, g_scanline, ri);
```

```
    // AGG v2.4
    typedef agg::span_allocator<color_type> span_alloc_type;
    span_alloc_type sa;
    agg::image_filter<agg::image_filter_hanning> filter;

    typedef agg::wrap_mode_reflect_auto_pow2 remainder_type;
    typedef agg::image_accessor_wrap<pixfmt,
                                     remainder_type,
                                     remainder_type> img_source_type;

    pixfmt img_pixf(rbuf_img(0));
    img_source_type img_src(img_pixf);

    typedef agg::span_interpolator_linear<agg::trans_affine> interpolator_type;
    interpolator_type interpolator(mtx);

    typedef span_image_filter_2x2<img_source_type,
                                  interpolator_type> span_gen_type;
    span_gen_type sg(img_src, interpolator, filter);
    agg::render_scanlines_aa(g_rasterizer, g_scanline, rb_pre, sa, sg);
```

For the sake of performance there is span_image_filter_rgba_bilinear_clip is left and it works as former span_image_filter_rgba_bilinear. In all other cases the difference in performance is very little (in some cases the performance is even better).

Compound Shape Rasterizer

Added rasterizer_compound_aa and the rasterizers were considerably redesigned. Nopw you can render compound shapes such as Flash in one pass. An example is in flash_rasterizer.cpp. Below is a very brief explanation of how it works. If you are nor familiar with Flash data model you can skip this part.

The rasterizer_compound_aa works similar to the rasterizer_scanline_aa, but it takes edges with two styles: left and right. Then it poroduces a number of scanlines with their own styles and mixes them into a single colored scanline. The mixing is done in function render_scanlines_compound mentioned before.

```
    template<class Rasterizer,
             class ScanlineAA,
             class ScanlineBin,
             class BaseRenderer,
             class SpanAllocator,
             class StyleHandler>
    void render_scanlines_compound(Rasterizer& ras,
                                   ScanlineAA& sl_aa,
                                   ScanlineBin& sl_bin,
                                   BaseRenderer& ren,
                                   SpanAllocator& alloc,
                                   StyleHandler& sh);
```

The main thing here is `StyleHandler` that maps styles onto particular colors and/or span generators. It shall have the following interface:

```
bool is_solid(unsigned style) const;
const rgba8& color(unsigned style) const;
void generate_span(rgba8* span, int x, int y, unsigned len, unsigned style);
```

If `is_solid()` returns true then function `color()` will be called for this style, otherwise `generate_span()`.

A single compound shape can consist of solid areas, images, patterns, and/or gradients (at the same time!). It means that the StyleHandler should contain all possible span generators (such as span_gradient, span_image_filter_rgba_bilinear, etc) and call their `generate()` functions accordingly. Before rendering, there can be some preparational steps, such as setting the source images, transformation matrices, and so on.

Another example is gouraud_mesh.cpp that handles an array of span_gouraud_rgba objects.

Path Storage

The path_storage class is now redesigned and its functionality is split into the container and path making functions.

Now the declaration of the path_storage looks like follows:

```
template<class VertexContainer> class path_base { . . . };
```

Where `VertexContainer` is one of the following:

- vertex_block_storage - functionality as it was before,
- vertex_stl_storage - use STL compatible containers, such as `std::vector`, `std::deque`, or **AGG** ones: pod_vector, pod_bvector.

Examples of declarations:

```
typedef path_base<vertex_block_storage<double> > path_storage;
typedef path_base<vertex_stl_storage<pod_bvector<vertex_d> > > path_storage;
typedef path_base<vertex_stl_storage<std::vector<vertex_d> > > path_storage;
```

For compatibility, path_storage is declared as:

```
typedef path_base<vertex_block_storage<double> > path_storage;
```

Also, instead of former confusing function `add_path()` there are:

```
    template<class VertexSource>
    void concat_path(VertexSource& vs, unsigned path_id = 0);

    template<class VertexSource>
    void join_path(VertexSource& vs, unsigned path_id = 0)
```

The first one concatenates a path as is, with all `move_to` commands, the second one joins the path, replacing `move_to` to `line_to` commands, that is, as if the pen of a plotter was always down (drawing).

Custom Clippers in the Rasterizers

The rasterizers, rasterizer_scanline_aa and rasterizer_compound_aa now have custom clippers that can be:

- rasterizer_sl_no_clip
- rasterizer_sl_clip_int
- rasterizer_sl_clip_int_sat (clipping with saturation to avoid overflow)
- rasterizer_sl_clip_int_3x (for future use, LCD optimized rasterization)
- rasterizer_sl_clip_dbl
- rasterizer_sl_clip_dbl_3x (for future use, LCD optimized rasterization)