Home/
Release Notes v2.1
Release Notes v2.2
Release Notes v2.3
Release Notes v2.4

THE AGG PROJECT Anti-Grain Geometry

News Docs Download Mailing List SVN

News and History

- **July 8, 2007** New article Texts Rasterization Exposures.
  - In the article I tried to summarize my experience and observations concerning the situation with text rasterization in Windows and Linux. The article also contains demo applications to play with my method of RGB sub-pixel text rendering. I admit some statements may sound questionable, so, I appreciate any comments, criticism, and discussions. Please post your comments to the Mailing List.
  - I expect to renew active development of **AGG** in September, 2007.
- **October 23, 2006** Changing the license
  - Released version 2.5 under the GNU GPL license. Essentially it's the same as agg-2.4, so, no efforts to migrate are necessary. However, from now on all changes and bug fixes will be done in agg-2.5. I do not expect any massive changes in agg-2.5, only minor ones. Most probably Version 2.6 will have some differences, in particular, I'm going to make the use of floating point arithmetic more flexible.
  - Current **AGG** users who are willing to continue using **AGG** under the old terms and conditions are encouraged to contact me and I will consider their requests.
  - **AGG** v 2.4 remains available under the old terms and conditions.
- **June 19, 2006** Update.
  - Radial gradient functoin with focal point is rewritten. Now it works about two times faster than the old one. See gradient_radial_focus.
  - New utility class template is added, gradient_lut. It allows you to easily create a color LUT for gradients from a set of color stops as they are defined in SVG, section Gradients and Patterns.
  - New Demo gradient_focal.cpp. It's evolved from testing code and performance measurements. In particular, it shows you how to calculate the parameters of a radial gradient with a separate focal point, considering arbitrary affine transformations. In this example the window resizing transformations are taken into account. It also demonstrates the use case of gradient_lut and gamma correction.
  - Added functions suggested by **Bill Baxter**.
    - rect_base: `init()` and `hit_test()`;
    - gsv_text: `text_width()`;
    - trans_affine: `translate()`, `rotate()`, and `scale()`; They perform optimized direct transformations, without explicit matrix multiplications.
  - Finished experimets with rasterizer_compound_aa to simulate Constructive Area Geometry, see Demo rasterizer_compound.cpp.

Now function render_scanlines_compound works as before, that is, it assumes logically correct compound shapes and flat geometry, like Flash data streams provide. To flatten the geometry on demand use render_scanlines_compound_layered. It has the same set of argumets except for `ScanlineBin` that is not used there.
The reason to do so is that render_scanlines_compound works about 20% faster in average, and most of all, it does not perform any operations inside the holes in the shape.

- Added method `layer_order()` to rasterizer_compound_aa. It allows you to control the order of the layers and accepts values layer_unsorted, layer_direct, and layer_inverse. The default value is layer_direct, which means that greater styles have higher priority. Value layer_inverse inverts the layers. Value layer_unsorted does not guarantee any particular order and can be used with render_scanlines_compound to save a few CPU cycles.

- **June 17, 2006** Light update.
  - New class template row_accessor is added, thanks to **David Piepgrass**. Instead of keeping the pointers to each row it calculates the pixel addresses directly, performing one extra multiplication. It doesn't affect the performance on regular rendering operations, but still there's some slowdown on massive pixel operations, such as Bresenham line, blur, image transformations with nearest neighbor and bilinear filters. So, now you can choose between row_ptr_cache and row_accessor. There's a typedef in renderin_buffer.h that can be redefined by default in agg_config.h. This file contains some user definitions and supposed to be fully replaced by your own if necessary.
    The rendering_buffer is used only in short hand typedefs like pixfmt_rgba32. In your applications you can use both, row_ptr_cache and row_accessor, depending on your needs. The tip is: for the target rendering buffer you can freely use row_accessor, for stack_blur, recursive_blur and source images in image transformers it makes sense to use row_ptr_cache sometimes. Remember, you can attach as many row accessors to one physical buffer as you need and as many pixel formats as you need.
  - Added word “explicit” to all constructors that have one argument and store a pointer in the object. It prevents from implicit type casting via temporary objects that potentially results in undefined behavior. Thanks to **Vladimir Alemasov** for reporting the problem.

- **June 11, 2006** Update and new demo examples.
  - New Demo rasterizer_compound.cpp demonstrates a rather advanced technique of using the compound rasterizer. The idea is you assign styles to the polygons (left=style, right=-1) and rasterize this "multi-styled" compound shape as a whole. If the polygons in the shape overlap, the greater styles have higher priority. That is, the result is as if greater styles were painted last, but the geometry is flattened before rendering. It means there are no pixels will be painted twice. Then the style are associated with colors, gradients, images, etc. in a special style handler. It simulates Constructive Solid Geometry so that, you can, for example draw a translucent fill plus translucent stroke without the overlapped part of the fill being visible through the stroke.
  - Added new file agg_pixfmt_transposer.h. It provides a simple class template pixfmt_transposer that simply exchanges horizontal and vertical dimensions. It's mostly used in new blur filters.
  - Added new overloaded function `attach()` to all pixel level renderers. it has the following signature:
    ```
    template<class PixFmt>
    bool attach(PixFmt& pixf, int x1, int y1, int x2, int y2);
    ```
    That is, now you can attach `pixfmt_nnnn` to some existing one as a child. It still requires a separate rendering_buffer object, but will use the shared frame buffer. See example in Demo blur.cpp.
  - Added function `stride()` to all pixel level renderers.

- New Demo blur.cpp.
- Added new file agg_blur.h and fast blur functionality. There two algorithms are used: Stack Blur by **Mario Klingemann** and Fast Recursive Gaussian Filter, described here and here (PDF). The speed of both methods does not depend on the filter radius. Mario's method works 3-5 times faster; it doesn't produce exactly Gaussian response, but pretty fair for most practical purposes. The recursive filter uses floating point arithmetic and works slower. But it is true Gaussian filter, with theoretically infinite impulse response. The radius (actually 2*sigma value) can be fractional and the filter produces quite adequate result.
  There are two class templates: stack_blur and recursive_blur. stack_blur is parametrized by color type and calculator type. The calculator can be: stack_blur_calc_rgba, stack_blur_calc_rgb, and stack_blur_calc_gray. They are parametrized by the basic data type that is typically `unsigned` for rgba8 and `int64u` for rgba16.
  recursive_blur is also parametrized by the color type and calculator type. The calculator can be: recursive_blur_calc_rgba, recursive_blur_calc_rgb, and recursive_blur_calc_gray. They are parametrized by `float` or `double`. It's better to use `doubles` because the algorithm is very sensitive to precision.
  Also, there are optimized versions for `RGBA32`, `RGB24`, and `GRAY8` formats. They are: stack_blur_rgba32, stack_blur_rgb24, and stack_blur_gray8 respectively. These functions work only with 8 bits per component colors and with blur radius not exeeded 254.
  Many thanks to **Mario Klingemann** who generously permitted me to include his algorithm into **AGG**.

- **May 8, 2006 AGG** v2.4 is released! After a while of silence **AGG** is here again. That was the time of active development and you will have to make some changes in your code too. Hopefully it isn't that hard. See AGG Version 2.4 Release Notes for details. Below there's a brief description of the achievements.
  - Redesigned scanline renderers and span generators. See AGG Version 2.4 Release Notes for details.
  - Low level renderers (pixfmt) now have a possibility to use a custom class-accessor to pixel data.
  - Redesigned image and pattern filters and resamplers. Now they are considerably generalized and in fact, images and patterns are now combined into single classes. You can use custom pixel accessors for the images.
  - The **major breaktrough** is rendering of **Flash** compound shapes. Now **AGG** provides two ways of Flash rendering. One is pretty simple and in fact, was available before (well, almost), the other is more complex, but allows you to have absolutely flawless result. It's demonstrated in Demo flash_rasterizer.cpp. The simple method, that decomposes the shape into separate multiply connected polygons is in Demo flash_rasterizer2.cpp.
  - New demo examples summary
    - Demo flash_rasterizer.cpp - Flash rasterization based on compound shapes
    - Demo flash_rasterizer2.cpp - Flash rasterization based in shape decompositing
    - Demo gouraud_mesh.cpp - Seamless Gouraud triangle mesh
    - Demo line_patterns_clip.cpp - New line clipping considering image patterns
    - Demo compositing2.cpp - Another extended compositing modes demo
    - Demo gamma_tuner.cpp - Yet another monitor gamma tuner
  - Redesigned path_storage. Now it's a class template and it's functionality is separated. You can use any kind of data container, like std::vector, std::deque, etc. Also, there are special adaptors added, such as:
    poly_plain_adaptor
    poly_container_adaptor

poly_container_reverse_adaptor
line_adaptor

- Added custom clippers to rasterizer_scanline_aa and rasterizer_compound_aa.
- Considerably redesigned clippers for renderer_outline_image. Now it maintains correct pattern repetition even with clipping.
- All memory allocations and deallocations are now done via pod_allocator and obj_allocator. You can replace them if you define AGG_CUSTOM_ALLOCATOR and provide file agg_allocator.h (it's your responsibility to provide this file).
- From now on **AGG** is released under dual licensing, the original **AGG** one and the Modified BSD license. See The License for details.
- Some other improvements.
- **Milan Marusinec** AKA **Milano** ported the whole **AGG** to Object Pascal. It's a great job! Don't forget to visit: http://www.aggpas.org

- **September 15, 2005** Update
  - Added function `transform()` to font engines. Now you can set internal affine transformations. They work **before** the glyphs are cached, which means you can now use rotated text together with fast raster cache. It works only with glyph_ren_outline, glyph_ren_agg_mono, and glyph_ren_agg_gray8. It's impossible to use it with glyph_ren_native_mono and glyph_ren_native_gray8 because the font engine already receives bitmaps. Also, as the practice showed it's a bad idea to use embedded font engine transformations in both, FreeType and Win32, because it doesn't work well with hints in certain cases.

    Note that the internal transformer is included into the font signature to locate the cache pool. It means that it's a bad idea to use it often. If you need to draw 360 lines of text rotated with 1° step, the internal transformer will work very slow because it will create a new cache pool every time you change the angle. For this case it's much better to use vector cache (glyph_ren_outline) and transform the glyphs externally.

    The usecase of the internal transformer is basically to display a whole rotated page of text. Also note that if the angle differs from 0°, 90°, 180°, or 270°, there's some positioning inaccuracy appears when you use raster cache. It's because pre-rasterized glyphs cannot be positioned with subpixel accuracy.

    See modified examples, freetype_test.cpp and truetype_test.cpp.

- **September 8, 2005** Bugfix
  - Gouraud shader is modified again. First of all, the calls to `floor()` were replaced to just “manual” rounding. It's much faster on some compilers (Microsoft). Functions `floor()` and `ceil()` work terribly slow. Johan Paulsson has discovered this problem.
    Also, thre were some inconsiderable algorithmic modifications that resulted in more accurate rendering. Still, there're some artifacts possible if the triangles are too narrow (narrower than one pixel).
  - Fixed a bug in bezier_arc. It produced NaN or -NaN in cases of zero sweep angle, that resulted in infinite values in consecutive converters. Thanks to Richard Smolak for reporting it.
  - All anonymous `enums` now have names :-)
    It was some interference of questionable C++ constructs from **Boost** with unnamed enums in GNU C++ v4. Most probably it's a bug in GCC, but I decided to make it GCC compliant because it doesn't affect anyhow other compilers. Thanks to Artem Pavlenko for reporting it.

- After discussion with Marco Manfredini I still decided to keep the basic compositing formula the same, that is,
`alpha = (255 + alpha * cover) / 256`
Marco is right of course, he has provided more accurate formula with rounding:
`int q=alpha*cover+128;`
`return (q+(q«8))»8;`
But one extra shift operation is critical on Pentium-4 processors. The used formula isn't correct, but it gives appropriate result for the cost of only one extra addition.
Anyway, thanks Marco for the correct formula, I will keep it in mind for future.

- **August 27, 2005** Update
  - Considerably modified Gouraud shader. Now it works with **Subpixel Accuracy** and produces much more accurate results.
  http://antigrain.com/stuff/gouraud_artifacts.png
  Many thanks to Johan Paulsson for the contribution and the idea.
  - Modified the "overlay" and "hard-light" compositing operations. Now they are fully compliant with **SVG** anf **PDF**. Actually, the "soft-light" and "hard-light" operations produce almost opposite results, probably because of some mistake introduced by Adobe engineers. However, for the compatibility and historical reasons it works as described in **SVG**:
  http://www.w3.org/TR/2004/WD-SVG12-20041027/rendering.html
  - Modified class alpha_mask_u8. It was my old bug that I used a simple blending formula:
  `alpha = (alpha * cover) / 256`
  The correct one is:
  `alpha = (alpha * cover) /` **`255`**
  But it works much slower. To keep it fast and reasonably accurate I now use:
  `alpha = (255 + alpha * cover) / 256`
  Thanks to Antti Nivala for highlighting it.

- **August 21, 2005** Bugfix
  - Fixed a bug in path_storage:
  `arrange_orientations()` and `arrange_orientations_all_paths()`. It was rewritten and simplified. Thanks to Stephan Assmus for finding it.
  - Fixed the formula for "soft-light" color compositing. Thanks to Craig Northway for providing the info. Now "soft-light" works correctly, but still, there is an issue with "hard-light" and "overlay":
  http://www.w3.org/TR/2004/WD-SVG12-20041027/rendering.html.
  The "hard-light" formula produces "overlay" result and vice versa, but Craig claims that both formulae are correct. If they are, the demo pictures are incorrect.
  - Fixed bugs in agg_trans_double_path.cpp and agg_trans_single_path.cpp. Thanks to Konrad Kokosa.
  - Some other inconsiderable fixes.

- **July 12, 2005** Update
  - Updated atricle Adaptive Subdivision of Bezier Curves.
  - Improved the stroker algorithm (it will pursue me for the rest of my life, and I accept my fate). Now you have the following values for the inner join types:
  `inner_bevel`
  `inner_miter`
  `inner_jag`
  `inner_round`
  Only `inner_round` produces the flawless result in all cases, But it also produces a lot of

inner edges. I hope one day I can implement a filter that simulates the non-zero fill behaviour. You can see the difference between the methods in the bezier_div.cpp example.

- **Mauricio Piacentini** (http://www.tabuleiro.com) has made a great contribution. He added directory `examples/sdl` and created a single Makefile to build all the examples. This Makefile even attempts to download the supplementary files from Antigrain.com. That's very cool! Our applause to Mauricio! Now the directory linux_sdl is obsolete and will be removed in the next release. Here's what Mauricio writes.

```
To build all examples using SDL (Mac or Linux) just type:

cd /examples/sdl
make
Individual examples can be built with

make aa_test

In the same way the native Carbon examples can be built with

cd /examples/macosx_carbon
make

In both cases the static library will be built (if it was not
already) from the existing global Makefile in /src/.

The Makefiles for both SDL and Carbon will also attempt to
download the required .bmp files if they are not found in the
system for a given example. If the files could not be fetched
(wget) the user will receive a message explaining where to
download the samples from (sphere.bmp, etc.) Since all programs
reside in the same directory there is no need to duplicate the
.bmp files for each program that needs to use them.
```

- **Mauricio Piacentini** also added `examples/macosx_carbon` using the very same technique. So that, from now on you can compile **AGG** examples on MacOS X with GCC. I will do the same thing with X11 demo examples in the next release (not sure how soon). That's a great clean-up.

- **July 1, 2005** Update
  - Added new article, Adaptive Subdivision of Bezier Curves. That was a very challenging research, so I decided to describe all my “throes of composition”. If you don't read it I'll be upset. :) Please mail me your comments, suggestions, criticism.
  - As the result of the research I have removed curve4_div1 and curve4_div3 and replaced it with curve4_div that encapsulates all the ideas.
  - curve3 and curve4 are now not typedefs, but classes that include both, incremental and subdivision methods. So, you can choose the approximation method at run-time. See curve_approximation_method_e and conv_curve.
  - Class conv_curve now doesn't have functions curve3() and curve4(), instead there are

- Added new classes of Bezier curves approximation: curve3_div and curve4_div. The old ones renamed to curve3_inc and curve4_inc. For the sake of compatibility there are typedefs curve3 and curve4. New classes work pretty well with very long curves and the number of points is proportional to the logarithm of the curve length keeping the approximation error the same. It allows for handling very large arcs and ellipses, with radius of 10,000,000 or more.
- Added template parameters Curve3 and Curve4 to conv_curve. Also added methods to access the curve interpolators. So that, you can set parameters, such as `angle_tolerance()`. By default conv_curve now uses curve3_div and curve4_div.
- Added new demo example Demo bezier_div.cpp that demonstrates the new adaptive subdivision method.
- Improved stroke math. First, changed method of estimation of the angle step for round joins and caps. Now it produces much less number of points keeping arcs pretty accurate. The new formula is:
  `da = acos(width / (width + 0.125 / approximation_scale)) * 2;`
- Added new types of inner joins for strokes, see inner_join_e. By default there `inner_miter` is used in conv_stroke and conv_contour. `inner_jag` simulates the behaviour as if each line segment was covered by a rectangle placed along that segment and the "outer jags" were filled according to the line_join_e type. This is the most accurate inner join method that is consistent with the stroker in the Adobe 2D engine. But it produces a lot of false line segments, so that, there's another `inner_smart` type that works as `inner_jag` only near line caps. It can produce some defects on very thick lines, but it happens very rear. A perfect stroker is impossible or it will work terribly slow. Even Adobe stroker can produce defects. The following **SVG** path fails to draw correct stroke:
  `<path d="M 475, 157 C 200, 100, 453, 100, 222, 157" fill="none" stroke="green" stroke-width="100"/>`
  You can play with different types of joins in Demo bezier_div.cpp.

- **June 8, 2005** Update
  - Added function besj() to agg_math.h, so that we now don't depend on Bessel functions j0, j1, jn. The problem was with Borland C++. Many thanks to Andy Wilk.
  - Fixed a bug in conv_bspline that resulted in an infinite loop when the source contained only 2 points. Thanks to Sergey Yershov.
  - Added operations to trans_affine: "/", "/=", and functions multiply_inv() and premultiply_inv().
    Actually, there's no division operation on matrices, they just multiply the matrix by the inverse matrix. It's convenient when you need to remove temporarily some transformation from a sequence of affine calculations.
  - Added functions to trans_viewport:
    transform_scale_only(), inverse_transform_scale_only(), to_affine_scale_only(), device_dx(), and device_dy().

- **May 27, 2005** Bugfix
  - Fixed a bug in scanline renderers that caused occasional crash when using with scanline containers. Many thanks to John Hunter who located it. The following files were changed:
    agg_renderer_scanline.h
    agg_scanline_boolean_algebra.h
    agg_scanline_storage_aa.h

agg_scanline_storage_bin.h

- **May 26, 2005** Update
  - Redesigned rasterizer_scanline_aa in such a way that it can work with 24 bit screen coordinates. So that, you can now use **AGG** to render images wider than 32767 pixels. Of course, it requires dividing the whole buffer into horizontal bands. The performance of the rasterizer is now even better, not considerablaby, just about 3-5%. Also added classes scanline32_u, scanline32_p, and scanline32_bin that support 32 bit coordinates. They are fully compatible with the existing scanline containers but works slightly slower. Accordingly modified files agg_scanline_storage_bin.h and agg_scanline_storage_aa.h. This work has been sponsored by **Liberty Technology Systems, Inc.**, see AGG Sponsors and visit http://www.lib-sys.com.
  - Updated **General Polygon Clipper**. Thanks to Pierre Arnaud for the reminder.
  - Fixed two bugs in the stroker. Thanks to Mark Junker and Pierre Arnaud who discovered them.
  - Added `__BORLANDC__` to agg_basics.h. So that, hopefully **AGG** is now compatible with Borland C++ without changes. Many thanks to Jiri Krivanek for testing **AGG** in Borland environment.
  - Pierre Arnaud has added function create_dib_section() to class `pixel_map`, see agg_win32_bmp.h. It's not a part of **AGG** itself, it's a Win32 dependent class, but it also can be useful in some cases.
  - Added flag cw/ccw to ellipse. It's not a big deal, but can be still useful. It allows you to draw a ring using two ellipses with opposite directions. And it also works with the non-zero filling rule.
  - Fixed a bug in gradients. Thanks to Jens Boschulte for placing an order to map **AGG** to **PDF** gradients (axial and radial only).
  - Added adaptors with horrible names:
    comp_op_adaptor_clip_to_dst_rgba_pre,
    comp_adaptor_clip_to_dst_rgba,
    comp_adaptor_clip_to_dst_rgba_pre
    to agg_pixfmt_rgba.h. They perform compositing operations with preliminary multiplication of the source color to destination alpha. So that, if the desination alpha is 0 (transparent) it won't have any effect. Thanks to Michael Kleps (reFX) for ordering this work.

- **May 12, 2005** Bugfix.
  - Fixed a bug in the stroker. In very rear cases there are defects appeared. It was possible when there were very short line segments, less than 1e-12 length.
  - Fixed a bug in bezier_arc. In some cases it added degenerate curves that showed up the bug in the stroker. :-)
  - Fixed a bug in agg_font_win32_tt.cpp. It didn't update the signature when creating a new font. Many thanks to Michael D. Tajmajer.
  - Changed the default value of m_char_set from ANSI_CHARSET to DEFAULT_CHARSET.

- **May 5, 2005** AGG is updated.
  - Added extended compositing operations and claass to RGBA pixel formats. They correspond to the SVG 1.2 specification:
    http://www.w3.org/TR/2004/WD-SVG12-20041027/rendering.html#comp-op-prop
  - Added new demo example: Demo compositing.cpp that demonstrates all available modes and how to use them.

  - Removed vertex_iterators as they were never used and there is no reason to use them. If you really need to use the iterator semantics you can always incorporate Iterator Adaptors from BOOST by Dave Abrahams. The changes should not affect your code, but you might have to remove `#include "agg_vertex_iterator.h"` from your files as this line could be added to your code by copy-pasting from **AGG** examples.
  - Removed transformations from the font engines. The problem was with both, FreeType and Win32 font engines. They produced differentfont hinting for `flip_y=false` and `flip_y=true`. Flipping was implemented using the internal transformation mechanisms, but it resulted in some nontrivial problems with hinting. The embedded transformations are pretty much useless, especially with hinting. From now on if you want to draw transformed text you will have to use the vector cache and **AGG** transformers. In practice all **AGG** users did namely so.
  - Renamed "id" to "path_id" to make the code compliant with Objective-C. Many thanks to Sergey Yershov who successfully tested and used AGG on MacOS with Cocoa.

- **April 28, 2005** AGG is updated.
  - Removed renderer_scanline_aa_opaque and renderer_scanline_bin_opaque. As the practice shows they are useless, but require a lot of code. The only purpose of them was to ignore the alpha channel and consider it as 1.0 with slightly better performance. Instead, Added renderer_scanline_bin_copy that performs direct copying of color information to the buffer (no blending). Many thanks to Jiri Krivanek for the help with it.
  - Added functions to transform curves: catrom_to_bezier, ubspline_to_bezier, and hermite_to_bezier. Many thanks to Tony Juricic, his name is included into agg_curves.h
  - The scanline rasterizer was redesigned. Now it uses a faster soring algorithm (a combination of the Radix sort and Quick sort). The sorting part works about twice faster that is noticeable when rendering thin lines (10-15% overall improvement).
  - Improved the performance of alpha-blending. Function blend_from now works about 40% faster on RGBA buffers.

- **April 13, 2005**
  - Finished image and pattern transformations with resampling for all available pixel formats, RGB, RGBA, and GRAY.
    There are new files:
    agg_span_image_resample.h
    agg_span_image_resample_gray.h
    agg_span_image_resample_rgb.h
    agg_span_image_resample_rgba.h
    agg_span_pattern_resample_gray.h
    agg_span_pattern_resample_rgb.h
    agg_span_pattern_resample_rgba.h
    And new classes:
    span_image_resample_gray_affine
    span_image_resample_gray
    span_image_resample_rgb_affine
    span_image_resample_rgb
    span_image_resample_rgba_affine
    span_image_resample_rgba
    span_pattern_resample_gray_affine
    span_pattern_resample_gray
    span_pattern_resample_rgb_affine
    span_pattern_resample_rgb
    span_pattern_resample_rgba_affine
    span_pattern_resample_rgba.

- Demo image_resample.cpp is modified and added new Demo pattern_resample.cpp.
- The basic interger types can be redefined from the compiler command line or by generating file agg_config.h. By default this file is empty, and the 64-bit integers are defined as before, that is, `__int64` for MSVC and `long long` for all other compilers. In most cases it's fair enough. Also see comments in agg_config.h. From now on the 64-bit types are legalized in the official package. The testing version with “a” suffix has been removed. See Download.
- Added a new type of a XOR operation in scanline boolean algebra, it's sbool_xor_abs_diff. This operation calculates the **Anti-Aliasing** values as `abs(a-b)` which guarantees zero result when XORing two coinciding Anti-Aliased shapes. See pictures:
  http://antigrain.com/stuff/scanline_boolean_xor_linear.gif
  http://antigrain.com/stuff/scanline_boolean_xor_saddle.gif
  http://antigrain.com/stuff/scanline_boolean_xor_abs_diff.gif
- Fixed some typos on the site, thanks to Kirill Smelkov.
- The **AGG** project has a new sponsor! http://www.epsitec.ch See AGG Sponsors.

- **April 4, 2005** Breakthrough in image resampling!
  - Finally implemented images transformations with resampling to achieve best possible quality when creating thumbnails. Currently it's implemented only for RGB color space, but the rest of the job is simple and will be done soon.
    There are two major transformation methods:
    span_image_resample_rgb_affine
    span_image_resample_rgb
    The differerence is that span_image_resample_rgb_affine works with constant scale factors by X and Y, so, it's faster.
    span_image_resample_rgb allows you to have arbitrary local scale. It's suitable for perspective transformations or any other ones if you write an appropriate span interpolator. Currently there is only perspective interpolator available.
  - Added new span interpolator classes:
    - span_interpolator_linear_subdiv. It works as span_interpolator_linear, that is, it interpolates between the points along a straight line, but it resynchronizes the result at every 8th, 16th, etc pixels (defined by subdiv_shift). So that, it can be used together with trans_perspective. It produces some inaccuracy, but speeds up perspective image transformations almost twice!
    - span_subdiv_adaptor. Essentially it's the same as span_interpolator_linear_subdiv. The difference is in the interface and it supports a function to calculate local scale. This adaptor is used mostly with perspective transformations with resampling.
    - span_interpolator_persp_exact. Used together with span_subdiv_adaptor to transform images with resampling. Provides exact calculation of the coordinates (no interpolation) and the linear interpolation of the scale factors.
    - span_interpolator_persp_lerp. The same as the above, but uses linear interpolation for calculating of the coordinates. So, it's less accurate, but faster.
  - Two new demo examples: Demo image_filters2.cpp
    and Demo image_resample.cpp
  - Bugfix.
    - Fixed a bug in agg_font_cache_manager.h that resulted in Bus Error on Sun SPARC architecture. Many thanks to Marcus.Gruendler.
    - Fixed a bug in agg_renderer_outline_aa.h that hid line segments returning exactly along the path.
  - Introduced new basic data types, int64, int64u. They are defined as `__int64` for MSVC

and `long long` for all other compilers and platforms. However, I temporarily created separate packages, agg23a.zip and agg23a.tar.gz, see Download. If there are no compatibility problems I'll include it into the official package.

- **March 2, 2005** Some fixes.
  - Color types (rgba8, rgba16, gray8, gray16) are redesigned. I just returned to the old design where the color types were simple structures, not templates. There is some duplicate code, but it's more convenient in use. Now you can construct rgba16, gray8, and gray16 from rgba8. The AGG Version 2.3 Release Notes are updated.

- **February 28, 2005** AGG v2.3 is released!.
  See AGG Version 2.3 Release Notes for details. The old version is still available to make it easier for you to track changes.
  - Added support for high capacity colors, such as RGB48 and RGBA64. Many thanks to Liberty Technology Systems, Inc., who has sponsored this work. Visit http://lib-sys.com.
  - Added repeat/reflect wrap modes to pattern transformers.
  - Fixed some inconsiderable bugs.
  - Added functions blend_from to pixel formats as well as some utility functions such as for_each_pixel(), premultiply(), demultiply(), apply_gamma_dir(), apply_gamma_inv().
  - Added page AGG Sponsors.

- **November 14, 2004** Update.
  - Fixed some bugs and glitches in the code and files.
  - Added new demo example Demo pattern_perspective.cpp.

- **November 7, 2004** Update.
  - The stroke converter has been considerably redesigned and now it works more correctly. In general the task of stroke generating is very difficult and I'm afraid it cannot be solved with **O(N)** algorithm complexity. Although, the algorithm was numerically stable, it could produce some artifacts, like the following:
    http://antigrain.com/stuff/stroke_defect1.png
    http://antigrain.com/stuff/stroke_defect2.png
    Now conv_stroke generates more self-intersecting segments, but they all are completely eliminated when using the Non-Zero filling rule. From the point of view of the final result this behaviour is more correct.
    http://antigrain.com/stuff/stroke_correct1.png
    http://antigrain.com/stuff/stroke_correct2.png
    First, the result is now correct when there are "returning" segments (a segment that comes back exactly along the path). Second, There were added the `inner_line_join` and `inner_miter_limit` parameters to `conv_stroke` and `conv_contour`. They are used when calculating inner joins and by default `inner_miter_limit` has minimal value of **1+1/64**.
    By default it uses `miter_join_revert`, that is, it doesn't cut the miter at the `inner_miter_limit` value, but switches to the bevel join. For the outer joins there are still `miter_join` and `miter_join_revert`, as it was before. Inner and outer joins are considered as follows. If the next segment turns right, the join on the right is inner one, the join on the left is outer.
    http://www.antigrain.com/stuff/stroke_defect.zip (Win32 exe) - it's how it works now.
    The `conv_contour` also has been changed. Now it has the same join types as `conv_stroke` (before there was only miter join):
    http://www.antigrain.com/stuff/conv_contour.zip
    So, there's a new file, agg_math_stroke.h and since this code is now reused by the

stroker and contour converters, I had to move enums line_cap_e and line_join_e to the agg namespace (before it was in the vcgen_stroke class). In the updated version you will have to remove those `vcgen_stroke::` specifiers. `line_cap_e` and `line_join_e` are now defined in the agg_math_stroke.h and it's automatically included when you include agg_conv_stroke.h or agg_conv_contour.h.

- Added new demo example Demo line_patterns.cpp

- **October 26, 2004** Update.
  - **AGG** has been tested on AmigaOS. Many thanks to Steven Solie, http://www3.telus.net/public/ssolie
    When building on AmigaOS 4.0 or higher type the following for instructions on what targets are available.
    `make -f Makefile.AmigaOS`
    To just build and install AGG into the standard AmigaOS SDK ready for use type:
    `make -f Makefile.AmigaOS install`
    If you just want to build one demo (e.g. lion) use:
    `make -f Makefile.AmigaOS bin/lion`
  - Added binaries for AmigaOS to the Demo page

- **October 7, 2004** Update.
  - Changed agg_span_converter.h, added two arguments `(x,y)` to function `convert()`. It allows you for creating pipelines when generating spans.
  - Added agg_span_gradient_alpha.h that can be used to apply arbitrary alpha gradients (transparency). This is a **span converter** that can be used with any span generator, such as images, gradients, gouraud shading, patterns, etc.
  - Added new example Demo alpha_gradient.cpp
  - Added radial gradient with focal center, see gradient_radial_focus
  - Added gradient repeat and reflect adaptors, see gradient_repeat_adaptor and gradient_reflect_adaptor

- **September 18, 2004**
  - Update. Fixed some inconsiderable bugs.
  - Added page Users and Customers.
  - Finally solved the issue of “0.5 pixel” when transforming images. See http://article.gmane.org/gmane.comp.graphics.agg/193 for details.
  - Convenience functions were added to the trans_affine.
  - Added new demo: Demo aa_test.cpp.

- **September 12, 2004**
  - **AGG** v2.2 released! See AGG Version 2.2 Release Notes and Download.

- **August 27, 2004**
  - Added a possibility to the font engines to use a 32-bit vector cache. Before it was "hardcoded" 16-bit coordinates (10.6 format) which caused integer overflow when caching glyphs of large sizes (200 and more). Now you have a choice whether to use the compact 16-bit vector cache or 32-bit one. Changed names of the font engines. Now, instead of `font_engine_freetype` and `font_engine_win32_tt` there are:
    `font_engine_freetype_int16`, `font_engine_freetype_int32`,
    `font_engine_win32_tt_int16`, and `font_engine_win32_tt_int32`.

See Download.

- **August 25, 2004**
  - Fixed some bugs in agg_path_storage_integer.h. Added method `premultiply()` to trans_affine. Fixed a bug in SVG Viewer with the order of affine transformations.
  - Added two new non-linear transformers, trans_single_path and trans_double_path.
  - Added two new "kinda-c00l-stuff" demo to draw text along an arbitrary curve and between two curves. See Demo trans_curve1.cpp and Demo trans_curve2.cpp.

- **August 15, 2004**
  - **AGG** code is cleaned up and now it's ANSI C++ standard compliant. It can be compiled with g++ version 3.4. The problem was in classes derived from template base.
  - Added support for MacOS, Carbon API and the building environment for Code Warrior. Many thanks and regards to **Hansruedi Baer** (http://www.karto.ethz.ch/baer/agg). The project file is rather big (3MB), so it isn't included into the main distribution package. Please download http://www.antigrain.com/agg2.mcp.xml.zip and unzip it into agg2/examples/macos_cw/agg2.mcp.xml. Also, to build the SVG example, you need to obtain expat.lib and put it to agg2/svg/macos_cw/expat.lib. Download: http://www.antigrain.com/expat.lib.zip If you don't need to compile the SVG example, just remove svg_test from the project.
  - Added support for BeOS, many thanks to **Stephan Assmus**, http://www.yellowbites.com/index.html.
  - Resumed support for **SDL** (http://www.libsdl.org/index.php). Many thanks to **Mauricio Piacentini** who fixed some things in `agg_platform_support.cpp` for **SDL**.
  - Added `automake` building environment. Thanks to **Nikolas Zimmermann** (http://sourceforge.net/users/wildfox/).
  - Added **AGG** newsgroup to the gmane server:
    - News URL: news://news.gmane.org/gmane.comp.graphics.agg
    - Web URL: http://news.gmane.org/gmane.comp.graphics.agg
      Thanks to **Adam Megacz** http://www.megacz.com
  - **Finally!** added support for FreeType font rendering engine with cache. Now one can render any fonts very easily. The speed is pretty good too. When caching vectorial glyphs, it's about 30-40 microseconds per glyph on a typical P-IV 2.0 GHz, when caching scanline shapes, it's 3-4 microseconds. See also Demo truetype_test.cpp.
  - Added support for TrueType fonts, based on WinAPI `GetGlyphOutline()`. There's a general caching system that can be used with any font source, and two different font engine wrappers, for FreeType and Win32 API. One can write his own font engine, based on some API or library.

- **June 4, 2004**
  - Fixed a bug in the AGG Lite rasterizer. The rasterizer did not produce any result if all poins of the polygon fell to one pixel. See Download.
  - Added new page The Problem of Line Alignment to Tips and Tricks.

- **May 27, 2004**
  - **AGG** is updated. There is a new method of the polygon boolean algebra is added. In the distribution package (Download) there are two new demo examples, scanline_boolean.cpp and scanline_boolean2.cpp.
    See also Demo scanline_boolean.cpp and Demo scanline_boolean2.cpp.

- **May 18, 2004**
  - Added new page Working with Gradients to Tips and Tricks.
  - **AGG** is updated, see Download. There was fixed a bug in the rasterizer and added iterator-like methods to retrieve the scanlines. The work on the scanline shape algebra (Intersection, Union, Substraction, and Exclusive OR operations) is in progress.

- **May 9, 2004**
  - **AGG** is updated. The alpha-mask classes were modified in order to achieve better performance.
  - Added a new example, alpha_mask3.cpp with functionality similar to gpc_test.cpp. See the Demo page. It's polygon clipping based on the alpha-mask that is more restrictive than **General Polygon Clipper**, but works much faster.

- **May 8, 2004**
  - **AGG** is updated. Added new classes pixfmt_amask_adaptor and amask_no_clip_u8. It allows you to use the alpha-mask with all possible primitives and renderers. Besides, if the alpha-mask buffer is of the same size as the main rendering buffer (usually it is) we don't have to perform clipping for the alpha-mask, because all the primitives are already clipped at the higher level, so, it works faster. New demo example alpha_mask2.cpp is added.
  - Added a new section to the Documentation page decribing the use of pixfmt_amask_adaptor.
  - Added new page Compiling AGG under Microsoft eMbedded VC 4.0.

- **May 5, 2004**
  - The library has been updated, see the Download page. There are some minor changes and bug fix (not critical). The only change that affects the interface is that there was removed the default value of the 4-th template argument from span_gradient. Before there was `class ColorF = const ColorT*`. It was done in order to compile **AGG** successfully on **SunOS**. It can affect your code if you use an array of 256 rgba8 values as a color function of gradients (most probably you do because it's the easiest way). Just add one extra argument `const agg::rgba8*` when declaring the span_gradient.
  - Added new type of line joins to conv_stroke (vcgen_stroke to be exact). Many thanks to Dirck Blaskey for the contribution. Now there is one more type `join_miter_revert` that is used for compatibility with the PDF and SVG specifications. It works like `miter_join`, but when the miter limit is exceeded it turns into the `bevel_join`. **AGG** type of `miter_join` works "smoother" and (in my opinion) more correct. See the conv_stroke.cpp example for details and demo.
  - The library and the demo examples were successfully compiled under **SunOS** and **Irix64**. Page Demo is updated (actually, written at last). There you can find screenshots, brief description, and the executables for different hardware/OS platforms.
  - Added new example lion_lens.cpp that demonstrates non-linear "fish-eye like" transformations. See page Demo at the end. The trans_warp_magnifier and conv_segmentator classes were in **AGG** before, but they were not exposed.
  - New page Using WinAPI to Render Text was added to Tips and Tricks.
  - Little progress in writing the docs. See Basic Renderers.

- **April 17, 2004.**
  - Official **Anti-Grain Geometry** v2.1 released. See AGG Version 2.1 Release Notes and Download.

- Added page Tips and Tricks.
- The docs are in process. From now on I will update the documentation pages, as well as the source files. I do not expect any changes in code or interfaces; I'll just add some comments and cross references.

- **April 09, 2004.** Antigrain.com reopened. I have created my own documenting tool (see AGDoc Formatter), so that, now I can easily write new pages and update the site. Actually, **HTML** with its terrible syntax was the main obstacle to maintain the web-site.
  All this time I've been working on new algorithms and the design of the library. There are some things that were implemented:
  - Gradients and Gouraud Shading.
  - Image affine transformations.
  - Strokes with different types of line joins and line caps.
  - Dashed line generator.
  - Markers, such as arrowheads/arrowtails.
  - Fast vectorial polygon clipping to a rectangle.
  - Low-level clipping to multiple rectangular regions.
  - Alpha-Masking.
  - A fast anti-alias line algorithm.
  - Using arbitrary images as line patterns.
  - Rendering in separate color channels.
  - Perspective and bilinear transformations of vector and image data.
  - Boolean polygon operations (and, or, xor, sub) based on Alan Murta's **General Polygon Clipper**.

- **Two years of silence.** It was a period of active development of **Anti-Grain Geometry** as well as discussions with very nice people.

- **April 09, 2002.** Antigrain.com opened