Scanlines and Scanline Renderers

Scanline Containers

Introduction

The low level renderers operate with simplest data and they are very simple too. In fact, the pixel format renderers are not the obligatory part of the library and can be replaced or rewritten. For example, if you have an API with similar functionality, but hardware accelerated, it will be better to use it instead of pure software blending (the low level renderers are essentially alpha-blenders). It's also possible to use Intel SSE/SSE2 to write optimized renderers. All other rendering functionality in AGG is based on these simple classes.

To draw Anti-Aliased primitives one shoud **rasterize** them first. The primary rasterization technique in AGG is scanline based. That is, a polygon is converted into a number of horizontal scanlines and then the scanlines are being rendered one by one. Again, the scanline rasterizer is not the only class that can produce scanlines. It can be some container or even your own super-ultra-mega rasterizer.

To transfer information from a rasterizer to the scanline renderer there scanline containers are used. A scanline consists of a number of horizontal, non-intersecting spans. All spans must be ordered by X. It means that there is no sorting operation provided, the order must be perserved when adding spans to the scanline. If the order is not guaranteed it can result in undefined behaviour.

In **AGG** there are three types of scanline containers:

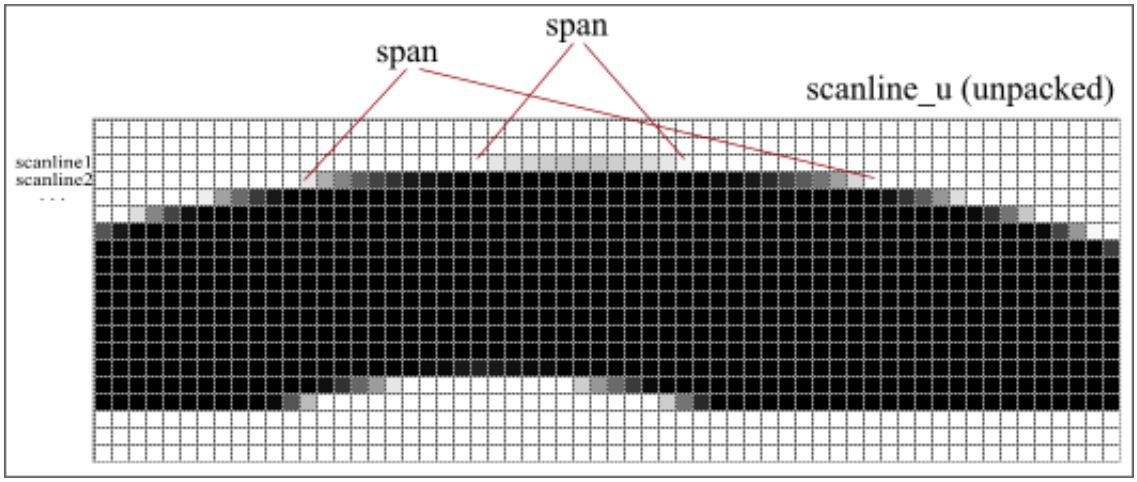
- `scanline_u` - unpacked scaline container
- `scanline_p` - packed scanline container
- `scanline_bin` - container for binary, "aliased" scanlines

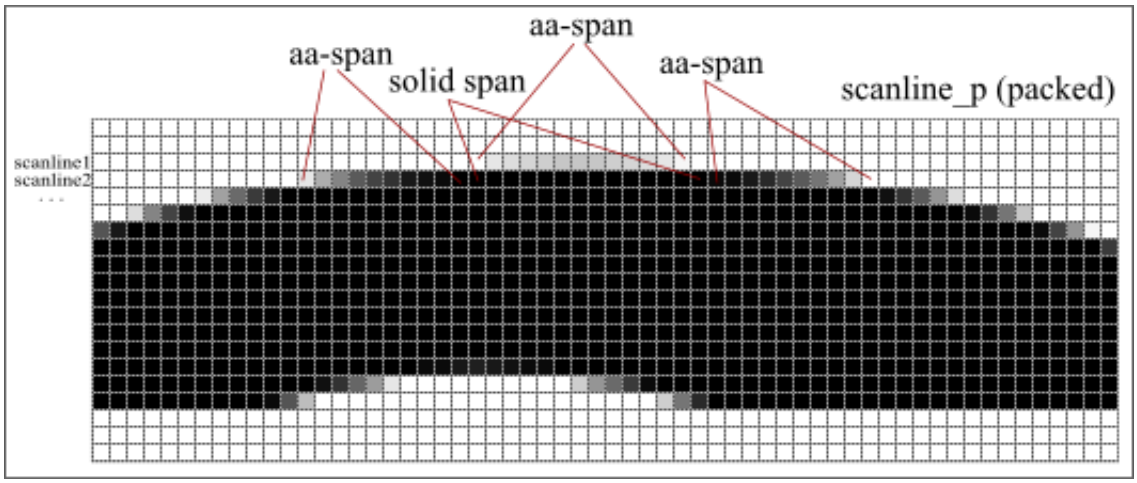
First two containers can keep **Anti-Aliasing** information, the third one cannot.

IMPORTANT!

All the scanline containers are optimized for speed, not for memory usage. In fact, it allocates memory for the worst case, so there is some overhead. It's not critical when you use only few scanline containers, but it's not a good idea to use an array of scanlines to store the whole shape because it will take more memory than the resulting image.

The difference between **packed** and **unpacked** scanline containers is that the **unpacked** scaline always keeps the coverage values for all pixels including the ones that are fully covered by a polygon. In the **packed** scanline all pixels with the same coverage value are merged into **solid** spans.





It can seem it's always better to use the packed version, but in practice it's not so. The scanline_p works faster when rendering large solid polygons, that is when the area of the polygon is much larger than the perimeter in the meaning of the number of pixels. But when renering text it's definitely better to use the scanline_u because of much less number of processed spans The difference is about three times and the number of spans is also important for the overall performance. Besides, in most of the span generators, such as gradients, Gouraud shader, or image transformers, the number of spans becomes even more critical, and so, the scanline_p is not used there.

Interfaces

Filling Interface (put)

Iterating Interface (get)

Scanline Renderers