A New Contour Method for Highly Detailed Geometry

Andreas Bauer
Polygon Pictures Inc.
andreasb@ppi.co.jp

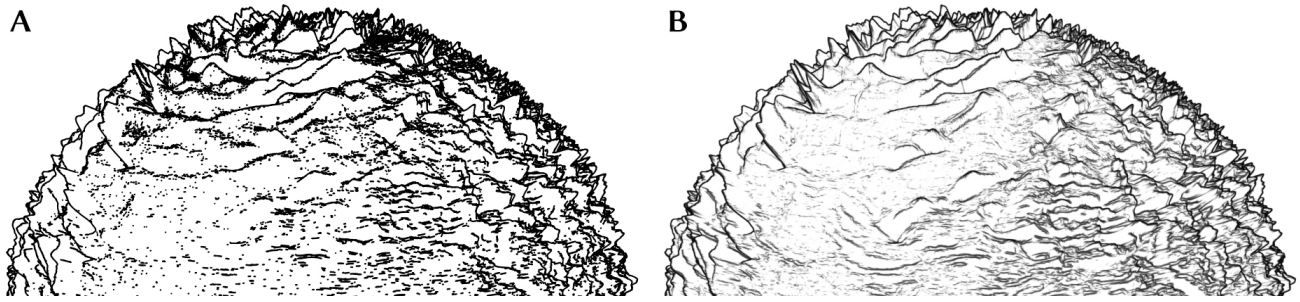

Figure 1: *A) Typical ray-traced contour result on highly detailed geometry. B) Same with the new contour method.*

ABSTRACT

Ray-traced contours are inherently challenged by highly detailed geometry, as commonly found in organic shapes. Existing contour methods cannot reflect such complexity in an artistically pleasing way (Figure 1 A), and animations are prone to flicker.

After a brief explanation of contour generation and its inherent challenges, this talk presents a novel approach to rendering aesthetic and flicker-free contours on highly detailed geometry (Figure 1 B). The new method employs sub-pixel-level sub-sampling to achieve a high level of detail quality, and supports contours in transparency, reflection and refraction.

The implementation uses mental ray (*Unified Sampling* mode) but could be realized in other ray tracing renderers as well.

CCS CONCEPTS

• **Computing methodologies~Non-photorealistic rendering** • *Computing methodologies~Ray tracing* • Computing methodologies~Antialiasing

KEYWORDS

non-photorealistic rendering, ray tracing, contours, shader, contour lens shader, mental ray

ACM Reference format:
Andreas Bauer. 2017. A New Contour Method for Highly Detailed Geometry. In *Proceedings of SIGGRAPH '17 Talks, July 30 - August 03, 2017, Los Angeles, CA, USA*, 2 pages.
DOI: 10.1145/3084363.3085052

SIGGRAPH '17 Talks, Los Angeles, CA, USA
 978-1-4503-5008-2/17/07.
DOI: 10.1145/3084363.3085052

1 MOTIVATION

The Godzilla Anime design brief demanded very high detail on the main character, at a level never before seen in anime. With existing contour methods, quality and motion stability was poor, even at high render settings. One solution was to render contour images separately at a larger size, and then to scale them down. But the immense detail required scale factors of at least ×4 with steep render time increases. In production this would be viable only for publicity stills, not for animation.

2 CONTOUR GENERATION

Contour generation is always a post process. Attributes of neighboring samples are compared in pairs and tested for a simple value difference, or whether their value delta exceeds a certain threshold. The two basic comparison tests are the angle between the surface normals N_1, N_2 versus an *nDelta*, and the difference between the camera distances Z_1, Z_2 versus a *zDelta*.

In production many more attributes are compared, such as object and material IDs, not only to generate contours but also to remove them where elements are to be visually merged.

3 INHERENT CHALLENGES

Unlike with surface color, the anti-aliasing of contours is not as simple as averaging comparison test results within a pixel. Averaging results is problematic since any single positive test is expected to draw a full contour. Transparency is also not a good way to express anti-aliasing of contours; modulating the contour width looks much more natural. But that width is independent of pixel size, so rather than within a pixel, results should probably be averaged over a disk region the diameter of the contour width. Anti-aliasing quickly becomes recursive.

The much better approach is to modulate each test result individually depending on by how far a threshold is exceeded. This works particularly well for *nDelta*. By defining a delta

range around the threshold, a weighting factor for the contour width can be created independent of the number of samples.

With geometry detail at sub-pixel level, comparison results tend to become more random, depending on where a ray hits. This problem is exacerbated by the fact that anime projects typically use flat shading, and that lack of color detail causes the renderer to cast fewer rays, which in return increases the chance of missing tiny geometry detail. In animation this frequently causes contour flicker, especially at grazing angles.

4 A NEW APPROACH

The solution is the inverse of rendering contour images at a larger size: a sub-pixel-level sub-sampling grid. All attributes for contour comparison tests and contour drawing are stored and independently tested in a per pixel grid of 4×4, 6×6 or larger, depending on the desired level of detail quality.

Contour processing can be fully deferred until after all data is collected, or semi-deferred until after each ray sample.

4.1 Fully Deferred Processing

In shader terms, an *output shader* is called after the main render completes. Since this type of shader has access only to *framebuffers* (*AOVs*), all contour related data has to be handled via framebuffers. With the expected number of attributes (30), grid size (8×8) and maximum ray depth (5), the total number of required framebuffers approaches a staggering 10,000. This is likely not practical in production.

4.2 Semi-Deferred Processing

In shader terms, a *camera lens shader* is called before each ray sample and continues after it. Since rendering is usually done in little blocks, called *tiles* or *buckets*, contour related data would only need to be stored per tile, making it small enough to fit into shader memory. While this approach does not require framebuffers, it has its own set of challenges:

Render Tile Overlap

The key challenge is tile discontinuity. Grid elements at the very edge of a tile have no neighboring attributes to compare. A little known feature in mental ray saves the day: *render tile overlap*. In *Unified Sampling* mode the image filter size creates an extra area around each tile, rendering a bit more than the actual tile size, just for filtering. This overlap is also sufficient for contour comparison tests along tile edges.

Continuous Processing

There is no event indicating *'the last sample of a pixel'*, after which contour processing could start. Instead, processing has to be continuous, updating the contour with every ray sample.

5 IMPLEMENTATION

The new method uses just two shaders, making setup easy:

- A *surface shader* which renders the surface color as well as collects contour related data at every ray intersection point.
- A *contour lens shader* that generates contours from this data.

5.1 Contour Generation Comparison Tests

nDelta tests compare 4 pairs of grid elements equally distanced but diagonally opposite around a center element (the current ray sample intersection point). *zDelta* tests compare the center element with the one furthest away from it in Z distance (from amongst its 8 neighbors). Object ID and similar tests compare min against max within nine grid elements (the center plus 8 neighbors). These tests are performed two times: once for the immediate 8 grid neighbors, to test for closest geometry detail, and once for 8 distant grid elements spaced half the contour width away, the maximum distance to test for contour generation. All the above tests are performed at each ray depth.

5.2 Contour Output

Once contour color and width are calculated, a disk image of diameter = width is written into an internal contour image buffer, at the same resolution as the sub-sampling grid. As the last step, all contour image buffer elements within the current pixel are averaged and output as the final pixel contour value.

The renderer's ray sample distribution algorithm considers the alpha channel when deciding where to cast extra rays. To bias that distribution towards areas with already found contour lines, the alpha value is output as 1.0 until ray samples inside a pixel reach the *Minimum Samples* setting. From then on the actual contour image buffer alpha value is returned. With this temporarily over-enhanced alpha, *Minimum Samples* can be set much lower than the number of grid elements per pixel.

5.3 Render Settings

For optimum results, the *Quality* setting must be very high, *Minimum Samples* a bit above half the number of grid elements per pixel, and *Maximum Samples* 100-250, low yet high enough to allow the *Quality* setting to cast extra rays if necessary.

5.4 Semi-Deferred Processing Trade-Offs

Since contours are processed continuously, from data that is only slowly improving with every ray sample, final contours are not as clean as those produced by a fully deferred process.

Semi-deferred processing depends on the *render tile overlap*, which itself is derived from the image filter size. That filter size ultimately limits the maximum contour width. In production this is not an issue, though, because contours are rarely thicker than 2-4 pixels, the typical image filter size range.

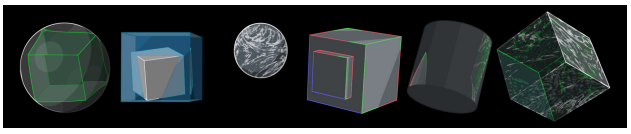

Figure 2: *Further contour cases supported by the new method (image rendered 'as is' in one pass with no post-processing). Left to right: refracted contours, auto-composited contour colors, clean contours from bump maps, shading-dependent contour colors, reflected contours, double-sided contours in transparent objects.*