Eurographics Symposium on Geometry Processing 2015
Mirela Ben-Chen and Ligang Liu
(Guest Editors)

Volume 34 (2015), Number 5

Supplementary Materials for Quaternion Julia Set Shape Optimization

Theodore Kim, University of California, Santa Barbara

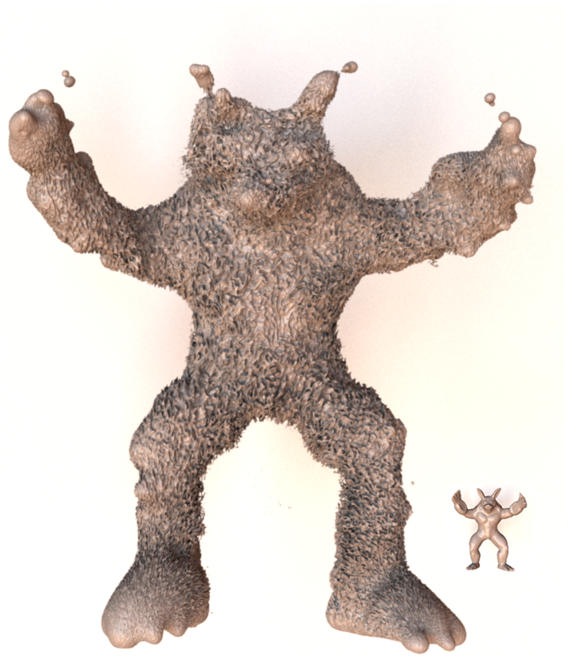

Figure 1: *The porous, lava rock-like surface of a 546-root approximation of the Armadillo (inset). The large numbers of roots eventually results in exponent overflow, and terminates the optimization early. The overall shape is visible, but thin features such as the fingers, ears and toes are under-resolved.*

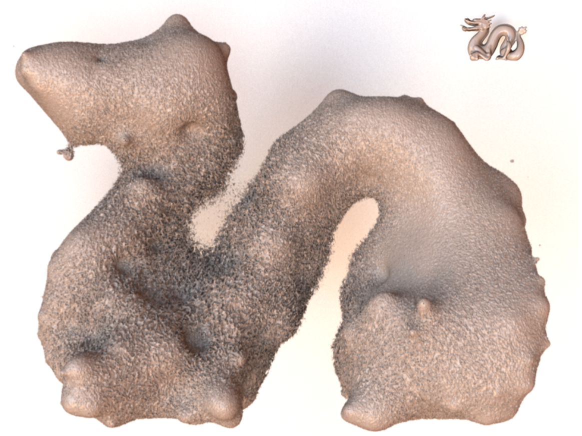

Figure 2: *A 708-root approximation of the Dragon (inset). This was the most difficult optimization attempted, and terminated the earliest due to exponent overflow. The broad shape is still captured, and the hyper-complexity is particularly visible between the Dragon's neck and body.*

1. Additional Results

Two larger optimizations using more complex target shapes were also attempted. The Armadillo (Fig. 1) and the Dragon (Fig. 2). Both terminated early because the computation ran out of exponent bits, but the overall shape of the target shape is still clearly visible.

2. Marching Cubes

The performance of our Marching Cubes implementation is shown in Table 1. The running time of the algorithm is roughly $O(nRN^3)$, where n is the number of recursions. The N^3 dependency is visible in the 1000^3, 2000^3 and 4000^3 runs for the Bunny example, as each doubling increases the running time by a factor of 8. Please note that these numbers are intended to show the scaling of the algorithm. The 294 hour, 124 million triangle, 4000^3 case is a stress-test, not the typical running time. All of the renderings instead use the 1000^3 resolution.

The R-dependency can be seen by the increase in running times as the root complexities grow. The Armadillo example deviates from this slightly, as its shape occupies the smallest number of grid cells of any of the models, so more of the exterior cells diverge to ∞ after only 1 or 2 iterations.

Marching cube extensions such as dual contouring or octrees could potentially improve the quality or generation time for the meshes, but it is not clear how to apply these methods. Octree methods require a method of determining whether regions of space can be skipped, but there is no reliably way of detecting spatial homogeneity in Julia sets. Dual contouring methods produce higher quality meshes by leveraging a surface normal, but computing the analytic normal for these Julia sets explode in complexity as the number of

Example	Running Time	Grid Res.	Triangles
Tooth	1:06:07	1000^3	5,473,218
Bunny	5:43:55	1000^3	7,588,775
Frog	6:20:26	1000^3	7,443,640
Armadillo	5:49:49	1000^3	9,555,448
Dragon	11:00:17	1000^3	25,846,616
Bunny	40:00:18	2000^3	30,373,962
Bunny	293:40:14	4000^3	124,284,594

Table 1: *Running time of non-linear Marching Cubes. In all cases,* **R** *is recursively evaluated 4 times.*

iterations increase. Adapting more sophisticated Marching Cubes extensions to that they can be leveraged for this problem remains future work.

3. Numerical Considerations

Since we are dealing with high-order rational functions containing R roots, numerical issues begin to appear. More details emerge as the number of recursions n are increased, but the degree of the rational function increases correspondingly, effectively scaling according to $O(R^n)$.

As a result, all of our computations had to be performed in 80-bit extended precision, the maximum precision natively supported by Intel hardware. Many numerical applications tend to run out of precision in the mantissa, but as we are raising quaternions to large, arbitrary powers, we found that our computations instead ran out of bits in the exponent. Unfortunately, this means that hardware support for 128-bit IEEE 754 quad-precision would not help, as it contains the same 15 exponent bits as extended precision. Porting the algorithm to GPUs also becomes non-trivial, as they currently support a maximum of 64-bit doubles. Horner's rule cannot be used to stabilize factored quaternion rationals, so no assistance is available in that direction either.

We ran several tests with the arbitrary-precision GNU MPFR library to confirm that additional exponent bits would reduce the number of NaNs that appeared under large n. While their number reduced, even 100 exponent bits were not sufficient to eliminate them all, even for a modest number of recursions ($n = 10$). MPFR already runs an order of magnitude slower than hardware-supported precisions, so the more practical alternative is to investigate more numerically well-behaved representations such as the Lagrange form described in the main paper.