Specular Polynomials: The Supplemental Document

ANONYMOUS AUTHOR(S)

ACM Reference Format:
Anonymous Author(s). 2024. Specular Polynomials: The Supplemental Document. *ACM Trans. Graph.* 43, 4, Article 1 (August 2024), 4 pages. https://doi.org/10.1145/3618360

1 IMPLEMENTATION

In this section, we describe our implementation details with pseudo-codes, and we will release the source code upon acceptance.

1.1 Construction of bivariate specular polynomials

We present the pseudo-code for computing the coefficients of bivariate specular polynomials, taking R chain for example. Most of the variables are bivariate polynomials or 3D vectors of them.

```
def bivar_poly_r(pD, pL, p10, p11, p12, n10, n11, n12:
↪  BivarPoly3D): # The arguments are 3D vectors of
↪  bivariate polynomials of degree 0
    # Variables
    u1 = BivarPoly([[0, 0], [1, 0]])
    v1 = BivarPoly([[0, 1], [0, 0]])

    # Specular vertex and normal
    xD = pD
    xL = pL
    x1 = p10 * (1 - u1 - v1) + p11 * u1 + p12 * v1
    n1 = n10 * (1 - u1 - v1) + n11 * u1 + n12 * v1

    e11 = p11 - p10 # Edge

    t11 = n1.cross(e11) # Tangent

    # Position difference
    d0 = x1 - xD
    d1 = xL - x1
    d0_dot_n1 = d0.dot(n1)
    d1_dot_n1 = d1.dot(n1)
    d0_dot_t11 = d0.dot(t11)
    d1_dot_t11 = d1.dot(t11)

    # Coplanarity constraints
    s = xL - xD
    cop = (d0.cross(s)).cross(n1.cross(s))
    a = cop.bvp[0]

    # Angularity constraints
    b = d0_dot_n1 * d1_dot_t11 + d0_dot_t11 * d1_dot_n1

    # The bivariate polynomial system in Eq. (23)
    return a, b
```

Author's address: Anonymous Author(s).

1.2 Construction of resultant matrices

In our implementation, we adopt a fast computation method [Chionh et al. 2002] to construct the Bézout resultant matrix:

```
def bezout_resultant(a: BivarPoly, b: BivarPoly):
    n = max(a.degree, b.degree)
    r = Matrix(n)
    for i in range(n):
        for j in range(i, n):
            r[i][j] = a[i]*b[j+1]-b[i]*a[j+1] # a[i] is a
            ↪  univariate polynomial in v_1
    for i in range(1, n-1):
        for j in range(i, n-1):
            b[i][j] += b[i-1][j+1]
    for i in range(n):
        for j in range(i):
            b[i][j] = b[j][i]
    return r
```

Note that n must be the actual degree. Using a degree larger than its actual value will leads to a zero determinant for all v_1.

1.3 Expansion of univariate polynomial determinants

For R and T chains, we expand the determinant of the univariate matrix polynomial using Laplacian expansion. Special care of computation order is required to minimize the number of multiplication operations. For R, the order of determinant is 4, and we divide it into computing 12 determinants of 2×2 matrices:

```
def det4(matrix: Matrix[UnivariatePolynomial]):
    # Extract matrix elements
    a, b, c, d = matrix[0], matrix[1], matrix[2], matrix[3]

    # Calculate the determinant using the expansion by
    ↪  minors formula
    temp=(a[0]*b[1] - a[1]*b[0]) * (c[2]*d[3] - c[3]*d[2])-\
       (a[0]*b[2] - a[2]*b[0]) * (c[1]*d[3] - c[3]*d[1]) + \
       (a[0]*b[3] - a[3]*b[0]) * (c[1]*d[2] - c[2]*d[1]) + \
       (a[1]*b[2] - a[2]*b[1]) * (c[0]*d[3] - c[3]*d[0]) - \
       (a[1]*b[3] - a[3]*b[1]) * (c[0]*d[2] - c[2]*d[0]) + \
        (a[2]*b[3] - a[3]*b[2]) * (c[0]*d[1] - c[1]*d[0])

    return temp
```

For T chain, the order is 6, and we compute 20 determinants of 3×3 matrices. The implementation is analogous.

1.4 Univariate solver

Univariate polynomial solver. Recursively determining the monotonic pieces of a univariate polynomial by its derivative can effectively determine the zeros of a polynomial, which serves as a form of root isolation [Collins and Loos 1976]. The derivative of a polynomial of degree d is a polynomial of degree $d-1$. Within a monotonic piece, the root finding is straightforward. We adopt

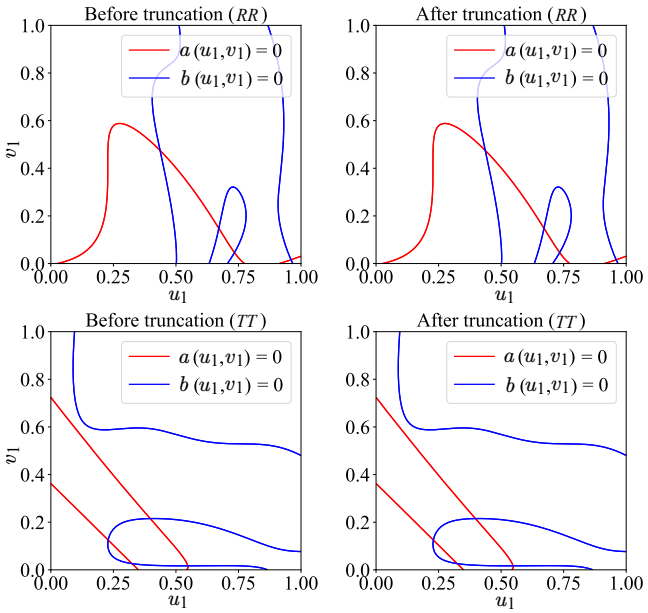

Fig. 1. Validation of the truncation.

the implementation of [Yuksel 2022], which has guaranteed global convergence and high performance.

Piecewise bisection solver. The algorithm begins by uniformly partitioning the $[0, 1]$ interval into m pieces and evaluating the determinant values at all endpoints. If the determinant signs at the endpoints of a piece are different, we perform a bisection on that segment. Here, we choose the bisection solver because it is free of derivative computation of high-order determinants and can be massively parallelized. Additionally, when evaluating the determinant, since the bisection solver only needs its sign, we do not need to compute its value. Note that computing the determinant of a large matrix can be slow and numerically unstable. We find it safe to truncate the matrix stemming from the bivariate specular polynomials to a small degree such that the maximum absolute value of the coefficients of the truncated terms is less than ϵM, with M being the maximum absolute value of the coefficients of all terms. We set ϵ to 10^{-9} in our implementation, which bounds the maximum error. Two examples are shown in Fig. 1. As seen, the truncation makes almost no difference to the contour of bivariate equations.

Eigenvalue solver. We empoly linearization [Golub and Van Loan 2012] to convert the univariate root-finding problem into a generalized eigenvalue problem. We use the RealQZ module in the Eigen library [Guennebaud et al. 2010] to perform real QZ decompositions of a pair of square matrices. We set the maximal number of eigenvalue decomposition iterations to 100. The resulting eigenvalues corresponds to the solution v_1.

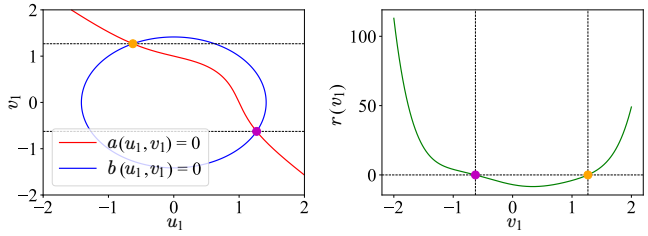

Fig. 2. Plot of the bivariate polynomial system in Eq. (1) and its resultant in Eq. (3). The two intersections in the left diagram correspond exactly to the two roots in the right diagram.

2 DISCUSSIONS ON RESULTANTS

A running example. Let's take the following bivariate polynomial system as an example:

$$\begin{cases} a(u_1, v_1) = u_1^3 + v_1^3 + u_1 v_1 - 1 = 0, \\ b(u_1, v_1) = u_1^2 + v_1^2 - 2 = 0. \end{cases} \tag{1}$$

Its resultant matrix is

$$\boldsymbol{R}(v_1) = \begin{bmatrix} -v_1^3 + 2v_1 & v_1^3 - 1 & 2 - v_1^2 \\ v_1^3 - 1 & -v_1^2 + v_1 + 2 & 0 \\ 2 - v_1^2 & 0 & -1 \end{bmatrix}, \tag{2}$$

and the corresponding resultant is

$$r(v_1) = \det \boldsymbol{R}(v_1) = 2v_1^6 - 2v_1^5 - 5v_1^4 + 6v_1^3 + 10v_1^2 - 8v_1 - 7. \tag{3}$$

Now, we can sequentially solve v_1 and u_1 according to Eq. (3) and Eq. (1). We visualize the bivariate polynomial system and the resultant in Fig. 2. As seen, each root of the bivariate polynomial system corresponds to a zero of the resultant.

Necessary condition. We briefly note that $r(v_1) = 0$ is only a necessary condition of the original system to have a solution, i.e.,

$$\begin{cases} a(u_1, v_1) = 0, \\ b(u_1, v_1) = 0, \end{cases} \Rightarrow r(v_1) = 0. \tag{4}$$

It is not sufficient, which means for a given v_1^* that satisfies $r(v_1^*) = 0$, there may be no valid u_1 such that $a(u_1, v_1^*) = b(u_1, v_1^*) = 0$. Fortunately, there are a finite number of solutions of $r(v_1) = 0$, so we can validate them one by one before finally reporting a solution.

Resultant forms. Various forms of resultants are available [Kapur and Saxena 1995]. In the case of two bivariate polynomials $a(u_1, v_1)$ and $b(u_1, v_1)$ with degrees n and m ($n \geq m$) respectively, the Sylvester [Sylvester 1853] resultant is simple to compute, where each element of the matrix only involves a specific coefficient of the polynomials. However, the size of the matrix is $n + m$. On the other hand, the Bézout resultant matrix is much more compact [Bézout 1779], with a size of only n. We opt for the Bézout resultant due to its smaller size, which leads to a more efficient computation.

Resultant vs. manual elimination. Recall that Section 4.4 entails a manual elimination process using the rational mapping between coordinates. The decision to opt for this method over a direct application of resultant elimination arised from considerations of the degree. Upon each application of bivariate resultants to eliminate

Table 1. Coefficients and maximum error of rational functions

Index	Left endpoint	Right endpoint	$c_{0,i}$	$c_{1,i}$	$d_{0,i}$	$d_{1,i}$	Max error
0	0.000	0.005	1.06939×10^{-1}	1.24883×10^{2}	6.44864	7.79412×10^{2}	0.0005
1	0.005	0.020	1.05021×10^{-1}	3.12337×10^{1}	3.20289	9.78683×10^{1}	0.0005
2	0.020	0.080	1.30984×10^{-1}	9.75997	2.00015	1.52961×10^{1}	0.0009
3	0.080	0.200	3.76068×10^{-1}	8.89489	3.19627	8.11322	0.0005
4	0.200	0.500	4.56906×10^{-1}	4.32322	2.45619	2.49402	0.0007
5	0.500	1.000	9.38873×10^{-1}	4.10143	3.41291	1.62996	0.0006

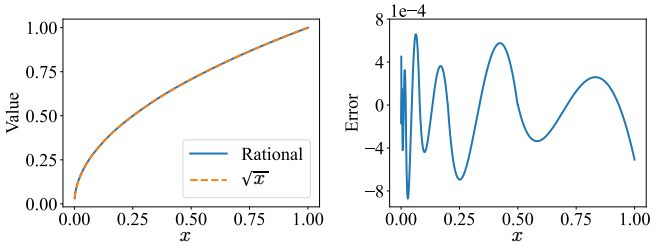

Fig. 3. Validation of our rational approximation to $\sqrt{x}$ in the range of $[0, 1]$. The error is less than 10^{-3}.

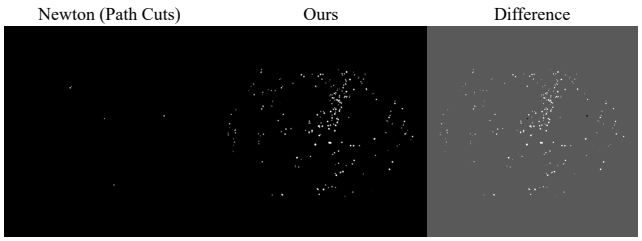

Fig. 4. The glint-only image of the relief scene featuring TT specular chains.

one variable, the resulting degree becomes nearly squared [Nakatsukasa et al. 2015]. Besides, employing multivariate resultants has proven impractical due to issues of numerical instability [Noferini and Townsend 2016]. This underscores the necessity for our manual elimination.

Implementation. Several existing packages are available for computing the zeros of bivariate polynomials using the Bézout resultant matrix [Meurer et al. 2017; Townsend and Trefethen 2013]. However, their performance typically ranges in the order of hundreds of *milliseconds* for each tuple of triangles, making them impractical for rendering purposes. In contrast, our pipeline is extremely efficient, requiring only several *microseconds* for two bounces and less than one *microsecond* for a single bounce.

3 RATIONAL APPROXIMATION FOR SQUARE ROOTS

As discussed in Sec. 4.4, we employ an approximant for $\sqrt{\beta_i}$. Let μ_i be the upper bound[1] of $\boldsymbol{n}_i^2 \boldsymbol{d}_{i-1}^2$ and $x = \beta_i / \mu_i$. Our objective is to find a good approximant of $\sqrt{x}$ for $x \in [0, 1]$. Using a polynomial approximant is inaccurate since the derivative of $\sqrt{x}$ approaches infinity as x tends to zero. Therefore, we employ a rational expression, i.e., the fraction of two polynomials. We divide the interval $[0, 1]$ into 6 bins and use a fraction of two one-degree polynomials. We optimize the interpolant's coefficients in each bin using the least square method. We weigh the samples such that the value at the left and right sides of each endpoint is nearly the same. Consequently, the maximum error on each interval is less than 10^{-3}, as shown in Fig. 3. The coefficients are shown in Table 1.

[1]Notice that $0 \le \beta_i \le \boldsymbol{n}_i^2 \boldsymbol{d}_{i-1}^2$. Therefore, the upper bound μ_i of $\boldsymbol{n}_i^2 \boldsymbol{d}_{i-1}^2$ can be obtained as we can compute an upper bound of $\boldsymbol{d}_{i-1}^2$ using the vertex locations and $0 \le \boldsymbol{n}_i^2 \le 1$.

4 COMPLEXITY

For a given tuple of triangles, the time complexity of computing bivariate polynomial coefficients, Bézout matrices, and piecewise bisections is $O(n^4)$, $O(n^3)$, and $O(Cn^3)$, respectively. Here, C is the product of the number of intervals and bisection iterations, and n is the degree of the bivariate systems:

$$n = \begin{cases} 4 \prod_{i=1}^{l-1} n_i, & \text{for specular reflection on } \boldsymbol{x}_k \\ 6 \prod_{i=1}^{l-1} n_i, & \text{for specular refraction on } \boldsymbol{x}_k \end{cases} \tag{5}$$

with k being the length of specular chains and the factor n_i further relies on the scattering type on $\boldsymbol{x}_i$:

$$n_i = \begin{cases} 4, & \text{for specular reflection on } \boldsymbol{x}_i \\ 8, & \text{for specular refraction on } \boldsymbol{x}_i \end{cases}. \tag{6}$$

Therefore, the time complexity for solving all chains given two separators is $O(t(n^4 + Cn^3))$. Here, t is the number of triangle tuples, which has an upper bound of m^k with m being the number of triangles in the scene, but it is much lower in practice thanks to the efficient pruning techniques [Walter et al. 2009; Wang et al. 2020]. The space complexity is $O(n^3 + C)$, which does not scale as the number of triangles grows. Additionally, for the eigenvalue-based solver, the time complexity will become $O(tCn^6)$, with C being the number of QZ iterations. The space complexity is $O(n^4)$.

ACKNOWLEDGMENTS

...

REFERENCES

Étienne Bézout. 1779. *Théorie Générale des Équations Algébriques.* Ph. D. Dissertation. Pierres, Paris.

Eng-Wee Chionh, Ming Zhang, and Ronald N. Goldman. 2002. Fast Computation of the Bezout and Dixon Resultant Matrices. *Journal of Symbolic Computation* 33, 1 (Jan. 2002), 13–29. https://doi.org/10.1006/jsco.2001.0462

George E. Collins and Rüdiger Loos. 1976. Polynomial real root isolation by differentiation. In *Proceedings of the Third ACM Symposium on Symbolic and Algebraic Computation* (Yorktown Heights, New York, USA) *(SYMSAC '76)*. Association for Computing Machinery, New York, NY, USA, 15–25. https://doi.org/10.1145/800205.806319

Gene H. Golub and Charles F. Van Loan. 2012. *Matrix Computations* (4th ed.). Johns Hopkins University Press.

Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3. http://eigen.tuxfamily.org.

Deepak Kapur and Tushar Saxena. 1995. Comparison of Various Multivariate Resultant Formulations. In *Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation (ISSAC '95)*. Association for Computing Machinery, New York, NY, USA, 187–194. https://doi.org/10.1145/220346.220370

Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. 2017. SymPy: symbolic computing in Python. *PeerJ Computer Science* 3 (Jan. 2017), e103. https://doi.org/10.7717/peerj-cs.103

Yuji Nakatsukasa, Vanni Noferini, and Alex Townsend. 2015. Computing the Common Zeros of Two Bivariate Functions via Bézout Resultants. *Numer. Math.* 129, 1 (Jan. 2015), 181–209. https://doi.org/10.1007/s00211-014-0635-z

Vanni Noferini and Alex Townsend. 2016. Numerical Instability of Resultant Methods for Multidimensional Rootfinding. *SIAM J. Numer. Anal.* 54, 2 (Jan. 2016), 719–743. https://doi.org/10.1137/15M1022513

James Joseph Sylvester. 1853. On a Theory of the Syzygetic Relations of Two Rational Integral Functions, Comprising an Application to the Theory of Sturm's Functions, and That of the Greatest Algebraical Common Measure. *Philosophical Transactions of the Royal Society of London* 143 (1853), 407–548. https://doi.org/10.1098/rstl.1853.0018

Alex Townsend and Lloyd N. Trefethen. 2013. An Extension of Chebfun to Two Dimensions. *SIAM J. Sci. Comput.* 35, 6 (jan 2013), C495–C518. https://doi.org/10.1137/130908002

Bruce Walter, Shuang Zhao, Nicolas Holzschuch, and Kavita Bala. 2009. Single Scattering in Refractive Media with Triangle Mesh Boundaries. *ACM Trans. Graph.* 28, 3, Article 92 (jul 2009), 8 pages. https://doi.org/10.1145/1531326.1531398

Beibei Wang, Miloš Hašan, and Ling-Qi Yan. 2020. Path Cuts: Efficient Rendering of Pure Specular Light Transport. *ACM Trans. Graph.* 39, 6, Article 238 (nov 2020), 12 pages. https://doi.org/10.1145/3414685.3417792

Cem Yuksel. 2022. High-Performance Polynomial Root Finding for Graphics. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 5, 3 (July 2022), 1–15. https://doi.org/10.1145/3543865