EUROGRAPHICS 2025 / A. Bousseau and A. Dai
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 44 (2025), Number 2

Axis-Normalized Ray-Box Intersection - Supplemental Material

F. Friederichs[1] and C. Benthin[2] and S. Grogorick[1] and E. Eisemann[3] and M. Magnor[1] and M. Eisemann[1]

[1]TU Braunschweig, Germany [2]Advanced Micro Devices, Inc. [3]Delft University of Technology, The Netherlands

1. Detailed Benchmark Results

1.1. Default Release Build

Tables 1 and 2 list the full results of our benchmarks, compiled with clang++ 15.0.7 and g++ 12.3.0, using the default CMake Release-build compiler flags. We include the existing algorithm implementations in the test suite in [EMGM07], and also keep the naming convention:

int In addition to the binary intersection test, the closest intersection distance is computed.
div For computing the intersection distance, a division by the ray direction is used.
mul The intersection distance is computed by multiplying with the precomputed reciprocal of the ray direction.
cls Precomputed ray classification information is used (For the Plücker and Slope tests).
cff The test uses precomputed quantities like Plücker coordinates.
branchless The test is a branchless implementation, omitting all early exit opportunities.
opt The test includes the load offset optimization from Embree.

Our method is prefixed with *normrays*. We recorded results for 10, 100 and 1000 boxes per ray, and otherwise used the same test parameters as mentioned in the paper (10k rays, 5k iterations). The standard deviations of the measured times were also recorded to ensure we don't have excessive noise in our data.

1.2. Compiling with Native ISA Extensions

In Tables 3 and 4 we show results for another test run, but this time with inclusion of the `-march=native` compiler flag, which allows the compiler to use any instruction set extensions available on the current CPU (AMD Ryzen R9 7950X in our case). For clang, both the binary and the distance-returning test still benefit from our method. Compiled with GCC, the binary test is also faster, but the distance-returning one is minimally slower. We also again include the performance plots for the algorithms compared in the paper, in Figs. 1 and 2. Note that these results strongly depend on the used CPU and compiler.

2. Ray Normalization Scheme Implementation

Listing 1 shows the code we used in our benchmarks to transform the original ray to our representation. Note that explicitly extracting the sign bit of `std::signbit` is mandatory here, because a comparison like `x < 0` for retrieving the sign would fail for -0 values. `std::signbit` simply extracts the most significant bit of the floating point representation. As explained in the paper, `safe_rcp` is necessary for preventing NaNs in both the original Embree test and ours.

References

[EMGM07] EISEMANN M., MAGNOR M., GROSCH T., MÜLLER S.: Fast ray/axis-aligned bounding box overlap tests using ray slopes. *Journal of graphics tools 12*, 4 (2007), 35–46. 1

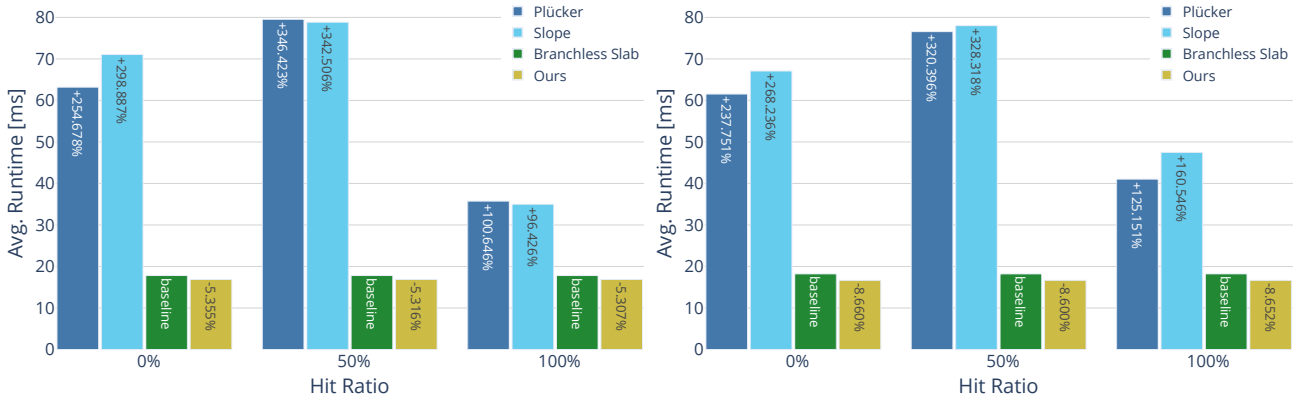

Figure 1: *Performance results of the binary intersection test for g++12.3.0 (left) and clang-15.0.7 (right) with* `-march=native`*. Relative performance to the optimized branchless slab test (baseline) is noted in the bars.*

Figure 2: *Performance results for g++12.3.0 (left) and clang-15.0.7 (right) with* `-march=native`*, with intersection distances.*

		Mean runtime in ms			Runtime std. dev. in ms		
	AABBs per Ray	10	100	1000	10	100	1000
0% Hit Ratio	standard_div	1.341	12.996	125.923	0.003	0.006	0.028
	standard_mul	0.819	7.760	74.724	0.002	0.004	0.026
	pluecker	0.918	7.355	63.789	0.002	0.005	0.025
	plueckerint_div	0.917	7.337	63.605	0.002	0.004	0.057
	plueckerint_mul	0.917	7.395	64.048	0.002	0.006	0.047
	pluecker_cls	0.991	7.960	67.901	0.004	0.012	0.030
	pluecker_cls_cff	0.992	7.922	66.753	0.007	0.024	0.050
	plueckerint_div_cls	1.081	8.491	69.814	0.005	0.008	0.038
	plueckerint_div_cls_cff	1.005	8.143	67.304	0.006	0.038	0.033
	plueckerint_mul_cls	1.177	9.125	72.181	0.006	0.063	0.278
	plueckerint_mul_cls_cff	0.992	7.809	66.141	0.006	0.014	0.037
	slope	1.011	8.005	67.995	0.006	0.013	0.031
	slopeint_div	1.033	8.179	68.434	0.005	0.019	0.036
	slopeint_mul	1.140	8.936	72.004	0.003	0.048	0.040
	smits_div	1.272	12.710	123.393	0.006	0.006	0.022
	smits_div_cls	1.291	11.567	104.192	0.006	0.010	0.030
	smits_mul_cls	1.169	10.254	91.822	0.006	0.010	0.023
	smits_mul	0.740	7.357	70.353	0.004	0.004	0.119
	smitsint_mul	0.734	7.244	69.102	0.004	0.013	0.119
	smits_mul_branchless_opt	0.411	3.921	39.926	0.002	0.013	**0.016**
	smitsint_mul_branchless_opt	0.421	4.056	40.327	0.001	**0.002**	0.016
	normrays_mul_branchless_opt	**0.360**	**3.446**	**34.416**	**0.001**	0.002	0.017
	normraysint_mul_branchless_opt	0.373	3.603	36.017	0.001	0.002	0.018
50% Hit Ratio	standard_div	1.584	16.022	161.788	0.002	0.004	0.021
	standard_mul	0.953	9.509	95.567	0.002	0.003	**0.017**
	pluecker	0.894	7.810	79.905	0.002	0.003	0.018
	plueckerint_div	1.290	11.521	115.443	0.002	0.006	0.109
	plueckerint_mul	1.225	10.950	109.645	0.002	0.006	0.103
	pluecker_cls	0.881	7.865	80.008	0.004	0.007	0.017
	pluecker_cls_cff	0.875	7.860	80.091	0.004	0.004	0.047
	plueckerint_div_cls	1.371	12.142	116.255	0.012	0.021	0.059
	plueckerint_div_cls_cff	1.344	11.934	116.979	0.007	0.009	0.033
	plueckerint_mul_cls	1.350	11.910	113.270	0.010	0.018	0.081
	plueckerint_mul_cls_cff	1.252	11.268	110.870	0.008	0.010	0.037
	slope	0.890	7.923	80.365	0.004	0.011	0.026
	slopeint_div	1.355	11.927	115.557	0.008	0.010	0.036
	slopeint_mul	1.394	12.254	115.543	0.007	0.030	0.180
	smits_div	1.545	15.626	157.894	0.002	0.004	0.020
	smits_div_cls	1.288	12.303	124.174	0.008	0.006	0.021
	smits_mul_cls	1.199	11.133	111.420	0.006	0.009	0.022
	smits_mul	0.898	9.161	92.551	0.002	0.007	0.029
	smitsint_mul	0.899	9.067	91.516	0.003	0.055	0.646
	smits_mul_branchless_opt	0.409	3.915	39.908	0.002	0.010	0.018
	smitsint_mul_branchless_opt	0.419	4.059	40.310	0.001	**0.002**	0.018
	normrays_mul_branchless_opt	**0.358**	**3.446**	**34.407**	0.001	0.002	0.020
	normraysint_mul_branchless_opt	0.371	3.608	36.012	**0.001**	0.002	0.019
100% Hit Ratio	standard_div	0.916	9.099	90.713	0.001	0.003	0.016
	standard_mul	0.367	3.592	35.556	0.001	0.002	0.020
	pluecker	0.555	4.555	44.311	0.001	0.003	0.020
	plueckerint_div	1.330	11.858	133.438	0.002	0.007	0.056
	plueckerint_mul	1.193	10.535	116.581	0.002	0.004	0.037
	pluecker_cls	0.519	4.750	48.288	0.004	0.089	0.235
	pluecker_cls_cff	0.514	4.559	45.332	0.008	0.073	0.646
	plueckerint_div_cls	1.272	11.854	134.591	0.007	0.043	0.107
	plueckerint_div_cls_cff	1.364	12.604	139.290	0.004	0.012	0.090
	plueckerint_mul_cls	1.160	10.527	116.569	0.008	0.068	0.195
	plueckerint_mul_cls_cff	1.214	11.279	123.511	0.003	0.016	0.068
	slope	0.516	4.744	45.580	0.007	0.051	0.390
	slopeint_div	1.336	12.297	134.689	0.003	0.011	0.109
	slopeint_mul	1.292	11.483	125.938	0.003	0.121	0.284
	smits_div	0.890	8.829	88.096	0.001	0.007	**0.015**
	smits_div_cls	0.581	5.046	49.643	0.008	0.002	0.017
	smits_mul_cls	0.613	4.600	45.276	0.017	0.008	0.027
	smits_mul	**0.305**	**3.059**	**30.385**	0.001	0.009	0.023
	smitsint_mul	0.312	3.084	30.861	**0.001**	0.016	0.023
	smits_mul_branchless_opt	0.409	3.915	39.920	0.001	0.010	0.019
	smitsint_mul_branchless_opt	0.419	4.058	40.319	0.001	**0.001**	0.019
	normrays_mul_branchless_opt	0.358	3.445	34.409	0.001	0.002	0.021
	normraysint_mul_branchless_opt	0.371	3.608	36.018	0.001	0.002	0.020

Table 1: *Benchmark results for clang (time in ms). More saturated green colors denote better mean runtimes and dark red worse variances. [clang++ 15.0.7 with compile flags* `-O3 -DNDEBUG -std=gnu++20 -MD -MT`*].*

		Mean runtime in ms			Runtime std. dev. in ms		
	AABBs per Ray	10	100	1000	10	100	1000
0% Hit Ratio	standard_div	1.095	9.125	84.129	0.002	0.005	0.018
	standard_mul	0.896	7.242	66.247	0.002	0.004	0.018
	pluecker	0.930	7.532	65.059	0.002	0.004	0.018
	plueckerint_div	0.941	7.605	65.627	0.002	0.004	0.019
	plueckerint_mul	0.938	7.555	65.245	0.002	0.004	0.032
	pluecker_cls	1.210	9.734	77.977	0.003	0.007	0.024
	pluecker_cls_cff	1.183	9.439	75.332	0.005	0.112	1.403
	plueckerint_div_cls	1.114	8.717	68.955	0.003	0.011	0.027
	plueckerint_div_cls_cff	1.199	9.361	75.871	0.003	0.008	0.037
	plueckerint_mul_cls	1.203	9.554	74.293	0.002	0.054	0.025
	plueckerint_mul_cls_cff	1.208	9.536	75.935	0.003	0.007	0.027
	slope	1.094	8.556	68.161	0.004	0.008	0.023
	slopeint_div	1.158	9.052	71.678	0.003	0.012	0.027
	slopeint_mul	1.116	8.709	69.126	0.003	0.014	0.029
	smits_div	1.059	8.984	82.973	0.002	0.004	0.019
	smits_div_cls	1.179	10.236	93.235	0.006	0.011	0.019
	smits_mul_cls	1.073	9.247	82.507	0.006	0.007	0.058
	smits_mul	0.821	6.682	61.320	0.002	0.004	**0.017**
	smitsint_mul	0.845	6.717	61.488	0.002	0.004	0.018
	smits_mul_branchless_opt	0.219	2.061	20.480	**0.001**	0.003	0.030
	smitsint_mul_branchless_opt	0.232	2.128	21.209	0.001	0.002	0.029
	normrays_mul_branchless_opt	**0.194**	**1.822**	**18.237**	0.001	**0.002**	0.033
	normraysint_mul_branchless_opt	0.219	2.046	20.450	0.001	0.002	0.031
50% Hit Ratio	standard_div	1.314	11.616	115.387	0.002	0.004	0.018
	standard_mul	1.028	8.628	85.009	0.002	0.003	**0.015**
	pluecker	0.895	7.942	82.330	0.001	**0.003**	0.040
	plueckerint_div	1.122	10.233	106.646	0.002	0.004	0.017
	plueckerint_mul	1.094	9.876	102.533	0.002	0.004	0.086
	pluecker_cls	1.009	8.908	80.664	0.003	0.005	0.022
	pluecker_cls_cff	0.933	8.212	79.716	0.004	0.087	1.022
	plueckerint_div_cls	1.232	10.582	99.985	0.004	0.011	0.030
	plueckerint_div_cls_cff	1.317	11.733	110.834	0.005	0.009	0.028
	plueckerint_mul_cls	1.244	11.008	102.053	0.003	0.006	0.357
	plueckerint_mul_cls_cff	1.252	11.164	105.526	0.004	0.009	0.030
	slope	0.997	8.413	75.391	0.003	0.006	0.021
	slopeint_div	1.170	10.508	102.897	0.003	0.010	0.028
	slopeint_mul	1.133	9.912	96.487	0.005	0.008	0.024
	smits_div	1.274	11.431	114.225	0.002	0.004	0.017
	smits_div_cls	1.168	10.694	106.516	0.006	0.012	0.024
	smits_mul_cls	1.089	9.439	90.388	0.005	0.027	0.675
	smits_mul	0.827	6.555	60.392	0.002	0.003	0.015
	smitsint_mul	0.974	8.141	80.316	0.002	0.004	0.016
	smits_mul_branchless_opt	0.231	2.062	20.473	0.001	0.003	0.030
	smitsint_mul_branchless_opt	0.232	2.131	21.205	0.001	0.003	0.029
	normrays_mul_branchless_opt	**0.194**	**1.824**	**18.240**	0.001	0.003	0.034
	normraysint_mul_branchless_opt	0.220	2.047	20.451	**0.001**	0.003	0.030
100% Hit Ratio	standard_div	0.617	4.336	41.331	0.001	**0.002**	0.018
	standard_mul	0.524	3.639	34.247	0.001	0.002	0.020
	pluecker	0.506	3.931	37.986	0.001	0.003	0.019
	plueckerint_div	0.953	8.461	91.168	0.004	0.033	0.029
	plueckerint_mul	0.893	7.818	83.376	0.003	0.021	0.026
	pluecker_cls	0.479	4.369	42.573	0.013	0.069	0.037
	pluecker_cls_cff	0.472	4.039	40.249	0.018	0.103	1.048
	plueckerint_div_cls	0.941	8.408	91.018	0.047	0.243	0.390
	plueckerint_div_cls_cff	0.972	9.118	100.294	0.024	0.142	1.060
	plueckerint_mul_cls	0.846	7.623	82.168	0.029	0.271	0.564
	plueckerint_mul_cls_cff	0.885	8.013	87.592	0.022	0.089	0.897
	slope	0.422	3.862	35.074	0.011	0.115	0.344
	slopeint_div	0.894	8.197	91.929	0.004	0.030	0.186
	slopeint_mul	0.833	7.663	80.923	0.005	0.104	0.044
	smits_div	0.595	4.301	41.249	0.002	0.003	**0.017**
	smits_div_cls	0.482	4.401	43.169	0.033	0.022	0.173
	smits_mul_cls	0.456	3.877	39.431	0.005	0.024	0.415
	smits_mul	0.450	3.011	28.388	0.001	0.002	0.023
	smitsint_mul	0.471	3.127	29.532	0.001	0.002	0.022
	smits_mul_branchless_opt	0.231	2.062	20.466	0.001	0.003	0.027
	smitsint_mul_branchless_opt	0.232	2.131	21.195	0.001	0.003	0.026
	normrays_mul_branchless_opt	**0.194**	**1.823**	**18.230**	0.001	0.003	0.031
	normraysint_mul_branchless_opt	0.220	2.047	20.443	**0.001**	0.003	0.029

Table 2: *Benchmark results for g++ (time in ms). More saturated green colors denote better mean runtimes and dark red worse variances. [g++ 12.3.0 with compile flags* `-O3 -DNDEBUG -std=gnu++20 -MD -MT`*].*

		Mean runtime in ms			Runtime std. dev. in ms		
	AABBs per Ray	10	100	1000	10	100	1000
0% Hit Ratio	standard_div	0.901	8.702	84.041	0.002	0.008	0.048
	standard_mul	0.769	7.199	69.442	0.002	0.004	**0.015**
	pluecker	0.889	7.113	61.536	0.002	0.003	0.042
	plueckerint_div	0.910	7.255	62.943	0.002	0.003	0.070
	plueckerint_mul	0.893	7.159	61.943	0.002	0.003	0.016
	pluecker_cls	1.096	8.765	72.394	0.002	0.006	0.016
	pluecker_cls_cff	1.096	8.492	72.424	0.004	0.006	0.053
	plueckerint_div_cls	0.971	7.749	65.820	0.003	0.013	0.056
	plueckerint_div_cls_cff	0.980	7.781	66.987	0.009	0.012	0.035
	plueckerint_mul_cls	0.968	7.739	66.464	0.006	0.015	0.155
	plueckerint_mul_cls_cff	0.967	7.748	66.352	0.008	0.008	0.029
	slope	0.993	7.886	67.090	0.004	0.018	0.039
	slopeint_div	1.014	8.142	68.645	0.007	0.016	0.059
	slopeint_mul	1.015	8.128	68.654	0.006	0.017	0.036
	smits_div	0.775	7.828	74.630	0.005	0.015	0.308
	smits_div_cls	1.182	10.326	91.555	0.005	0.013	0.026
	smits_mul_cls	1.113	9.387	80.828	0.007	0.023	0.064
	smits_mul	0.693	6.763	64.287	0.003	0.004	0.020
	smitsint_mul	0.698	6.712	63.753	0.004	0.005	0.023
	smits_mul_branchless_opt	0.196	1.816	18.219	0.001	0.003	0.027
	smitsint_mul_branchless_opt	0.205	1.925	19.276	**0.001**	0.003	0.026
	normrays_mul_branchless_opt	**0.180**	**1.659**	**16.641**	0.001	0.003	0.029
	normraysint_mul_branchless_opt	0.202	1.848	18.395	0.001	**0.002**	0.029
50% Hit Ratio	standard_div	1.043	10.638	107.260	0.003	0.004	0.021
	standard_mul	0.900	8.930	89.753	0.002	0.004	0.014
	pluecker	0.851	7.443	76.573	0.002	0.003	**0.010**
	plueckerint_div	1.254	11.141	111.219	0.002	0.005	0.066
	plueckerint_mul	1.184	10.530	105.914	0.002	0.005	0.025
	pluecker_cls	0.911	8.207	82.342	0.003	0.004	0.011
	pluecker_cls_cff	0.908	8.152	81.760	0.004	0.010	0.038
	plueckerint_div_cls	1.252	11.088	108.836	0.007	0.010	0.030
	plueckerint_div_cls_cff	1.267	11.322	111.251	0.009	0.011	0.028
	plueckerint_mul_cls	1.196	10.575	103.939	0.011	0.009	0.037
	plueckerint_mul_cls_cff	1.215	10.724	104.957	0.012	0.011	0.028
	slope	0.874	7.773	78.016	0.003	0.004	0.012
	slopeint_div	1.358	11.851	114.260	0.011	0.013	0.026
	slopeint_mul	1.298	11.353	109.454	0.010	0.014	0.029
	smits_div	0.955	9.983	101.001	0.004	0.011	0.115
	smits_div_cls	1.198	11.153	109.513	0.005	0.008	0.021
	smits_mul_cls	1.122	9.884	92.879	0.003	0.018	0.052
	smits_mul	0.845	8.401	84.457	0.002	0.003	0.012
	smitsint_mul	0.848	8.337	83.688	0.003	0.004	0.032
	smits_mul_branchless_opt	0.198	1.816	18.215	0.001	0.002	0.031
	smitsint_mul_branchless_opt	0.206	1.925	19.281	0.001	**0.002**	0.023
	normrays_mul_branchless_opt	**0.182**	**1.658**	**16.648**	0.001	0.002	0.027
	normraysint_mul_branchless_opt	0.204	1.845	18.405	**0.001**	0.003	0.024
100% Hit Ratio	standard_div	0.371	3.571	35.332	0.001	**0.002**	**0.015**
	standard_mul	0.322	3.031	29.798	0.001	0.002	0.017
	pluecker	0.520	4.224	41.028	0.002	0.009	0.015
	plueckerint_div	1.269	11.381	128.254	0.002	0.006	0.018
	plueckerint_mul	1.145	10.043	111.433	0.002	0.006	0.096
	pluecker_cls	0.500	4.796	49.466	0.003	0.109	0.485
	pluecker_cls_cff	0.490	4.653	48.075	0.003	0.057	0.443
	plueckerint_div_cls	1.264	11.240	120.444	0.012	0.013	0.825
	plueckerint_div_cls_cff	1.241	11.699	129.340	0.005	0.043	0.088
	plueckerint_mul_cls	1.105	9.955	110.167	0.004	0.066	0.121
	plueckerint_mul_cls_cff	1.186	10.506	110.642	0.025	0.040	0.087
	slope	0.510	4.576	47.478	0.004	0.069	0.709
	slopeint_div	1.335	12.238	133.297	0.007	0.021	0.301
	slopeint_mul	1.218	11.237	120.851	0.004	0.008	0.194
	smits_div	0.275	2.713	27.051	0.001	0.008	0.021
	smits_div_cls	0.551	3.562	33.636	0.021	0.017	0.092
	smits_mul_cls	0.480	3.317	32.816	0.031	0.077	0.213
	smits_mul	0.281	2.597	25.306	0.001	0.011	0.020
	smitsint_mul	0.285	2.708	26.684	0.001	0.003	0.016
	smits_mul_branchless_opt	0.198	1.816	18.223	0.001	0.003	0.024
	smitsint_mul_branchless_opt	0.206	1.925	19.277	**0.001**	0.003	0.025
	normrays_mul_branchless_opt	**0.182**	**1.658**	**16.646**	0.001	0.002	0.025
	normraysint_mul_branchless_opt	0.204	1.846	18.407	0.001	0.002	0.025

Table 3: *Benchmark results for clang (time in ms). More saturated green colors denote better mean runtimes and dark red worse variances. [clang++ 15.0.7 with compile flags* `-O3 -DNDEBUG -std=gnu++20 -MD -MT -march=native`*].*

		Mean runtime in ms			Runtime std. dev. in ms		
	AABBs per Ray	10	100	1000	10	100	1000
0% Hit Ratio	standard_div	0.971	9.289	89.594	0.002	0.005	0.016
	standard_mul	0.799	7.384	71.033	0.002	0.004	0.015
	pluecker	0.910	7.338	63.183	0.002	0.004	0.021
	plueckerint_div	0.915	7.325	63.238	0.002	0.004	0.017
	plueckerint_mul	0.925	7.443	64.099	0.002	0.004	**0.012**
	pluecker_cls	1.094	9.127	70.217	0.011	0.360	0.526
	pluecker_cls_cff	1.059	9.035	71.215	0.024	0.006	0.021
	plueckerint_div_cls	1.131	9.025	71.590	0.003	0.009	0.221
	plueckerint_div_cls_cff	1.156	8.965	71.680	0.004	0.020	0.843
	plueckerint_mul_cls	1.140	9.111	73.986	0.017	0.076	0.459
	plueckerint_mul_cls_cff	1.171	9.332	75.456	0.005	0.009	0.023
	slope	1.136	9.021	71.059	0.016	0.134	1.078
	slopeint_div	1.165	8.761	72.535	0.004	0.010	0.835
	slopeint_mul	1.167	9.209	74.037	0.015	0.109	2.269
	smits_div	0.913	9.097	87.428	0.005	0.005	0.019
	smits_div_cls	1.235	10.862	92.078	0.005	0.017	0.402
	smits_mul_cls	1.154	9.661	79.371	0.003	0.011	0.187
	smits_mul	0.718	6.982	66.514	0.004	0.005	0.017
	smitsint_mul	0.731	6.970	66.485	0.004	0.005	0.020
	smits_mul_branchless_opt	0.193	1.789	17.814	0.001	0.003	0.027
	smitsint_mul_branchless_opt	0.203	1.890	18.909	0.001	**0.003**	0.027
	normrays_mul_branchless_opt	**0.179**	**1.684**	**16.860**	**0.001**	0.003	0.027
	normraysint_mul_branchless_opt	0.203	1.896	18.964	0.001	0.003	0.026
50% Hit Ratio	standard_div	1.167	11.990	121.049	0.002	0.006	0.019
	standard_mul	0.920	9.118	91.835	0.002	0.003	**0.011**
	pluecker	0.865	7.692	79.532	0.001	0.003	0.015
	plueckerint_div	1.072	9.672	99.957	0.002	0.004	0.123
	plueckerint_mul	1.059	9.633	99.668	0.002	0.006	0.023
	pluecker_cls	0.931	7.845	76.692	0.001	0.012	0.213
	pluecker_cls_cff	0.874	7.611	73.817	0.003	0.010	0.045
	plueckerint_div_cls	1.114	9.947	99.940	0.006	0.019	0.163
	plueckerint_div_cls_cff	1.237	10.660	101.945	0.002	0.023	0.338
	plueckerint_mul_cls	1.128	10.104	98.746	0.010	0.051	0.049
	plueckerint_mul_cls_cff	1.167	10.351	99.987	0.002	0.008	0.082
	slope	0.979	9.043	78.834	0.003	0.035	0.621
	slopeint_div	1.168	10.203	100.197	0.005	0.187	0.068
	slopeint_mul	1.152	9.971	96.134	0.005	0.023	0.610
	smits_div	1.110	11.398	115.069	0.002	0.003	0.014
	smits_div_cls	1.266	11.914	111.956	0.004	0.008	0.200
	smits_mul_cls	1.124	9.760	88.621	0.005	0.225	0.208
	smits_mul	0.702	6.855	64.190	0.003	0.006	0.021
	smitsint_mul	0.865	8.611	86.498	0.002	0.003	0.012
	smits_mul_branchless_opt	0.194	1.786	17.815	0.001	0.003	0.027
	smitsint_mul_branchless_opt	0.202	1.891	18.916	0.001	0.002	0.022
	normrays_mul_branchless_opt	**0.179**	**1.684**	**16.868**	**0.001**	0.003	0.030
	normraysint_mul_branchless_opt	0.203	1.897	18.967	0.001	**0.002**	0.028
100% Hit Ratio	standard_div	0.433	4.247	42.489	0.001	0.002	0.015
	standard_mul	0.326	3.061	30.517	0.001	0.002	0.017
	pluecker	0.490	3.723	35.737	0.002	0.007	0.015
	plueckerint_div	0.888	7.761	84.274	0.002	0.014	0.133
	plueckerint_mul	0.831	7.235	77.920	0.003	0.013	0.125
	pluecker_cls	0.416	3.592	35.545	0.002	0.033	0.256
	pluecker_cls_cff	0.405	3.539	35.646	0.002	0.036	0.349
	plueckerint_div_cls	0.871	7.851	86.887	0.005	0.028	0.202
	plueckerint_div_cls_cff	0.865	8.149	87.528	0.026	0.175	0.137
	plueckerint_mul_cls	0.826	7.374	78.620	0.024	0.098	0.852
	plueckerint_mul_cls_cff	0.788	7.510	77.991	0.008	0.251	0.351
	slope	0.485	4.012	34.986	0.049	0.168	0.341
	slopeint_div	0.891	8.173	88.363	0.012	0.154	0.136
	slopeint_mul	0.870	7.242	78.253	0.022	0.128	0.837
	smits_div	0.428	4.229	42.351	0.001	**0.001**	**0.013**
	smits_div_cls	0.459	4.232	41.715	0.011	0.012	0.028
	smits_mul_cls	0.521	4.035	38.596	0.033	0.055	0.148
	smits_mul	0.289	2.708	27.017	0.001	0.002	0.021
	smitsint_mul	0.296	2.696	26.808	0.001	0.002	0.025
	smits_mul_branchless_opt	0.194	1.786	17.811	**0.001**	0.003	0.026
	smitsint_mul_branchless_opt	0.202	1.891	18.921	0.001	0.002	0.026
	normrays_mul_branchless_opt	**0.179**	**1.684**	**16.866**	0.001	0.003	0.026
	normraysint_mul_branchless_opt	0.203	1.897	18.971	0.001	0.002	0.026

Table 4: *Benchmark results for g++ (time in ms). More saturated green colors denote better mean runtimes and dark red worse variances. [g++ 12.3.0 with compile flags* `-O3 -DNDEBUG -std=gnu++20 -MD -MT -march=native`*].*

```cpp
// --- ray data structures
struct standard_ray {
  float x, y, z;    // ray origin
  float i, j, k;    // ray direction
  float ii, ij, ik; // inverses of direction components
  float tn, tf;       // ray bounds
};
struct norm_opt_ray {
    norm_opt_ray(const standard_ray& r); // Conversion ctor
    float rd[2];
    float nrdo[2]; // precomputed -o * rd terms
    float tn, tf;
    std::uint32_t bnx0, bfx0, bnx1, bfx1, bnx2, bfx2; // load
      offsets
};
struct norm_opt_int_ray {
    // [...] same as norm_opt_ray
    std::uint32_t tret; // load offset [0, 1] for returning the
      correct intersection distance
};
// --- safe reciprocal for preventing NaNs
inline float safe_rcp(float x) {
    return 1.0f / (std::abs(x) > 1e-8f ? x : std::copysign(1e-8f,
      x));
}
// --- construction of norm_opt_ray
norm_opt_ray::norm_opt_ray(const standard_ray& ray) {
    // dominant axis
  const auto [dom_ax, dom_d] = get_dominant_direction(ray);
  // determine component permutation, such that the dominant
      component is last (x2)
  constexpr std::uint32_t cperm[3][3] = {
    {1u, 2u, 0u},   // x
    {2u, 0u, 1u},   // y
    {0u, 1u, 2u}  // z
  };
  const auto [x0, x1, x2] = cperm[dom_ax];
  // ray quantities in array form for swizzling
  const float ray_o[]   = {ray.x,   ray.y,  ray.z};
  const float ray_d[]   = {ray.i,   ray.j,  ray.k};
  const float ray_rd[]  = {ray.ii,  ray.ij, ray.ik};
  // direction (only temporary, used to prevent NaNs in nrdo
      calculation)
  const float d[] = {ray_d[x0] / dom_d, ray_d[x1] / dom_d};
  // reciprocal direction (rd[x2] is implicitly 1.0)
  rd[0] = ray_rd[x0] * dom_d;
  rd[1] = ray_rd[x1] * dom_d;
  // transformed origin (o[x2] is implicitly 0.0)
  const float to = -ray_o[x2] * ray_rd[x2];
  const float o[] = { ray_o[x0] + to * ray_d[x0], ray_o[x1] + to *
       ray_d[x1] };
  // tn, tf
  tn = ray_o[x2] + ray.tn * ray_d[x2];
  tf = ray_o[x2] + ray.tf * ray_d[x2];
  // swap if ray direction was flipped
  const std::uint32_t dom_sign = static_cast<std::uint32_t>(std::
      signbit(dom_d));
  if (dom_sign) std::swap(tn, tf);
  // precomputed -rd * o terms (nrdo[x2] is implicitly 0.0)
  nrdo[0] = -o[0] * safe_rcp(d[0]);
  nrdo[1] = -o[1] * safe_rcp(d[1]);
  // aabb load offsets
  const std::uint32_t dsign[] = { std::signbit(ray_d[x0]) ^
      dom_sign, std::signbit(ray_d[x1]) ^ dom_sign };
  bnx0 = x0 << 1u ^ dsign[0];
  bnx1 = x1 << 1u ^ dsign[1];
  bnx2 = x2 << 1u;
  bfx0 = bnx0 ^ 1u;
  bfx1 = bnx1 ^ 1u;
  bfx2 = bnx2 ^ 1u;
}
// --- construction of norm_opt_int_ray
norm_opt_int_ray::norm_opt_int_ray(const standard_ray& ray) {
    // [...] same as above
    // tret load offset
  tret = dom_sign ? 0xFFFFFFFFu : 0u; // bound selection mask
}
```

Listing 1: *Implementation of our ray normalization scheme.*