Vol. 9, No. 1: 35–46

Fast Ray-Axis Aligned Bounding Box Overlap Tests with Plücker Coordinates

Jeffrey Mahovsky and Brian Wyvill
University of Calgary

Abstract. Fast ray-axis aligned bounding box overlap tests can be performed by utilizing Plücker coordinates. This method tests the ray against the edges comprising the silhouette of the box instead of testing against individual faces. Projection of the edges onto a two-dimensional plane to generate the silhouette is not necessary, which simplifies the technique. The method is division-free and successive calculations are independent and consist simply of dot product operations, which permits vectorization. The method does not compute an intersection distance along the ray to the box, but this can be added as an additional step. Storage of Plücker coordinates is unnecessary, permitting integration into existing systems. Test results show the technique's performance is up to 93% faster than traditional methods if an intersection distance is not needed.

1. Introduction

Ray-Axis Aligned Bounding Box (AABB) intersection tests are commonly used in ray tracing acceleration schemes, such as bounding volume hierarchies [Goldsmith and Salmon 87]. It is important that these tests be as fast as possible, since millions of them may be needed to generate an image.

The most popular ray-AABB test used today (termed the "standard" test) computes the distances from the ray origin, along the ray, to each of the six planes that define the box [Haines 89]. Three intervals are formed—one for the two planes perpendicular to the x-axis, one for the y-axis, and one for the z-axis. These three intervals must overlap, or the ray does not intersect the box. By exploiting the properties of the IEEE standard floating-point arithmetic, the algorithm can be simplified and still produce correct results [Smits 98] (termed the "smits" test).

Plücker coordinates have been previously used in computer graphics [Teller and Hohmeyer 99], [Mann and Dorst 02], [Bell 45], [Shoemake 98]. We present a new Plücker coordinate-based algorithm to determine the overlap between a ray and an AABB. This algorithm tests the ray against the silhouette of the AABB, instead of testing against individual faces of the box or comparing intersection intervals. The test is performed using only dot products and comparisons (in addition to this, the "smits" test requires division). Its computational simplicity results in excellent performance.

The idea of intersecting with the silhouette of the box was also proposed in [Haines 01]. However, that technique required projecting the box faces onto a two-dimensional plane perpendicular to the ray before performing the ray-silhouette intersection test. The Plücker-based test does not require that the edges be projected onto a two-dimensional plane.

Sample source code is available at the web site listed at the end of the paper.

2. Plücker Coordinates

A line L passing through two points A and B in three-space with coordinates (A_x, A_y, A_z) and (B_x, B_y, B_z) can be expressed as six Plücker coordinates $L_0 \ldots L_5$:

$$\begin{aligned}
L_0 &= A_x B_y - B_x A_y \\
L_1 &= A_x B_z - B_x A_z \\
L_2 &= A_x - B_x \\
L_3 &= A_y B_z - B_y A_z \\
L_4 &= A_z - B_z \\
L_5 &= B_y - A_y \,.
\end{aligned}$$

A ray R can also be expressed as Plücker coordinates $R_0 \ldots R_5$. The first point in space is the ray origin O with coordinates (O_x, O_y, O_z). The second point is the origin O plus the direction vector $\vec{D}$ with components (D_i, D_j, D_k), giving coordinates $(O_x + D_i, O_y + D_j, O_z + D_k)$. Substituting into the Plücker coefficient equations gives

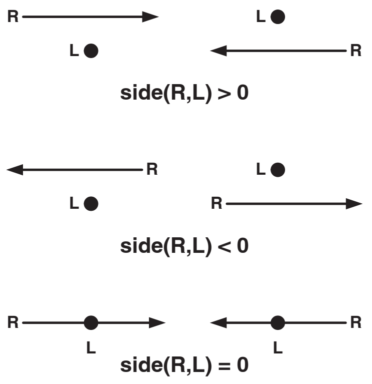

Figure 1. Side relation and relative orientation of lines. L is perpendicular to the page and points toward the reader. *side*(R, L) is positive if the relative orientation is clockwise (top), negative if counterclockwise (middle), and zero if the lines intersect.

$$\begin{aligned}
R_0 &= O_x D_j - D_i O_y \\
R_1 &= O_x D_k - D_i O_z \\
R_2 &= -D_i \\
R_3 &= O_y D_k - D_j O_z \\
R_4 &= -D_k \\
R_5 &= D_j \,.
\end{aligned}$$

An important relation is $side(R, L)$, which provides information about the relative orientation of the ray and the line (see Figure 1).

An expression can be derived for $side(R, L)$ as the permuted inner product of the ray's Plücker coordinates R_n, and line's Plücker coordinates L_n [Yamaguchi and Niizeki 97], [Bloomenthal and Rokne 94]. Given a line L passing through points A and B, and a ray R with origin O and direction vector $\vec{D}$, we wish to determine the relative orientation of the ray and the line (see Figure 2). Using a cross product, compute the surface normal $\vec{N}$ of the plane that contains the three points A, B, and O:

$$\vec{N} = (A - O) \times (B - O)\,.$$

The relative orientation of the ray and line (or side relation) is determined by the sign of the dot product of $\vec{D}$ and $\vec{N}$:

$$side(R, L) = -(\vec{D} \cdot \vec{N})\,.$$

Hence,

$$side(R, L) = -\vec{D} \cdot ((A - O) \times (B - O))\,.$$

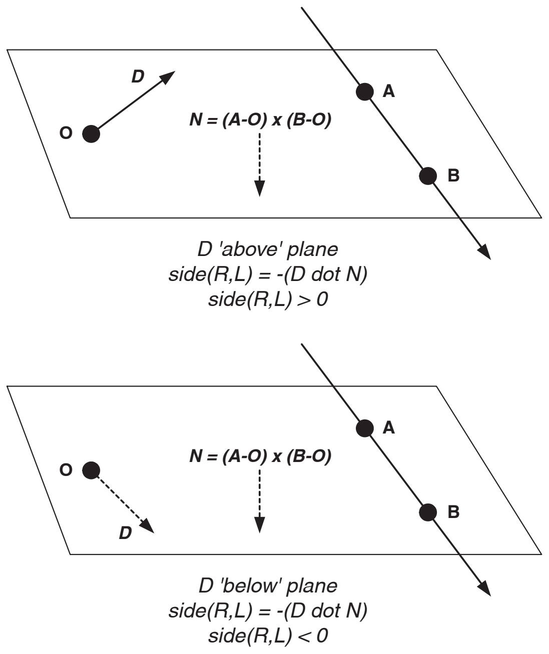

Figure 2. The side relation.

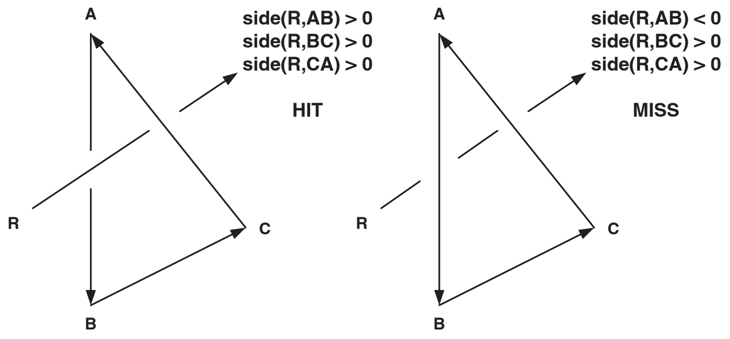

Figure 3. Ray-polygon intersection with Plücker coordinates.

Expanding and simplifying, the side relations are obtained:

$$\begin{aligned}
side(R,L) &= -D_i((A_y-O_y)(B_z-O_z)-(A_z-O_z)(B_y-O_y)) \\
&\quad -D_j((A_z-O_z)(B_x-O_x)-(A_x-O_x)(B_z-O_z)) \\
&\quad -D_k((A_x-O_x)(B_y-O_y)-(A_y-O_y)(B_x-O_x)) \\
side(R,L) &= -D_i(A_yB_z-B_yA_z)+D_j(A_xB_z-B_xA_z) \\
&\quad -D_k(A_xB_y-B_xA_y)+(O_xD_k-D_iO_z)(B_y-A_y) \\
&\quad +(O_xD_j-D_iO_y)(A_z-B_z)+(O_yD_k-D_jO_z)(A_x-B_x)\,.
\end{aligned}$$

Substituting back in the definitions of the Plücker coordinates, we get the permuted inner product:

$$side(R,L) = R_2L_3 + R_5L_1 + R_4L_0 + R_1L_5 + R_0L_4 + R_3L_2\,.$$

The need to store R_2, R_4, and R_5 can be removed by substituting for $-D_i$, D_j, $-D_k$:

$$side(R,L) = -D_iL_3 + D_jL_1 - D_kL_0 + R_1L_5 + R_0L_4 + R_3L_2\,.$$

Plücker coordinates can also be used for ray-convex polygon intersection tests (see Figure 3). If the polygon vertices are defined counterclockwise, then the side relations for the ray and each of the edges must be positive or zero for the ray to intersect the front side of the polygon [Amanatides and Choi 97]. Alternatively, one of the side relations must be negative for the ray to miss the polygon. (If all the side relations are negative or zero, the back side of the polygon is intersected, but only front-facing intersections are desired in this case so any negatives can be treated as a miss.) It is important to note that the distance along the ray where the intersection occurred isn't produced. This must be computed separately.

3. Axis-Aligned Box Intersection

The Plücker-convex polygon test may be applied to the problem of determining an intersection with an axis-aligned bounding box. Recall that an axis-aligned bounding box has faces perpendicular to the x, y, and z axes (see Figure 4).

Normally, each face of the box (or planes of the faces, when using the "standard" or "smits" methods) would be tested to determine if the ray passes through one of them, and hence intersects the box. The number of faces tested can be reduced from six to three by classifying the ray based on the signs of its direction vector components (D_i, D_j, D_k). For example, a ray with all three

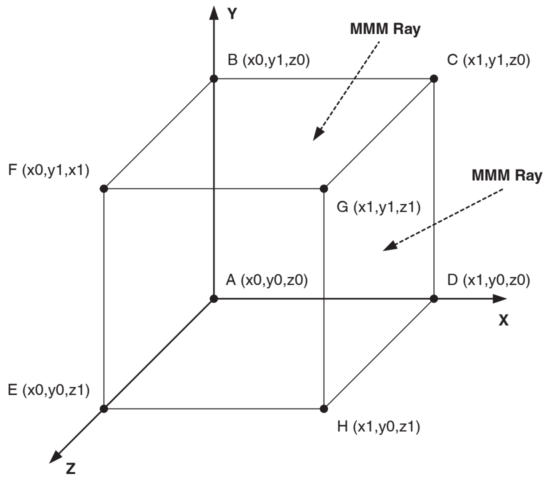

Figure 4. Axis-aligned bounding box and MMM rays.

components negative (classified MMM, for “minus minus minus”) will pass through one of the three faces $FEHG$, $CGHD$, or $BFGC$ if it intersects the box. The other seven ray classes are MMP, MPM, MPP, PMM, PMP, PPM, and PPP with the letters M and P corresponding to the signs of the D_i, D_j, and D_k components of the ray. Each ray class is tested against a different set of three box faces. The ray class can be determined when the ray is created and stored as a three-bit value within the ray.

To determine if a ray intersects the box, the side relation of the ray and several of its edges must be computed. The Plücker coefficients for the box edges are simpler than those of an arbitrary convex polygon due to the axis-aligned nature of the faces. When computing the coefficients, the edges may be treated either as rays or as lines passing through two points in space. Equivalent results are produced with either method, the ray form is used here as it results in a shorter derivation.

For example, the coefficients for edge FE with origin (x_0, y_1, z_1) and direction $(0, -1, 0)$ are

$$\begin{aligned}
FE_0 &= -x_0 \\
FE_1 &= 0 \\
FE_2 &= 0 \\
FE_3 &= z_1 \\
FE_4 &= 0 \\
FE_5 &= -1\,.
\end{aligned}$$

Consequently, the side relation of R and edge FE is

$$side(R, FE) = -R_1 - D_i z_1 + D_k x_0 \,.$$

Instead of testing each of the four edges per front-facing box face (maximum 12 tests), the test is optimized so that only the edges of the silhouette of the box are tested. Thus, only a maximum of six tests are needed. For example, an MMM ray is tested against box edges BF, FE, EH, HD, DC, and CB.

It does not matter that the six edges are not within the same plane, as the convexity of the silhouette and counterclockwise ordering of the edges is preserved as long as the ray is MMM. A different set of edges is evaluated for each of the other seven ray classes (see Figure 4).

Hence, for an MMM ray, the algorithm is as follows:

```
if(side(R,DH) > 0) then MISS
else if(side(R,BF) < 0) then MISS
else if(side(R,EF) > 0) then MISS
else if(side(R,DC) < 0) then MISS
else if(side(R,BC) > 0) then MISS
else if(side(R,EH) < 0) then MISS
else HIT .
```

Depending on the directions of the edges, some of the inequalities may be reversed. For example, the side relation and inequality $side(R, EF) > 0$ can be rewritten as $side(R, FE) < 0$.

A separate test is needed to determine if the box is on the positive portion of the ray. The Plücker tests assume the ray is infinitely long in both directions. This can be determined from the ray origin. For an MMM ray, test that the ray origin $O_x > x_0, O_y > y_0$, and $O_z > z_0$. The ray is still assumed to be infinitely long in the positive direction, but a similar additional test could be used for a truncated ray if the x, y, and z coordinates of the ray's endpoint are known.

The logic for the seven other cases is not difficult to derive. For convenience, the logic is implemented in the source code available online at the web site listed at the end of the paper.

Computation and storage of the ray's Plücker coordinates R_0, R_1, and R_3 can be eliminated. Before each ray-box test, subtract the ray's origin O from the box's edge coordinates, producing new temporary box coordinates used for the intersection test. The ray's origin is now assumed to be $(0, 0, 0)$, which makes the ray's R_0, R_1, and R_3 Plücker coordinates equal to zero in the side relation equations. Hence, no Plücker coordinates need to be computed or stored to use this technique despite it being based on Plücker coordinates. This permits integration into existing ray tracing systems. See the source code available online for complete details of this optimization.

The Plücker technique does not, by itself, compute an intersection distance. This is easily added by computing the distance along the ray to the three closest planes of the box, once an intersection has been confirmed. The three closest planes are already known from the ray classification. The correct distance is the largest of the three distances. The distances to the planes can be computed with either division or by multiplication with precomputed ray direction vector component inverses (see Section 4).

The Plücker technique has several useful characteristics. The technique is free of division operations, unless an intersection distance is needed. Even then, division can be avoided by multiplying by ray direction vector component inverses. (The astute reader will argue that the traditional methods can also become division-free by multiplying by precomputed inverses. The difference is that the Plücker technique does not require precomputed inverses to avoid division, provided that an intersection point is unnecessary.) Also, computation of the six side relations are independent of each other, allowing them to be computed in parallel. Computation of a side relation is simply a three-term dot product as the other three coefficients are 0. This algorithmic simplicity should permit vectorization of the Plücker code, potentially improving performance substantially. This could be implemented using Intel's SSE [Intel Corporation 02] vector instructions, available on all Pentium III and Pentium 4 processors.

4. Test Results and Conclusions

The Plücker method of ray-box intersection was compared to two other methods experimentally; the "standard" method and the "smits" method. Performance tests were run under two different types of system: a 2533 MHz Pentium 4 running Microsoft Windows XP using the Visual C++ .NET 2003 compiler, and a 733 MHz Pentium-III running Red Hat Linux 7.3 using the gcc 2.96 compiler. Optimization was set to "/Ox" or "-O3" and no special settings were used and no vectorization of the Plücker calculations was performed (this remains to be explored). Significant speed-up may be possible, but effort should also be made to vectorize the other algorithms as well, if possible.

To test the algorithms, 500,000 pairs of random rays and boxes were generated beforehand and stored in arrays. For each test, each ray/box pair was tested for intersection 100 times, for a total of 50 million intersection tests. The percentage of ray/box hits was varied between 0%, 50%, and 100%. This was done by predetermining which ray/box pairs intersected and then storing the appropriate mixture in the ray/box arrays. The same set of ray/box pairs was used for each test, provided the hit percentages were the same. Tests were performed with both single- and double-precision floating-point arithmetic.

Additionally, the correctness of the Plücker technique was verified by comparing its results to those of the "standard" and "smits" tests for all 500,000 ray/box pairs.

Tables 1 and 2 provide details of the results. The results are broken down into several types of tests:

- *pluecker*: Determines whether the ray hits the box, does not compute an intersection distance. Ensures the box is on the forward extension of the ray.

- *plueckerint*: Determines whether the ray hits the box and computes an intersection distance. Ensures the box is on the forward extension of the ray.

- *standard*: The "standard" test. Determines whether the ray hits the box and computes an intersection distance. Explicitly checks for ray direction components equaling zero to avoid +/- infinity values.

- *smits*: The "smits" test. Determines whether the ray hits the box and computes an intersection distance. Does not check for zero ray direction components; instead relies on IEEE arithmetic to correctly handle the $+/-$ infinity values.

Several variations of the methods were devised in order to test some optimization heuristics. These designations were appended to the test type above to distinguish between them.

- *cls*: Uses precomputed ray classifications (MMM, etc.) that must be computed once for each ray and carried in the ray. If absent, the ray classification is determined within the intersection test and is performed "on-the-fly" each time.

- cff: Uses three precomputed Plücker coordinates R_0, R_1, and R_3 that must be computed once for each ray and carried in the ray. If absent, the ray's origin is subtracted from the box's edge coordinates as described in Section 3.

- *div*: Uses division to compute the ray-box intersection distance, or ray-plane intersection distances.

- *mul*: Uses multiplicative inverses of the ray direction vector components to compute the ray-box intersection distance, or ray-plane intersection distances. These must be computed once for each ray and carried within the ray.

	DOUBLE PRECISION			SINGLE PRECISION		
	hit/miss			hit/miss		
	0M/50M	25M/25M	50M/0M	0M/50M	25M/25M	50M/0M
pluecker:	4.6	5.0	5.3	3.2	3.6	4.0
pluecker-cls:	3.8	4.0	4.1	2.7	3.0	3.4
pluecker-cls-cff:	3.6	3.8	3.9	2.6	2.8	2.9
plueckerint-div:	4.7	5.8	7.0	3.2	5.0	6.8
plueckerint-div-cls:	3.8	5.1	6.4	2.7	4.4	6.1
plueckerint-div-cls-cff:	3.7	5.0	6.2	2.6	4.3	5.9
plueckerint-mul:	4.6	5.5	6.2	3.2	4.2	5.1
plueckerint-mul-cls:	3.8	4.5	5.1	2.7	3.6	4.5
plueckerint-mul-cls-cff:	3.7	4.3	4.9	2.7	3.4	4.2
standard-div:	7.3	8.9	10.5	7.0	8.8	10.6
standard-mul:	5.9	7.0	8.1	5.1	6.3	7.5
smits-div:	7.2	8.8	10.4	6.8	8.5	10.2
smits-div-cls:	6.3	7.3	8.4	5.4	6.6	7.7
smits-mul:	5.5	6.6	7.6	4.8	6.0	7.1
smits-mul-cls:	4.7	5.3	5.8	4.1	4.9	5.6

Table 1. Ray-AABB performance on Pentium 4 2533MHz (WinXP, VC++ .NET 2003). Times are measured in seconds; lower is faster.

	DOUBLE PRECISION			SINGLE PRECISION		
	hit/miss			hit/miss		
	0M/50M	25M/25M	50M/0M	0M/50M	25M/25M	50M/0M
pluecker:	28.7	32.0	35.2	20.1	22.5	24.9
pluecker-cls:	19.4	24.8	30.4	15.0	18.4	21.7
pluecker-cls-cff:	19.6	25.6	31.5	14.7	18.1	21.6
plueckerint-div:	29.3	37.3	45.5	20.3	27.6	35.1
plueckerint-div-cls:	20.0	30.0	40.1	15.3	23.5	31.7
plueckerint-div-cls-cff:	19.5	30.7	41.8	15.0	23.5	32.1
plueckerint-mul:	29.0	35.2	41.4	20.1	24.0	28.0
plueckerint-mul-cls:	19.7	27.6	35.5	15.4	20.2	24.9
plueckerint-mul-cls-cff:	20.1	28.6	37.0	15.0	20.2	25.4
standard-div:	32.3	39.7	47.1	27.5	32.9	38.4
standard-mul:	26.3	30.6	35.0	20.3	23.0	25.8
smits-div:	33.1	39.3	45.4	27.0	32.0	37.1
smits-div-cls:	30.9	36.9	42.9	25.1	29.7	34.3
smits-mul:	24.4	28.4	32.4	19.3	21.8	24.2
smits-mul-cls:	22.7	26.4	30.3	18.1	20.1	22.2

Table 2. Ray-AABB performance on Pentium III 733MHz (Red Hat Linux 7.3, gcc 2.96). Times are measured in seconds; lower is faster.

It is important that the reader make "apples to apples" comparisons when examining the results. For example, it is not completely fair to compare a Plücker test that uses division to compute an intersection distance to a "smits" test that uses multiplication. Instead, compare the tests that use division to each other, and so on.

In general, the Plücker method performs significantly better than the other methods, provided an intersection distance is not needed. The fastest Plücker methods ($pluecker - cls$ and $pluecker - cls - cff$) are faster than the fastest traditional method ($smits - mul - cls$) in 11 out of 12 tests. Performance is 93% faster in one case (2.9 seconds versus 5.6 seconds). Computation and storage of ray Plücker coordinates R_0, R_1, and R_3 is not required for good performance as the results for $pluecker - cls$ and $pluecker - cls - cff$ are very similar.

If an intersection distance is needed, there is less advantage to using the Plücker method. It is still significantly faster on the Pentium 4, but the advantage is less clear on the Pentium III.

An interesting observation is that all the methods benefited significantly from ray classification and from multiplying by inverses instead of dividing. Note that "standard" was not tested with ray classification due to its similarity to "smits."

We have also implemented the Plücker method in a real ray tracer and compared results to the standard ray bounding box method. We consistently observed an improvement of between 2 and 2.5 times in rendering speed. We used C# for this implementation, rather than C++ so the speed-up is not directly comparable to the results shown above.

Acknowledgments. This work is supported in part by grants from the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Alberta Informatics Circle of Research Excellence (iCORE). We would also like to thank the students and staff who helped us in the Graphics Jungle and the Computer Science Department at the University of Calgary.

References

[Amanatides and Choi 97] John Amanatides and Kia Choi. "Ray Tracing Triangular Meshes." In *Proceedings of the Eighth Western Computer Graphics Symposium*, edited by Valerie Summers, Alain Fournier, and Kelly Booth, pp. 43–52. Vancouver: The University of British Columbia, 1997.

[Bell 45] E. T. Bell. *The Development of Mathematics*, Second edition. New York: McGraw-Hill Book Company, 1945.

[Bloomenthal and Rokne 94] Jules Bloomenthal and Jon Rokne. “Homogeneous Coordinates.” *The Visual Computer* 11 (1994), 15–26.

[Intel Corporation 02] Intel Corporation. “IA-32 Intel Architecture Software Developer’s Manual, Instruction Set Reference.” *Intel Corporation* 2 (2002).

[Goldsmith and Salmon 87] J. Goldsmith and J. Salmon. “Automatic Creation of Object Hierarchies for Ray Tracing.” *IEEE Computer Graphics and Applications* 7:5 (1987), 14–20.

[Haines 89] Eric A. Haines. “Essential Ray Tracing Algorithms.” In *An Introduction to Ray Tracing*, edited by Andrew Glassner. San Francisco: Morgan Kaufmann, 1989.

[Haines 01] Eric A. Haines. “Bounding Box Intersection via Origin Location.” *Ray Tracing News* 14:1 (2001).

[Mann and Dorst 02] Stephen Mann and Leo Dorst. “Geometric Algebra: A Computational Framework for Geometrical Applications (Part 2).” *IEEE Computer Graphics and Applications* 22:4 (2002), 58–67.

[Shoemake 98] Ken Shoemake. “Plücker Coordinate Tutorial.” *Ray Tracing News* 11:1 (1998).

[Smits 98] Brian Smits. “Efficiency Issues for Ray Tracing.” *journal of graphics tools* 3:2 (1998), 1–14.

[Teller and Hohmeyer 99] Seth Teller and Michael Hohmeyer. “Determining the Lines Through Four Lines.” *journal of graphics tools* 4:3 (1999), 11–22.

[Yamaguchi and Niizeki 97] Fujio Yamaguchi and Masatoshi Niizeki. “Some Basic Geometric Test Conditions in Terms of Plücker Coordinates and Plücker Coefficients.” *The Visual Computer* 13:1 (1997), 29–41.

Web Information:

Sample C++ source code implementing the algorithm and the performance tests is available online at http://www.acm.org/jgt/MahovskyWyvill04/.

Jeffrey Mahovsky, Department of Computer Science, University of Calgary,
2500 University Drive NW, Calgary, Alberta T2N 1N4 (mahovskj@cpsc.ucalgary.ca)

Brian Wyvill, Department of Computer Science, University of Calgary,
2500 University Drive NW, Calgary, Alberta T2N 1N4 (blob@cpsc.ucalgary.ca)

Received June 5, 2003; accepted in revised form December 1, 2003.