kd-Tree Traversal Techniques

John A. Tsakok∗
William Bishop†
Andrew Kennings‡

Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada

1 Omni-directional Ray Bundles

The following two modifications must be done to the traditional kd-tree, ray bundle traversal in order to allow rays of differing direction signs per axis to be traced together:

1. Independent of ray directions, always trace from front voxel to back voxel which is defined as the voxel on the same and opposite side of the split plane as the camera, respectively

2. Treat negative intersection distances as large (infinite) positive distances

The following is an example of a $2 \times 2$ bundle traversal code which has been modified to handle rays with no direction restriction:

```c
while ( !node.isLeaf ) {
    active[i] = ( t[i][near[i] < t[i][far[i] ];
    if ( for all i=0..3(!active[i]) ) break;
    dist = split - origin[axis];
    d[i] = dist / dir[i][axis];
    for all i=0..3 d[i] = (d[i] < 0 ? FLT_MAX : d[i]);
    int node_index = ( dist < 0.0f ) ? 1 : 0;
    Node *front = ( KDTreeNode * ) ( node.left + ( node_index ? 0x1 ) );
    Node *back = ( KDTreeNode * ) ( node.left + node_index );
    stack.push( back, max( d[i], t[i][near[i] ), t[i][far[i] ] );
    ( node, t[i][far[i] ) = ( front, min( d[i], t[i][far[i] ) );
}
```

By using this modification, direction signs no longer have to be checked per bundle which can be helpful for a fixed-function hardware implementation.

2 Cone Traversal Algorithm

Cone proxies provide an alternative to a pyramidal proxy for ray traversal acceleration. When tracing multiple samples to a spherical light source for soft shadow rendering, cone proxies provide a tighter bound than pyramids to internal rays and thus yield better performance.

Figure 1 shows a cone with origin $s$, direction $v$ and angle $\alpha$ intersecting a split plane with normal $n$, distance $d$ at an angle $\phi$.

Figure 1: a cone with origin $s$, direction $v$ and angle $\alpha$ intersecting a split plane with normal $n$, distance $d$ at an angle $\phi$.

$r1$ and $r2$ along vector $v$ from $s$ in which the cone intersects the split plane at a point is shown:

$$t = \frac{d - n \cdot s \tan \alpha}{\sqrt{1 - (n \cdot v || n \cdot v)^2}}$$

The following shows the traversal algorithm for cones where $\text{denom}$ are precalculated values for the denominator in the above equation:

```c
while ( !node.isLeaf ) {
    split_minus_o = split - origin[axis];
    temp1 = split_minus_o + denom[0][axis];
    temp2 = split_minus_o + denom[1][axis];
    temp1 = (temp1 < 0 ) ? FLT_MAX : temp1;
    temp2 = (temp2 < 0 ) ? FLT_MAX : temp2;
    t1 = min(temp1, temp2);
    t2 = max(temp1, temp2);
    if ( t2 <= t[i][near] )
        node = Back( node );
    else if ( t1 >= t[i][far] )
        node = Front( node );
    else {
        stack.push( Back( node ), max( t[i][near], t[i][far] );
        ( node, t[i][far] ) = ( Front( node ), min( t[i][far] ) );
    }
```