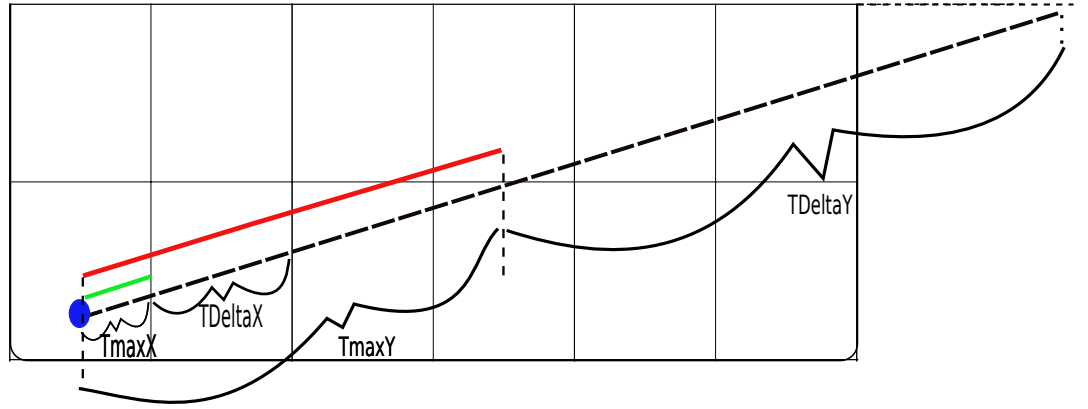


# Ray Marching Algorithm

## Cubic Cells

This section demonstrates the procedure of the ray marching algorithm in determining the piecewise path taken by a ray as it moves through the computational domain. For simplicity, this description will be carried out in two dimensions, although the same principles are applied in the three dimensional cases.

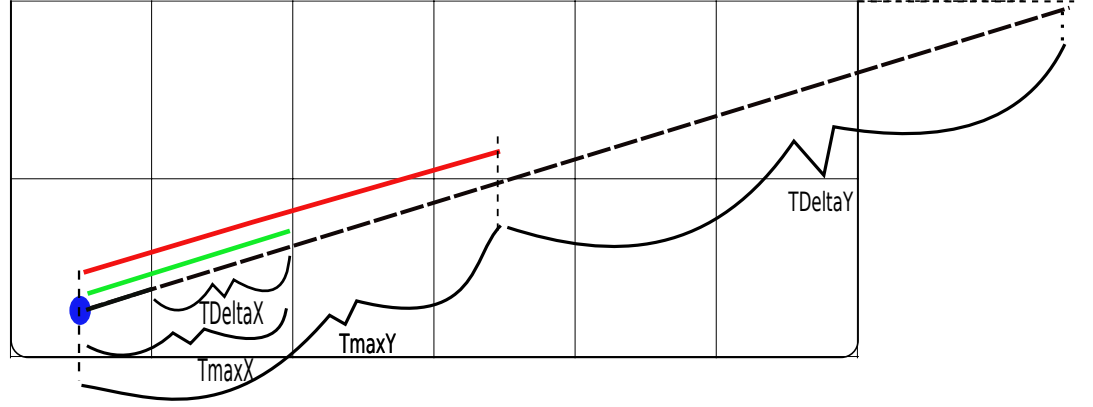
Let us begin with a small computational domain with two rows of cells, and six columns of cells, indexed as  $(i,j)$ , where  $i$  represents the row number starting from the bottom of the domain, and  $j$  represents the column number starting from the left of the domain. Let the vertical lines represent  $x$  planes, and the horizontal lines represent  $y$  planes. We wish to trace a ray within cell  $(1,1)$  from the ray origin indicated by the blue circle in the below figure. Assume that a ray direction has already been determined, and will be represented by the long dashed line below.



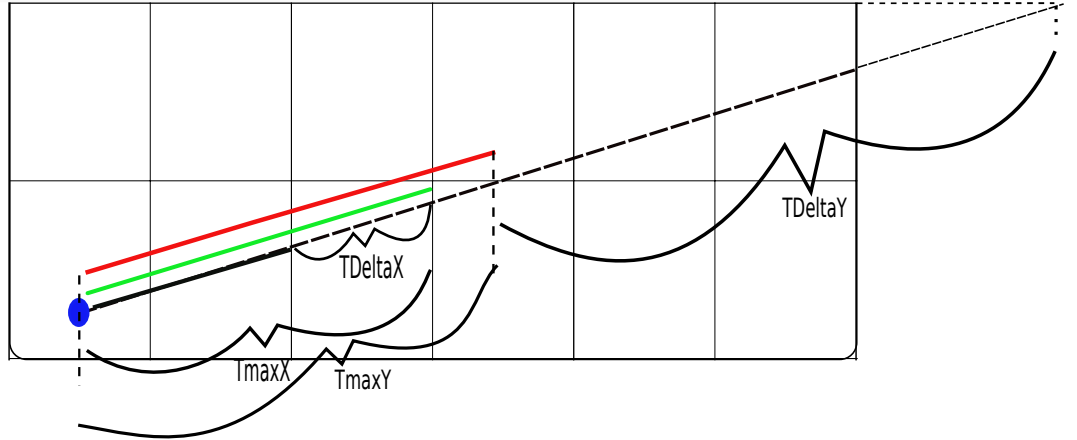
We must first determine the length of the ray segment from the origin to the first cell wall that is breached by the ray. But at this time, we do not know if it will be an  $x$  plane or a  $y$  plane that will first be breached by the ray. To determine which plane will be breached (and subsequently determine the next cell that the ray will enter) we compare the distance from the origin to the first  $x$  plane,  $T_{maxX}$  (green), to the distance from the origin to the first  $y$  plane,  $T_{maxY}$  (red) in the direction of the ray. In two dimensions, this is accomplished by a simple “if/else” statement (additional comparisons are necessary in three dimensions).

Once the shortest of the two distances is determined, the current cell is updated by use of the step variable. In this case, the shortest direction is  $T_{maxX}$ , so we step in the  $x$  direction. The distance traveled (in this case the green line above) is stored as  $disMin$ , and will be used later in an algorithm that determines ray attenuation. Because the  $x$  component of the direction vector is positive, the cell index is incremented by 1 in the  $x$  direction, and the current

cell becomes (1,2). The ray has progressed, and the scenario is now represented by the following figure.

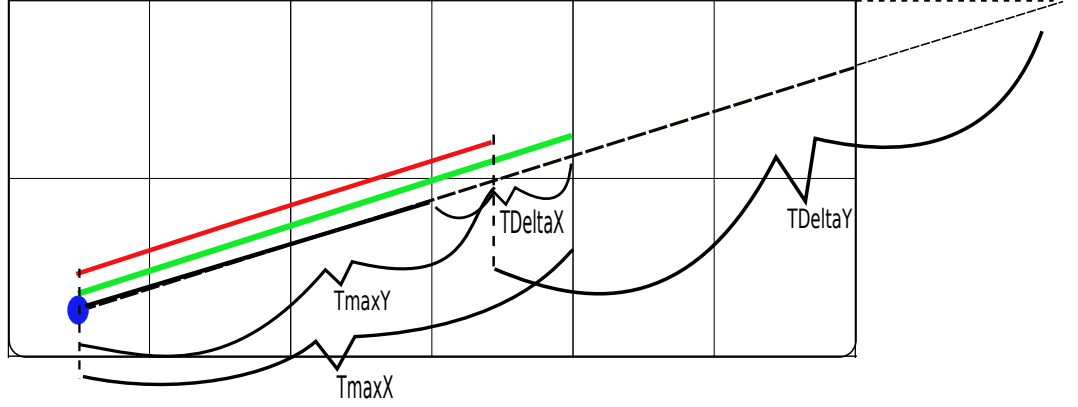


Note that the first segment of the dashed ray has now become solid. The distance from the origin to the first y plane has not changed, so the red line representing  $T_{maxY}$ , remains unchanged. However, the distance from the current location to the next x plane has changed, and its length has been increased by the distance  $T_{DeltaX}$ .  $T_{DeltaX}$  represents the distance required to traverse one cell length in the x direction. With our updated value for  $T_{maxX}$ , we again compare the green line to the red line. Because  $T_{maxX}$  is still shorter than  $T_{maxY}$ , we again step in the x direction, incrementing  $i$ . The current cell then becomes (1,3), and after storing the  $T_{DeltaX}$  as  $disMin$  for later use, we advance to the following figure.

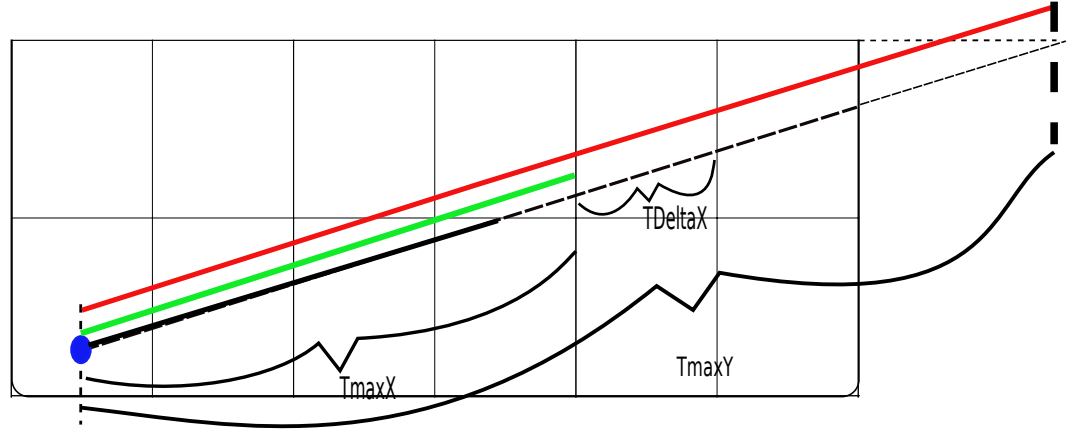


Now, the second segment of the dashed ray has become solid, and we have a new value for  $T_{maxX}$ , which was increased by  $T_{DeltaX}$ . Note that for a given ray in a uniform mesh,  $T_{DeltaX}$  and  $T_{DeltaY}$  do not change, as the distance

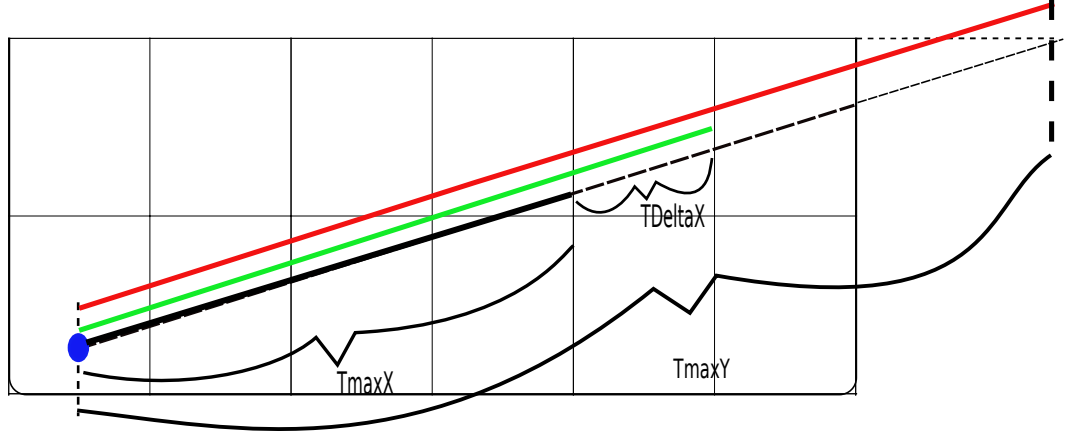
required to traverse a cell in the x or y direction is independent of the current cell. We again compare  $T\Delta x$  to  $T\Delta y$ . The green line is still shorter than the red, so we again step in the positive x direction, store  $T\Delta x$  as  $disMin$ , and reach cell (1,4) as shown below.



The third segment of the ray has now become solid. We increase the value of  $TmaxX$  by  $T\Delta x$ , and for the first time in this example, reach the scenario where  $TmaxX$  exceeds the length of  $TmaxY$ . We therefore step in the y direction and store  $T\Delta y$  as  $disMin$ . Because the y component of the direction vector is positive, we increment  $j$  and enter into cell (2,4) as shown below.



The fourth segment of the ray is shown as solid, and  $TmaxY$  has been increased by the distance  $T\Delta y$ . It is visually apparent that  $TmaxX$  is much shorter than  $TmaxY$ , and therefore the comparison in the algorithm would lead to a subsequent step in the x direction into cell (2,5) as shown below.



The fifth segment has become solid, and  $TmaxX$  has been increased by  $TDeltaX$ . At this time, the reader should be familiar enough with the algorithm to predict that the next two steps will be in the positive x direction, at which point the ray would either terminate if the wall is black, or reflect based on the reflection algorithm which will be discussed in later sections. Also in later sections, the reader will find a discussion of the attenuation of radiation from each of the cells along the ray path back to the origin, and the importance of  $disMin$  will become apparent.

## Non-cubic Cells

For domains which contain cells that have non-unity aspect ratios, additional considerations become necessary in the ray marching algorithm. Throughout the above described algorithm, distances are handled in units of cell width, which can then be converted to physical units simply by multiplying by the cell size of  $Dx$ . However, when  $Dx$  is not equal to  $Dy$  or  $Dz$ , this conversion becomes non-trivial, and requires additional computations. By normalizing the lengths of  $Dy$  and  $Dz$  by  $Dx$ , we are able to minimize the number of additional computations, such that only 6 lines of code require modification. Explanation of this procedure is as follows.

Take the distance  $Dx$  to be of unit length. Then  $Dy$  and  $Dz$  have normalized lengths of  $\frac{Dy}{Dx}$  and  $\frac{Dz}{Dx}$ , respectively. The first section of the algorithm that requires modification is then the determination of ray origins. Previously, for cubic cells, we assumed that the origin is located at  $(i + rand(), j + rand(), k + rand())$  where  $rand$  represents a function call to the random number generator which returns a random number distributed between 0 and 1. For non-cubic cells, we must scale the random numbers for the y and z directions by  $\frac{Dy}{Dx}$  and  $\frac{Dz}{Dx}$ , such that the origin of a ray becomes  $(i + rand(), j + \frac{Dy}{Dx} rand(), k + \frac{Dz}{Dx} rand())$ . In this manner, we ensure that the origins are randomly distributed throughout the cell, and not simply throughout a cube.

The second portion of the algorithm in need of modification is the determina-

tion of the original TMax values. Recall that the initial TMax values represent the distance from the origin to each of the respective x,y, and z planes. For example, recall that for cubic cells, TMaxY is calculated as follows

$$TMaxY = (j + sign[1] - rayLocation[1]) * invDirVector[1] \quad (1)$$

where  $sign[1]$  is a boolean with a value of 1 if the y component of the direction vector is positive, and zero otherwise.  $invDirVector[1]$  represents one divided by the y component of the direction vector. For instance, in figure 1, the origin is located at 17.343, 8.617. The direction vector has components of 0.7071, and 0.7071, such that the sum of their squares is equal to 1. Because the sign of the y component of the direction vector is positive, we add one to 17 to represent the location of a y breach, and subtract that value from the y value of the origin, then multiply by the inverse of the y direction vector. Implementing equation (1) we find that TMaxY as follows

$$TmaxY = (8 + 1 - 8.617) * \frac{1}{.7071} = 0.5416$$

This value is smaller than that of TmaxX which is calculated as follows

$$TmaxX = (17 + 1 - 17.343) * \frac{1}{.7071} = 0.9291$$

For cubic cells, however, when a given component of the direction vector is positive, we need to subtract the origin value not from  $1 + j$ , but from  $1 * \frac{Dy}{Dx} + j$ . When the component of the direction vector is negative, this multiplication of  $\frac{Dy}{Dx}$  becomes unnecessary. This is because the negative face value (8 in figure 1) is independent of the skewness ratio. To elegantly handle the condition of multiplying by the ratio  $\frac{Dy}{Dx}$  the following formulation is used for the more general case of non-cubic cells.

$$TMaxY = (j + sign[1] * \frac{Dy}{Dx} - rayLocation[1]) * invDirVector[1]$$

To illustrate, see figure 2 where a cell with  $\frac{Dy}{Dx} = 2$  is superimposed onto the same setup as illustrated in figure 1. Here, TmaxX is still equal to 0.9291, but TMaxY is solved as follows

$$TMaxY = (8 + sign[1] * \frac{2}{1} - 8.617) * \frac{1}{.7071} = 1.959.$$

Therefore, the ray will not breach the y face during the first step, but will instead breach the x face and enter into the cell to the right. TDeltaY and TDeltaZ are solved in a similar manner, such that for non-cubic cells, the following formula holds.

$$TDeltaY = invDirVector[1] * \frac{Dy}{Dx}.$$

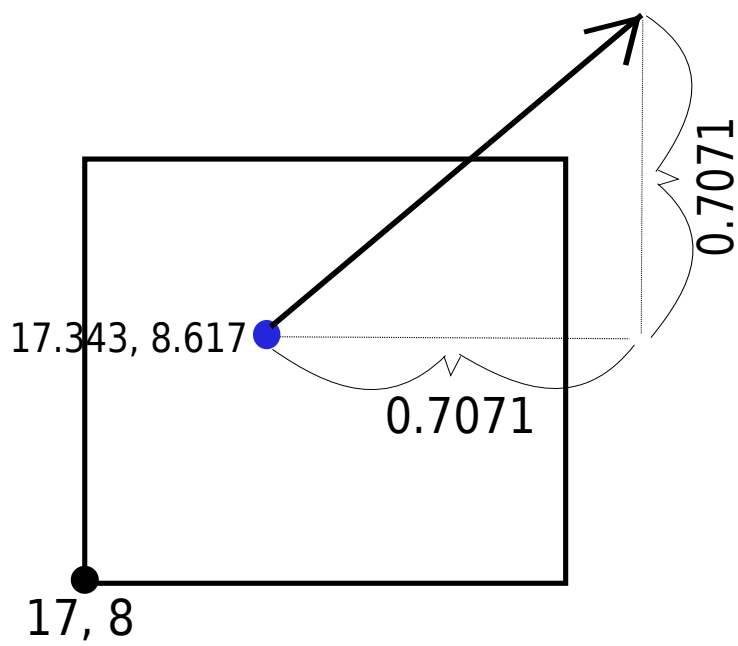


Figure 1: First step in a cubic cell

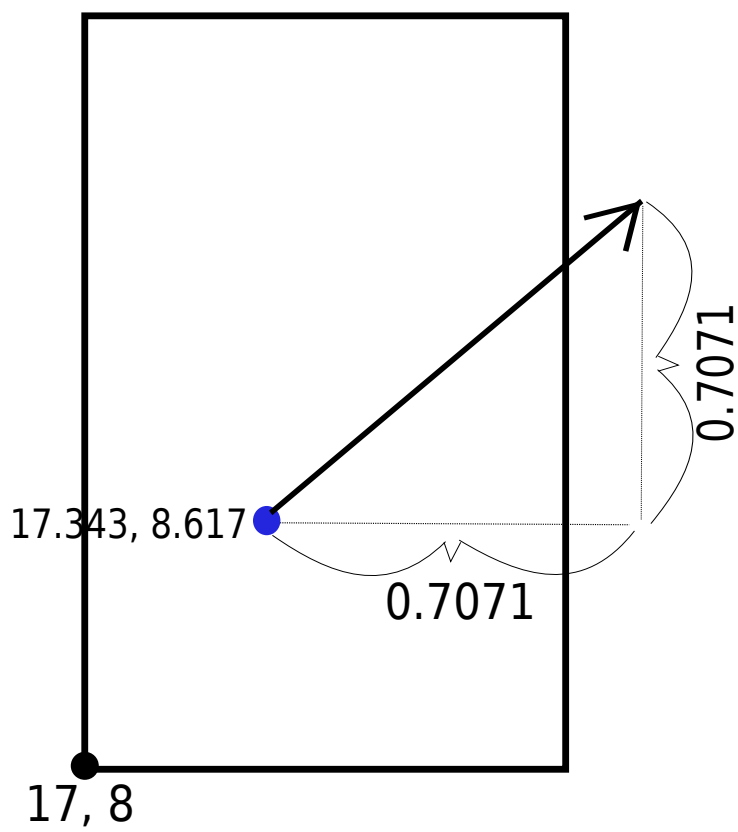


Figure 2: First step in a non-cubic cell with  $\frac{Dy}{Dx} = 2$