Best Target Selection for Auto-Aiming

Nicholas Gorski

March 13, 2012

1 Introduction

We wish to find the best target for auto-aiming. "Best" is defined to be the target that the player is most likely aiming for. Not being able to read the player's mind, we need to assume two things to calculate the best target: that the player is trying to center the reticule on the target, and that the player is aiming for the closest target. These two assumptions are not mutually exclusive (the player may intend to shoot a target quite off-center yet closer than the center-most enemy, or vice versa), so we combine them in a meaningful way to select an appropriate target.

2 Outline of Method

In order to find best target, we must score each potential target by distance and angle. For ease of calculations, we will choose the *lowest* scoring target.

2.1 Distance to Target

Calculating the distance to a target is trivial. Given player position P and target position T, let the vector V be $T - P$. The distance d between the target and player is $||V||$.

In order to later meaningfully combine distance (an unlimited value) with angle (a limited value), distance needs to be limited as well. This is easy to do by introducing a maximum considered distance, D_{Max}. Targets beyond this distance should be considered too far away to aim at; we calculate $D = \frac{d}{D_{Max}}$, and cull values above one. We call this the *distance score*. See Figure 2.1 for a visual reference.

2.2 Angle to Target

To calculate the angle to a target, we use the dot product. Recall the dot product between vectors $\vec{u}$ and $\vec{v}$:

$$\vec{u} \cdot \vec{v} = \vec{u}_x\vec{v}_x + \vec{u}_y\vec{v}_y + \vec{u}_z\vec{v}_z = \cos\theta \, \|\vec{u}\| \, \|\vec{v}\|$$

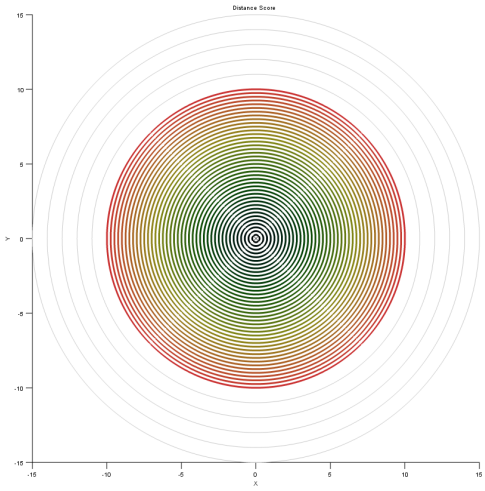


Figure 2.1: Influence of distance score, with D_{Max} set to 10. The closer the target, the lesser the value. Targets with a score greater than one are ignored (represented by sparse light contours).

where θ is the angle between the vectors. Let $\vec{u}$ be the direction the player is heading (which has unit length), and let $\vec{v}$ be the vector V defined above. In order to remove any bias of distance from the angle measure, the result of the dot product is divided by the distance to the target (making it normalized). The result of this calculation is a number in the range $[-1, 1]$, where -1 corresponds to the angle directly behind the player, while 1 corresponds to the angle directly in front of the player.

Note that the result of the dot product, being a function of cosine, is not linear. (That is, a ship 45°off-center will have a score of ≈ 0.707 instead of 0.5.) In order to recover a linear map, we take the arc-cosine of the normalized dot product to get the angle, which is linear:

$$\theta = \arccos \frac{\vec{P}_{Direction} \cdot \vec{V}}{\left\|\vec{V}\right\|}$$

This measurement provides another method to cull targets out of consideration. Given a maximum angle θ_{Max}, any targets with $\theta > \theta_{Max}$ can be considered out of the player's auto-aiming field of view (this prevents the system from aiming for a target directly behind the player). We can map values

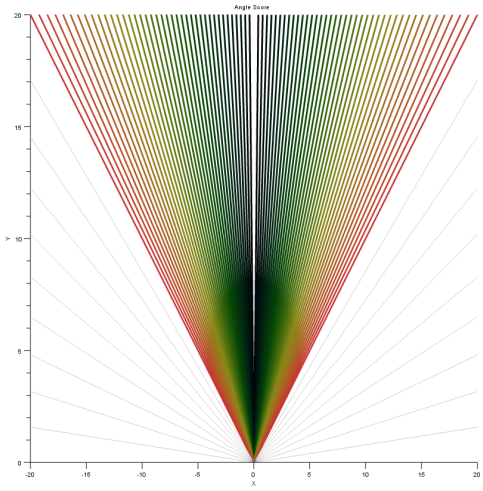


Figure 2.2: Influence of angle score, with θ_{Max} set to 45°. The less angular the target, the lesser the value. Targets with a score greater than one are ignored.

within the field of view into the range $[0, 1]$ via the calculation:

$$A = \frac{\theta}{\theta_{Max}} = \frac{\arccos \frac{\vec{P}_{Direction} \cdot \vec{V}}{\|\vec{V}\|}}{\theta_{Max}}$$

Targets with $A > 1$ have an angle greater than θ_{Max}, and should be culled. We call this the *angle score*. See Figure 2.2 for a visual reference.

2.3 Combination of Measurements

We combine measurements into a *final score* by a weighted multiplication between the distance score and angle score:

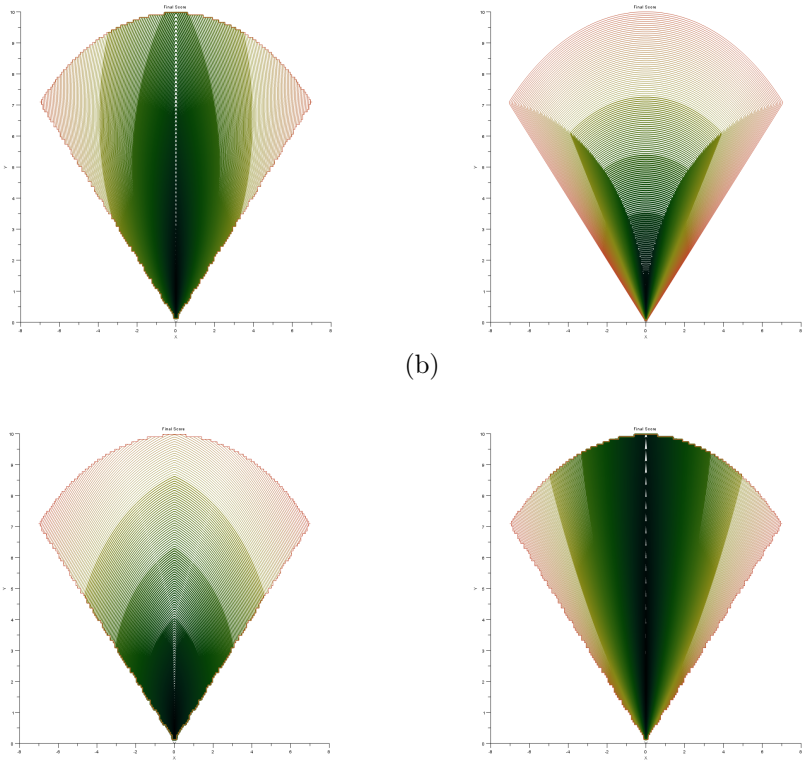
$$S = (1 - w)D + wA$$

Where w is the weight.[1] Alternatively, we can take the maximum value between D and A:

$$S = \sup(D, A)$$

See Figure 2.3 for a visual reference.

[1] A weight of 0 makes the final score equal to the distance score, and 1 makes it equal to the angle score. This should start at $\frac{1}{2}$ and be tweaked from testing.



(a) (b)

(c) (d)

Figure 2.3: Four differently computed final scores, using the example distance and angle scores above. (a) has a weight of $\frac{1}{2}$, (b) is the maximum between distance score and angle score, (c) has a weight of $\frac{1}{6}$, and (d) has a weight of $\frac{5}{6}$.

2.4 Aiming

Once a target is selected, we can use the final score to introduce artificial inaccuracy. Note that D and A are between 0 and 1, so S will also be between 0 and 1. This maps directly into an *inaccuracy score*, where a perfectly placed target has an inaccuracy score of zero, and a target at the edge of detection has an inaccuracy score of one.

Let R be a random number between 0 and 1; if $S > R$ then purposefully miss. (Aiming for a target is not covered here.)

3 Pseudo-Code

The pseudo-code for the procedure is given below:

```
function score_target(shooter, target)
    V = target.position - shooter.position
    VLength = V.length

    D = VLength / maxDistance
    if (D > 1) then
        // skip target, out of range
        continue
    end

    angle = acos(dot(V, shooter.direction) / VLength)
    A = angle / maxAngle
    if (A > 1) then
        // skip target, out of field of view
        continue
    end

    S = (1 - weight) * D + weight * A
    // alternatively: S = max(D, A)

    return S
end
```

```
function select_target()
    lowestScore = 1
    bestTarget = null

    for each target in targets
        S = score_target(player, target)

        if (S <= lowestScore) then
            lowestScore = S
            bestTarget = target
        end
    end

    return bestTarget, lowestScore
end
```

```
function fire_at_target()
    target, inaccuracy = select_target()

    if (target != null) then
        // predicting where to aim is not covered here
        targetPoint = aim_at_target(target, inaccuracy)

        shoot_in_direction(targetPoint - player.position)
    else
        // no target, just fire straight ahead
        shoot_in_direction(player.direction)
    end
end
```