Perfecting Ender Pearl Throws

Discord: Julia_bwah

October 2023 – March 2024

Contents

1 Introduction

Since the ender pearl's release in 2011, the question of what the best angle to throw it at remains. The current popularized answer is a range of around 35° to 40°. Why the answer is a range rather than a definitive number is due to the pearl's inherent randomness in its travel: the game introduces small deviations to the flight path of each pearl, even when they are thrown at exactly the same angle.

In this paper, we will calculate the *exact* optimal angle by analyzing a decompiled code of Minecraft and verify the result empirically with pearl noise removed.

In our research, we develop equations that model the trajectory of a pearl, enabling further calculations to be done with them, such as: calculating flight paths, finding the best angle given uneven elevation, and finding the best angle for speed.

2 Problem Setup

2.1 Pitch & Yaw

Figure 1: The F3 debug menu with the player's pitch and yaw highlighted.

The *pitch* of a player is their line of sight with respect to the horizontal plane; it is the throwing angle of the pearl. The *yaw* of a player is their rotation with respect to the vertical axis; it is the cardinal direction the player is facing.

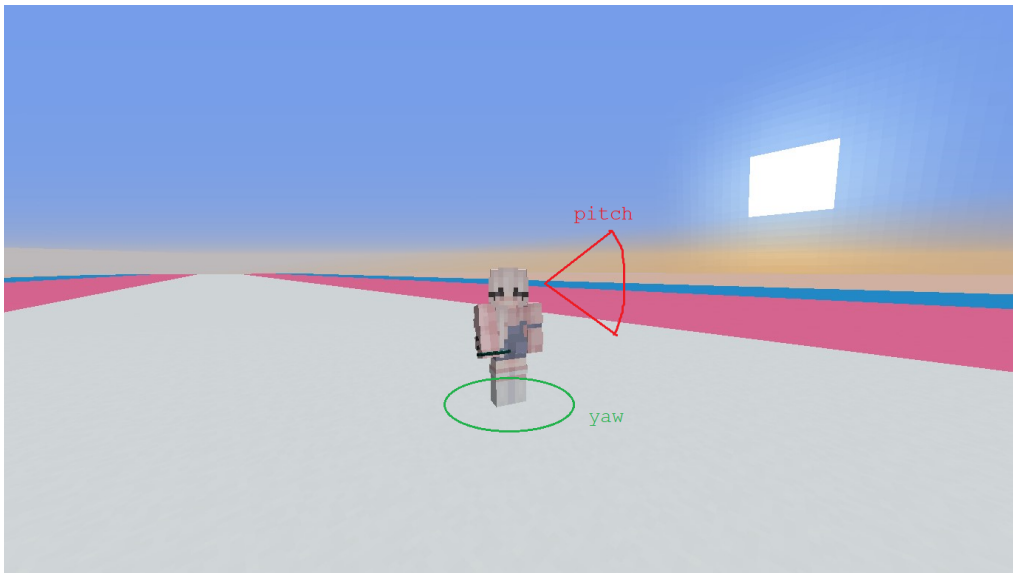

Figure 2: Pitch and Yaw Visualized

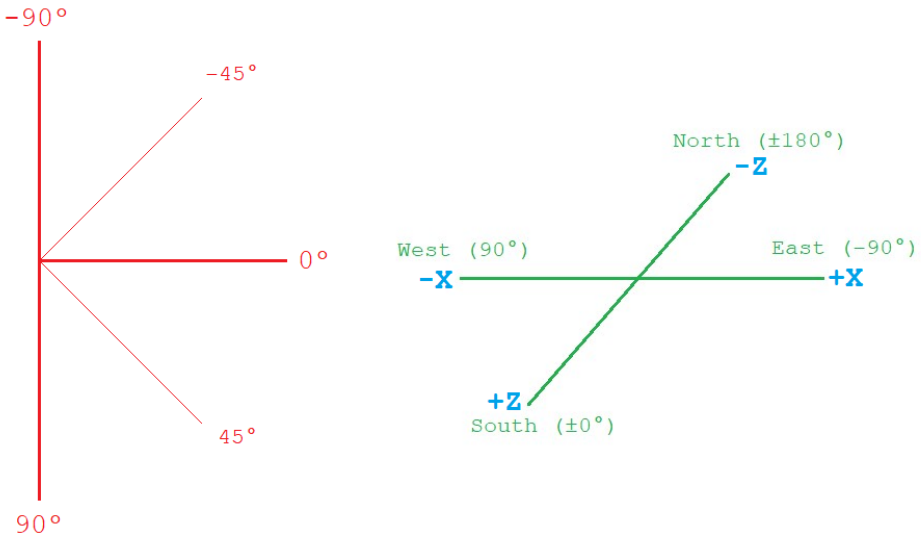

Figure 3: Common Pitch Values, and common yaw values and their cardinal direction. Note that an upwards-inclined pitch corresponds to negative-valued angles.

Our calculations will assume a constant yaw of 0°, travelling completely south, so a thrown pearl originating at $(0, 0)$ would have its distance measured by simply reading the z-coordinate at landing. We will assume flat ground. We will assume there is no randomness induced by the game into the pearl, so by answering the question of best angle, we would be finding the best angle *on average*. **Our goal** is to determine the pitch in the interval $[-90, 90]$ that achieves the furthest distance.

3 Velocity

Minecraft's smallest increment of time is a *tick* with 20 ticks present in 1 real-time second. In-game events occur only on every tick and none in between, which will be a foundational feature in our calculations.

Below is the code that dictates the behavior of ender pearls, from the tick at which the player right clicks with a pearl in their hand to the tick where it collides. High level comments are added to aid in understanding. Hereafter we will reference parts of the code with their labelled code names, e.g. Code A.

https://imgur.com/a/FCSpqAH

```
public void tick() {
   super.tick();

   /* Check for collision */
   RayTraceResult raytraceresult = ProjectileHelper.func_234618_a_( p_234618_0_: this,
    this::func_230298_a_, RayTraceContext.BlockMode.OUTLINE);
   boolean flag = false;
   if (raytraceresult.getType() == RayTraceResult.Type.BLOCK) {...}

   if (raytraceresult.getType() != RayTraceResult.Type.MISS && !flag) {...}

   /* Compute new position */
   Vector3d vector3d = this.getMotion();
   double d2 = this.getPosX() + vector3d.x;
   double d0 = this.getPosY() + vector3d.y;
   double d1 = this.getPosZ() + vector3d.z;
   this.func_234617_x_(); // Set new rotation and pitch
   float f;
   if (this.isInWater()) {...} else {
      f = 0.99F;
   }

   /* Compute new velocity for next tick */
   this.setMotion(vector3d.scale((double)f));
   if (!this.hasNoGravity()) {
      Vector3d vector3d1 = this.getMotion();
      this.setMotion(vector3d1.x, y: vector3d1.y - (double)this.getGravityVelocity(),
       vector3d1.z); // getGravityVelocity returns 0.03
   }

   /* Set new position (based on the velocity for this tick) */
   this.setPosition(d2, d0, d1);
}
```

Figure 4: The ender pearl code and explanations

The game provides velocity equations from which we can derive position equations. The position equations will serve as our primary tool in determining the optimal angle.

3.1 Initial Velocity

Referencing Codes A, D, E we will collect the equations for initial velocity, that is, the starting velocity values at tick 0. Let θ_y, θ_p be the yaw and pitch of the player at time of throwing, respectively. Code A calls Code D to begin setting the initial velocity vector of the pearl. Code D receives the player's pitch θ_p and yaw θ_y and sets three constants:

$$\begin{aligned} f &= -\sin(\theta_y)\cos(\theta_p) \\ f_1 &= -\sin(\theta_p) \\ f_2 &= \cos(\theta_y)\cos(\theta_p) \end{aligned}$$

Code D then calls `this.shoot(`$f, f_1, f_2, 1.5, 1.0$`)`, which is Code E. `this.shoot` uses all the arguments, along with the player's initial velocity (v_{px}, v_{py}, v_{pz}), to calculate and set the initial velocity vector:

$$v_{x0} = 1.5\frac{f}{\sqrt{\sin^2(\theta_y)\cos^2(\theta_p) + \sin^2(\theta_p) + \cos^2(\theta_y)\cos^2(\theta_p)}} + v_{px}$$

$$v_{y0} = 1.5\frac{f_1}{\sqrt{\sin^2(\theta_y)\cos^2(\theta_p) + \sin^2(\theta_p) + \cos^2(\theta_y)\cos^2(\theta_p)}} + v_{py}$$

$$v_{z0} = 1.5\frac{f_2}{\sqrt{\sin^2(\theta_y)\cos^2(\theta_p) + \sin^2(\theta_p) + \cos^2(\theta_y)\cos^2(\theta_p)}} + v_{pz}$$

It is at this step where the pearl's randomness originates – unwritten in the equations above is a random value

added to each velocity through `.nextGaussian()` in Code E. This perturbation in initial velocity carries through the whole pearl's airtime, leading to different ending positions. As mentioned in our assumptions, we will ignore this randomness.

These velocities could be simplified by recalling that we assume a constant yaw of 0°. This zeroes out all occurrences of $\sin(\theta_y)$ and forces $\cos(\theta_y)$ to 1. This, combined with the Pythagorean trig identity $\sin^2 A + \cos^2 A = 1$, neatly reduces the denominator of each equation to 1. We also ignore the player velocity as that is constant, independent of pitch and yaw. Applying these, and letting $\theta = \theta_p$ for brevity, we yield great simplifications:

$$\begin{aligned} v_{x0} &= 0 \\ v_{y0} &= -1.5\sin(\theta) \\ v_{z0} &= 1.5\cos(\theta) \end{aligned} \tag{1}$$

Hereafter, we will refer to these rather than the generalized forms.

3.2 Velocity at Tick t

The velocity for subsequent ticks after tick 0 is determined by Code F, the `this.tick` method for the airborne pearl entity. In line 91, the current velocity vector is multiplied by 0.99 (provided the pearl isn't in water), and its y-coordinate subsequently has 0.03 subtracted. Hence, letting $v(t, \theta)$ be the velocity function at tick $t > 0$, we have the following recursive functions:

$$\begin{aligned} v_x(t,\theta) &= 0.99 \cdot v_x(t-1) \\ v_y(t,\theta) &= 0.99 \cdot v_y(t-1) - 0.03 \\ v_z(t,\theta) &= 0.99 \cdot v_z(t-1) \end{aligned}$$

To enable further computation, we will eliminate the recursions by computing closed forms for them, that is, not dependent on the previous tick. At a given tick t, there will have been t multiplications by 0.99 over ticks 1 to t. And down t levels in the recursion has the base case of the initial velocity. Hence, $v_x(t) = 0.99^t v_{x0}$ and $v_z(t) = 0.99^t v_{z0}$.

The same applies to v_y, but more care needs to be given to the -0.03 term. In the tick that is i ticks before t, its -0.03 term is multiplied i times by 0.99 (consider $i = 0$ and $i = 1$ for clarity). Summing this over $i = 0$ to $i = t - 1$, an additive value of $\sum_{i=0}^{t-1}(-0.03)0.99^i$ is accumulated over ticks 1 to t (tick 1 corresponds to $i = t - 1$ and tick t to $i = 0$). Using the formula for the sum of a finite geometric sequence, [1] this sum is $-0.03\left(\frac{1-0.99^t}{0.01}\right) = -3 + 3 \cdot 0.99^t$. Therefore, we have $v_y(t,\theta) = 0.99^t v_{y0} + (-3 + 3 \cdot 0.99^t) = (v_{y0} + 3) \cdot 0.99^t - 3$.

Putting it all together, we obtain the velocity equations:

$$\begin{aligned} v_x(t,\theta) &= v_{x0} \cdot 0.99^t \\ v_y(t,\theta) &= (v_{y0} + 3) \cdot 0.99^t - 3 \\ v_z(t,\theta) &= v_{z0} \cdot 0.99^t \end{aligned} \tag{2}$$

[1] Sum of finite geometric sequence: $\sum_{k=0}^{n-1} ar^k = a\frac{1-r^n}{1-r}$ for $-1 < r < 1$.

4 Position Equations

The velocity functions enable us to find the distance functions of a pearl. Since velocity is by definition meters travelled per tick, the position at a given tick t is obtained by summing the velocity values over all ticks prior. [2] We write this algebraically for each axis below, letting s be the position function. We then again use the sum of finite geometric sequences.

$$\begin{aligned} s_x(t,\theta) &= \sum_{i=0}^{t-1} v_{x0} \cdot 0.99^i & s_y(t,\theta) &= \sum_{i=0}^{t-1} \big((v_{y0}+3)\cdot 0.99^i - 3\big) & s_z(t,\theta) &= \sum_{i=0}^{t-1} v_{z0} \cdot 0.99^i \\ &= v_{x0}(100 - 100 \cdot 0.99^t) & &= (v_{y0}+3)(100 - 100\cdot 0.99^t) - 3t & &= v_{z0}(100 - 100 \cdot 0.99^t) \end{aligned}$$

Substituting in formulas (1), we have our position equations:

$$\begin{aligned} s_x(t,\theta) &= 0 \\ s_y(t,\theta) &= (-1.5\sin(\theta)+3)(100 - 100\cdot 0.99^t) - 3t \\ s_z(t,\theta) &= 1.5\cos(\theta)(100 - 100\cdot 0.99^t) \end{aligned} \tag{3}$$

Rephrasing the question of best angle – our goal is to find the pitch that achieves the highest s_z right when the pearl lands.

5 Best Angle

To find the ending distance for an angle θ, we need to know the landing time T_θ. This is because $s_z(T_\theta, \theta)$ is precisely the ending distance of a pearl thrown at θ. Letting the ground level be g, then T_θ can be computed by solving $s_y(t,\theta) = g$ and rounding down to the previous tick. [3] Therefore, we devise the following procedure: for each possible angle θ in $[-90, 90]$, solve $s_y(t,\theta) = g$ and plug its answer into s_z; take the angle that produced the maximum s_z as the best throwing angle.

However, doing this procedure for all angles in $[-90, 90]$ would be computationally infeasible if we want float-level precision (that is, to many decimal places of precision) as there would be billions or trillions of angles to test. Fortunately, there is a solution:

Fact 5.1. Although we have treated our angle θ as able to take on any decimal value, its domain is in fact much more limited due to the way Minecraft does its math, only able to take on multiples of $\frac{360}{65536}^\circ \approx 0.0055^\circ$. We will call these *significant angles* as they are by the technical parkour community. [4] The significant angles of our working pitch range $[-90, 90]$ are given by $\frac{360}{65536}k$ for $-16384 \le k \le 16384$ because $\frac{360}{65536}(\pm 16384) = \pm 90$. The player's angle θ is stored as a float, and at time of throwing, the pearl adopts the highest significant angle lower than θ.

With this observation, we turn the space of billions of angles to only a couple thousand, making our earlier algorithm feasible. Below is our revised algorithm, iterating over all significant angles and computing the distance for each.

This procedure was implemented in Python (`https://pastebin.com/33whTy1J`) to compute the distance for all significant angles, yielding the following top 10 entries, sorted by predicted distance. They were also tested against

[2] Those familiar with calculus may know this as taking the definite integral of velocity up to $t-1$. This approach is allowed provided that time is floored to preserve the discreteness of ticks.

[3] More technically, the tick at which the pearl lands is the latest tick before which $s_y(t,\theta) < g$.

[4] `https://www.mcpk.wiki/wiki/Parkour_Nomenclature#Technical_parkour_terms`

Algorithm 1 Determine Best Pitch

```
for k = −16384 to 16384:
    θ = (360/65536) k
    Solve s_y(t, θ) = g, and round down the result; call this T_θ
    Compute s_z(T_θ, θ)
Take the pitch with the highest s_z as the best angle
```

the actual distance obtained in game with a margin of error of (the small inaccuracies are discussed later).

Pitch (°)	Predicted Distance	Actual Distance
-37.9742431640625	52.89209945925462	52.890613835325986
-37.979736328125	52.88814101272806	52.88665432209301
-38.86962890625	52.885939185946924	52.8843444437028545
-37.9852294921875	52.884182080065074	52.88269480886004
-38.8751220703125	52.88185210098631	52.8802550219269
-37.99072265625	52.880222661302035	52.77145306831007
-38.880615234375	52.87776452994706	52.87616560682527
-37.9962158203125	52.876262756475334	52.874775782394124
-38.8861083984375	52.873676472866755	52.872080240649446
-38.001708984375	52.87230236562137	52.87081226965284

Figure 5: Top 10 Pitch-Distance Combinations

Pitch (°)	Predicted Distance	Actual Distance
-38	52.876262756475334	52.874775782394124
-40	52.66673513971652	52.66499709116808
-45	51.427474832292816	51.42502374586834
-35	52.17929959463825	52.1781583024189

Figure 6: -38 Compared to Commonly Believed Optimal Angles

Our predictions have a small margin of error, but on average, it is only 0.012 blocks of the actual distances. It is currently unknown what causes this, but the sequence of errors is monotonic with the pitch, so the cause is likely not a randomization issue.

We obtain that the best pitch to maximize distance is $-37.9742431640625°$. Since the F3 menu displays one's pitch to only 1 decimal place, $-38.0°$ would be the best angle in practical settings. However, the precision obtained in this angle provides a definitive resolution (and in my opinion, a very satisfying one) to the longstanding debate of the best pearl angle.

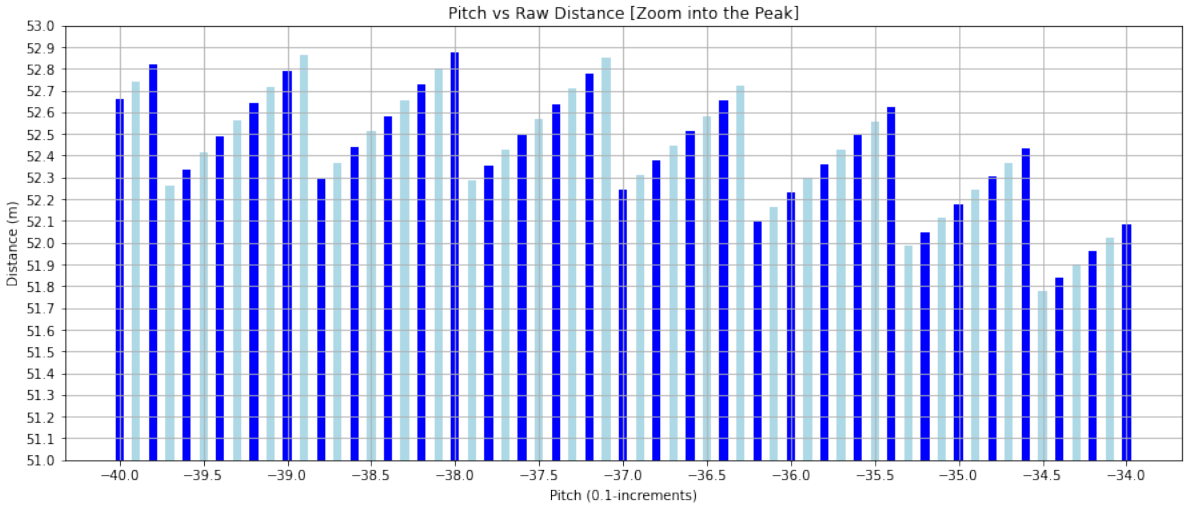

Figure 7: Distances travelled for the optimal pitch range. If curious about the oscillations, feel free to contact me.