VariaBULLET2D - Tips and Performance Guide

by Neon Dagger

1. General Usage

Object Pooling RealTime Update – There are some cases in which you won't see changes in real-time to a bullet's property/behavior when pooling is enabled. This is because the property/behavior is established when the object is created or is not regularly updated. In these cases, it's best to remove pooling at least temporarily in order to do real-time testing.

OnCollisionEnter2D vs OnTriggerEnter2D – VariaBULLET2D's collision system relies on the OnCollisionEnter2D event method. There are a variety of reasons for this design choice. Using trigger collisions is often considered better for performance (particularly in older versions of Unity), but in some cases does not carry some collision information which might be important (such as the exact point of the collision). You can modify existing references to OnCollisionEnter2D in order to implement an alternative collision detection but in the vast majority of cases this should not be necessary. Additionally, newer versions of Unity have reduced the performance gap between these two different collision methods.

Modifying Prefabs – It's important to remember that whenever you modify a Shot prefab, it will modify it for all instances in which it is used. For this reason, it is considered best practice to make copies of the prefab and make any changes on those. Note that newer versions of Unity have more robust prefab systems that allow for extension and flexible modification.

Scene View Editing – It's often helpful to switch to the Scene view tab in order to select emitters, bullets and other gameobjects in order to better isolate them for editing/viewing in the inspector.

Removing Emitters – At some point you might want to simply remove emitters from a given pattern that you've constructed. However, by deleting the emitter you will break not only the pattern's underlying array structure but also likely shift the pattern in an undesirable way. The easiest way to remove an emitter without causing these unwanted effects is to simply disable its enable/disable toggle directly in the inspector.

2. Test Bench

During development of VariaBULLET2D a series of tests were carried out to ensure a balance between ease-of-use, functionality and performance. A low-spec system was purposefully used to test general performance.

Below are the general findings of the tests along with the system specs. You can use this information in order to predict performance on low-spec systems as well as what modifications can be done to improve performance if required.

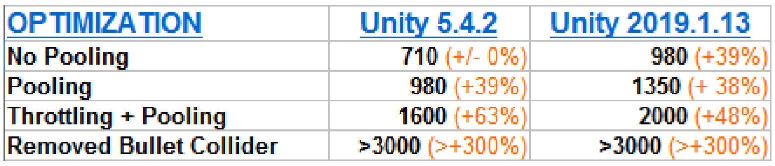
Note that the tests were carried out in actual builds – rather than when running live in the editor – as determining actual performance in the editor is not a reliable measure due to the overhead required.

Test System Specs

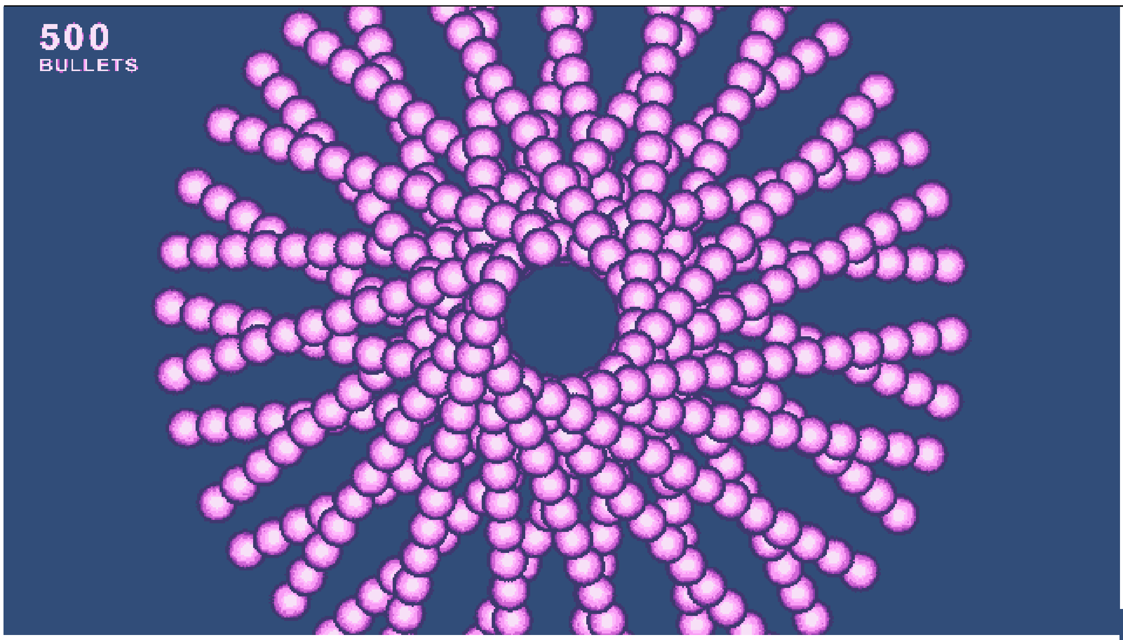
CPU: AMD FX6300 (Release Year: 2012)
GPU: NVIDIA 8800GT (Release Year: 2007)
OS: Windows 7 SP1
Unity Builds: v5.4.2 and v2019.1.13

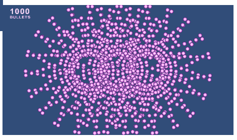
Test Results

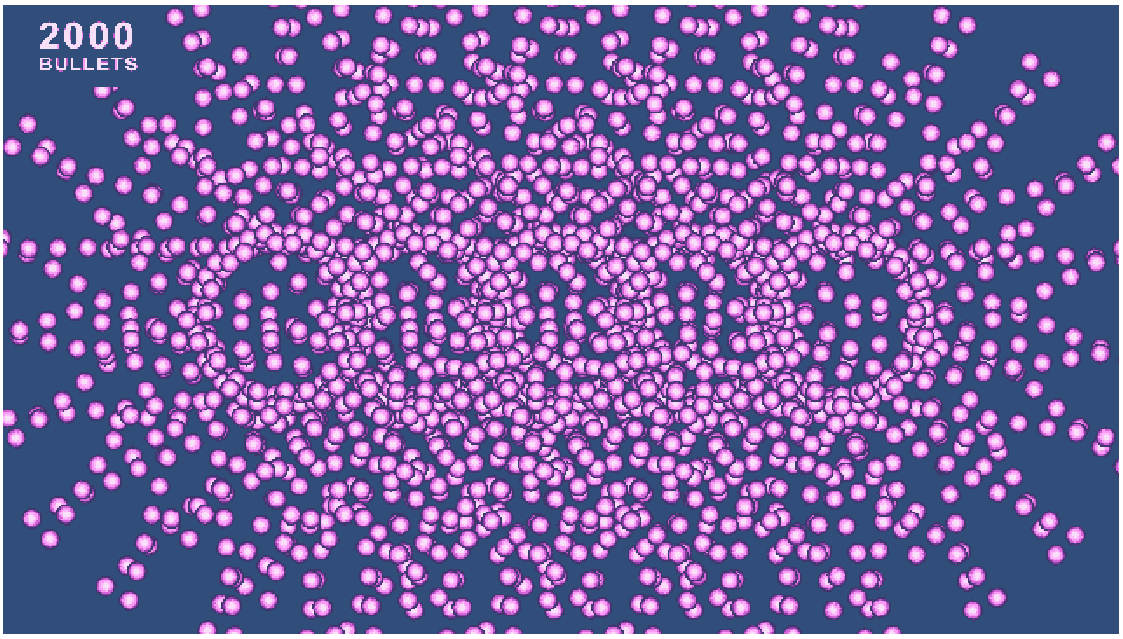
Below you will find the results of how many concurrent bullet objects were typically achievable before a drop in framerate (<60FPS) occurred, relative to the version of Unity and the built-in performance optimizations that were used.



OPTIMIZATION	Unity 5.4.2	Unity 2019.1.13
No Pooling	710 (+/- 0%)	980 (+39%)
Pooling	980 (+39%)	1350 (+ 38%)
Throttling + Pooling	1600 (+63%)	2000 (+48%)
Removed Bullet Collider	>3000 (>+300%)	>3000 (>+300%)







Interpretation

The test clearly shows a significant performance boost simply by using a **newer version** of Unity (average 42% performance advantage). This is due to Unity’s optimizations in the underlying collision engine.

Object **pooling** is another clear benefit that should be used in almost all cases (average 39% performance advantage).

Throttling can produce an even more dramatic performance advantage well beyond 60%, while producing the trade-off of slowing the engine (a rate and amount of which is user defined and dependent on the amount of bullets on screen). However, this is often a desirable effect, and is used in a variety of retro titles including many games in the Bullet Hell genre.

By far the biggest performance boost was from simply **removing the colliders** from the shot prefabs. The aging test bench system was able to handle well beyond 3000 concurrent bullets simultaneously at 60FPS.

However, in bypassing the collider, you will need to implement your own collision detection that does not depend on the underlying physics engine.

Note: as of v1.03, there is a simple performance enhancement that produces all of the performance benefits of removing colliders while maintaining most of the user-friendly benefits of using colliders. For more info see the **Collider Flux** section at the bottom of the Bullet Shot(https://neondagger.com/variabullet2d-in-depth-shot-guide/#bullet-shots)
section in the shot guide.

Conclusion

For the vast majority of cases, you should expect a solid 60FPS with well over 1000 concurrent bullets – even on a low-spec system – by simply enabling object pooling and using a newer version of Unity. This is typically more than enough bullets to completely fill the screen unless they are extremely small in size.

For instances that require thousands of concurrent bullets, consider using throttling and/or shot **ColliderFlux**/**StaticColl** options. For more info on performance enhancements, see the tips section below.

3. Performance Tips

The following is a list of tips you can use – in order of impact – to improve performance in VariaBULLET2D.

Enable **ColliderFlux** on bullet shots which have only a single collision source (EG: enemy bullets which only are going to collide with player1). Using this one special-case tweak could triple performance. Particularly useful when many 1000’s of bullets might be in play at once.

Enable **Throttling**. Throttling is a great option to produce predictable “slowdown” emulation, but it can also dramatically improve performance, particularly when concurrent shot counts end up in the thousands.

Enable **Pooling** strategically. Identify which shots are most common in the scene and consider adding them to the **GlobalShotBank** and utilizing **Pre-Banking** + **Banking Enabled** to keep them in persistent memory. All other shots can be pooled with Pooling Enabled however those shots will be destroyed when the emitter they belong to is destroyed, so is best used with uncommon shots or low-quantity shots.

Consider using a **single sprite** frame for shots types that consistently end up on the screen in large quantities. Using a single frame bullet when there are 1000’s of it on the screen ensures that the GPU draw call count remains low. As an example, a single frame bullet instantiated 4000+ times will only result in 8 batches/draw calls.

Use **FreezeEdits** on any controller which is not going to be automated in real-time. This is particularly useful when there are a lot of emitters attached to the controller.

While not a performance enhancement per se, you could create the illusion of having more bullets on screen by simply increasing the size of the bullet sprite images to maximize the visual effect while minimizing performance load. A similar possibility is to narrow the visible screen space, causing the bullets to be re-pooled faster while at the same time increasing the ratio of bullets relative to screen space despite having less concurrent bullets on screen.