VFX Optimization Guide

References and Supporting Docs

- https://docs.unrealengine.com/en-US/Engine/Rendering/ParticleSystems/Optimization/index.html
- https://docs.unrealengine.com/en-US/Engine/Rendering/ParticleSystems/Optimization/Results/index.html#gamethread

General Guidelines for Building Optimization into VFX Creation

Goal: Build basic optimization practices, that once fully integrated, only add a few minutes to particle creation time and avoid the VFX team burying itself with performance issues in the future (see Note about LOD's below as an example). Ideally, following these best practices will keep FX running smoothly until very late dev, or stay off performance reports entirely.

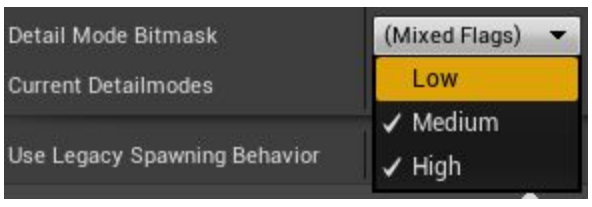
1. **Make Quick LODs even for "temp" VFX systems AND StaticMeshes used in MeshEmitters.**
    - **Why:** Significance manager depends on LOD levels to do any rate/tick improvements. In general, temp FX end up staying in-game for months, if not always, and drawing max LOD always from all distances introduces avoidable performance problems.
    - **Additional Tips:**
        - Disabling tiny/smaller/less noticeable emitters are quick, easy perf gains per LOD levels.
        - Tech art likes to create 4 LODs with 0, 1000, 3000, 6000 as base distances.

- Reduce spawn rate by 0.5 each LOD level for “quick LODs”.
- Reduce lifetimes of long-living particles to despawn faster.
- Disable collision modules and any other modules with high cost.
- Mesh emitters are very expensive because each “sprite” is a draw call. Mesh emitters need LODs with tri reductions too. UE4 can generate them quickly from the Reduction Settings.
    - The Auto-Compute LOD Distance is bad, uncheck that, some good baseline numbers for 4 LOD levels are: 1.0, 0.7, 0.25, 0.05

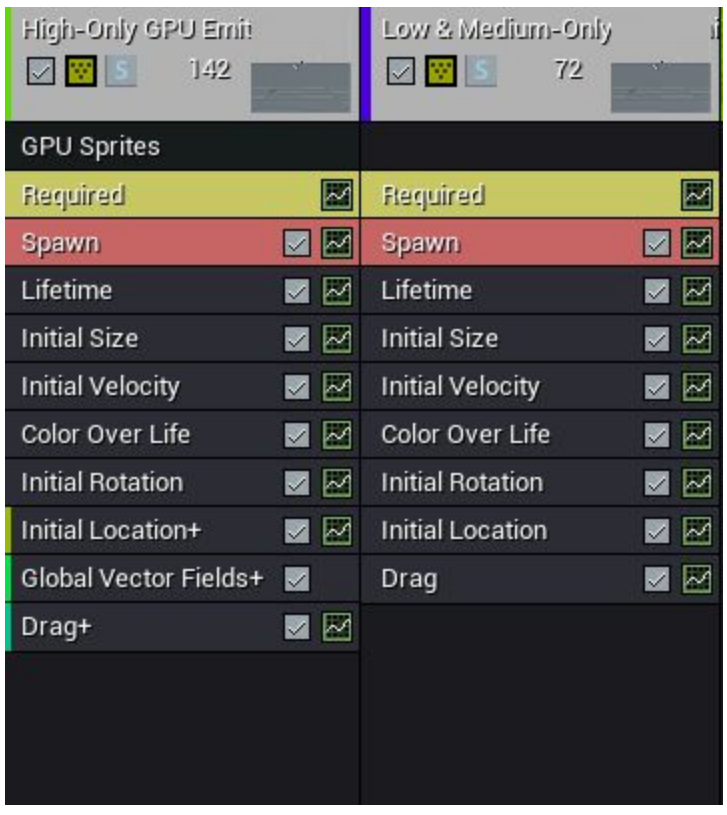
2. Have the least amount of emitters possible to achieve an effect.

- **Why:** Every emitter in a system is processed separately and hits the game thread/render thread separately. The more of these that have to process the more an effect will tax the frame.
- **Limitations:** Try to keep emitter count to 4 or less, especially if they have to interact with scene translucency (like water/ocean). The more frequent the use of an effect the less emitters it should have, in general.
- **Additional Tips:**
    - Drop low visual impact emitters on your different LOD levels.
    - Kill on Complete/Kill on Deactivate for non-looping emitters will help them get out of game thread faster.

3. Use inclusive/exclusive detail modes to disable emitters for low end platforms.



- **Additional Tips:**
    - For small things like embers, tiny water splashes, tiny blood drops try to disable them on Low AND Medium.
    - This can also be used for things like bubbles to be “High Only” for GPUSprites and have a really low regular emitter version for Low/Medium.
    - **Example:**

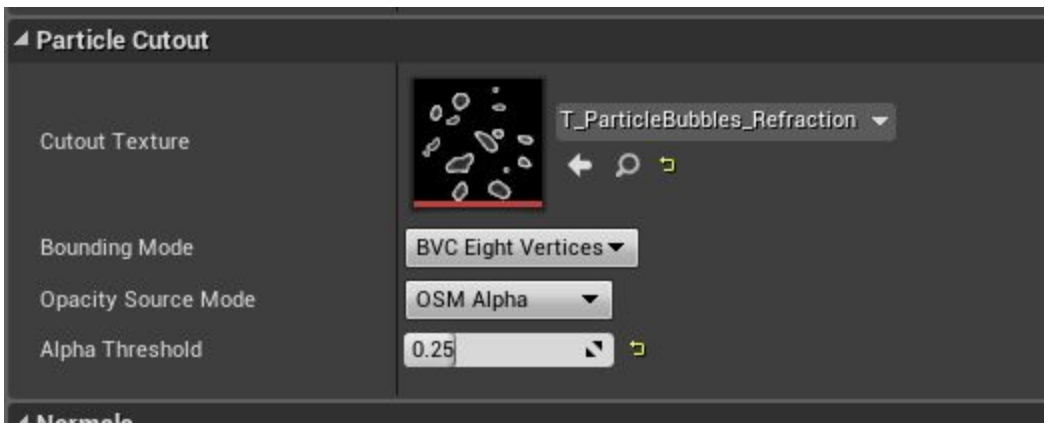


    - This bubble effect has a GPU Sprite w/ VectorField and draws a lot of bubbles by default. In the example, the GPU Sprites emitter was set to ***High*** only in it’s Detail Mode Bitmask settings.
    - The emitter to the right was created, removed GPU sprite, removed the vector field, cut the spawn rate and set the Detail Mode Bitmask to ***Medium and Low*** checked.
        - This disabled the GPU Sprite version on low end platforms like Xbox One and Switch.

4. Optimize & Use Quality Switch Material Nodes in VFX Materials

- **Why:** Pixel shader instructions contribute to the overall expense of particle systems especially Overdraw. Low-end platforms particularly struggle with this.
- **Additional Tips:**
    - Material Optimization Doc
    - Game Tech recommends quality switching Normal channel off on Low by default and Medium (if Xbox uses Medium, which it does on [video game]).

    - Panners/Rotators and other complicated material math dropping out would be beneficial. In general, small/nuanced details only seen when authoring, but not practically in-game should be dropped on ***Low/Med***.

5. **Set Required -> Particle Cutout Alpha Thresholds**
    - We recently enabled automatically applying particle cutout texture (using the mask from the material) automatically. For existing effects, just re-apply the same material for it to pick up the Cutout.
    - Set alpha threshold.
    - This will help reduce overdraw by having particle sprites that trim closer.
    - 

    - **Why:** Particle overdraw is one of the most significant costs of particle systems. Reducing overdraw where possible will hlep overall game performance.
    - **Additional Notes:** Some systems, that might not have overdraw issues may not benefit from this practice, and the added triangles could end up being a net-negative. Balance usage of this with good judgment but do not avoid using it either, giant sprites with lots of “black space” are most likely worth the bit of extra geo.
6. **Set Fixed Bounds (Especially of GPUSprites exist)**
    - **Why:** #1- If not fixed the game has to calculate bounds at every tick of the particle to encapsulate whatever the biggest/furthest emitter is. #2- The bigger the bounds, the less chance of the system being culled/occluded. If a pixel of a bounding box exists in the view frustum it will render/process.

GPU Particles

The CPU cost of GPU particles is dominated by the spawning of particles. Because particles are spawned on the CPU using the same methods as traditional CPU particles, the performance characteristics are similar.

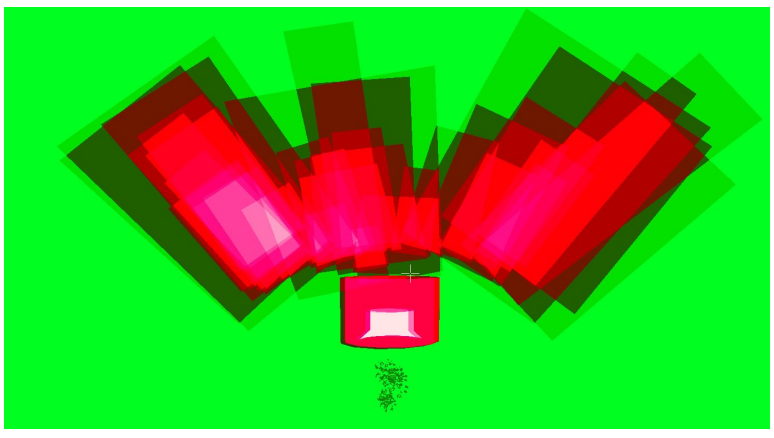
The GPU cost of particles is primarily determined by the number of particles. Few features add additional cost to GPU particles above the fixed cost that already exists. The majority of the GPU cost can be attributed to sorting and rendering. Sorting is optional and should be enabled only when required for a particular emitter. Rendering is often dominated by fill rate. Reducing the size of particles, the number of instructions on a particle's material and the total number of particles can all help. In some cases when the particles are very small, rendering is dominated by vertex cost in which case reducing the number of particles is the only way to reduce cost.

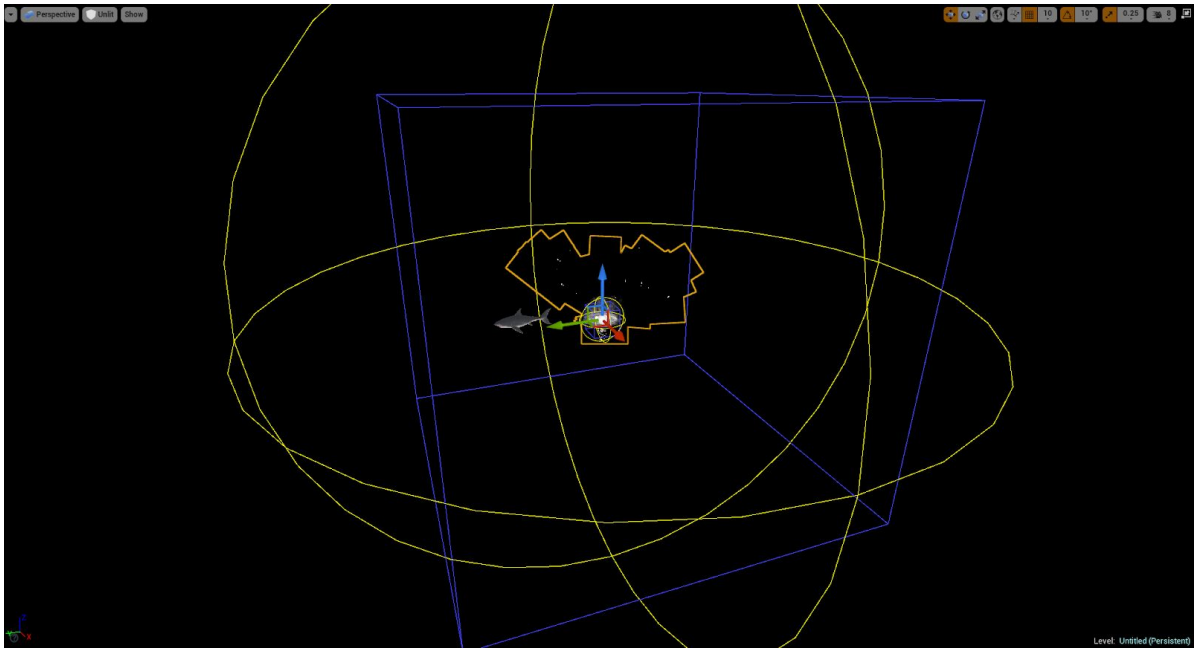
Simulation cost does scale with the number of vector fields that overlap an emitter, so reducing the number of overlapping vector fields can help in reducing simulation cost.

Practical Example

FX_Entry_Splash_Perfect at the time of writing is one of the more taxing effects in [video game]. Here are some facts about this effect:

Relevant Stats

- 11 Active Emitters (1 GPUSprites, 1 Mesh Emitter)
    - Mesh Emitter: Uses StaticMesh'/Game/VFX/xxTESTxx/FX_Husk_weapon_glow_01.FX_Husk_weapon_glow_01'
    - 168 tris, no LODs
- Peaks at 2520 tris
- Sprite Particles: 150 max
- Particle Draw Calls: 44 max
- Overdraw
    - 
- No particle LOD levels.
- No fixed bounds (shark shown to display how large they get).

- 

Xbox Test Case

I placed 9 of FX_Entry_Splash_Perfect evenly spaced at different distances (not overlapping) to simulate a projected “worst case scenario” to get numbers on Xbox.

Scenario	**FPS**	**Game Thread**	**Draw Thread**	**GPU Thread**
Base Scenario (At Rest)	170	5.62ms	3.13ms	3.83ms
Particles Fired (Unoptimized)	139	7.24ms	4.22ms	4.02ms

First Glance Assessment

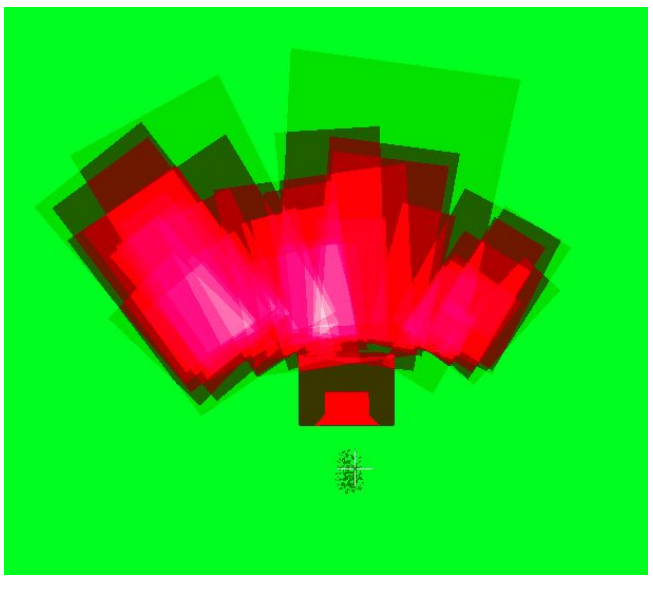
More or less just hitting the stuff from the General Guidelines section should fix this up easily. Seems particle cutouts and fixed bounds would make the biggest impact (and of course LODs). I’d imagine for most of these splashes, the bounds get so large that boats behind you would probably have their effects/splashes rendering because the bounds penetrate through the back of the view frustum.

Post-Optimization Test

I did the following optimizations to the effect:

- Set fixed bounds
  - Set them to -500, -500, 0 : 500, 500, 500
- Applied particle cutout by re-assigning materials with alpha threshold of 0.5 on all emitters.
- Set detail modes on Emitters
  - High spawn rate BubbleSplash was disabled altogether, redundant with the other one.
  - Disabled Shockwave emitters that didn’t seem to do anything
- Made 60% reduction LOD1 of the FX_Husk_weapon_glow StaticMesh
- Made 3 LODs with dropping spawn rate by 50% on most emitters, disabling small emitters and shortening lifetime on long bubbles.
- Reduced LOD0 emitter spawn rates for overlapping emitters that didn’t change the particle behavior and did not have random spawn location variation (The mesh data one specifically).

Post-Optimization Relevant Stats

- 7 Active Emitters (1 GPUSprites, 1 Mesh Emitter)
  - Mesh Emitter: Uses StaticMesh'/Game/VFX/xxTESTxx/FX_Husk_weapon_glow_01.FX_Husk_weapon_glow_01'
  - 168 tris, 1 LOD
- Peaks at 840 tris
- Sprite Particles: 116 max
- Particle Draw Calls: 28 max
- Overdraw
  - 
- 3 Particle LODS (besides Base)

Scenario	FPS	Game Thread	Draw Thread	GPU Thread
Base Scenario (At Rest)	170	5.62ms	3.13ms	3.83ms
Particles Fired (Unoptimized)	139	7.24ms	4.22ms	4.02ms
Particles Fired (Optimized)	163 **(+24)**	6.08ms **(-1.16ms)**	3.51ms **(-0.71ms)**	3.89ms **(-0.13ms)**

Note: Rest FPS was 15fps better just carrying the optimized versus un-optimized. The non-activated Game thread was 0.3ms better.