If we picture the world from above, let's divide it into uniform cells as in **fig 1**:

Every position in the world will fall within one of these cells.

For every cell, we can place a camera at the center of the cell (e.g. the red dot in **fig 2**) and then render all geometry beyond a certain radius r (the distance at which the effects of perspective has gotten too low to notice) to a skymap (a cubemap, most likely).

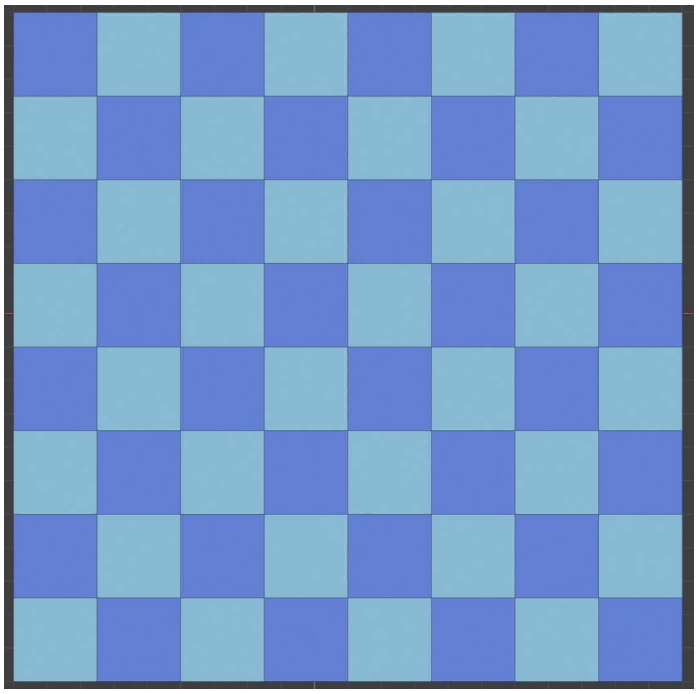

Fig 1.

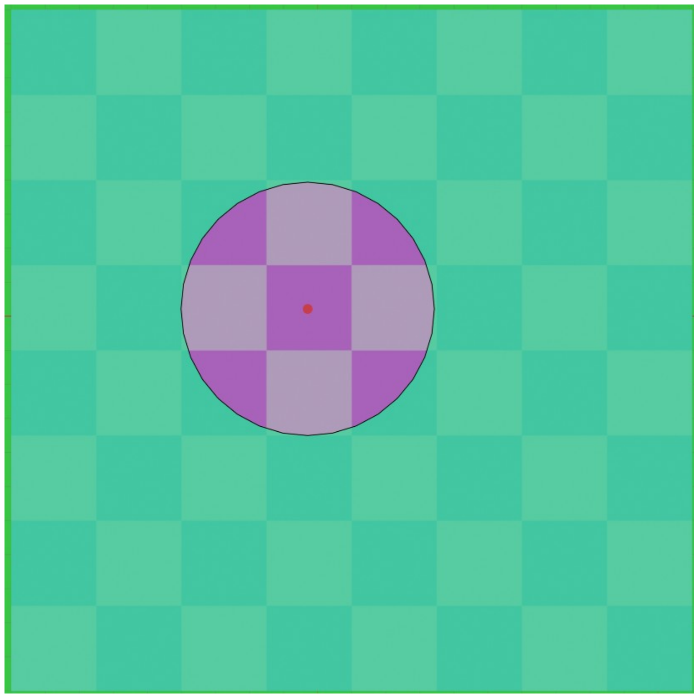

Fig 2

So, for every cell we'd end up like **fig 2**. All static geometry within the magenta sphere's geometry would have to be rendered as normal at runtime, but all static geometry outside of it would've been baked to the cell's skymap. Thus, the only geometry in the green space that we need to render at runtime is non-static geometry and the skybox.

During the baking, infinite view distance can be used, since the complexity of rendering the skymap at runtime is constant. Additionally, all of the static geometry in the green space won't need any level-of-detail variants.

In the baked textures, anything that's alpha can be replaced with a dynamically shaded sky at runtime. And depending on rendering requirements it would be compatible with both classical rendering (diffuse, specular, normal) workflows as well as PBR workflows, since all of the channels of every fragment can be baked into separate skymap textures and be composited at runtime with deferred rendering. The z-depth can be baked as well.

To avoid the skymap changing noticeably as the player's position transitions into a neighbouring world cell, the four closest cells' skymaps can be stored in memored and then it's just a question of interpolating between the colours of the four based off of proximity.

So I'd reckon that the potential net gains would be:

1. Higher visual fidelity (since LoD is only an issue at runtime).

2. Much greater view distances at no extra performance cost
(since the render distance can practically be infinite during baking).

3. Since distance geometry is baked, the runtime render distance can likely be decreased.
(at least for static geometry).

4. Better runtime performance, since:

> **a.** Distant static geometry and textures no longer needs to be in memory,
> since they've already been rasterized.
> => less culling, freed memory*, less rasterization
>
> **b.** Every fragment of the skymap just needs to be shaded once,
> and that's only if it hasn't been occluded by the real-time rasterization.

Note: The memory footprint of the loaded skymap data might be bigger than the freed memory.

It should still be compatible with complex environental shaders (e.g. day/night, fog, clouds, etc) and distant non-static geometry can potentially be occluded by it, if the depth value is baked.

(Potential) cons:

Requires a mostly static world, so it's not a good fit for any game that features dynamic terrain terraforming or has a large focus on building). Nor is it suited for games that rely on runtime procedural generation for the world itself. **Note:** Some seasonal dynamic elements ought to be possible with some sly data baking and interpolation during shading (e.g. for snow and vegetation).

Likely best suited for "grounded" games where the player generally is at ground level, since the camera that bakes will be at a fixed altitude. **Potential workarounds:** Some transform similar to a fish-eye effect that is based off of the player's altitude or increasing the radius of the "run-time rendering space" sphere as the height delta increases.

The biggest con would be memory footprint (both on disk and in VRAM) depending on the size of the world (=> number of cells), the resolution of the baked texture data, and the number of texture channels. But some channels would require less data per pixel and/or can be fine with lower resolutions, and any sky fragment would just be empty so compression would be handy.

Unknowns:

Since I’ve never implemented it and tested it, I don’t know at what distance radius r perspective stops having a noticeable influence when you compare the view rendered from the center of a cell with what is seen from any of the other positions in the cell’s space. Nor do I know how much interpolating the fragment from the four closed 2x2 cells’ baked data will resolve it or if interpolating the data might have unforeseen side effects resulting in artefacts. At this point, a lot of page two is just supposition, but that’s part of the reason I’m interesting in researching it to see whether it’s a viable technique or a dud. Perhaps the memory costs would be prohibitably high, since the G-buffers in deferred rendering are already expensive as is.

Other thoughts:

The scope of the idea might be overwhelming. But a few ideas:

The static “world terrain” during could be generated with some basic noise function or third party library, and for additional static geometry boxes ought to suffice. So setting up a test world shouldn’t be too hard.

As for testing, two test setups could be run:

> One that uses classical rendering (every fragments is from geometry that’s been rasterized and textured at runtime).
>
> One that has a much lower render distance but uses the baked skymaps during an extra render pass.

Then one could ask test participants to grade the fidelity of the two and compare the grades to the framerates of the tests perhaps. If this technique yields better performance or fidelity without the other being degraded, that would be a good indicator that it can be useful in practice.