Reflections have always been a captivating element in the visual design of video games. They add depth, realism, and immersion to the game world by simulating how light bounces off surfaces like mirrors, water, glass, or even wet streets. The complexity of creating reflections is often overlooked by players, yet it is a highly technical feature that involves various strategies, each with its pros and cons.

Over the years, game developers have experimented with different techniques to simulate reflections, each evolving with the advancement of hardware and software capabilities. The White Noise team, a prominent game development studio, has been at the forefront of utilizing different reflection methods in their projects, and they've shared their expertise to break down some of the most widely used reflection techniques in modern game development. Let’s dive deeper into these methods, exploring how they work, the technical requirements involved, and their respective advantages and disadvantages.

1. Planar Reflections: Simple and Direct

Planar reflections are one of the more straightforward approaches to creating reflections in video games. This technique involves rendering the entire scene again, but from the perspective of a virtual camera placed opposite the reflective surface, such as a mirror or a polished floor. Essentially, the level designers only need to create one room, and the reflection is handled by the additional camera, which captures the mirrored version of the scene.

How it Works:

A virtual camera is placed at the appropriate angle to mimic the reflection in a flat surface (such as a mirror or a glossy floor). The virtual camera renders the scene from this reverse perspective, and the resulting image is used to create the illusion of a reflection.

Pros:

Realistic reflections for flat surfaces: Planar reflections can achieve a high degree of realism on perfectly flat and simple surfaces, like mirrors or calm water.

Accurate reflections: Since the entire scene is re-rendered, the reflections can capture all objects, lighting, and shadows in the environment, ensuring high fidelity.

Cons:

Performance-intensive: Since the game engine must render the entire scene twice—once for the actual scene and once for the reflection—this approach can be very resource-demanding.

Limited to flat surfaces: Planar reflections only work well on simple, flat surfaces. Complex or curved surfaces do not handle this method effectively.

Not suitable for high-detail environments: For scenes with a lot of detail or complex geometry, planar reflections become too computationally expensive and impractical.

Use Cases:

Planar reflections are ideal for small, enclosed spaces like bathrooms with mirrors or calm water bodies where the reflection doesn't need to account for a large number of objects or high levels of detail.

2. Cubemaps: The Classic Approach from NVIDIA

Cubemaps were first introduced by NVIDIA in 1999, and the technology has been a staple in game development ever since. Initially used in PlayStation 1 games, cubemaps simulate reflections by mapping the surrounding environment onto the six faces of a cube. This technique was highly popular in early 3D games, especially in racing games, where developers used cubemaps to create the appearance of reflections on car surfaces.

How it Works:

A cubemap consists of six images—one for each face of a cube—capturing the scene from the reflective object's point of view. These images are then blurred slightly and applied to the reflective surface to create the illusion of reflection. While cubemaps are not true reflections in the sense that they don’t dynamically change, they are precomputed and mapped to static or semi-static surfaces.

Pros:

Low performance cost: Cubemaps are computationally inexpensive since they do not require real-time rendering. The reflection is essentially "baked" into the environment.

Simple implementation: This technique is straightforward and relatively easy to implement in most game engines, making it a common choice for many games.

Cons:

Static reflections: Since cubemaps are precomputed, they don’t work well with dynamic objects that move or change. The reflection remains the same regardless of what happens in the scene.

Not suitable for mirrors: Cubemaps are ideal for reflective objects like shiny cars or glass buildings, but they fall short when used for highly accurate mirror reflections.

Requires blurring to hide seams: Without proper blurring, players may notice the edges of the cube, which can break immersion.

Use Cases:

Cubemaps are often used in racing games, flight simulators, or any game where large reflective surfaces like glass skyscrapers or metal objects are involved. They are also commonly used in skies and environmental reflections that don't need to be dynamically updated.

3. Screen Space Reflections (SSR): Resource-Efficient but Limited

Screen Space Reflections (SSR) have become a popular technique in modern game engines due to their relatively low resource usage. This method relies on information already rendered in the current frame, specifically data from the G-buffer or Z-buffer. SSR traces rays in the screen space to simulate reflections, meaning it can only reflect what is visible on the screen at that moment.

How it Works:

SSR captures reflections using the screen space of the game. It traces rays from the reflective surface into the depth buffer (Z-buffer) and normal buffer (G-buffer) to calculate the reflection. However, since SSR only uses the information available on the screen, it cannot reflect anything outside the camera's view.

Pros:

Low resource usage: Since SSR relies on data already rendered in the current frame, it is much less performance-intensive compared to other techniques like ray tracing or planar reflections.

Works well for certain surfaces: SSR can effectively simulate reflections on flat or semi-reflective surfaces, like wet streets, water, or shiny floors.

Cons:

Limited scope: SSR can only reflect objects that are currently visible on the screen, which means it cannot account for objects behind the player or outside the camera’s view. This limitation can lead to visual artifacts or incomplete reflections.

Not suitable for complex scenes: SSR works best when combined with other techniques, such as cubemaps, to compensate for its limitations. Using SSR as a standalone technique often results in inaccurate or incomplete reflections.

Use Cases:

SSR is commonly used to simulate reflections in puddles, wet streets, or shiny floors in first-person or third-person games. It's often employed in combination with other techniques like cubemaps to create a more comprehensive reflection system.

4. Ray-Tracing: The Next Generation of Realism

Ray-tracing has become the gold standard for realistic reflections in games, thanks to its ability to simulate light rays bouncing off surfaces in real-time. Unlike other techniques that approximate reflections, ray-tracing calculates how light interacts with surfaces and objects, creating highly realistic reflections and shadows. This technique has gained significant attention with the introduction of hardware-accelerated ray-tracing on modern GPUs, such as NVIDIA’s RTX series.

How it Works:

Ray-tracing simulates how rays of light travel through the environment, bouncing off objects and creating reflections, shadows, and global illumination. This method traces the path of individual light rays and calculates their interactions with objects in the scene, resulting in highly accurate reflections and lighting effects.

Pros:

Unmatched realism: Ray-tracing delivers the most realistic reflections possible, accurately simulating how light behaves in real-world environments.

Dynamic reflections: Unlike cubemaps or planar reflections, ray-tracing handles dynamic objects and complex scenes without the need for additional tricks or approximations.

Next-gen technology: As ray-tracing becomes more widely supported by next-gen consoles and powerful GPUs, its implementation in games is expected to increase.

Cons:

Extremely resource-intensive: Ray-tracing requires a significant amount of computational power, making it difficult to run on older hardware or mid-range systems. It is still resource-heavy even with optimized hardware.

Requires high-end hardware: Only next-gen consoles and modern GPUs (like NVIDIA’s RTX series) can effectively handle real-time ray-tracing.

Use Cases:

Ray-tracing is increasingly being used in AAA games to deliver next-level realism in reflections, shadows, and global illumination. Titles like Control, Cyberpunk 2077, and TwinSoul have showcased the impressive capabilities of ray-tracing in creating highly immersive game worlds.

Conclusion

The creation of reflections in video games is a multifaceted challenge that requires balancing realism with performance. From the classic approach of cubemaps to the cutting-edge technology of ray-tracing, each technique offers different advantages depending on the requirements of the game. While planar reflections and SSR provide resource-efficient solutions for simple reflections, ray-tracing delivers the ultimate level of realism but at a steep performance cost. In many modern games, developers combine several of these techniques to achieve the best possible results, blending the strengths of each method to create immersive and believable worlds.