To start, I will compress the images to WebP. WebP compression presents a range of compelling advantages for web and game development. Firstly, it achieves significantly smaller file sizes compared to traditional image formats like JPEG and PNG, resulting in faster loading times for web pages and games, particularly beneficial for users on slower internet connections or mobile devices. Additionally, WebP supports both lossless and lossy compression, allowing developers to choose the appropriate compression level based on image content and quality requirements. This format also supports alpha channel transparency, making it a suitable replacement for PNG files with transparency information, and enables the creation of images with transparent backgrounds or soft edges. Furthermore, WebP supports animated images, akin to GIFs but with smaller file sizes, making it useful for incorporating lightweight animations or interactive elements. Despite its smaller file sizes, WebP maintains a high level of image quality due to modern compression algorithms, resulting in visually pleasing images with reduced artifacts. Most modern web browsers natively support WebP, ensuring broader compatibility and allowing a wider range of users to benefit from optimized images.

Additionally, we create texture atlases to combine multiple small textures into larger ones, reducing draw calls and memory usage. Implementing mipmapping helps optimize rendering performance by generating pre-scaled versions of textures at different levels of detail, based on their distance from the camera, By "camera," I mean the one in the Unity game engine. The approach for loading 200 8 MB images will depend on the client's preference for implementation. They can choose to implement it as a card game where these textures will be loaded into a GUI, or they can load them into an actual game space, allowing players to manipulate the images in a 2D or 3D environment rather than just drawing on them directly. To prevent noticeable lag during gameplay, asynchronous loading is employed for large images, allowing them to load in the background. This ensures a smoother gaming experience for players. We can also use Unity's Streaming Assets feature to load assets directly from the filesystem at runtime, speeding up the initial loading process. To further enhance performance, we implement Level of Detail (LOD) techniques, providing different versions of images based on the camera's distance. Caching is introduced to store loaded images temporarily, reducing loading time when players access the same image multiple times. Finally, image streaming and progressive loading ensure that low-resolution versions are loaded first, followed by higher resolutions during gameplay, optimizing image loading on-demand. Regular profiling and monitoring help identify and address any performance bottlenecks, ensuring a seamless gaming experience for players on various devices and connections.

Memory and storage management are vital components of computer systems and game development, each serving distinct purposes. Memory management revolves around handling a computer's primary memory, or RAM, which is volatile and used to store data actively accessed by the CPU during program execution. Its primary goal is to efficiently allocate and deallocate memory for running programs, ensuring optimal performance. Memory management is concerned with short-lived data, which is cleared when the computer is powered off or restarted. It involves tasks like allocating memory for variables and data structures, managing assets like graphics textures and audio buffers in real-time games, and implementing garbage collection to free up unused memory.

On the other hand, storage management deals with non-volatile, long-term storage devices like hard drives or solid-state drives, responsible for retaining data even when the computer is powered off. It focuses on organizing, storing, and retrieving data for long-term use, such as files, databases, and other data structures. Unlike memory management, storage management deals with data that has a longer or indefinite lifetime. It impacts data access times and overall system performance, especially in tasks like loading game assets or saving game progress. Examples of storage management include managing files, directories, and permissions, implementing data persistence for game progress or user settings, and using databases to store game-related information like high scores or player profiles. Both memory and storage management play crucial roles in creating efficient and reliable game experiences, ensuring smooth gameplay and optimal data handling.