Technical Art Requirements 0 A.D.

Wildfire Games

December 2, 2021

Introduction

Art content sets the style of the game and makes it beautiful, but at the same time it significantly affects performance and storage space. So we always have to balance between visual quality and technical limitations to make the game attractive and enjoyable.

When you're not sure does a guideline rule is applicable for you or not ask a question on the forum or ask some competent programmer.

Textures

Texture formats and sizes

The game engine supports two common texture formats: **DDS** and **PNG**. All new textures should be PNG by default, the engine will automatically convert them to DDS.

The engine supports only textures with a size of power of two: 4, 8, ..., 1024, 2048. Maximum size is 2048. It's allowed to have different width and height, like 1024×4.

A texture screen density should be accounted for the texture size choice. If an object covers a small part of a screen (like a flower), then the object doesn't need a big texture.

Do not use 64-bits textures, they will loose their precision anyway but will cost much more space in a repository.

Ambient occlusion textures

Ambient occlusion (AO) textures should not have noise. Noise increases sizes of PNG textures and adds artifacts to DDS textures. If you don't see any noise in the game but only on a texture, then you use a too big texture and you have to reduce its size and remove the noise.

AO textures should not have alpha channel.

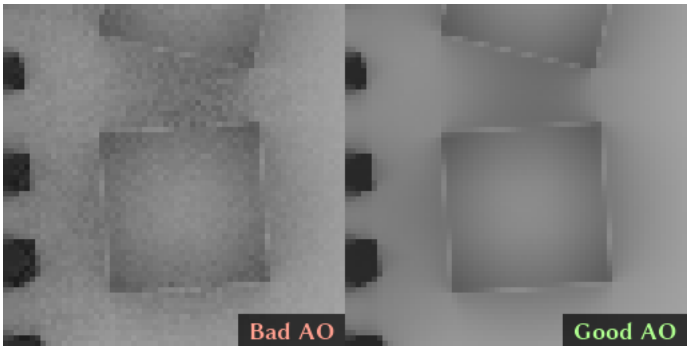

Figure 1: Ambient occlusion textures.

Normal maps

Normals in the game engine are encoded using a classic method:

$$(Pixel_{Red}, Pixel_{Green}, Pixel_{Blue}) = (Normal_X * 0.5 + 1, Normal_Y * 0.5 + 1, Normal_Z * 0.5 + 1)$$

You have to avoid the following cases which might add noise and/or artifacts:

- item normals lying on the surface, example $Normal = (1, 0, 0), Pixel = (255, 127, 0)$
- item neighbour pixels with opposite directed normals, example $Normal1 = (1, 0, 0), Normal2 = (-1, 0, 0), Pixel1 = (255, 127, 0), Pixel2 = (0, 127, 0)$

Geometry

There is a good generic guideline about low-poly geometry.

Vertex and triangle count

The game engine doesn't support meshes with more than 65536 vertices. It means a mesh with 3 unique vertices per triangle might have at most 21845 triangles. Reusing vertices might increase the maximum number of triangles.

You should avoid spawning many small meshes, especially if they have less than 128-256 vertices. It's better to pack them into a single mesh.

Invisible faces

You should avoid invisible faces, if a player can't see a face in the game (during a usual gameplay or cinematic scene), then you have to remove it. For example you don't need to have 5mm details on helmets.

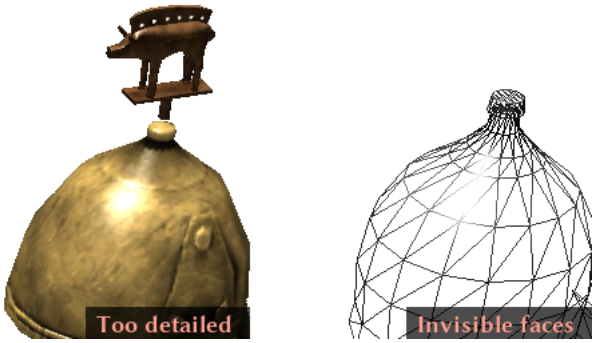

Figure 2: Invisible faces for helmets.

Transparent meshes

Avoid using transparent materials for meshes. If you can add some (5% - 10%) triangles but use opaque materials then use opaque materials.

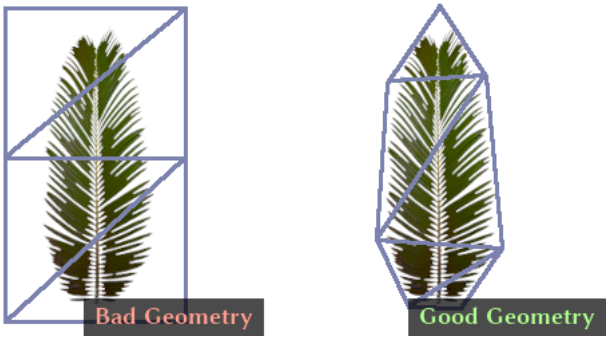

Figure 3: Geometry for transparent meshes.

Bones

For a good performance the number of bones should not exceed 32 bones for a single mesh. More bones means worse performance and bigger memory usage. The engine supports at most 64 bones per a mesh.

Materials

Parallax mapping

Parallax mapping is allowed only on big and flat surfaces. Normals of these surfaces (of all their vertices) should be co-directed (look in the same direction) and be perpendicular to the corresponding surface.

Figure 4: Usages of parallax mapping.

References

- https://docs.unrealengine.com/4.27/en-US/TestingAndOptimization/PerformanceAndProfiling/Guidelines/
- https://developer.arm.com/solutions/graphics-and-gaming/developer-guides/game-artist-guides
- http://fragmentbuffer.com/gpu-performance-for-game-artists/
- https://docs.cryengine.com/display/SDKDOC2/Rendering+Performance+Guidelines
- https://blog.unity.com/technology/artists-best-practices-for-mobile-game-development