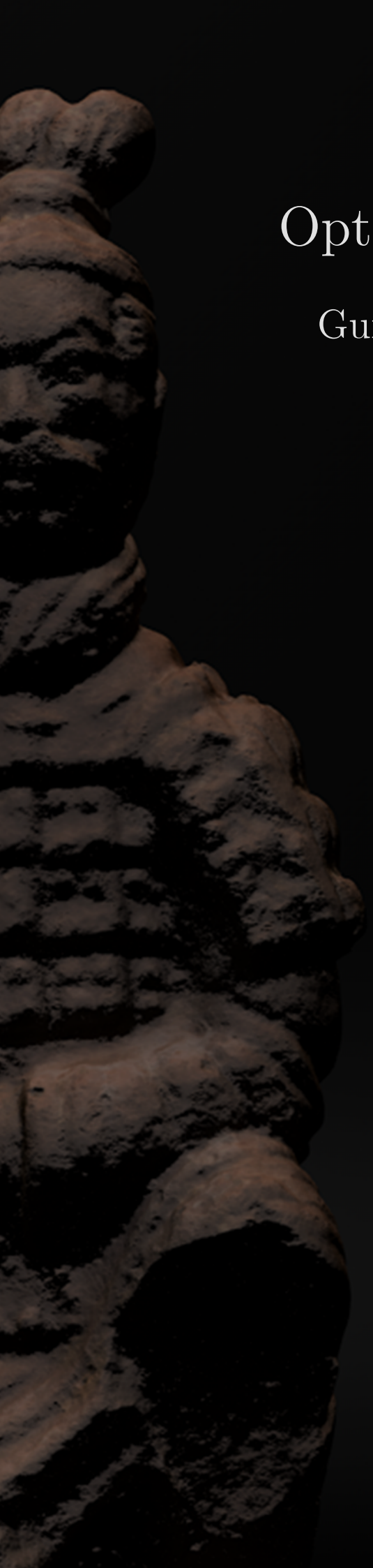


Optimization for Unity

Guidebook originally made for the game Refuge

Author: Nikola Kunzova

Introduction

At first this handbook was supposed to be made as a guide for Unity optimization for the game Refuge. As I dived in into the research its focus spread widely.

On the next pages can be found general information about code, models and textures optimization. Last section is for tutorials that shows you some optimization techniques.

This is just a beginning. I did not include many interesting information. Some updates, changes and new tutorials will be added into this handbook later.

For now, this is all you get.

Have fun.

Contents

1 General Scene Set-up

1.1 Good General Tips

- **200,000 to 3,000,000 polygons for the whole scene (desktop)** *(Source of numbers and tips below.)*
- Reducing amount of memory: reduce the complexity of objects and resolution of textures
- Keep the number of different materials per scene low, and share as many materials between different objects as possible.
- Bake lighting rather than using dynamic lighting.
- Use compressed texture formats when possible, and use 16-bit textures over 32-bit textures.
- Avoid using fog where possible.
- Use Occlusion Culling to reduce the amount of visible geometry and draw-calls in cases of complex static scenes with lots of occlusion. Design your levels with occlusion culling in mind.
- Use skyboxes to “fake” distant geometry.

Found softwares, that can help with the 3D content optimization:

- *Simplygon*

1.2 Take steps to improve the performance

Let’s start with the basics:

- Analyze your game
- Analyze the profiler data
- Make a change
- Profile the effect of that change

These are the common reasons that can make a single frame of your game slow:

- Trying to do too much (obvious)
- Bottlenecks
  - CPU-bound: the CPU takes too long to execute his tasks.
  - GPU-bound: the GPU takes too long to execute his tasks.
    - These information can be found in Game stats window; CPU & GPU time (CPU time and render thread) to detect CPU/GPU bound

How can you analyze your game? Unity offers some instruments, like **the profiler**, which will help you to understand what happens in each frame of your game.

1.3 Profiler

You can find the Profiler under Window>Profiler and it will run when you play your game.

Some tips for using the profiler:

- **Do not use it directly in the Unity editor because this can add unwanted information.** Profile your game directly from the (development) build in order to have only the information strictly connected to the game (e.g. if you're making a mobile game, try to profile it directly from your mobile device in order to have the most accurate results)
- **You can see exactly which function is taking too much time** - try to understand the function and find out what is causing the problems
- You can turn on the deep profiler options which will give you even more information of what operations are slowing down the game (the downside: deep profiler only works in editor mode)

If you are interested in using Profiler, check this documentation on UnityDocs: *Official Manual for Profiler.*

1.4 Static and Dynamic Batching

Batching makes sure that no unnecessary Draw Calls are used. Batching comes in two flavours: Static and Dynamic.

1.4.1 Static Batching

Static gives you the best performance, so we always try to use Static Batching.

- Have as few different materials as possible (combine all your materials in one big texture).
- Requires you to set the flag "Static" in the Object Properties panel.
- It can only be used on objects that do not move, rotate or scale in the scene.

1.4.2 Dynamic Batching

It is always active on objects that are not static. Dynamic Batching is very useful for star pickups and other small objects that are animated but are otherwise the same in the scene.

Some useful tips from the Unity manual:

- Batching dynamic objects has certain overhead per vertex, so batching is applied only to meshes containing less than 900 vertex attributes in total.

- If your shader is using Vertex Position, Normal and single UV, then you can batch up to 300 verts and if your shader is using Vertex Position, Normal, UV0, UV1 and Tangent, then only 180 verts.
- Don't use scale. For example, Unity will not dynamically batch a cube with scale (1,1,1) with another cube with scale (2,2,2).
- Uniformly scaled objects won't be batched with non-uniformly scaled ones. Objects with scale (1,1,1) and (1,2,1) won't be batched. On the other hand (1,2,1) and (1,3,1) will be.
- Using different material instances will cause batching to fail.
- Objects with lightmaps have additional (hidden) material parameter: offset/scale in lightmap, so lightmapped objects won't be batched (unless they point to same portions of lightmap)
- Multi-pass shaders will break batching. E.g. Almost all unity shaders supports several lights in forward rendering, effectively doing additional pass for them
- Using instances of a prefab automatically are using the same mesh and material.

More information about batching can be found on UnityDocs: *Draw Call Batching.*

1.5 Code Related Improvements

- General
  - Avoid foreach & LINQ (because of the garbage collector)
  - Avoid too much "new" calls, i.e, object/memory allocation (GC problems)
- Materials
  - If you use a string to get or set properties like GetFloat, SetFloat, GetTexture, SetTexture, etc. on materials and shaders, that property will be hashed (the string will be mapped into an integer). Instead, directly use an integer ID to access them.
  - Don't use transparent textures when not necessary, as it will cause *fill-rate problems* (fillrate = screen pixels * shader complexity * overdraw). When you need to use it, prefer alpha blended shaders instead of shaders with alpha testing or instead of cutout shaders for mobile platforms.
- Right Data Structures
  - If you iterate a lot through a List, use an Array instead.
  - If you need to constantly add or remove objects, use a Dictionary or an HashSet

  - If you're mostly indexing by keys, use a Dictionary
  - Iterating through a HashTable or a Dictionary is expensive because the compiler has to check all the elements, instead use a struct or a tuple, and store a list or an array of that struct or that tuple.
  - Won't allow duplicates: Dictionary or HashSet (we can have character run twice as fast)
  - Generally constant-time insertion: List, Dictionary, HashSet
  - Need low-overhead iteration: Array or List
- Object Pooling
  - Pooling commonly used objects allows you to reuse them over and over again without destroying them.
  - This is useful for objects that are used very frequently.
- Enable Instancing
  - On the Standard material you'll see a checkbox at the very bottom under Advanced Options that says Enable Instancing.
  - Check that on and you'll see your draw calls drop drastically for all objects that utilize that same mesh and material.
  - Instancing does not always improve performance.
- Update Loops
  - Don't use performance intensive things in update loops, use caching instead.
  - Cache everything in Awake() or Start() methods, or change your architecture to a more event-driven approach to trigger things just when they're needed.

1.6 Other Improvements

- **Culling**: use to reduce what is rendered (Frustum Culling - by default automatically, Occlusion Culling - need to be set up, manual)
- **Raycasts**: Find only what you need when using a Raycast. Don't use multiple rays if one will suffice and don't extend it past the length that you need it to travel. Great tip is to utilize objects into layers.
- **Bake the Lights**: You should use the minimum amount of lights necessary to achieve your desired style. Baked lighting can and should be used whenever possible. This allows you to add lighting to your world while giving you the performance benefits of not having to calculate dynamic light at all times.

1.7 Great sources worth mentioning

- **The whole workflow of game making**: *What NOT TO DO in Unity during game development*.
  This article pinpoints all important mistakes that should not be present in the project during the development.

- **Tips**: *Unity Optimization Tips: Mobile & Desktop*.
  This complete Unity optimization guide will show you how to increase performance in Unity correctly, so your game will run fast & stable. It is mainly (in full detail) focused on Profiling and Benchmarking, but at the end are some other tips and tricks for optimization.

- **Mistakes**: *The 10 Most Common Mistakes that Unity Developers Make*.
  This article will provide advice on how to overcome most common problems and how to avoid fundamental mistakes in your new or existing projects.

- **Collection of links**: *Sites where you can learn how to optimize your project*.
  You will learn how to optimize your code in Unity and other tips and tricks as well as best practices.

- Understanding optimization in Unity *(Official)*.

2 Assets

2.1 Good General Tips

- Use a single skinned Mesh Renderer (mostly for bone animations)
- Use as few bones as possible (the fewer, the better - desktop game: 15 - 60 bones, 30 is optimum)
- Keep FK and IK separate
- Polygon count (desktop platforms: 1500 - 4000 per model (These numbers are rough approximation, all depends on conditions.)) (not just number of polygons, but influenced also by normals, UV coordinates and vertex colours)
- Airtight model (no gaps)
- Model's scale should be correct and baked into vertices, so the normal model transformation matrix has no scaling in it (usually 1 Unity Unit = 1 meter). To find out how to import objects to Unity check the 'Tutorial' section: **4.4**
- The fewer subobjects within the model, the better (moving parts separated only)
- Pivot influences transformations. Every object and its subobjects should have its pivot properly aligned and rotated with regards to its main function. The main object should have the Z axis pointing forward and pivot should be at the bottom of the object for better placement to the scene.
- Models from 3d scanning has many vertices - they need to be simplified
- Different LODs (Level of Detail) - adding LOD Component to game object
- Use Convex Mesh Collider for complex objects (non convex ones have bigger performance load and work just for static objects).

2.2 Level of Detail

How can we improve the performance of objects that are not being culled but are too far to see in detail?

Levels Of Detail (LODs) are a way to render a lower poly version of a mesh when it's outside of a certain range. A mesh that is quite far away can be made incredibly low poly to improve performance.

For more information check the UnityDocs: *Official Manual for LODs.*

To find out how to create LODs check the 'Tutorial' section: **4.1, 4.2**

2.3 Colliders

Use primitive colliders whenever possible. These are your basic shapes for colliders such as box, sphere, or capsule.

Mesh Colliders take the shape of whichever mesh that you indicate. This is incredibly expensive to use and should be avoided if possible. If absolutely needed, **create a low poly version of the mesh and designate that as your mesh collider instead**. Raycasting against mesh colliders is also quite expensive.

Possible replacement for mesh colliders could be **compound colliders**.

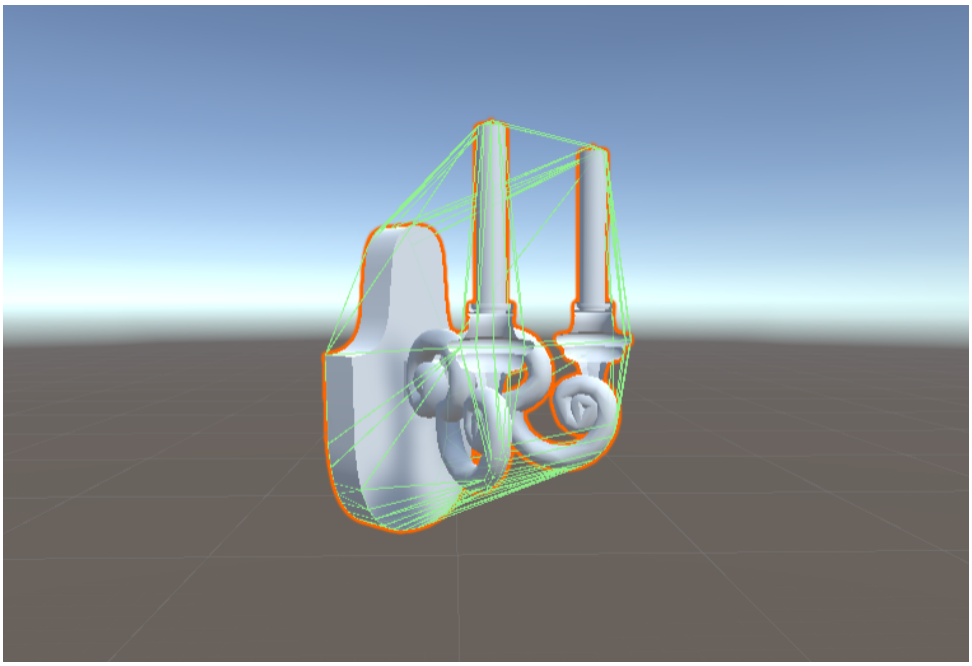


Figure 1: (Convex) Mesh Collider

3 Materials

3.1 Good General Tips

- Use as few materials as possible (more than one material only necessary when one needs to use more shaders; two or three materials should be sufficient for one object). If it has to switch between materials too often, it makes new draw calls and becomes inefficient.
- Clean UV unwrap for texture coordinates after reducing polygon count
- Lightmap UVs (vastly improve quality of baked light on static objects)
- Max size resolution of a texture can be reduced (it doesn't change original texture size, just import size)
- Normal maps resolution can be much lower than colour or albedo maps - less visible
- Using Texture Atlas (see 4.5)
- Using MipMaps

3.2 Texture Improvements

Since batching works based on like materials, you can combine many objects together if they share one big texture.

Multiple high-resolution textures will slow down performance. While you can have these in your game, you need to ensure that you're being selective about how they are being used.

3.2.1 Texture Atlas

- Use a Texture Atlas to combine multiple texture maps into one larger texture map.
- This not only helps to reduce the number of texture maps used, it also makes everything much easier to organize.
- Use of Megatextures and virtual texturing.

Tutorial for making a Texture Atlas can be found in the 'Tutorials' section: **4.5**

3.2.2 MipMaps

MipMaps allow textures to be reduced in resolution when far away from the camera. They can also be used if a lower end system is struggling to render a texture at the specified resolution.

MipMaps are enabled by default on textures that are imported into Unity and should be enabled unless you are using a camera with a fixed distance at all times and/or using your own unique tools to achieve better performance with your textures.

4 Tutorials

4.1 LODs: Blender

Source:

- Video: How to use the decimate modifier
- Video: How to create level of detail in Blender for Unity

Step-by-step:

- Naming conventions of leveled models: name_LOD[X] (X is a number, e.g.: tree_LOD0, tree_LOD1...)
- Apply a modifier called 'Decimate' on a high poly mesh (flat shading mode)
- Select category 'Un-subdivide' and increase the number of iterations to get meshes with less polygons
  OR
  Select category 'Collapse' and use decimate ratio (it seems that decimation preserves UV coordinates):
  - LOD0 = 100%
  - LOD1 = 65 - 70%
  - LOD2 = 45 - 50%
  - LOD3 = 25 - 30%
  - LOD4 = 12 - 18%
- Apply modifier on all leveled objects when you are satisfied with LODs
- Check all meshes if everything looks fine
- Export: set visibility to all of them, select the meshes from bottom up (from the lowest poly to the highest poly mesh), export as .fbx just selected meshes (no naming convention for the name of the export file)
- Import: place the file into the assets and place it into the scene, Unity recognizes automatically it is a LOD object

4.2 LODs: Cinema 4D

Source: -

Step-by-step:

- Naming conventions the same like in Blender

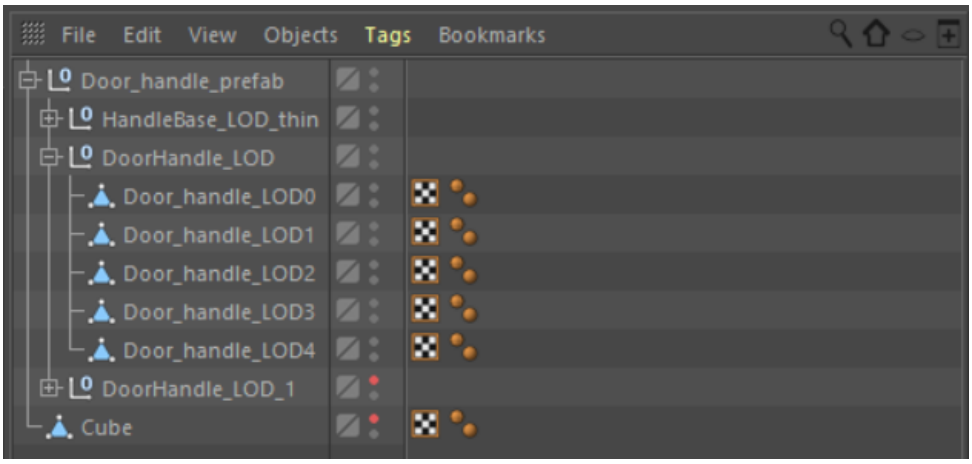


Figure 2: Naming Convention

- To create different levels of detail, we use polygon reduction deformer
- We make the polygon reduction deformer a child of the model and set these parameters:
  - Reduction Strength
  - Mesh Quality Factor
  - Co-Planar Optimization
  - Boundary Curve Preservation
  - Polygon Quality Preservation

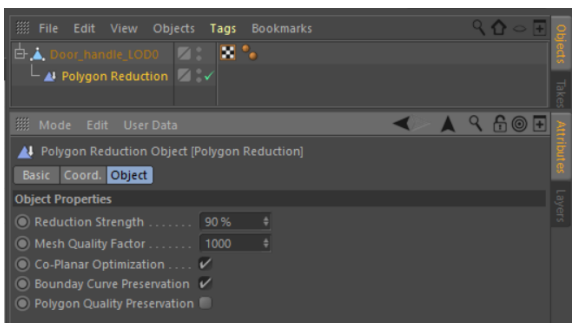
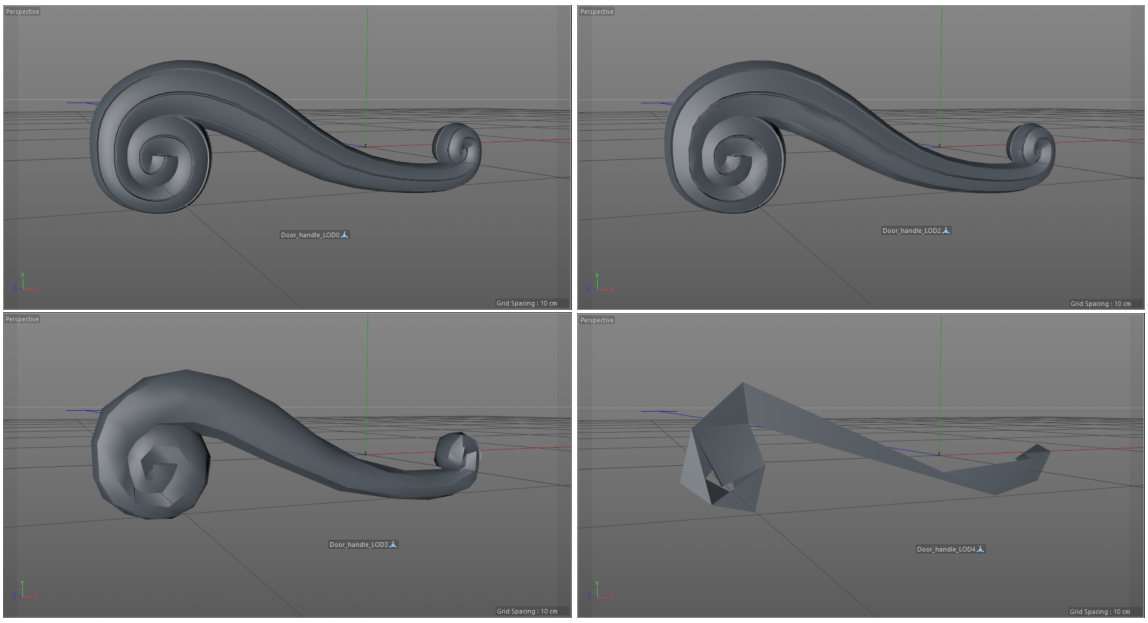


Figure 3: Polygon Reduction Deformer

- To get the new model, right-click on the current model and choose ‘Current State to Object’ / Select all children of the object, right-click on the object and choose ‘Connect Objects + Delete’
- Make few levels of details and use them in the game



- Export the same like in Blender

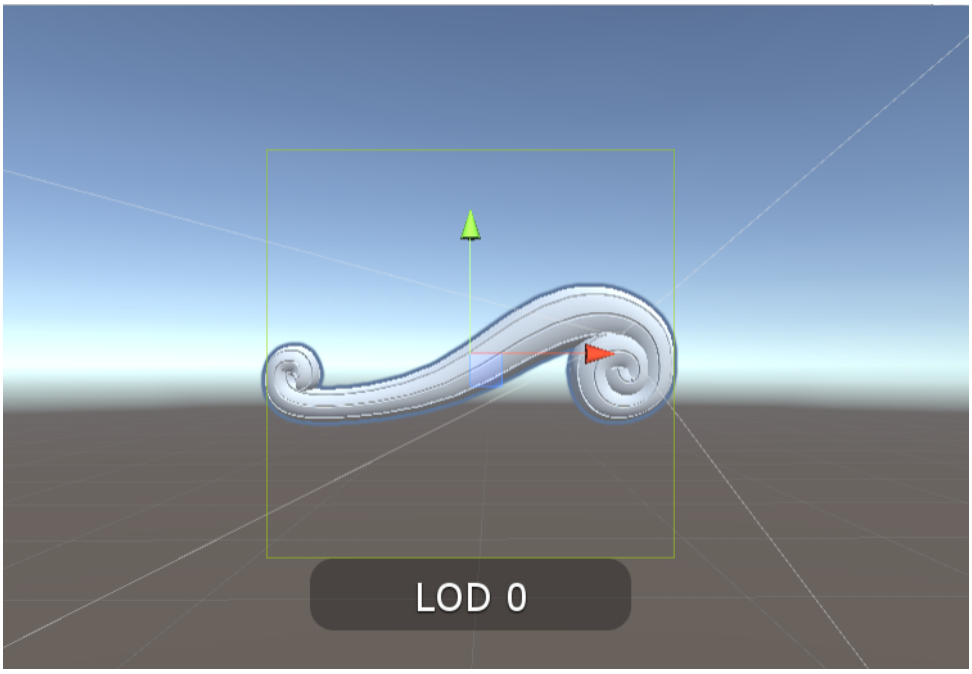


Figure 4: Imported Model_LOD0

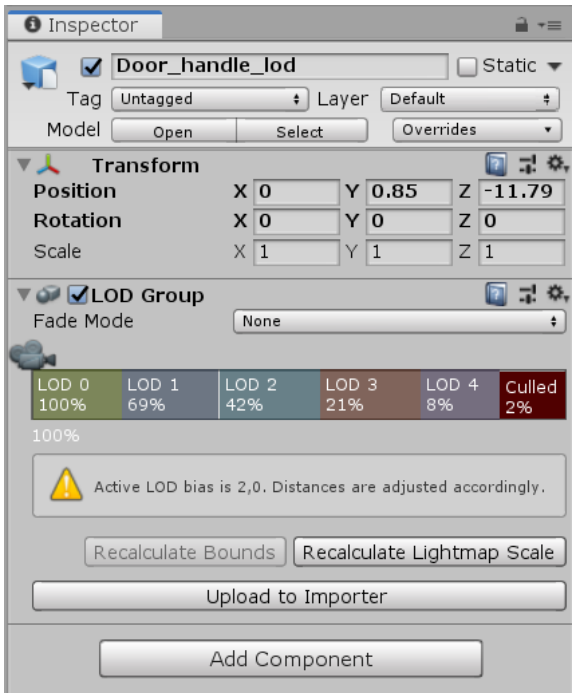


Figure 5: Attributes of imported LODs model

4.3 Import to Unity: Blender

Source: Blender 2.8 to Unity Export - Correct Scale/Rotation

Make sure that in Blender, your item is facing the Negative Y direction.
Negative Y in Blender becomes forward (Positive Z) in Unity.
Make sure that your Location, Rotation, and Scale are all applied.
In Object Mode, hit Ctrl + A and apply them as needed.

Two options:

- OPTION 1 - APPLY TRANSFORMS ON EXPORT
  - If you have a simple mesh, this works, but it may start breaking if you have parenting relationships like with armatures.
  - **Choose !EXPERIMENTAL! Apply Transform when you export to .fbx**
  - Using the 'File - Export - FBX (.fbx)' menu, select the checkbox for Apply Transform. This will work in many cases.
- OPTION 2 - COUNTER-ROTATE THE OBJECT, THEN EXPORT
  - Rotate the object -90 degrees on the X axis
  - Apply the rotation
  - Rotate the object +90 on the X axis
  - **Do NOT apply the rotation**
  - Export with FBX Units Scale

4.4 Import to Unity: Cinema 4D, 3ds Max, Modo

Source:

- Forum: Problem with dimensions

Step-by-step:

- Unity natively imports a .c4d file from C4D, it is enough to place the file to project's Assets folder (automatically updated when the original file is edited). Info: there cannot be found any information if you must have Cinema4D installed in the computer or not.
- Size: It is necessary to set size in Cinema 4d to meters. Then we can use this conversion: 100 c4d units = 1 unity unit (unity unit is meter). During import to Unity the model goes through a scaling converter, that scales is down to 0.01 of its height). That means that a cube in C4D with side long 200m will be in Unity a cube with side long 2m.

4.5 Texture Atlas

A texture atlas is just a texture. That's all. There is nothing special on a texture atlas. It's just one big texture with several textures usually arranged in a grid.

First option: Using Unity's function *Texture2D.PackTextures*
Source: *What is the simple way of applying Texture Atlas to Unity*

- Texture2D.PackTextures is just a helper function. You provide a texture array with all your textures you want to pack.
- The function pack all those textures into one texture and retuns a Rect for each texture.
- This rect have to be used to adjust the uvs of each vertex in your mesh. Basically your uv coordinates are usually between 0 and 1 in each axis. After the packing they have to be remapped to the rectangle. *For example if PackTextures returned a rect like this: Rect(0.2,0.4,0.2,0.2) the new area for the texture is not 0 to 1 and 0 to 1, it's 0.2 to 0.4 and 0.4 to 0.6.*
- You just need to multiply each uv with the rects size (of course seperately for each axis [x,y] or [u,v]) and add the rects position after the scaling. This have to be done with every vertex in a mesh.
- Tiling within an atlas is usually not possible. There are some shader tricks, but they aren't very efficient.

Second option: Using Unity's extension - MA_TextureAtlasser
Source: *YouTube Tutorial - Texture Atlas Pro*

This plugin creates an interactive environment for Texture Atlas making.

You can combine textures and automatically remap the UV's for the 3D models. By having full control over the size and position of the textures that are being placed in the atlas you will never stand for surprises when exporting. This will cost some more time than auto-generating your texture atlases but you know whats going on and which models/textures are getting priority.

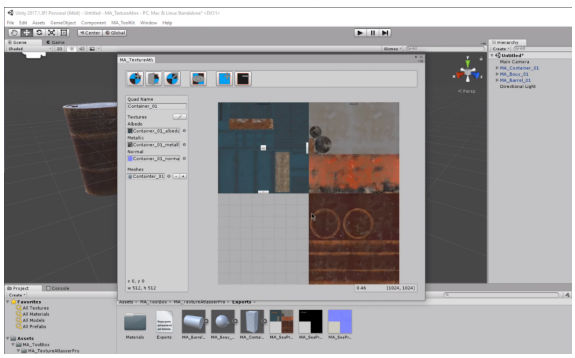


Figure 6: Preview of the Texture Atlasser

4.6 Photogrammetry Post-process

Photogrammetry workflow for Unity is described in detail in this handbook: *Unity Photogrammetry Workflow*.

We are interested in these stages:

- Create a Low Resolution Mesh for Baking.
  Video tutorial in 3dsMax, but contains universal practices: *Retopology of High Poly Model*.
- Textures Baking
- Remove lighting
- Tileable Material
- Create a Game Asset

4.7 Potentially usable tutorials

These problems were mentioned, but tutorials were not made (can be fixed..):

- Dynamic Batching in Detail
- Reduction of Polygons in walls in one plane

5 References and External Links

Click on the titles to be redirected to the sources:

- *How To Optimise And Increase Performance In Unity Games*
- *4 Ways to Increase Performance of your Unity Game*
- *Maximizing Your Unity Game's Performance*
- *The 10 Most Common Mistakes That Unity Developers Make*