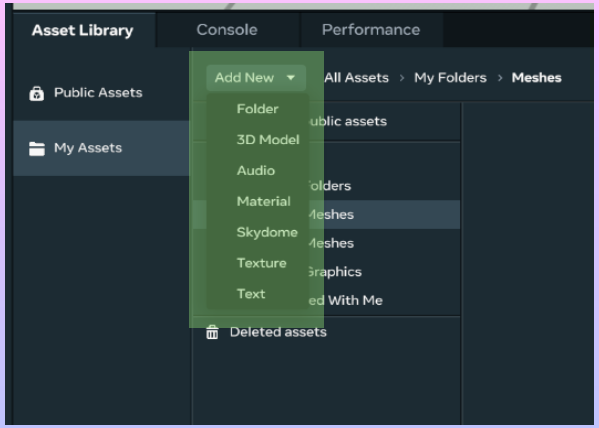
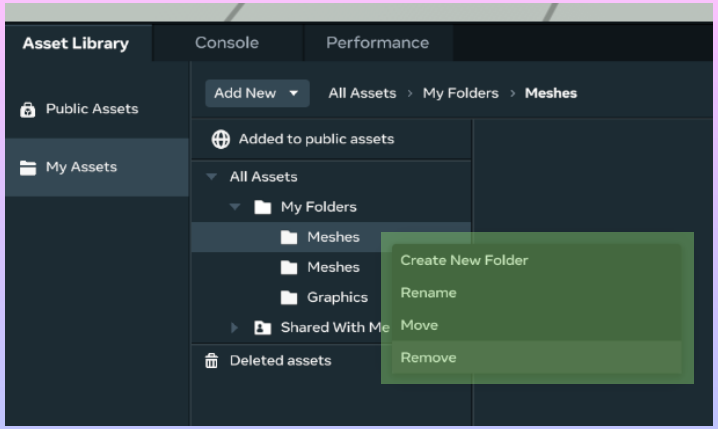
META HORIZON WORLDS DESKTOP

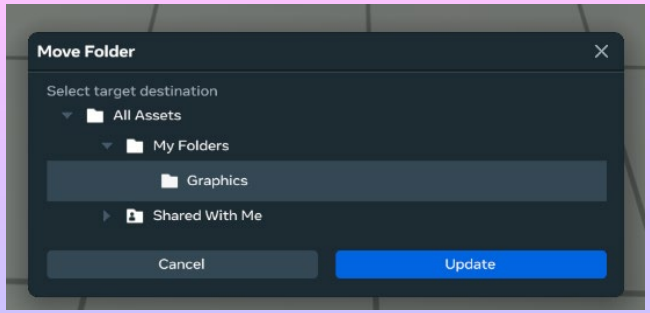
Documentation for artists

FOLDERS

The very first thing to do when creating a new project is to create a project with the name of the game: since everything is online in Horizon, all the assets we import will be shared across all our different projects.

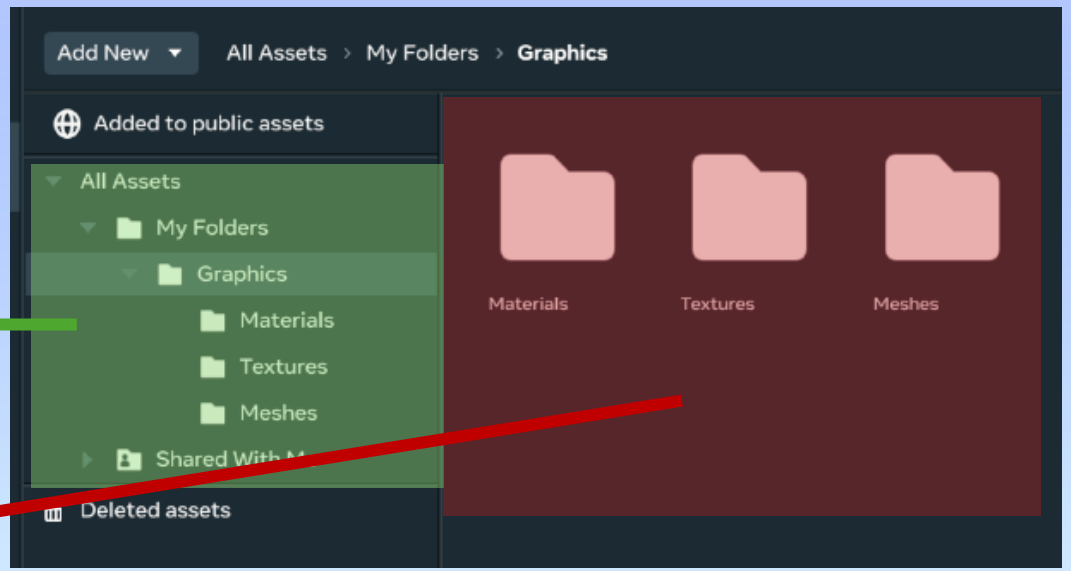






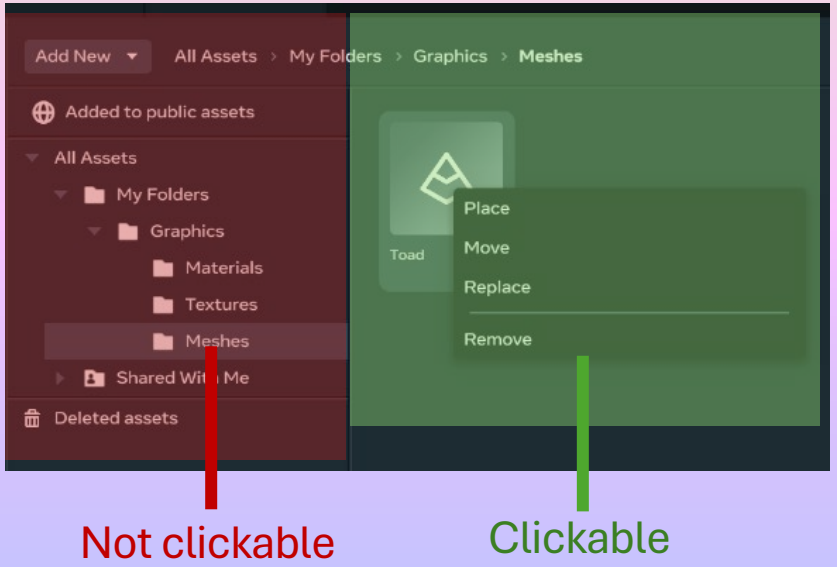
For anything related to folders, right-click to access the settings in the list view window on the left (it won't work on the one on the right).

To create, delete, or move folders, everything must be done through the Inspector; no shortcuts work.

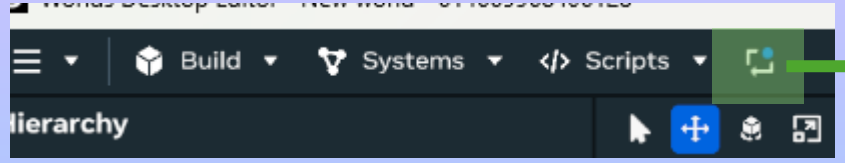


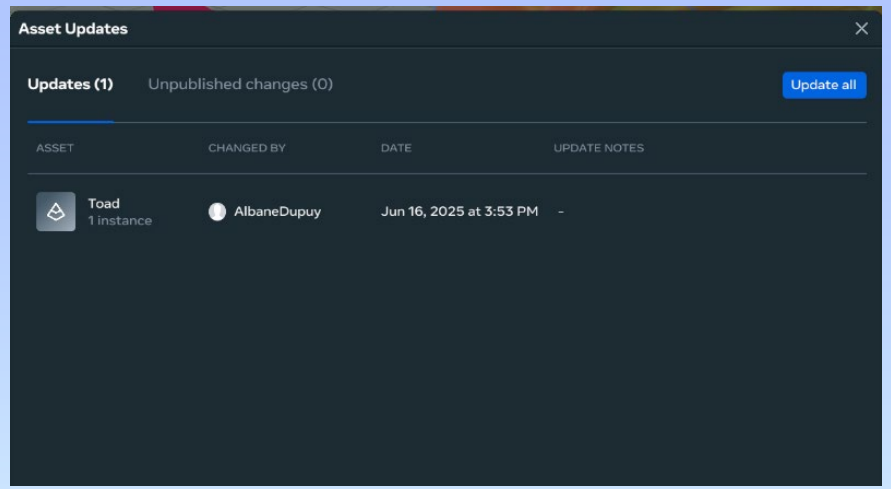
REPLACE

To modify assets, it's the opposite of folders: you can only click on them in the window on the right. Re-importing is called 'Replace'.

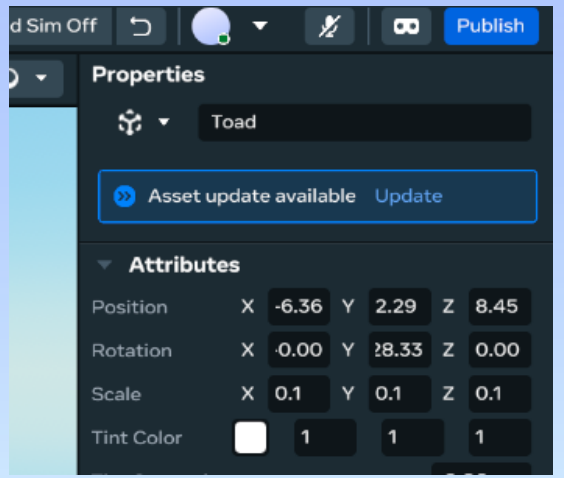


The asset will load, and once it's ready, it won't automatically update in the scene. You either click on the small update icon at the top left, which opens the window with all the assets that can be updated,



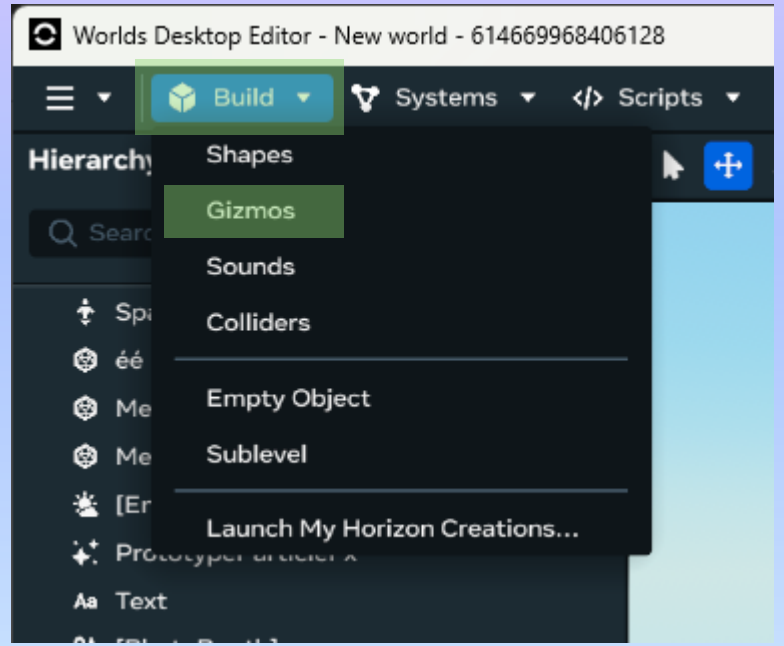


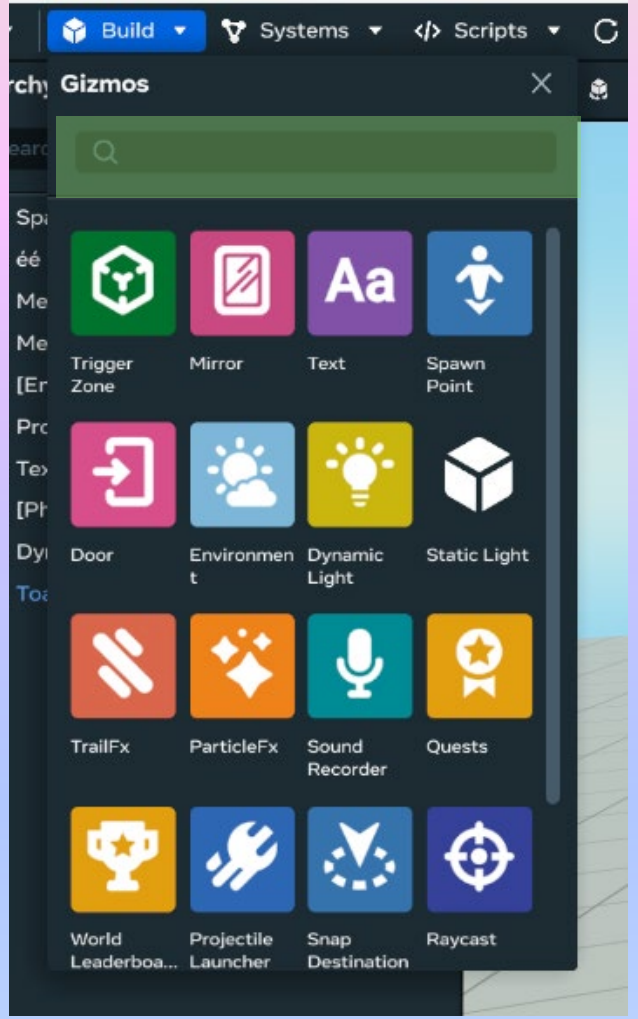
Or you can click directly on the object in your scene, and in its Properties panel it offers the update option.



ADD ACTORS

To add any actor to the scene, you need to go to the 'Build' window and open 'Gizmo'.





I believe the list of available Gizmos is exhaustive, but if you can't find something, you can always use the search to go faster (though you need to know the exact name).

MESHES

Link of the best practices doc

When modeling for Horizon, there are several things to keep in mind:

First, the documentation clearly states that the recommended format for 3D objects is FBX; any other format won't work.

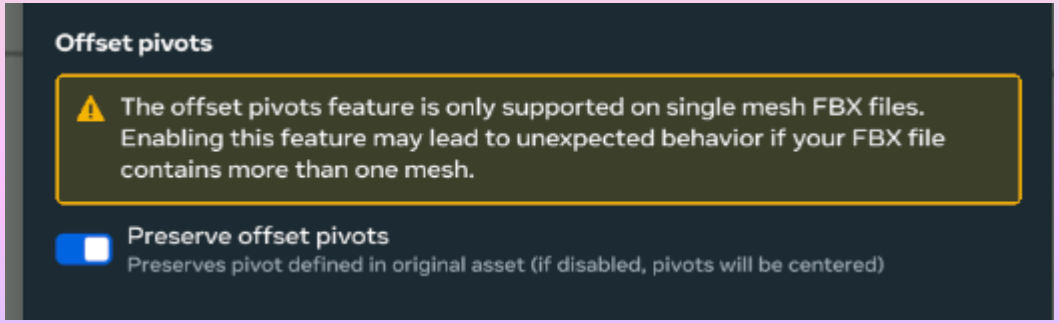
	Recommended
Formats	FBX

Next, open edges are supported, but double-sided materials are not. This means that if you create an extremely thin object (like a leaf, for example), you either accept that it will only be visible from one side, or you need to model both sides for it to be rendered correctly (which will more than double the polycount).

Finally, it seems that the same issue we sometimes encounter in Unity exists here: the model is at the correct scale, the scales are set to 1, and freeze transform/delete history have been applied, yet when importing, the mesh has the correct size but is scaled to 0.01. They don't provide a solution for this case, but the warning is mentioned in the documentation, so I think it's important to point out.

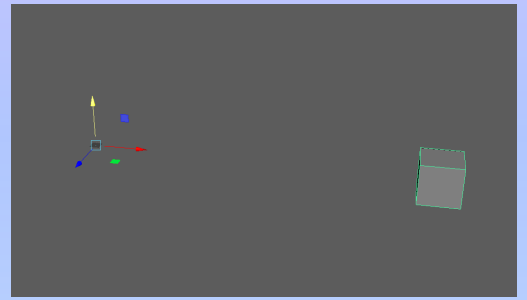
Maya - The is a known scale issue where models will come in at the correct size but will have their transforms set to 0.01 scale.

PIVOTS

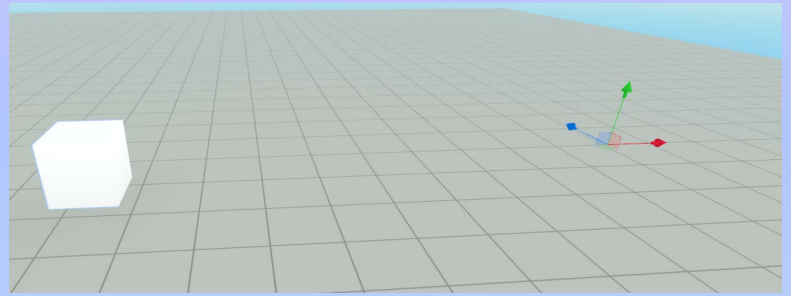


When importing a mesh, there is a boolean option you can uncheck (it's checked by default). Essentially, it lets you keep the pivot point you placed or automatically center it on the object.

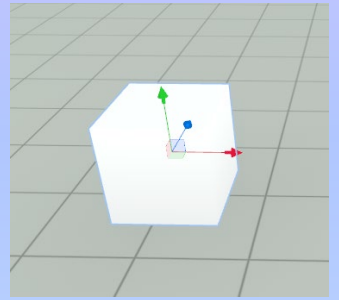
For example, in Maya I have this:



If I leave it checked, my pivot point will remain at the distance where I placed it from my cube.



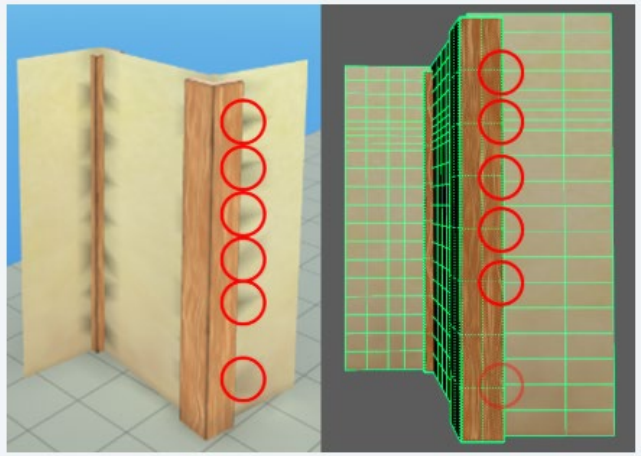
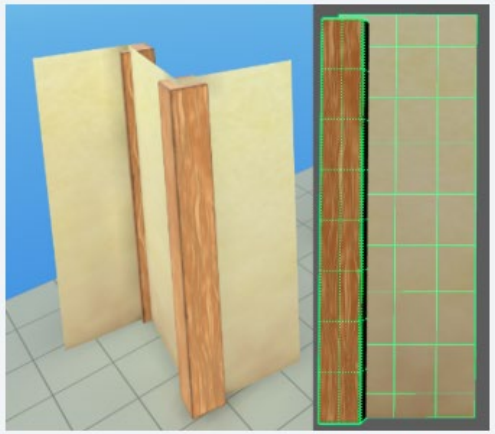
However, if I uncheck it, it will automatically center on my cube.



PS: As the yellow warning says, preserving the pivot only works if you import meshes one by one. You can import multiple FBXs at once, but then all the pivots will be centered.
Think about it, also for meshes that need a custom colliders (see slide about custom colliders). As it counts as 2 meshes (the mesh and the collider mesh), you won't be able to place your pivot anywhere else than centered.

MESHES AND BAKED LIGHTING

Link of the best practices doc

Topology creating GI artifacts	GI artifacts fixed with topology adjustment
	
Topology on custom models will sometimes produce artifacts when GI is calculated. This will cause extremely dark vertices when imported into Horizon. They will occur when a vertex is close to or snapped to another vertex or edge on a different contiguous mesh.	To remedy this when working with multiple contiguous meshes, place vertices toward the middle of intersecting faces. The image below shows an adjusted model, and no GI lighting artifacts.

When working with static meshes (where lighting will be baked), you need to plan the topology accordingly. Light baking in Horizon relies on vertex colors, so on the vertices.
The documentation states the following: having vertices that are too close together or snapped onto other vertices can cause lighting artifacts. You need to ensure that each vertex respects a 'safe distance' to avoid this kind of problem (the exact distance isn't specified, but basically, if you encounter lighting artifacts, keep this in mind).

- 512 textures use 8 pixels.
- 1024 textures use 16 pixels.
- 2048 textures use 32 pixels.

As a result of vertex-based baked lighting, there are recommended UV padding guidelines to prevent bleeding.

LIGHTING

Link of static light doc

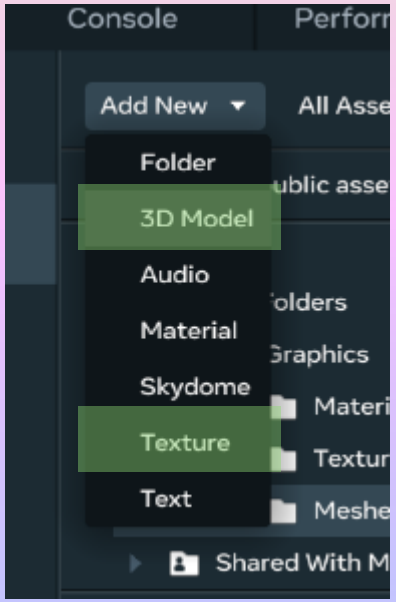
There are two types of lights: static and dynamic. You can't switch directly between them, as they are two different gizmos.

Comparison of the two types of lights:

Type de light	Shapes	Paramètres	Cas d'utilisation
Static	• Cuboid • Elipsoid • Disk • Rectangle	• Activation • Color • Intensity • Distance (scale of the light)	If you just want to light areas in the scene, without the light needing to move during gameplay.
Dynamic	• Spot • Point	• Activation • Color • Intensity • Falloff	If you want to light areas and the light needs to move during gameplay, or if we need to setup a quick lighting.

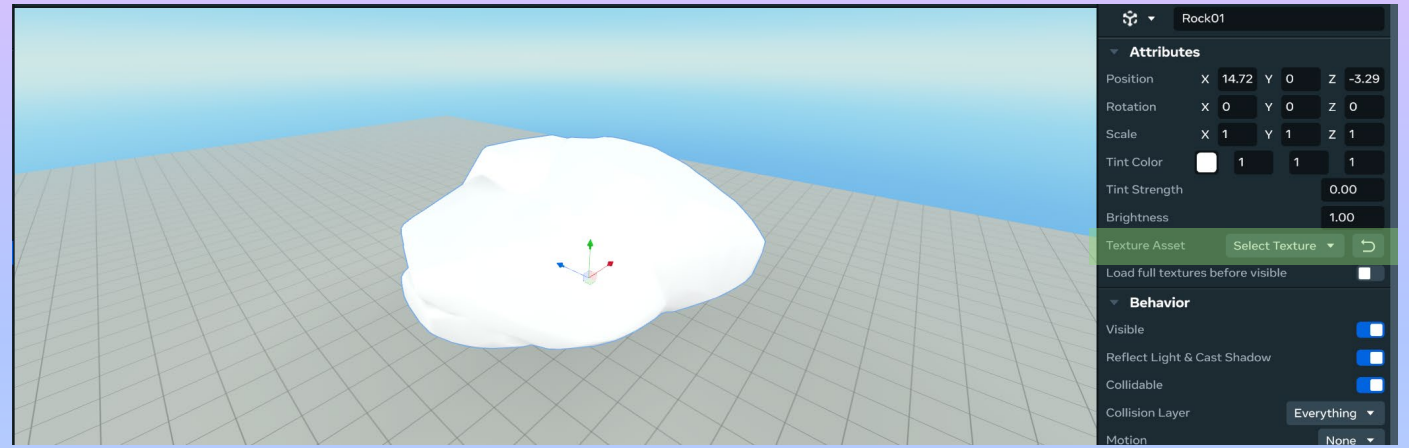
The static lighting can take time to update, so if you need to work with lighting quickly, or just want to make your life easier, I recommend using dynamic lighting. Keep in mind that it is more costly at runtime than static lighting.

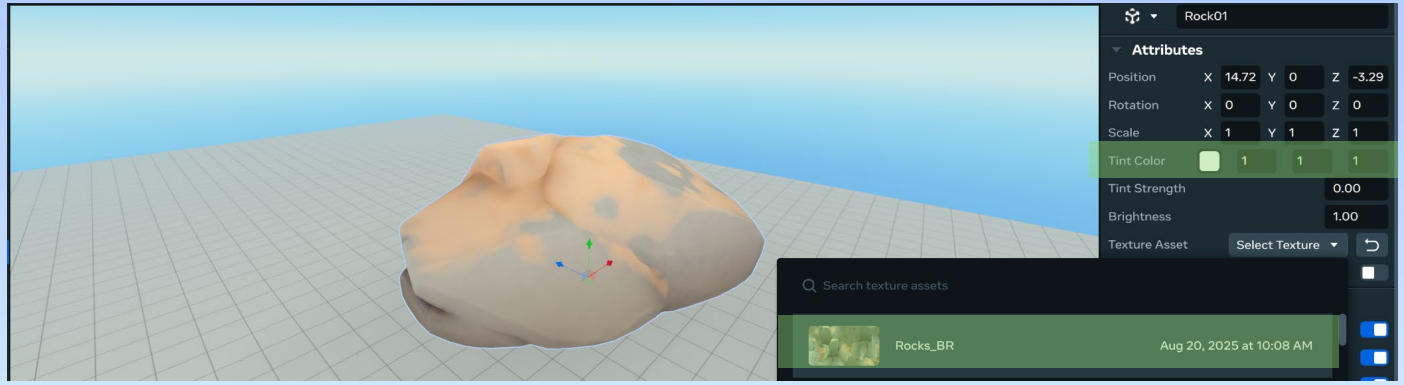
MESHES AND TEXTURES IMPORT



To import assets, it's in the same place as folders (click/drag doesn't work). Choose 3D model for mesh, and texture for textures.

There are 2 ways to assign a texture to a mesh. The first method is the easiest and more modular method, I recommend using this one if you only need base color, even more if your texture is shared between different assets.





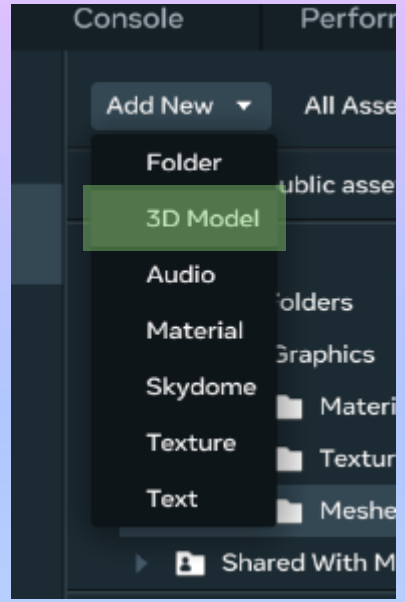
In my example, I'm using a single texture for multiple rocks, so it's important to use just one texture shared across different assets for better optimization.

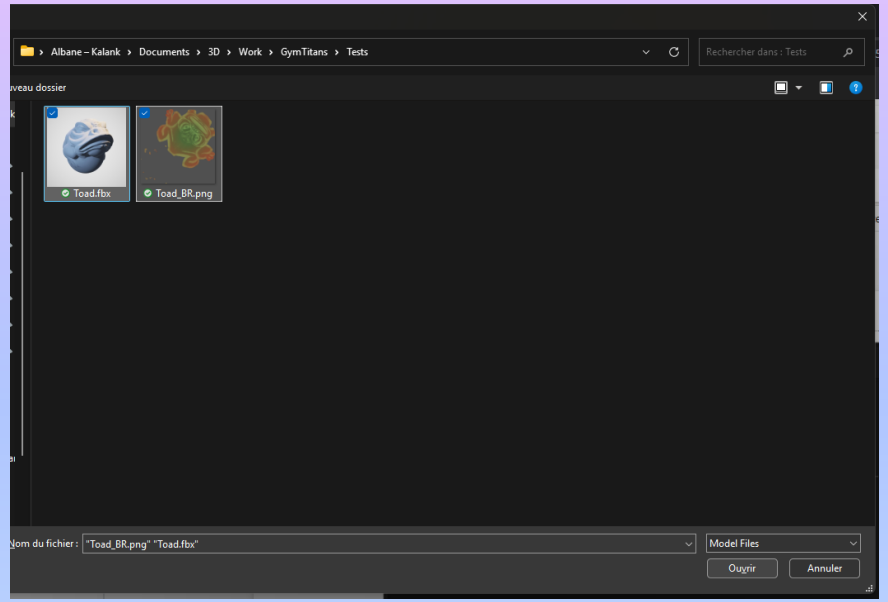
First, I import my model (here, my rock), then the texture (my rock texture). I drag the rock into the scene, and in the Properties panel, I can override the texture directly within the scene.

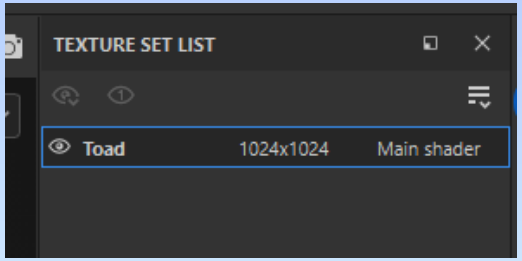
This way, you can use a single texture across multiple objects. If you need variations, you can combine this approach with a tint color, which is also available in the Properties panel, or with vertex paint (see slide on vertex paint materials).

MESHES AND TEXTURES IMPORT

The second method is used in case you want your mesh to always be used with its texture, or if you have different materials for the same mesh : while importing the mesh, you also need to select the texture and import both at the same time.







For this method, naming conventions are extremely important and STRICT. To import a mesh with its texture, everything must be named the same.

In my example, my mesh is called 'Toad'. If possible, my material should also be called 'Toad' (special cases will be covered in the additional info), and my textures should be named 'Toad_//'.

If you need variations, you can combine this approach with a tint color, which is also available in the Properties panel, or with vertex paint (see slide on vertex paint materials).

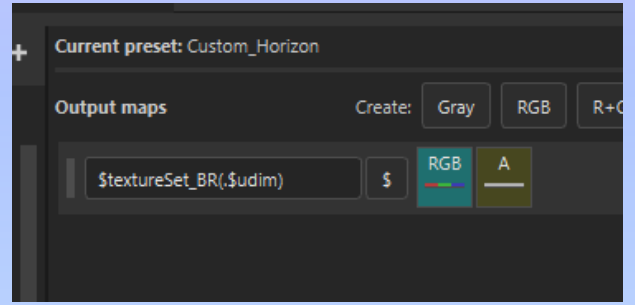
If anything is not named correctly, you will simply get an import error.
The texture channels are explained in the next slide.

MATERIALS

Link of the textures and materials doc

Main channels for textures :
We use _BR for Base Color in RGB and Roughness in Alpha.
If a second texture is needed, we use _MEO: Metalness in R, Emissive in G, and Ambient Occlusion in B.

	Naming	Channels
Texture A	MyMaterialName_BR.png	BaseColor (sRGB) + Roughness (linear)
Texture B	MyMaterialName_MEO.png	Metalness + Emissive + AmbOcclusion (all linear)



So you can create a custom export in Painter to make things easier (I only did _BR because I'm not sure I'll need _MEO, but both work the same way).

All this information only applies to a basic opaque material. If you want to use other types of materials, see the next slide.

OTHER TYPE OF MATERIALS

Link of the textures and materials doc

In general, as soon as you want a material that is something other than a basic opaque PBR, everything is controlled by the material's name.
Using my earlier example of the Toad, here's a summary of the material naming conventions and their associated textures:

Type de material	Nom du material	Nom des textures	Channel des textures
Opaque	Toad	Toad_BR (RGB+A) Toad_MEO (R+G+B)	Base Color / Roughness / Metallic / Emissive / Ambiant Occlusion
Opaque metallic	Toad_Metallic	Toad_BR (RGB+A)	Base Color / Roughness
Unlit	Toad_Unlit	Toad_B (RGB)	Base Color
Transparent	Toad_Transparent	Toad_BR (RGB+A) Toad_MESA (R+G+B+A)	Base Color / Roughness / Metallic / Specular / Alpha
Masked	Toad_Masked	Toad_BA (RGB+A)	Base Color / Alpha
UI Optimized	Toad_UIO	Toad_BA (RGB+A)	Base Color / Alpha
Unlit Blended	Toad_Blend	Toad_BA (RGB+A)	Base Color / Alpha

VERTEX PAINT MATERIALS

Link of the textures and materials doc

You can use vertex painting to texture a mesh without any textures.
You just need to paint the vertices directly (for example, ZBrush's polypaint is a vertex painting technique). In this case, the vertex colors are applied directly in the engine.

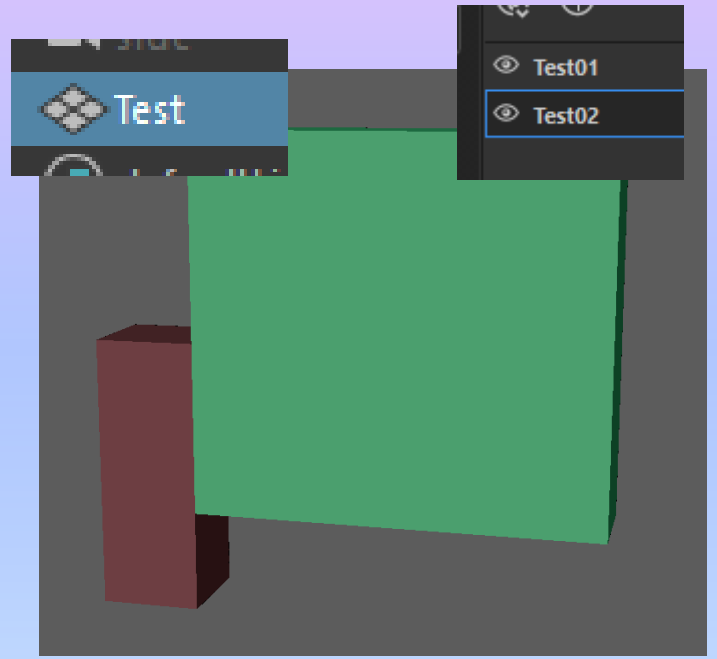
If you want to use vertex painting on top of a texture, that's also possible. Here's the table of naming conventions:

Type de material	Nom du material	Nom des textures	Channel des textures
Vertex color	Toad_VXC	/	/
Vertex color + single texture	Toad_VXM	Toad_BR (RGB+A)	Base Color / Roughness
Vertex color + double texture	Toad_VXM	Toad_BR (RGB+A) Toad_MEO (R+G+B)	Base Color / Roughness / Metallic / Emissive / Ambiant Occlusion

For vertex colors combined with textures, the vertex color is applied directly to the mesh and multiplied with the texture. This allows the same texture to be reused across multiple surfaces while still providing color variation.

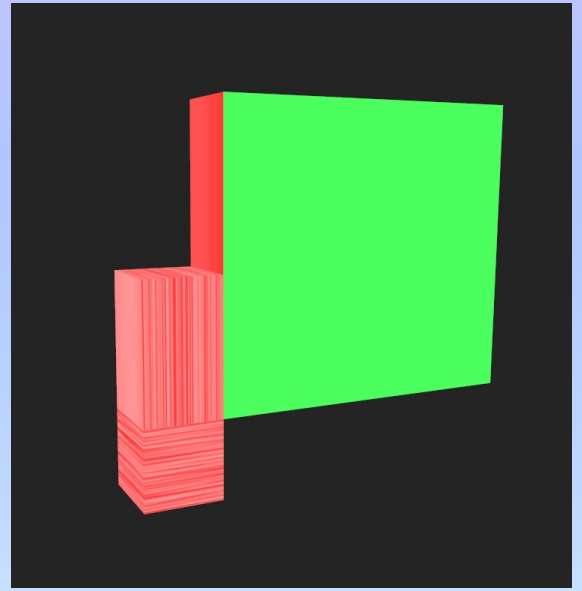
MATERIALS MULTIPLE : EXEMPLE

My mesh is called 'Test'. I create a first material called 'Test01', then I select the areas where I want to assign my second material, 'Test02'.

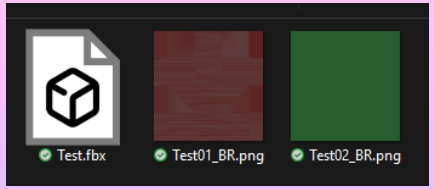


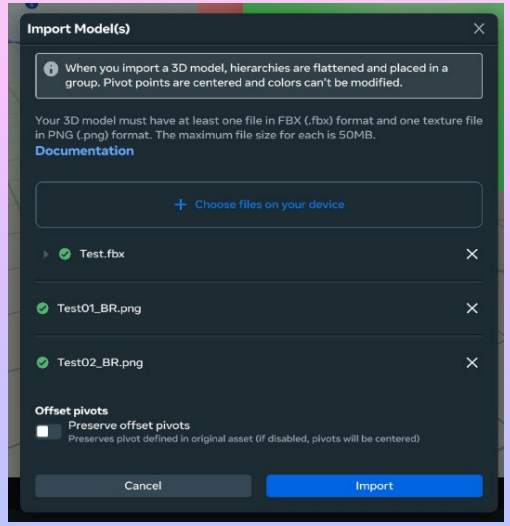
Reminder: it’s important to name them 'Test01' and not 'Test_01', otherwise it won’t work.

I texture my mesh (here’s what it looks like in Painter), and I export my maps following the naming conventions we saw earlier.

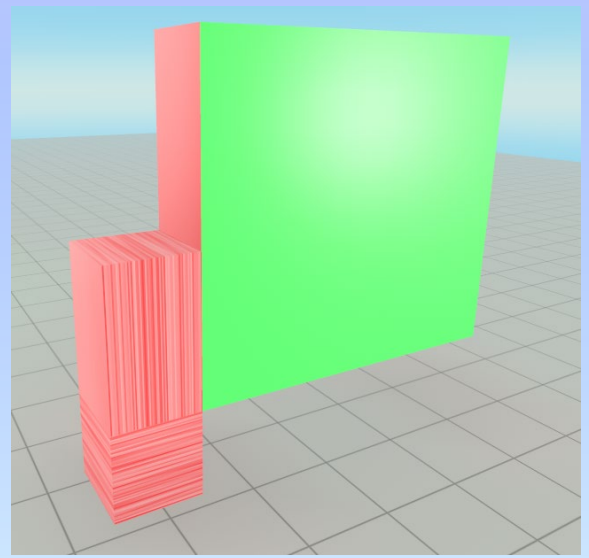


Add new – 3D model – Select the mesh and then its associated textures - import





And here’s the result in Horizon!



ADDITIONAL MATERIALS INFOS

Link of the textures and materials doc

Link of multiple materials and limitations doc

Normal maps are currently not supported in the game engine, so you'll need to fake as much as possible in the base color, or include the details in the modeling if absolutely necessary.

As mentioned earlier, if you need multiple materials for a single mesh, you obviously can't name all the materials the same. In that case, you can make an exception and give them different names, for example, ToadGreen and ToadYellow.

If your name has multiple parts, for example Toad_Green_01, you MUST NOT use underscores (_) in the name, as it will break everything. It should be called ToadGreen01.

For Masked or Vertex Color materials, ALL materials on the mesh must be of that type. You cannot mix Masked and opaque materials on the same mesh, same for Vertex Colors.

In the table, I listed two types of opaque materials: the classic _BR, where you can have an additional texture for M, E, and AO if needed. But if you want a basic opaque material and only need metallic set to 1 without Emissive or AO, you can just add _Metallic at the end of the material name. In this case, you only need a _BR texture, and the Metallic will automatically be set to 1.

If you need multiple materials and the order matters, you can enforce it using the suffix _skin##. For example, ToadGreen_skin00 will appear in slot 0 and ToadYellow_skin01 will appear in slot 1. I believe if you need to combine multiple suffixes, the _skin## comes last (e.g., ToadYellow_Masked_skin00).

VFX

Link of the Meta doc
Link of the PopcornFX doc

It’s possible to create your own VFX that can be directly integrated into the game!

However, VFX need to be created using an external tool called PopcornFX (a well-known software for making VFX).

In short, you need to:

- Download the correct version of the PopcornFX editor and create your VFX there.
- Once the VFX is finished, export it as a .pkkg file in Unity (using the correct Unity version as well).
- In Unity, install the specific PopcornFX Package Uploader plugin.
- Then click on the Horizon tab -> PopcornFX Package Uploader and log in.

ENVIRONMENT GIZMO

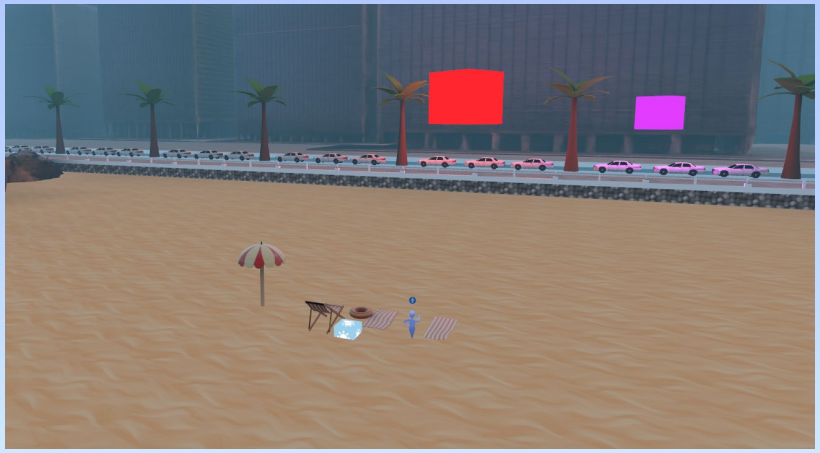
Link of the environment doc

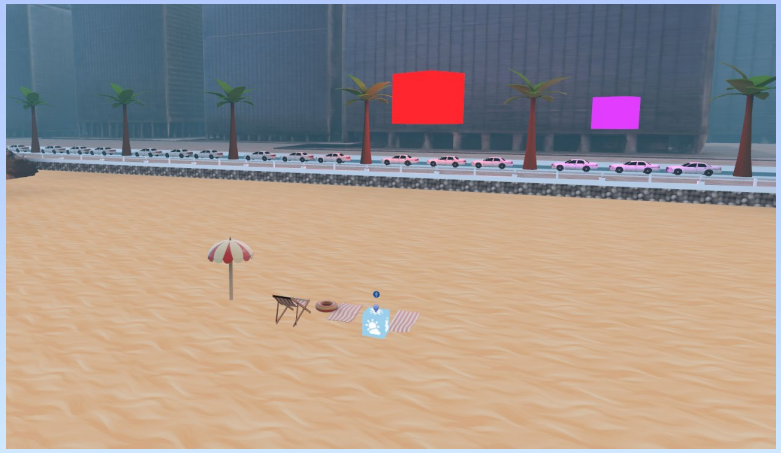
The Environment gizmo includes:

- Sun settings: shadow, reflection, intensity
- HDRI settings (called Skydome): rotation, exposure, gradient
- Fog settings: color, density

For the Skydome, there are two ways to work with it: you can either create a custom gradient with 3 colors (top, horizon, bottom), or use a cubemap (see following slide).

For the Sun, I ran into an issue for a while where it simply stopped emitting light, and my entire world was in shadow (see left image). Actually, if the Environment gizmo is below the ground level, it seems like it doesn't emit any light.

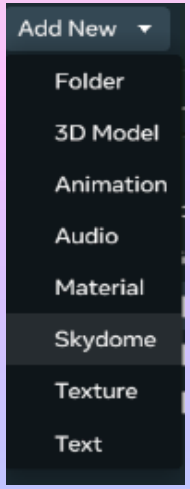


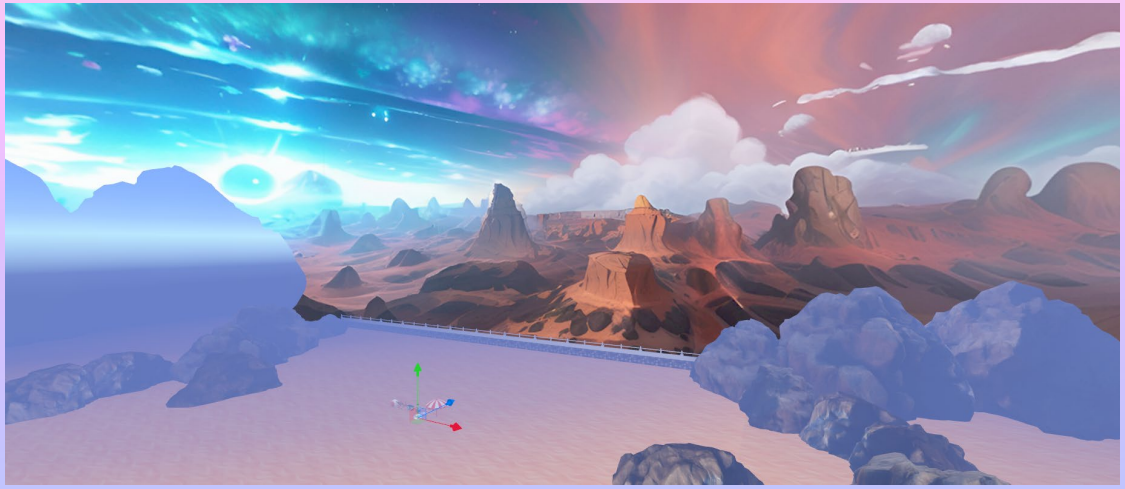


CUSTOM SKYDOME

Link of the Skydome doc

The Skydome is Horizon's custom HDRI. It allows you to set any image you want as the background.





To create a custom skydome, you first absolutely need a PNG with this exact size:

Horizontal Strip: 6144 x 1024 pixels

But a simple 360° image is not enough. You need to convert it into a cubemap, then align the given squares in the exact order shown next to it (hence the 6144x1024 size):

Note: you cannot switch Skydomes using TypeScript.

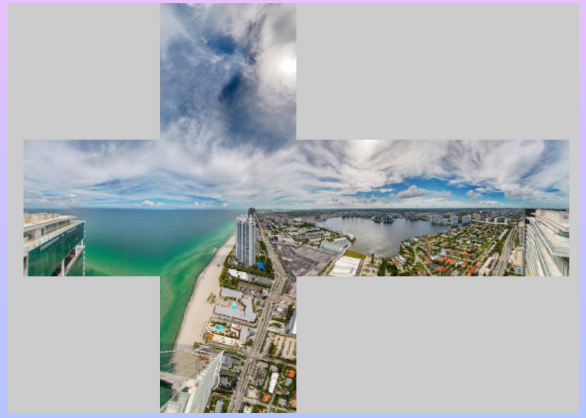
CUSTOM SKYDOME : EXAMPLE

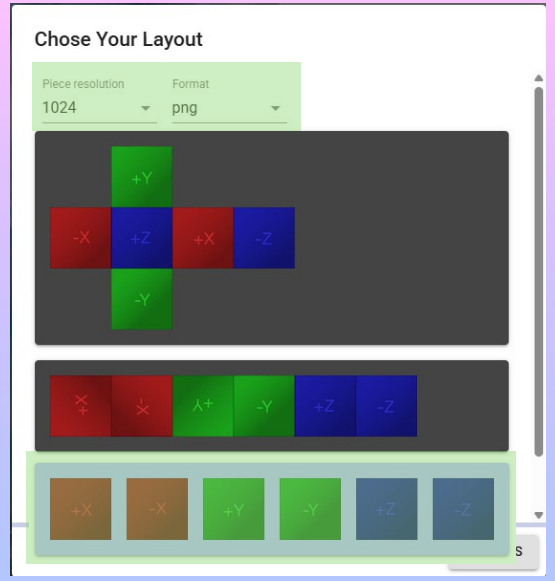
[Link of the Skydome doc]

[Link HDRI to cubemap converter]

I have this HDRI

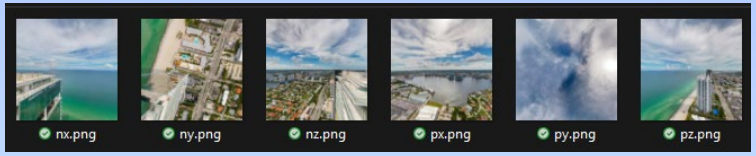
Using the website linked at the top of the slide, I convert it into a cubemap.

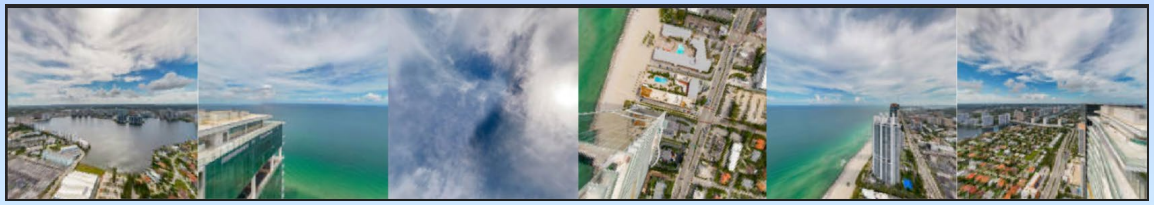




I export the results in the correct format and at the right resolution.

I end up with each square exported and correctly renamed, with 'p' for positive, 'n' for negative followed by the axis. I then align them (for example, in Photoshop).





I import it into Horizon, drag and drop it into the scene, and that's it!

CUSTOM SKYDOME : FOG MAP

Link of the Skydome doc

You can also import a custom fog map to go with the HDRI, which allows for color variation in the fog instead of a single uniform color.

For greater distances between camera and mesh, the further away, the mesh is more tinted by the fog map. It's a non-linear gradual change.

Unlike all other assets in Horizon, which only support PNG, the fog map only supports EXR

- Fog maps are EXR format only. Do not use a PNG, which yields poor results.

The size is also strict, just like for the HDRI:

- a **horizontal strip**: 384 x 64.

POLYCOUNT

Link of the budget doc

There’s a whole documentation on recommended polycounts for a complete scene depending on your game style (static or dynamic), per asset according to size.

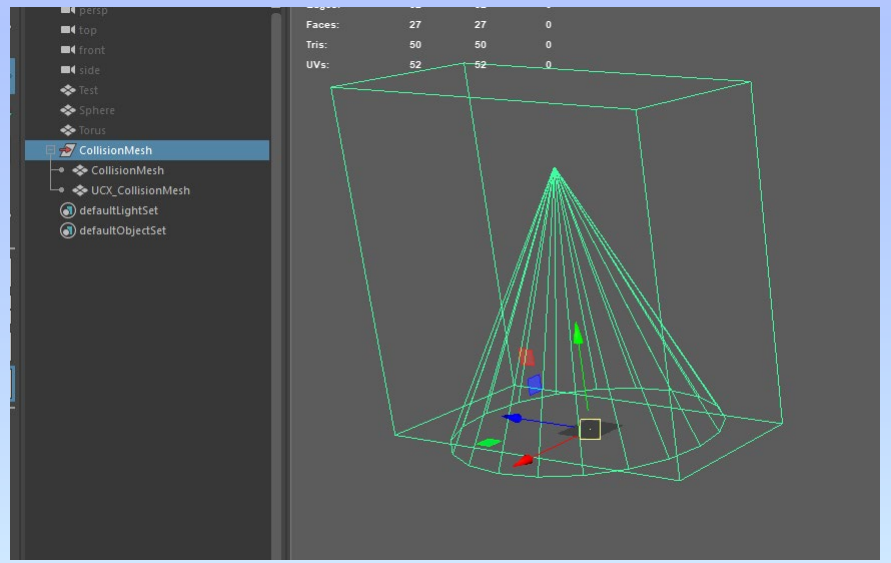
The two most important things to remember are in these two tables:

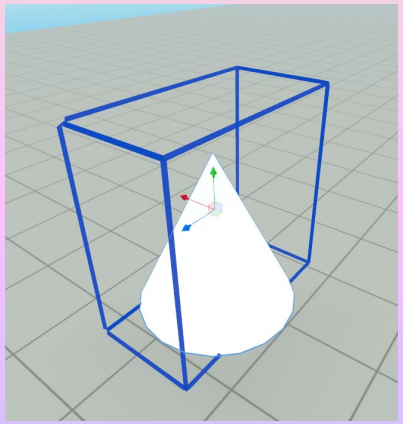
	Polygons	Vertices	Texture Size
5 m3Object	2000	1000	2048 x 2048
1 m3Object	1000	500	512 x 512

	Approximate limits for a gameplay heavy world	Maximums, when you have a more static world
Total Vertices	600,000	1 Million
Draw Calls	150	250

CUSTOM COLLIDERS

You can import custom colliders by following a specific naming convention in the mesh hierarchy.
In my example, my mesh is called **“CollisionMesh”**, so its collider is named **“UCX_CollisionMesh”** because I want a convex collision.
I group the two under the mesh name so that the engine recognizes it as a single, unified mesh and interprets it correctly.





Lien de la doc colliders

In the engine, you can immediately tell if the collider was imported correctly. It shouldn’t be visible as a mesh on screen, but the object’s bounding volume will conform to the shape of the collider.

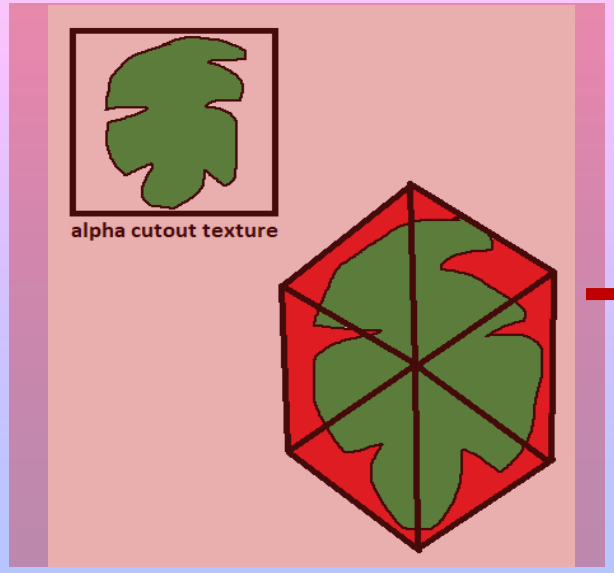
Type	Mesh Prefix Naming	Requirements
Box	UBX_[VisibleMesh]_##	A Box must be created using a regular rectangular 3D object. You cannot move the vertices around or deform it in any way to make it something other than a rectangular prism, or else it will not work.
Sphere	USP_[VisibleMesh]_##	A Sphere does not need to have many segments (8 is a good number) at all because it is converted into a true sphere for collision. Like boxes, you should not move the individual vertices around.
Capsule	UCP_[VisibleMesh]_##	A Capsule must be a cylindrical object capped with hemispheres. The capsule is expected to be vertical in local space. It does not need to have many segments (8 is a good number) at all because it is converted into a true capsule for collision. Like boxes, you should not move the individual vertices around.
Convex Hull	UCX_[VisibleMesh]_##	A Convex object can be any completely closed convex 3D shape.
Concave Mesh	UCC_[VisibleMesh]_##	Any concave mesh. This is the most flexible type of collider but is also the least performant. Unity will generate a convex hull if it is marked as dynamic.

Naming convention depending on the needed collider

OPTIMISATION

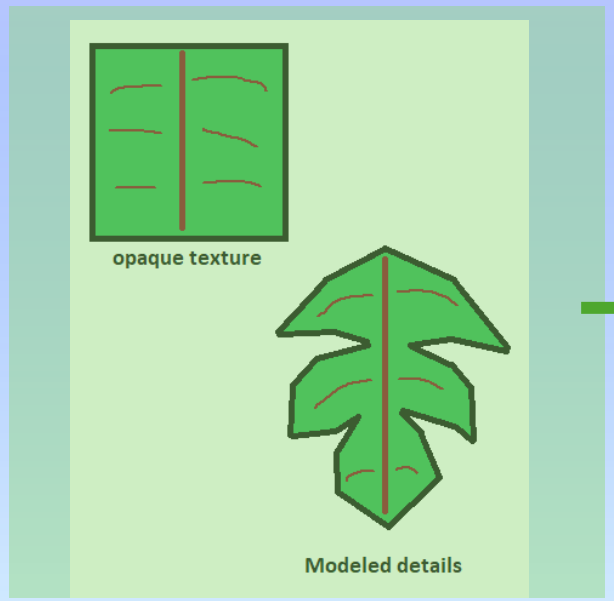
Link of the performance doc
Link of the GPU best practices doc

It is recommended to minimize the use of materials that rely on opacity (masked, transparent, blend, etc.).
If possible, model as many details as you can directly, since today we are much less constrained by polygon count.



Bad practice

The method to choose is also depending on the size of the object. For a tiny daisy that only covers about 4 pixels on the screen, a bit of transparency is fine. This recommendation mainly applies to objects that are reasonably large and can be seen up close.



Good practice

LEVEL DESIGN AND OPTIMISATION

Link of the performance doc
Link of the GPU best practices doc

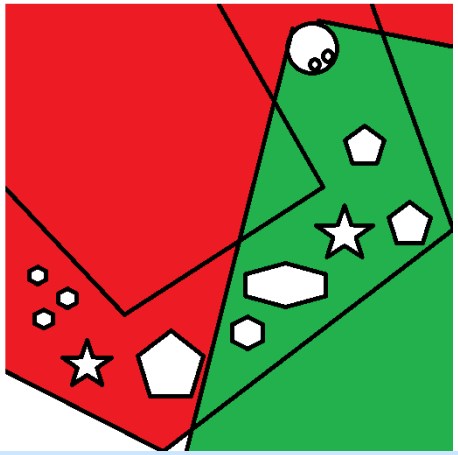
The documentation explains that there is no occlusion culling system in the engine, which means that if you're facing a wall, everything behind that wall will still be processed, even if the player can't see it.

> Meta Horizon Worlds does not currently support occlusion culling to avoid drawing objects hidden behind other larger objects. This makes world layout, mesh merging, and visibility control the main tools available to us for keeping the number of vertices sent through the graphics pipeline as low as possible.

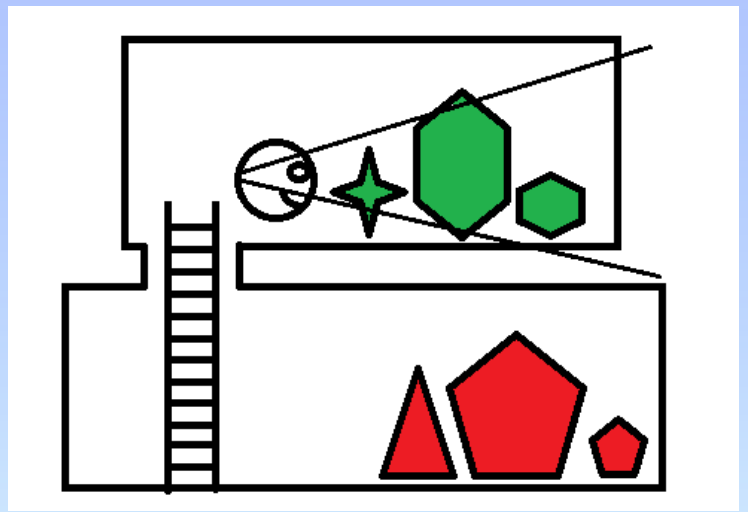
As a result, here are some level design tips to optimize your scenes as much as possible, which focus on the camera occlusion instead of the culling:

Avoid having straight paths where the player can see infinitely; force them to turn.

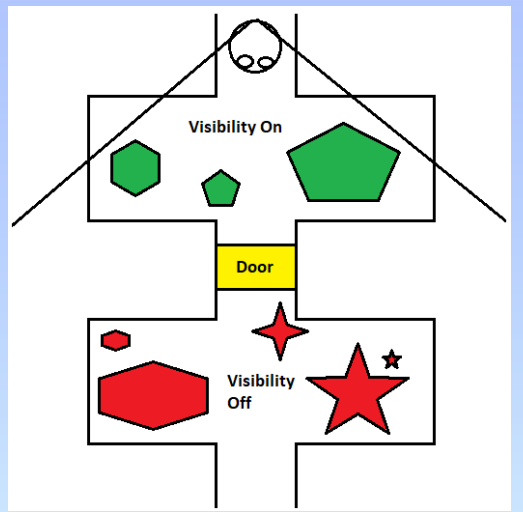
By adding twists and turns to your world, you can limit the amount of objects visible at once. objects outside the view frustum will be culled out.



Add verticality to help cull what's below.



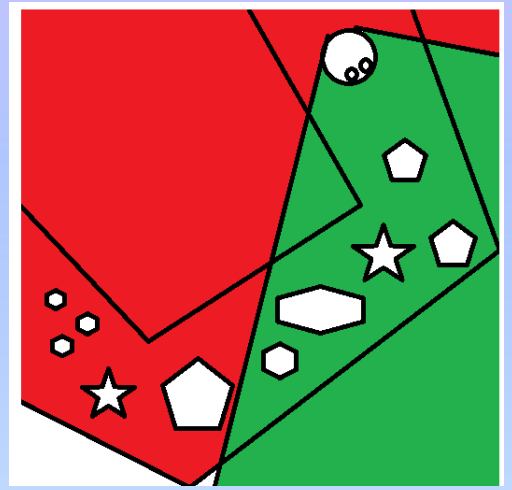
Add mandatory visual obstacles for the player to pass through, which allows you to manually cull objects and activate/deactivate them using triggers.



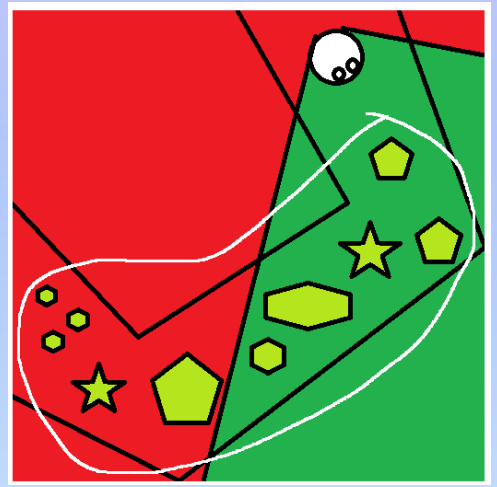
DRAW CALLS

Each object in the scene generates a draw call. To optimize as much as possible, the documentation suggests merging elements logically. Taking the example of the turn:

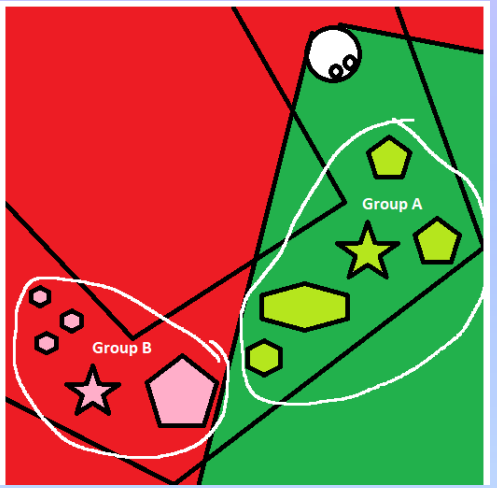
Here, nothing is merged—each object generates 1 draw call, which makes a total of 5 draw calls.



Here, everything is merged, which results in a single very large draw call that ends up processing objects that aren’t even in the player’s field of view.



Here, the objects are merged into 2 groups, organized logically according to the player’s path.



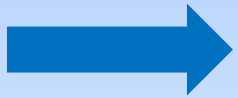
WORLD STREAMING

Link of the world streaming doc

World streaming allows you to subdivide a large world into multiple smaller worlds. This unloads everything that isn't immediately relevant to the player, saving performance. It also allows multiple people to work on the world simultaneously by assigning different sublevels to each person.

Here are the existing limitations of this system as outlined in the documentation:

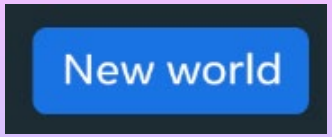
- Loading a sublevel for one user will load it for everyone. So, if multiple people are working on a world simultaneously, many sublevels might be loaded at the same time, which can negatively impact performance.
- Automatic switching of sublevels based on player movement is not supported. This means a custom script is needed to handle streaming the sublevels.
- Setting up the sublevel system can be difficult because each sublevel is treated as a completely separate world loaded into a parent world, rather than just a partition of an existing parent world like in Unreal Engine (the step-by-step process is detailed in the next slides).

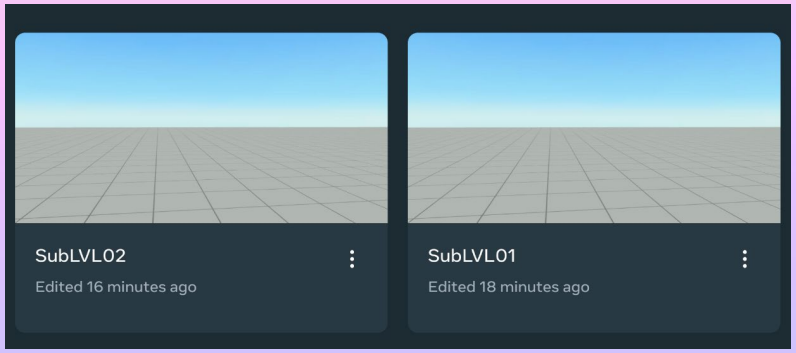


WORLD STREAMING : EXAMPLE

Link of the world streaming doc

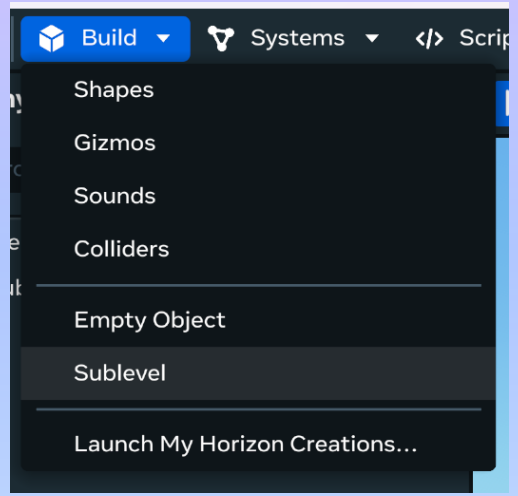
In my example, we want to load 2 streaming levels into a persistent level. So, we create two levels and rename them as desired (here, “SubLVL01” and “SubLVL02”).

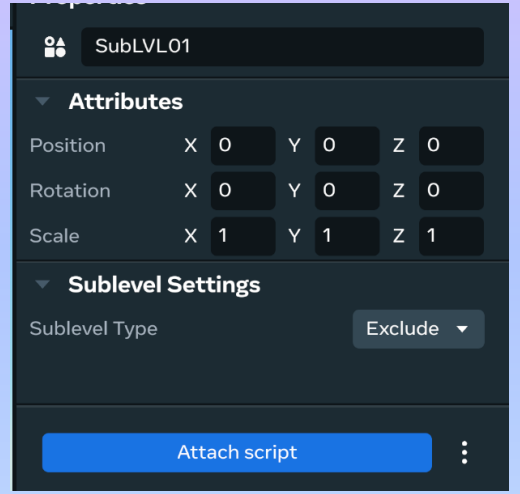


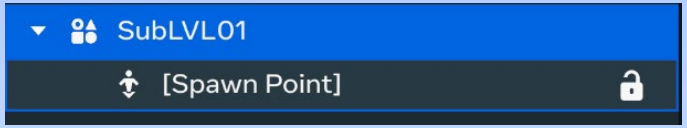


I’ll detail the procedure for SubLVL01, but it’s exactly the same for SubLVL02.

We open our SubLVL01 and add a "Sublevel" gizmo from the "Build" tab. Click on it, rename it, and change its type to "Exclude."







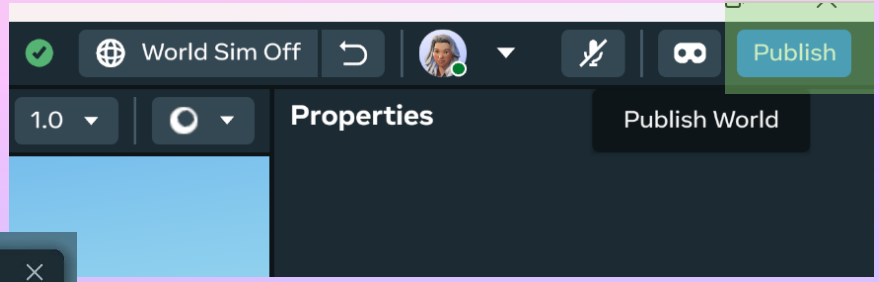
As a child of our Sublevel gizmo, we add the Spawn Point (this won’t work until the gizmo is set to “Exclude”).

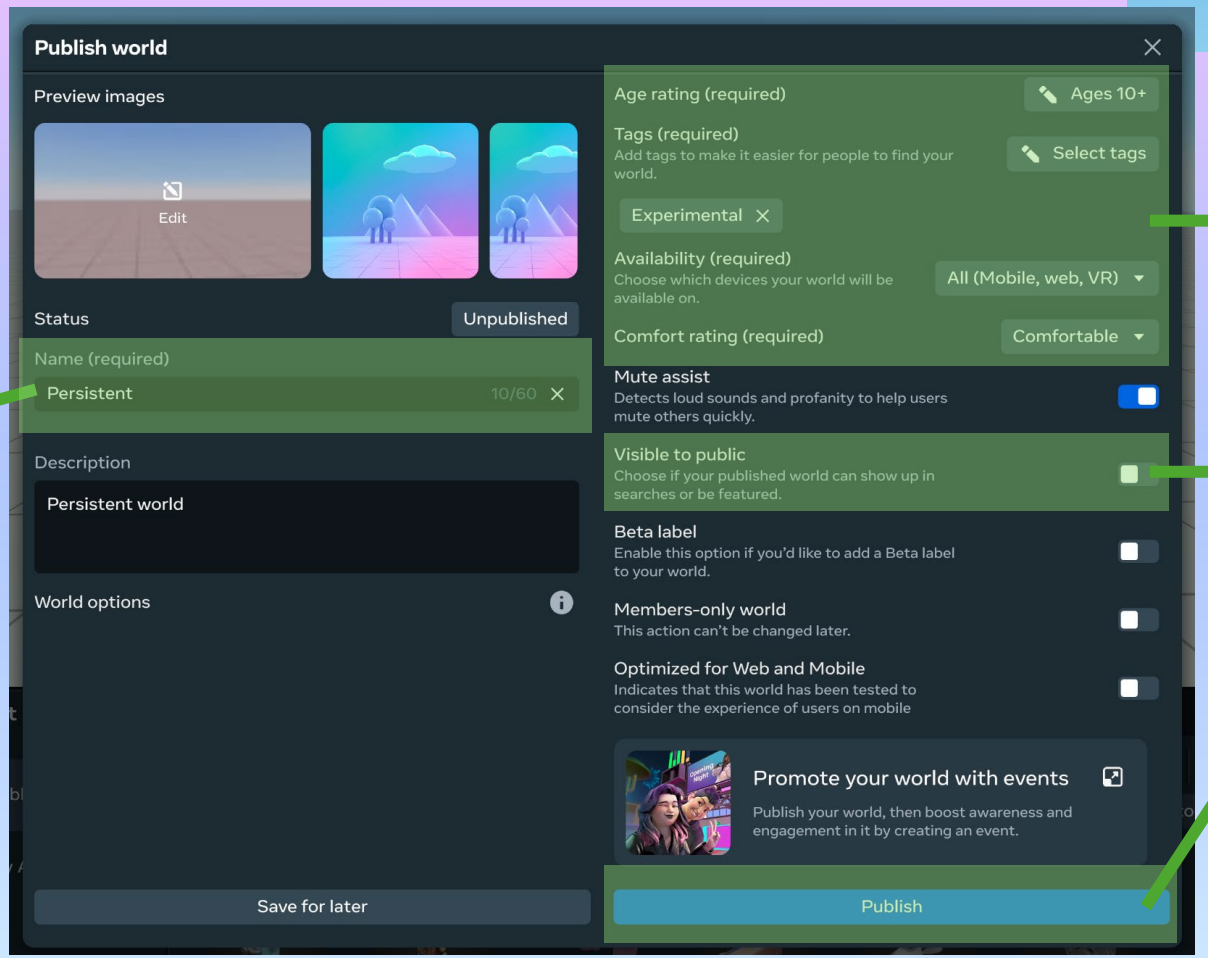
We repeat the same process for SubLVL02, then set them up so they can be loaded.

WORLD STREAMING : EXAMPLE

Link of the world streaming doc

To access our SubLVLs from the persistent level, like everything in Horizon, it has to go through the online system, so they need to be published.





Mandatory parameters to fill in to be able to publish the world

Name of the world (in the screen it is called persistent, but it is the SubLVL that we need to publish)

Make sure to leave this option unchecked so that you're the only one who can modify it.

Publish !

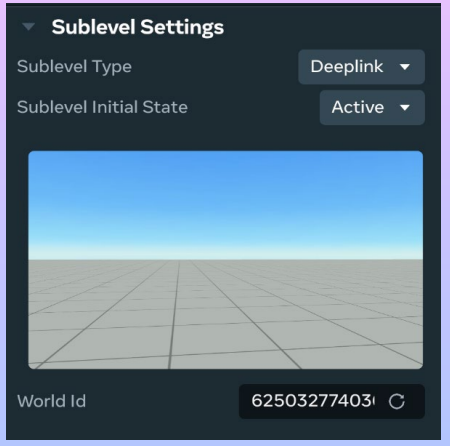
Publish both of your SubLVLs, then move on to the persistent level.

WORLD STREAMING : EXAMPLE

Link of the world streaming doc

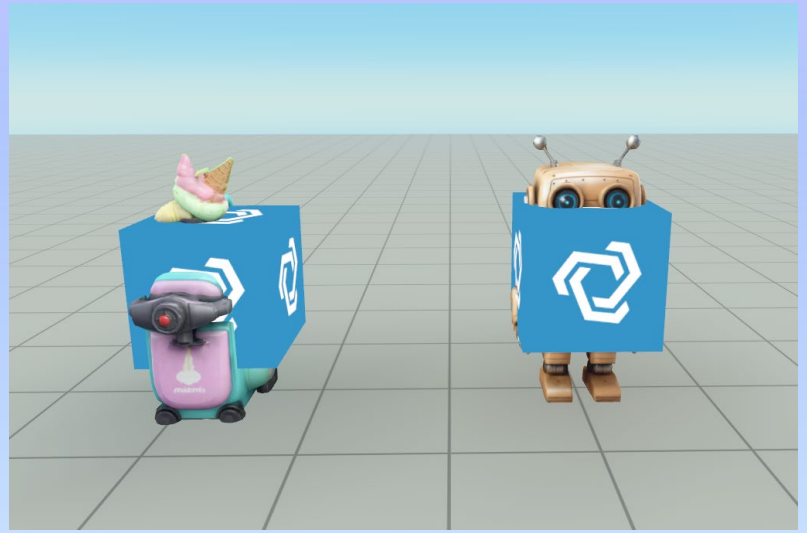
Create a world that will contain the sublevels, and rename it as you like (here, “Persistent”).

Add two sublevel gizmos as before, but this time they will be used to load the levels you’ve already created. Set their type to “deeplink,” and if you want to preview the levels, switch them to “active.”
Then, you can select them by clicking the world preview in the Sublevels properties panel.



SubLVL01

SubLVL02



Persistent

To differentiate them and make sure everything works, I placed a mesh in each sublevel. This way, I can see that the link works correctly and I’m able to manipulate my sublevels as I want!

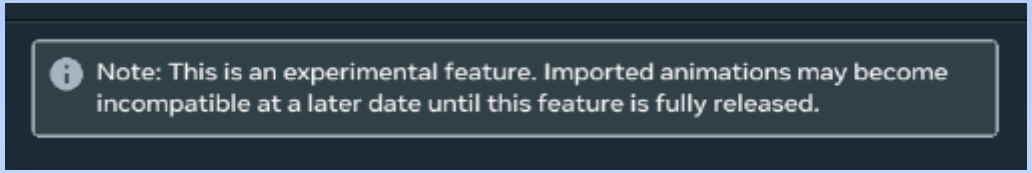
ANIMATIONS

Link of the avatar animations doc

To import custom animations, you must use their rig (a pre-rigged avatar is available for download in the documentation linked at the top of the slide). Trying to use a custom rig will not work.

In fact, in the doc there's a 4-minute video that shows the process from A to Z, so I won't rewrite everything here—it's honestly pretty clear.

IMPORTANT: To be able to import animations into the engine, it is also necessary to download the fbx2anim file. This file must be placed at **C:/bin**.
/!\ Make sure to launch the engine as an administrator /!



Just a heads-up: when you do "add new → animation," a window similar to the FBX import opens. There's a note saying that the custom animation import feature is experimental, so bugs or weird animation results can happen—that might be the reason.

UI

Link of the UI animations doc

As mentioned earlier in the materials section, UIs use a special _UIO material (you *can* make UIs with any material, but _UIO gives better image clarity, and I believe it's the only material that can be animated via TypeScript—so yes, UIs *can* be animated!).

```
class AnimatedGIF extends Component<typeof AnimatedGIF> {
  static propsDefinition = {
    speedSeconds: { type: PropTypes.Number, default: 1 },
    texture0: { type: PropTypes.Asset },
    texture1: { type: PropTypes.Asset },
    texture2: { type: PropTypes.Asset },
    texture3: { type: PropTypes.Asset },
    texture4: { type: PropTypes.Asset },
    texture5: { type: PropTypes.Asset },
    texture6: { type: PropTypes.Asset },
    texture7: { type: PropTypes.Asset },
    texture8: { type: PropTypes.Asset },
    texture9: { type: PropTypes.Asset },
    texture10: { type: PropTypes.Asset },
    texture11: { type: PropTypes.Asset },
    texture12: { type: PropTypes.Asset },
    texture13: { type: PropTypes.Asset },
    texture14: { type: PropTypes.Asset },
    texture15: { type: PropTypes.Asset },
    texture16: { type: PropTypes.Asset },
  };
```

The documentation provides a full case study on how to animate a UI from scratch. Basically, there's no other method presented—other than importing the frames one by one and switching them individually via a script.

Note: The documentation specifies that images can only be converted into billboards if multiple images are grouped. If you have a single UI element, the procedure is to duplicate it, group the copies, convert the group into a billboard, and then delete the extra copy.

> As of the time of writing (June 2024), billboarding can only be applied to grouped objects, not singular planes like this, so we will need to duplicate (Ctrl + D), and then group the two GIFs. After grouping we will delete the extra GIF.

UI

Link of the UI animations doc

For the UI, you can’t place elements just anywhere. Here is a map of allowed zones:

- Red : No custom UI is allowed in these areas.
- Yellow : Custom UI is allowed, but it can occasionally be hidden by notifications.
- Green : Custom UI ok