# VTOL VR
## Custom Static Objects
Creator's Guide

This is a quick guide to getting custom objects set up for importing to VTOL VR. It assumes you have created or sourced your own 3D models and have some basic understanding of how to use the Unity editor. If you need additional help with creating 3D models or using Unity, there are tons of resources already online. You will **not** need to do any programming/scripting.

## Part 1: Installation

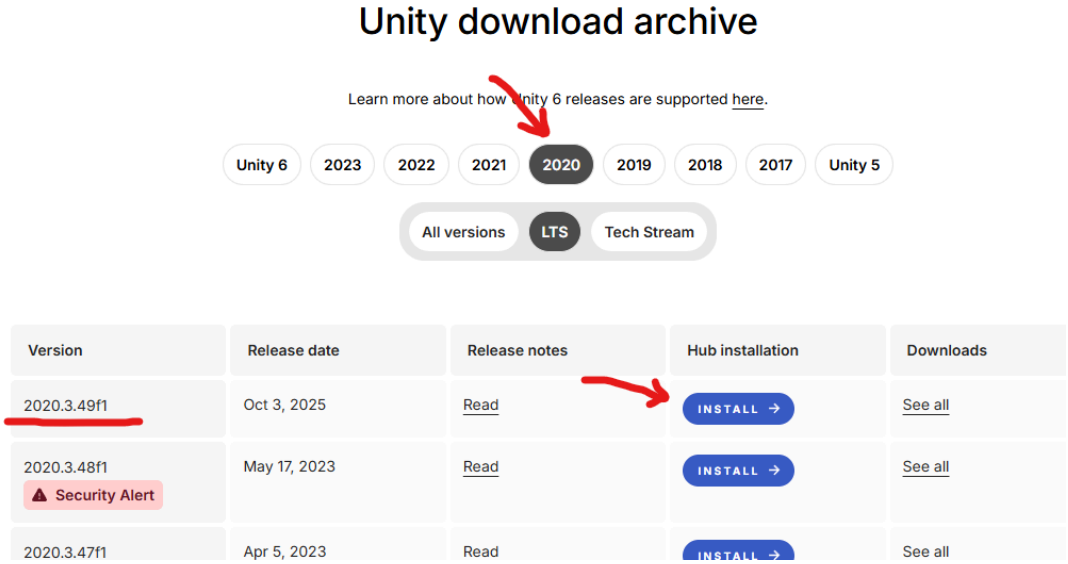
Installation involves installing Unity Hub, a compatible version of Unity Editor, and importing the VTOL VR Modding Tools package.

**Step 1: Get the Unity Hub from [https://unity.com/](https://unity.com/)**

- Click "Download Unity" at the top.
- Install Unity Hub. This is where you manage your Unity installs.

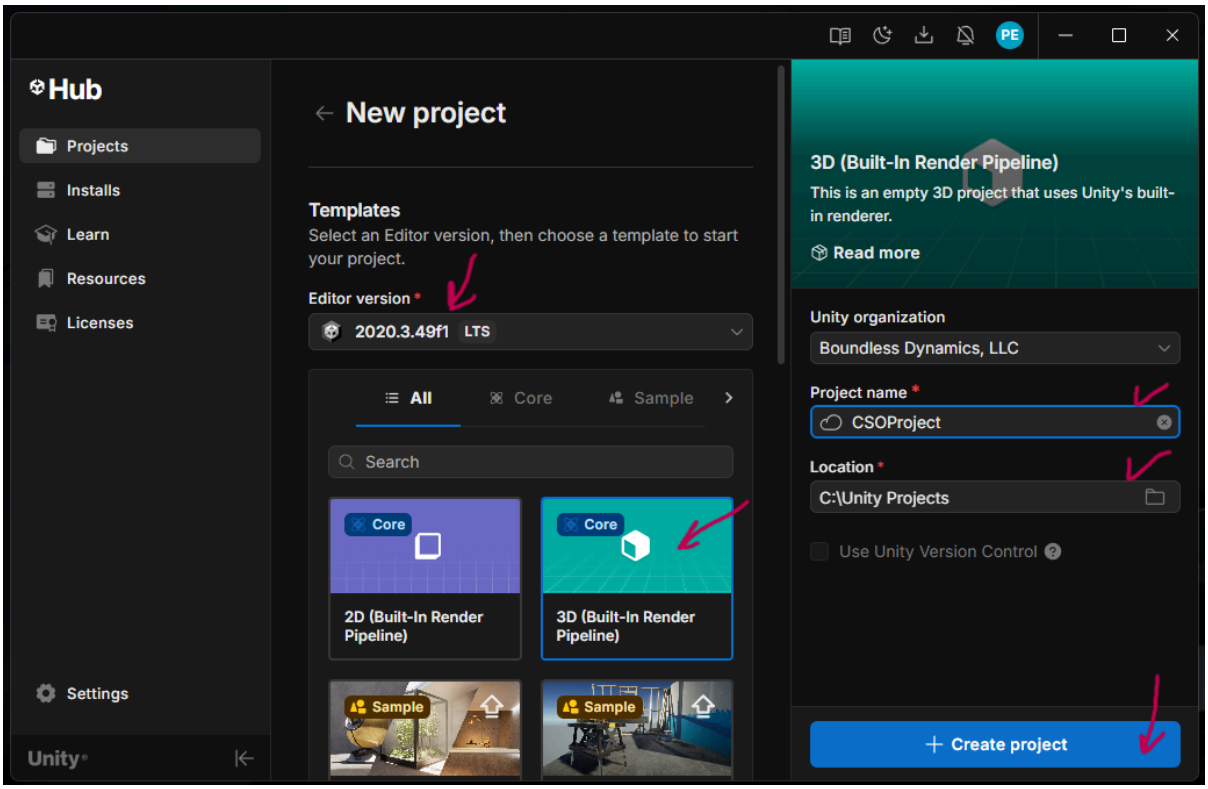
**Step 2: Install Unity Editor**

- In Unity Hub, click on Installs on the left
- Click Install Editor
- Click on the Archives tab and click "download archive"
- I recommend to install the same version of Unity that VTOL VR is built on:
- Click on the 2020 tab and Install Version 2020.3.49f1


**Step 3: Create a project**

- Once the editor has finished install in Unity Hub, go to the Projects tab and click "+ New project"
- Select the Editor version you installed (2020.3.49f1)
- Select 3D (Built-In Render Pipeline)
- Enter a project name and save location
- Create project




**Step 4: Install VTOL VR Modding Tools**

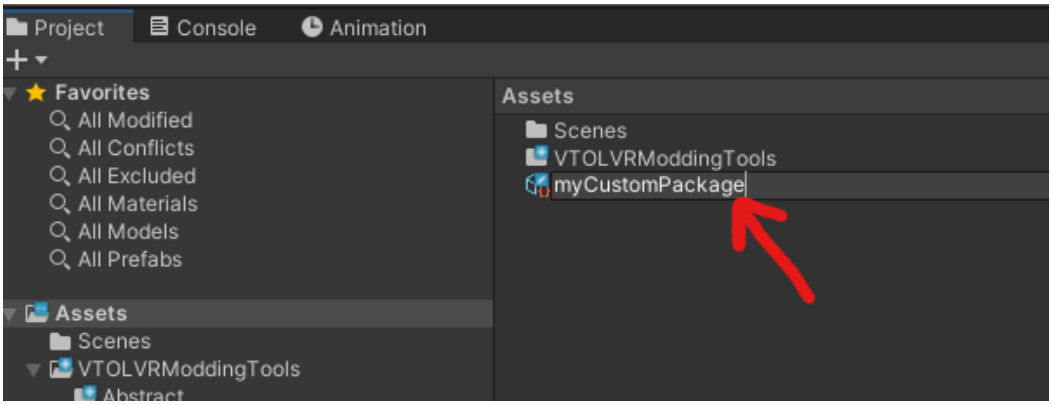
- Unzip the modding tools archive
- Drag “VTOLVRModdingTools.unitypackage” into the Project/Assets panel in Unity Editor.
- Leave all items selected, and click Import

## Part 2: Creating a CSO Pack

Custom static objects are organized into packs to make it easier to share multiple objects, and allow the game to share common resources such as materials and textures between many different items. Even if you only intend to make one custom object, a CSO pack must be defined in order to import it to VTOL VR.

**Step 1: Create a pack**

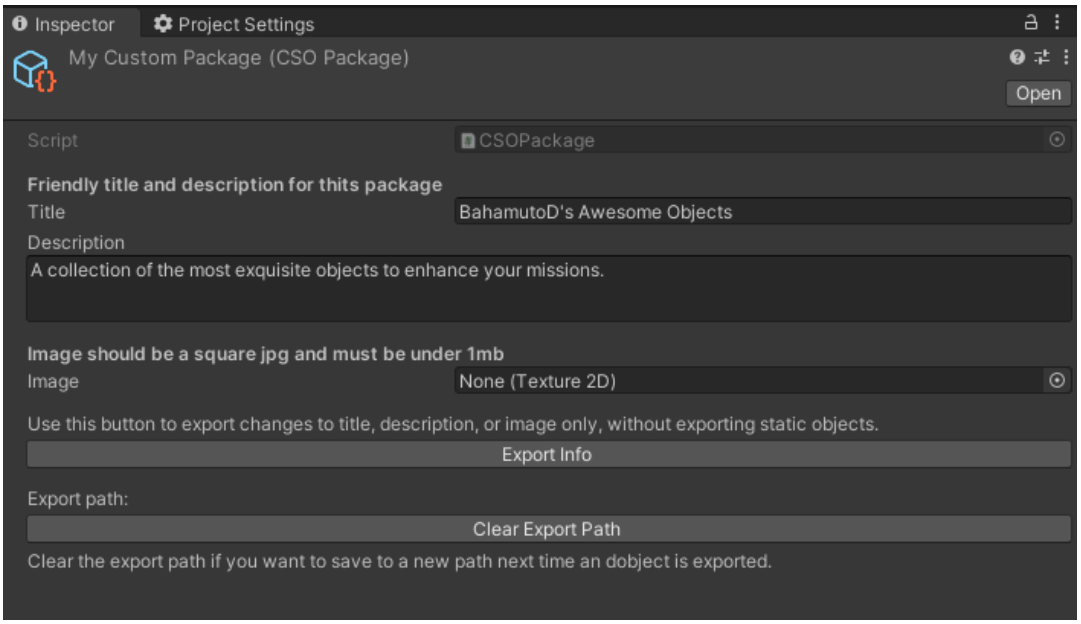
- In the Unity Editor Project panel, right click the assets folder (or any subfolder -- you can organize as you wish), then select Create → VTOL VR → CSO Package
- Name the package asset something unique to your collection




**Step 2: Enter information**

- In the inspector, enter a title and description for the pack.

- To assign an image for the pack, import your image to the Unity assets then drag in or select it in the Image slot.



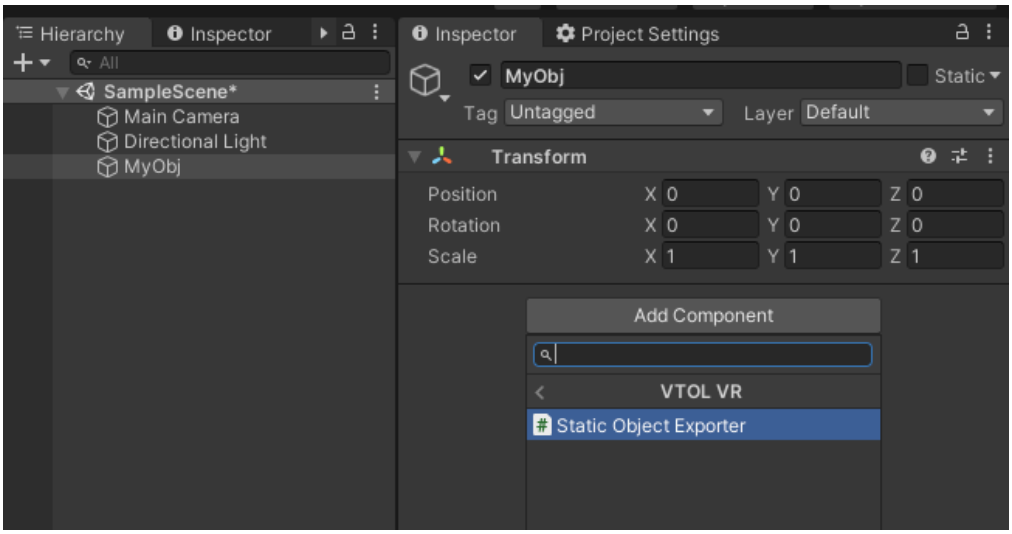

## Part 3: Creating a Static Object

Static objects in VTOL VR are meant to be set pieces, decorations, or destructible stationary targets in custom missions. The modding tools package defines a set of scripts which are supported for use in a VTOL VR static object. Any custom scripts you write or default Unity components which are not supported by VTOL VR will not be included when you export your static object.

Be mindful of performance when creating a static object. Limiting the number and size of textures, mesh vertices, collider complexity, etc will improve the gameplay experience when there are many static objects present in a mission. Use LODs for smaller objects which don't need detail when viewed from far away. Remember that players will be high in the sky most of the time.

**Step 1: Create the object root**

- Create a new GameObject in the scene with no parent (root). This will be the origin point of the object when placing it in the mission editor.
- Add a **StaticObjectExporter** script to the GameObject.




- Select your previously created CSO pack in the Package slot.

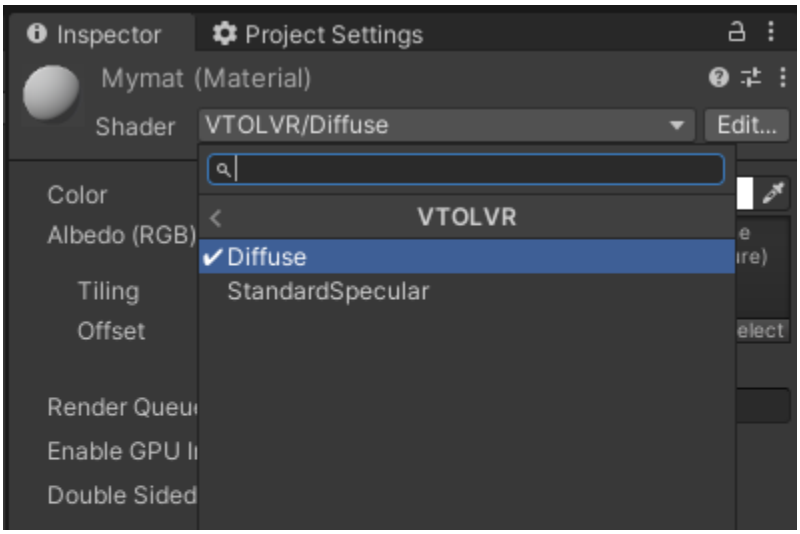
- Fill out the information about the object in the rest of the exporter's fields.

**Step 2: Build the Static Object**

- Import your 3D model(s) and place it as a child of the root exporter object.
- Position the object so that it appears correctly relative to the root. You can enable the "Show Floor Gizmos" option on the exporter script to see how the object would sit on the ground in game.
- To scale the object properly, one unit in the editor measures 1 meter in the game world. You can create a primitive cube as a root object and scale it to measure the size of your object.

**Step 3: Apply materials**

- Custom materials **must** use one of the shaders in the VTOLVR category
- **OR** if you want a part of your object to match the terrain of where it is placed, you can use the included VTOLVR_GROUND material. Do not change the name of this material -- the game checks the name to replace it with the appropriate terrain material.




**Step 4: Add scripts**

- Add any of the following supported scripts/components to objects in your hierarchy:
  - **Supported Unity Components**
    1. Mesh renderer + Mesh filter
    2. Colliders (any type)
    3. LOD Group
  - **Plus any of the scripts in the Scripts folder of VTOLVRModdingTools.**

**Step 5: Test the object in Play Mode**

- Some scripts, such as RotationAnimator and DestructionDebris can be tested in Play Mode to see how they behave.
- Create and/or move the main camera to view your object and press play.
- Be sure to exit play mode before trying to edit your object.

## Part 4: Exporting

Once you have finished creating your object or want to test it in-game, use the StaticObjectExporter script to export it.

**Step 1: Export**

- Select your object's root where the StaticObjectExporter script is attached. You can select multiple objects if applicable.
- Click Export
- If it's the first time exporting, it will prompt you for the location to save it.

- You can save it anywhere, but the most convenient way is to export it directly to the folder where VTOL VR will look when importing custom objects. This will be in the **CustomStaticObjects** folder in the game folder. To find it, right-click VTOL VR on Steam, then Manage→Browse Local Files. If the **CustomStaticObjects** folder isn't there, you must create it.

**Step 2: Solve errors**

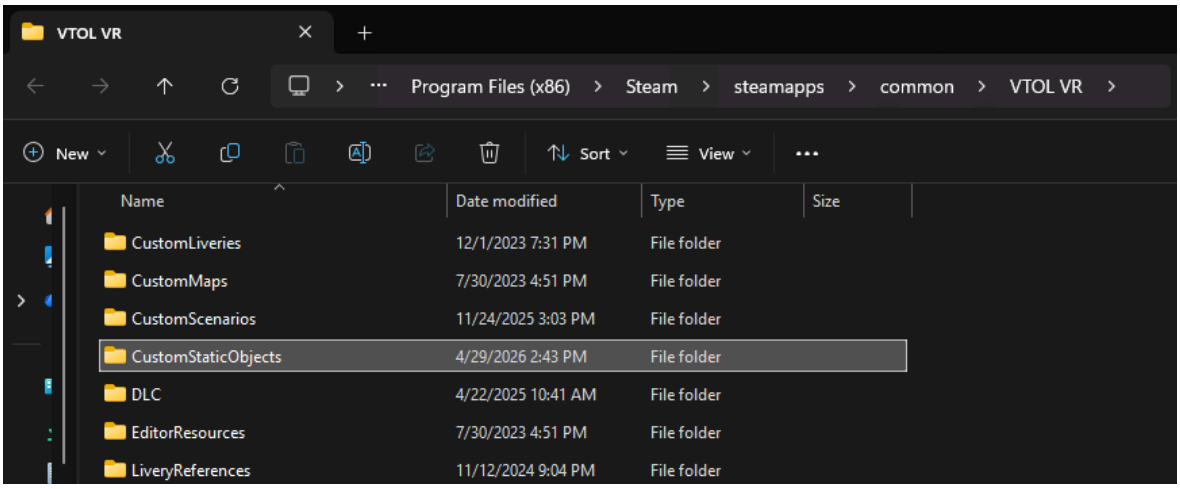
- If there's anything wrong with how your object is set up, whether it's missing references, or has invalid components/materials, an error popup will appear telling you which object has a problem. Fix the issues then try exporting again.

## Part 5: Importing to VTOL VR

Importing custom static objects is as simple as moving the pack into the correct folder.

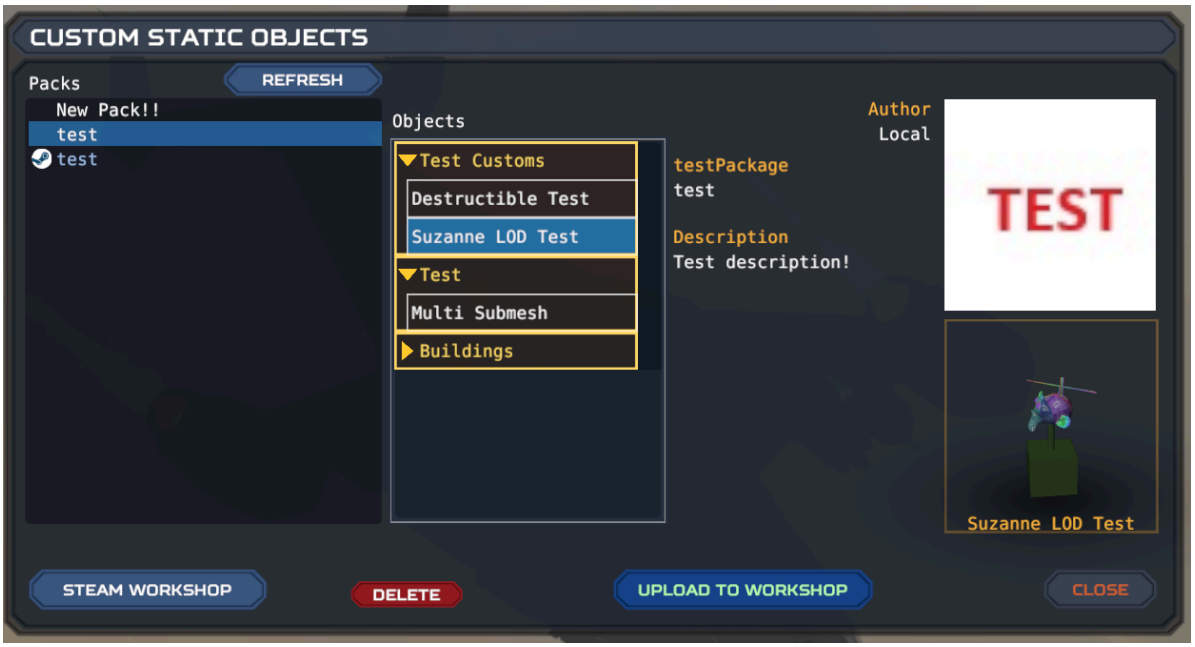
**Step 1: Put the CSO pack folder in the CustomStaticObjects Folder**

- If you haven't saved it there already, just move the folder with the name of your CSO pack (it should include an info.csop file, Materials subfolder, and subfolders for each of the static objects in your pack) into the **CustomStaticObjects** folder in the **VTOL VR** game folder.
- To find it, right-click VTOL VR on Steam, then Manage→Browse Local Files.
- If the **CustomStaticObjects** folder isn't there, you must create it.




**Step 2: Check if the pack was imported**

- In VTOL VR, go to the VTEDIT main menu, and click on Custom Static Objects
- You should see your custom pack on the left menu. Click on it to see the objects in the pack.




**Step 3: Place the Object(s) into a custom mission**

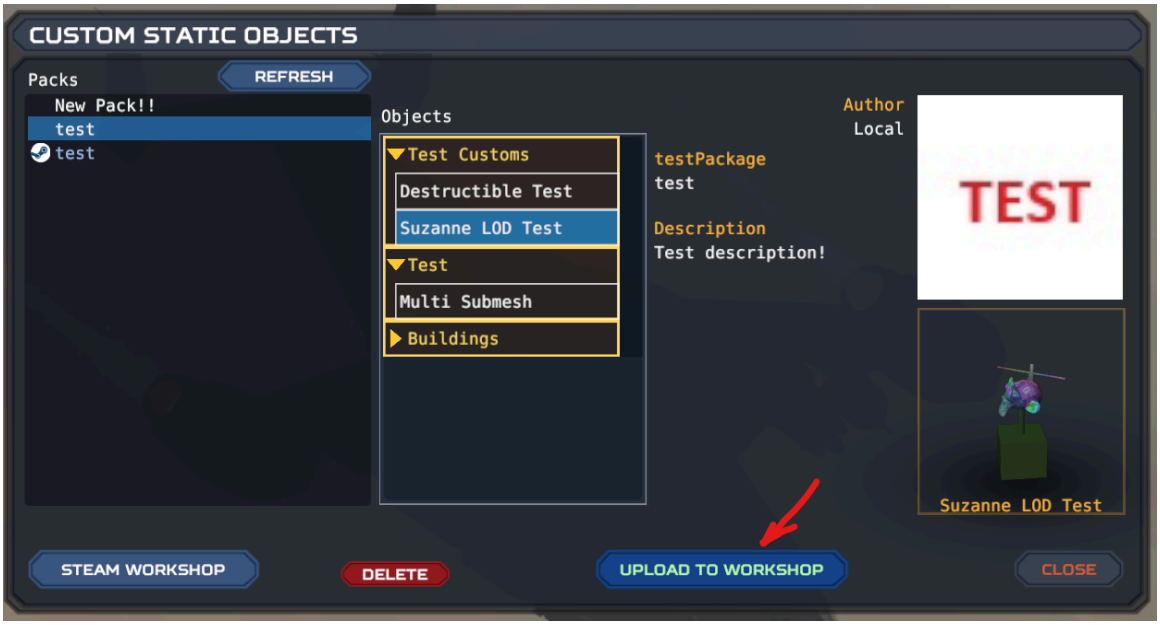
- Create or open a custom mission, click on the Static Objects tab, create a new object, and you will find your CSO pack there.

## Part 6: Uploading to Steam Workshop

Once you have verified that your static object pack is working properly, you can share it with the world on Steam Workshop.

**Step 1: Upload from the CSO Menu**

- Go back to the main VTEDIT menu and click on Custom Static Objects.
- Click on your pack on the left.
- Click Upload to Steam Workshop.



# Destructible Objects

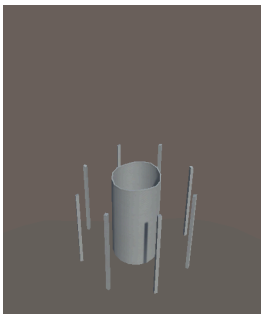
Custom static objects can be made to be destructible. This is great for mission objectives, storytelling, cinematic effect, or just creating an immersive environment. They take a bit of extra work to set up, so this section will guide you through it.

## Modelling

There are 3 categories of models you will need:

1. Alive static object
2. Destroyed static object
3. Debris

**Alive static object**
This model is the object in its normal state, undamaged. In this example, a water tower.

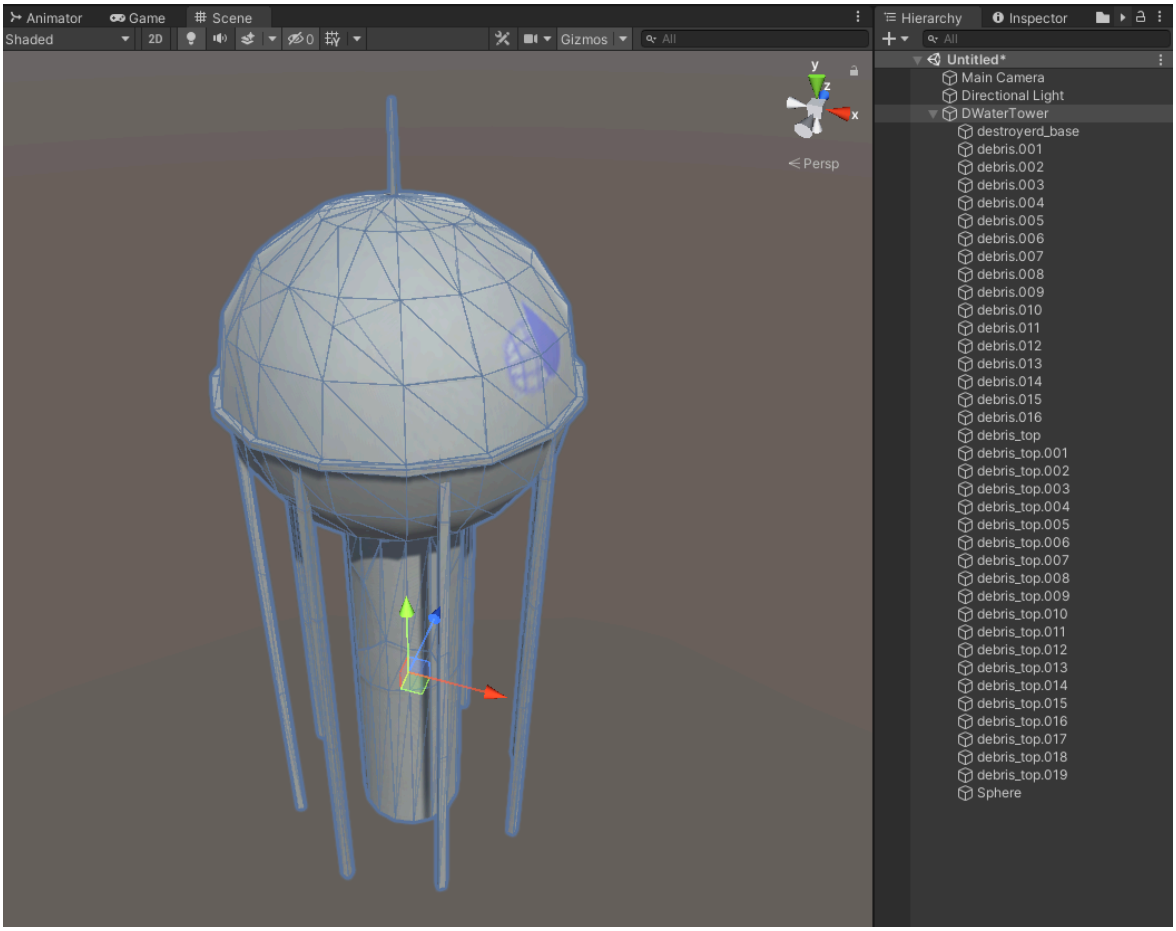


**Destroyed static object**
This is the model in its destroyed state. This is the part that will be left behind when the dust settles. This is usually the base of the structure, which will remain embedded in the ground.

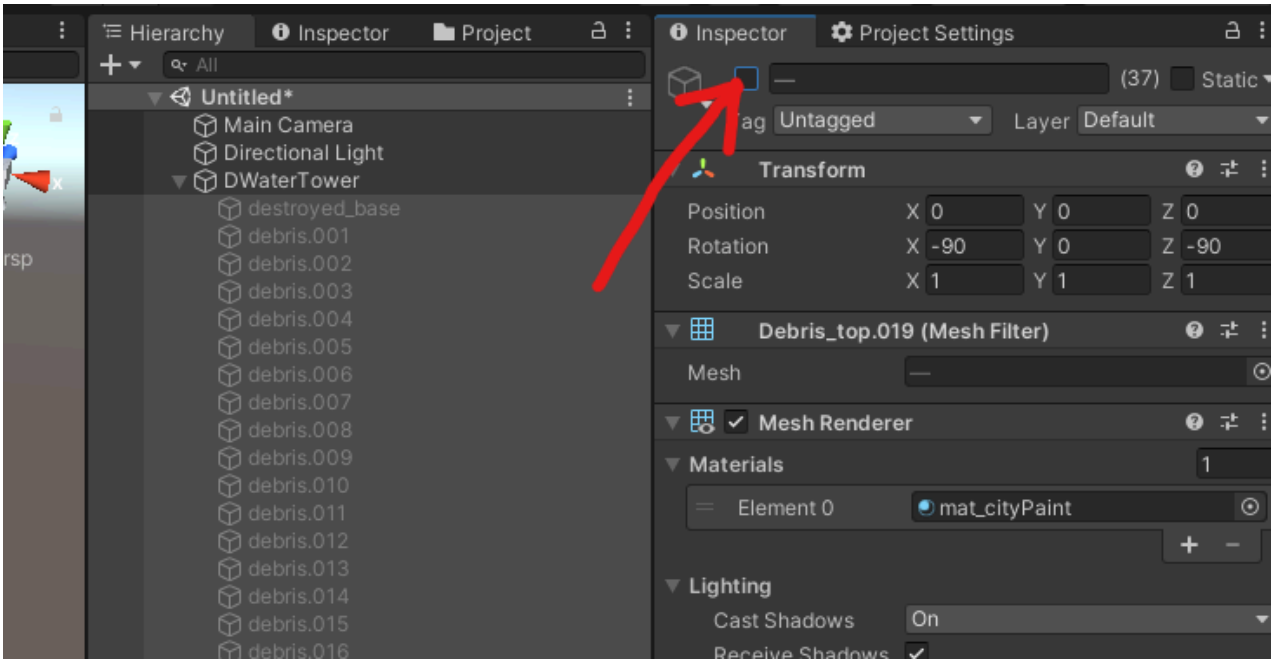
**Debris**
Debris objects will be physically thrown around by the destruction explosion and will despawn after some time. They are not strictly necessary but it helps to visually transition from the normal object to the destroyed object state.

## Setting up the Destructible CSO

First, import all of the models and set them up the same way as a standard CSO. The destroyed/debris objects should be overlapping the alive model.



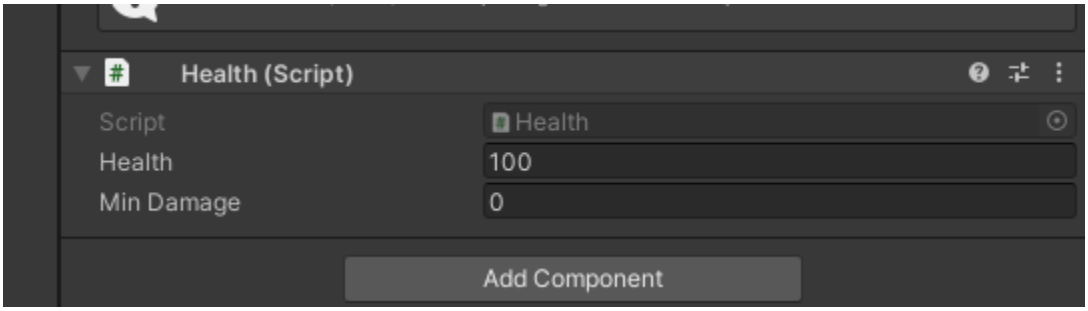

**Untick** the check mark in the Inspector next to the name of the destroyed/debris objects in order set them **initially disabled (hidden).**



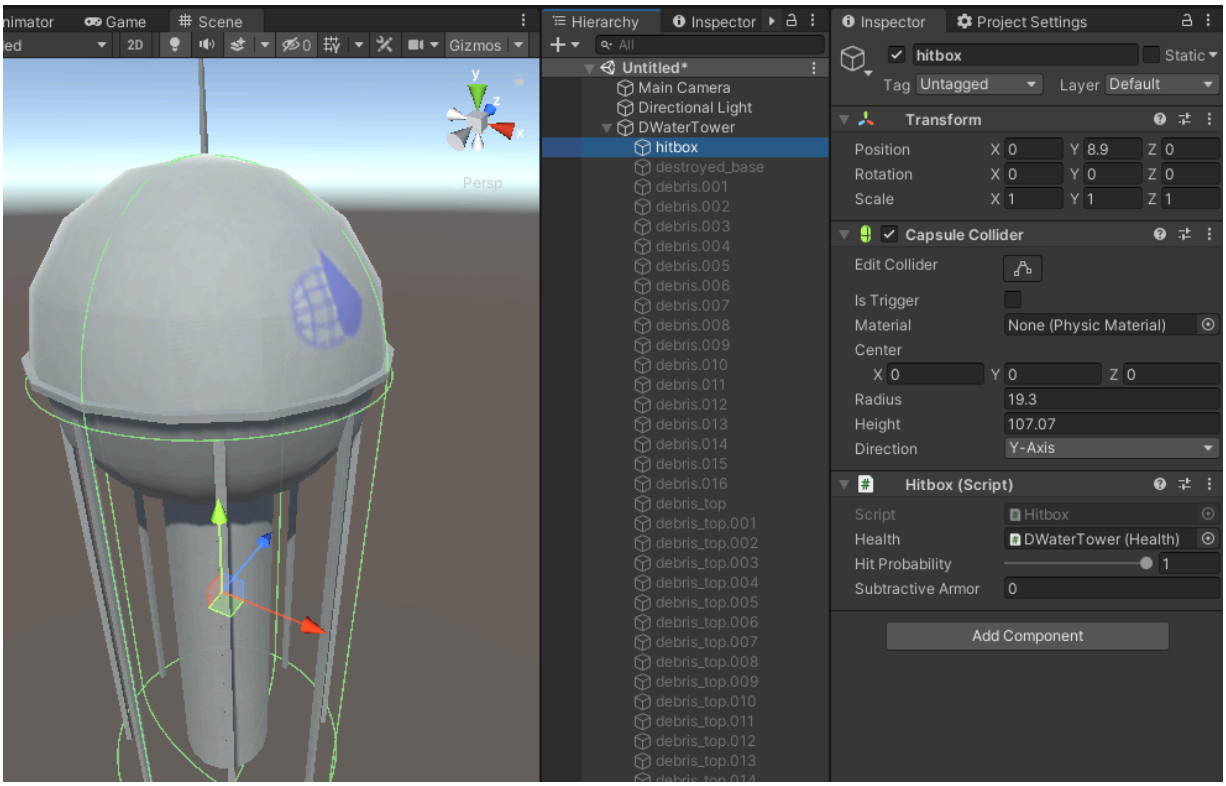
## Components (Scripts)

There are a few components necessary for completing a destructible CSO:

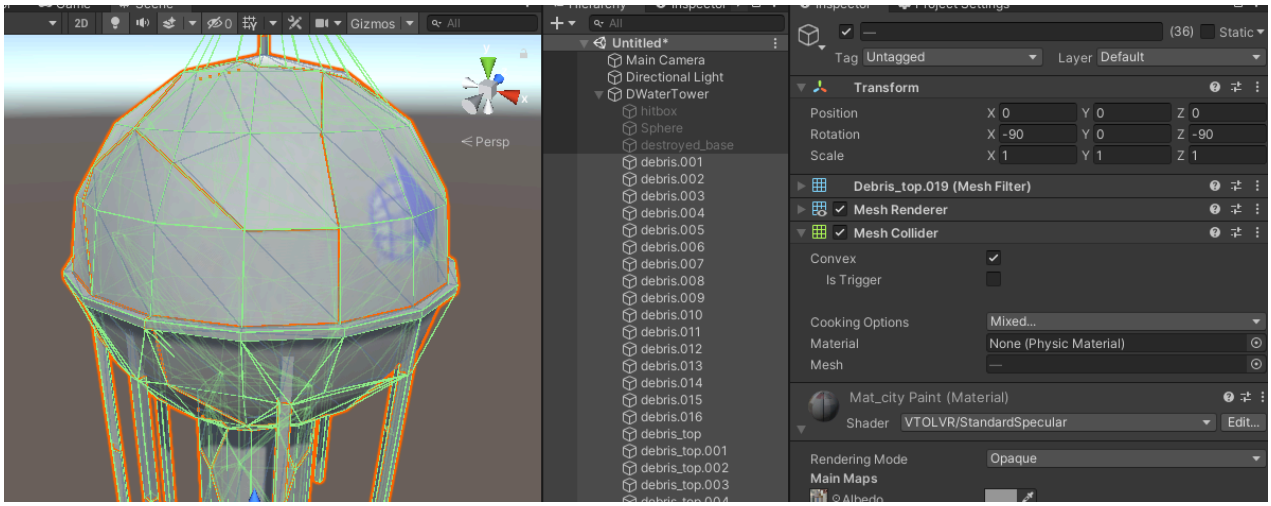
1. **Health**
    a. The **Health** component should be added to the root object (same as StaticObjectExporter)
    b. The Health component defines how much damage must be dealt to the object in order to destroy it.
    c. It also has a property for Min Damage which is how much damage must be dealt at once in order to do any damage to it. This can be used to prevent weaker attacks (like small arms fire) from slowly chipping away and eventually destroying a hardened target which should require heavy ordnance.



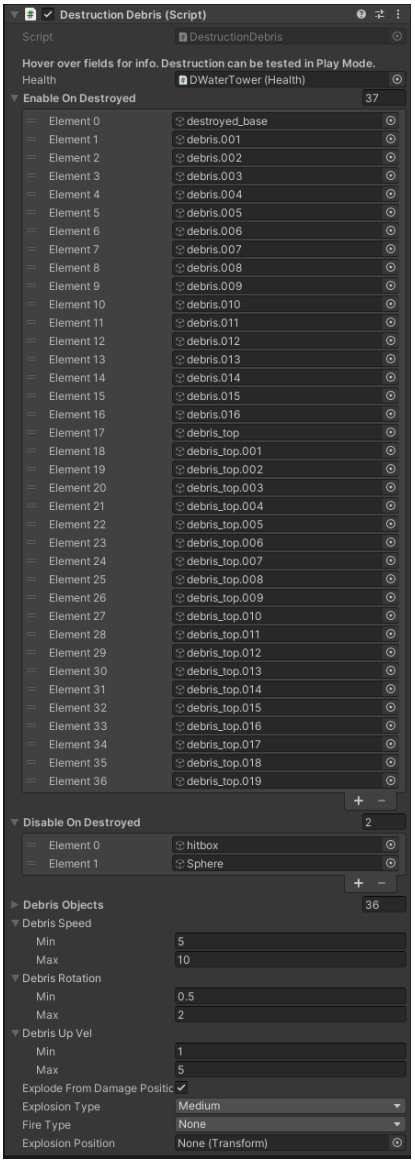

2. **Hitbox + Collider**
    a. There needs to be at least one **Collider** that can be attacked by explosions or bullets in order to destroy the object.
    b. Create or select an object in the Alive object with a collider, and add the **Hitbox** component. Hidden destroyed/debris objects do not need hitboxes.
    c. Assign the previously mentioned Health component to the Health slot in the Hitbox component. Any damage done to this hitbox will take hit points from that Health component.



3. **Debris colliders**
    a. Each piece of debris will need a collider if you want it to land on the ground properly. Since these are just visual effects and physical accuracy is not important, it will be best for performance to use primitive colliders such as sphere, box, or capsule colliders. If they must have mesh colliders, they should be set to Convex.




4. **Destruction Debris**
    a. This component is where you define the alive, destroyed, and debris models.
    b. Add the **Destruction Debris** component to the root object.
    c. Enabled On Destroyed: drag **destroyed** models here (including debris), which should be revealed when the object is destroyed
    d. Disable On Destroyed: drag **alive** models here, which should be hidden when the object is destroyed
    e. Debris Objects: drag each **debris** object here.
    f. Debris configuration: Set values to Debris Speed, Debris Rotation, and Debris Up Vel to define how the debris will fly out when the object explodes. Hover over these properties to view the tooltip for info.
    g. Effects: Choose explosion and fire effects to use when the object is destroyed.




## Testing Destruction

Once your object is set up, align the Main Camera to view the object and enter Play Mode with the play button (or Ctrl+P). This will allow you to preview how it will look when switching from alive to destroyed models, and how the flying debris behaves. The explosion and fire effects will not be visible yet.