**Clayxels Documentation**

Official Twitter Account: twitter.com/clayxels
For support please use our discord server: discord.gg/Uh3Yc43
Created and owned by Andrea Interguglielmi twitter.com/andreintg
For any non-technical kind of enquiry, please get in touch directly via email: ainterguglielmi@gmail.com

**Table of Contents**



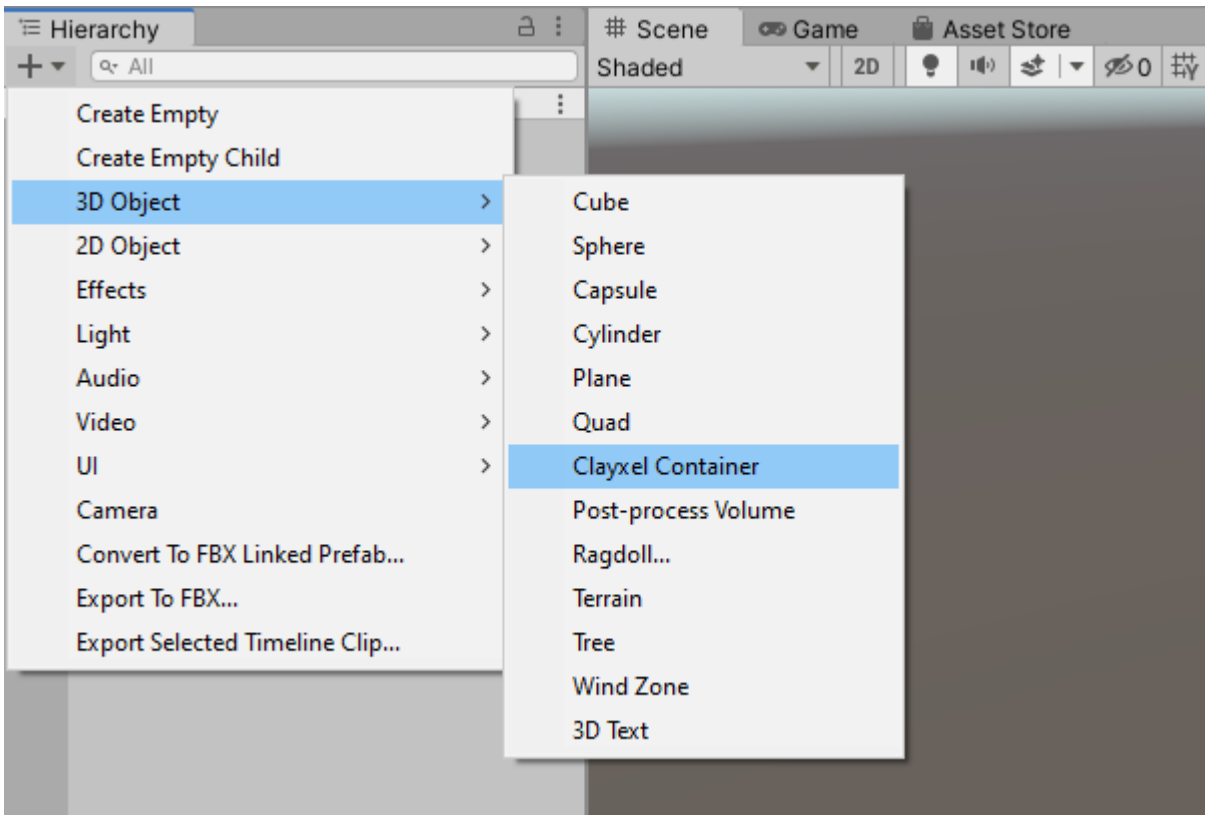
**Introduction**

Hi! Welcome to Clayxels, whether you're here to sculpt assets or you plan on using its volumetric capabilities in your game, follow this guide and you'll be using this tool in a matter of minutes. Working with volumetric clay can be as liberating as it can be disorienting at first, just allow yourself some time to familiarize with this new workflow.

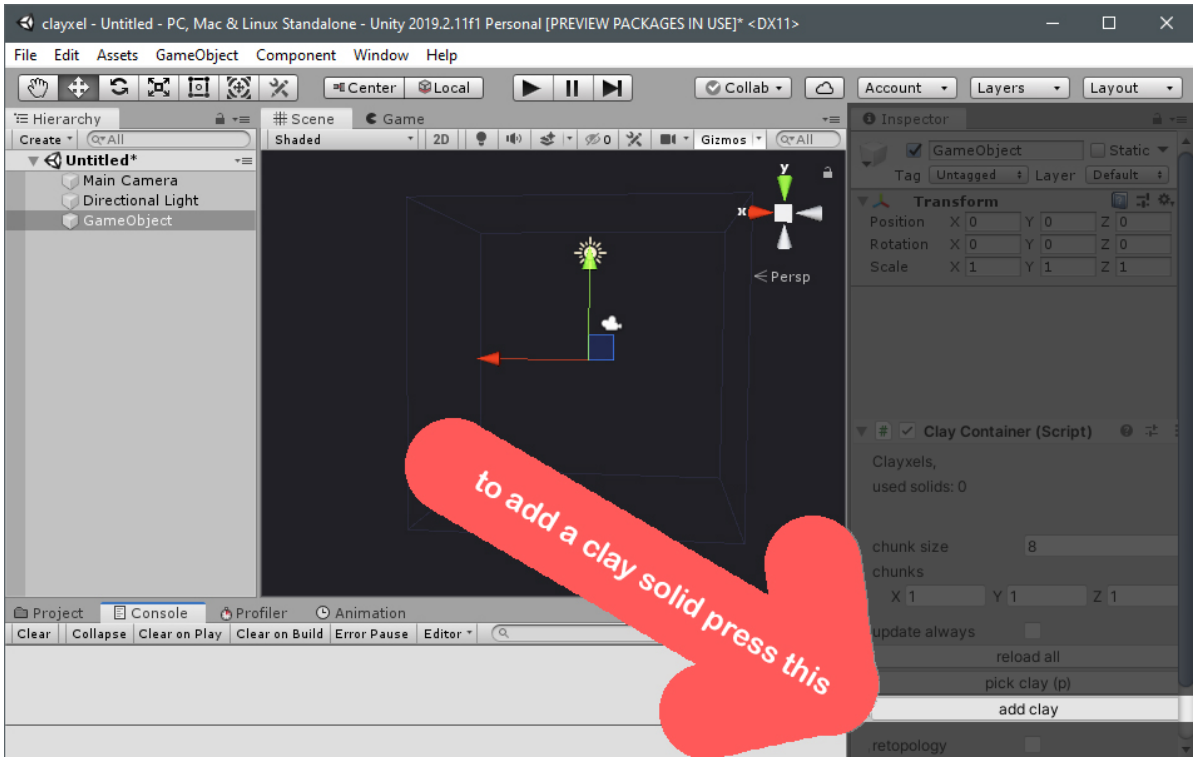

*(image captured straight from Unity, HDRP render pipeline)*

## Getting Started

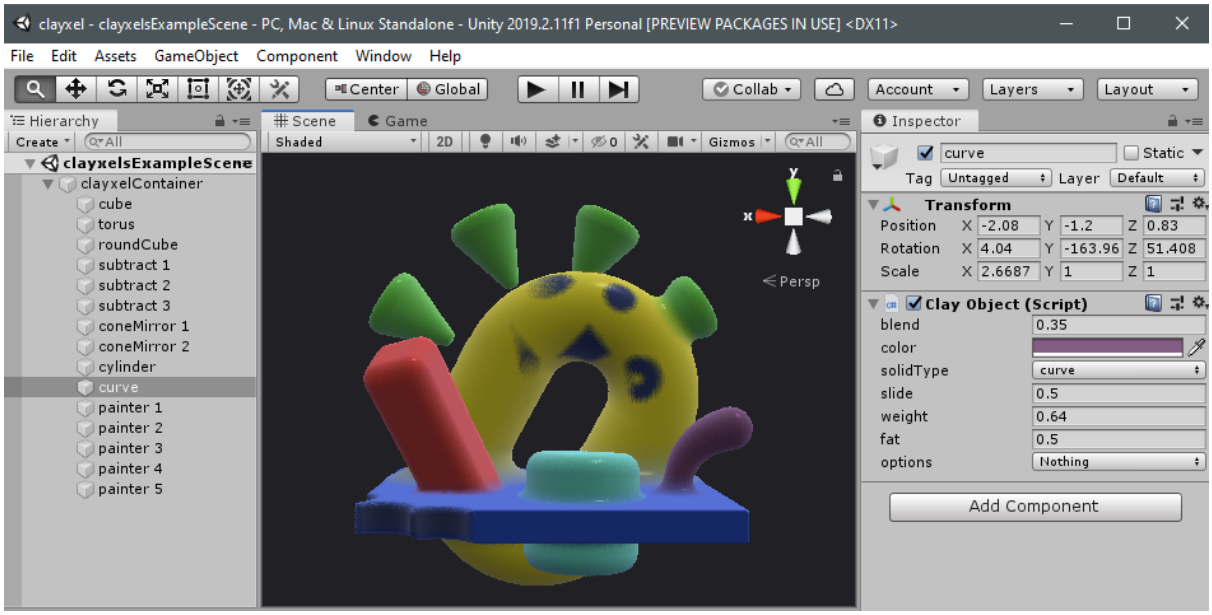
Getting started with Clayxels is simple, first create a ClayContainer using the sub-menu located under 3D Objects.




Now that you have a ClayContainer you can start nesting ClayObjects under its hierarchy.
To add a ClayObject press the "add clay" button, it's located in the custom inspector that shows up whenever you select a ClayContainer.



ClayObjects have a few options of their own showing in the inspector. You can change their shape, the blend value to add or subtract shapes together, change their color and other properties that will vary depending on the solid-type.




Please check the tooltips on the inspector, that's the best way to learn this simple UI.
A list of examples is provided under the examples folder in the form of packages that you can import depending on your current render pipeline.

**Render Modes**

(example scene for URP showing all render modes)

Clayxels has three render modes, each one with unique capabilities.

- *PolySplats*:
Generates a point cloud made of lots of camera-facing triangles with a texture, it's ideal for vegetation and effects where big crispy splats are necessary.
Available on all render pipelines.

- *MicroVoxelSplats*:
This render-mode is the most light-weight, it doesn't use polygons at all, thus avoiding bottlenecks that come from having many dense meshes in the scene.
You can tweak the actual render resolution independently from the viewport using clayxels' global config.
Only works in URP and HDRP render pipelines.

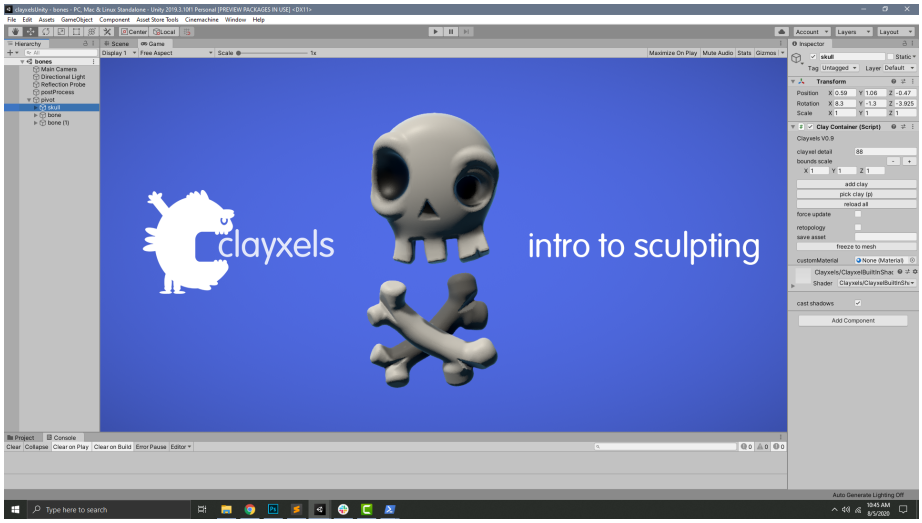
(example scene for HDRP using microVoxelSplats to achieve many different effects without polygons)

- *SmoothMesh*:
This mode outputs a mesh in real-time, it's the heaviest mode for real-time changes, but it's convenient if you need a simple mesh that integrates nicely with your existing scene and it's easy to create custom shaders for.
This mode also offers a voxelizer option to turn your sculpt into blocky shapes.
Available on all render pipelines.

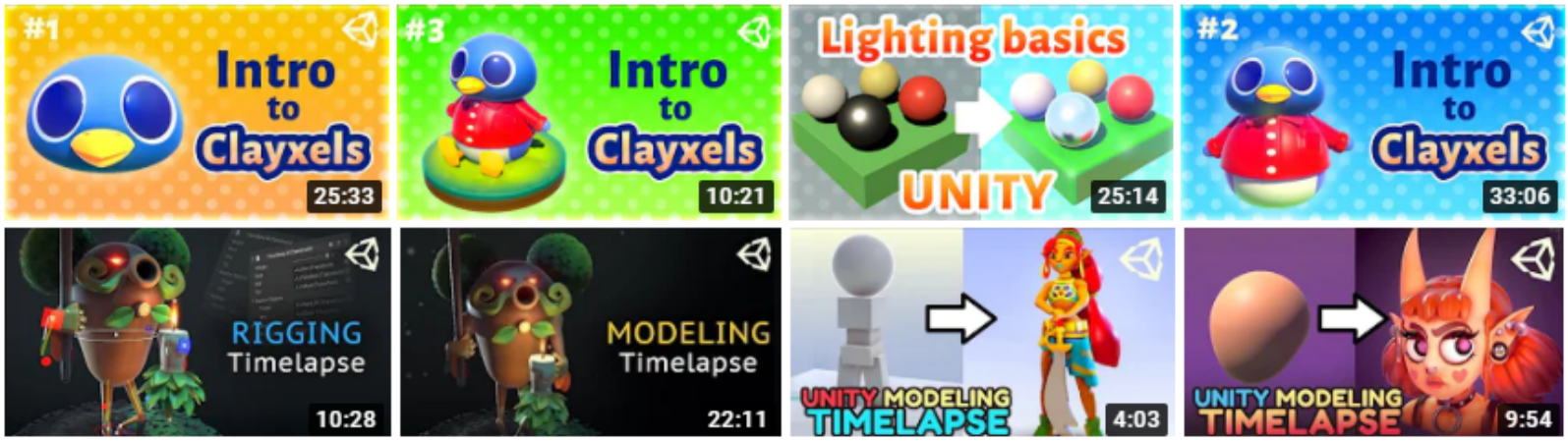
**Usage Tips**

- Duplicate clayObjects with Ctrl-D rather than using the Add Clay button each time.
- Drag clayObjects  up and down the hierarchy to change how they affect each other (clayObjects influence each other from bottom to top in the hierarchy).
- Split your sculpt into many containers to make large and complex models.
- Disable auto-bounds and work with small bounds that fit your sculpt to improve performance.

Here's a basic tutorial to get you started on sculpting with Clayxels:
https://youtu.be/pJOk7aRLYzY




Make sure to also check Alex Strook's youtube channel !




Don't forget to have a look at the examples shipped with the package, Clayxels can do much more than just sculpting static assets, the examples are the best way to learn some of the extra features.
You can find more user-made tutorials in the resources section of our discord channel:
discord.gg/Uh3Yc43

**Freezing to mesh and exporting FBX files**

Your sculpts can be frozen to a mesh at any time, and if you have nested containers the whole hierarchy will be frozen/unfrozen.
If you need for whatever reason to cleanup or simplify your topology in another application, you can do so by exporting an FBX file.
You can enable the FBX exporter package in Unity as described in this guide:
https://learn.unity.com/tutorial/working-with-the-fbx-exporter-setup-and-roundtrip-2019#

## Animation and Rigging

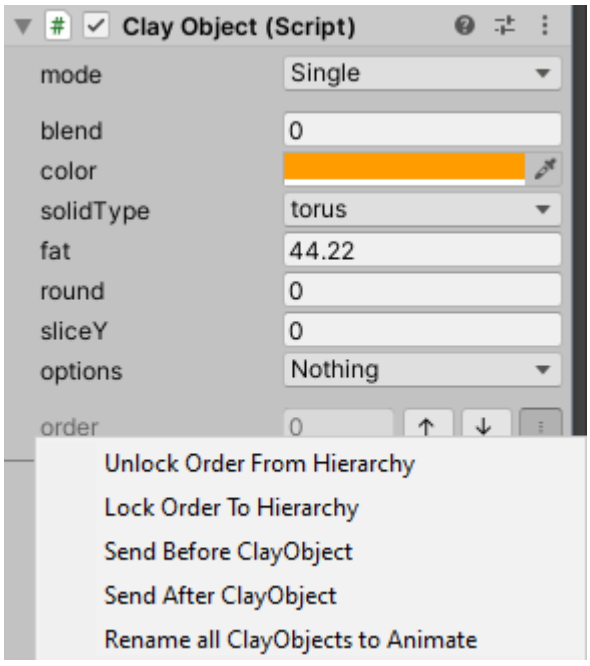
Animating with clayxels doesn't necessarily involve exporting your sculpt as a mesh and performing manual weight-painting with external tools.

**Rigid Parts:**
Regardless if you decide to freeze your containers or not, the most efficient way to rig a character is to divide it into limbs using many separate ClayContainers and parent each container/limb under your rig's bones.
This method is simple and doesn't require skinning, but of course you will see the seams separating each limb.

**Auto Rigging:**
Clayxels has an auto-rig button available when you freeze a container to a mesh. When using the auto-rig functionality all your clayObject blends will be used to generate the skin weights, therefore your clayObjects will become the bones of the rig itself. Note that when a clayObject is set to offset or spline mode the auto-rig will generate a new set of bones to gain fine control over the model.
*Important: Auto-rig will generate a lot of overlapping weights, it's best to increase the max amount of bones per vertex from the Unity settings window.*

**Live Clayxels:**
By setting your ClayContainer to interactive (the first button in the inspector), you can animate your clayObjects directly, you can organize them in hierarchies within your container and facilitate keyframed or procedural motion, just follow these steps:

1) By default your clayContainer will use the hierarchy order as a way to determine how each clayObject blends with another (you can see an example of that in the video tutorial from the previous section). To freely organize your clayObjects into hierarchies without risking to compromise a sculpt, select any clayObject and enable "Unlock Order From Hierarchy".




2) Now you can create empty gameObjects and nest your hierarchy as you need for animating it.
Once you're happy with your hierarchy, use the same menu to click "Rename all ClayObjects to Animate". This will generate unique names for all your clayObjects, which can come in handy when you use a timeline.
*Note: clayObjects are auto-renamed only if their name starts with "clay_".*

## Optimizing

Clayxels is a GPU heavy system, but there are many ways to optimize it and eventually reduce its impact on the scene, it mostly boils down to whether you need interactive shapes or not.

**Non-interactive:**
By default clayContainers are non-interactive, these are used when you don't need to change the clay shapes while the game is running. This workflow is the easiest to handle, as it will not impact your game's runtime performance.
Clayxels will re-compute the point cloud only when you move one of the ClayObjects inside a ClayContainer. But we don't have to do that at runtime unless we need to. The easiest way to use Clayxels in a game and avoid slow downs, is to simply move/ rotate/scale the ClayContainer and never touch the ClayObjects inside it once the game starts.
Clayxels will automatically optimize each sculpt to use as little memory as possible and you can expect each container to perform like a regular mesh in your game.

**Interactive:**
If you plan on changing the content of your ClayContainers while the game is running, then all you need to do is set your clayContainer to interactive and you will be able to change the clay shapes at runtime.
One thing to note is that the bounds-size plays an important role when a container is interactive. To maximize performance, switch off auto-bound and try to fit your real-time animated sculpt into as few bounds as possible.

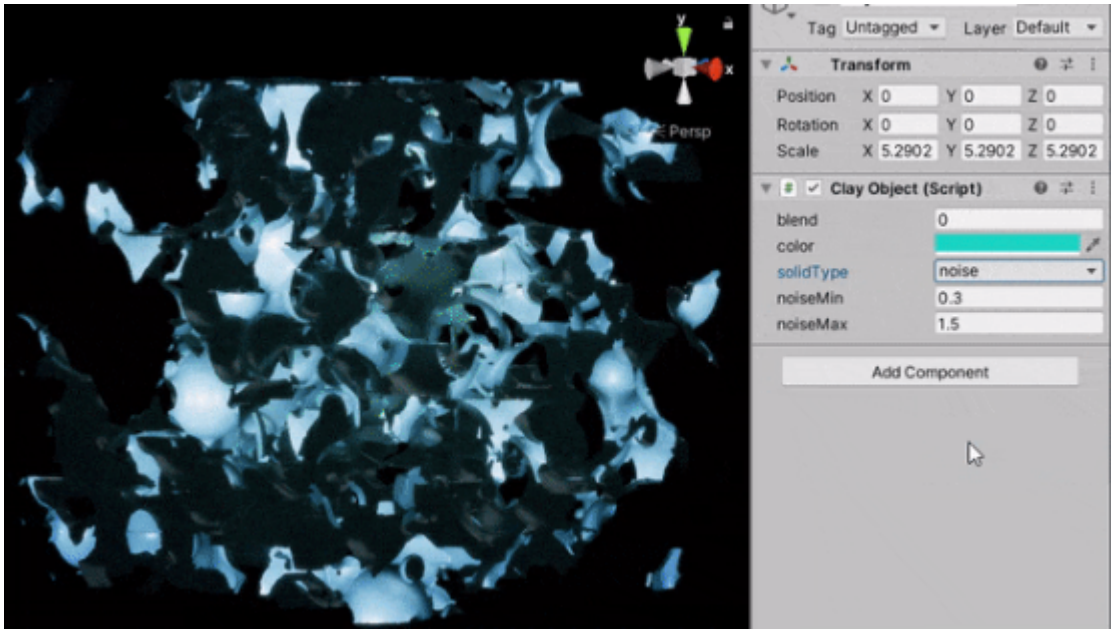
**Freezing:**
Freezing containers to plain standard meshes is the best way to avoid having clayxels impacting on your game at runtime. Once frozen, the clayxel runtime will never run on the target platform, making it possible to deploy your game on lower-spec platforms.

## Programming

Clayxels has three main parts that can be used from code, c# APIs, signed distance functions, shading.

**C# APIs:**
The c# apis of clayxels are defined in ClayContainer.cs and ClayObject.cs, clayxels has all its c# code open for those that need a deep understanding of its inner workings.
The high level public interface is kept at the very top of the file and each public member has been documented for clarity.

**Signed Distance Functions:**



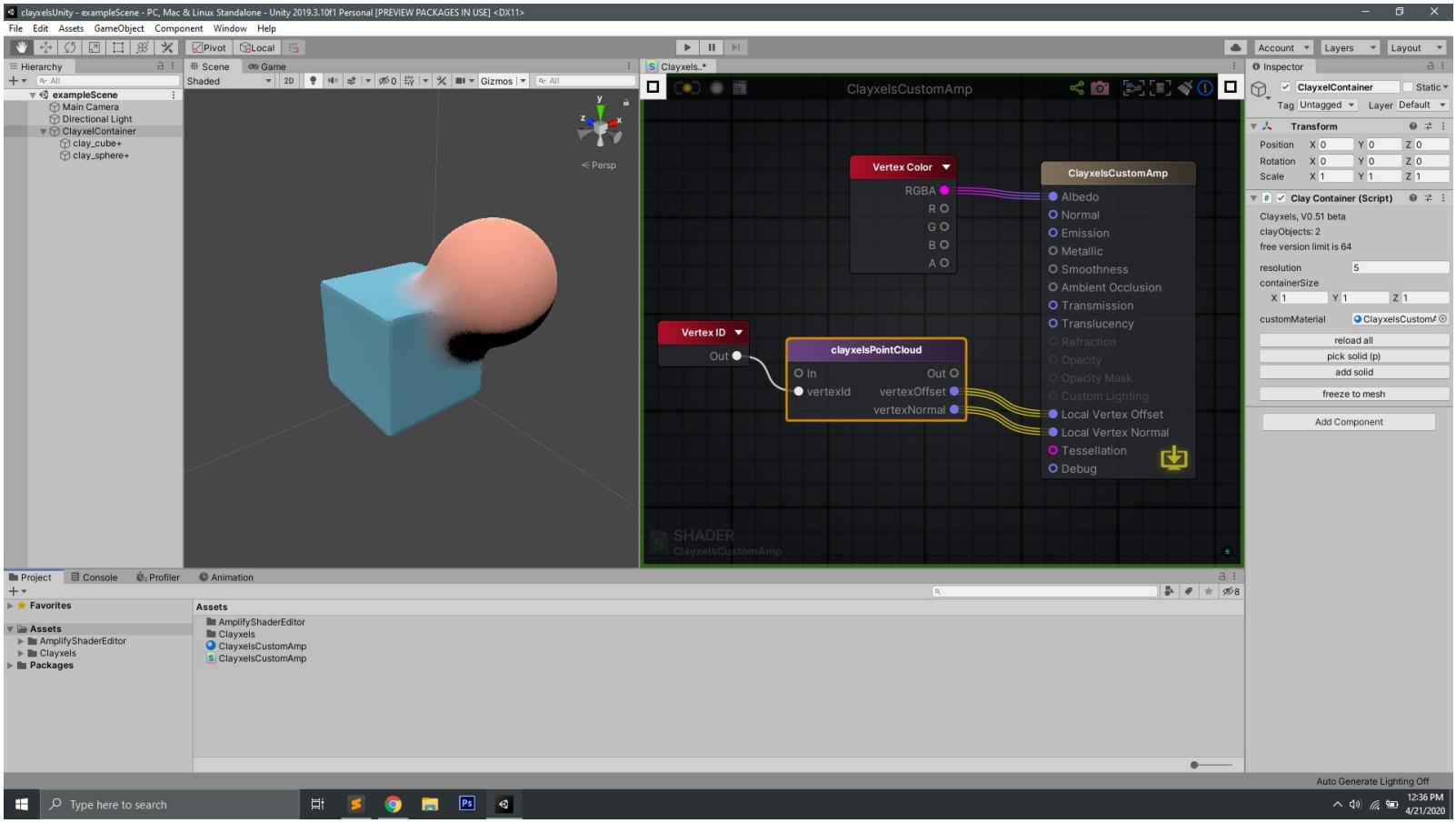
The core of clayxels is a compute shader that can be expanded via claySDF.compute, *computeClayDistanceFunction* is the single function responsible for handling all the volumetric shapes you see available from the inspector.
To expand clayxels with new solid-types all you need to do is chain your code within that function and declare a new interface.

Here's an example of how to do that (code to be added in claySDF.compute, *computeClayDistanceFunction*):

```
else if(solidType == 7){// label: mySDF, x: myParameter1 1.0
  // the comment above is auto-parsed from the inspector to register your custom SDF function
  // myParameter1 uses extraAttrs.x to get a value from the user, the inspector will show this parameter when your custom SDF solid is selected.
  // 1.0 is the default value we are using for myParameter1
  dist = length(pointRotated) - myParameter1; // we just made a sphere with a radius equal to myParameter1
}
```

Whenever you change something inside claySDF.compute remember to click "reload all" from a container.

**Shading**




Shaders in clayxels can be written via code or wired visually using Amplify Shader Editor*.
These shaders have to be specifically written to render the buffer coming from the compute shader.
The best way to start with shading clayxels is to refer to the examples provided and the default shaders for each render pipeline.
Adapting shaders to the different render pipelines is usually just a matter of starting from the default clayxels shader on a given render pipeline (example ClayxelHDRPShader.shader), and then copying the nodes over from one of the existing shaders (example ClayxelFoliageShader.shader) using Amplify Shader Editor.

* The reason why you can’t use Unity’s built-in shader-graph is that it lacks some of the nodes required by clayxels to function.

**Troubleshooting**

Clayxels crashing unity with a gpu timeout error:
- Open Clayxels Global Config (press the button on the clayContainer inspector), then lower “max bounds size”, try 2 or 1.
- Alternatively, disable “auto-bounds” while sculpting and manually set your bounds to be no more than 1,1,1 or 1,2,1 if you need more space to sculpt in one of the axis.
- Ultimately, this crash seems to happen on certain gpu models if your Windows OS settings are forcing an early timeout for your gpu, but given how each gpu might have different settings, we are unable to provide a final solution in this regard.

OS X:
If you experience a crash under Mac Os X intel (this workaround is not for M1 macs), or if Unity gives you a warning about clayxels exceeding the maximum allowed number of threads, open this file:
*Clayxels/Resources/claySDF.compute*
Edit the first line:
#define MAXTHREADS 8
Set it to 4:
#define MAXTHREADS 4

If you are still experiencing problems, please get in touch via email or on our discord channel, but please remember that we only support clayxels on M1 macs.