

# Clayxels Documentation

Official Twitter Account:

[twitter.com/clayxels](https://twitter.com/clayxels)

For support please use our discord server:

[discord.gg/Uh3Yc43](https://discord.gg/Uh3Yc43)

Created and owned by Andrea Interguglielmi

[twitter.com/andreintg](https://twitter.com/andreintg)

For any non-technical kind of enquiry, please get in touch directly via email:

[ainterguglielmi@gmail.com](mailto:ainterguglielmi@gmail.com)

## Table of Contents

- 1) Introduction
- 2) Getting Started
- 3) Usage Basics
- 4) Animating
- 5) Building your project
- 6) Optimizing
- 7) Programming

## 1) Introduction

Clayxels is an interactive volumetric toolkit to sculpt models in editor and in game. It doesn't rely on per-pixel ray-marching, it uses voxels to generate a lightweight point-cloud that can be used in a whole bunch of different ways. Works both in editor and in game, everything made with clayxels can be changed interactively at any time.

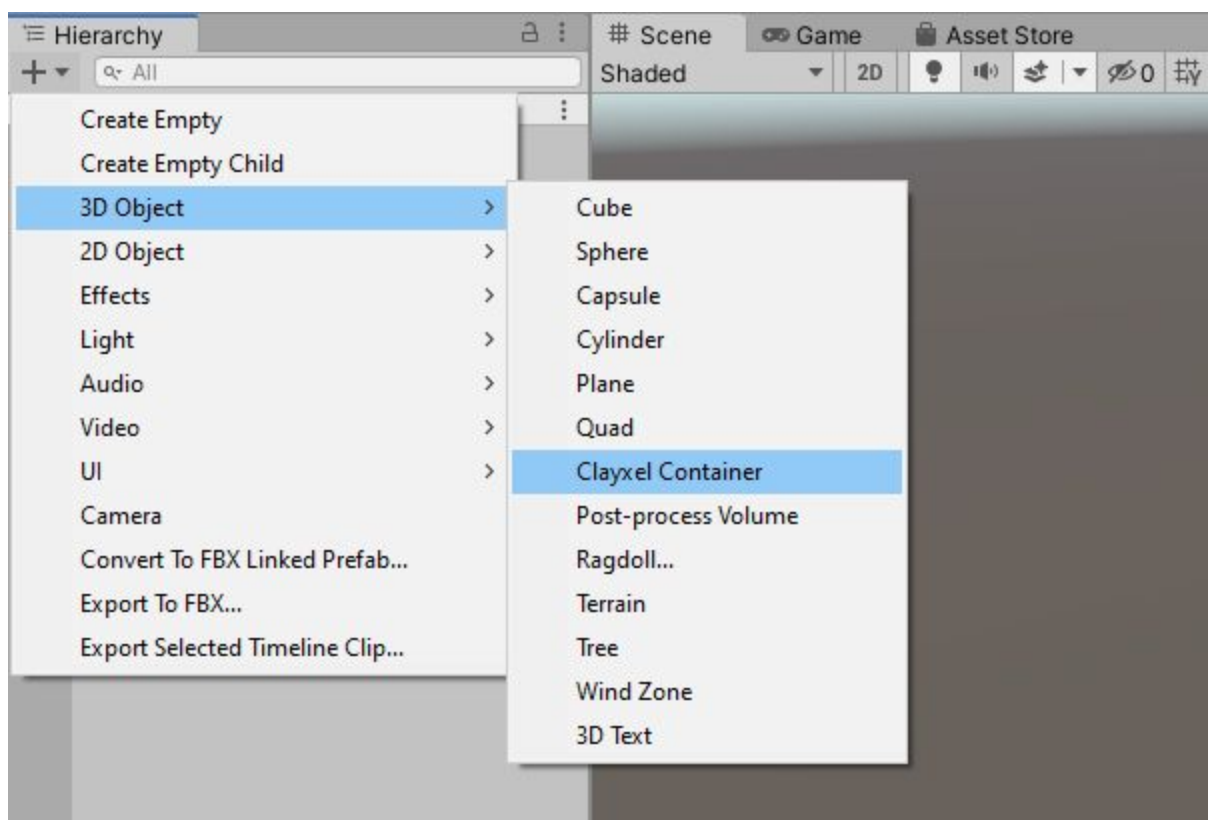


Interactive volumetric sculpting,  
in editor and in game.

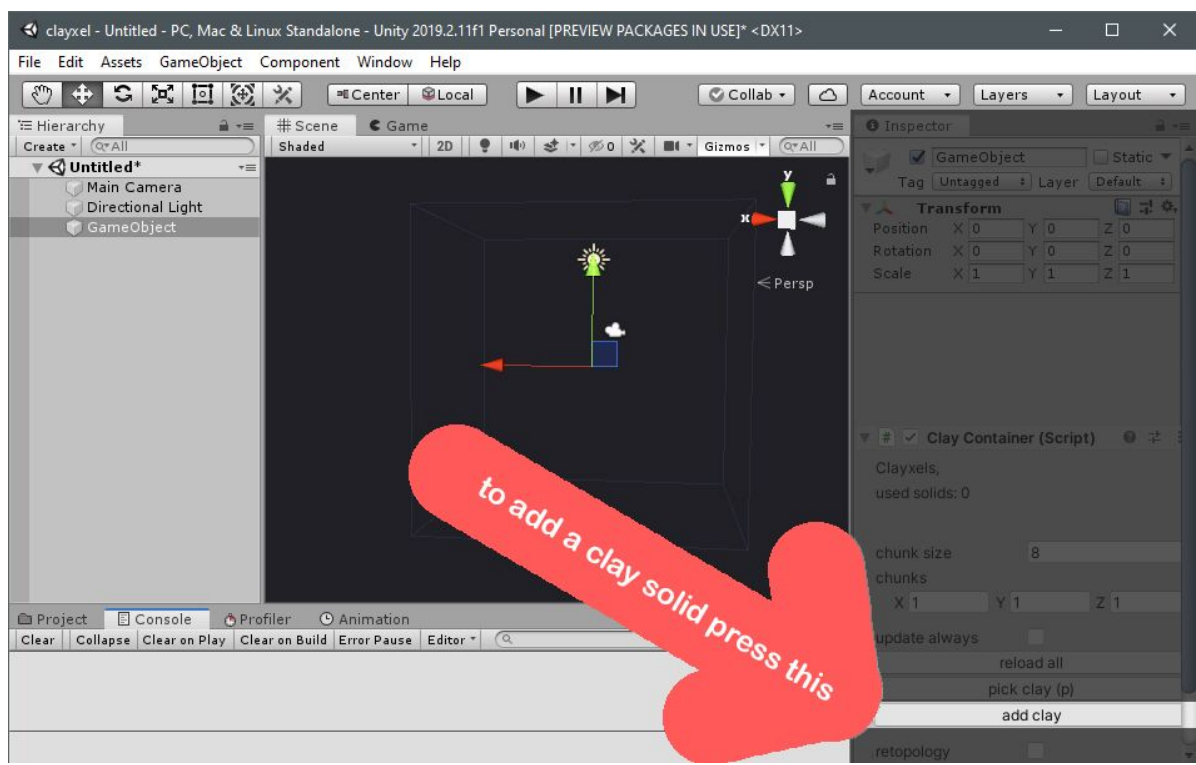
*(image captured straight from the editor, Unity 2019.3, HDRP render pipeline)*

## 2) Getting Started

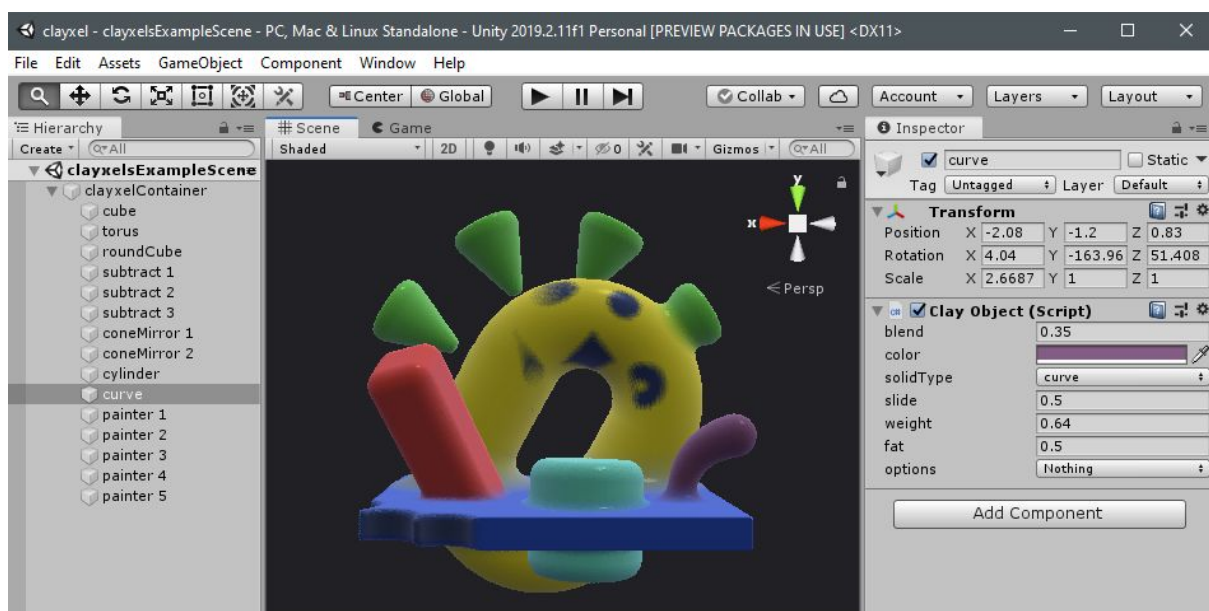
Getting started with Clayxels is simple, first create a ClayContainer using the sub-menu located under 3D Objects.



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.

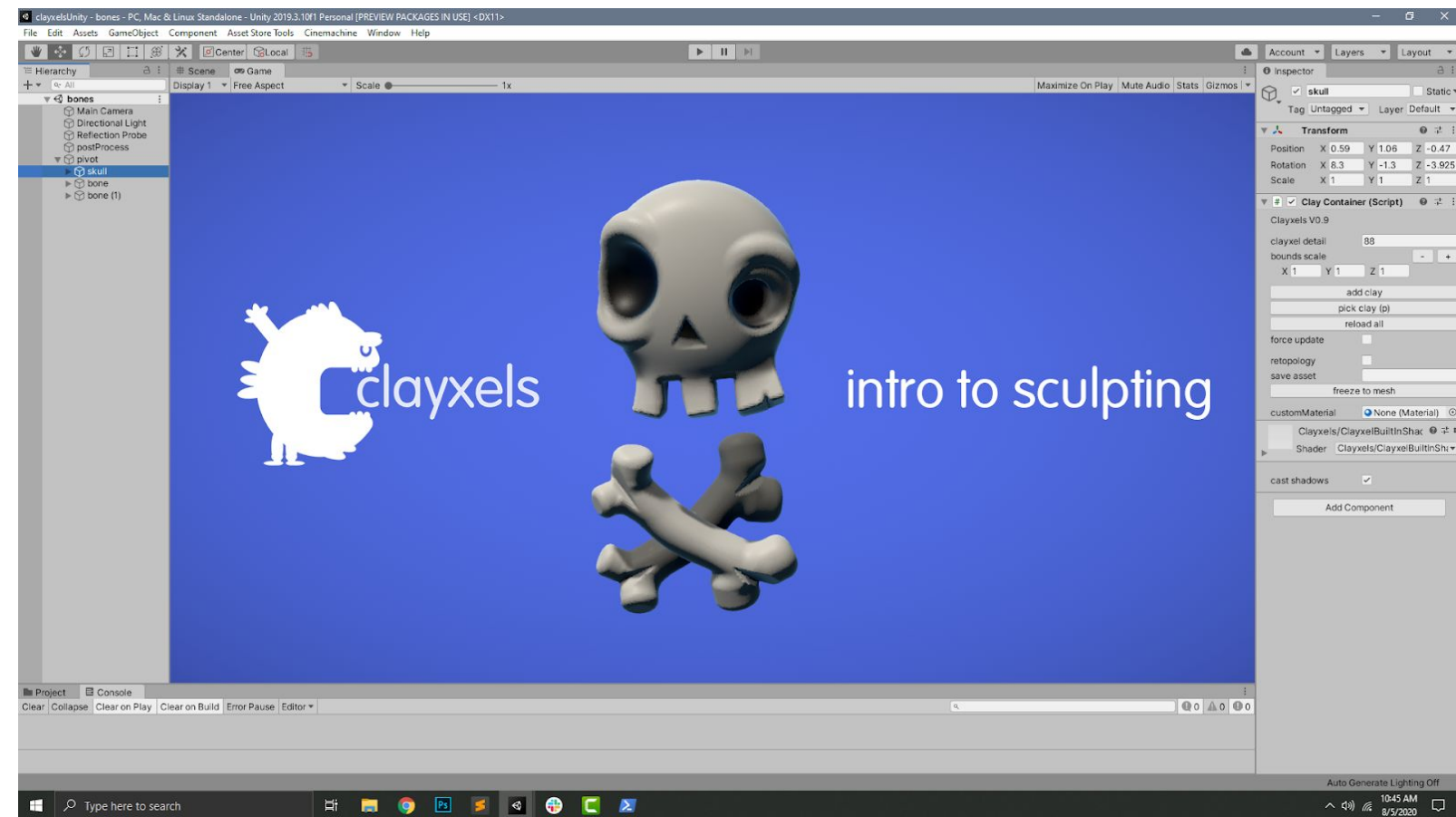
A list of examples is shipped with this package to showcase all the things you can do with clayxels.

### 3) Usage Basics

- Duplicate solids with Ctrl-D rather than using the Add Clay button each time.
- Press "p" on your keyboard to pick-select clayObjects directly from the viewport.
- Drag solids up and down the hierarchy to change how they affect each other (solids influence each other from bottom to top in the hierarchy).
- Split your sculpt into many containers to make large and complex models.

Here's a video tutorial to get you started on sculpting with clayxels:

<https://youtu.be/pJOk7aRLYzY>

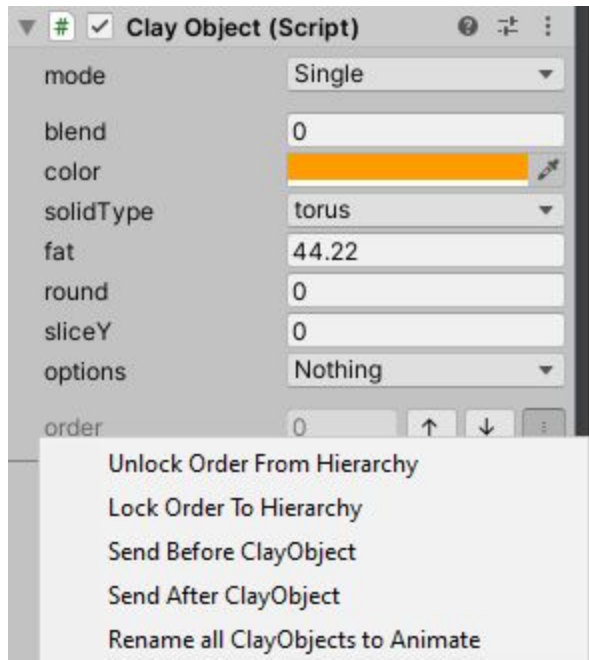


Just don't forget to have a look at the examples shipped with the package, Clayxels can do so much more than just sculpting static assets, the examples are the best way to learn all the different features.

#### 4) Animating

Animating with clayxels doesn't necessarily involve converting your sculpt to polygons and performing weights-painting. If you decide to not go the traditional mesh route, then the most efficient way to animate a complex character is to divide it into smaller clayContainers and parent each limb under your rig's hierarchy structure. You can also 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\_"*.

3) Once you are done with your hand-keyframed or procedural animation, you could just use that in your game, but as you will discover in the Optimizing section of this doc, having clayObjects move around while your game is playing, comes at great cost, and you should use this feature only if you really need interactivity. If all you need is playing an animation in loop, we suggest you freeze this container to a Claymation file.

When you freeze to claymation you have the option to specify an animationClip where you stored your keyframes, or you can programmatically record procedural motion with this api:  
`ClayContainer.bakeClaymationFile("myClaymatonFilename", startFrame, endFrame, myProceduralMotionCallback);`

There's more on the claymation file in the Optimizing section.



## 5) Building your project

After you import clayxels to your project, please remove the unnecessary render pipelines folders that you won't need. These folders are located inside Clayxels/Resources, here you will find: RenderURP, RenderHDRP, RenderBuiltIn. For example if you are on a URP project, just remove Clayxels/Resources/RenderHDRP and Clayxels/Resources/RenderBuiltIn.

## 6) Optimizing

Clayxels is a GPU heavy system, but there are many ways to optimize it and eventually reduce its impact on the scene to the point of making it just as light as any other polygonal mesh. It mostly boils down to whether you need interactive shapes or not.

### Non-interactive:

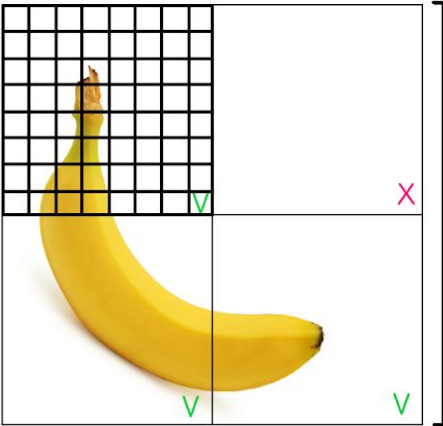
We refer to non-interactive clayxels when you sculpt from the editor but you don't change the clay shapes while the game is running. This workflow is the easiest to handle, as it will not impact your game's runtime performance. Clayxels will re-compute the point cloud only when you move one of the ClayObjects inside a ClayContainer. But we don't need to do that at runtime unless we want 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 as fast as any other mesh in your game.

### Interactive:


If you plan on changing the content of your ClayContainers while the game is running, then it's important to understand how to use the bounds of your container effectively. The following scheme is for artists and developers to understand how the underlying voxel-grid impacts your game when clayxels updates the point-cloud.

What is a ClayContainer: a 3d grid of voxels, internally it's split into sub-grids to handle big sculpts with little memory.


clayxelDetail attribute:  
Determines the size of the single sub-grid.  
When the detail is high the grid becomes small, so you might need to enlarge the boundsScale to fit a large sculpt.



boundsScale attribute:  
Determines how many sub-grids we want in this container.  
In this picture the container has 2 sub-grids on the X axis and 2 on the Y axis.  
This sculpt will only occupy 3 sub-grids, we marked them in green V.  
Internally Clayxels will only process the sub-grids that have something inside.



A high clayxelDetail will make your dots smaller but you will need a bigger boundsScale.



For optimal performance try to stick to boundsScale 1,1,1 and increase clayxelDetail until your sculpt fits just right. Containers with bigger boundsScale will be heavier on GPU.

## Freezing:

Freezing containers is yet another way to make clayxels more transparent on the final game executable.

Once frozen, the clayxel runtime will never run on the target platform, making clayxels possible to deploy on lower-spec platforms.

You have two options to freeze your sculpts and they are both available from the ClayContainer inspector:

- freeze mesh: it will freeze your container to a plain standard mesh that you can export using Unity's builtin fbx exporter.
- freeze claymotion: this is a special file format that preserves your point cloud, weights less than a mesh, lets you use the same shader you use for live clayxels, and most importantly it lets you bake an animation clip.

## Claymotion baked files:

Because of how compact clayxels data is, it seemed like a good idea to bake it into its own file format.

Claymotion files are easily baked from a ClayContainer by pressing the button "freeze claymotion".

A Claymotion component is automatically added and the same material used to shade your baked point cloud.

You can even add the Claymotion component to an empty GameObject to play any claymotion file in your project.

Why this is great:

Clayxels is a gpu intensive system, for this reason having too many containers updating at the same time with some animated content, can and will become problematic on most consumer hardware.

- Claymotion files don't rely on the gpu aside from drawing dots on the screen.
- Not having compute-shaders means lower-end hardware can be targeted more easily.
- You can use the same shaders used to draw live clayxels, even foliage or anything custom you can come up with as long as it's compatible with clayxels' point-clouds.

## Fine tuning performance from clayxelsPrefs.cs:

- *ClayContainer.setMaxSolids*: keep it to a minimum to reduce video memory consumption.
- *ClayContainer.setMaxSolidsPerVoxel*: keep it to a minimum to improve runtime performance. Avoid placing too many clayObjects in a single voxel if you set this to a small number.
- *ClayContainer.setUpdateFrameSkip*: try skipping as many frames as possible to increase FPS in your game.

## 7) Programming

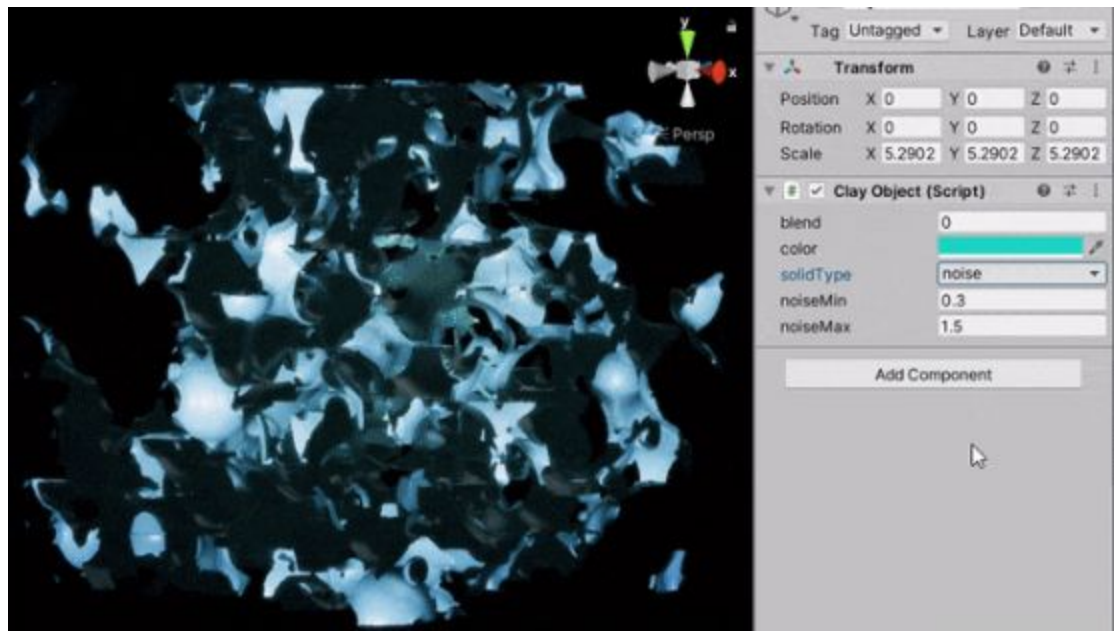
Clayxels has three main parts that can be used from code, c# APIs, signed distance functions, shading.

### 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.

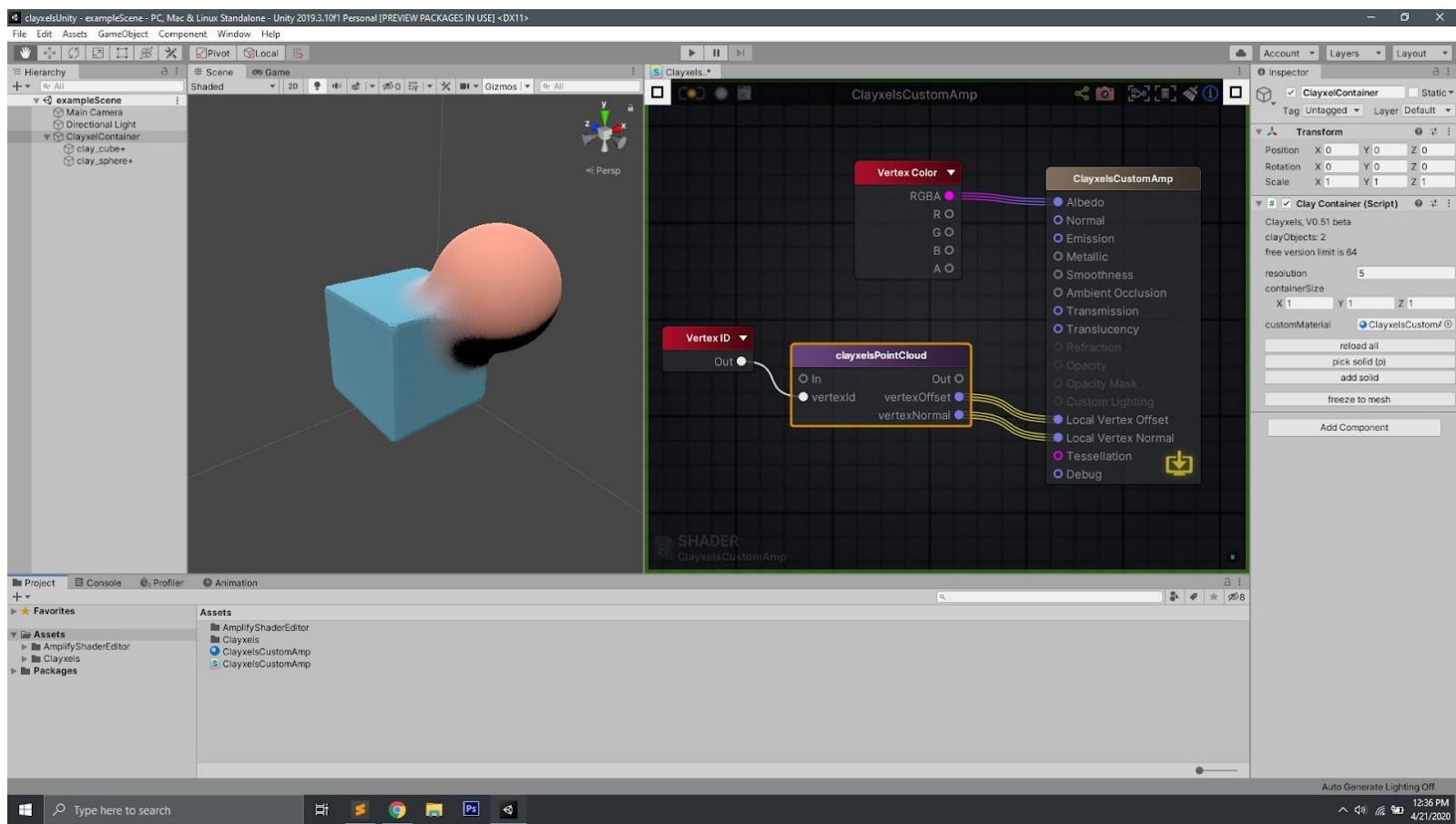
Here's an example of how to do that (code to be added in `claySDF.compute`, `computeClayDistanceFunction`):

```
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 the vertexId node. Hopefully this is something that Unity will eventually resolve in future versions.