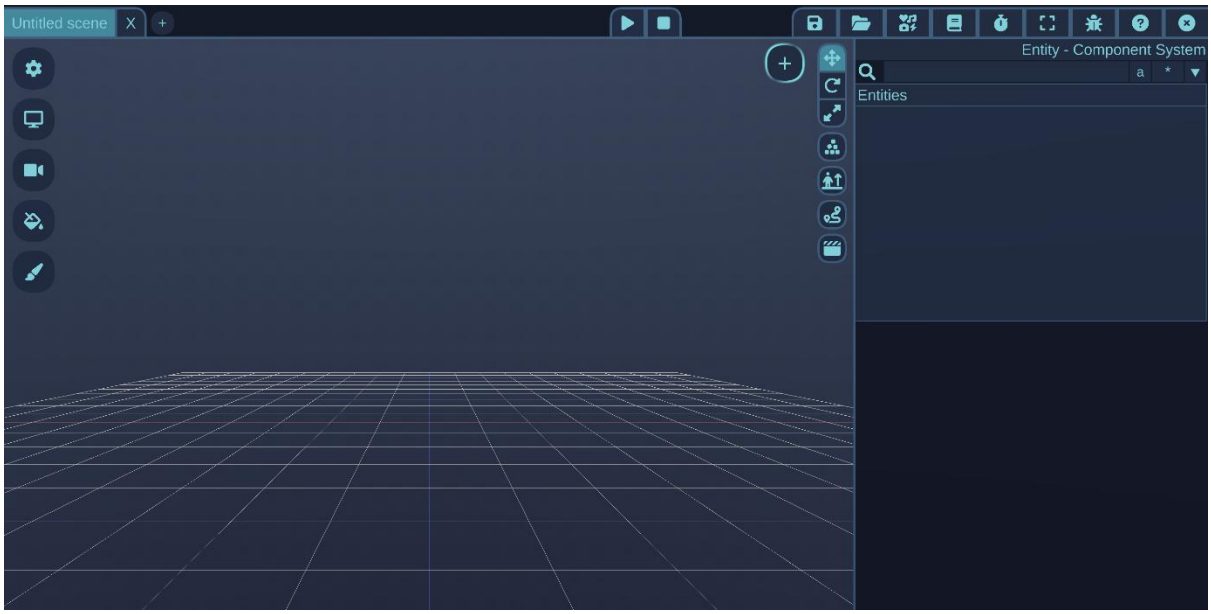
Wicked Engine Editor

Documentation

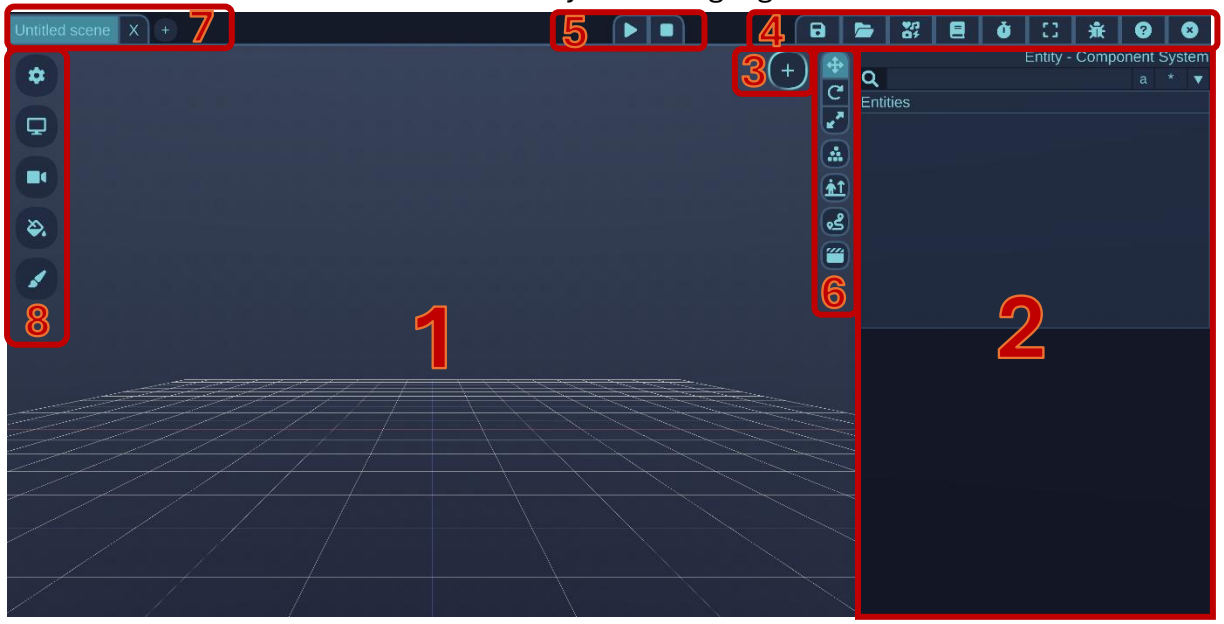
Contents

Basics

Wicked Engine comes with a simple and lightweight editor application [download], which you can use to open and edit 3D models and save them into the engine's optimized file format. You can also test and set up various engine features, like graphics, physics, audio. You can run scripts right from the editor, which lets you rapidly test functionality or even play complete games written in Lua, completely from inside the editor. The editor looks like this in idle state:



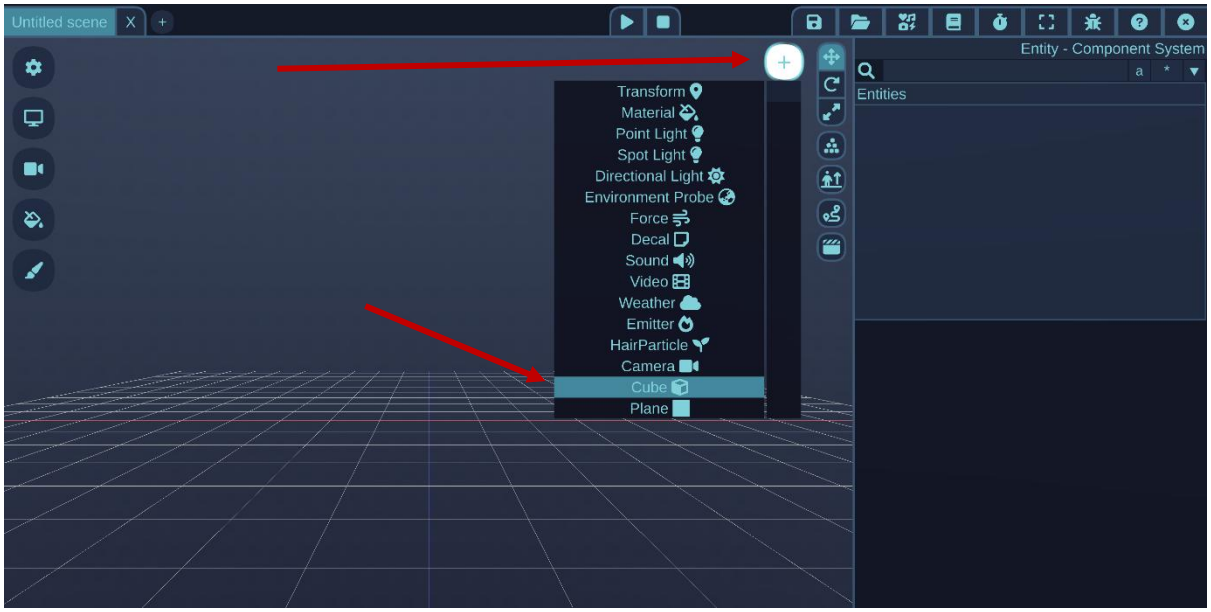
The main areas of interest of the editor layout are highlighted below:



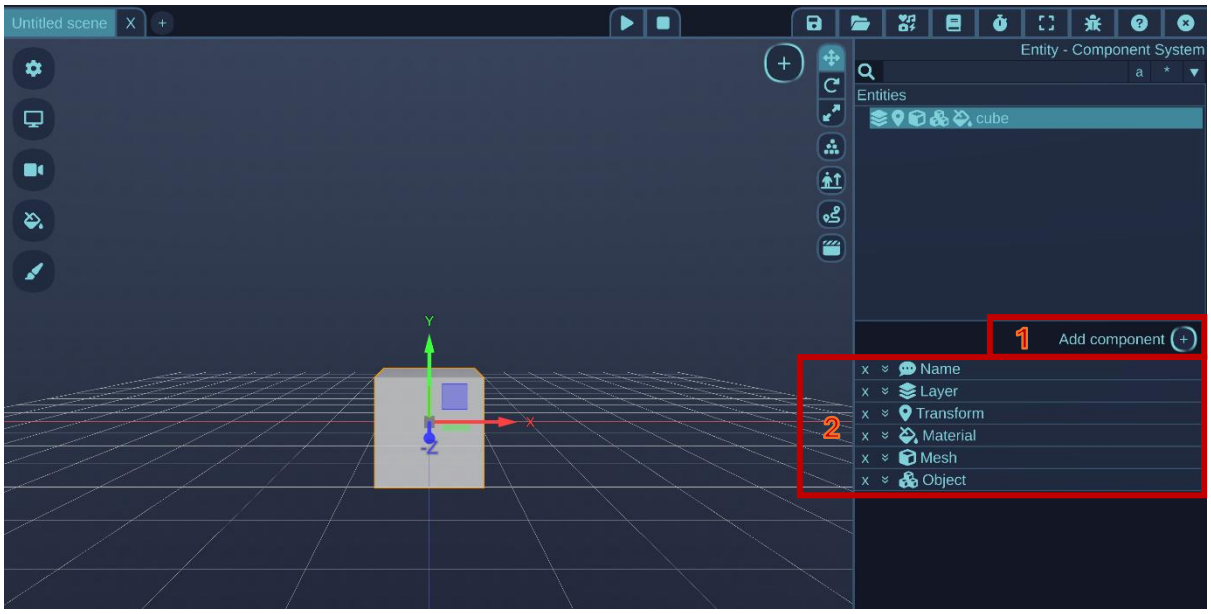
1. The main 3D work area. If you put down new entities, they will be visualized here. You can move the camera with the WASD keys by default, or the controller analog stick, or directional pad if a controller is connected to the first player slot. You can look around in first person controls by default with the mouse, while holding down the right mouse button, or with the right analog stick of the controller. The left mouse button will be used to select 3D objects which the mouse points to.
2. The entity-component system panel on the right will display a hierarchical list of entities that are in the current scene in the Entities panel. If an entity is selected, there will be more controls that appear under the Entities panel (shown later).
3. The + button in the top right corner of the 3D work area is the main button that you can use to add new entities to the scene, like objects, materials, lights, sounds, terrain and much more. By clicking on the button, a long drop-down list will be displayed where you can select the thing you want to add. You can use the mouse wheel to scroll this list, or use the scrollbar on the left, because not all of the items can be displayed at once.
4. The top menu has many buttons, each of which can do a main action of the editor. If you hover over each icon, then additional information will be displayed about each. First, the basic text of the action will be expanded just by hovering the mouse over it, then a tooltip will be also displayed if you keep the mouse hovered over it for more time. These actions include:
    a. Saving the current scene
    b. Opening a model, scene, script, or other asset
    c. Opening the content browser
    d. Opening the backlog
    e. Opening the profiler
    f. Toggling between windowed and full screen modes
    g. Going to the bug report website
    h. Displaying the help
    i. Closing the editor
5. The script control buttons, where you can run the previously used script quickly, or stop all running scripts that are running in the background as Lua processes. If you didn't use any script previously, then the "play" button will open a file browser window which you can use to open a Lua script file.
6. On the right-hand side of the 3D work area, there are several small buttons that map to common operations:
    a. Toggling between translate/rotate/scale modes, this applies to the transform tool controls when something is selected that has a Transform component.

    b. Toggling physics simulation on/off, with this you can turn off physics simulation, so dynamic objects will not fall down. It's useful to turn off physics simulation while editing a scene which has physics objects.
    c. Toggling reference dummy visualizer on/off. This can display a ghost character which you can place in the scene with holding the middle mouse button and pointing on surfaces. Useful to measure scaling of objects relative to a generic human scale, without placing additional objects into the scene. To view further info about exact controls, look at the tooltip of this button by hovering the mouse over it for a short time.
    d. Toggling navigation testing mode on/off. This can be used to test putting two waypoints into the scene and displaying path finding result between them. This requires the scene to have a voxel grid in it. To view further info about exact controls, look at the tooltip of this button by hovering the mouse over it for a short time.
    e. Entering cinema mode, where the whole interface will be disabled, and the whole window will show the 3D render result. Useful for making screenshots. To exit, press Escape.
7. Scene selector, where you can create a new scene, switch between scenes, or close a scene. All operations will only apply to the currently selected scene.
8. Additional tools, clicking on any of these buttons will open up a panel on the left of the 3D work area where you can do additional tasks:
    a. Options: all the general options will be displayed here, like choosing a language, theme, or selecting additional things to display.
    b. Graphics: all the rendering options will be listed, like shadows, post processing, and many more.
    c. Camera: options for the main camera, like movement speed, view distance, field of view, and some additional helpers like placing and jumping to secondary cameras.
    d. Material list: lists all the materials in the current scene with a preview showing their first texture (if any). Selecting materials here will also select them in the Entities panel.
    e. Paint tool: This can be used to paint various parameters in the scene, like painting into textures, vertex colors, physics, particle systems and other things.

Now let's look at what we can do when the scene is not empty, and something is selected. Click on the + button and add a new Cube:



When the cube is added, it also became selected, and you can see the additional controls for the selected entity in the Entity-component system panel:

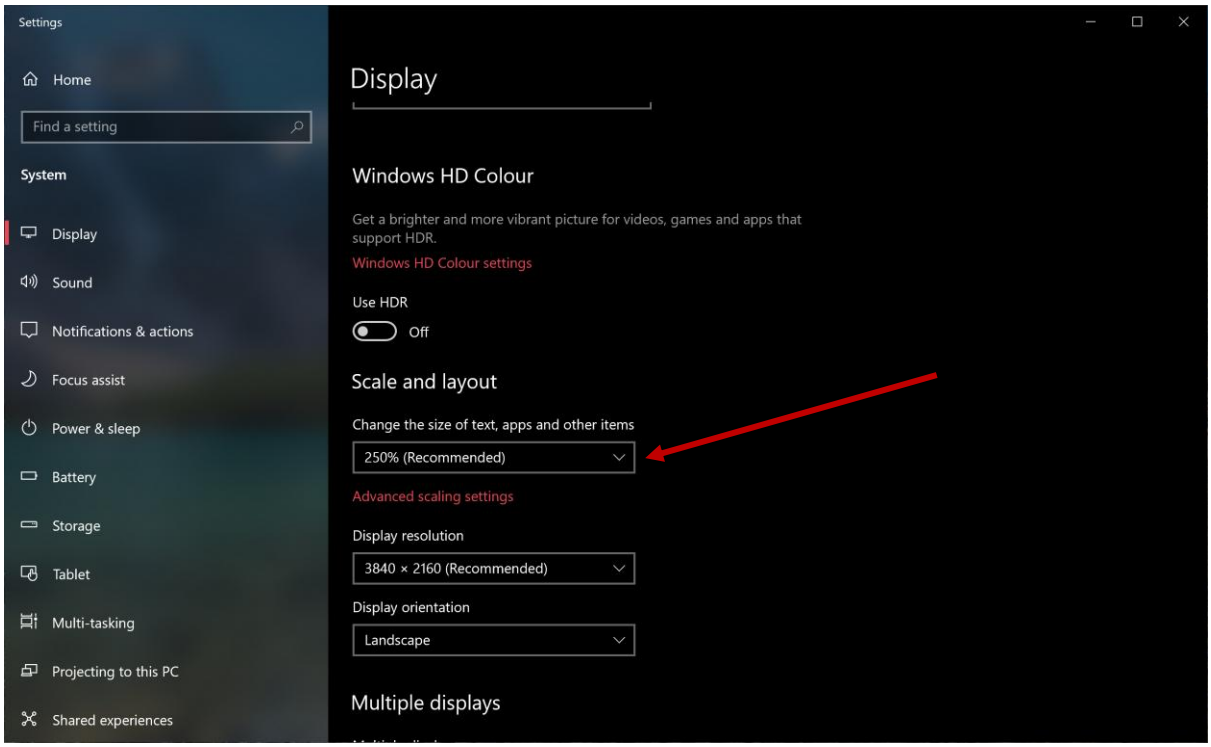


1. The “Add component” button is used to add a component to the selected entity. A component type adds a certain functionality to the entity, like a name, material (textures, colors, surface...), mesh (geometry), layer (filtering) and many more different component types exist.
2. The list of components that are attached to the selected entity. Each component type can be opened by pressing the downwards arrow near its name. The component can also be removed from the entity by pressing the X button.

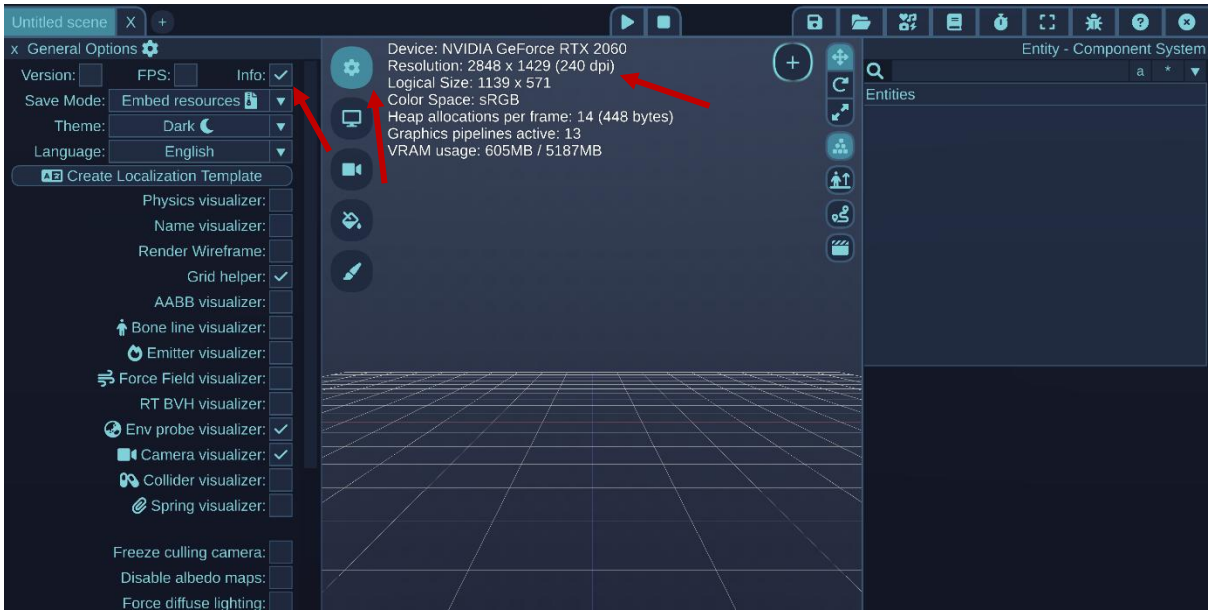
Additionally in the 3D workspace, the selected cube is also highlighted by an orange outline and the transform tool is visible. The transform tool is used to place the selected object in the 3D space.

Display scaling

The editor's interface will follow the display scaling that is specified in the operating system settings in Windows:

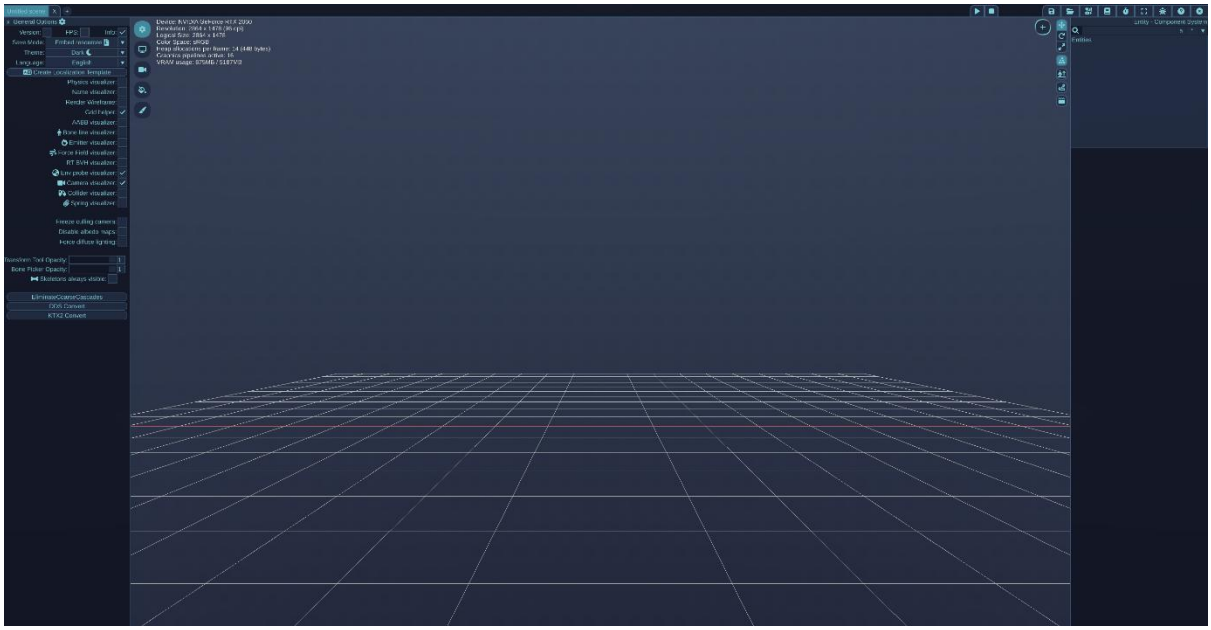


This can be verified if you turn on the Info display in the General options:



The dpi value corresponds to the scaling setting in windows, 96 dpi is the default that corresponds to 100% scaling. On high resolution displays like 4K, the scaling is often increased to make the on-screen elements larger to help readability.

The Editor with reduced scaling on the same monitor can become less readable:



The scaling can be overridden by a specific value if you enter the following lines into the startup.lua files near the exe, which will set the scaling to be 1.5x (= 150%):

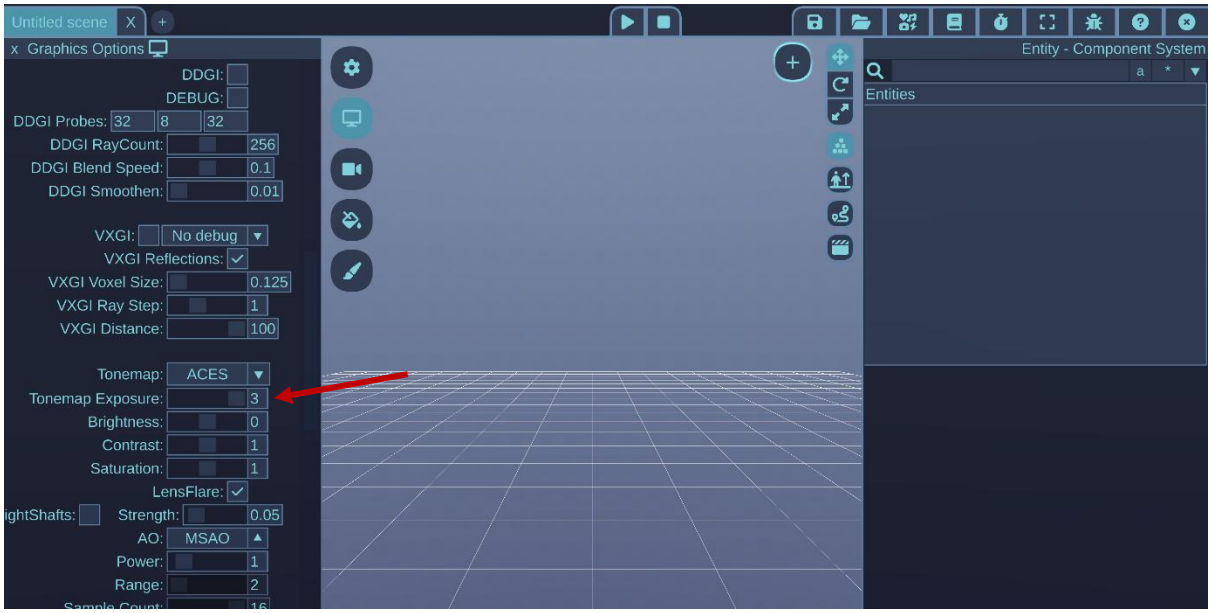
```
-- You can set custom scaling for 2D screen elements here (GUI, sprites, fonts):
local canvas = application.GetCanvas()
canvas.SetCustomScaling(1.5)
application.SetCanvas(canvas)
```

Apart from this, you can execute arbitrary lua scripts too in this file, which will be executed once at startup. For more lua commands, check out the LUA scripting documentation in Content/Documentation.

HDR display output

Wicked Engine supports HDR display output, which is enabled by default but only takes effect when a HDR monitor is used. This can be checked in the Graphics options -> Display output combo box. By default, it will be SDR 10bit if the display is non-HDR, or HDR 10bit when the monitor is HDR. Here you can also force it to be SDR on a HDR display if needed.

With HDR enabled, taking screenshots can produce somewhat unexpected results, or the 3D render area can become overly dark. It is expected that when using HDR display, the camera exposure will need some tweaking to achieve a nice result. In a game, this could be put onto a HDR calibration option screen. In the editor, you can tweak the exposure in the graphics options, which will affect the 3D render result, and not the GUI:

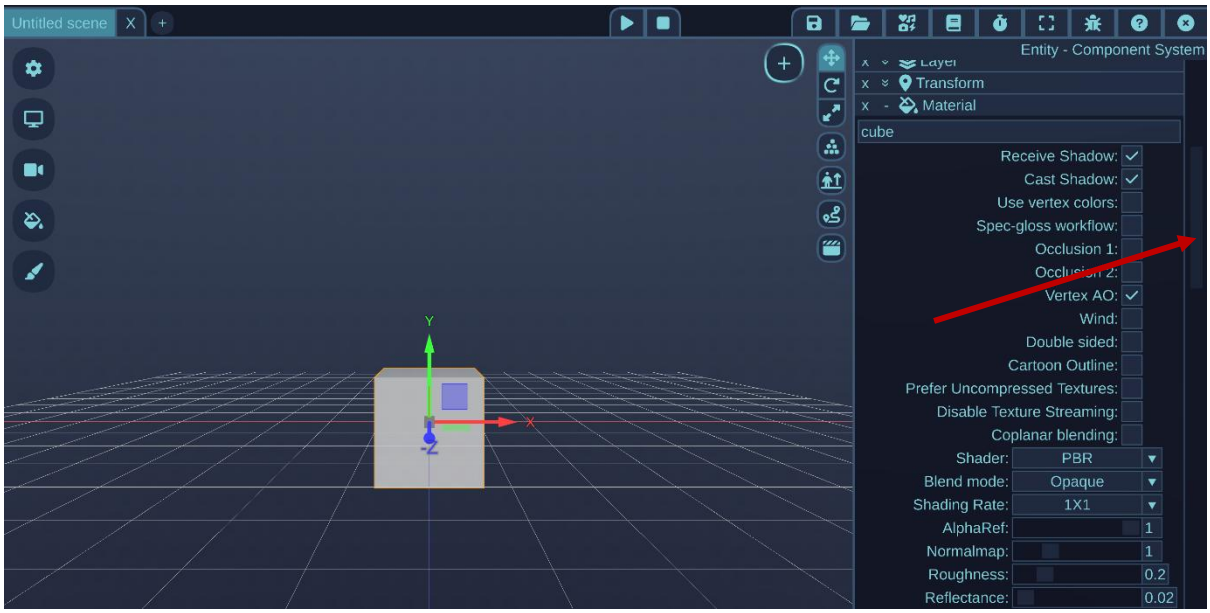


Editing

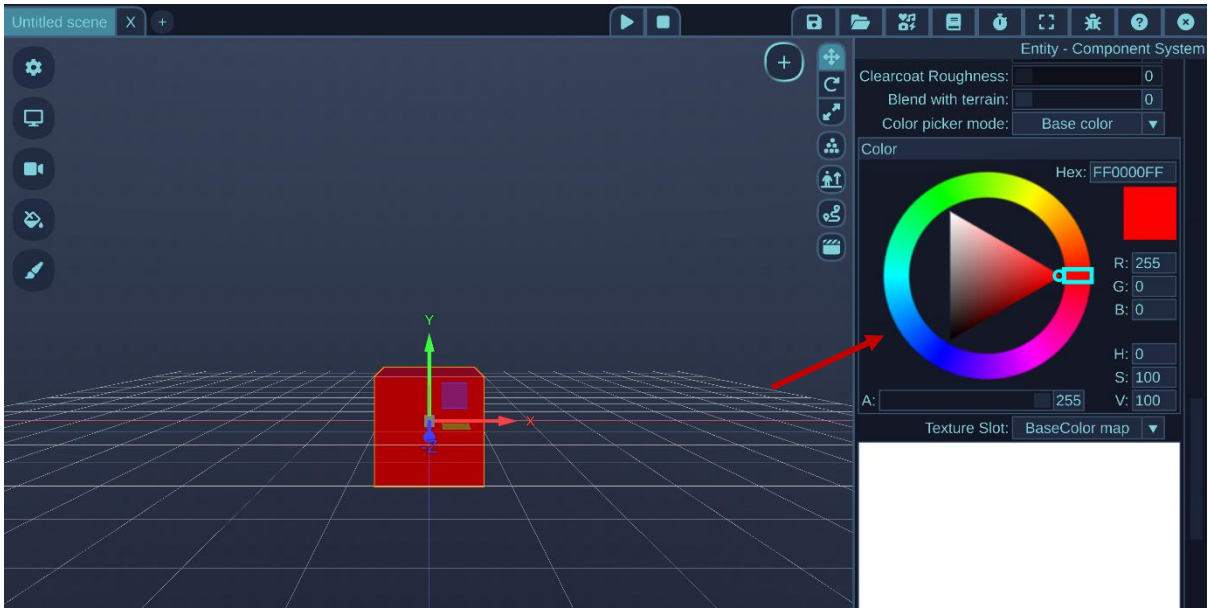
Setting material color

As the first exercise, let's set the cube's color to red. Open the Material settings by clicking on the downward arrow near the Material icon.

When the material panel is opened, scroll down until you see the color picker:

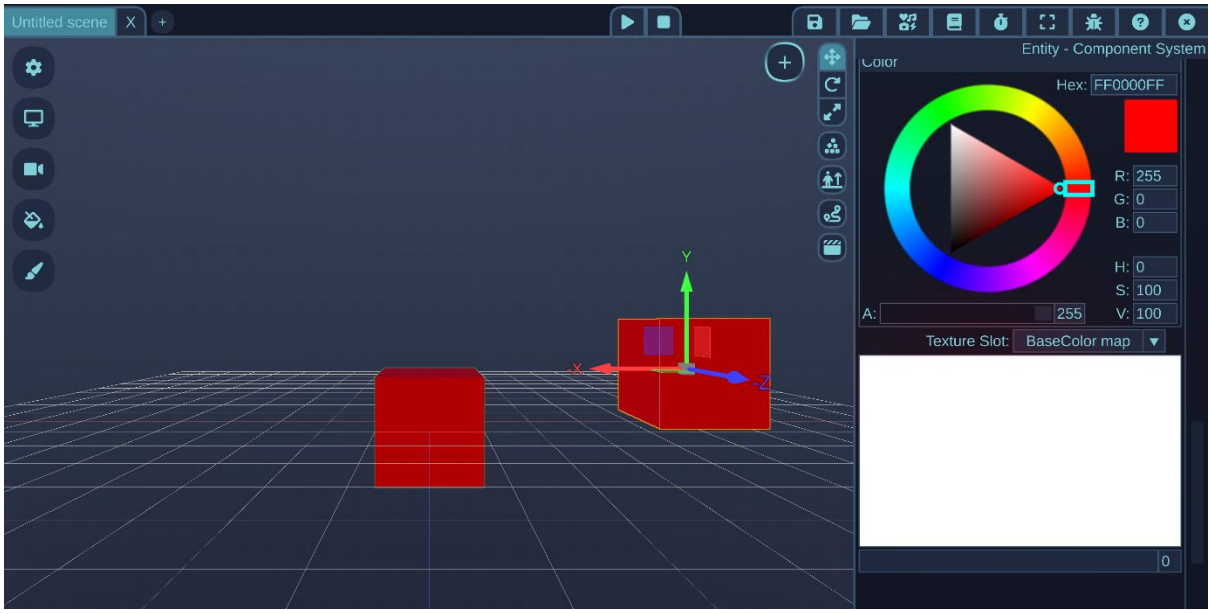


Then the color picker can be used to select the color of the selected object's selected material:

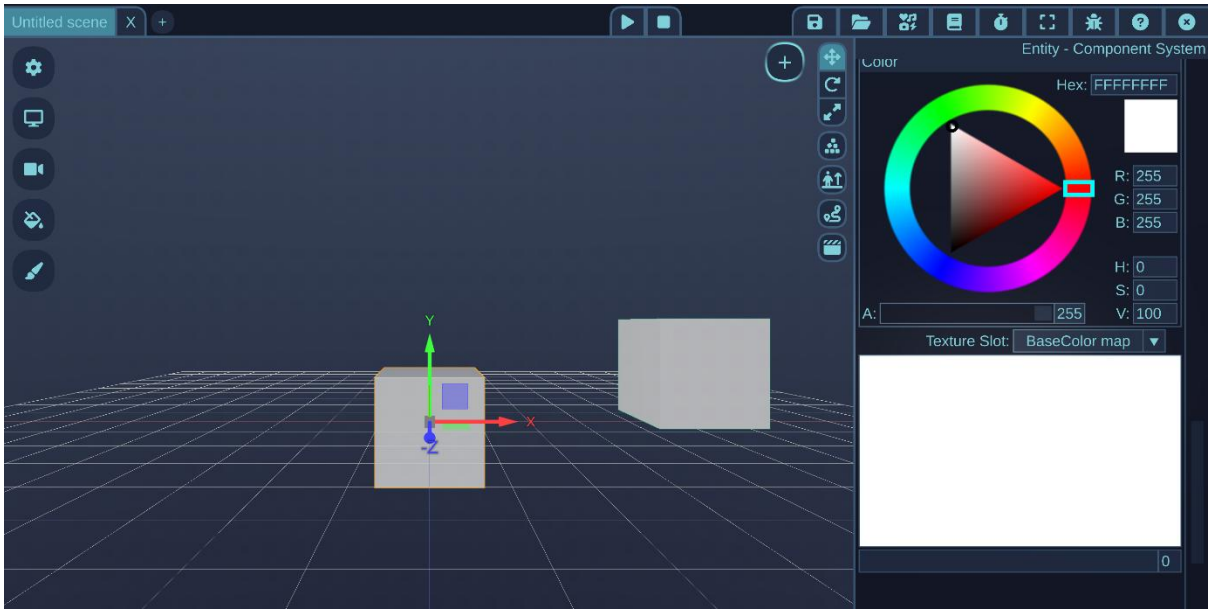


1. Duplicating object

Press Ctrl + D to duplicate the selected object, then grab it by the transform tool to move it away a bit. You can also use Ctrl + C and Ctrl + V to copy and paste the object, which does the same as Ctrl + D.

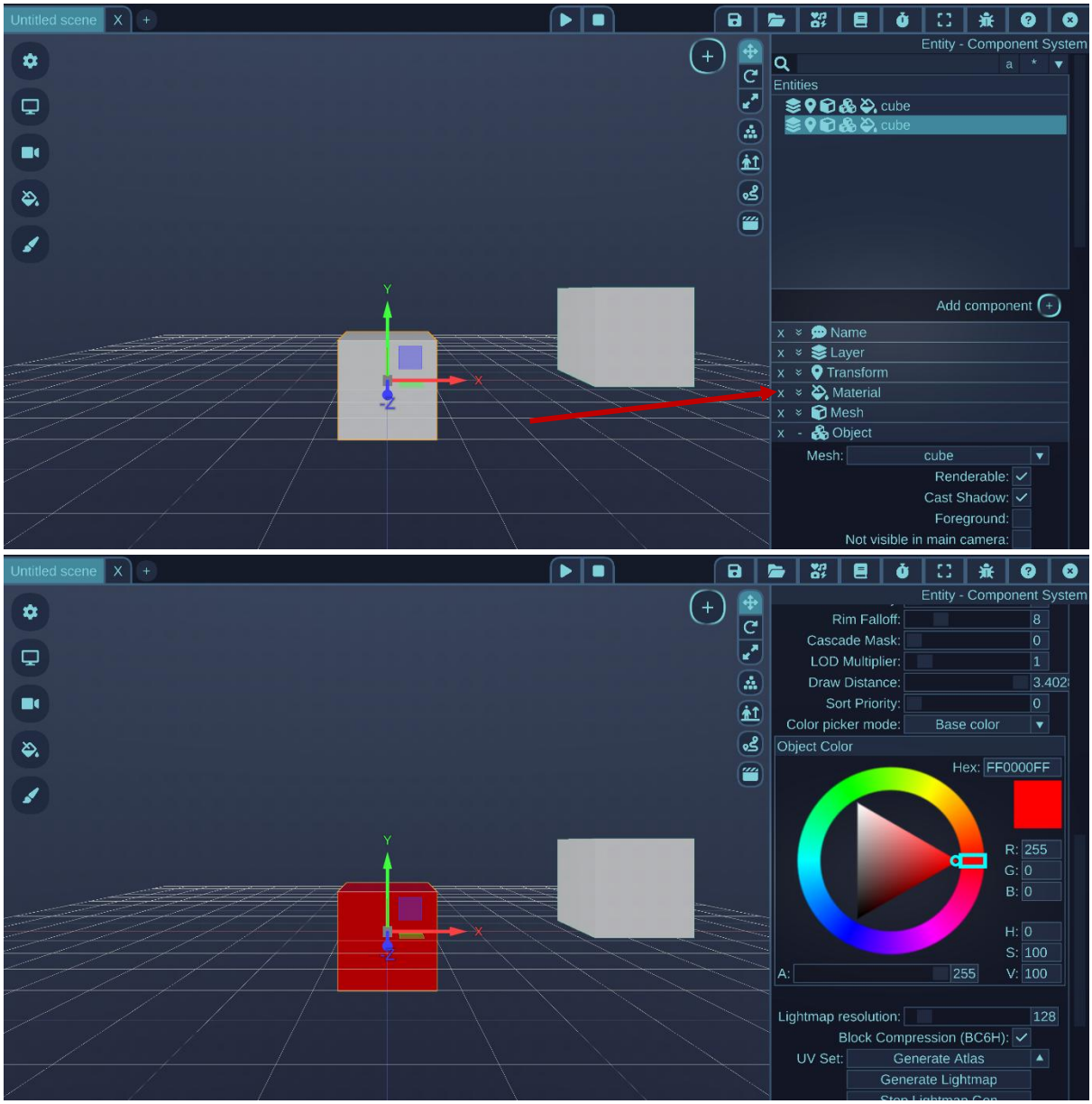


You will notice that both objects are red, and if you modify the material color, both of them will use the same color, let's set them to back to white now with the color picker in the Material settings:



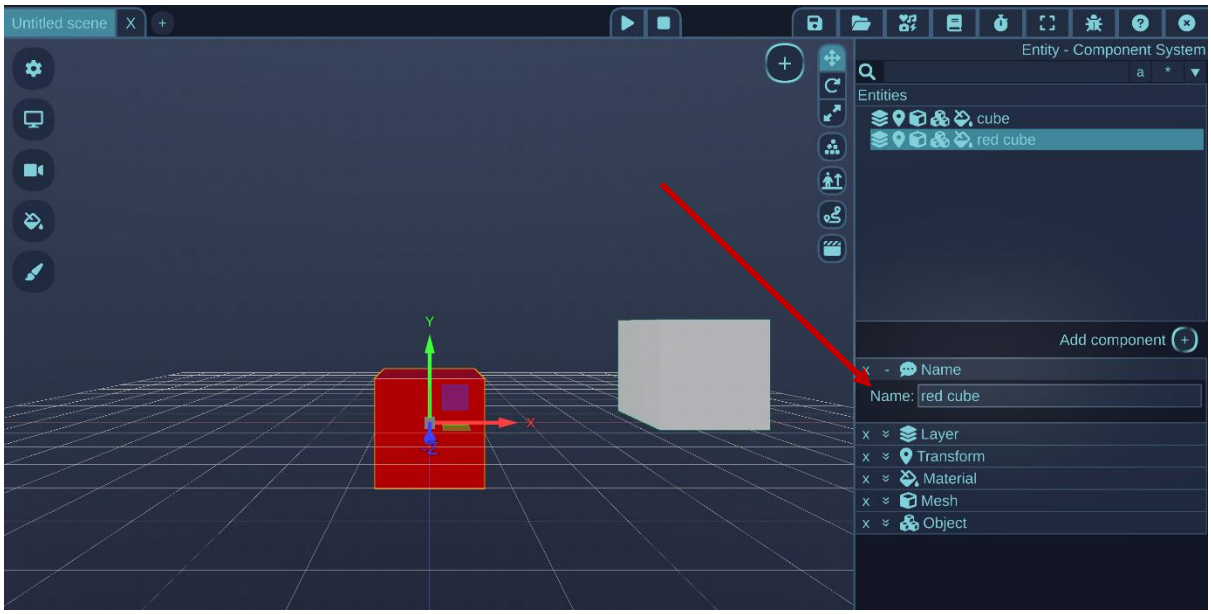
This happens because both objects are using the same mesh and through that mesh the same material, so they are instanced. Instancing is useful, because it helps the renderer to batch them all into one draw command, which is more optimal. However, instancing only works if they both use the same mesh and the same material. But you can also control some per-instance properties of objects, like the color, which you can

control from the Object settings. We can set one of the objects to red, while keeping the color of the other one by setting the color in the Object settings instead of Material settings:

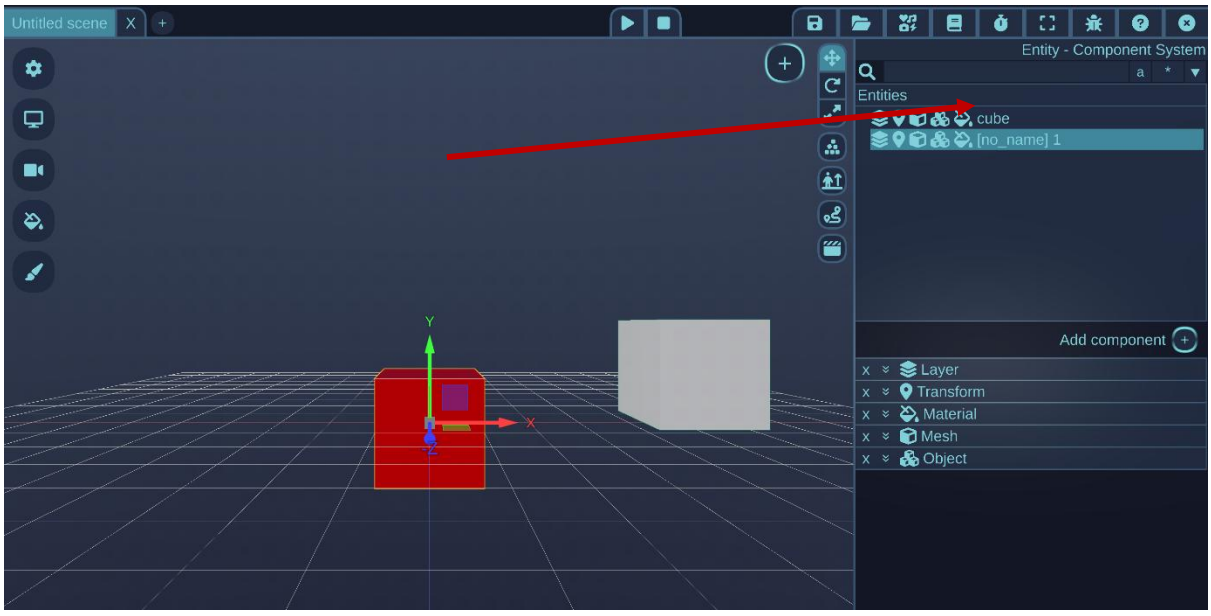


Renaming

To set a name for the cube, you can go to the Name component settings and enter a different name:



It's not necessary to have any of the components on an Entity, so it's also valid to delete the name component. In this case, the red cube will be an Entity without a name, but other components. The editor will indicate this as [no name] followed by the Entity ID in the Entities panel:

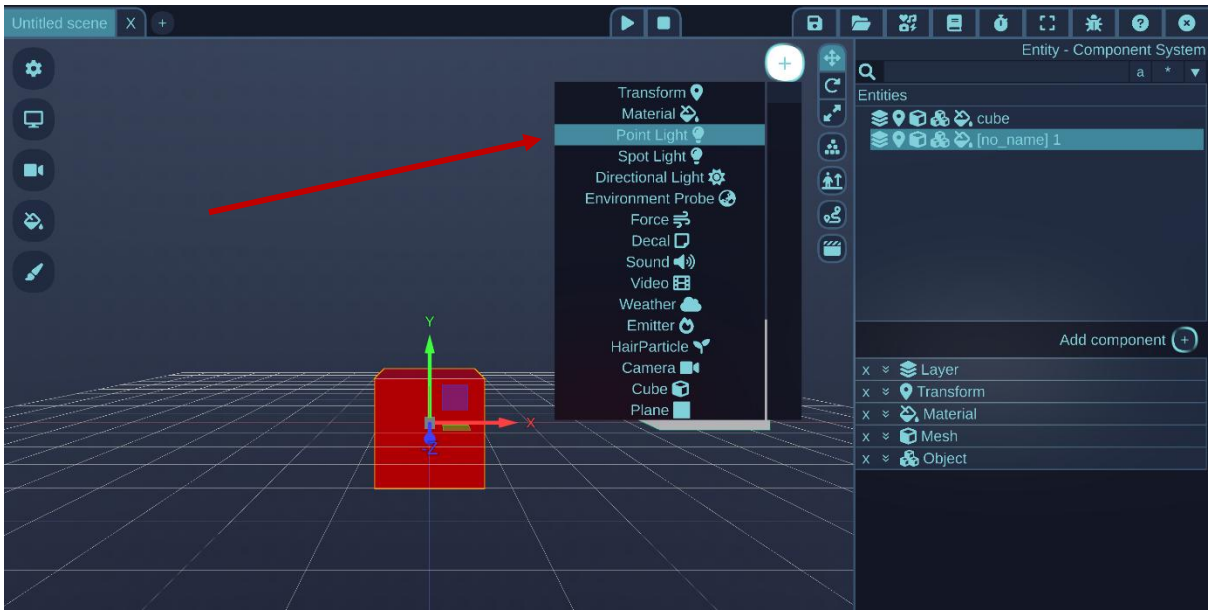


If you hover the mouse over an object in the 3D work area while holding down the "i" button, then the name and Entity ID will be displayed for them on the mouse position. Having a name for an object is useful if you want to find objects by name in a script, otherwise the engine doesn't care. Existing Entity IDs are guaranteed to stay valid for the

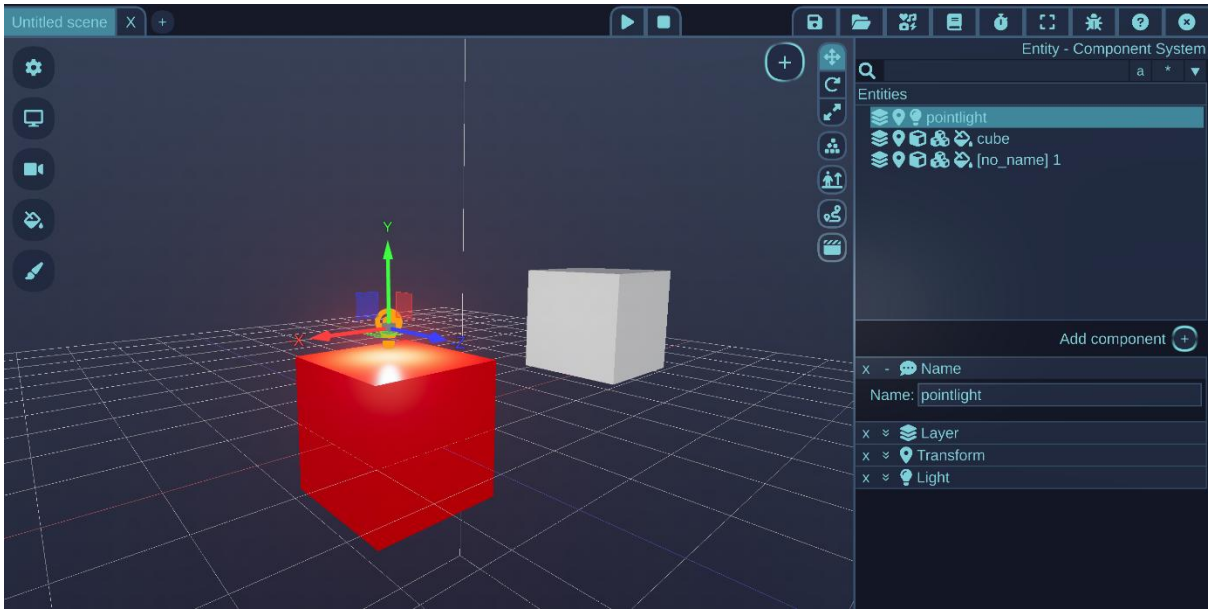
duration of the application, but not for an other application run, so they can’t be relied upon in all circumstances.

Adding lighting

Add a point light by clicking on the + button:

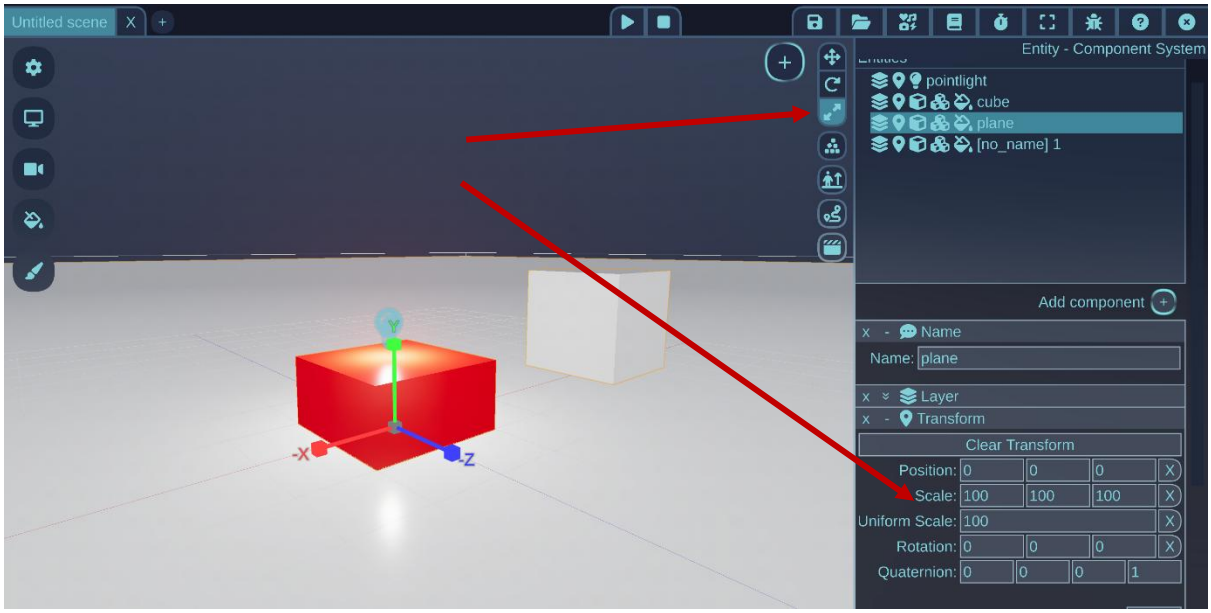


The point light is added in front of the camera and selected, move the camera and light so it positioned somewhere above the red cube:

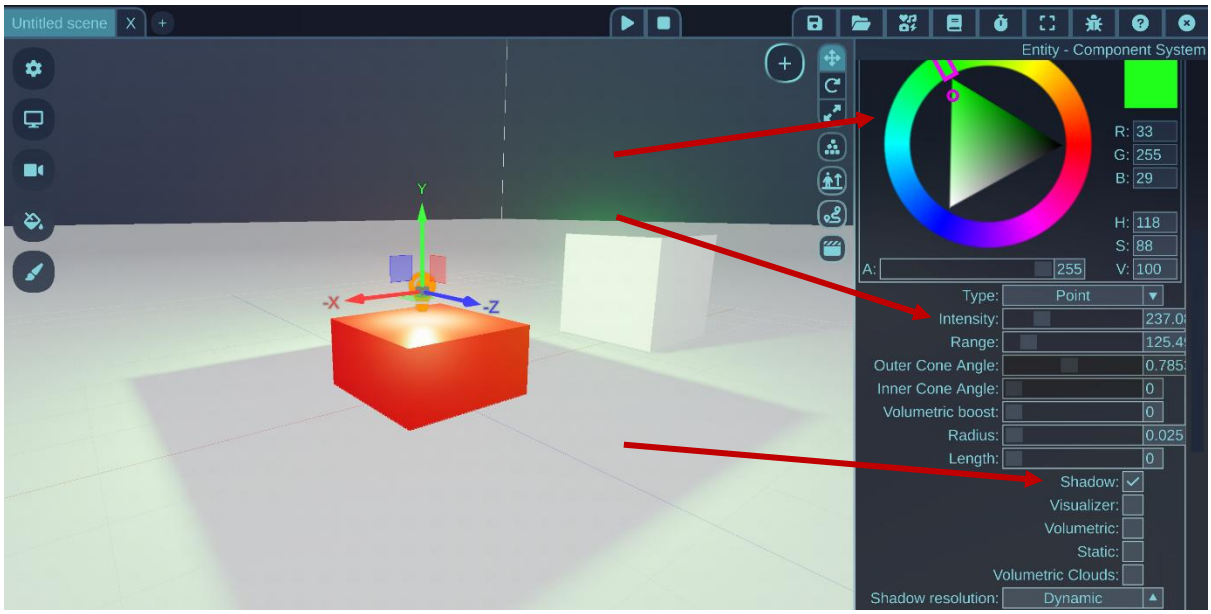


The default lighting in the editor is bright, so it might be a bit difficult to see the effect of the light at first, but if you put it close to the object, it will be well-visible. Also put down a new plane object (it might be obstructed by the red cube if it still sits in the origin, but that's not a problem) and scale it up so it acts as the ground of the scene. To scale it, you can switch to the scaling mode by pressing the scaling icon and using the mouse with the transform tool, or open up the Transform settings and enter values. I decided to

uniformly scale it to 100 in all axes:

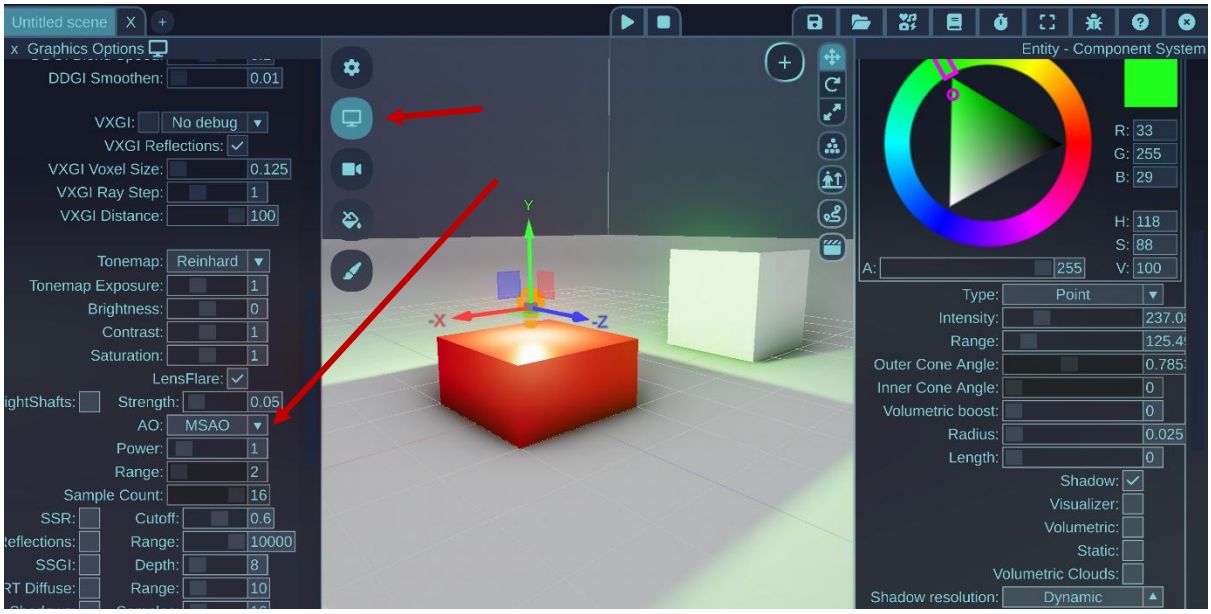


Now try to adjust the light parameters like color, intensity, range and set it to cast shadow:

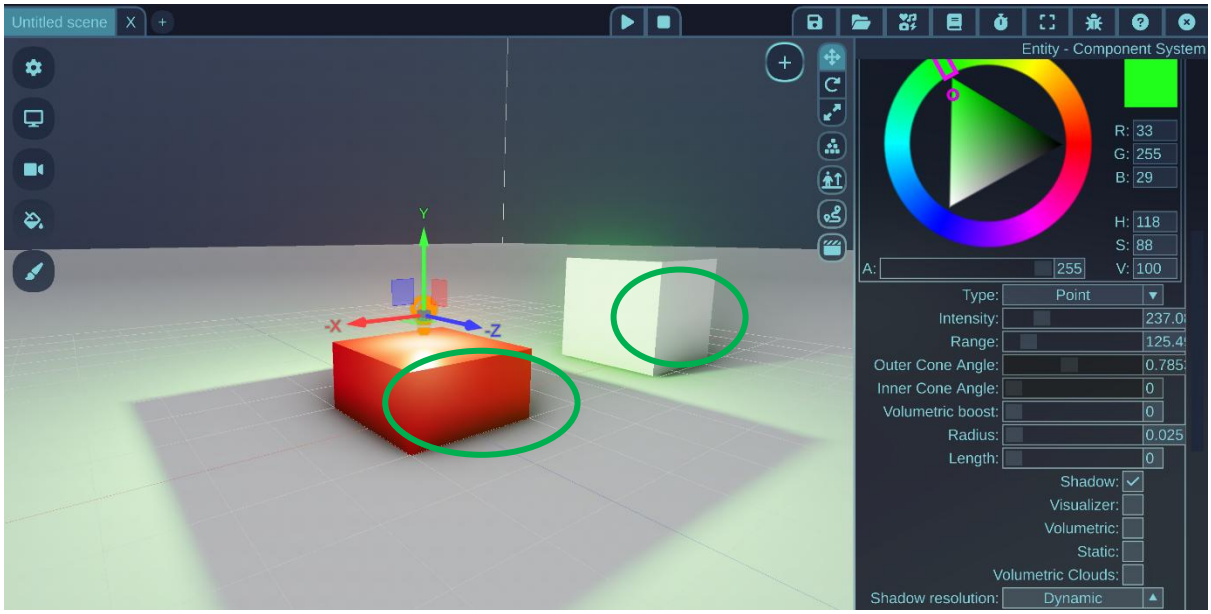


Now you see that it casts shadow on the ground from the cube below it, though it looks a bit bland, and not much detail in the shadowed areas. A post process ambient occlusion can be easily enabled in the graphics options to improve the look a bit. I recommend to use the “MSAO” option by default if you choose an Ambient Occlusion

method, as it has a pretty good performance/quality tradeoff:



The MSAO effect will act on the whole 3D scene to add contact detail inside shadowed areas, improving the depth perception. You can see it in shadowed corners or geometries that are close by or intersecting.



There are several other ambient occlusion and other graphics techniques that can change the look of the scene. When first using the Editor, the default settings have most of the additional effects like this turned off but modifying them will be remembered for the next run of the editor application. The settings are saved into the config.ini file near the executable.

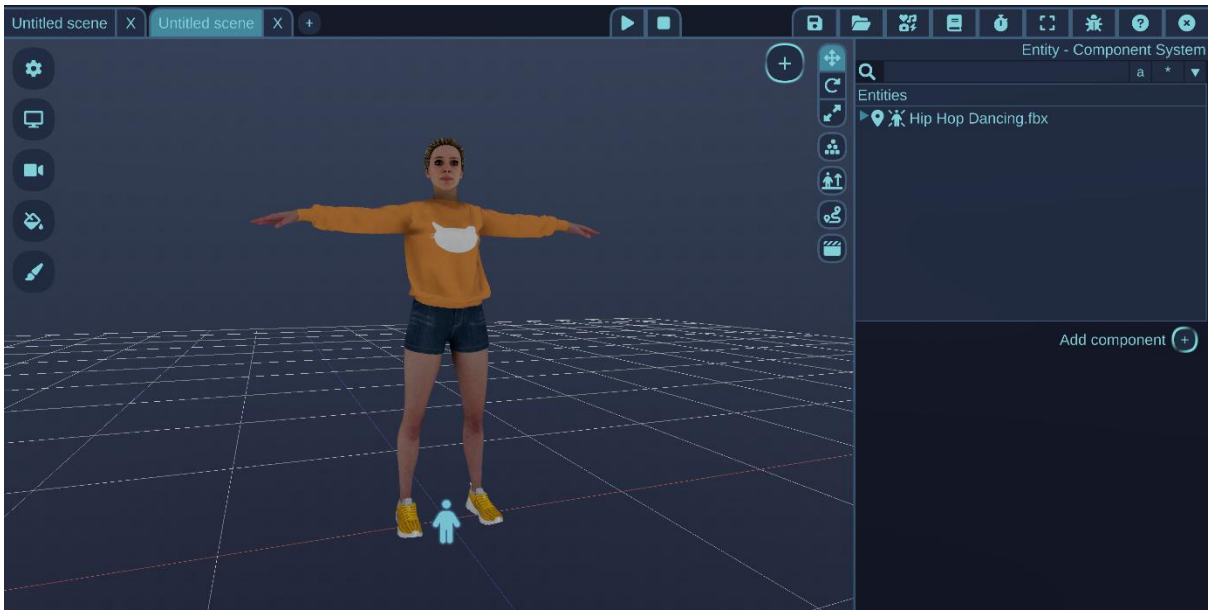
Now you can save the model as a wiscene by pressing Ctrl + S.

Importing a model

You can import common model formats into the editor, the supported formats are these:

- GLTF 2.0: GTLF or GLB extensions
- FBX
- OBJ
- VRM

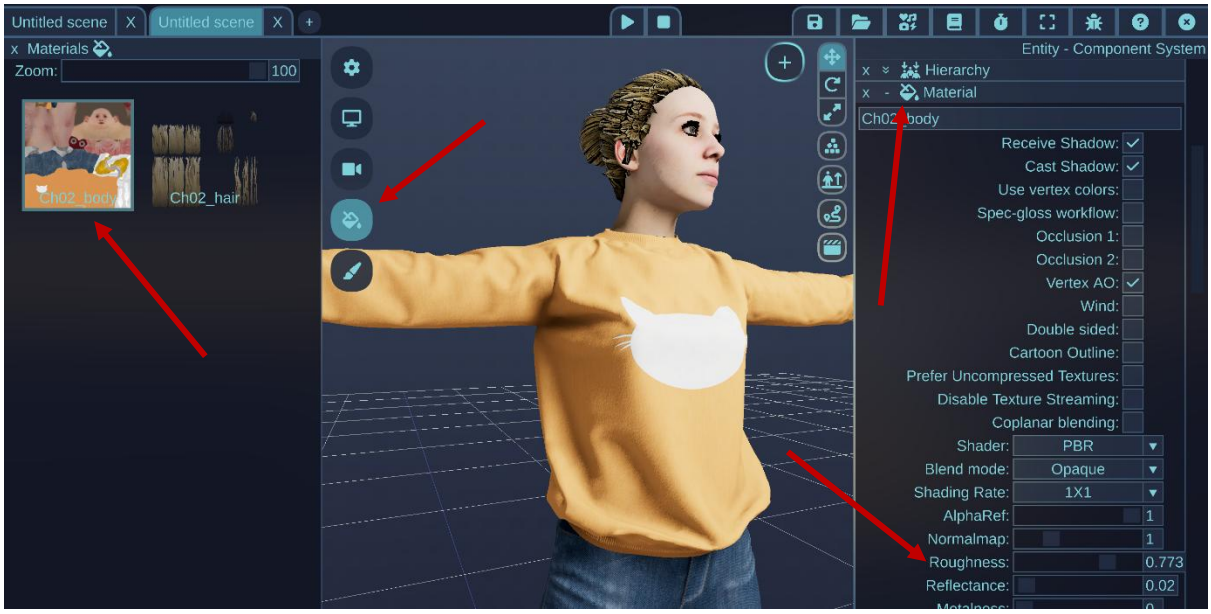
To import a model, the easiest way is to drag and drop the model or multiple models into the editor's window (only supported on Windows right now). Other than that, the Open button in the top menu will work too, it will let you browse in your files and open a single model. You can also specify which scene to open a model into with the scene selector on the top toolbar's left side. You can download a rigged FBX model with animation easily from Mixamo.com, in the example I will use one from that site and import it into a new scene, then moved the camera closer to it:



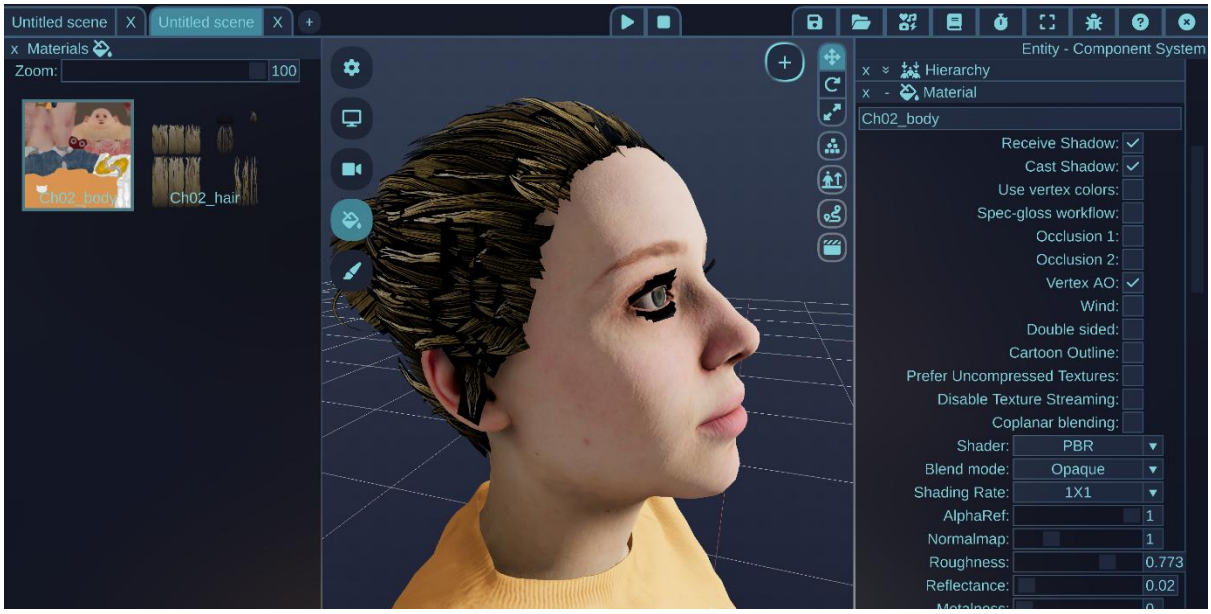
The FBX model can have a complex hierarchy, with not just meshes, but also skeletons and animations. Because of complications, some materials can be weird looking when imported, because they can follow other material authoring conventions. Usually the materials will work best when importing GLTF and VRM models. For this FBX model I will make some quick fixup modifications.

The first problem is visible when I add a light, the materials of the face and sweater appear too shiny:

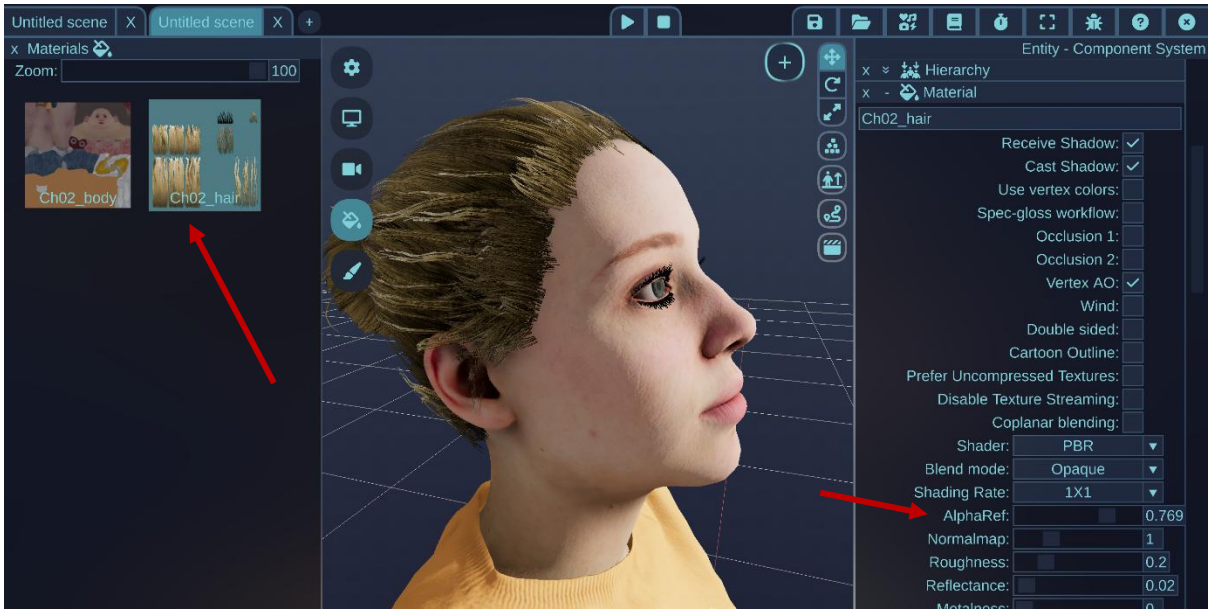
Select the materials from the material browser window on the left, or by clicking on them on the 3D object, and increase the roughness parameter to make the shininess more matte:



Another issue is the hair cards which have transparency in their texture, but the FBX material didn't supply proper parameters on it:

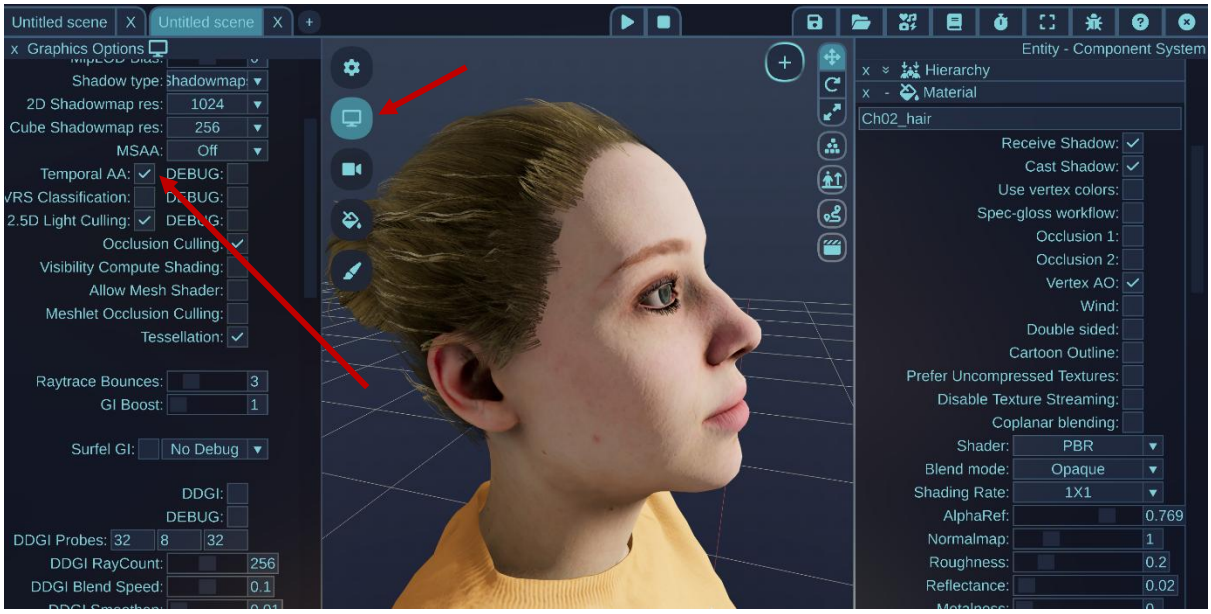


Instead of using real transparency on the hair material, it is more efficient to use alpha testing to retain correct render order in all viewing angles and more optimal rendering. Decrease the alphaRef value for the hair material to get this:

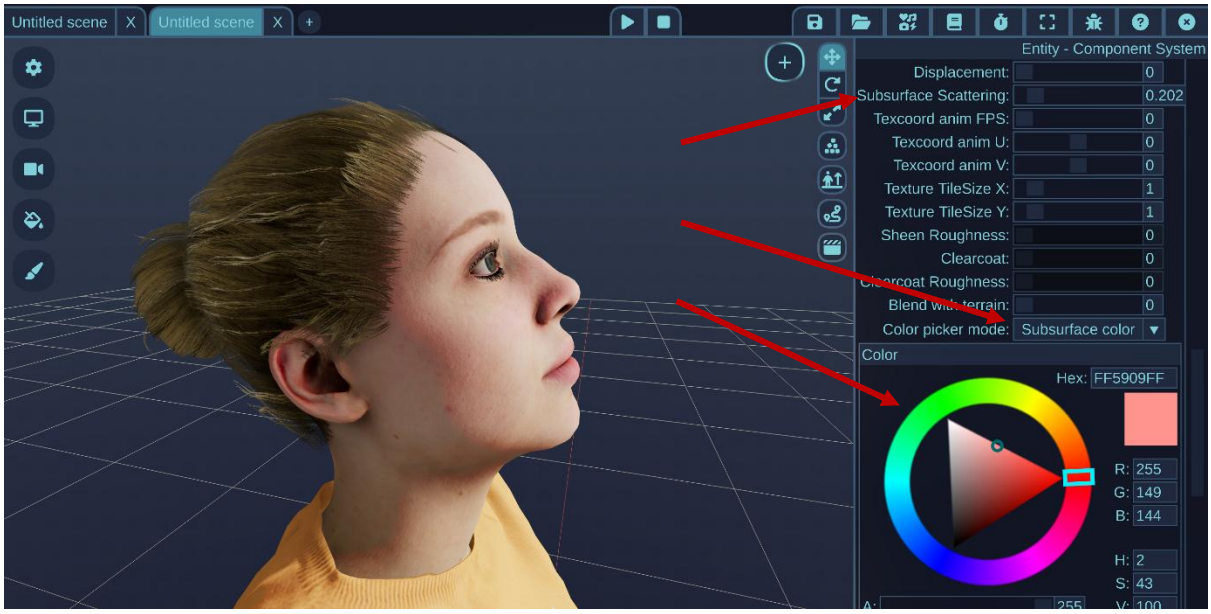


AlphaRef, also called alpha testing keeps the material opaque, so properly working with other depth buffer effects, while simply discarding pixels which have alpha below the alphaRef value. To improve the look of alpha tested materials, optionally we can turn on temporal anti aliasing in the graphics options (“Temporal AA”). Temporal AA can have some disadvantages, but apart from anti aliasing, it can also soften the alpha tested

textures a lot:



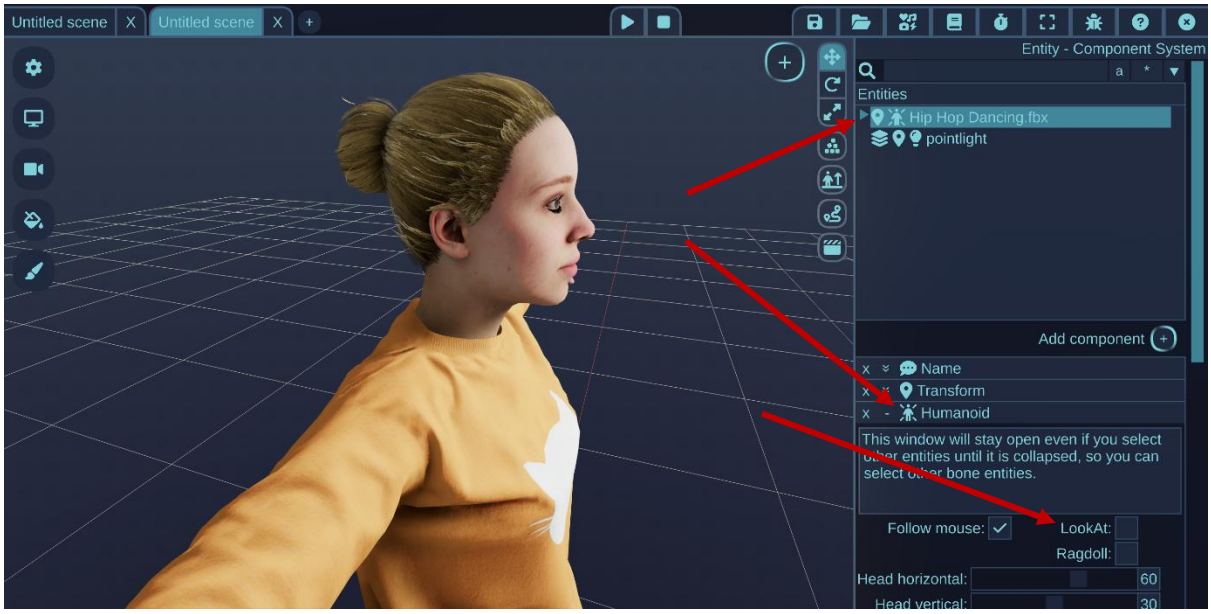
One other thing I like to set for skin materials is the subsurface scattering, to make it a bit softer in various lighting conditions. Even though the whole character is using the same material as the skin (except for the hair), you can try this to see the effect. Increase the subsurface scattering material value and set a reddish subsurface color by changing the color picker mode to “Subsurface Scattering”:



You can see the area between shadow and light on the skin gets smoothed out and receives a reddish tint.

With the Mixamo and VRM models, you can notice that by default the character was always trying to look towards your mouse, this is because they have been imported with a humanoid rig:

The humanoid rig has some useful properties, but for now just see how you can turn off this behaviour if you want. Open the humanoid component of the whole model entity and turn off “LookAt”:



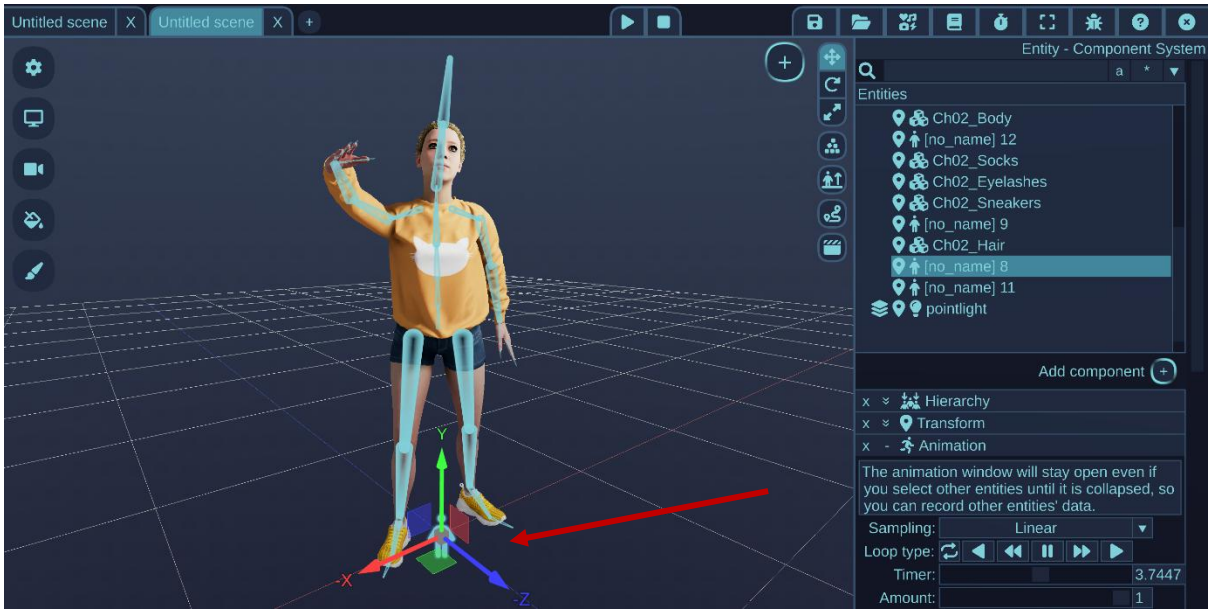
Disabling “LookAt” will make the character to face into the default direction, while turning off “Follow mouse” will keep it looking in the current direction, but no longer follow the mouse movement.

This model also has an animation (on Mixamo.com you can easily select which animation should the downloaded model have), which you can simply play by finding the entity which has an animation component and pressing the play button inside the Animation settings.

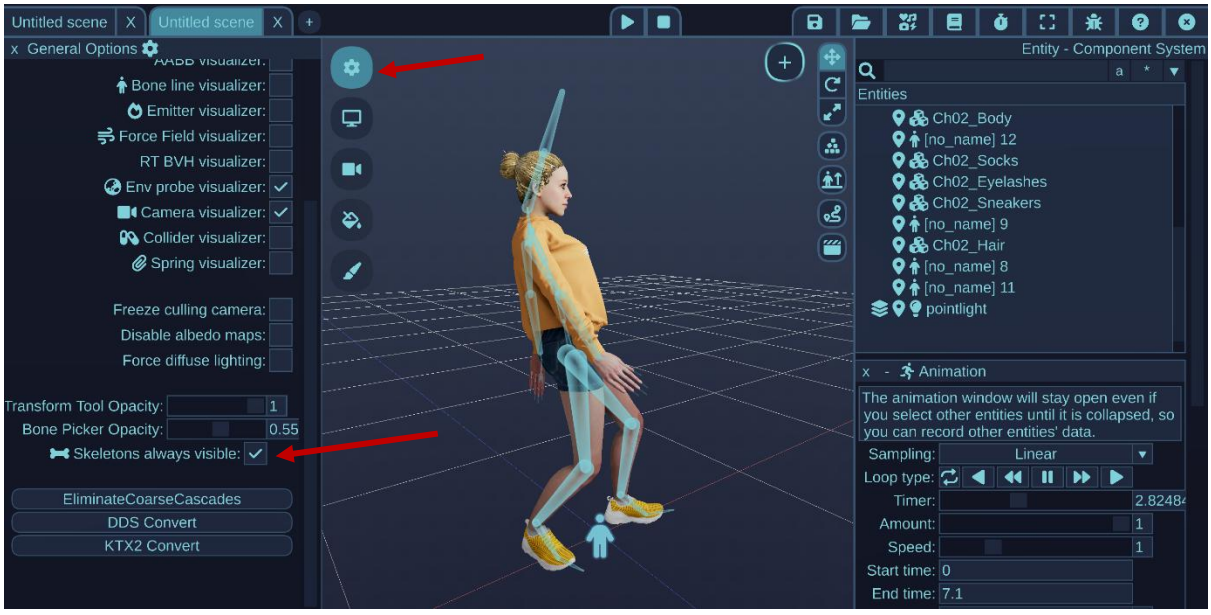
To find the animation entity, you can open the hierarchy of the model file and find the entity with a running man icon:

Well, in case there are two and both are unnamed, but I happen to know that the second one is the animation I applied on Mixamo.com, so I select that, and open the Animation component settings, then press the forward Play button in there. The animation starts playing immediately:

Because this is a model with a skeleton (Armature), you can also see a little human icon under the character in the 3D work area. If you click on it, you can see all the bones that make up the skeleton:



All the bones are entities that you can also select and modify in the editor by clicking on them when they are visible like this, or they can also be selected in the Entity hierarchy panel. Because the skeleton has to be selected for the bones to be visible, but when you select a bone, the skeleton is not necessarily still selected, the Editor will keep the skeleton visible while any of its bones are selected too. Additionally, you can make the Editor always render all skeletons in the scene whether anything is selected or not in the General Options:



It is also possible to adjust the opacity of the bone visualizer here.

Now you can save the model as a wiscene by pressing Ctrl + S.

Content browser

The content browser is a nice way to view the provided sample content as well as your previously used content from the Editor. To open it, select its entry from the top menu toolbar. Pressing it will immediately open a large window in the middle:

The Models and Scripts folder are always displayed from the basic Content folder, while the Recently used folder is not a real folder, rather it will list your most recently used models and scripts. Recently used folders will be also displayed below on the left side if there were any. Preview icons will be displayed for wiscene models (which are saved from the editor), and some specific scripts that have thumbnail images in their folder. For FBX, and other model formats there will be no preview image. By clicking on any of these once, they will be opened. Try to run the character_controller.lua script for example to run the character mini-game sample within the editor. You can exit this script with the Escape key to return to the editor.

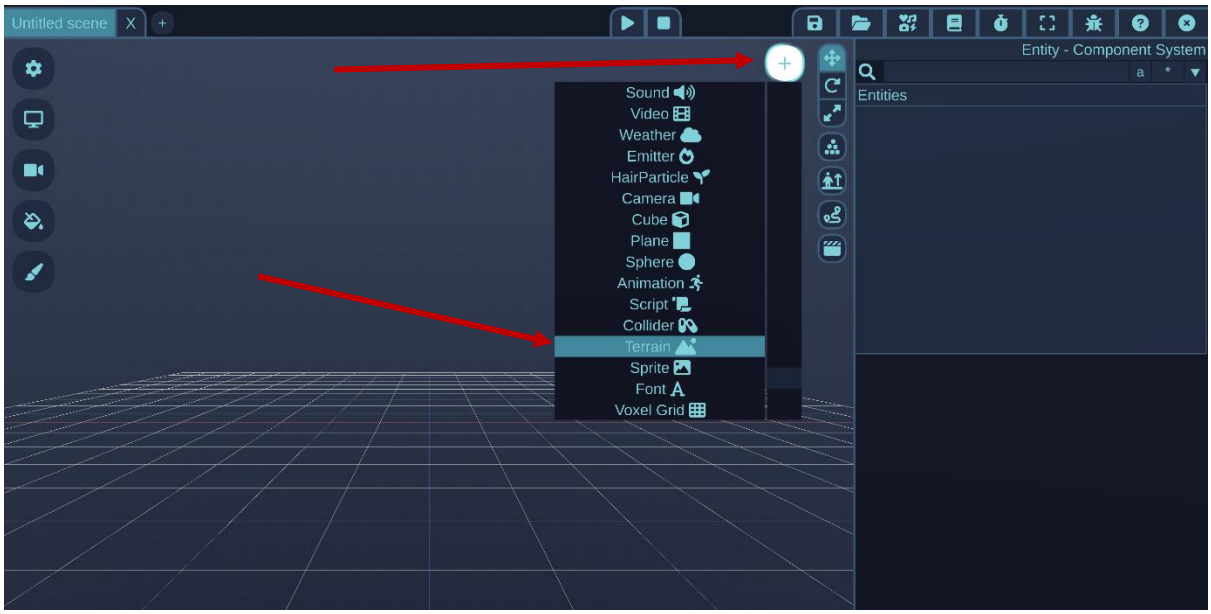
Backlog

The backlog is a useful tool to see text information that is coming from the engine, like basic information or warnings, errors. To open it, press the Home button, or select its icon from the top menu toolbar:

You can also enter lua commands here. For more information, check out the Scripting API documentation in the Content/Documentation folder or the example scripts in Content/scripts of what you can do in Lua.

Terrain

You can easily create a new terrain in the Editor, by adding a new terrain from the + button:



Once you click on it, a terrain will be generated:

If your terrain is mostly white, doesn't have trees, grass and other models on it, it means that the Content/terrain folder is not found, or is not containing these assets. This is valid, but as convenience, the Editor comes with a couple simple assets to show a relatively nice-looking terrain by default. As with other newly created entities, the terrain entity gets selected by default, and you can access the Terrain component's settings from the Entity-component system panel on the right. You can also open the terrain's hierarchy in the Entity list to see what it consists of. A simple thing that you can try is to set a new terrain preset from the Terrain settings, and you'll get a new kind of look, for

example the Islands preset:

There are a lot of parameters for terrain creation, most of them will need to be tried out. Tooltips can be found for every setting if you hover the mouse over it for a short time. To make some hand-edits to the terrain, we can also use the paint tool's various features. Open the paint tool on the left and select the "Terrain material" mode:

This lets you select from a number of predefined material assets, and use the mouse to brush paint on the terrain surface:

You can adjust the brush parameters with the sliders, but you can also adjust brush size by scrolling the mouse wheel. You can easily paint complete materials on the terrain with this. But other things can also be painted, for example sculpting to create hills can be accessed with the Sculpt modes:

The terrain is generated as a height map, but with sculpting you are no longer limited to a heightmap, it is also possible to create simple overhangs for example. In sculpting mode, you can also limit affection to terrain meshes only with the “terrain only” checkbox.

You can also paint grass coverage onto the terrain with the HairParticle painting modes. For example, Hairparticle – Add Triangle mode will add terrain triangles into the grass

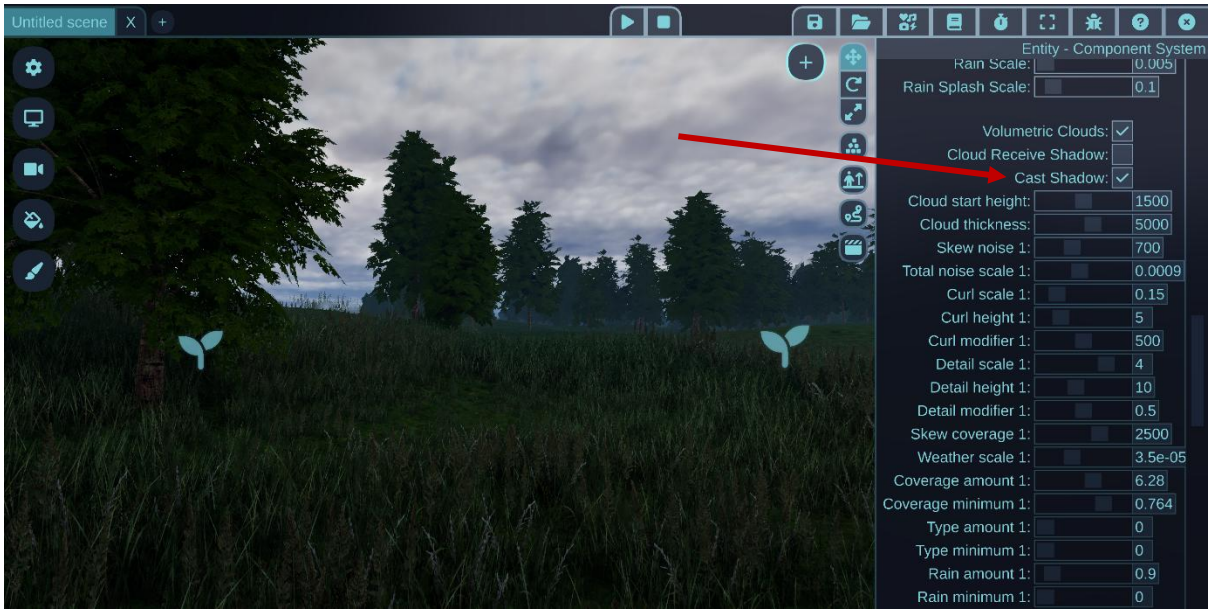
generation, meaning those triangles will be growing grass, so you can paint it where it didn't exist before:

Note: the terrain is special to other objects, because it is using a virtual textures system, meaning that textures are generated at run time for the best level of detail. As such, regular texture painting mode will not work with it. Texture painting will work only on regular objects. There are other useful painting modes as well, such as painting which vertices should receive a wind effect, of which vertices should be part of a soft body or cloth simulation.

One other interesting aspect of the terrain is the weather system which is set up by default to have a nice realistic look to it. It's not required to have terrain to make this, but by generating a terrain you get a nice default weather that you can edit and play with. You can access the weather parameters from the terrain entity, Weather component settings:

There are a lot of weather settings, but for the most interesting ones, try modifying the cloud parameters, namely the “Coverage amount 1” and “Coverage minimum 1” to create an overcast weather:

The weather is very cloudy now, but the sun still seems to be shining, you can enable “Cast shadow” for volumetric clouds as well to block sunlight at some additional performance cost:



Now add some rain with the “Rain Amount” slider:

The time of day can be controlled by the directional light in the scene, which represents the sun. For now, revert the cloudiness and changes so we get a clearer sky, and find the directional light which has a sun icon, then select it:

Now switch to rotate mode by pressing 2 or clicking on the rotation mode button. Rotate the sun until it goes behind the horizon to create a dusk or night scene:

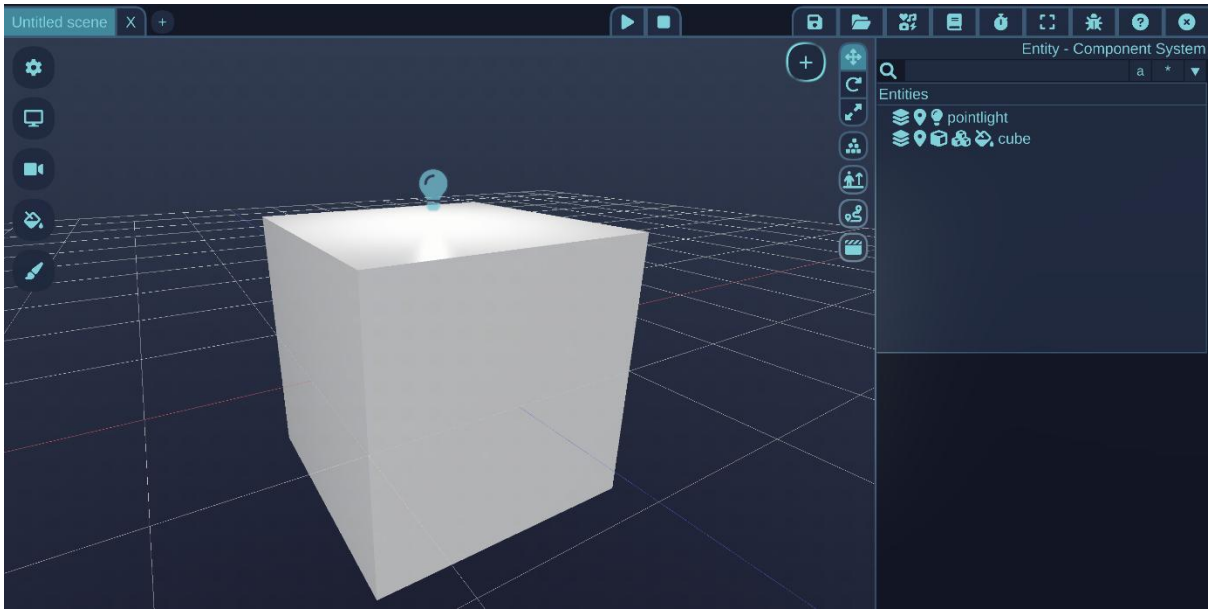
As you can see, the sky colors are changing dynamically with the sun direction, and stars also start to appear when it starts getting dark. You can also replace the realistic sky with a simple sky, or a custom cubemap texture, which will not be affected by the sun direction. In the weather system, you can find a “Load sky” button which will let you import a cubemap or a 2D spheremap image and set it as the sky. You can find the Content/models/sky.dds file is an example cubemap that you can try:

After selecting the sky.dds, you will see the sky will be replaced with the static cubemap texture. Volumetric clouds will still be visible on top of it, although you can also turn them off:

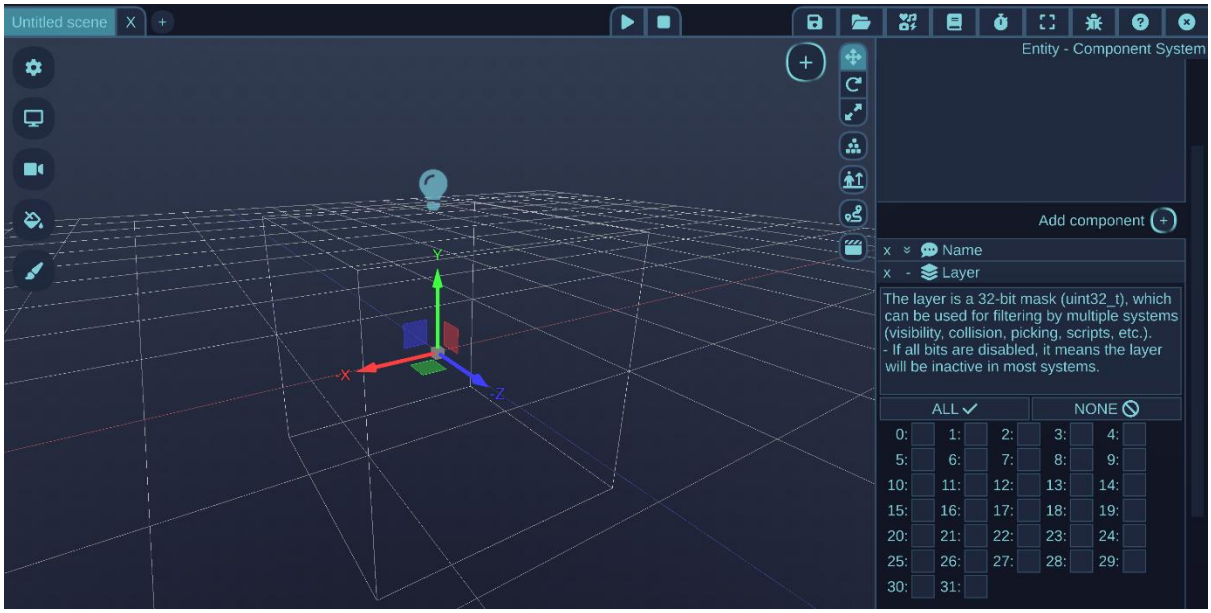
The terrain system has many more features, but I hope showing these basics will help you get started with terrain editing.

Layers

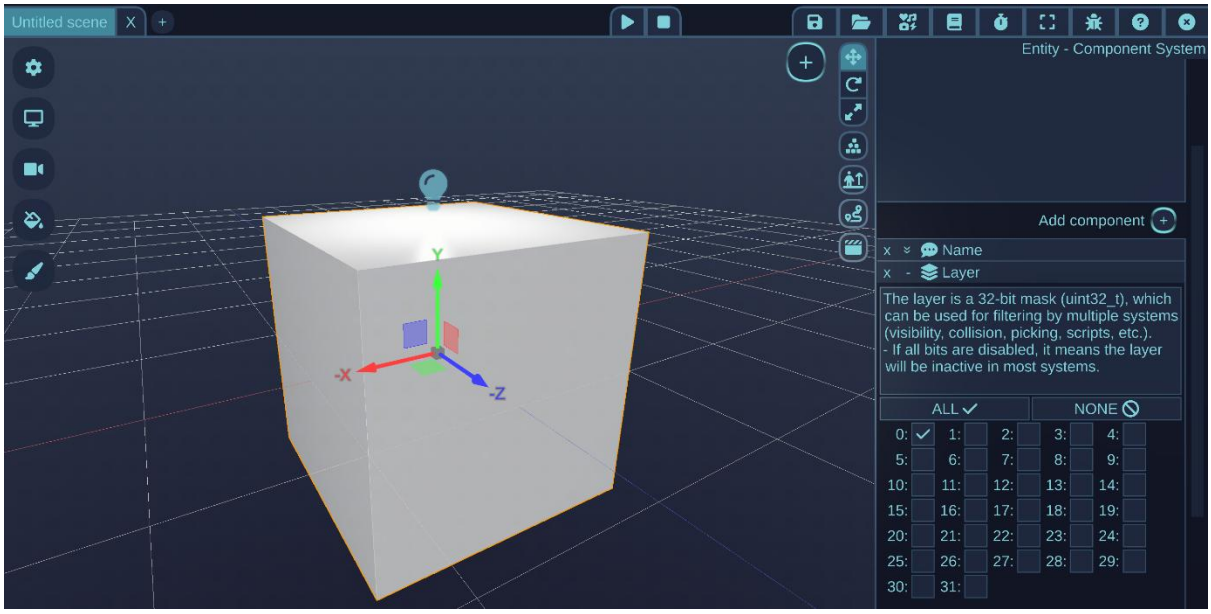
Layers can be used to filter various things. For example telling that a material shouldn't receive light from a specific light source, or an object not to be visible in one of the cubemap reflections, or a decal to not affect a specific surface, or just completely deactivating an object from participating in the main systems. For a simple test, add a cube and a light:



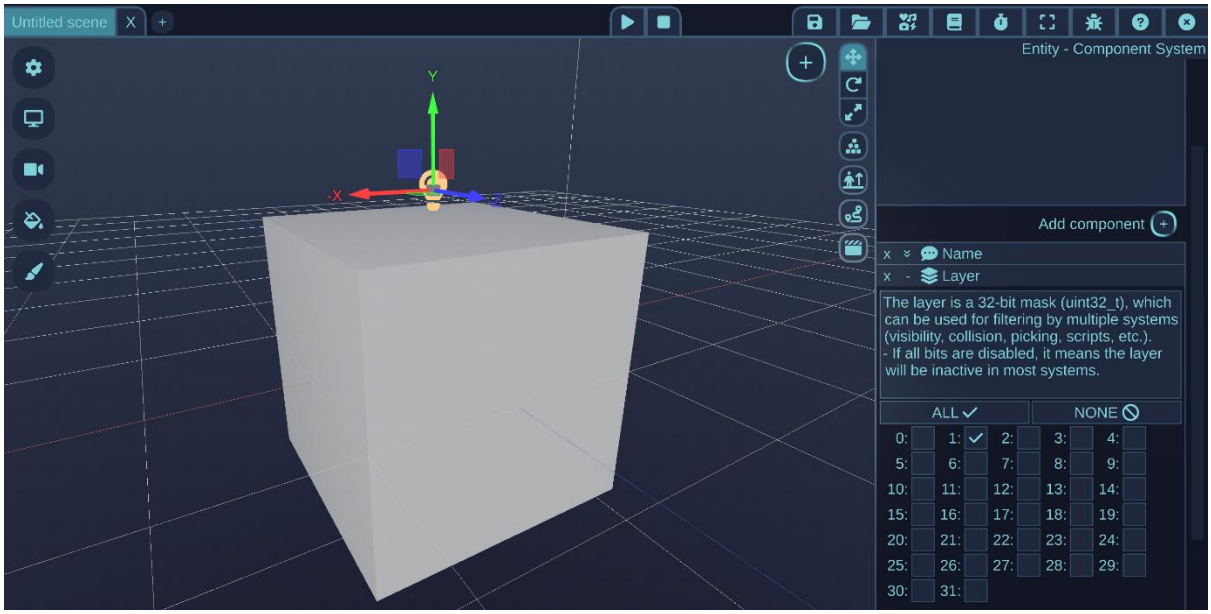
Select the cube and then from the Layer settings disable all layers, so the cube will disappear:



Now enable layer 0, it appears again. This means if none of the layers are active, that entity is disabled. When I set layer 0 to active, it activates again:

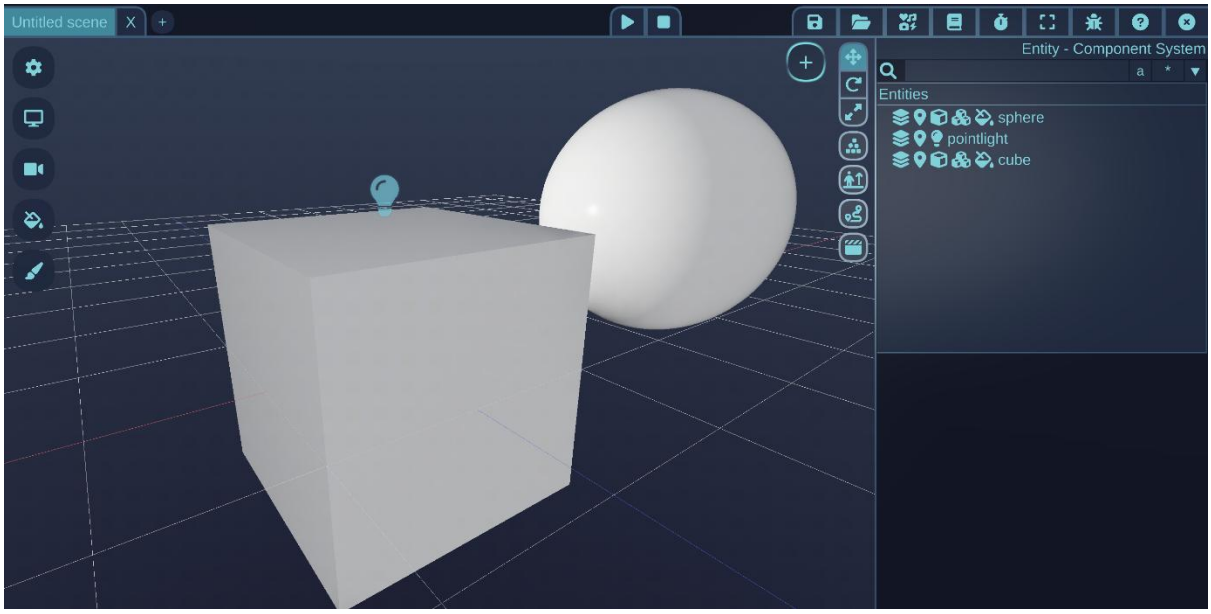


Now select the light, disable all layers of it (it disappears), then activate only layer 1 for the light:



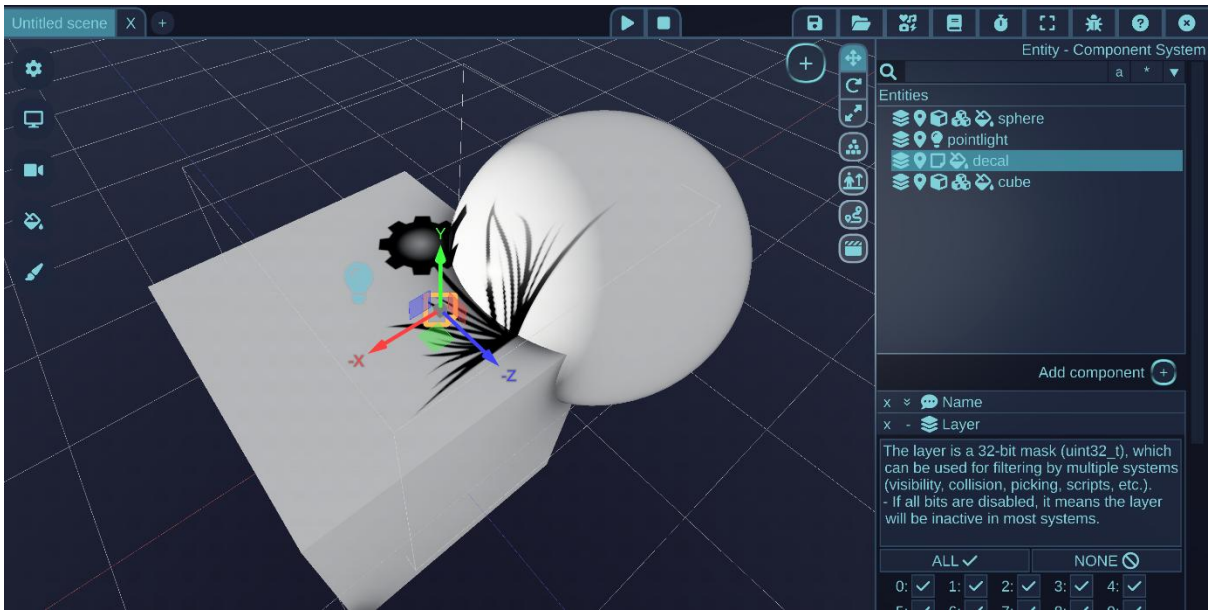
The light still didn't appear, because the layer of the cube and the layer of the light are not including each other. Now add a sphere object and move it away from the cube, the

light will affect the sphere but not the cube:

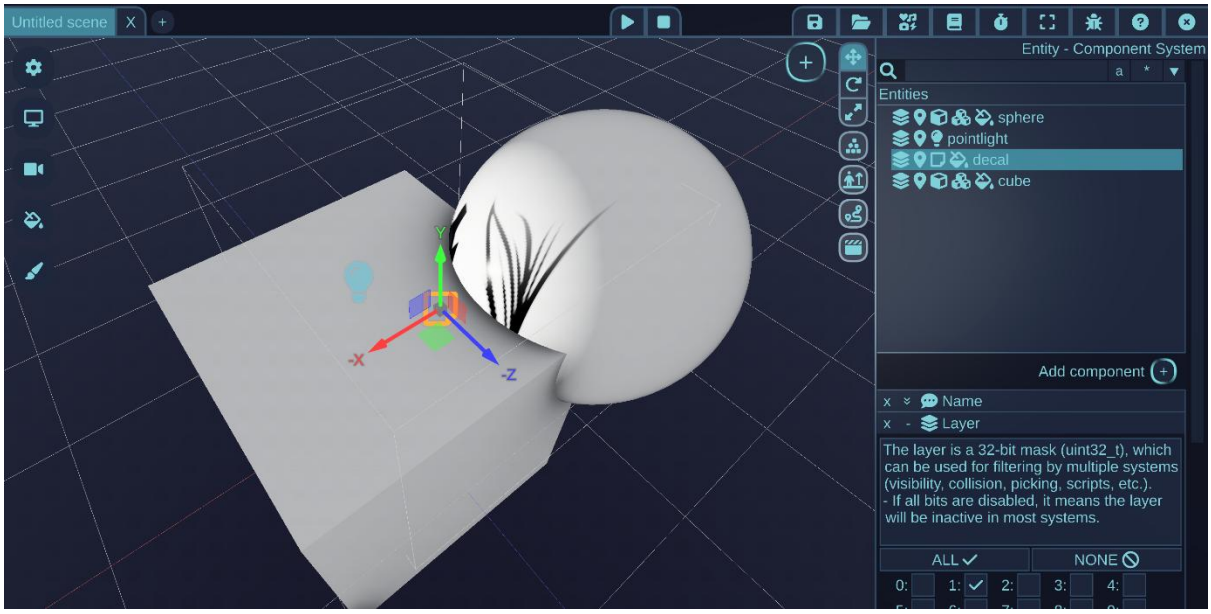


This is because new objects have all layers enabled by default. It's worth to note that if the cube consists of entities where the material is on a separate entity, for the light filtering the material's layer mask needs to be modified.

Similar effect can be achieved for decals too, add a new decal and put it on the cube, but let it also be partly on the sphere:



Now set only layer 1 of the decal to active, so similar to light it will only affect the sphere and not the cube:



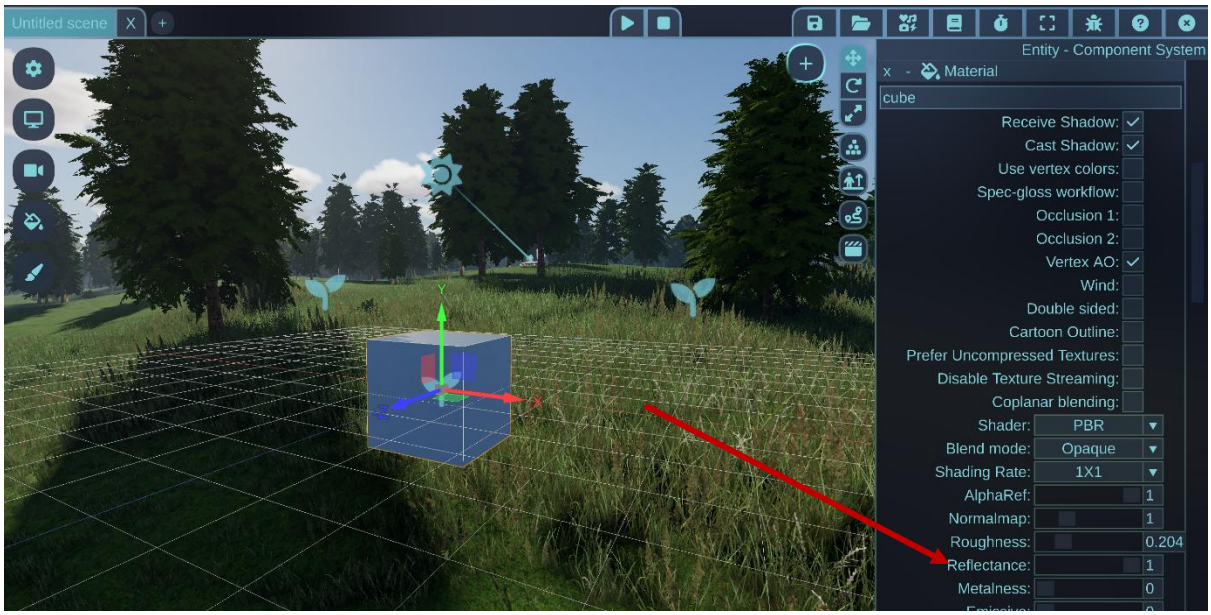
Layers are never modified by the engine, they are completely up to the user, and optional. If there is no layer component for an entity, it is assumed to be active in all layers. Layers are mostly expected to use in gameplay script code to filter things in intersections for example. Some engine systems will also make use of layers, but never modify them:

- Lights
- Decals
- Environment probe capturing
- General visibility
- Ray tracing effects

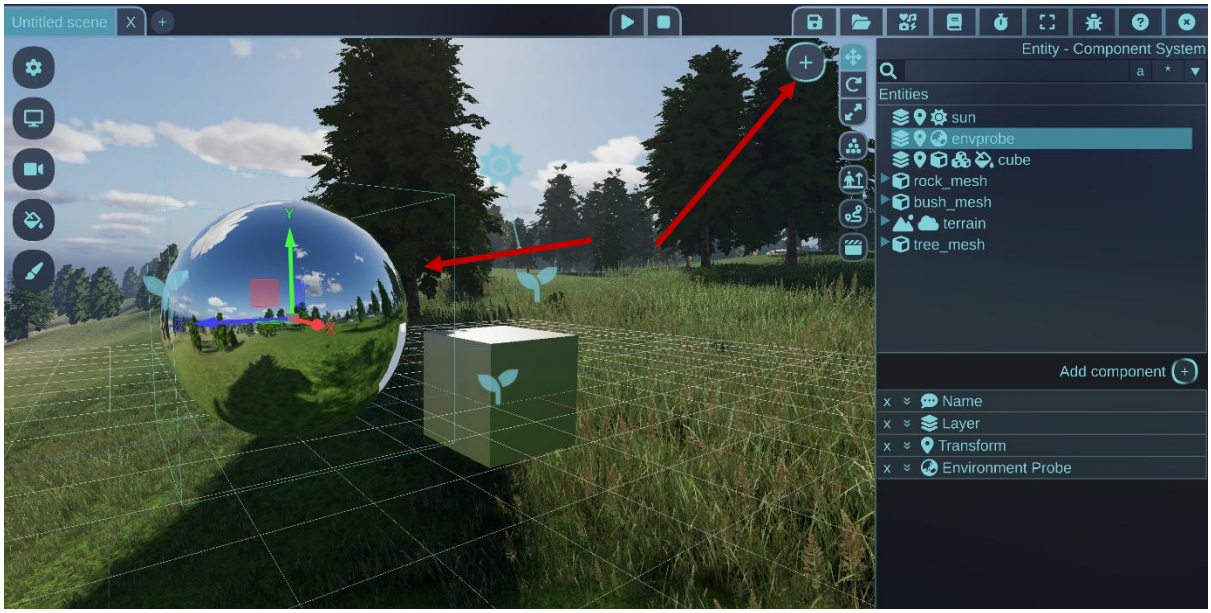
The layers are inherited via the entity hierarchy, so if you disable one layer for the parent, the children will also not have that layer active until they are attached.

Reflection probes

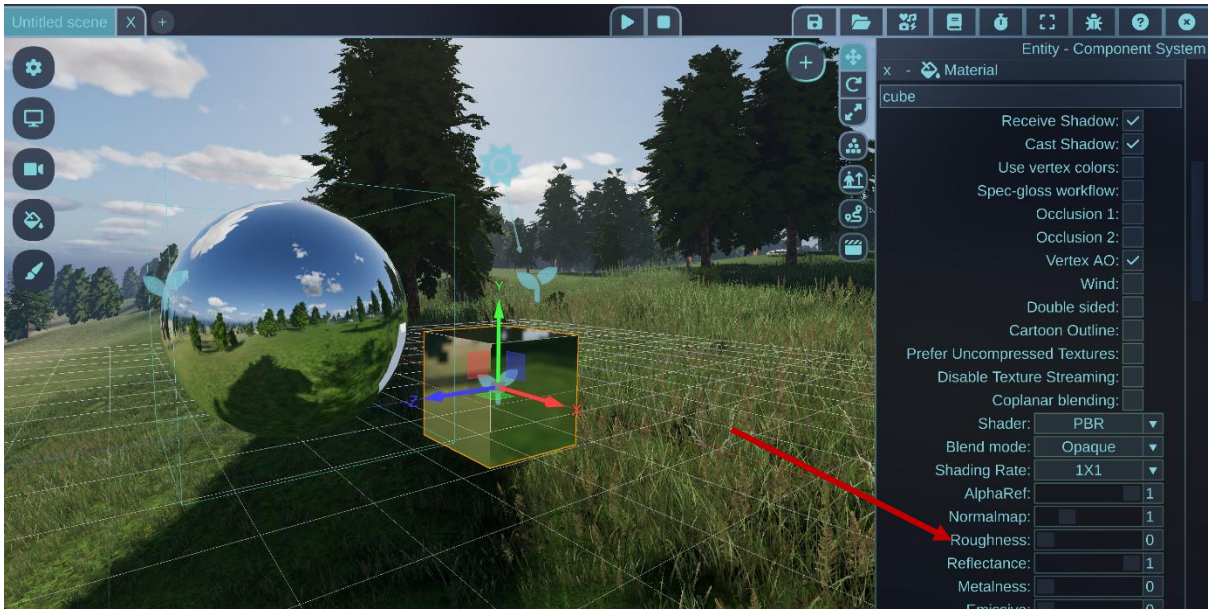
You can bake static reflections for the scene by placing environment probes. This is essential for physically based rendering, where metallic and reflective objects will lose their own color to have reflections. If there are no reflections, then the sky will be used as a fallback. The next thing that you can do is to use reflection probes, possibly with real time reflections (screen space or ray traced). Create a default terrain, and put a cube, then set the cube's material reflectance value to 1, you will get a no-so nice result:



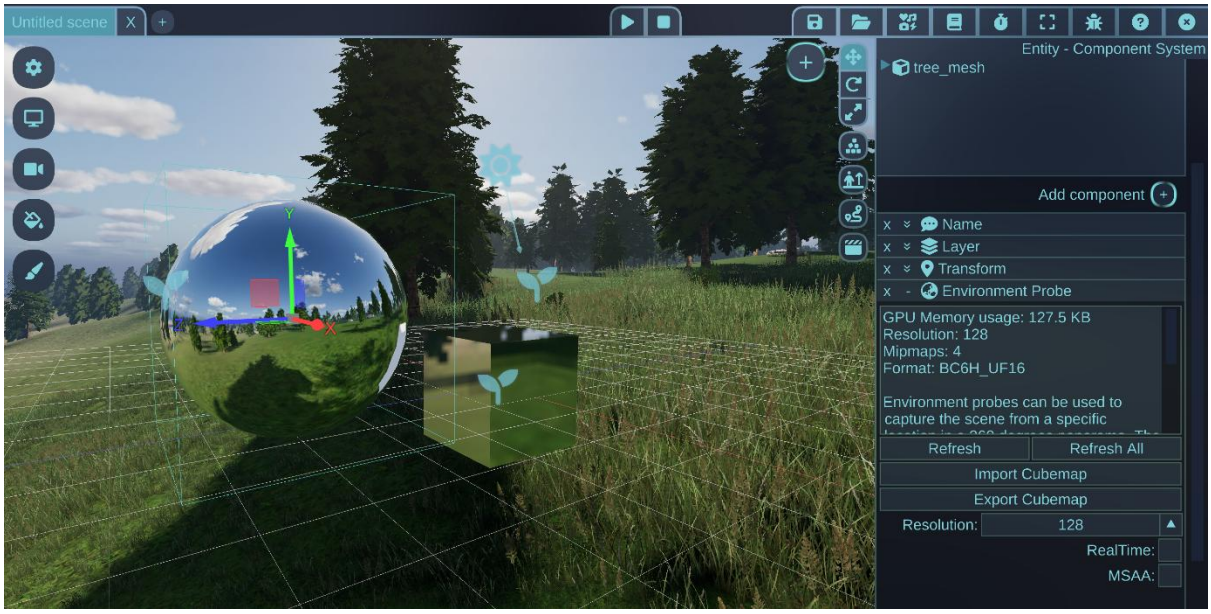
You can place an environment probe from the + button's dropdown list. The probe will be placed in front of the camera, and the scene will be captured in all directions into a cubemap texture. This will be used as static reflection now:



If you decrese the cube's material roughness, the reflections on it will become clearer:



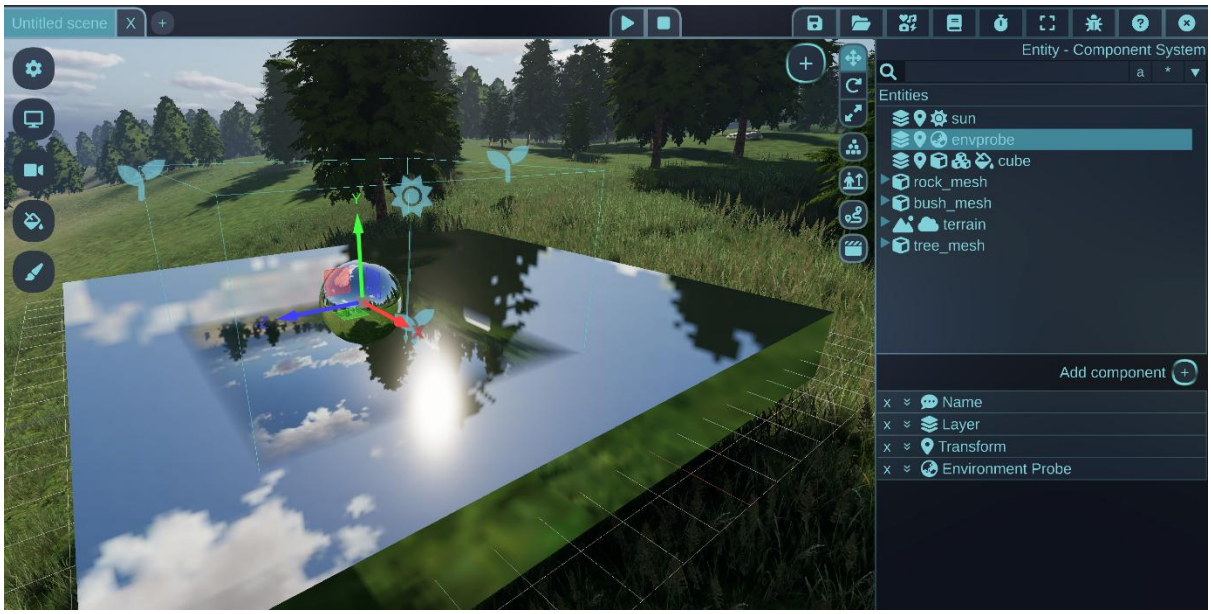
If you select the probe sphere, which is not a real object, but just a visualizer, you can check the Probe settings, where you can also save its texture to an external file or load a cubemap into it. You can also re-render the probe or set it to real time, where it would be rendered every frame. The MSAA option will use multisampled rendering for the probe, which can improve its quality. You can also set resolutions for it, which can increase memory usage but drastically improve their quality.



You can also use layers to filter which objects are rendered into each probe, based on the layer of the probe and object inclusivity.

The first reflection probe that is placed in the scene will act as the global reflection. Any other probes you will place will only act within their bounds (cyan box). You can move, scale and rotate the probes to set the area of influence. The reflection will also appear

more correct the closer the probe bound matches the actual scene geometry, as parallax-correction will take place for them, so the reflections will not appear as infinitely far away. To demonstrate it, scale the cube horizontally and adjust the probe to cover it only partially:



As you can see, within the probe influence, the reflection is mapped onto a cube, and outside of it the global reflection is visible as infinitely far away.

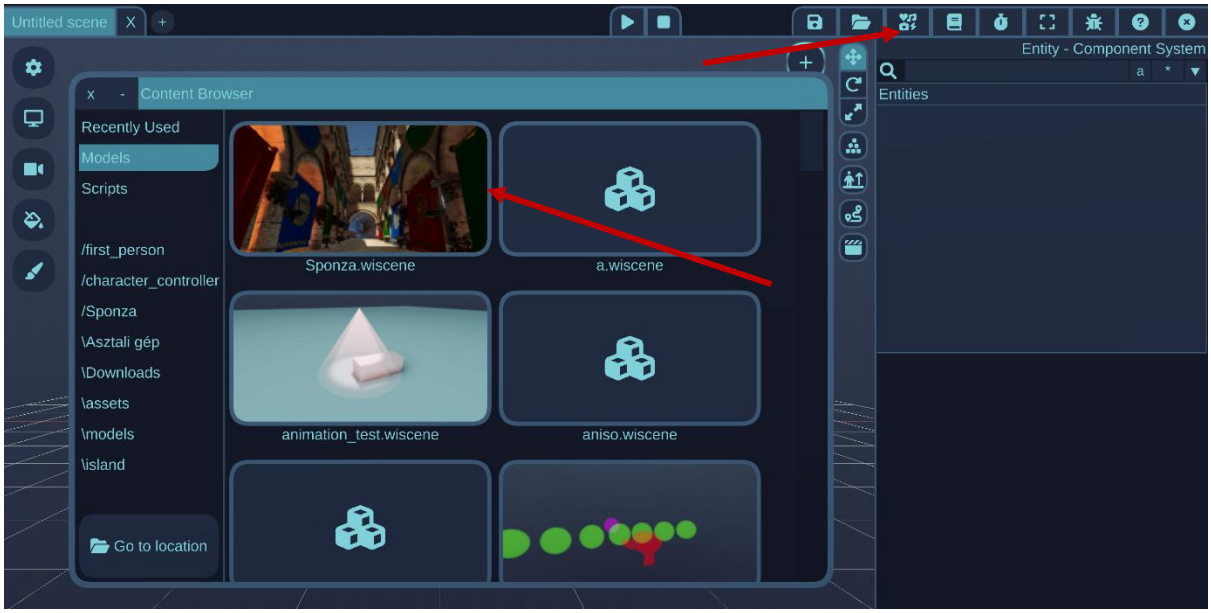
If you use screen-space reflection, the environment map reflection will be overridden by the screen-space effect where it can, so it will appear even more accurate. However, screen-space effect will not be visible on all parts of the screen, and in those cases it will fall back to the environment map.

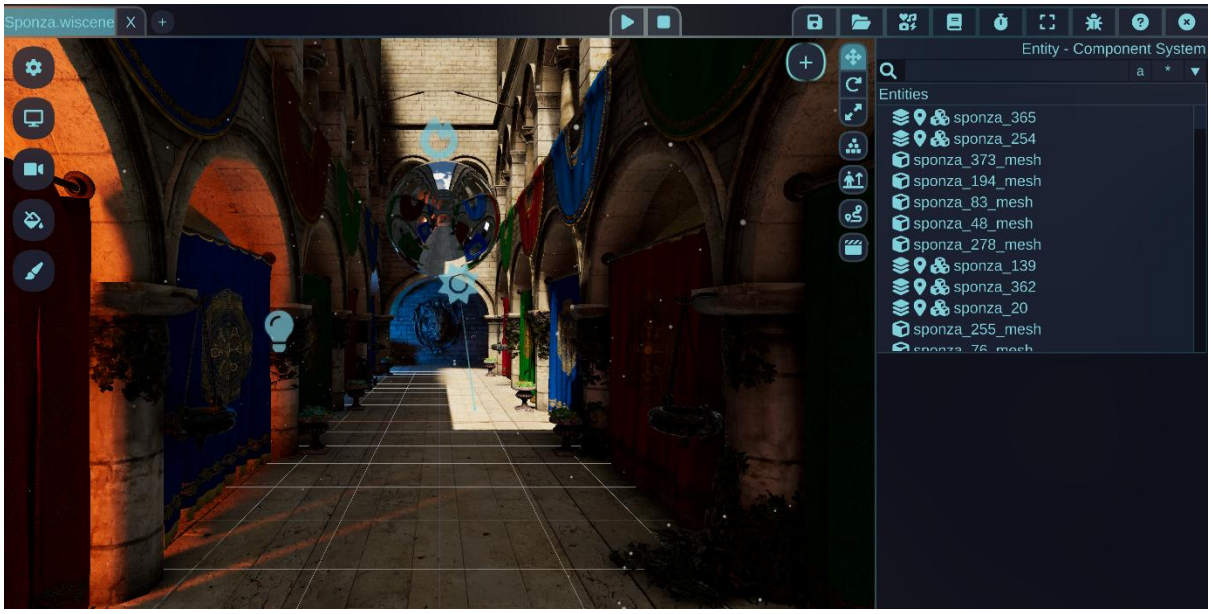
In case of ray-traced reflection, the environment map will be used where there is a reflective surface visible within the reflection, to avoid computing recursive reflection, which would be much more expensive in terms of performance than the already expensive ray-traced reflection.

Additionally, the average colors of the environment probes will also be used for a simple diffuse color fallback when there is not a more accurate solution being used.

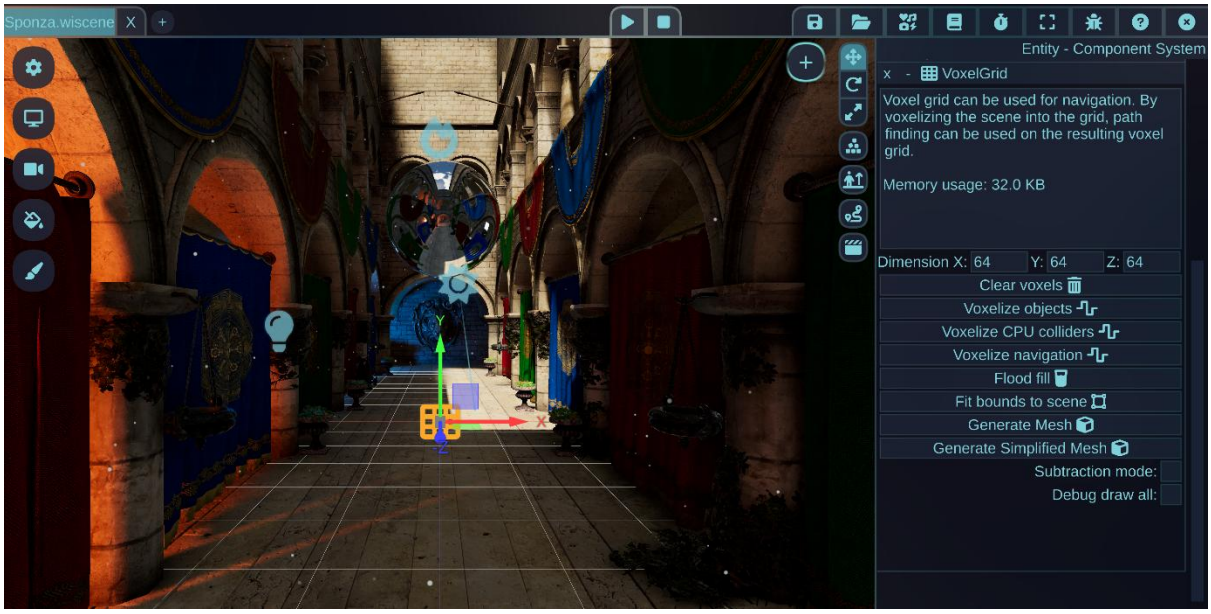
Navigation/path finding

You can create navigation structures right from inside the editor. To be able to use navigation, you have to create a voxel grid, and voxelize the scene. Open the Sponza model from the Content/models/sponza folder. You can also select it from the Content browser easily:





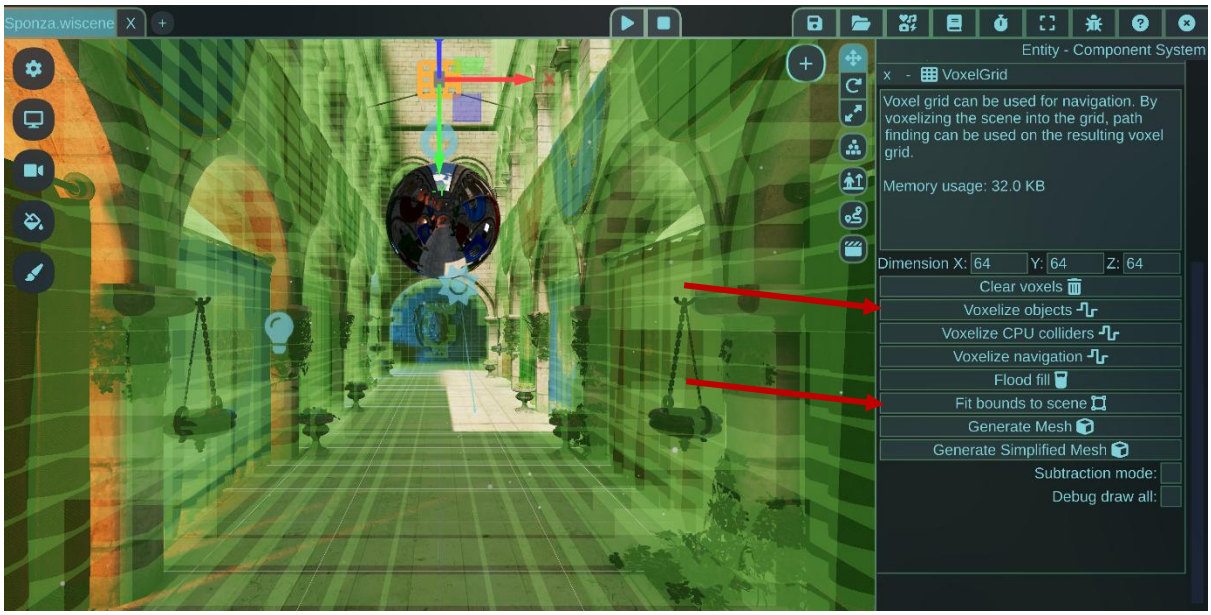
Once it's loaded, add a Voxel Grid from the + button's drop down menu. It gets selected immediately, so open the Voxel grid settings:



From here, you can press these buttons:

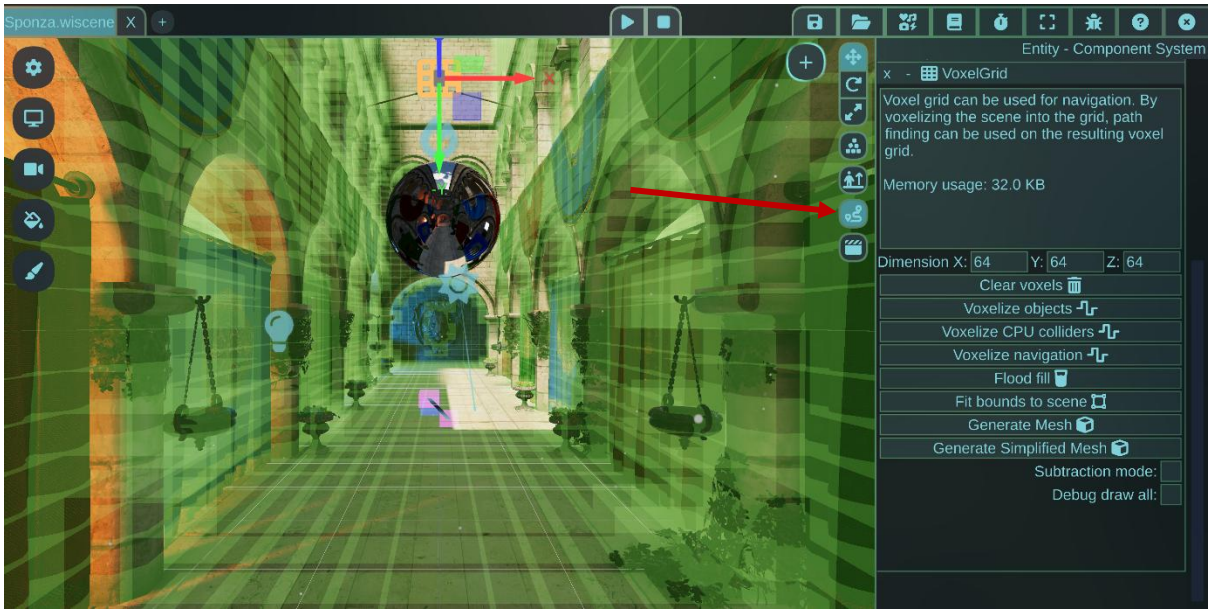
- "Fit bounds to scene", this will automatically determine the voxel grid size to include everything in the scene. An other option is to scale the grid by hand with the transform tool's scale mode, in this case it's like scaling a box.
- After that, "Voxelize objects" will voxelize every object and insert it into the grid

Now the voxel visualizer will appear as green transparent cubes for as long as the voxel grid entity is selected:

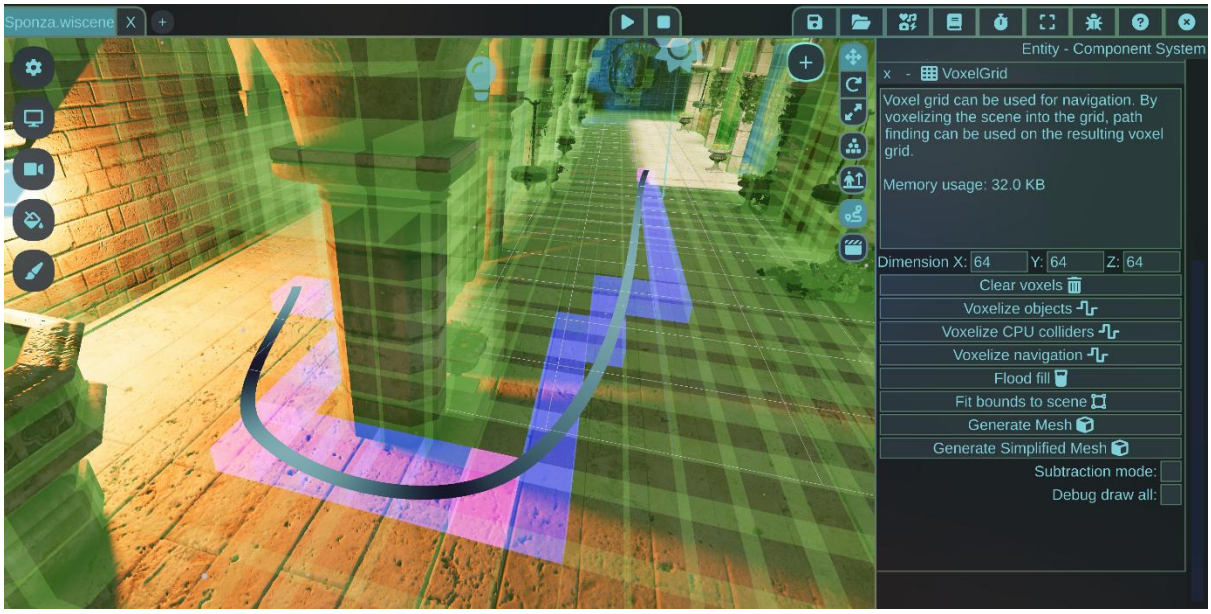


This voxel grid can be used by game logic to query paths from one point to another. You can also test it in the editor by activating the navigation testing button in the toolbar on the right-hand side of the 3D work area. When it's activated, two waypoints will appear

on the floor:

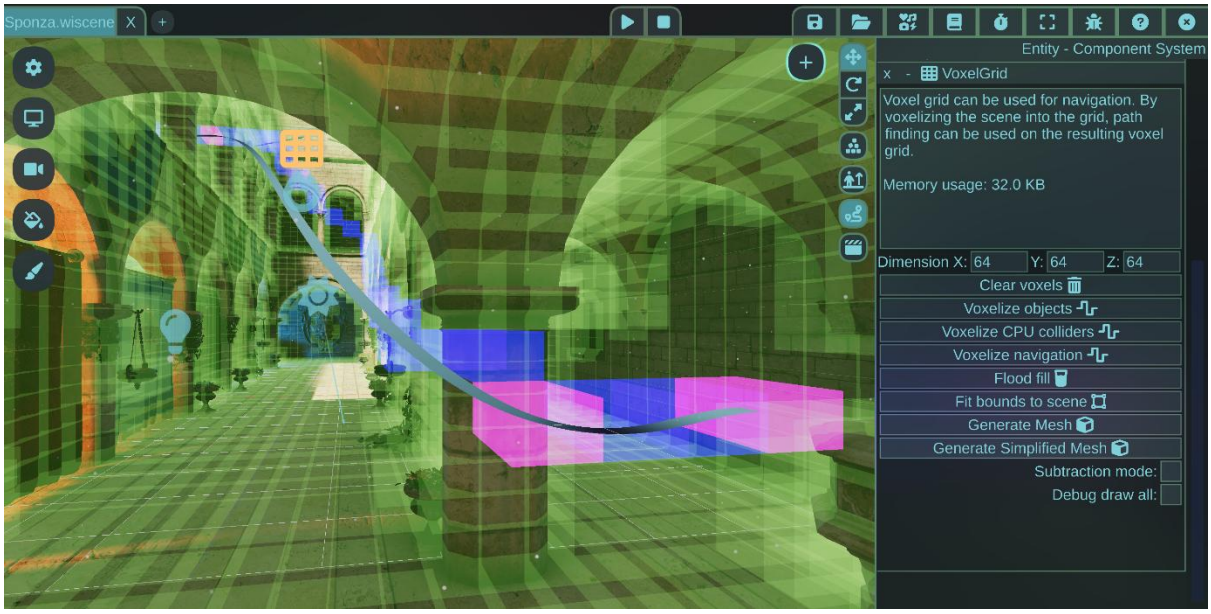


You can move these START/GOAL waypoints by holding down the F5 and F6 keys, and clicking somewhere with the middle mouse. For example when I place one of them on the other side of the columns and curtains, you will see the path being computed immediately and drawn as a curve:



There are two modes for path finding, the first one that is shown above which is path finding on the ground or surfaces. The second one is path finding in the air. To do this in the air, you can use the F7 and F8 buttons and middle mouse to place waypoints into

the air, in front of the camera:

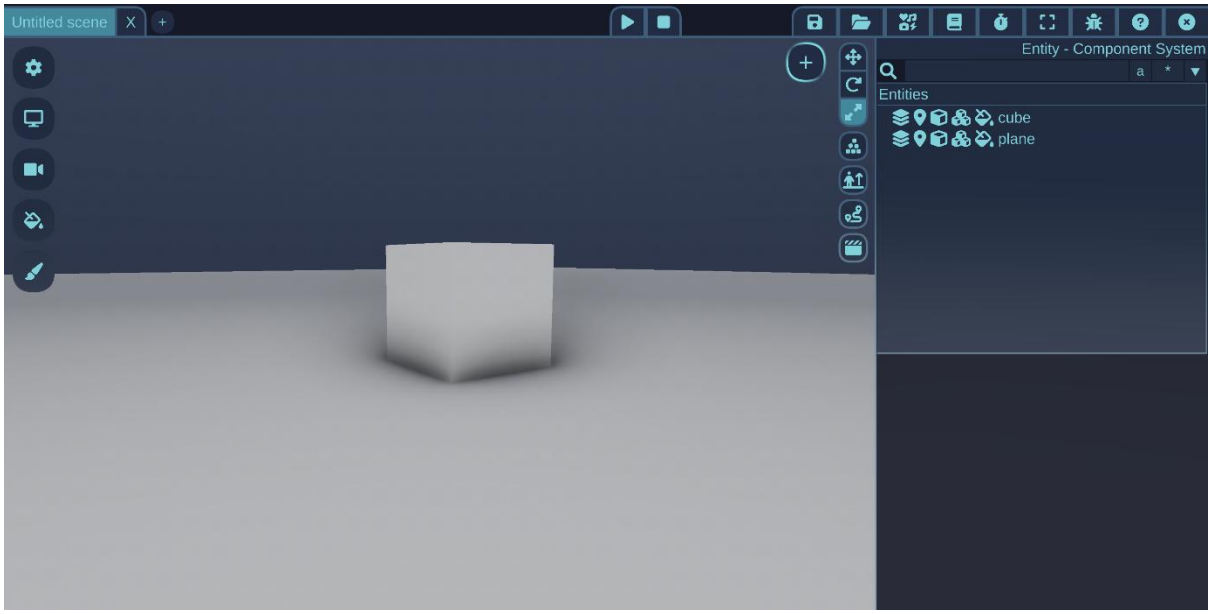


The path finding visualizer has a curve, blue voxels and pink voxels. The curve shows the recommended curve for the path, blue voxels show the traversed voxels for the path, pink ones show the main voxel waypoints that need to be visited. The gameplay scripts can make use of the pink voxel positions very easily to tell a general direction that the non-player character should be going towards at any given time. For example, look at the character controller script in Content/scripts/character_controller.

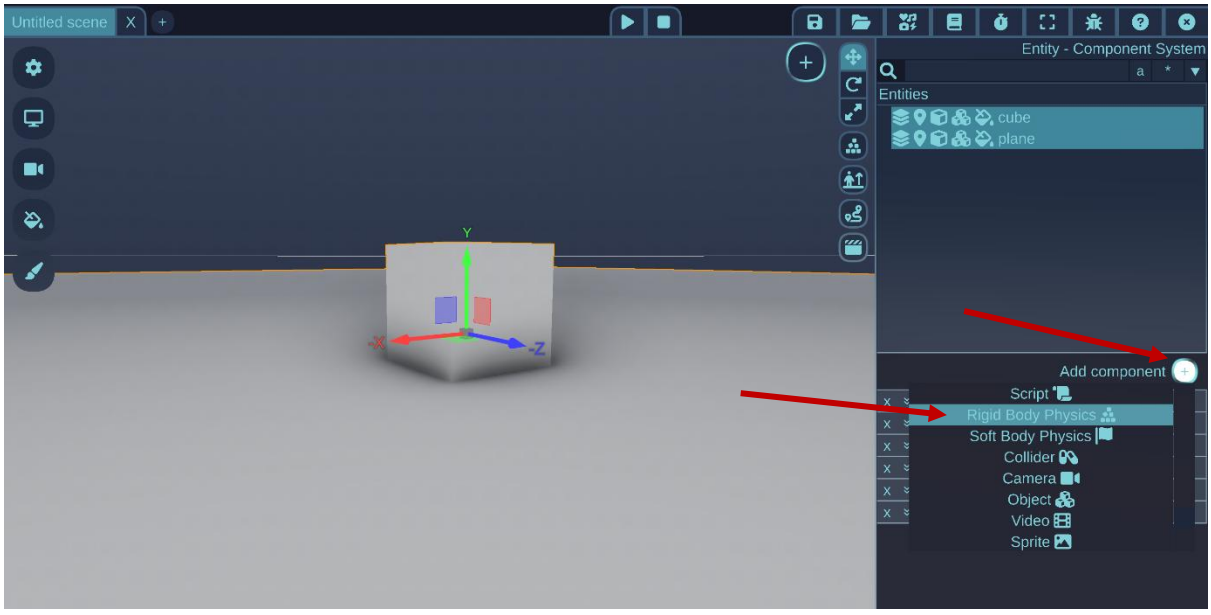
To customize the voxel grid more, you have some more options too, like setting resolution, voxelizing colliders, or voxelizing navigation (voxelizing only objects that are tagged as “navigation mesh”). You can also use subtraction mode, so the voxelization process will not add voxels into the grid but remove them instead. The idea behind this is that you can cut holes into the voxelization with a shape, for example a voxelizing a collider with subtraction mode into a filled voxel grid.

Physics

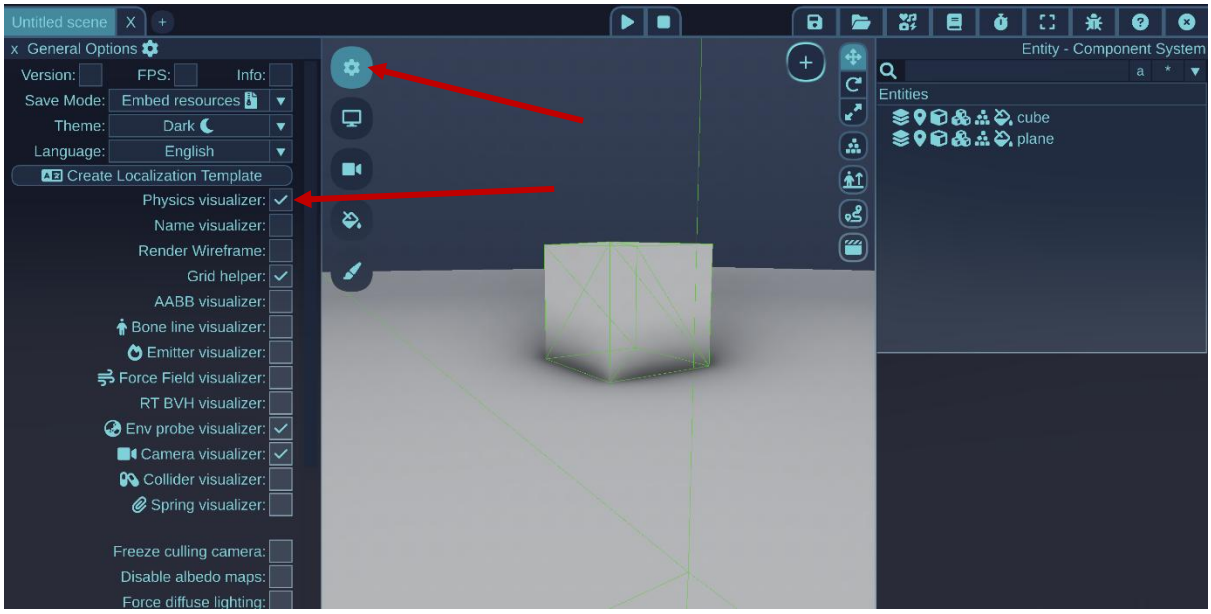
To understand how to set up physics, let's walk through a simple example. Create a plane and a cube object and set them up as if the plane is the floor and the cube is sitting on top of it:



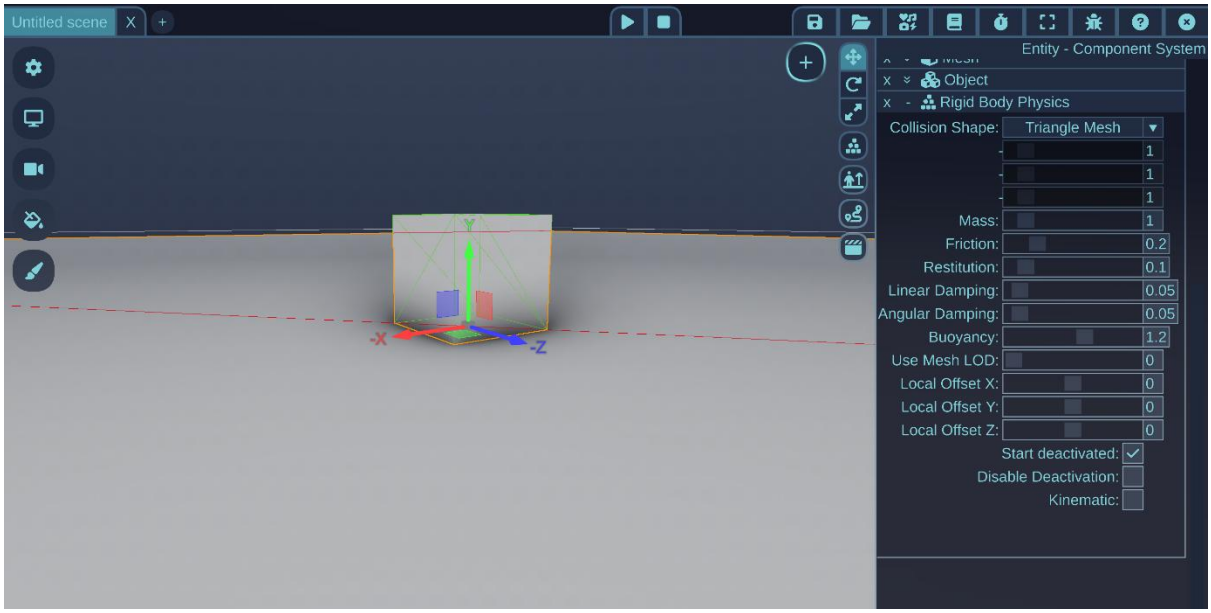
Now add rigid body physics components for both of them. You can select both at the same time and add the component:



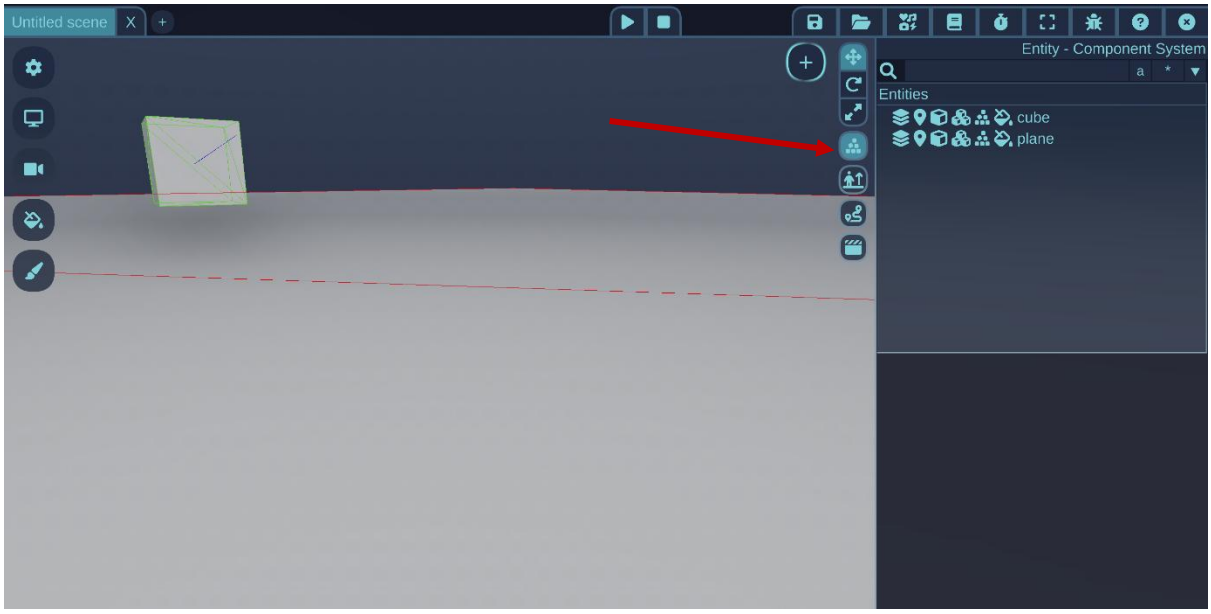
Turn on the physics visualizer from the general options:



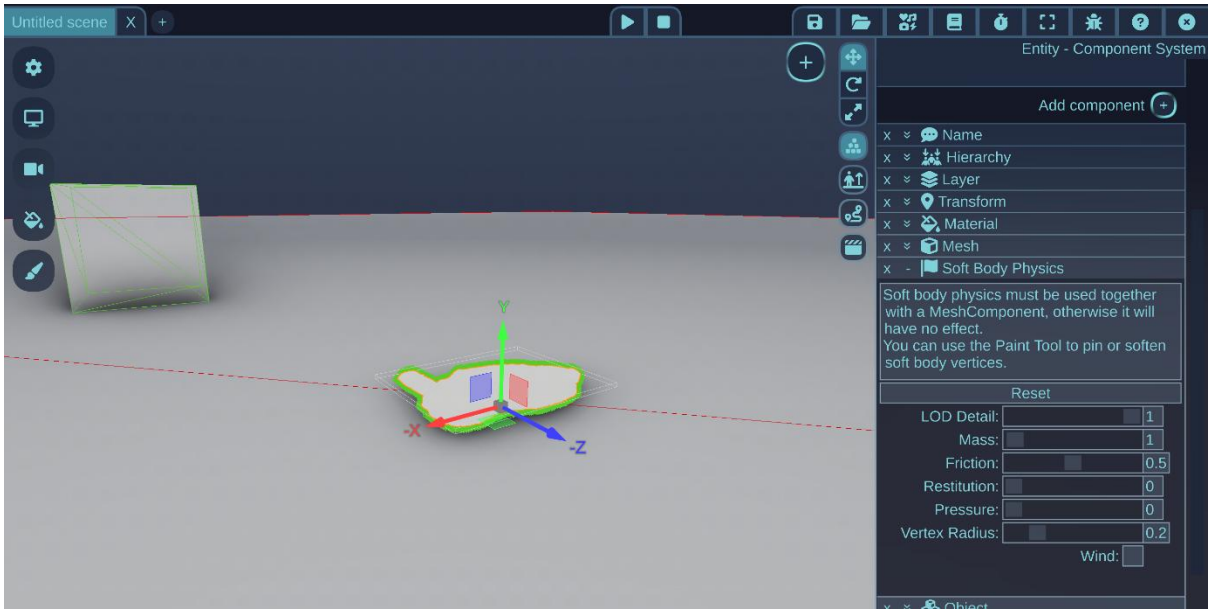
Since both are using the “Cube” physics shape, the cube is good, because the collision shape (green wireframe) is closely matching the graphics mesh. Select the plane and set its physics shape to “Triangle mesh” which is a good choice in general for static physics objects, which the floor will be. Now the collision shape matches both objects:



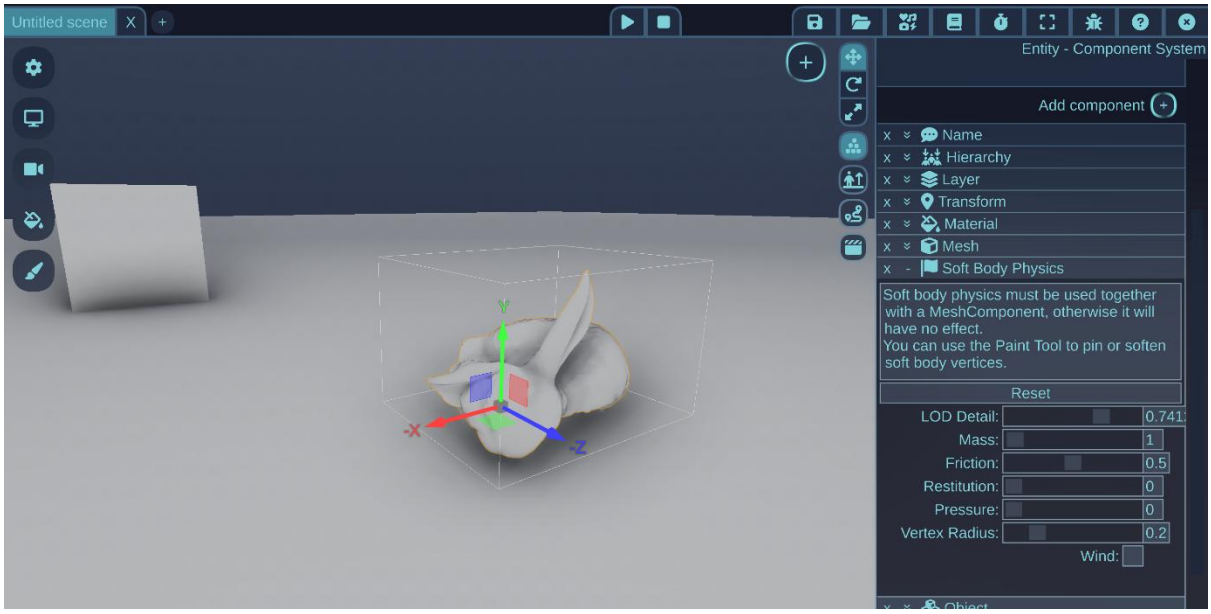
Now both objects are dynamic (but deactivated), but we want the floor to be static, so set its mass to 0. Enable the physics simulation with the button on the right side of the 3D work area if not already enabled. Now you can grab the cube with the middle mouse and drag it around/drop it with physics. It doesn’t need to be selected for physics dragging to work.



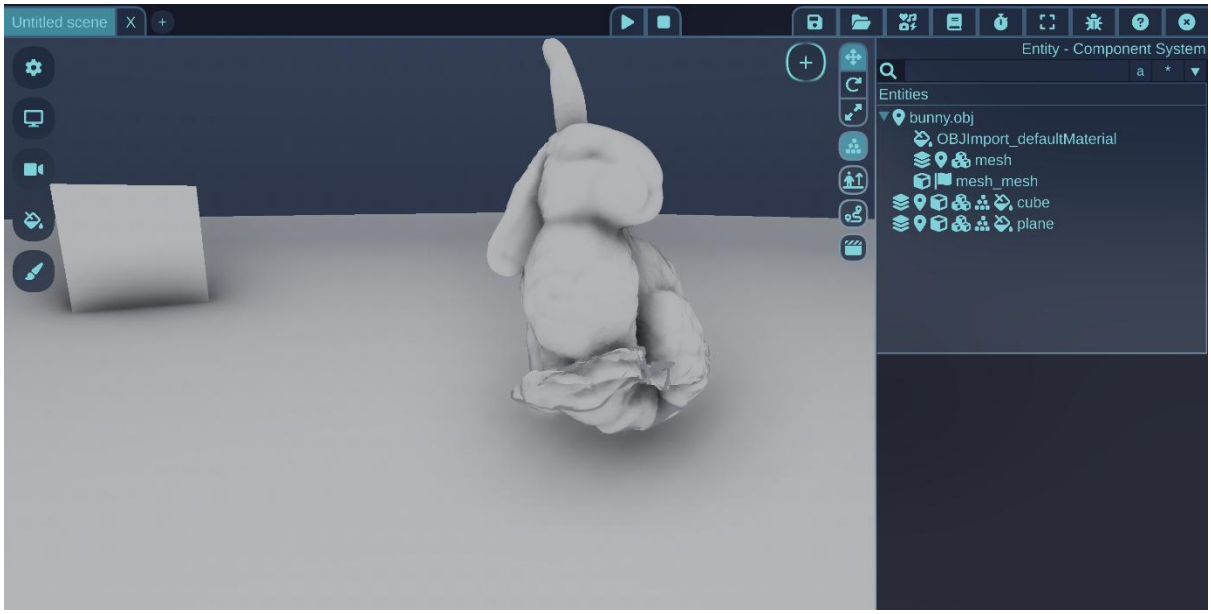
Now add the bunny.obj mesh from the Content/models and place it on top of the floor and add a soft body component to it. It can take some time and since this is a pretty high poly mesh, the soft body physics will be slow and the bunny deflates:



If you reduce the LOD Detail slider, you can make it more stable, and I also disabled the physics visualizer for now:

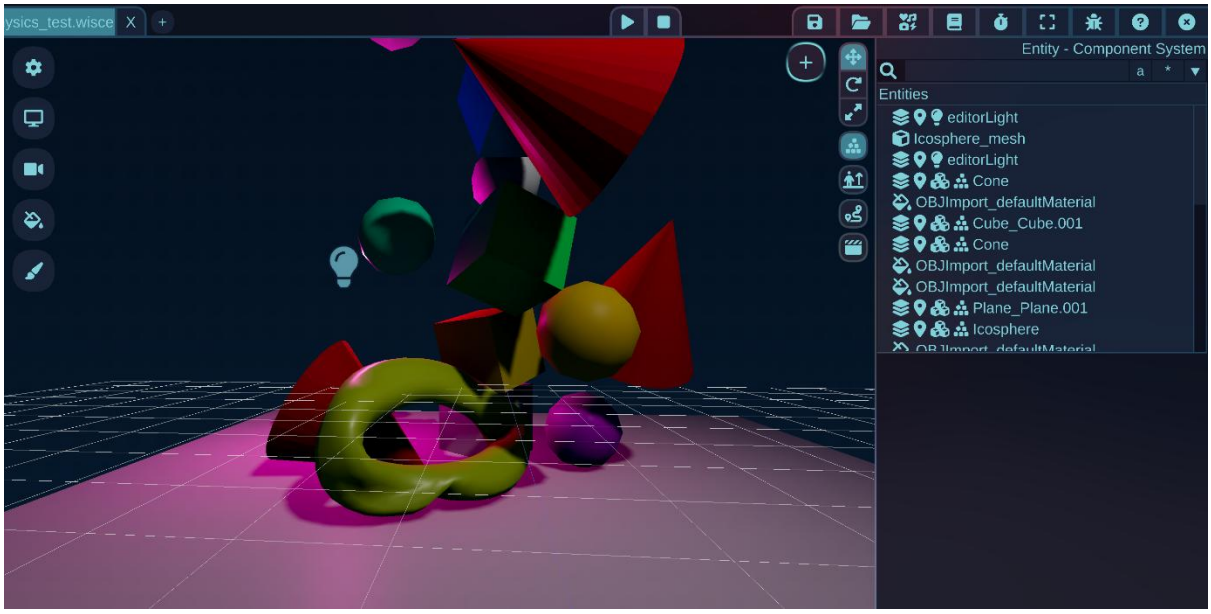


You can also drag soft bodies around with the middle mouse, they don't need to be selected:

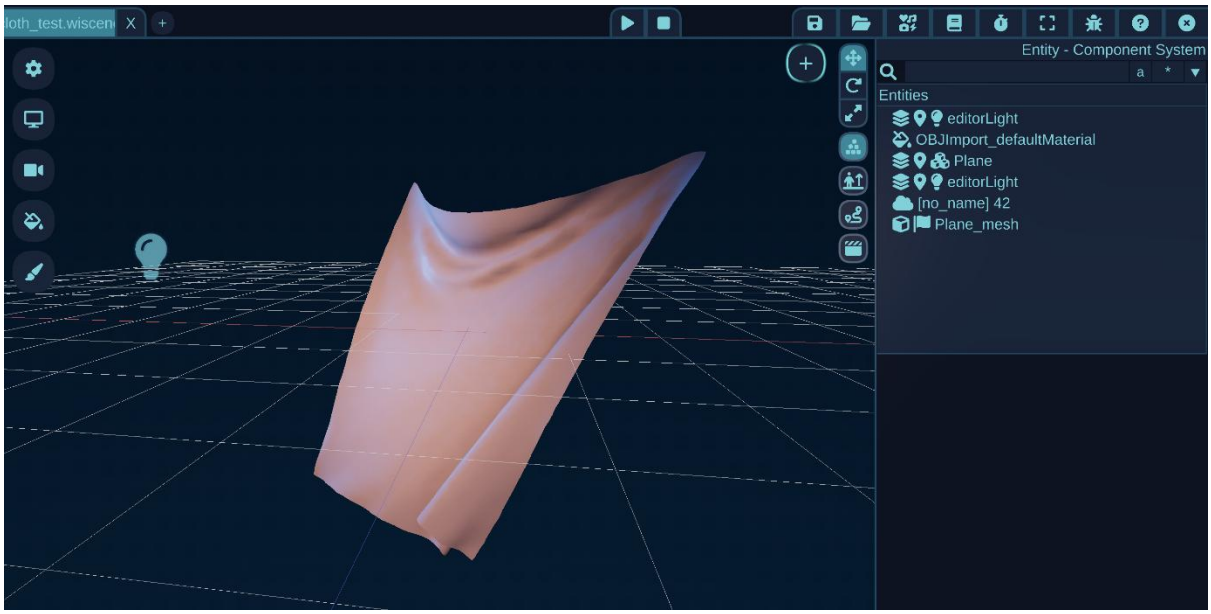


For nicer examples, check out the physics_test.wiscene and cloth_test.wiscene models in Content/models folder. The physics_test is showing a bunch of falling rigid bodies

and a soft body:

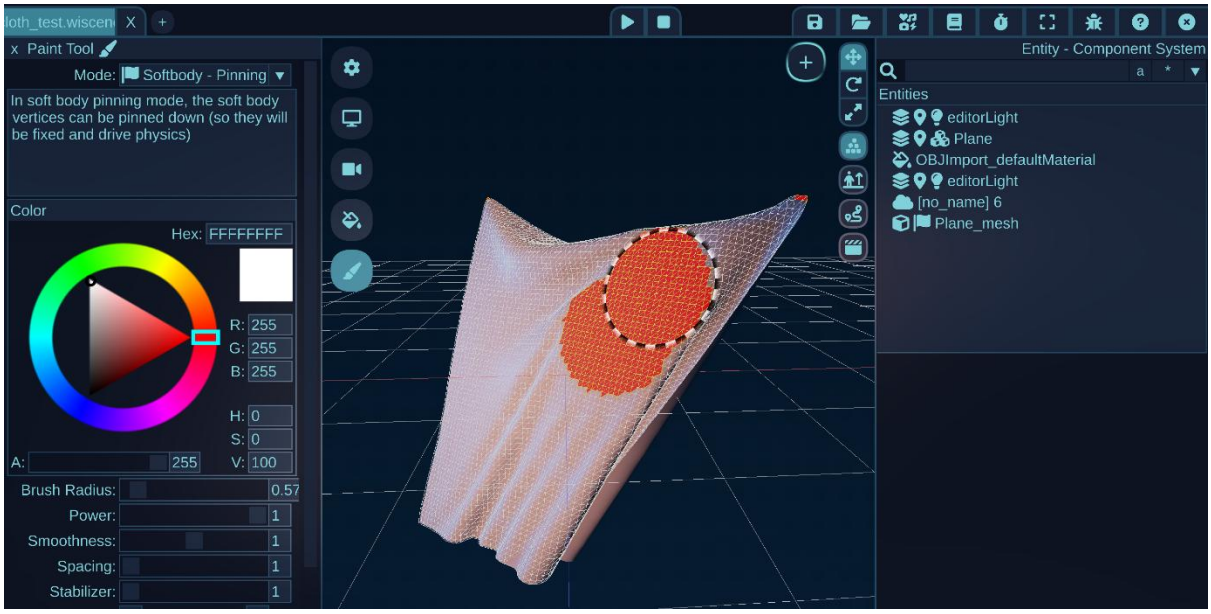


The cloth_test is showing a simple subdivided quad shape used as a flag:



This flag is also showing something else; the soft body is pinned in place so the whole thing doesn't fall down, but some of its vertices are simulated. Enter the Soft body

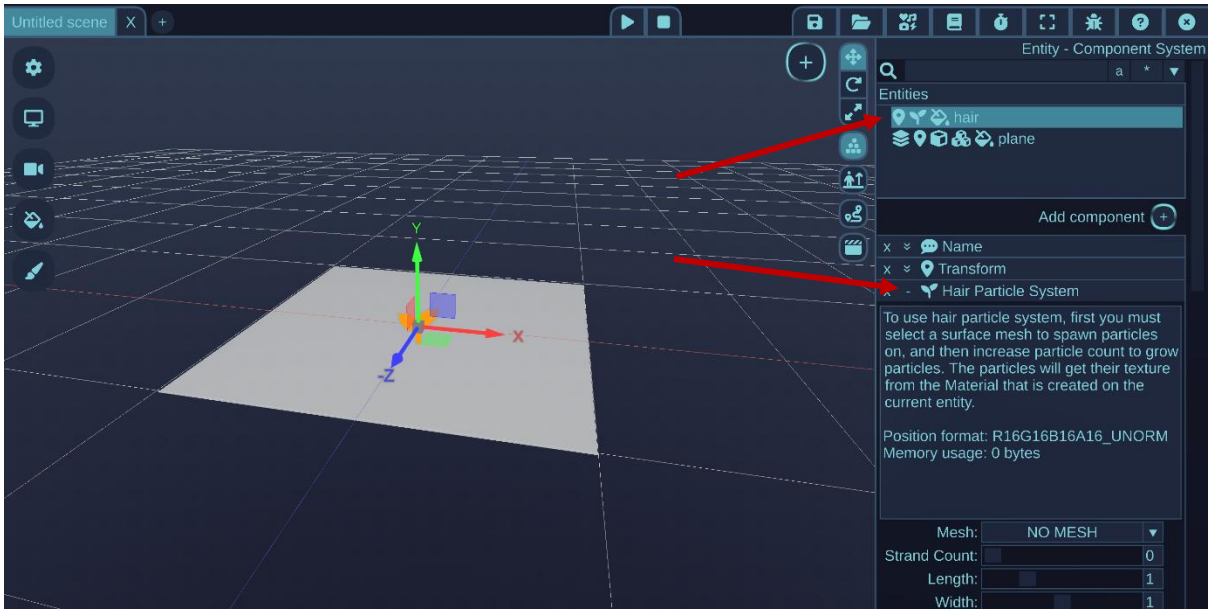
pinning mode of the paint tool to paint these pinned vertices onto the soft body mesh:



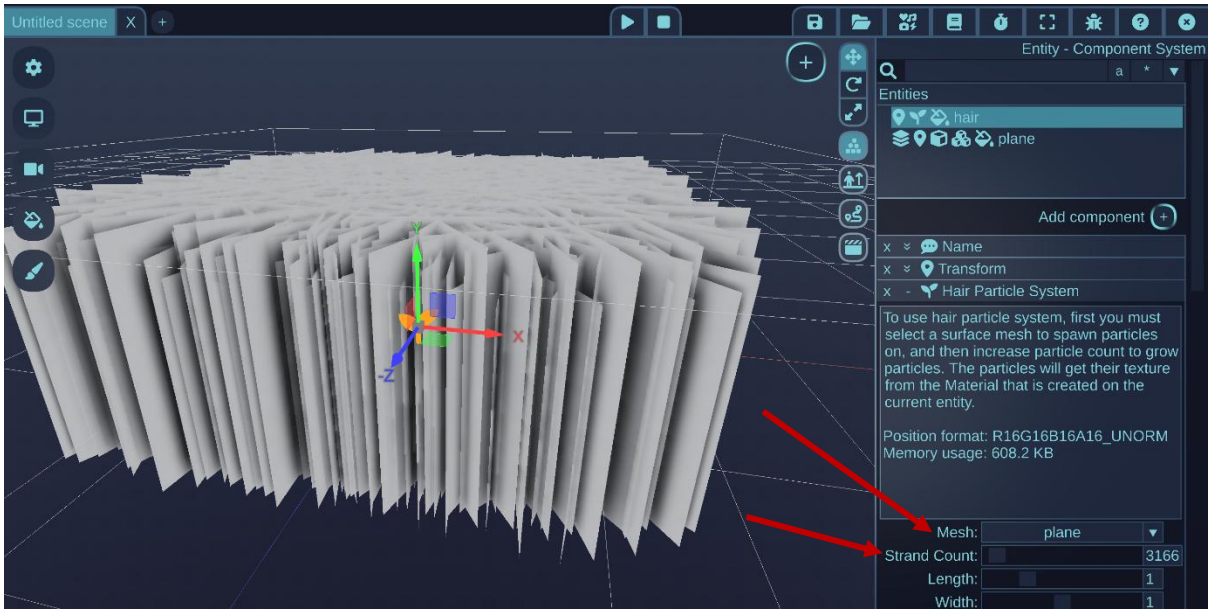
The reverse is also possible with the paint tool's Softbody – Physics mode, to remove pinned vertices by painting. Note that painting this pinning on high poly meshes can be slow, so prefer to use meshes with as low polygon count as possible for soft bodies.

Hair Particle system / grass

The hair particle system can be used for generating particles sticking to mesh surfaces procedurally. To create a patch of grass from scratch, it can be done by first creating a mesh to grow particles from. Add a new plane object, scale it up a bit, then add a Hair Particle System entity with the + button in the top right corner of the 3D work area:

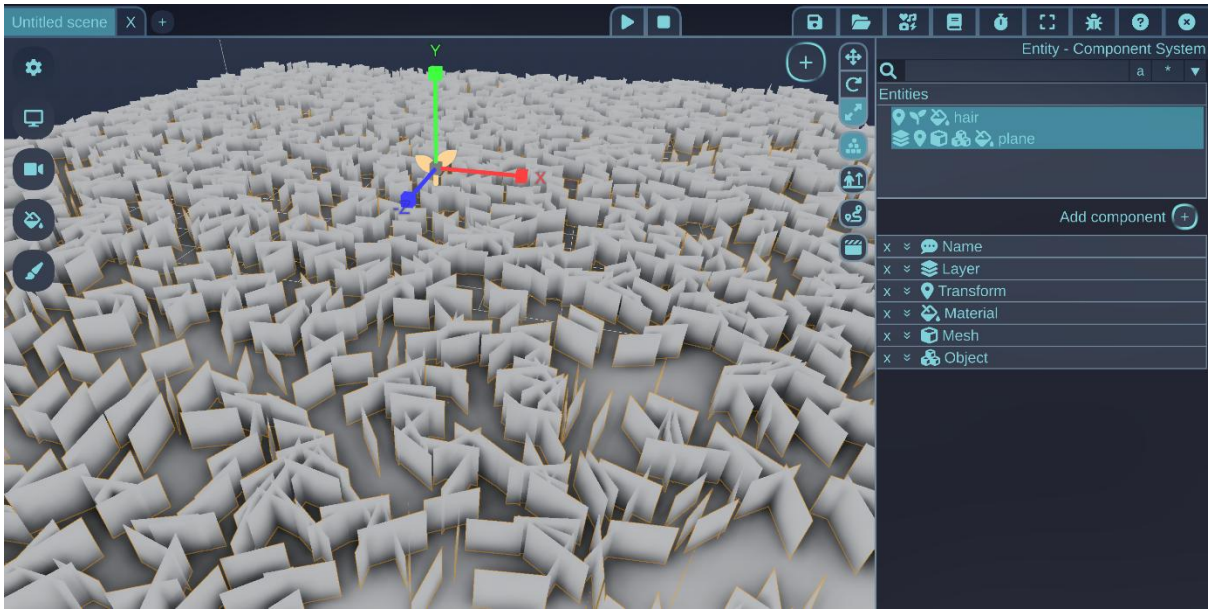


Then select the plane mesh from the hair particle settings, and increase the strand count:



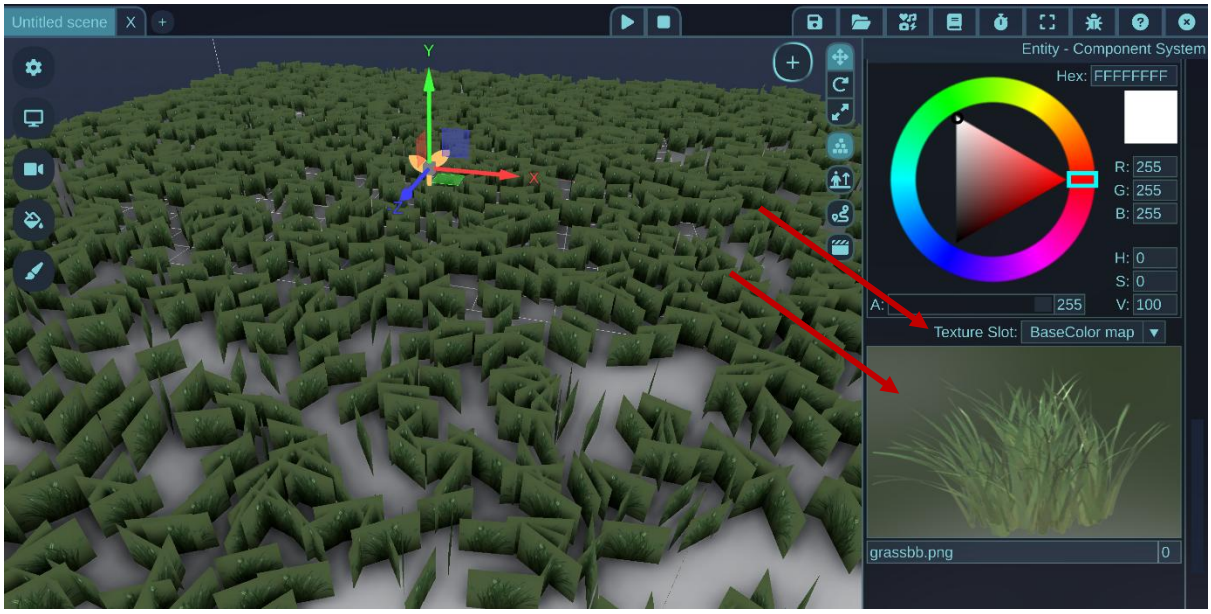
Now you see the particles are added, but they are too large, you can either adjust their length, or scale them up. To scale them up together with the plane object, select both the object and the hair and scale them. If you scale them up, their length will remain the

same, but the surface area is larger, so they will be distributed in a larger area:

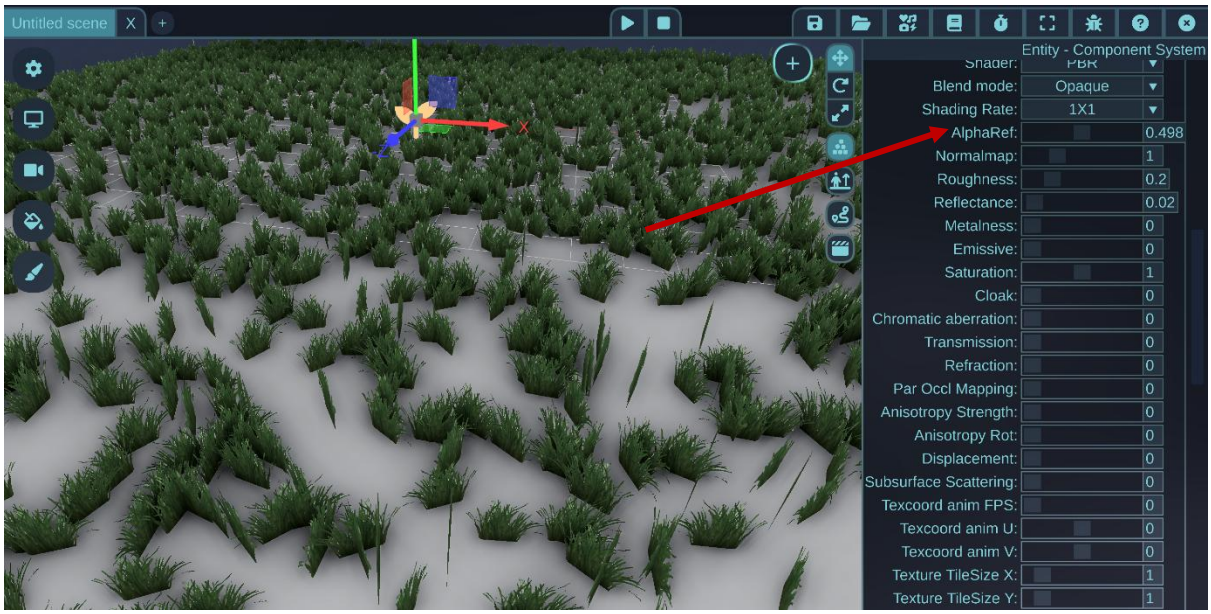


The particles are a bit boring because they are all just white quads. To create something like grass, you can go to the hair entity's material and set its texture. The simplest way is to set it to a texture that contains a single grass patch on a transparent background, something like this (you can find this texture in Content/models/grassbb.png):

So you open the material of the hair entity and set this texture for the “BaseColor map” slot:



To cut out the transparency of the grass texture, reduce the material’s AlphaRef below 1:

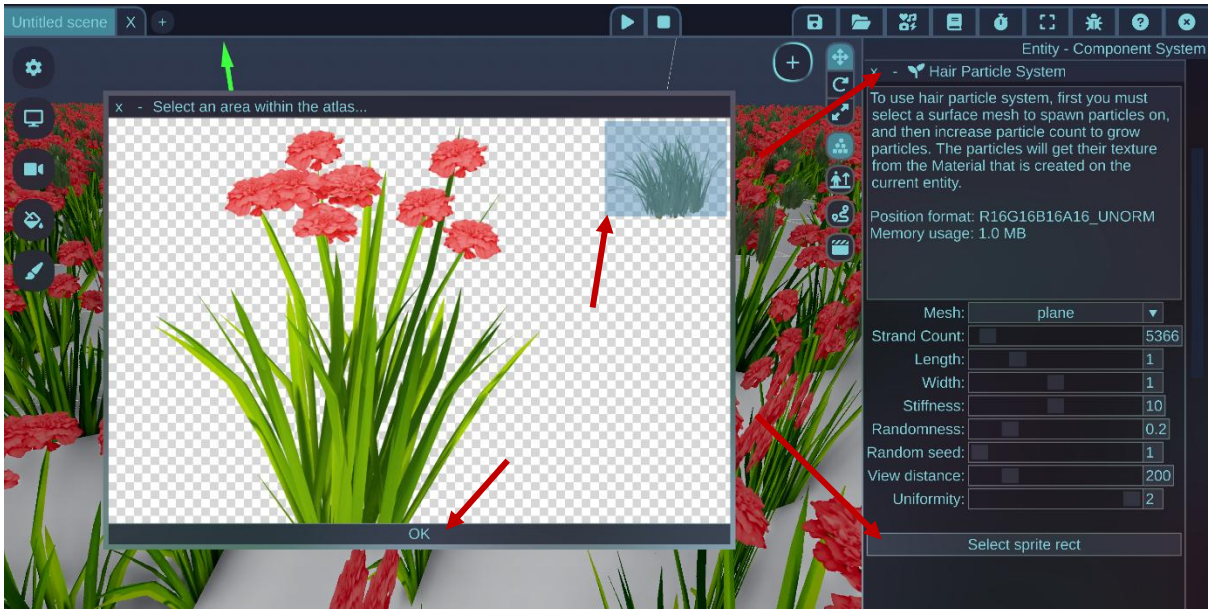


The hair particle system also lets you use more variety by using a texture atlas instead of a single grass sprite. For example, I can create an atlas from Content/models/grassbb.png and billboardedflowers.png by combining them into a single texture:

This atlas contains two grass types with different sizes in a single texture. If you set this texture to the hair material, you will get this:

It is not good yet, because each particle simply displays the whole atlas. To add smaller parts of the texture to be used, go to the Hair Particle System settings, and use the

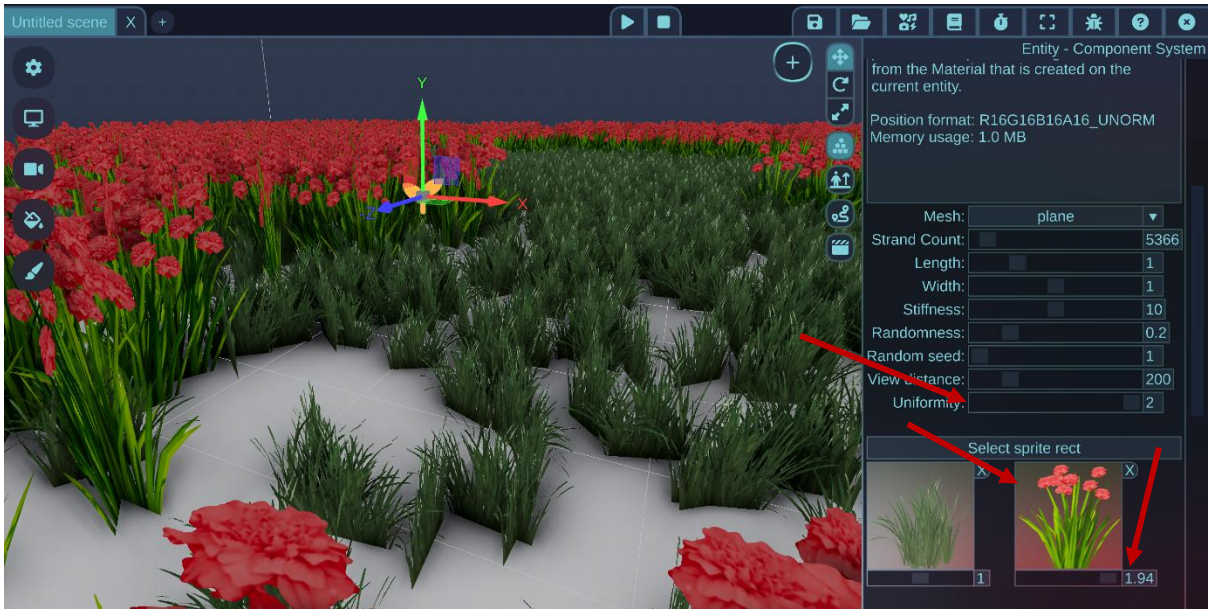
“Select sprite rect” button, which will open a rect editor:



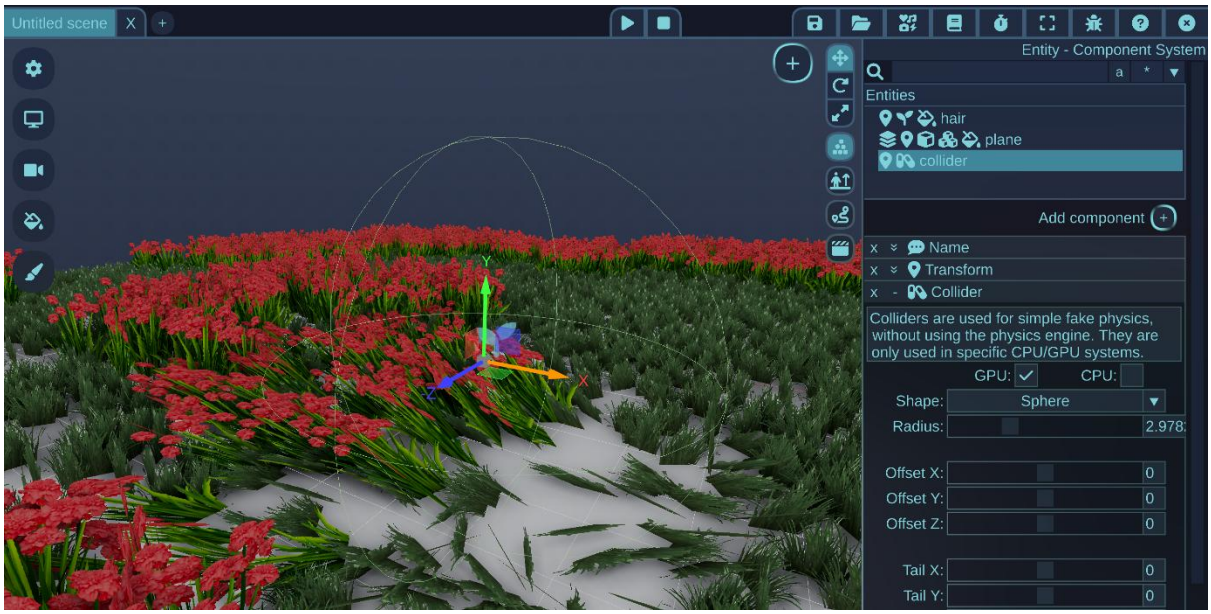
After you select a grass part and click on OK, the particles will change to only include that texture part:

You can repeat this process another time to add the flower texture as well. For each rect, you can go back to adjust it, and set its length with the little slider under it (relative to the overall particle system length). You can also adjust the Uniformity slider to control

the random distribution of the grass types:



The grass will automatically react to forces and GPU colliders. For example, you can add a Collider entity, adjust its shape and set it to GPU. They will push away hair particles:

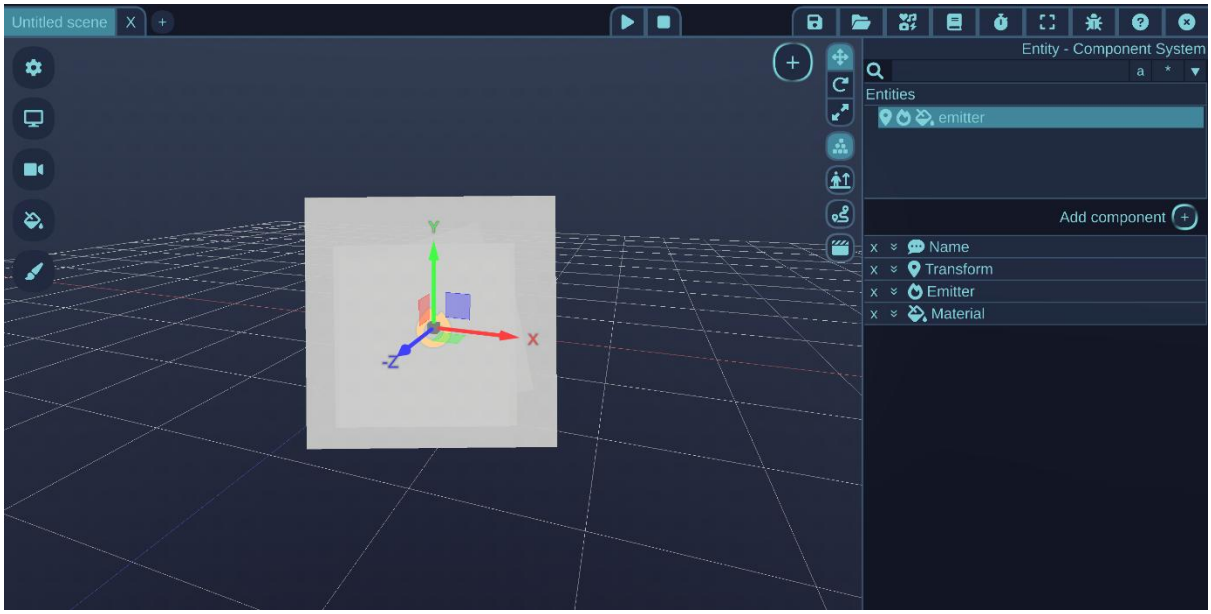


This technique is also used in the Content/scripts/character_controller sample script to let the character move through the grass while pushing it away. The collider in that case is simply parented to the character.

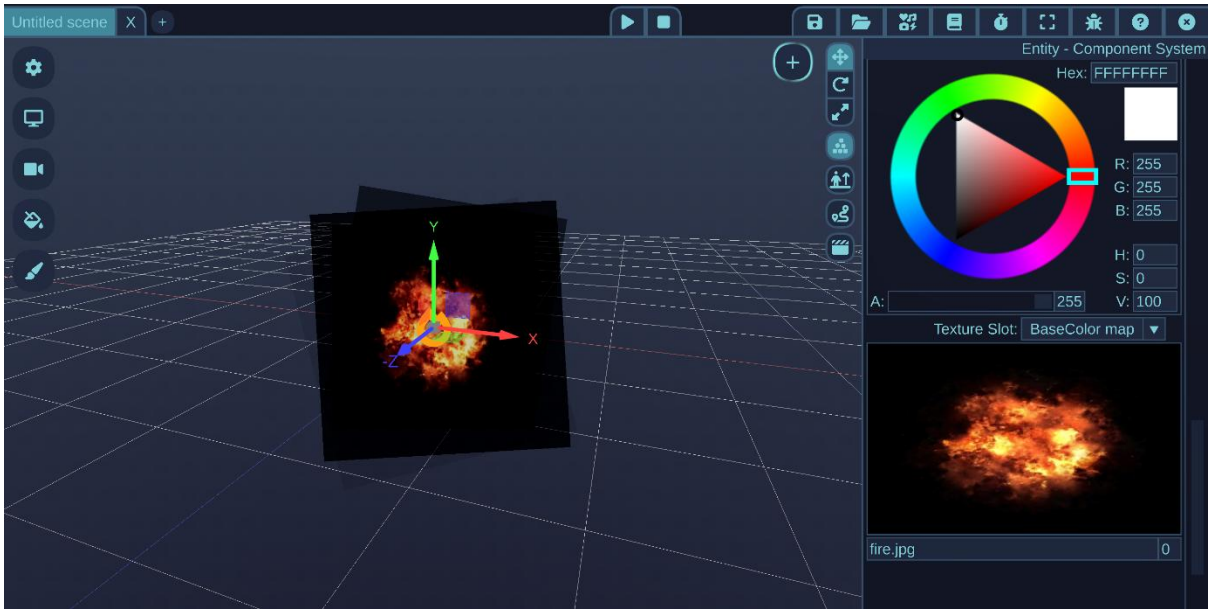
The paint tool can also be used to modify grass, for an example you can see the Terrain chapter in this document where it shows an example on the terrain grass.

Emitters

Emitted particle systems can be used to create floating particles. To create one, simply add an Emitter from the + button. You will get a ready-to use simple emitter which is emitting camera facing quads without texture:

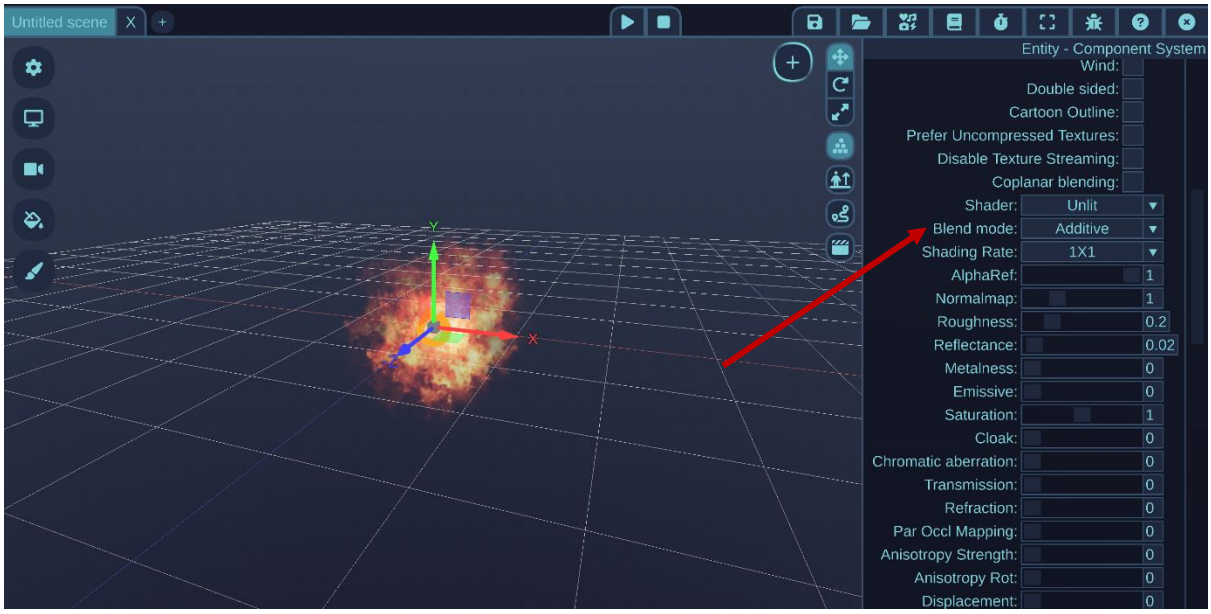


To set their texture, go to the Material settings of the entity and add a texture to the “BaseColor map” slot:

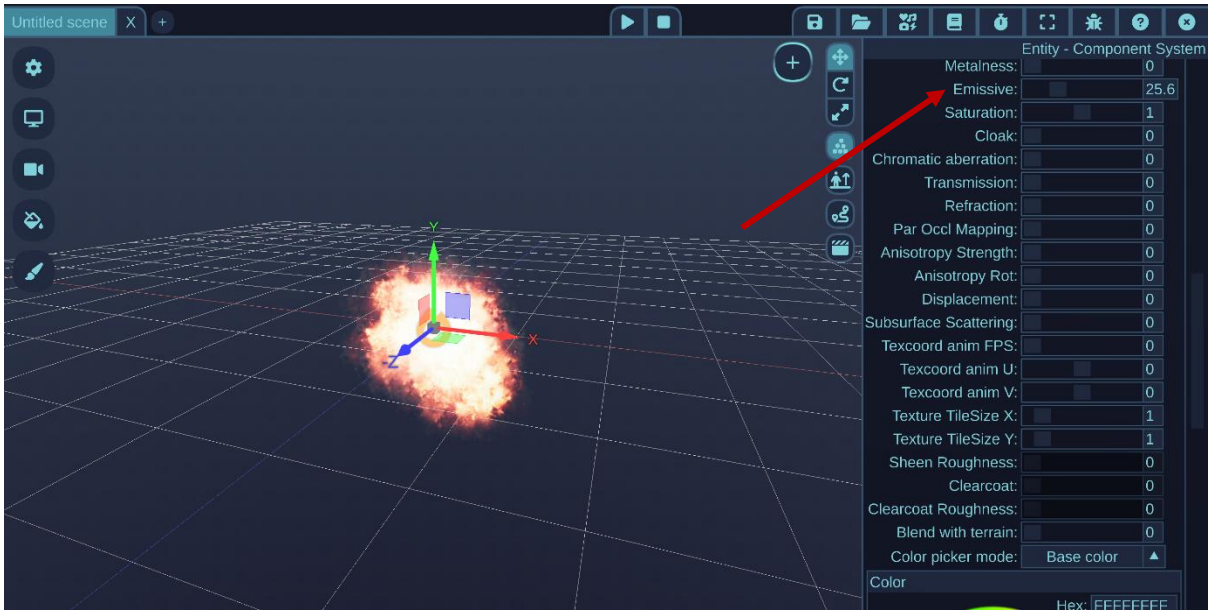


This fire texture would look better with additive blending to remove that black background and if multiple fire particles overlap, it will also become brighter. You can

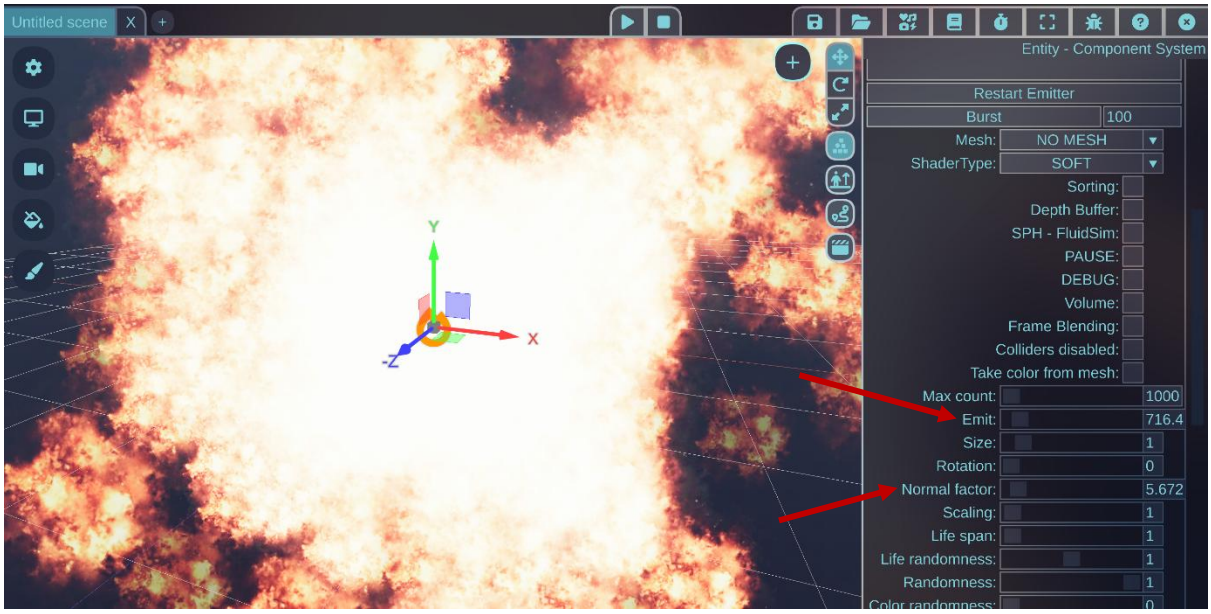
set this in the material's Blend Mode settings, choose "Additive":



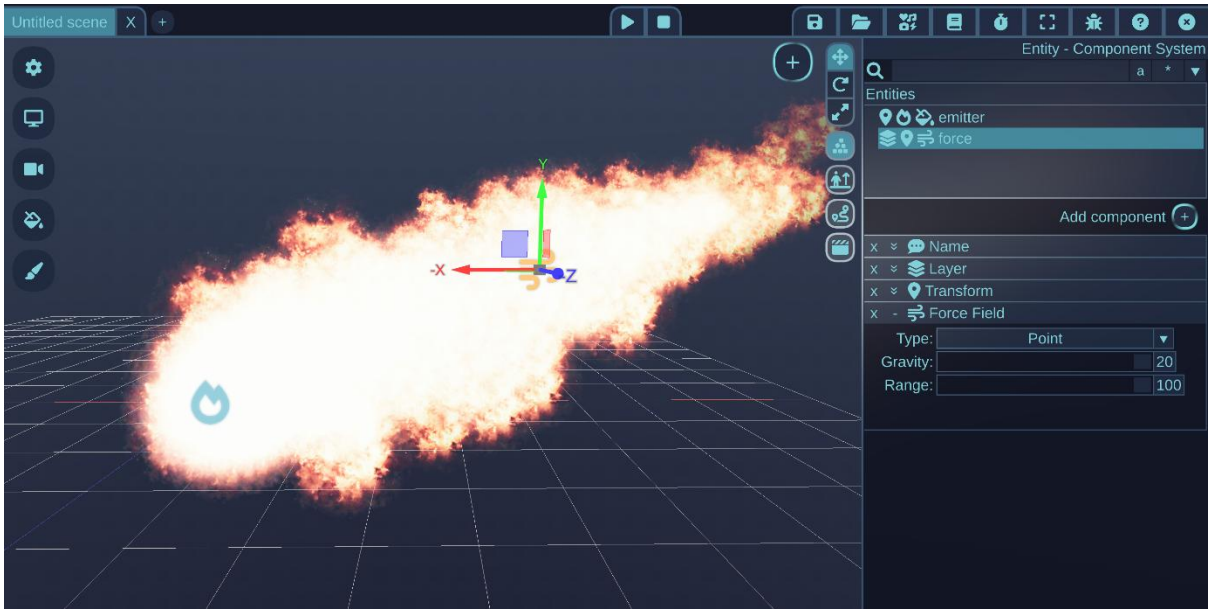
The upside of Additive blending is also that this blending mode is order-independent, meaning that the order of particles is not important for the rendering result. With a non-additive blending mode, you can experience flickering particles, because the GPU can simulate them in non-deterministic order. To fix it, you can simply enable sorting in the Emitter settings. Sorting will result in additional performance penalty. With Additive blending it's also simple to make the fire particles brighter by using the Emissive property:



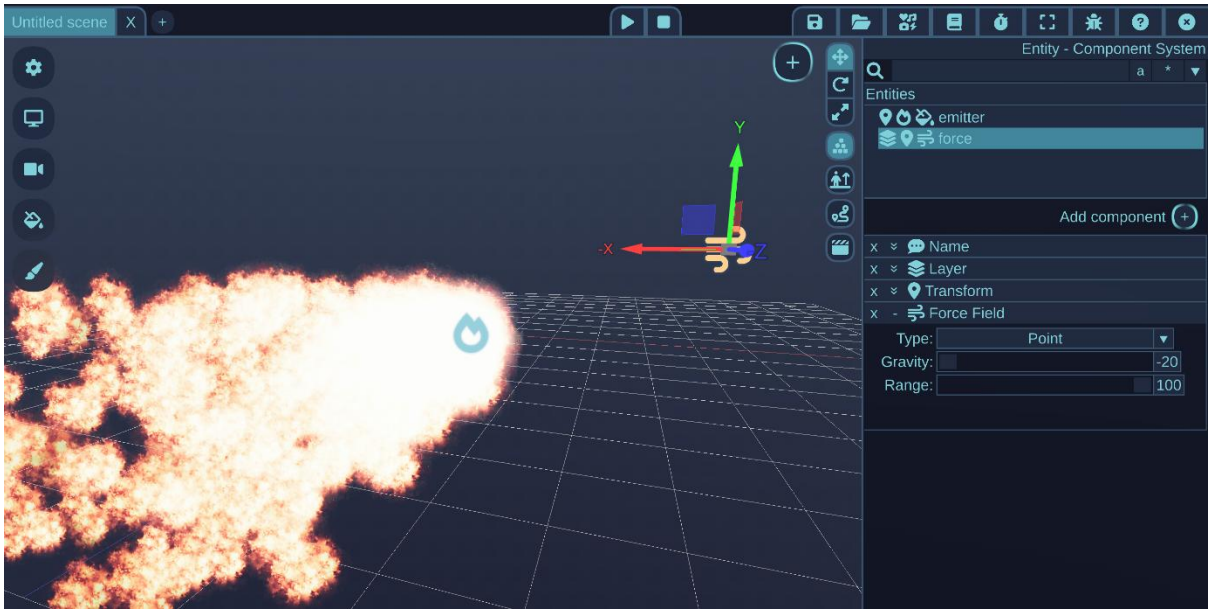
Now let's go back to the Emitter settings and increase the emit count and normal factor to let them fly away from the center:



Forces and GPU colliders will also affect the movement of emitted particles. You saw an example of using GPU collider for hair particle systems, so now let's see an example for using a force. Add a force from the + button and move it away a bit from the emitter. Increase its range and set a positive gravity to it and some range to attract particles:



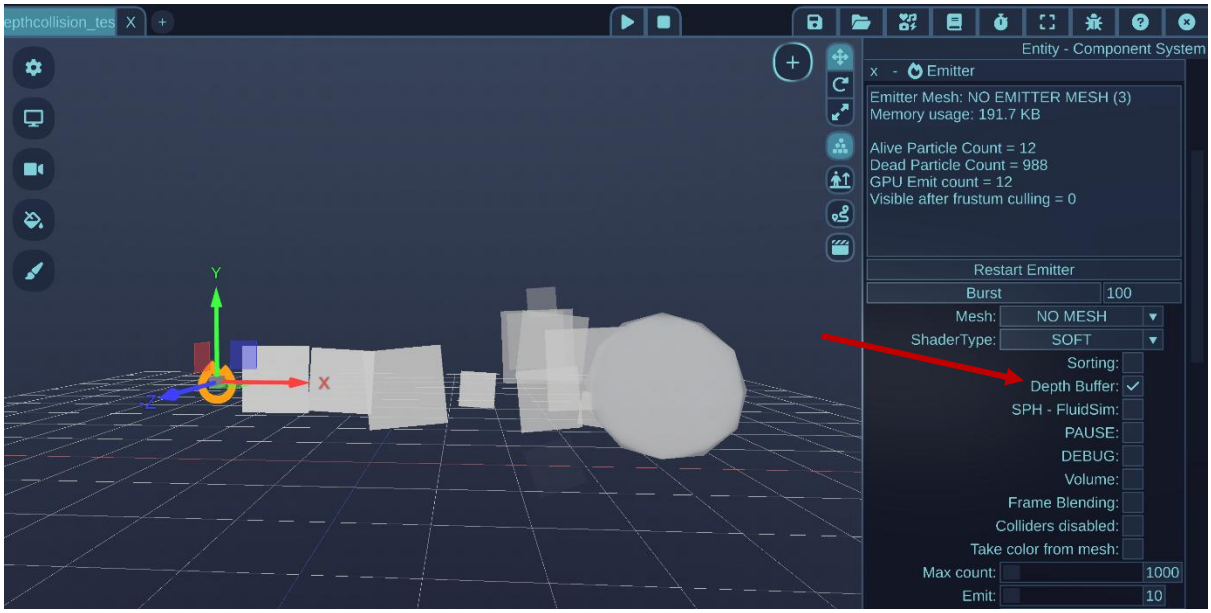
Negative gravity will deflect particles:



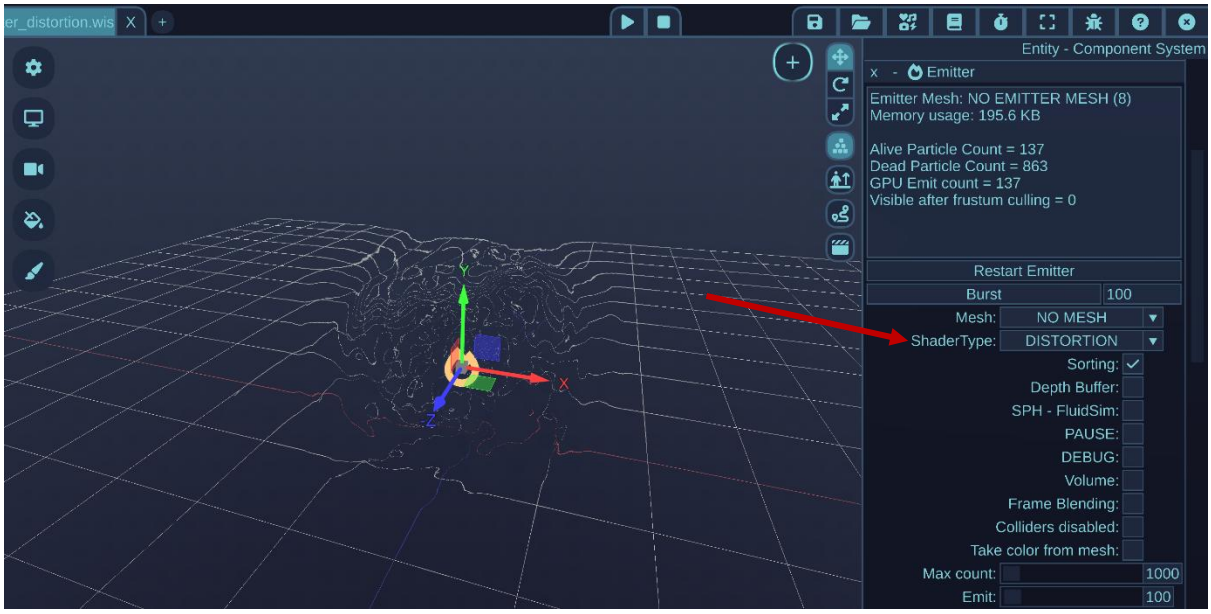
The force field range can be visualized if you turn on the “Force field visualizer” from the general options.

There are more example models for the particles that you can check out.

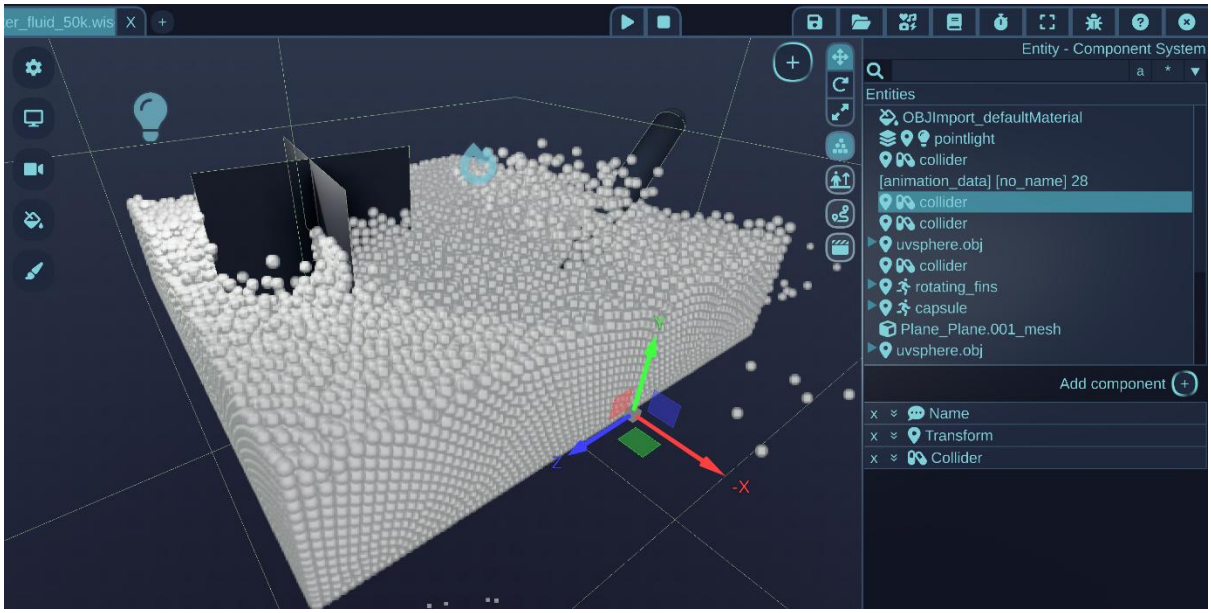
1. Depth collisions can be used for particles to collide with the depth buffer, so opaque object which are on-screen (emitter_depthcollision_test.wiscene):



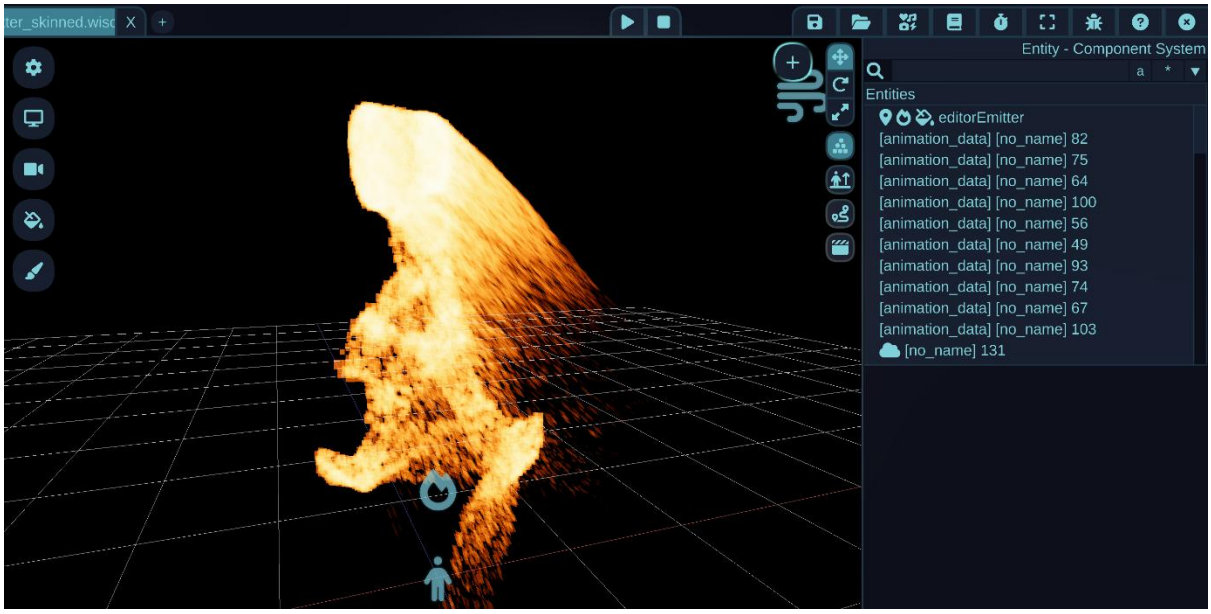
2. Distortion shader can be used to use a normal map in the emitter's material to distort the rendering (emitter_distortion.wiscene):



3. Fluid simulation, showing GPU-based fluid particle physics with colliders (emitter_fluid_50k.wiscene):



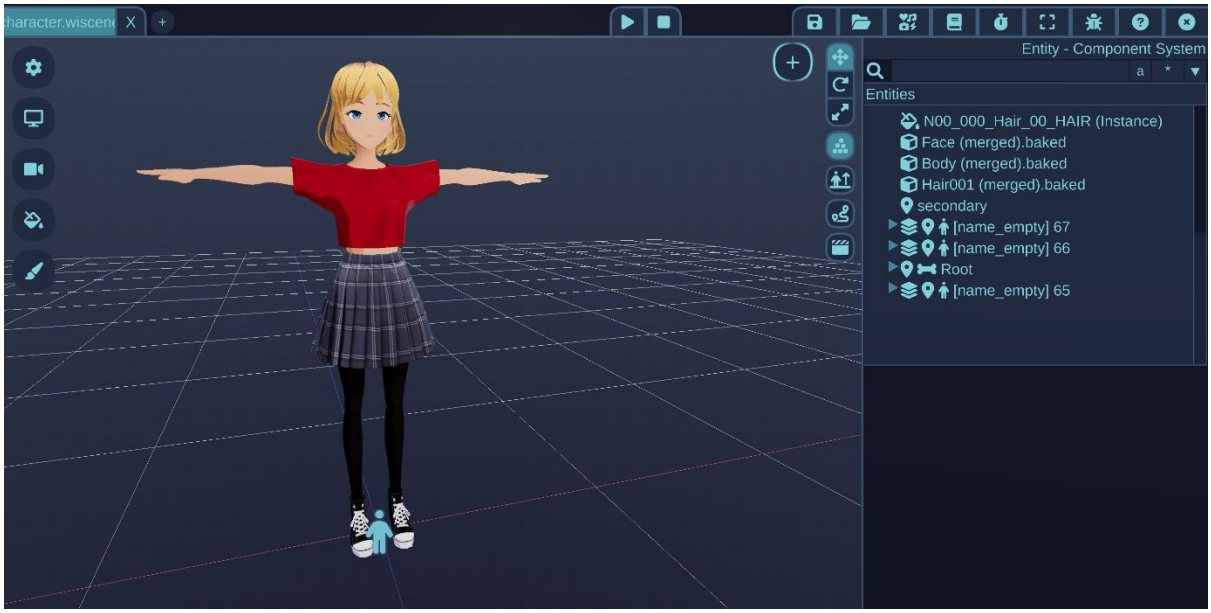
4. Emitting particles from a skinned and animated mesh surface (emitter_skinned.wiscene):



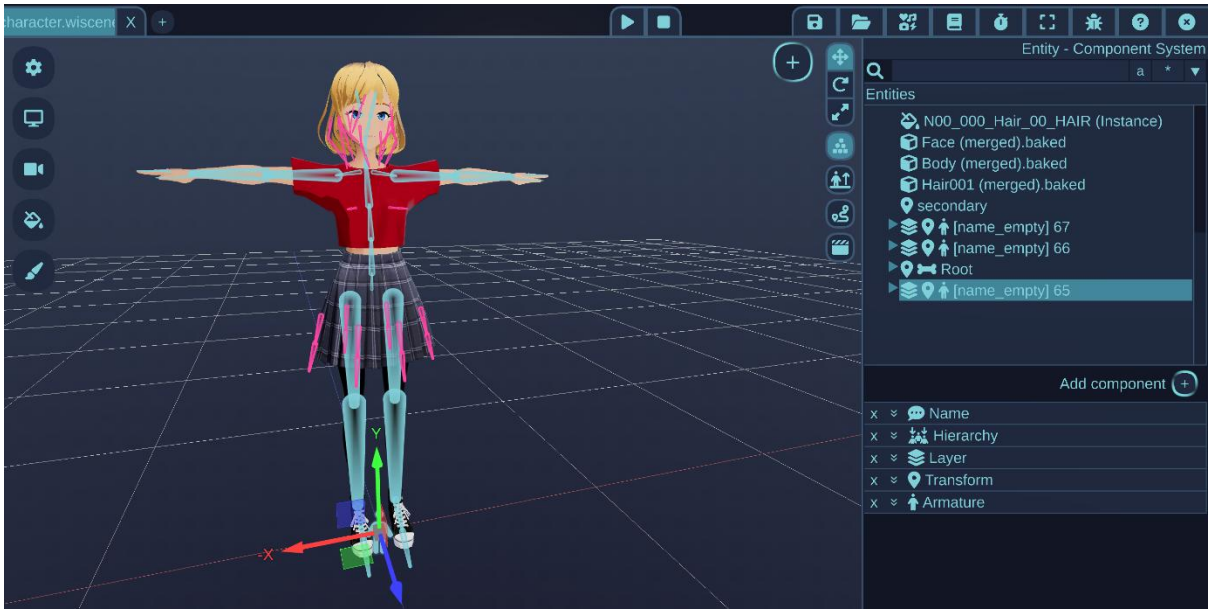
There are a lot of emitter features, most of which are not described in detail in this document yet.

Springs / jiggle bones

When importing VRM characters (which can be created in Vroid Studio), the characters will automatically receive special joints in their skeleton which have spring components. A sample character like this can be found in Content/scripts/character_controller/assets/character.wiscene. Or you can open a VRM model that you created too:

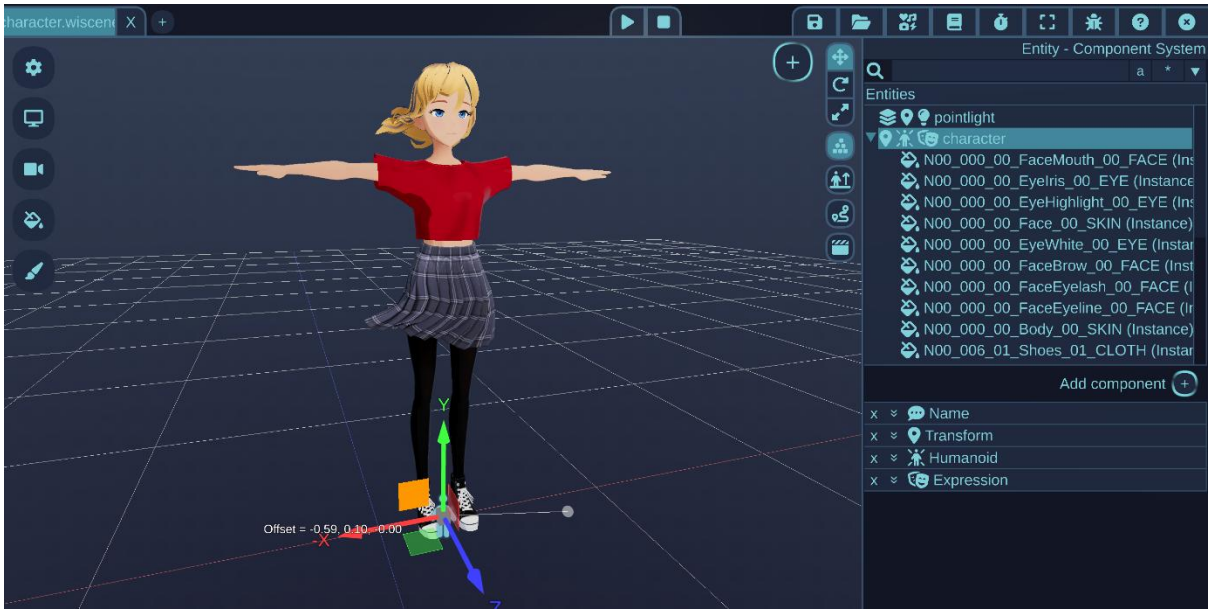


Select the Armature (the little human icon under the character) to view the skeleton:

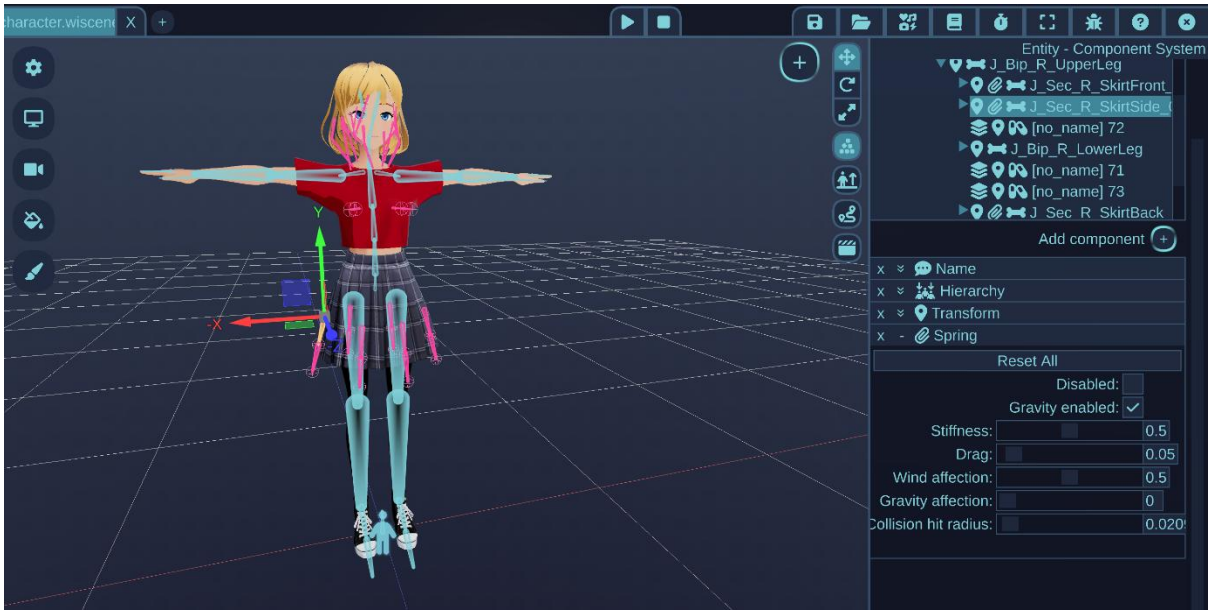


The blue bones are regular bones, and the pink bones are bones with spring components. These are used for jiggle physics and are usually animating the hair and clothes. These bones will not have a keyframe animation usually, but they will add extra procedural animation on top of the animation that was authored. To simply view the effect, grab the whole character model's root entity and move it with the transform tool.

You will notice the skirt and hair of the character will be jiggling with a simplified physics:

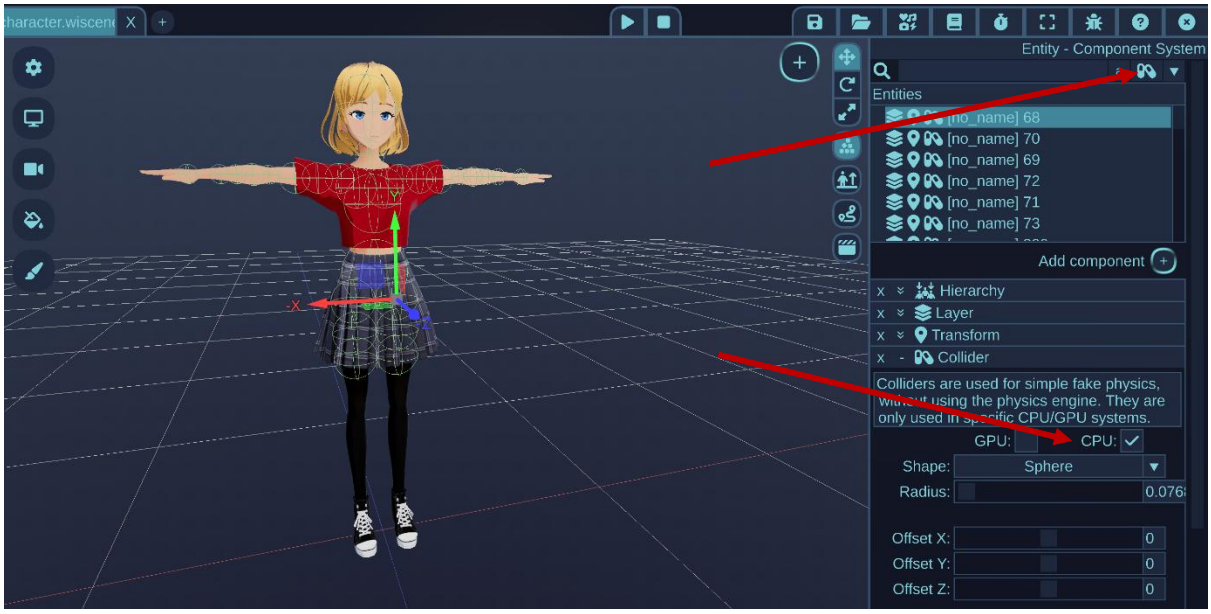


This is not actual physics, as it will also work when you disable physics simulation, rather a simple procedural animation. This also makes them a lot faster and recommended if you don't need the precise physics simulation like cloth physics. Compared to physics simulation, they also cannot stretch and can be more stable. If you select one of the spring bones, you will see the spring settings and additional debug sphere at most springs' tips:

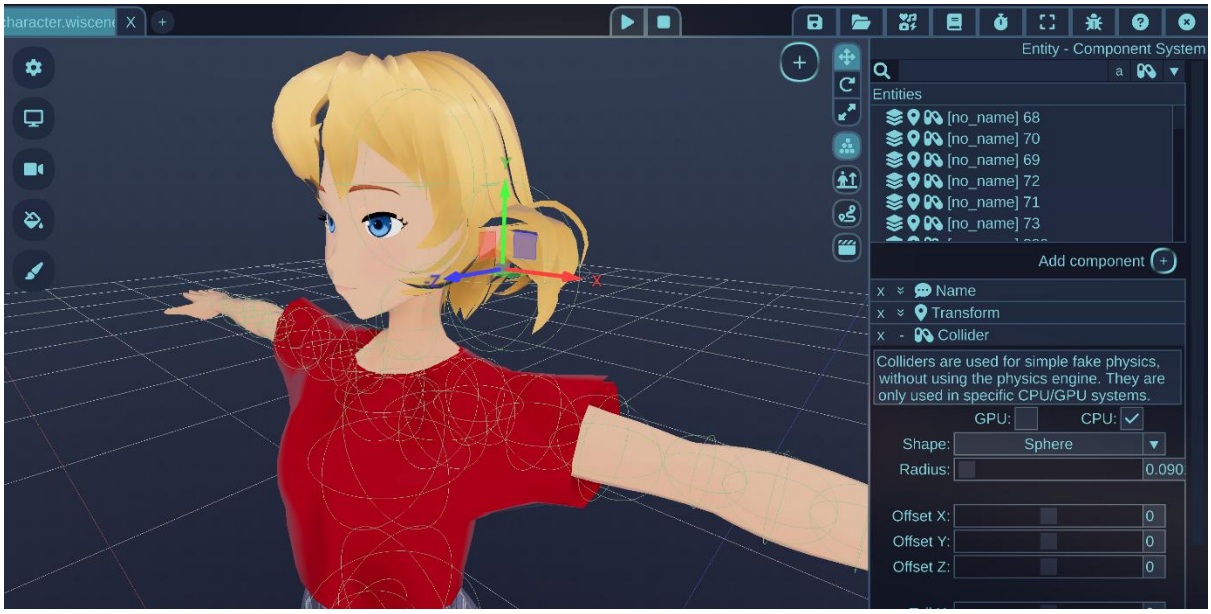


The little sphere corresponds to their collision hit radius. This is used when colliding with CPU colliders. The character itself will also be rigged with some CPU colliders, which helps the hair not go into the head, and the skirt to not go into the thighs when

they are jiggling. You can filter to see only colliders in the Entity panel:



If you also select one of the colliders, they all will be visualized with debug spheres. As you can see, the character consists of several spherical colliders, tagged as “CPU”. You can also create a CPU collider and test spring collision yourself. Add a new collider and set it to CPU, then move it close to for example her hair to collide with it:

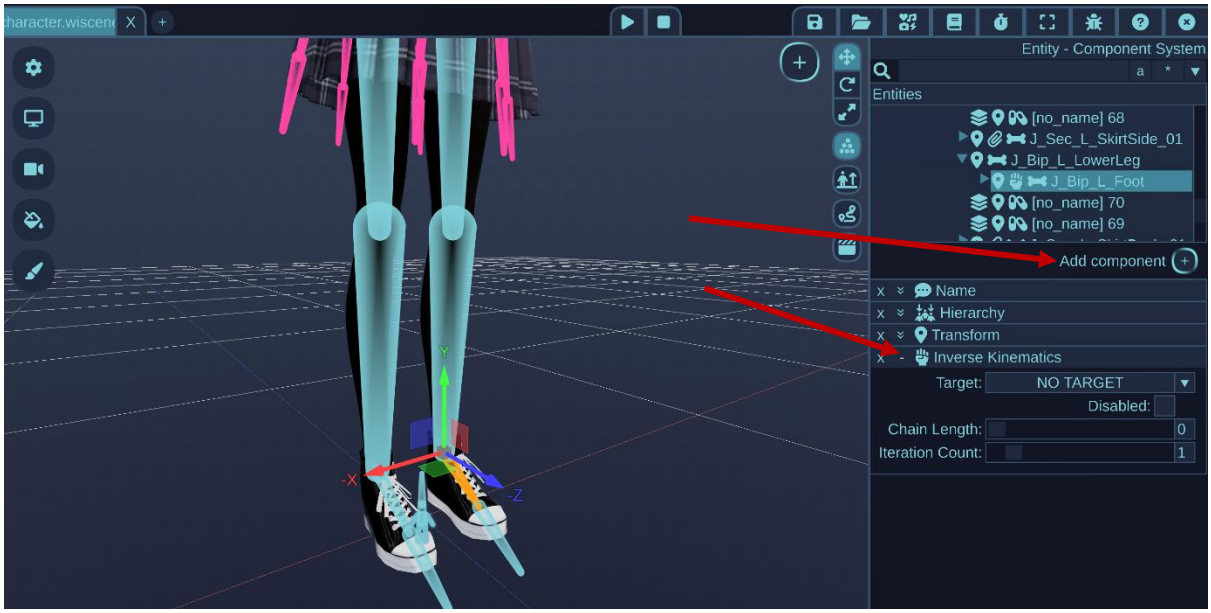


Note again that the collider component is not a physics object, it will only act on these specific systems:

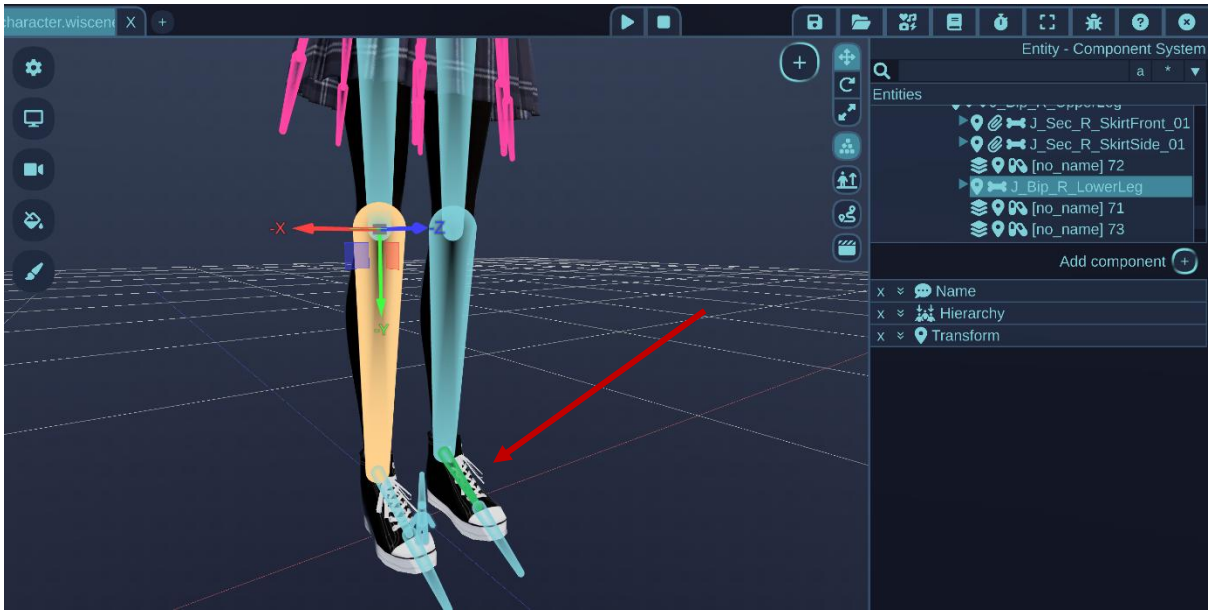
- Emitters
- Hair particles
- Springs
- User scripts

Inverse kinematics

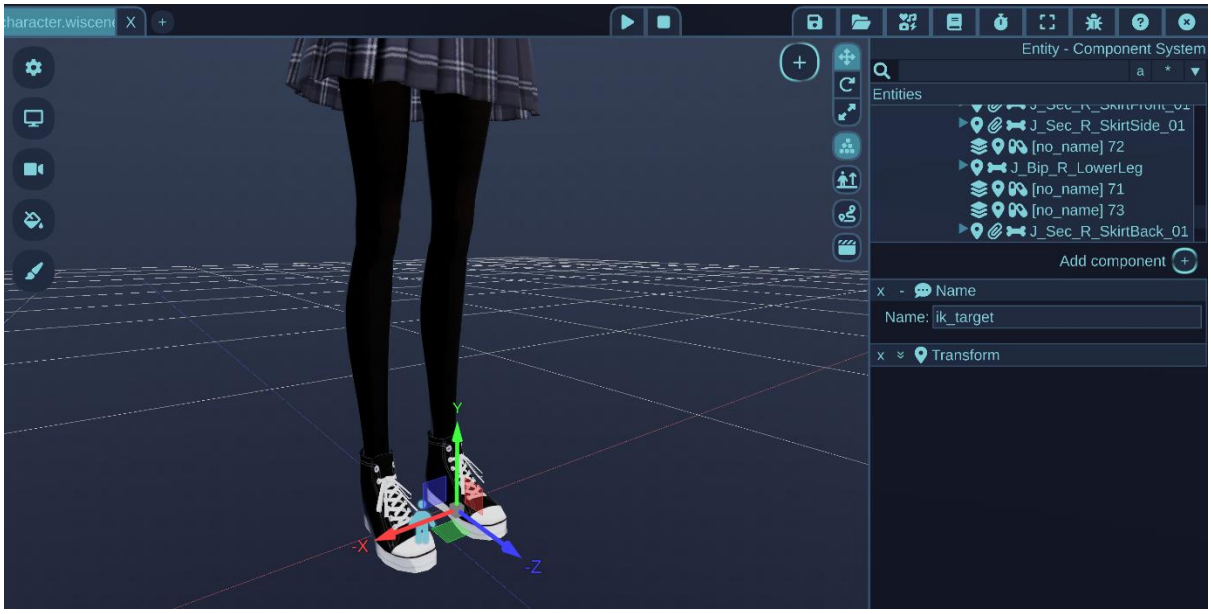
You can create inverse kinematics from within the editor. This is usually done for skeletons, although it's not limited to them. Usually it is done for arms and legs of a character. In Wicked Engine, if you do this for arm or leg bones that are part of a humanoid rig, they can also have additional automatic constraining behaviour. With the previous character, select one of her feet and add an inverse kinematics component to it:



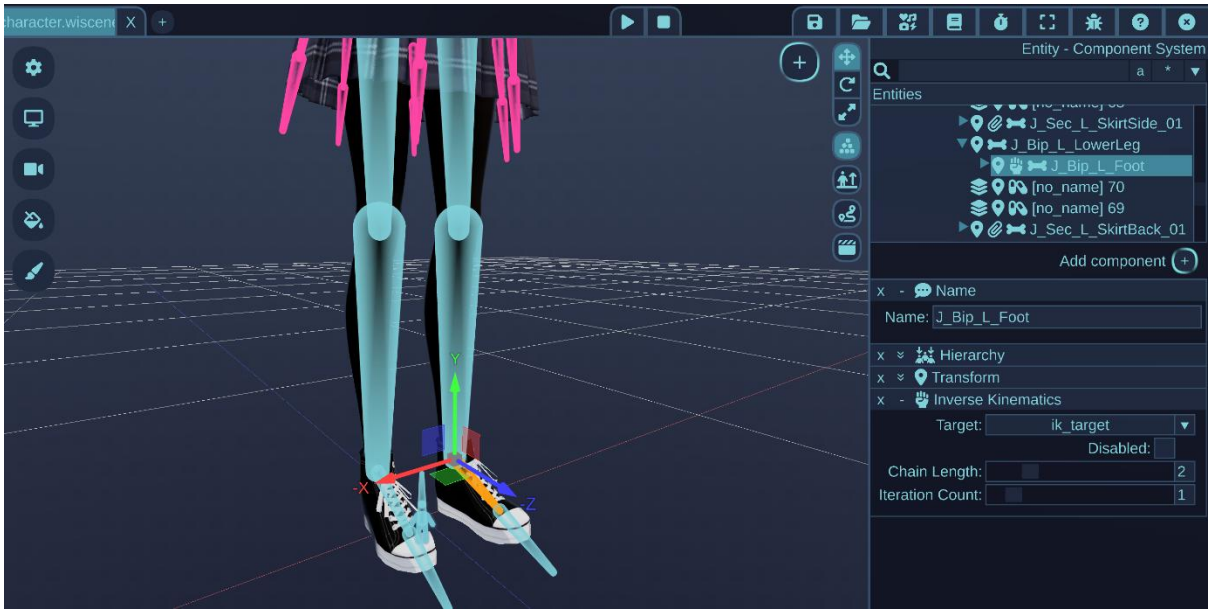
If you deselect a different bone now, you will see that bones with inverse kinematics have a special green color for them to easily differentiate:



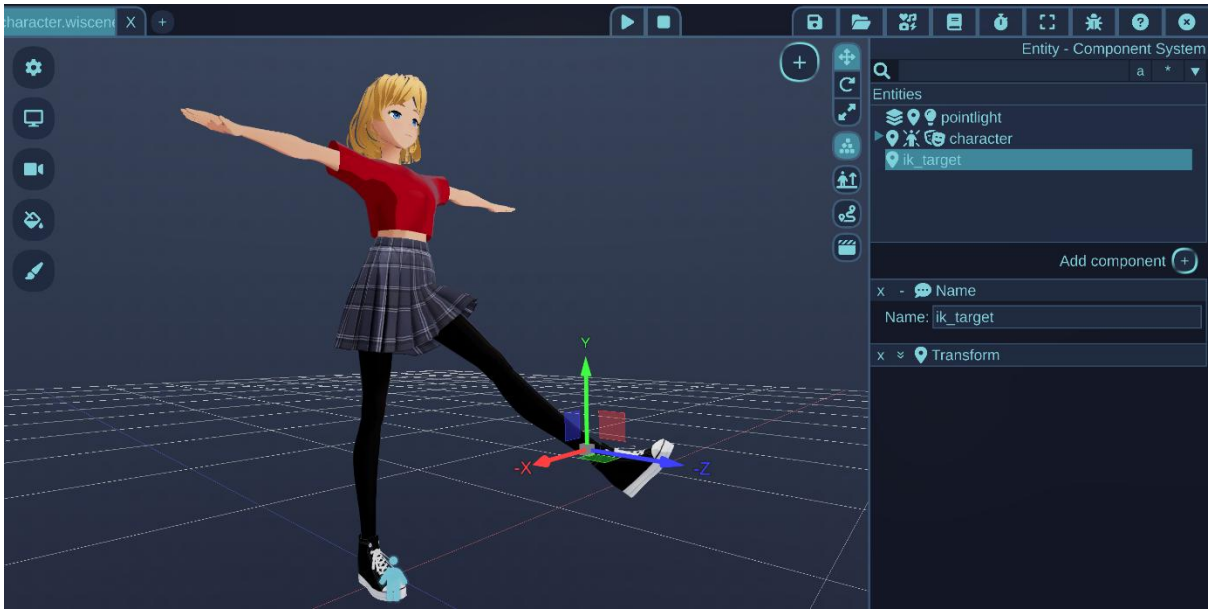
For inverse kinematics we will need something to target, so create an empty transform and move it close to that foot. I also named it ik_target:



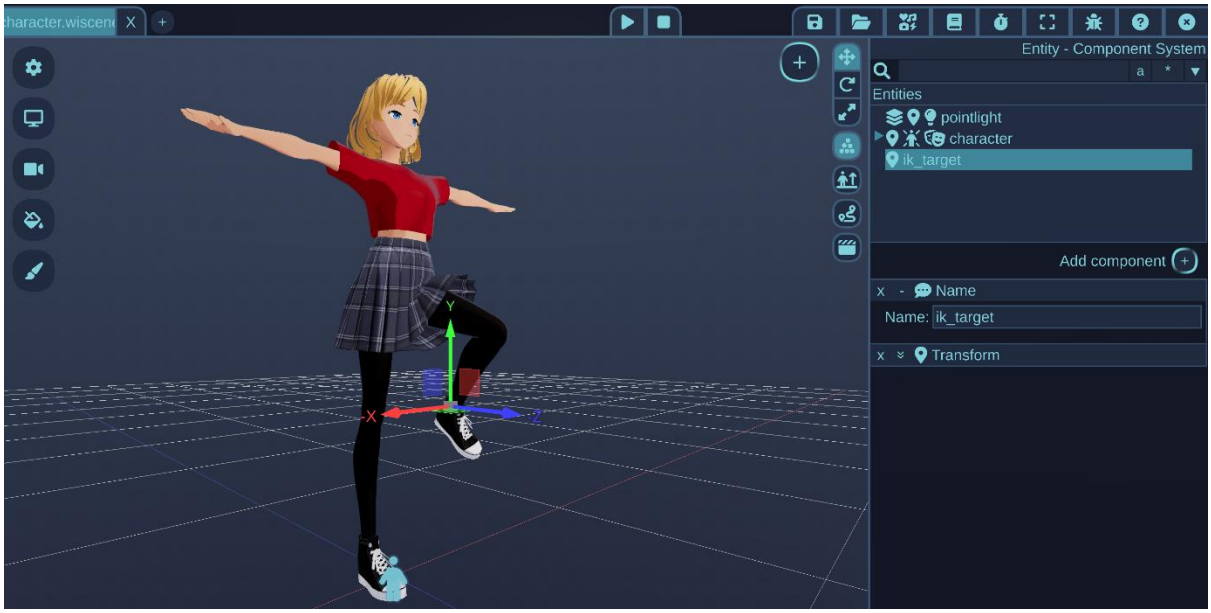
Now select the foot bone with the IK and select the ik_target as the target (will be at the bottom of the list), then increase chain length to 2 (so it includes the shin and thigh bones):



Now if you grab the ik_target transform (select it in the Entity list panel) and move it, the whole leg will follow:



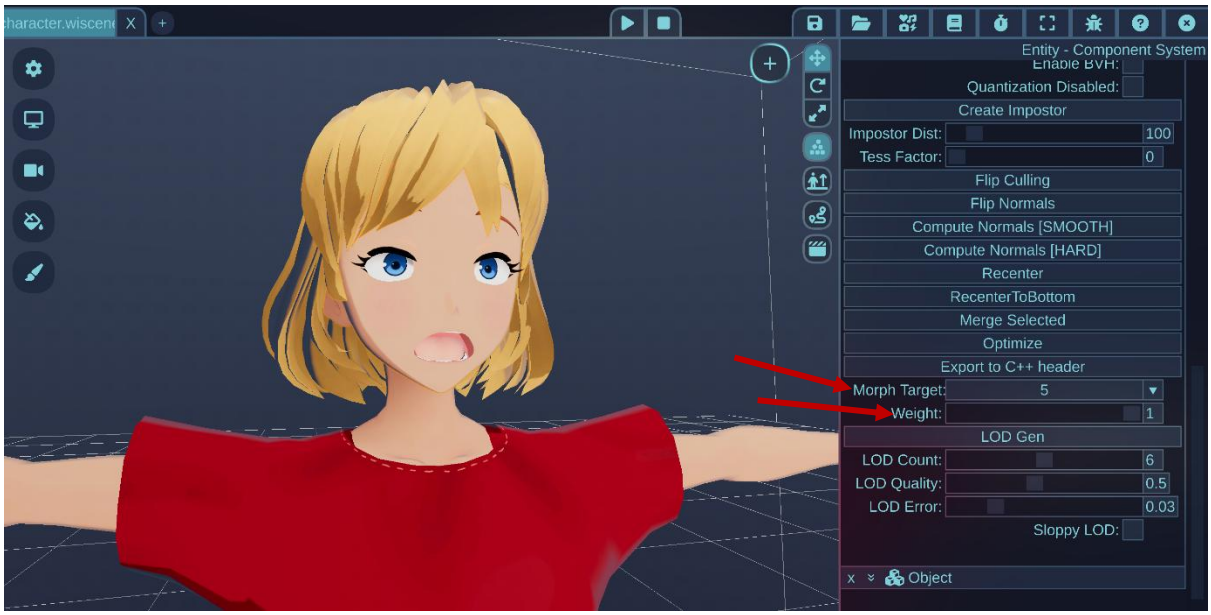
It’s not very accurate right now, because it’s using a low iteration count. Go back to the foot IK and increase the iteration count to 10. The leg will follow the ik_target more accurately and the knee will also bend more easily:



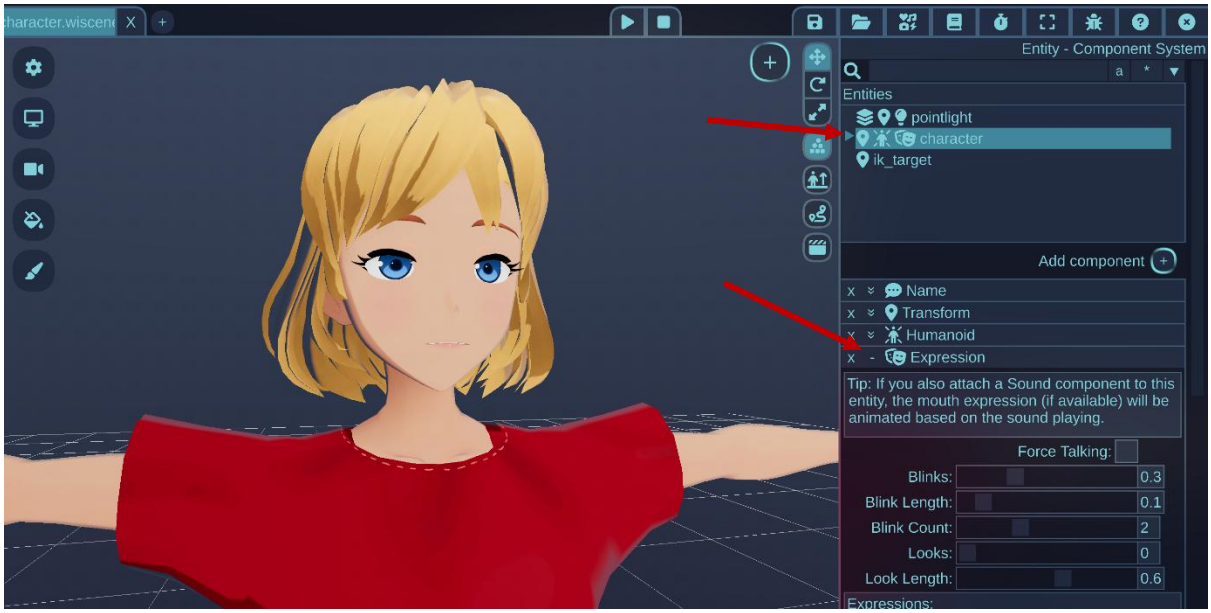
Because the skeleton of this model is part of a humanoid component (VRM and Mixamo models will have this), the engine also ensures that the knees or elbows will not bend backwards.

Expressions, morph targets

Let's stay with this character model as it also has expressions and morph targets that I can show here. Morph targets are a way to have mesh deformation without bones, using vertex deformation poses, they can be controlled from the mesh. Expressions are a collection of morph targets that are operated by a single value, they can be controlled by the Expression component. First let's see morph targets. Select the face of the character by clicking on it, then find the morph targets and select [5] then increase its weight to 1:

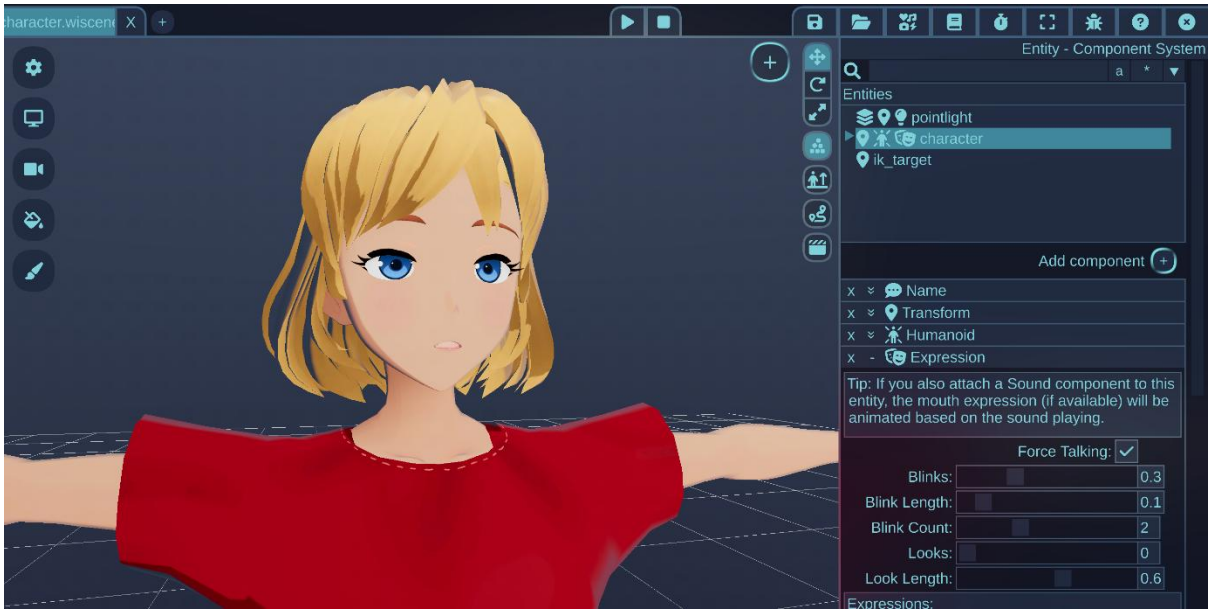


You see the effect, that the face mesh got deformed into a surprised pose. Now reset the morph target and find the expression component on the character's base entity:

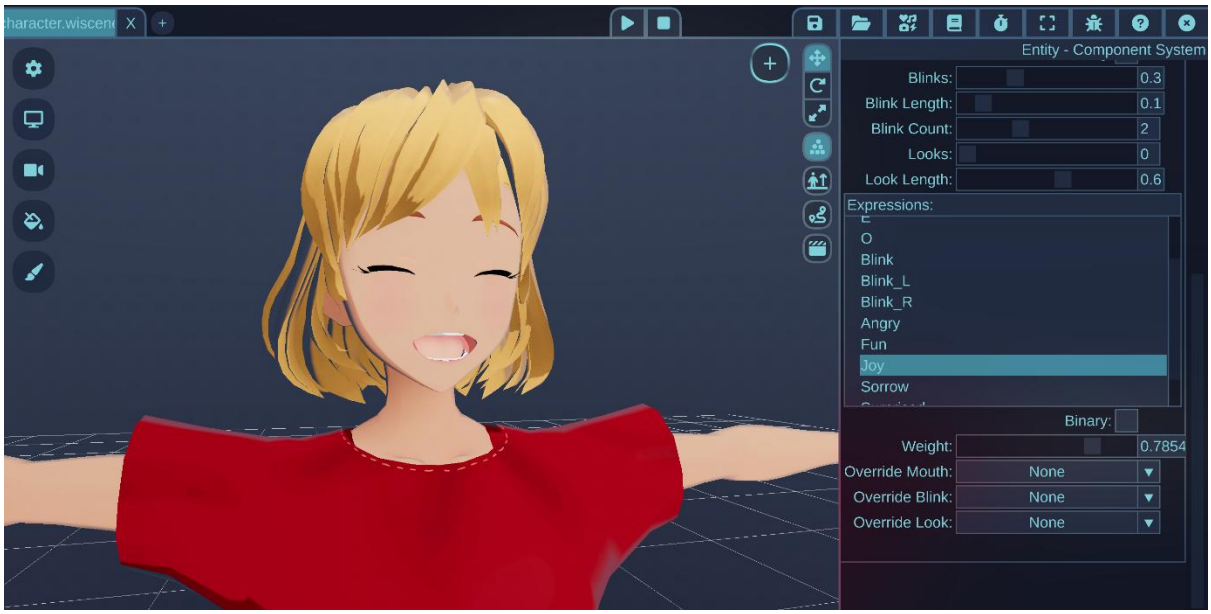


The expression is a nicer way to control a collection of morph targets that have predefined functionality. Here you can specify pre-built behaviours, like blinking and

talking. Try to enable force talking, the character will start to articulate randomly:



Turn it off, and check one of the preset expressions, like fun and increase its weight:



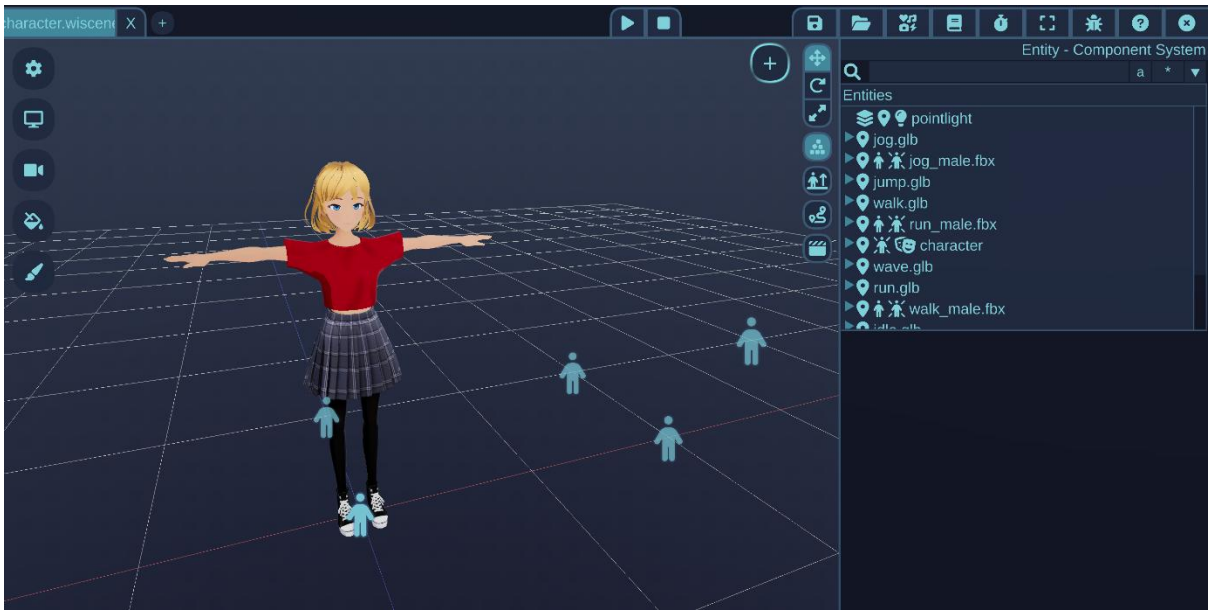
This expression will control multiple morph targets to create a joyful expression, as you can see the face and mouth are both affected. In this specific expression it's also useful to override the automatic blinking, because it's weird to blink when the eyes are already closed. For that you can set "Override blink" to "Block".

VRM models created in Vroid Studio will always have these built-in expressions that you can control. For example on how to control them from a script, check out the Content/assets/character_controller script.

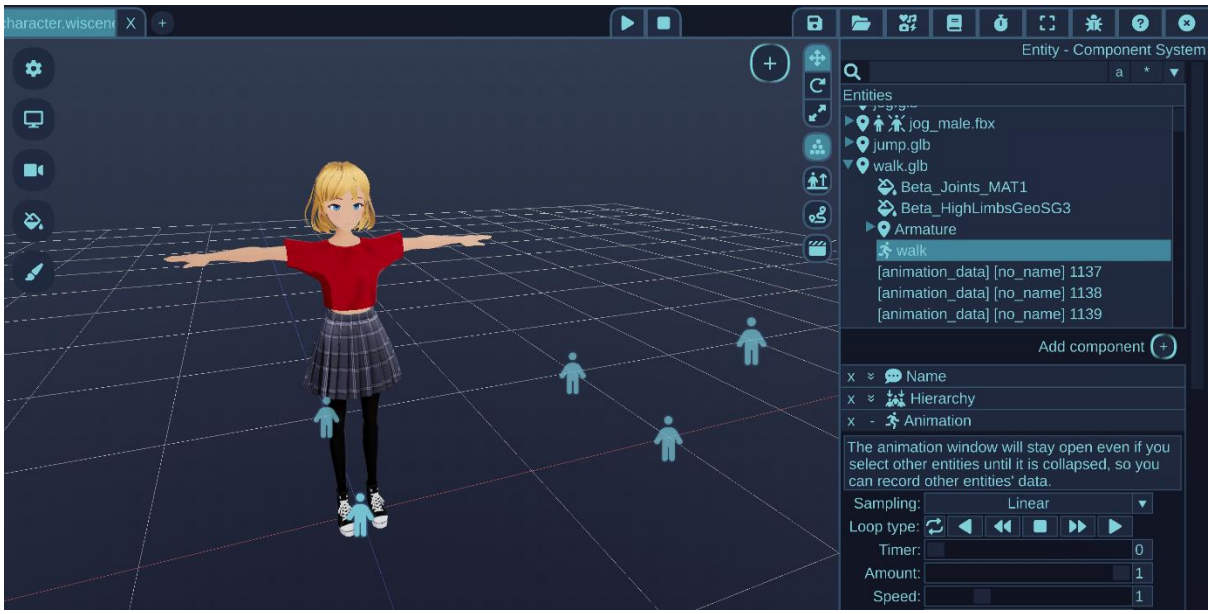
Animation retargeting

To apply animations on a character, both the animation and the character must be acting on a humanoid rig. Humanoid animations can easily be downloaded from Mixamo.com as FBX models. Humanoid characters can be downloaded from Mixamo.com as FBX models, or created in Vroid Studio as VRM models. Let's find two sample models and open them both in the Editor:

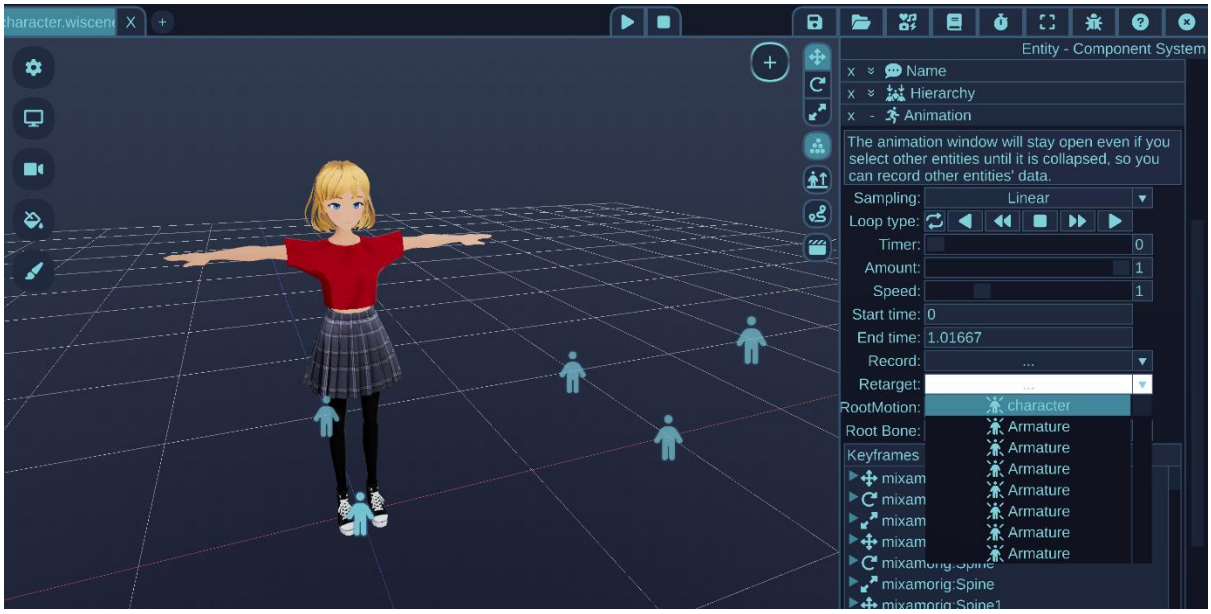
- Content/scripts/character_controller/assets/animations.wiscene
- Content/scripts/character_controller/assets/character.wiscene



The character model and several animations are in the scene now. You can also turn on skeleton visualizing for this, or just leave it like this. The character doesn't have any animation of its own. Select the walk.glb/walk animation:



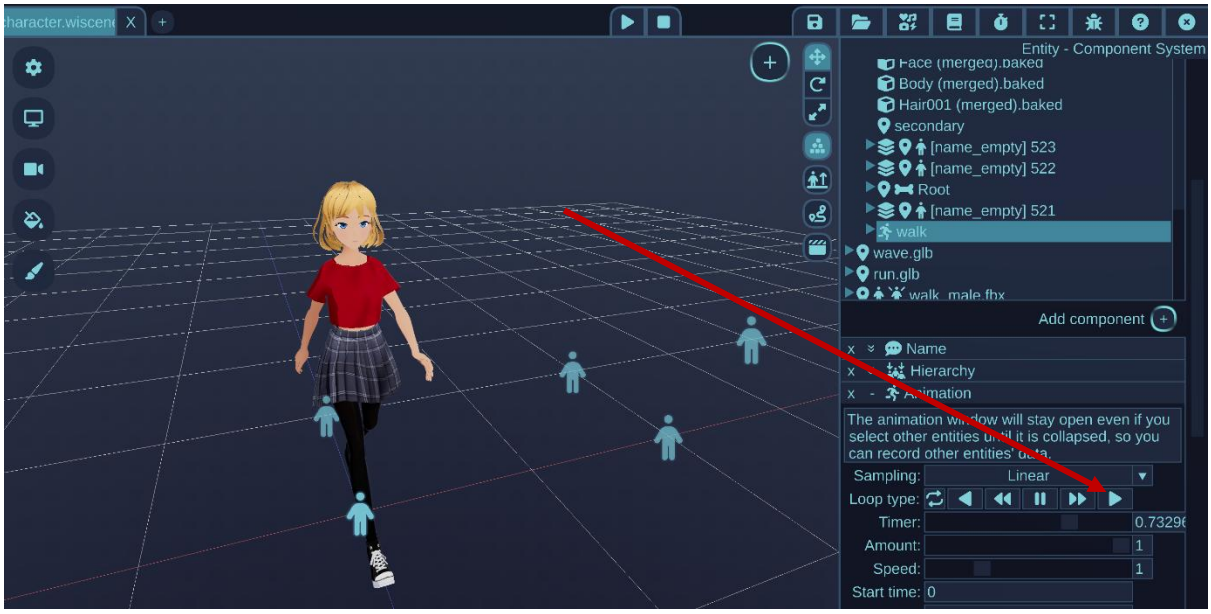
Scroll down to the Retarget option inside the Animation settings, and choose “character”:



Seemingly nothing happens when you click on it, but now you can go to the character entity and open it to find the walk animation inside:


Entity - Component System
walk.glb
run_male.fbx
character
N00_000_00_FaceMouth_00_FACE (Ins
N00_000_00_EyeIris_00_EYE (Instance
N00_000_00_EyeHighlight_00_EYE (Ins
N00_000_00_Face_00_SKIN (Instance)
N00_000_00_EyeWhite_00_EYE (Instar
N00_000_00_FaceBrow_00_FACE (Inst
N00_000_00_FaceEyelash_00_FACE (I
N00_000_00_FaceEyeline_00_FACE (Ir
N00_000_00_Body_00_SKIN (Instance)
N00_006_01_Shoes_01_CLOTH (Instar
N00_000_00_HairBack_00_HAIR (Insta
N00_001_03_Bottoms_01_CLOTH (Inst
N00_003_01_Tops_01_CLOTH (Instanc
N00_000_Hair_00_HAIR (Instance)
Face (merged).baked
Add component
Name
Transform
Animation
The animation window will stay open even if you select other entities until it is collapsed, so you can record other entities' data.
Entity - Component System
Face (merged).baked
Body (merged).baked
Hair001 (merged).baked
secondary
[name_empty] 523
[name_empty] 522
Root
[name_empty] 521
walk
wave.glb
run.glb
walk_male.fbx
Add component
Name
Hierarchy
Animation
The animation window will stay open even if you select other entities until it is collapsed, so you can record other entities' data.
Sampling: Linear
Loop type:
Timer: 0
Amount: 1
Speed: 1
Start time: 0

You can start the animation to check that it works correctly:



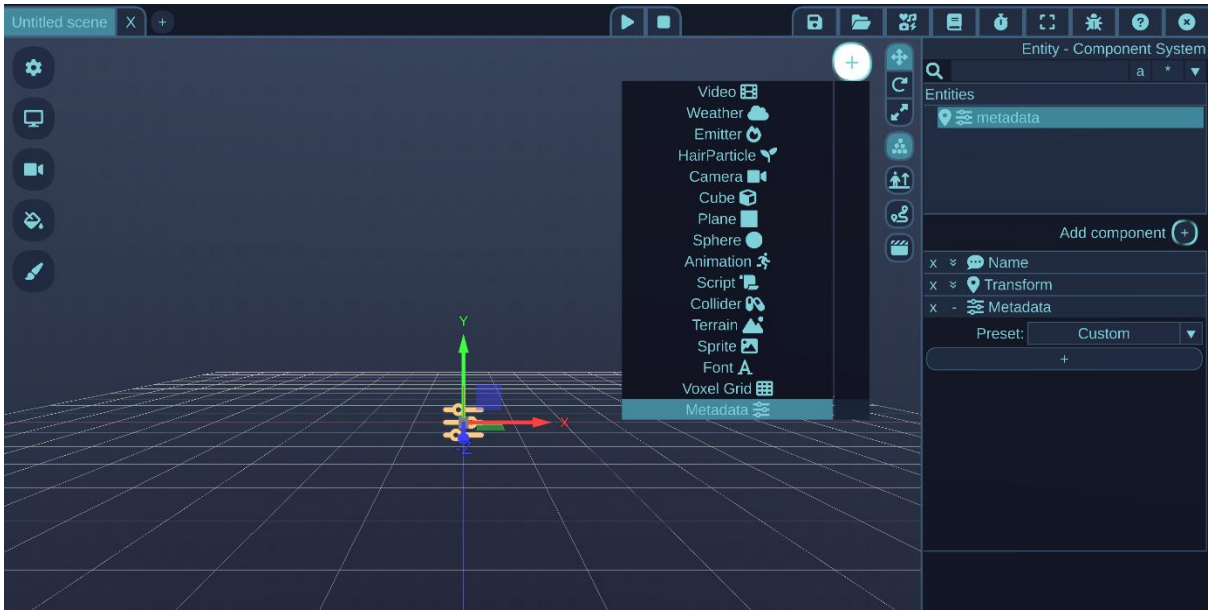
This is a good way to check how animation retargeting would work on a character inside the editor. But this kind of retargeting creates additional animation data stored inside the character which can take up additional memory. In a gameplay script, you can also specify the retargeting to do “runtime” retargeting, which doesn’t make copy of the animation data, but the character can instead reuse animation from a separate scene.

Thus with runtime retargeting it is not required to put the animations inside our main scene, and instead reuse the animations for multiple characters with not much additional memory usage. That method will not work however if you want to save a character with embedded animations with it, that is why the Editor chooses this approach instead.

For runtime retargeting, check out the Content/scripts/character_controller script.

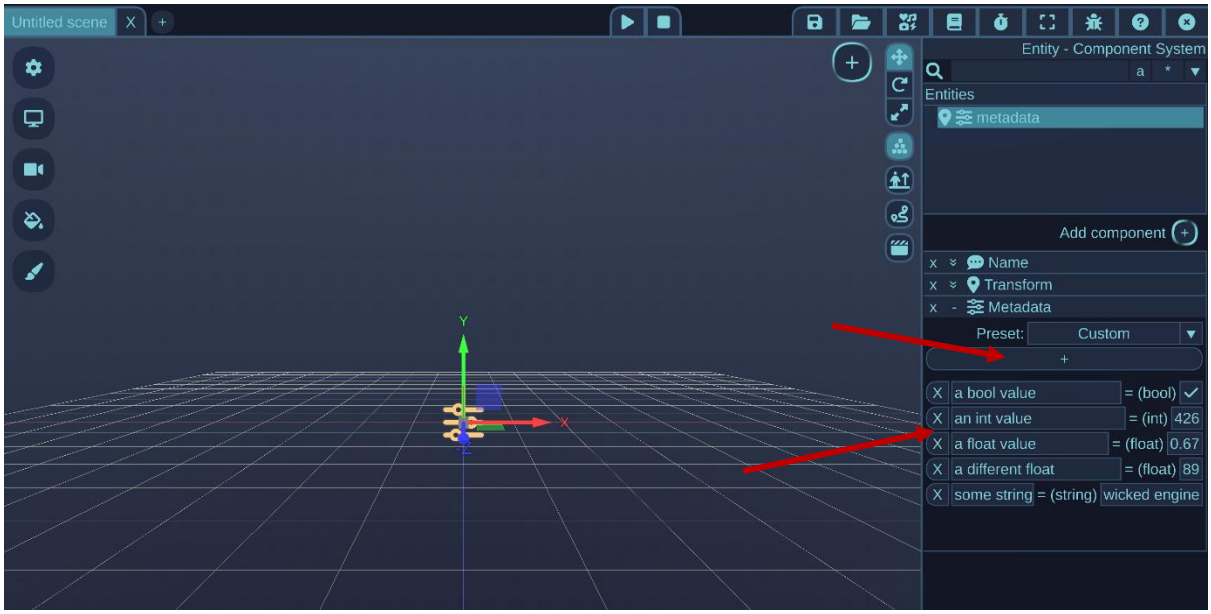
Metadata

Metadata components can be created and placed in the scene to help identifying entities visually, and attach custom values that can be retrieved easily from gameplay code. You can add one simply from the + button menu:



There is nothing in it at start, but you can add custom properties to it with different types, that can be now:

- Bool
- Int
- Float
- String



The engine will not do anything with these, it's completely up to you what you want to do with them. For example the character controller sample uses custom properties to define a character's name which will determine the character model to load.

It is also possible to set presets for this, which will just be an enum for the gameplay code, but the editor has specific visualizers for them to help you place them to the scene:

- Waypoint
- Player
- Enemy
- NPC
- Pickup
- Possibly more in the future...

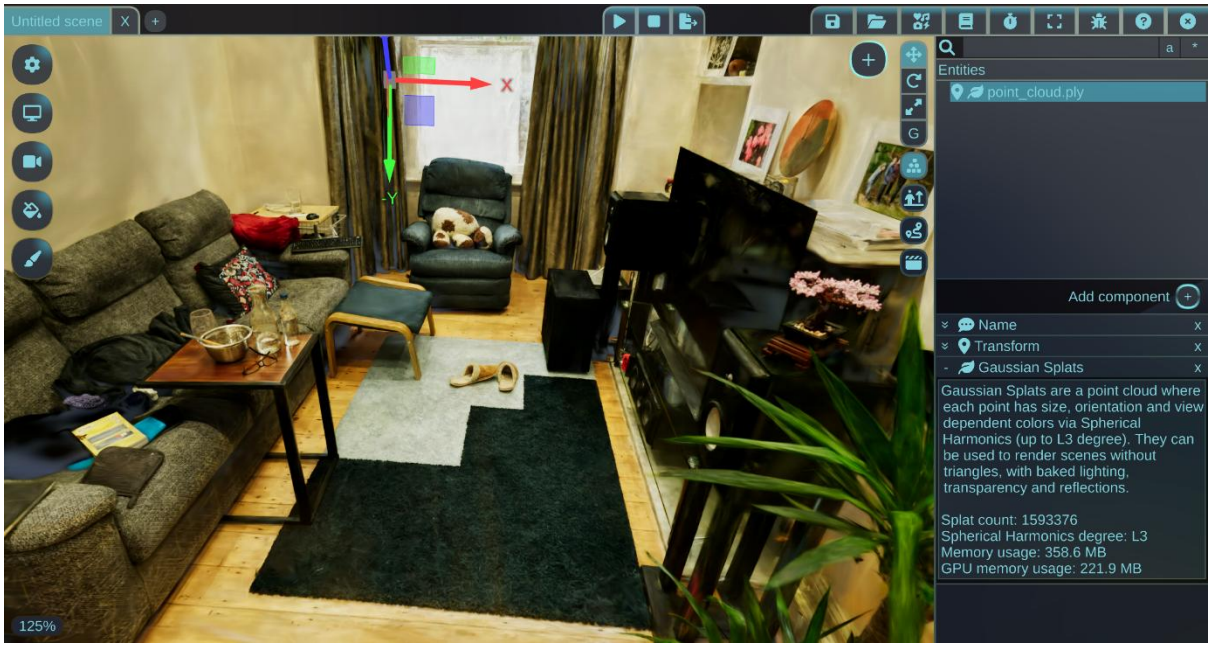
The character controller sample determines what kind of game object to put in the place of the metadata components placed across the level based on its preset value. If a preset value is not available that you would like, it is always possible to use a custom property instead.

Gaussian Splatting

Gaussian splat models can be imported from .PLY files by simply drag and dropping them into the Editor window or using the Open file browser as regular models. This is what a Gaussian Splat model might look like:

This model is part of the reference implementation of Gaussian Splatting. [dowload sample models]

A gaussian splat model can be one huge point cloud data, with each point having size, orientation, and view dependent colors, ad containing multiple objects. It is rendered differently compared to regular triangle meshes, and they support effects like transparency, reflections and realistic static lighting.



Closing

Thank you for reading the documentation for Wicked Engine Editor. At this time, it is meant to show some features that are commonly used and questioned, but it is not a complete explanation of all the editor's features. The documentation will continue to be updated.