SDF Sculptor

Atler Gibby

Table of Contents

1. Requirements

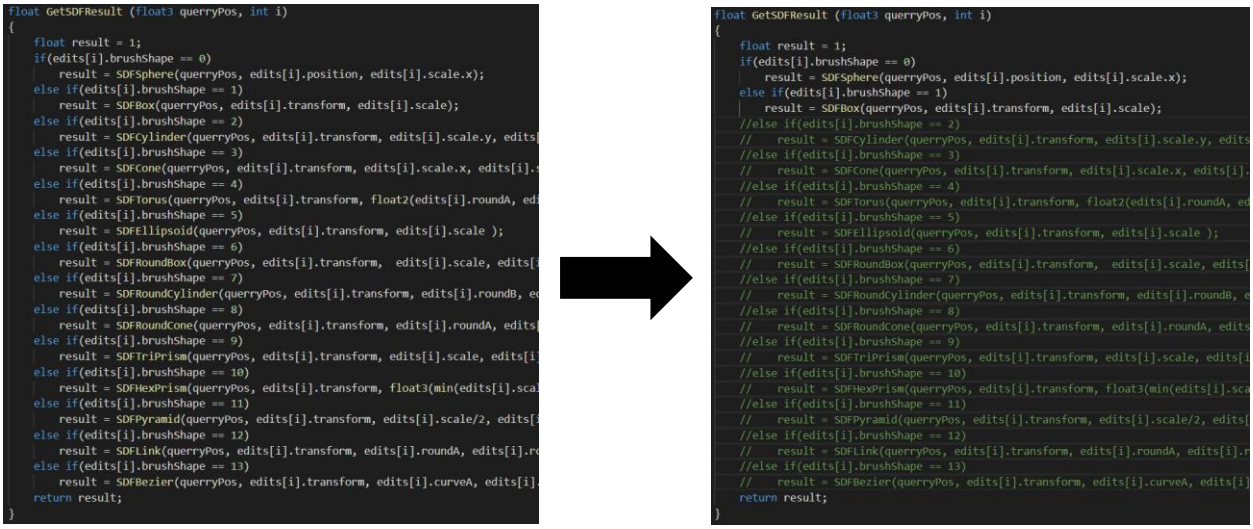
SDF Sculptor was originally made on Unity 2022.3.7f1 for the URP and HDRP.

SDF Sculptor makes use of compute shaders for generating sculpts, doing frustum / occlusion culling, and for determine the screen coverage of geometry for LOD swapping. Therefore, it is recommended that you build for platforms that support Compute Shaders. Generating a mesh from a signed distance field is also expensive on its own and having a dedicated GPU is highly recommended. Also, *SDF Sculptor* has been tested on Windows / Direct3D11 and Non-Windows platforms will not receive the same support.

2. Compute Shader Compilation

This section is more of a warning. Upon downloading the *SDF Sculptor* asset, Unity will compile the SDF Compute Shader (as well as other Compute Shaders). This process will take a long time and you are likely to get a Time Out error after about 10 minutes. You may need to set the environment variable: UNITY_SHADER_COMPILER_TASK_TIMEOUT_MINUTES to something like 20 minutes. You may also get the X4174 warning about not having enough temp registers. You will still be able to use *SDF Sculptor* and this warning will only show up ounce: the first time you generate a mesh.

The issues above are a result of the complexity of the SDF Compute Shader / generating and querying the signed distance field, so this is something that cannot be helped. After the first time you compile the compute shaders though, you will not need to do it again. However, if you wish to expand upon the compute shader / test new SDF shapes, the best way would be to comment out portions of the compute shader where it checks if brushes are different shapes (A sculpt being composed of many brushes/edits). You could leave spheres and cubes and it will compile a lot faster for you if you want to test something.



3. Basic Setup

The basic setup for a scene using *SDF Sculptor* is the same as the set up for the Sculpting Demo Scene:

1. Go to **SDFSculptor > Prefabs** and drag and drop the *SculptSceneManager* into your scene.
2. Go to **SDFSculptor > Prefabs** and drag and drop the *MeshGenerator* into your scene.
3. Go to **SDFSculptor > Prefabs** and drag and drop the *SDFPlayer* into your scene.
4. Go to **SDFSculptor > Prefabs** and drag and drop the *SDFSculpt* into your scene.
5. On the *SculptSceneManager* > **MyCameras** add the Main Camera of the *SDFPlayer*.
6. On the *SculptSceneManager* > **ParentSculpts** add the *SDFSculpt.*
7. On the *SDFSculpt* > **MyName** give it a unique name.

For more unique sculpts, repeat steps 4, 6, and 7. Any sculpt that has the same “My Name”, “My Recipe Path”, and “My Mesh Path” strings as a Parent Sculpt from the Sculpt Scene Manager, will inherit its global *SDF Sculpt* properties.

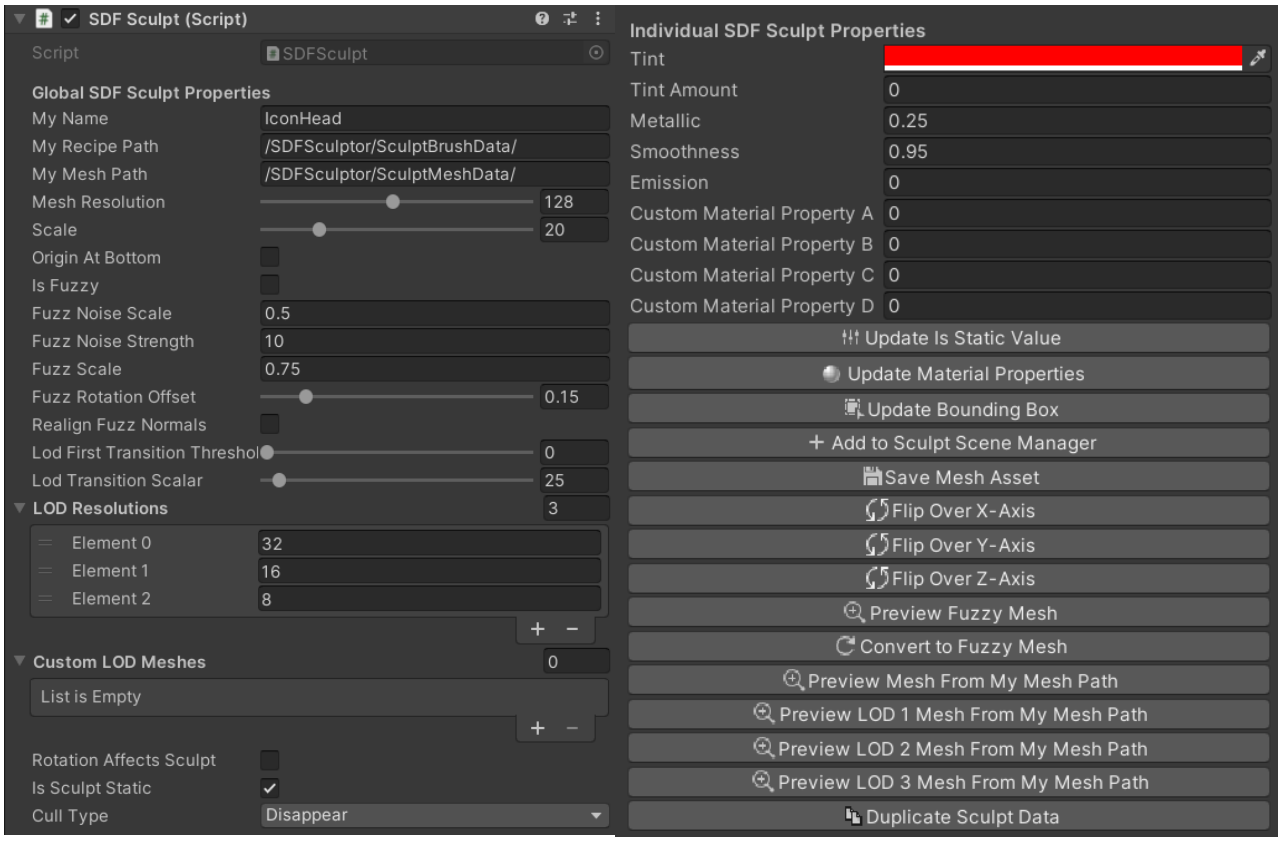
The *Sculpt Scene Manager* handles the LOD, frustum culling, occlusion culling, and GPU instancing systems. It also does part of the work of generating fuzzy meshes. This isn’t entirely necessary if you plan on using Unity’s LOD and Occlusion culling systems.

The *Mesh Generator* will generate a mesh, as well as its LODs, out of a signed distance field. It also does the work of creating data files for when new sculpts are created. This is needed if you want to do any sculpting.

The *SDF Player* is a camera setup to view the global illumination. It contains the *GI Camera Properties* component, with the *GI Camera Shader* material pre-assigned. Spectator Mode disables HUD and makes the player a free camera controller.

The *SDF Sculpt* has its own dedicated section just below.

4. SDF Sculpts



What is an *SDF Sculpt*? It is as a collection of SDF brushes, shapes, or edits resulting in some final SDF shape or sculpt. We then take this signed distance field and generate a mesh or multiple meshes at different resolutions to get LODs. Here are the inspector variables:

- **My Name**: The name of the sculpt, used when saving recipe and mesh files.
- **My Recipe Path**: Application.dataPath + "/ Insert Recipe Path /" Make sure to include slashes at start and end.
- **My Mesh Path**: Application.dataPath + "/ Insert Mesh Path /" Make sure to include slashes at start and end.

Every unique SDF Sculpt has a unique name and path to a mesh / recipe data file, as well as mesh files for LODs. These data files are JSON files containing mesh info (mesh vertices, UVs, normal directions, vertex colors) and recipe info (brush positions, colors, blend values, etc.). You can set up your own folders, but by default the *SDF Sculptor* folder has a folder for sculpt brush data and sculpt mesh data already.

- **Mesh Resolution**: Resolution of Sculpt. Best to keep everything in powers of 2: 16, 32, 64, 128, 256. Resolution of LOD0.
- **Scale**: Scale of the sculpt in terms of the raw geometry. The mesh can be scaled using the transform like normal after it is generated.
- **Origin At Bottom**: Make the bottom center of the sculpting space the origin. By default, the middle is the origin.
- **Is Fuzzy**: When generating a mesh, this will make it fuzzy.
- **Fuzz Noise Scale**: How big is the noise pattern.
- **Fuzz Noise Strength**: How strong is the effect of the noise.
- **Fuzz Scale**: The size of each fuzz quad.
- **Fuzz Rotation Offset**: A uniform offset for the fuzz rotation.
- **Realign Fuzz Normal**: For each fuzz quad, make all normal info match.

Setting is fuzzy to true will make this sculpt a fuzzy mesh. The noise scale determines the scale of a 3D noise pattern and the strength determines how much the noise affects the fuzz's random rotation. This value is assigned in the vertex color's alpha channel. The scale determines the size of the fuzz quad. The fuzz quad is just a quad of the sculpt. The fuzz rotation offset will add a rotation value to all fuzz quads uniformly. Realign fuzz normal makes all normal info on each fuzz quad match the direction the quad is facing. Otherwise, the normal info from before being turned fuzzy will be used. That may look better depending on your opinion. Once the fuzz options are set, you still need to regenerate the Sculpt or click the Convert to Fuzzy Mesh button. If you do not do this, the old non-fuzzy sculpt mesh will be used.

- **LOD First Transition Threshold**: Increase or decrease the screen coverage threshold before the first LOD transition. 1 = never uses first LOD, 0 = never transitions.
- **LOD Transition Scalar**: Increase or decrease the distance before LOD transitions are triggered.
- **LOD Resolutions**: Resolution of LODs in descending order. [0] = LOD1's resolution, [1] = LOD2's resolution, and so forth.

The first transition threshold is how much screen coverage there must be before transitioning from LOD0 to LOD1. Increasing the LOD transition scalar increases how far away successive transitions occur. LOD resolutions is a list of integers that determine the mesh resolutions of LODs from LOD-1 to LOD-n. This is used by the *Mesh Generator* to generate the SDF Mesh and all its LODs.

- **Custom LOD Meshes**: If this list contains any meshes, it will be used as a substitute for the sculpt / sculpt LODs. [0] = LOD0, [1] = LOD1, and so forth.

This allows custom meshes to be used instead of SDF generated geometry. If you wanted to take advantage of the LOD / Occlusion Culling systems of *SDF Sculptor*, you can. If nothing is assigned, the *SDF Sculpt* will work normally.

- **Rotation Affects Sculpt**: When entering edit mode, the rotation of this Game Object will rotate all brushes.

When the SDF Player edits a sculpt, the sculpt will normally align itself to the X, Y, and Z axis. If rotation affects sculpt is true, the initial rotation of the sculpt game object will rotate all the SDF Brushes used to create the mesh.

- **Is Sculpt Static**: Static sculpts can be culled more easily since they never move and their bounding box information stays the same.

Dynamic sculpts have their bounding box information update every frame while static sculpts only have the information calculated ounce at the beginning. Trying to change the position of a static sculpt will cause the frustum / occlusion culling to not work.

The following are individual *SDF Sculpt* properties (Used by the SDF Shader) that are unique for each sculpt even when they share the same name, mesh path, and recipe path:

- **Cull Type**: How to cull this sculpt. Disappear, Lowest LOD, Only Shadow, and Only Shadow and Lowest LOD.
- **Tint**: Color to tint the sculpt.
- **Tint Amount**: How much to tint the sculpt.
- **Metallic**: How metallic is this sculpt.
- **Smoothness**: How smooth or rough the sculpt is.
- **Emission**: How emissive is the sculpt.
- **Custom Material Property A**: Decide how to use this value in the shader graph.
- **Custom Material Property B**: Decide how to use this value in the shader graph.
- **Custom Material Property C**: Decide how to use this value in the shader graph.
- **Custom Material Property D**: Decide how to use this value in the shader graph.

The following are buttons for the *SDF Sculpt*:

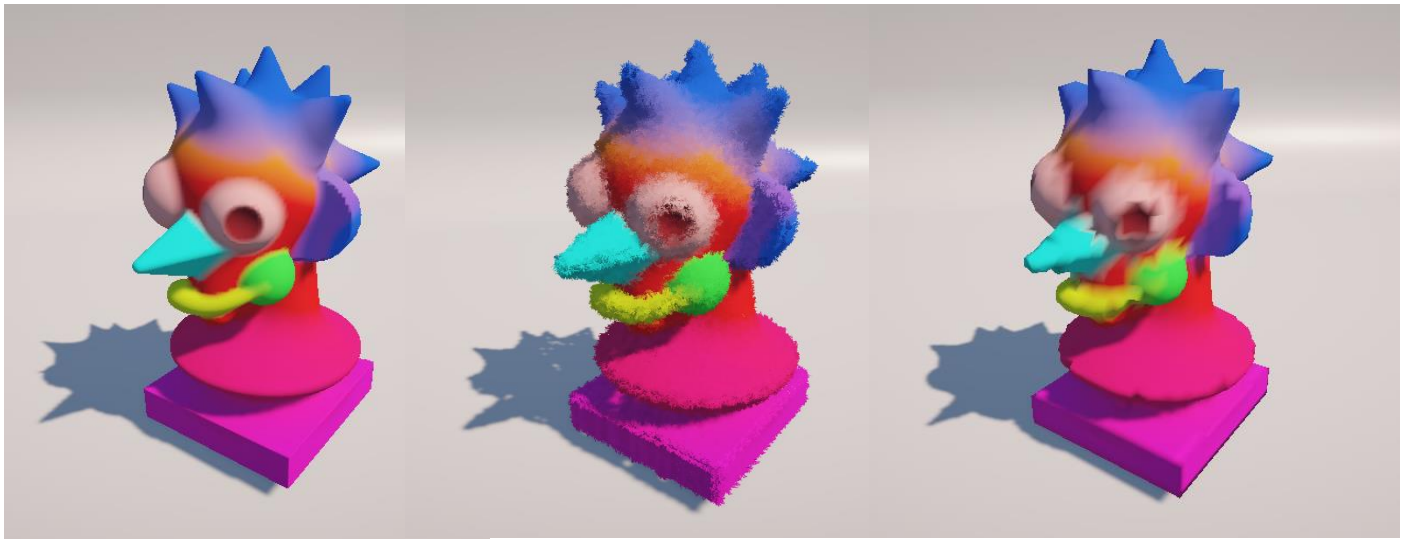
- **Update Is Static Value**: In Play Mode, update Sculpt Scene Manager that "Is Sculpt Static" changed.
- **Update Material Properties**: In Play Mode, update Sculpt Scene Manager that this Sculpt's material properties changed (Only necessary if this sculpt is not being Instanced).
- **Update Bounding Box**: In Play Mode, update Sculpt Scene Manager that the transform or shape of this sculpt has changed.
- **Add to Sculpt Scene Manager**: In Play Mode, add this sculpt to the sculpt scene manager if it is a new sculpt. The manager does not automatically know if a new sculpt is made so you must notify it.
- **Save Mesh Asset**: Save the mesh in this Game Object's Mesh Filter as an asset file.

- **Flip Over X-Axis**: In Edit Mode, flip the SDF Sculpt / LODs over the x-axis. This will overwrite the mesh data of the original.
- **Flip Over Y-Axis**: In Edit Mode, flip the SDF Sculpt / LODs over the y-axis. This will overwrite the mesh data of the original.
- **Flip Over Z-Axis**: In Edit Mode, flip the SDF Sculpt / LODs over the z-axis. This will overwrite the mesh data of the original.
- **Preview Fuzzy Mesh**: In Edit Mode, preview the mesh in the mesh filter as a fuzzy mesh.

Flipping geometry over an axis will not affect a Sculpt's brush positions, so when you go to edit a sculpt the geometry will return to its original orientation. Previewing a sculpt as a "Fuzzy Mesh" does only that. On play it will return to an ordinary mesh.

- **Convert to Fuzzy Mesh**: In Edit Mode, convert a non-fuzzy sculpt into a fuzzy sculpt. This will overwrite the mesh data of the original.
- **Preview Mesh from My Mesh Path**: In Edit Mode, preview the mesh if a mesh file has been created.
- **Preview LOD 1 Mesh from My Mesh Path**: In Edit Mode, preview the first LOD of the sculpt if a mesh file has been created.
- **Preview LOD 2 Mesh from My Mesh Path**: In Edit Mode, preview the second LOD of the sculpt if a mesh file has been created.
- **Preview LOD 3 Mesh from My Mesh Path**: In Edit Mode, preview the third LOD of the sculpt if a mesh file has been created.
- **Duplicate Sculpt Data**: In Edit Mode, duplicate the current sculpt data to work on it without changing the original.

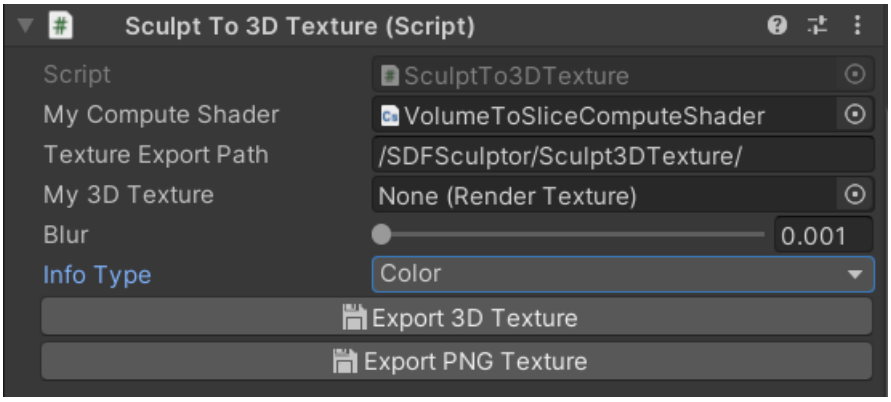
left: Preview Mesh (High Res), middle: Preview Fuzzy Mesh, right: Preview LOD1 (Low Res)



5. Export to FBX

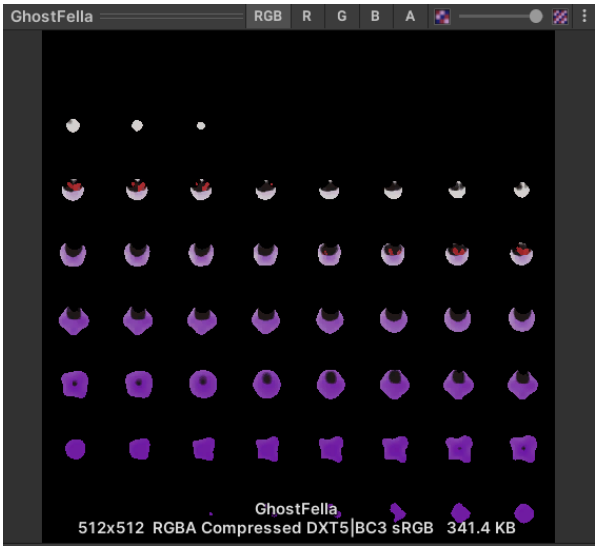
Meshes generated using *SDF Sculptor* can be saved as mesh assets and exported Unity's official FBX exporter. Exporting as an FBX is necessary if you want the vertex color information.

6. Sculpt to 3D Texture



Exporting a 3D texture using this component only works when playing your game with the Unity Editor and not after publishing the game, since exporting uses the Unity Editor libraries. Make sure you have the "Volume to Slice Compute Shader" assigned to the "My Compute Shader" variable. A 3D texture can still be generated if the Mesh Generator generates a mesh. By attaching this component to an object that has the SD Sculpt component attached, a 3D texture will be sent to this component when editing a sculpt that can be viewed in the inspector.

Clicking the "Export 3D Texture" button will generate a 3D Texture asset and send it to the specified texture export path. You can then use that texture however you would like.



Clicking the “Export PNG Texture” button will generate a texture atlas (Using the assigned compute shader) from that 3D texture which can be used outside of Unity.

7. Converting To HDRP

The demo scenes and prefabs are set up to work with URP by default. To convert all the materials in a scene to HDRP materials, go to the menu bar, click on **Tools** > **SDFSculptor** > **Convert SDFSculptor Materials in Scene To HDRP**. You will have to manually change the materials used by the prefabs to HDRP materials, and you will have to manually set up global volume settings to match the URP in HDRP.

8. SDF Player: Controls / Tools

The following are the tools for the *SDF Player* / settings that can be manipulated on the *SDF Brush*:

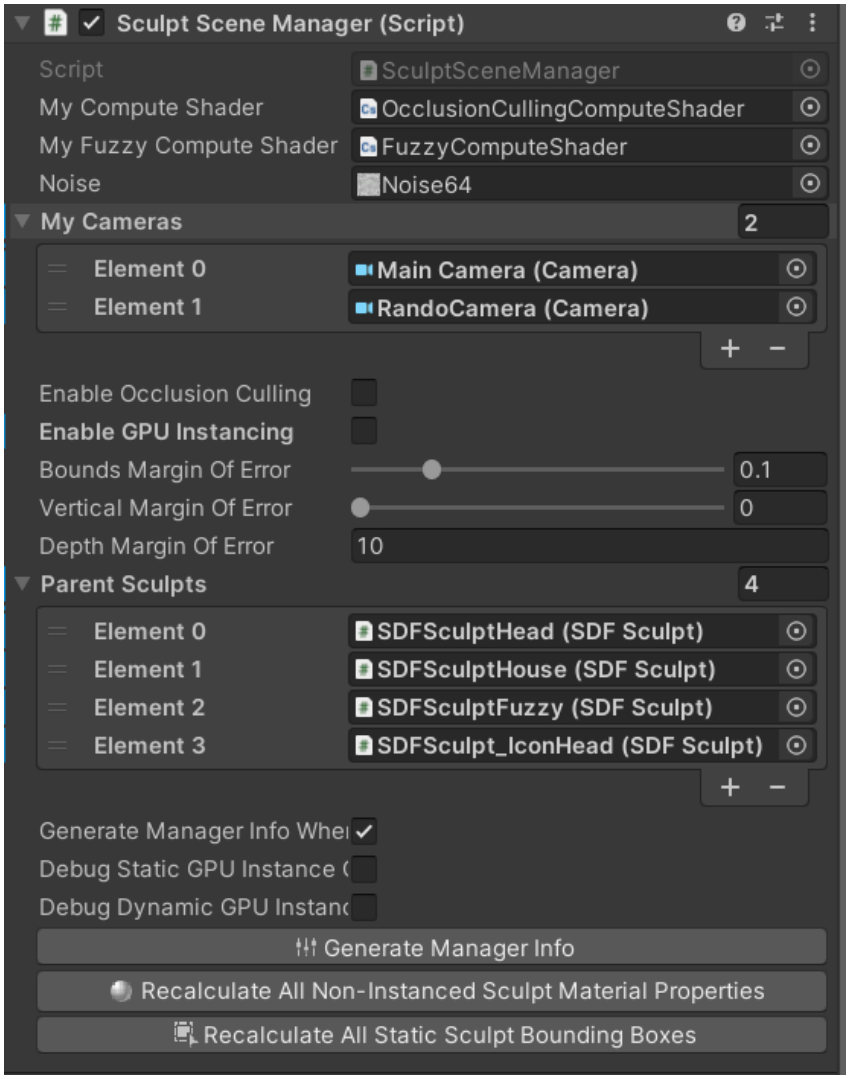
- **Grab (G)**: Move a brush.
- **Rotate (R)**: Rotate a brush.
- **Scale (T)**: Scale a brush.
- **Blend (B)**: Blend brush with other brushes.
- **Roundness (C)**: Roundness of a brush (Only some shapes).
- **Warp Strength (N)**: Warp strength of a brush.
- **Warp Scale (M)**: Warp scale of a brush.
- **Warp Pattern (V)**: Swap between warp patterns, there are 2 patterns.
- **Color (H)**: Color of a brush.

- **Brush (F)**: Pick a brush to add to the sculpt.
- **Add (+)**: Set any brush to "ADD" mode.
- **Subtract (-)**: Set any brush to "SUBTRACT" mode.
- **Intersect (Right Parenthesis)**: Set any brush to "INTERSECTION" mode.
- **Inverse Intersect (Left Parenthesis)**: Set any brush to "INVERSE INTERSECTION" mode.
- **Paint (*)**: Set any brush to "PAINT" mode.
- **Duplicate (Q)**: Duplicate a brush.
- **Delete (Backspace)**: Delete a brush.
- **Rotate Around X**: Rotate Sculpt Around X Axis by 180 degrees.
- **Rotate Around Y**: Rotate Sculpt Around Y Axis by 180 degrees.
- **Rotate Around Z**: Rotate Sculpt Around Z Axis by 180 degrees.
- **Grid X**: Toggle Grid facing Left/Right or on Y, Z plane.
- **Grid Y**: Toggle Grid facing Up/Down or on X, Z plane.
- **Grid Z**: Toggle Grid facing Front/Back or on X, Y plane.
- **Undo (,)**: Undo the last edit.
- **Redo (.)**: Redo the last undone edit.
- **Done**: Exit sculpt mode. Save mesh / recipe data.
- **Fuzzy Preview**: Preview what this sculpt looks like with fuzz settings.

The following are extra controls for the *SDF Player*. All controls / key bindings are in the SDF_HUD_DATA scriptable object.

- **Regenerate Mesh (P)**: Force the regeneration of a mesh when you edit the brushes in the Unity editor and it is not updating in game.
- **Rotation Snapping Key (Space Bar)**: When rotating to snap the rotation for more exact angles.
- **Selection Depth Key (Shift)**: With a tool selected and no brush selected, with this key pressed and scrolling the mouse wheel, you can select a brush behind the one you are looking at to select brushes that are hard to select.
- **Right Click**: Cancel current action.
- **Confirm Key (Enter) / Left Click**: Confirm current action.
- **View Tools Key (Escape)**: view all tools.
- **Spectator Key (Left Alt)**: Toggle spectator mode. In spectator mode, all HUD is disabled and sculpting is disabled.

9. Sculpt Scene Manager



The Scene Sculpt Manager oversees the LOD, frustum culling, occlusion culling, and GPU instancing systems. It is used to make using *SDF Sculpts* more performant and viable beyond prototyping and concepting. It also does part of the work of generating fuzzy meshes after a regular Sculpt's mesh has been generated.

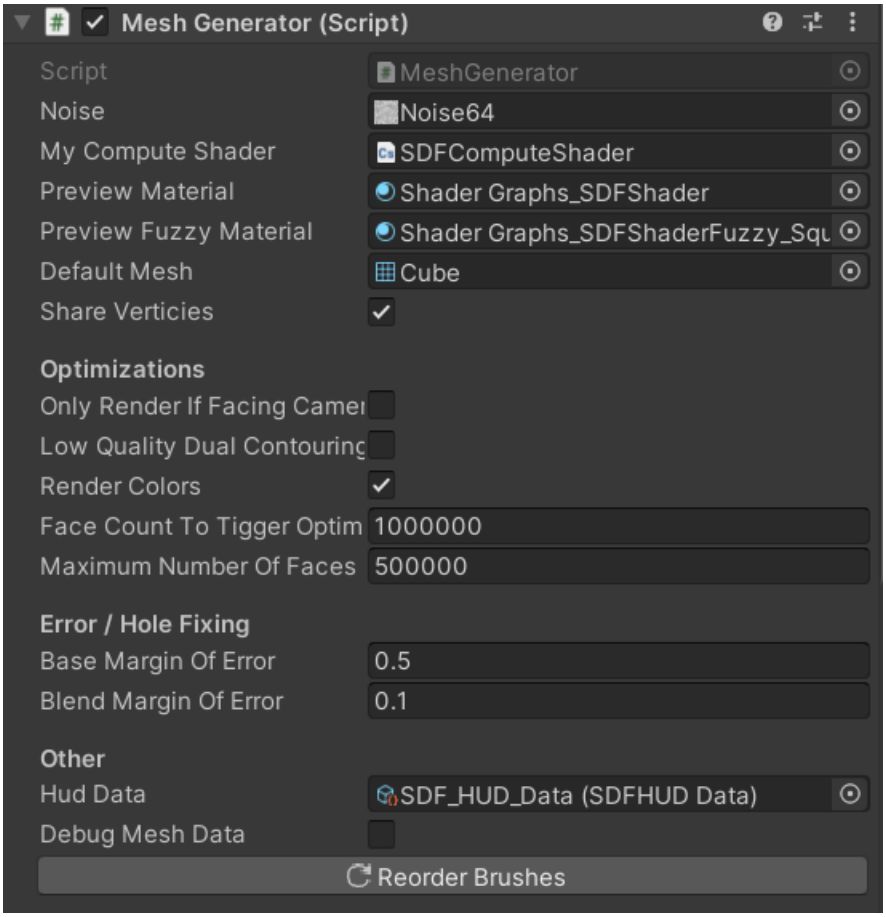
The LOD system / culling systems are intertwined. The first pass will be a frustum culling pass, which also gets the screen coverage amount of each SDF Sculpt. This info will allow us to know which LOD to use. Frustum culling only occurs when GPU instancing is enabled, since Unity automatically does frustum culling for non-GPU instanced geometry. If Occlusion Culling is enabled, the second pass will execute. Foreach pixel on the screen, each bounding box is checked against a depth texture

- **My Compute Shader**: The Occlusion Culling Compute Shader, handles frustum / occlusion culling and LODs.
- **My Fuzzy Compute Shader**: The Fuzzy Compute Shader, converts a sculpt into a fuzzy sculpt.
- **Noise**: The texture used when texture warp is enabled on an SDF Brush.
- **My Cameras**: List of all cameras that can see the GPU instanced sculpts. If no GPU instancing, there should only be 1 camera.
- **Enable Occlusion Culling**: Cull objects not directly visible.
- **Enable GPU Instancing**: Handles instancing manually over just checking the Enable Instancing checkbox on a shader. Max of 1022 instances per sculpt, per LOD.
- **Bounds Margin of Error**: Margin of error when determining the bounds of a mesh during frustum / occlusion culling.
- **Vertical Margin of Error**: Margin of error when for frustum occlusion on the top and bottom of the screen (needed for VR, 0 is fine without VR).
- **Depth Margin of Error**: Margin of error when depth testing in occlusion culling.
- **Parent Sculpts**: The parent sculpts are used to determine mesh resolution, scale, levels of detail needed for each copy / instance of this sculpt assuming the name, recipe path, and mesh path are the same. Try to not delete these in game.
- **Generate Manager Info When Scene Begins**: Generate manager info when the scene begins. Can be disabled if you want to trigger this manually / through code.
- **Debug Static GPU Instance Counts**: Show how many instances there are of each static sculpt LOD (if GPU instancing enabled) (Max of 1022).
- **Debug Dynamic GPU Instance Counts**: Show how many instances there are of each dynamic sculpt LOD (if GPU instancing enabled) (Max of 1022).

The following are buttons for the *Sculpt Scene Manager*:

- **Generate Manager Info**: In Play Mode, update manager info when sculpts have been created, deleted, or changed. This recalculates all information for all sculpts.
- **Recalculate All Non-Instanced Sculpt Material Properties**: In Play Mode, recalculate all non-instanced sculpt material properties if they have changed. Instanced sculpt material properties are updated every frame.
- **Recalculate All Static Sculpt Bounding Boxes**: In Play Mode, recalculate all static sculpt bounding boxes for frustum/occlusion culling.

10. Mesh Generator



The Mesh Generator will generate a mesh, as well as its LODs, out of a signed distance field. It also does the work of creating data files for when new sculpts are created.

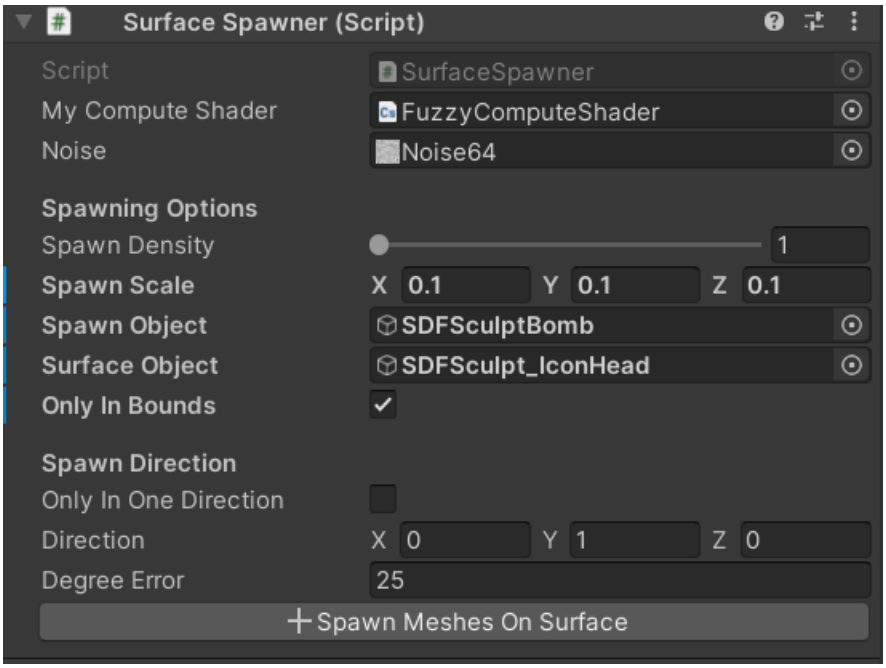
- **Noise**: The texture used when texture warp is enabled on an SDF Brush.
- **My Compute Shader**: The SDF Compute Shader that is responsible for defining a Signed Distance Field and building a mesh from it.
- **Preview Material**: The material used to preview how a sculpt will look like when editing a sculpt.
- **Preview Fuzzy Material**: The material used when previewing how a sculpt would look like if it was fuzzy.
- **Default Mesh**: The default mesh to assign to a sculpt when there is no sculpt information.
- **Share Vertices**: Smooth the normal info of the mesh. Will get a blocky result if disabled (more performant if disabled since we skip dual-contouring).
- **Only Render If Facing Camera**: Only render geometry that faces the player's camera.

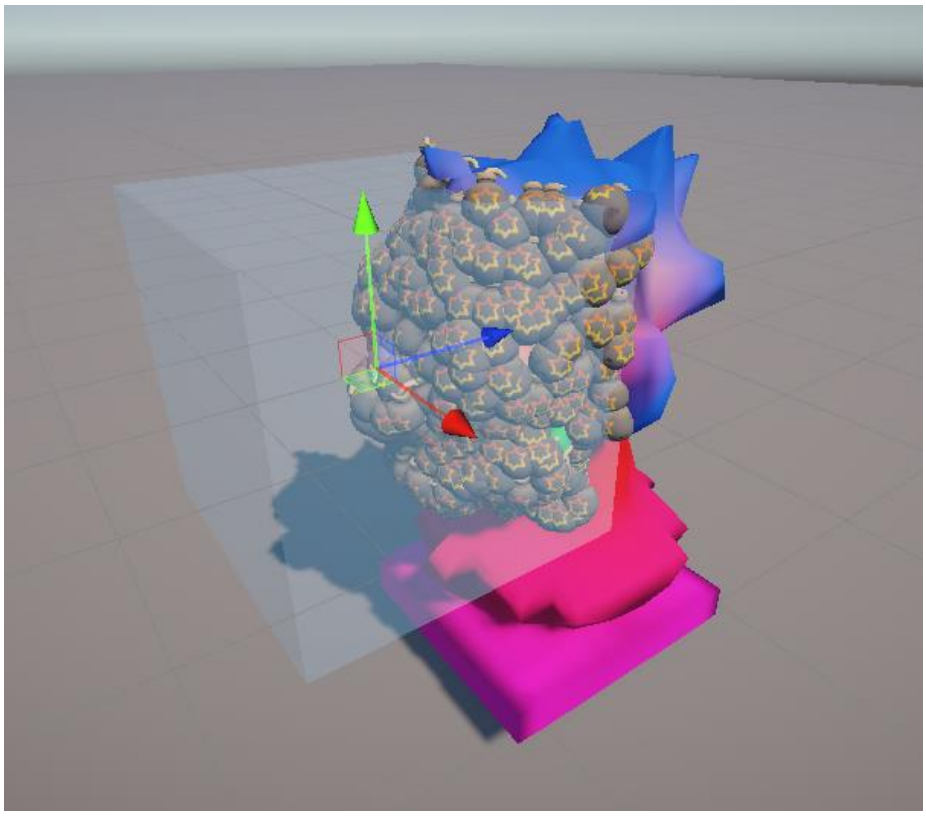
- **Low Quality Dual Contouring**: Only do a single smoothing pass on the mesh. Will get a slightly blocky result around the blended regions of the mesh.
- **Render Colors**: Calculate vertex colors.
- **Face Count to Tigger Optimization**: Trigger the "Only Render If Facing Camera" optimization when this face count limit is surpassed. May want to reduce sculpt complexity at this point.
- **Maximum Number of Faces**: Stop rendering faces if this limit is surpassed. When a sculpt gets too complex it can take a long time to update, so this is the final safe guard.
- **Base Margin of Error**: Expand the area around a brush to look for overlapping brushes (Bigger => Less Holes, Smaller => More Performance).
- **Blend Margin of Error**: Expand the area around a brush multiplied by the blend of the brush to look for overlapping brushes (Bigger => Less Holes, Smaller => More Performance).
- **HUD Data**: Get prefabs for brushes from SDF HUD Data.
- **Debug Mesh Data**: Show face and voxel information to see how complex the mesh is.

The following is the one button for the *Mesh Generator*:

- **Reorder Brushes**: In Play Mode, if the order of brushes has been changed / sibling index has changed, notify the Mesh Generator.

11. Surface Spawner





The Surface spawner is a level editing tool that will automatically place objects on the surface of other objects. You can also set a direction you want the surface to face for spawning as well as set the bounds of the spawning. (This isn't limited to sculpts, any mesh can be used)

- **My Compute Shader**: Move a brush.
- **Noise**: Rotate a brush.
- **Spawn Density**: How many objects to spawn per triangle on a mesh's surface.
- **Spawn Scale**: Scale of the objects to spawn.
- **Spawn Object**: The object to spawn.
- **Surface Object**: The object to spawn on (Must have a Mesh Filer / Mesh) (Mesh needs Read/Write enabled).
- **Only In Bounds**: Only spawn objects inside the bounds of this Surface Spawner.
- **Only In One Direction**: Only spawn objects on parts of the mesh that face a certain direction.
- **direction**: The direction a triangle needs to face for objects to spawn on it.
- **Degree Error**: In degrees, how different a triangle's normal can be and still have objects spawn on it.

The following is the one button for the *Surface Spawner*:

- **Spawn Meshes on Surface**: In Edit Mode, spawn sculpts on the surface of a mesh.

12. Shader Variables

The following are shader variables for the SDF Shader and SDF Shader Fuzzy:

- **Debug Instance ID**: Using the Instance ID, each instance (if GPU instancing is enabled) will be colored a different hue.
- **Debug Instance ID Max**: The biggest number of instances you want to debug.
- **Debug Instance ID Min**: The biggest number of instances you want to debug.
- **Fuzzy Texture 1-4 (Fuzzy Shader)**: The textures used for each fuzz quad.
- **Noise Texture (Fuzzy Shader)**: Used to sample a random value.
- **Sample Scale (Fuzzy Shader)**: The scale for sampling the random value.
- **Grainy Combine Method (Fuzzy Shader)**: Will combine each of the fuzzy textures instead of picking one texture per fuzz quad. If disabled, a fuzzy texture is selected using the sample scale multiplied by the vertex color alpha channel. This is only here since it gives an interesting result, but not everyone may like it.

Since the shader is just a shader graph shader, it should be easy to expand upon. The custom material properties do not do anything by default but could do whatever you want them to do. The SDF Shader could be changed to use vertex color to sample from a list of textures: using the red, blue, and green channels (The Alpha channel is used for random fuzz textures). The tri-planar mapping node can be used to get around UV problems. You could also use custom fuzz textures and use normal maps since by default the SDF shader does not use normal maps.

13. SDF Sculptor API

Below is a list of functions that can be called from your own scripts when using the *SDFSculptor* namespace. Look at *SDFSculptorController.cs* for how some of these functions can be used. Some of these functions are implemented in the inspector buttons if you need an example of their use.

Mesh generator:

- *Mesh Generator*: **ReorderBrushes ()**: If the order of the brushes has changed, it will not be automatically known, so call this function.
- *Mesh Generator*: **AddBrush (SDFBrush brush, int brushIndex)**: Add a new brush to the mesh generator / sculpt.
    - **brush**: The brush to be added.
    - **brushIndex**: The sibling index of the brush so the mesh generator knows the brush order.
- *Mesh Generator*: **RemoveBrush (SDFBrush brush)**: Remove a brush from the mesh generator / sculpt.
    - **brush**: The brush to be removed.
- *Mesh Generator*: **UpdateBrush (SDFBrush brush)**: Called after a brush's properties have been changed and the sculpt needs to be updated.
    - **brush**: The brush that needs to be updated.
- *Mesh Generator*: **TriggerMeshUpdate ()**: Called by the SDF Player after removing a brush, adding a brush, updating a brush, and after cancelling an edit.
- *Mesh Generator*: **ForceRegenMesh ()**: Called when you press the regenerate mesh button and internally in the reorder brush. Has more error checking than TriggerMeshUpdate().
- *Mesh Generator*: **IsUpdating ()**: Check if the Mesh Generator is busy generating a mesh.
- *Mesh Generator*: **FlipOverAxis (Vector3 axis)**: Called to rotate all brushes around an axis that runs through the center of the bounding box 180 degrees.
    - **axis**: The axis to rotate all brushes around.
- *Mesh Generator*: **PreviewFuzzy ()**: Called to make the mesh generator's mesh a fuzzy mesh for. Will also temporarily change the material to the preview fuzzy material.
- *Mesh Generator*: **SaveRecipe ()**: Called when done editing a sculpt and exiting edit mode as the SDF Player.
- *Mesh Generator*: **LoadRecipe ()**: Called when beginning to edit a sculpt and entering edit mode as the SDF Player.
- *Mesh Generator*: **ToggleGridXY ()**: Toggle the grid plane facing front and back.
- *Mesh Generator*: **ToggleGridXZ ()**: Toggle the grid plane facing up and down.
- *Mesh Generator*: **ToggleGridYZ ()**: Toggle the grid plane facing left and right.

Sculpt Scene Manager:

- *Sculpt Scene Manager*: **AddToAllSculpts (SDFSculpt sculpt, Renderer ren)**: All sculpts is an internal list of all sculpts the sculpt scene manager oversees. All SDF Sculpts call this function automatically at the start.
    - **sculpt**: The brush that needs to added.
    - **ren**: The renderer of the brush.
- *Sculpt Scene Manager*: **GetAllMaterialPropertyRenderTextures ()**: Part of generating manager info: creates all render textures for assigning material properties to GPU instanced sculpts.
- *Sculpt Scene Manager*: **BuildSculptsFromMeshData ()**: Part of generating manager info: converts mesh data JSON files into meshes.
- *Sculpt Scene Manager*: **SplitDynamicAndStaticSculpts ()**: Toggle the grid plane facing left and right.
- *Sculpt Scene Manager*: **AddToAllSculpts ()**: Part of generating manager info: separates all sculpts list into separate lists for dynamic and static sculpts. Important to know if GPU instancing.
- *Sculpt Scene Manager*: **GetBoundingBoxCorners (bool staticBBs)**: Part of generating manager info: get bounding box info for all sculpts. Dynamic sculpts call this function every frame.
    - **staticBBs**: True = recalculate all static bounding boxes. False = recalculate all dynamic bounding boxes.
- *Sculpt Scene Manager*: **SetNonInstancedMaterialProperties ()**: If not GPU instancing, the sculpt scene manager needs to know when the material properties have changed on a sculpt. Otherwise, this would only be called ounce.
- *Sculpt Scene Manager*: **SculptStaticDynamicChange ()**: Call this function to let the sculpt scene manager know a sculpt has changed from being static or dynamic, since it will not know automatically.
    - **sculpt**: The SDF Sculpt that is being updated.
- *Sculpt Scene Manager*: **AddSculpt (SDFSculpt sculpt)**: Add a new sculpt to the sculpt scene manager after start function has already executed.
    - **sculpt**: The sculpt to be added.
- *Sculpt Scene Manager*: **RemoveSculpt (SDFSculpt sculpt)**: Remove a sculpt from the sculpt scene manager after start function has already executed.
    - **sculpt**: The sculpt to be removed.
- *Sculpt Scene Manager*: **RecalculateSingleSculptMaterialProperties (SDFSculpt sculpt)**: If not GPU instancing, let the sculpt scene manager know when a single sculpt has changed material properties.
    - **sculpt**: The sculpt that needs its material properties updated.

- *Sculpt Scene Manager*: **RecalculateSingleStaticSculptBoundingBox (SDFSculpt sculpt)**: When a static sculpt has changed its transforms, this function will only update for that one SDF sculpt.
    - **sculpt**: The sculpt that needs its bounding box info updated.
- *Sculpt Scene Manager*: **GenerateManagerInfo ()**: Update manager info when sculpts have been created, deleted, or changed. This recalculates all information for all sculpts.
- *Sculpt Scene Manager*: **GetFuzzyMesh (Mesh m, float fuzzNoiseScale, float fuzzNoiseStrength, float fuzzScale, float fuzzOffset, bool recalcNorms)**: Converts a regular mesh into a fuzzy mesh based on the given parameters and returns it. Uses the vertex color alpha channel to encode some of this information.
    - **m**: The mesh that will be turned into a fuzzy mesh.
    - **Fuzz Noise Scale**: How big is the noise pattern.
    - **Fuzz Noise Strength**: How strong is the effect of the noise.
    - **Fuzz Scale**: The size of each fuzz quad.
    - **Fuzz Offset**: A uniform offset for the fuzz rotation.
    - **Recalc Norms**: For each fuzz quad, make all normals match.
- *Sculpt Scene Manager*: **FlipOverAxis (SDFSculpt sculpt, bool xFlip, bool yFlip, bool zFlip)**: Flips / mirrors sculpt geometry over the x, y, or z axis. Does the same for LODs. This will overwrite the mesh data of the original.
    - **sculpt**: The sculpt being flipped or mirrored.
    - **X Flip**: Flip SDF sculpt over x-axis.
    - **Y Flip**: Flip SDF sculpt over y-axis.
    - **Z Flip**: Flip SDF sculpt over z-axis.
- *Sculpt Scene Manager*: **FlipOverAxisMesh (Mesh m, bool xFlip, bool yFlip, bool zFlip)**: Flips / mirrors a mesh over the x, y, or z axis and returns it.
    - **m**: The mesh being flipped or mirrored.
    - **X Flip**: Flip mesh over x-axis.
    - **Y Flip**: Flip mesh over y-axis.
    - **Z Flip**: Flip mesh over z-axis.
- *Sculpt Scene Manager*: **ConvertToFuzzy (SDFSculpt sculpt)**: Converts a regular sculpt into a fuzzy sculpt. This includes the LODs. This will overwrite the mesh data of the original.
    - **sculpt**: The sculpt being turned into a fuzzy sculpt.
- *Sculpt Scene Manager*: **RandomizeMeshAlpha (Mesh m, float fuzzNoiseScale, float fuzzNoiseStrength)**: Randomizes the vertex color alpha channel on the vertices of a mesh. Called when generating all meshes and is used in the fuzzy SDF shader.
    - **m**: The mesh being encoded with random values in its vertex color alpha channel.
    - **Fuzz Noise Scale**: How big is the noise pattern.
    - **Fuzz Noise Strength**: How strong is the effect of the noise.

Surface Spawner:

- *Surface Spawner*: **Spawn (float randomOffset)**: Spawns “spawn objects” on the “surface object” (assuming you have set these variables) using the surface spawner's parameters.
    - **randomOffset**: Pass a random float to get randomized placement This can be the current time for example.

SDF Sculpt:

- *SDF Sculpt*: **UpdateIsStatic ()**: Calls Sculpt Scene Manager->SculptStaticDynamicChange.
- *SDF Sculpt*: **UpdateMaterialProperties ()**: Calls Sculpt Scene Manager-> RecalculateSingleSculptMaterialProperties.
- *SDF Sculpt*: **UpdateBoundingBox ()**: Calls Sculpt Scene Manager-> RecalculateSingleStaticSculptBoundingBox.
- *SDF Sculpt*: **AddToSceneManager ()**: Calls Sculpt Scene Manager->AddSculpt.
- *SDF Sculpt*: **FuzzFunction ()**: Calls Sculpt Scene Manager->GetFuzzyMesh.
- *SDF Sculpt*: **ConvertToFuzzy ()**: Calls Sculpt Scene Manager->ConvertToFuzzy.
- *SDF Sculpt*: **FlipOverAxis (bool xFlip, bool yFlip, bool zFlip)**: Calls Sculpt Scene Manager-> FlipOverAxis.
    - **X Flip**: Flip SDF sculpt over x-axis.
    - **Y Flip**: Flip SDF sculpt over y-axis.
    - **Z Flip**: Flip SDF sculpt over z-axis.

14. Empty Compute Buffer

When sculpting, you may run into the Empty Compute Buffer error where you will not be able to sculpt anymore. Typically, this is caused when there are brushes outside of the bounding box, so when SDF Sculptor tries to calculate how many faces need to be generated to fill a compute buffer for verts, triangles, colors, etc. it would create compute buffers of size 0 which causes an error. This can also happen if you have a subtraction brush that fills the whole bounding box, causing no geometry to generate. The solution is to make sure you have a brush inside the bounding box and allow some geometry to generate, then click the “Regenerate Mesh” button.

15. Limitations

A Signed Distance Field is defined using a 3D texture and the size of it is determined by the SDF Sculpt’s Mesh Resolution. When you are done editing a sculpt, the texture is no longer stored in memory, so it will not impact performance if you are not trying to edit a sculpt. 64x64x64 is the default and should run decently smooth when editing, 128x128x128 still runs fine and provides double the resolution, and 256x256x256 is when performance gets choppy even with few SDF brushes. All the sculpts in SDF Sculptor made for demonstration purposes are 64 at the max, just to show what you are capable of even at lower resolutions.

When GPU instancing, we can render the correct LODs for each camera and cull geometry per camera. If not GPU instancing, the LOD / culling systems will only work with one camera, since the Sculpt Scene Manager would directly enable / disable sculpt renderers and swap out meshes in mesh filters. Also, when GPU instancing, we can edit material properties of SDF sculpts and see the change in real-time without having to call the Sculpt Scene Manager to update.

The function used to instance geometry is *Graphics.RenderMeshInstanced()*. This function has an upper limit to how many meshes it can render: 1024. SDF Sculptor caps the number of sculpts you can instance to 1022. Keep in mind, this limit is only the limit on how many instances of 1 unique sculpt can be rendered. For example, 1022 of the same rock, 1022 of the same tree, 1022 of the same pillar, etc. Each of those instances can have variations in their material properties, so they do not have to all look the same. Also, by creating a duplicate but renamed version of a sculpt, you can effectively double this limit. Also, each LOD counts as its own unique sculpt, so you could have more than 1022 of 1 SDF Sculpt on screen. This is because each mesh must be instanced separately and each LOD has a different mesh.

Typically, with GPU instancing compatible shaders, to pass the material properties to each GPU instance would require an instancing buffer containing all your material properties. Shader

graph does not support passing buffers to it natively, but it does have the ability to get the Instance ID. So, to pass material property data to shader graph, SDF Sculptor uses 32x32 textures. This gives us 32*32 = 1024 float4s to work with. Conveniently, 1024 is also the limit to how many meshes we can instance. Each sculpt being instanced gets 3 32x32 render textures with the default render texture format and with point filtering. The first render texture passes tint color. The second passes tint amount, metallic, smoothness, and emission values. The third contains custom material property values a, b, c, and d. To calculate the total memory being used, we can use the following approximate formula:

# of Cameras * # of Sculpts * # of LODs * 3 textures * ~4 KB (size of a 32x32 texture)

Therefore, if we had 2 cameras, 100 sculpts, and 4 LODs per sculpt, we would come up to 2*100*4*3*4 = 9600 KB or 9.6 MB in memory (a 1024x1024 texture is ~4MB big). This is on top of the cost of storing depth textures for each camera, which would just be the screen resolution. Part of how occlusion culling works is getting the depth texture of each camera and seeing if the depth matches the distance between the bounding box and the camera. Therefore, a list is created to hold the depth textures of each camera (multiple cameras only when GPU instancing).

There is a performance cost to occlusion culling, therefore you may want to turn it off if it does not make sense for your scene. For example, a vast open landscape where little will be occluded. Also, there is a performance cost to instancing large amounts of alpha clipped geometry / fuzzy meshes (trees, bushes, foliage). In the showcase demo scene, all fuzzy sculpts have a mesh resolution at a maximum of 32, not 64. Using lower resolution geometry and higher resolution fuzz textures might be the trick for rendering large amounts of alpha clipped geometry / fuzzy meshes.

The current setup allows editing an SDF sculpt in real-time, but is not set up for something like animation. It also is not setup to work with multiple mesh generators if you wanted to edit multiple sculpts at the same time, although it would not be impossible, it is untested. It is possible to hide the SDF Brush / Bounding Box materials to create the appearance of an animation, but this would require some manual work. The reason why this is not supported out of the box is because the performance of live editing an SDF sculpt can be poor, especially at higher resolutions. Trying to do that with multiple sculpts would be unreasonable. Again though, not impossible. This is especially true when the SDF sculpt is not that complicated.

16. Tips

If the performance during sculpting is bad even at a mesh resolution of 64, there are other settings on the Mesh Generator to improve performance. Disabling rendering colors improves performance by removing the step of sampling color at every vertex. Low quality dual contouring will cause the mesh to look blockier around areas where sculpts blend, since we only do a single dual contouring pass (normally, multiple dual contouring passes happen). Only render if facing camera will only render faces that face the camera. This will improve performance the most. Face count to trigger optimization is how many faces before the “only render facing camera” optimization is triggered automatically. Less faces to render mean less calculations and less sampling the signed distance field. The Error / Hole fixing settings in the Mesh Generator work well by default, but can be lowered even further to get more performance. These settings make more of a difference when you have multiple overlapping SDF brushes in the same area.

Some things are easier to do / need Unity’s editor tools. If you want to reorder the brushes, you would have reordered the brushes in the hierarchy and press the “Reorder Brushes” button to make the changes come to affect. Also, certain things like brick walls are easier to make by copying and pasting boxes in the hierarchy than duplicating them with SDF Player’s duplicate tool. The SDF Player Controller does not have a way of creating new sculpts, so if you want to do that you have to create a new sculpt in the editor or extend the player controller to do that. You would have to instantiate an SDF sculpt, then call the sculpt’s AddToSceneManager(), UpdateMaterialProperties(), UpdateBoundingBox() functions.

If you wish to ignore the LOD / culling systems, you can use SDF Sculptor for making art while using Unity’s built-in LOD and occlusion culling systems. Of course, the advantage of using SDF Sculptors LOD / culling systems is that there are no baking times since occlusion culling is calculated in real-time. Alternatively, you could ignore the sculpting tools and use the LOD / culling systems with your own meshes. The SDF Sculpt has a list called custom LOD meshes, which is normally empty. However, if you put meshes inside, those meshes will be used instead of the mesh stored in the SDF mesh data file if the meshes have read/write enabled. You can also turn your custom meshes into fuzzy meshes / flip them over the x, y, or z axis like you can with regular SDF sculpts using the SDF Sculptor API (FlipOverAxisMesh() / GetFuzzyMesh()). The surface spawner can be used with custom meshes too if the mesh has read/write enabled.