StructureBoneZ

&

SBZ_Painter

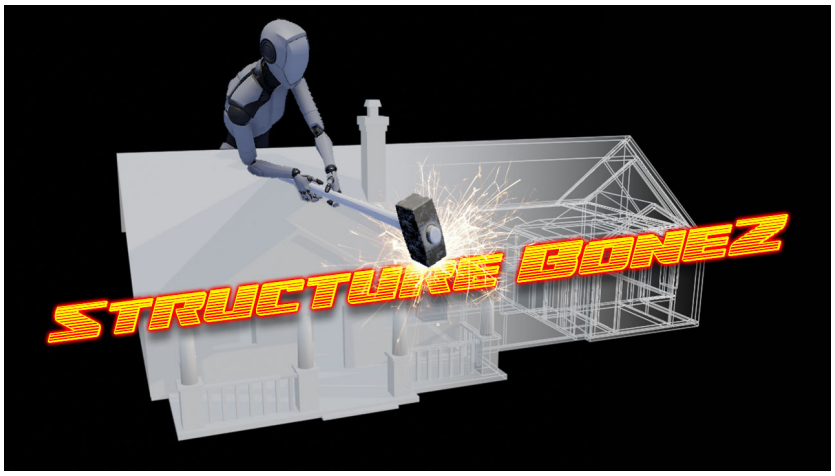


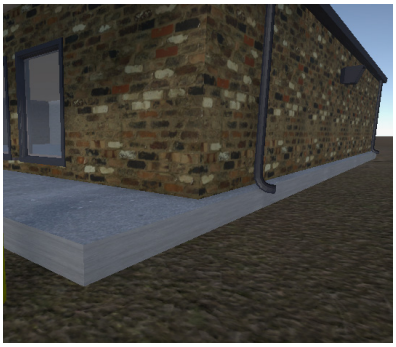
Table of Contents

What exactly are StructureBoneZ?

StructureBoneZ (aka: StructureBones, SBZ) are mesh buildings with full interior and exterior that constitute the “bones” of the structure. Their textures can be customized on-the-fly in the Unity editor and are naturally persistent in the Unity scene. They use a single shader and very few materials for all static elements but allow extreme variety through the use of texture2Darrays and per-triangle RGB tint, scale, shift, rotation, metallic & smoothness settings.

StructureBoneZ are perfect for rapid prototyping of gameplay and artistic style in full-size, enterable buildings. Regardless of the texturing method, the mesh are built to serve as a great starting point for your own art. With the StructureBoneZ starter kit we provide a small subset of the building types that we are working on. Many more will be offered in near future to add to your collection. Everything from police departments, schools, warehouses, manufacturing facilities, and additional homes are in the works and will be available separately.

We focus on getting the walls, floors, and roofs in ideal condition. The details that we include are prominent ones that are likely to use tiled textures and have no interaction.



In the starter pack, we do include some sample interactive doors and interaction components to get you started.

We try to stop short of delving too far into what will make your game unique.

The expectation is that you will add your own furniture, appliances, wall and floor decals, lighting, and other props to achieve the environment that you want.

The provided textures are all CCO licensed. You can keep, edit, add, or swap them out as needed. We create some of them and others were produced by third parties and are freely available under CCO license.

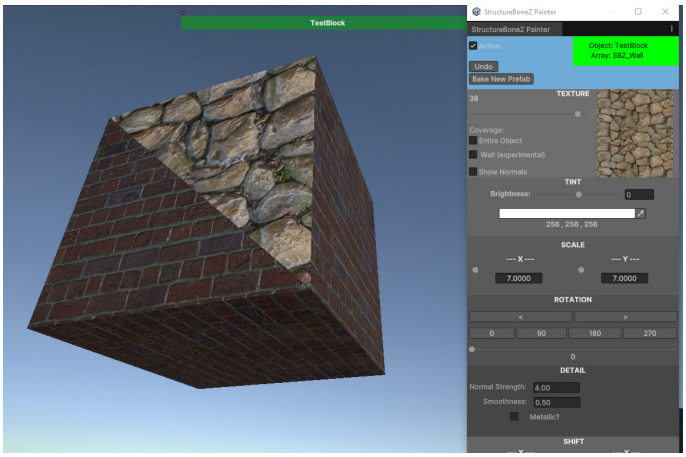
We supply these buildings textured with a texture2Darray shader. At this time we offer the shaders created with Amplify Shader Editor. The latest Beta version of Unity shadergraph gives access to the extended UV channels that we need for this setup. So we plan to provide shadergraph based shaders as well in the very near future.

The texture array materials are great for early stage prototyping since you can “paint” directly in the editor by manipulating the mesh UV data. This allows for quick change and analysis of your texturing work. Making changes to the various parts of your building doesn't require cutting and separating submesh to apply different materials. Instead, you apply different textures to each triangle of the mesh with a single material. This allows you to sort out over time what all textures you need where.

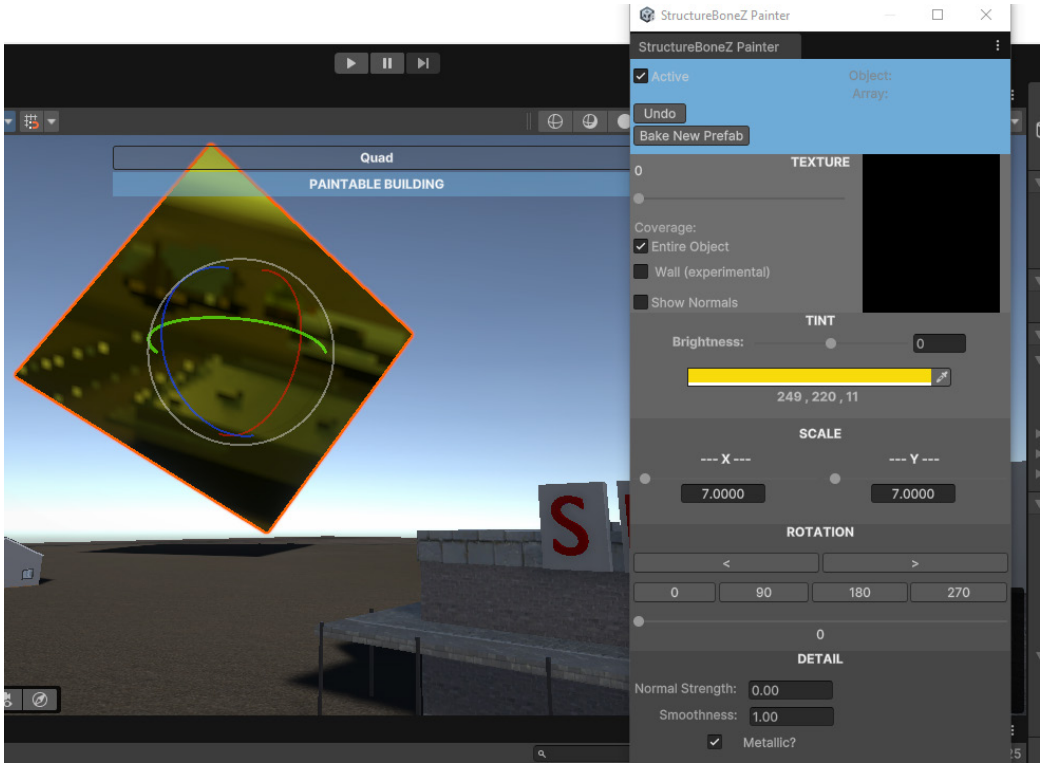
It also sets up perfectly for static batching by using minimal materials across all buildings that use it. Initially, we use only 4 materials for all buildings in scene (Walls, Floor, Roof, Details). While static batched, this translates into only 4 draw calls per frame for these structures. The moving items (ex: doors) cannot join in on the static batch so they will cost additional draw calls regardless of whether you texture them with array materials or other.

Of course, StructureBoneZ do not require the texture array materials. You can replace them entirely, or in part, with standard materials and edit them in a more traditional workflow if you prefer.

Likewise, the UV editor that we supply (SBZ_Painter) not only works on the structures that we provide, but you can also paint on practically any mesh with a mesh collider as long as it has a compatible texture2Darray material.



It will not, however, currently “play nice” with submeshes. A warning will pop-up if you try to paint a submeshed object. They can co-exist with the texture array compatible objects in the same structure, but you cannot currently “paint” submeshed objects.

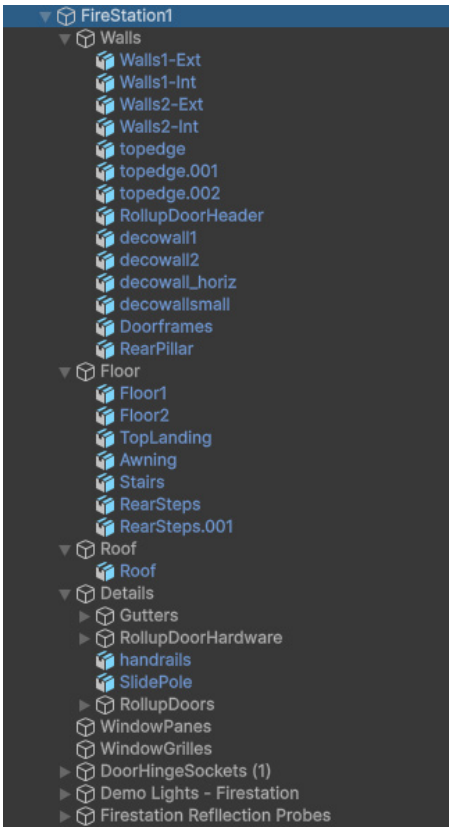


SBZ_Painter writes and adjusts UV data within the mesh to record all needed information. Unity serializes all of this data in the scene, so that when you reload the project, all changes to the mesh are persistent. All of these changes will also be properly translated in a standalone build.

UV changes can also be made at runtime, but those are not inherently persistent and SBZ_Painter does not attempt to function at runtime even within the editor environment.

SBZ DATA FORMAT

Each SBZ structure is a collection of mesh building objects.



Each object has mesh collider, and texture array based material.

4 main materials are used for the buildings (1 additional for window panes)

- Walls
- Floor
- Roof
- Details

We texture some dynamic detail objects (like doorknobs) with standard materials instead of the texture array materials since they will not be batched anyway. This is also because doorknobs and door handles will many times be covered by a collider for interaction purposes. That collider can interfere with raycast “painting” by catching the raycast instead of the intended mesh collider on the mesh object. There are other ways to alleviate this conflict but we try to keep these things as simple as possible and just use a standard material approach for objects like these.

For each of the 4 main materials, we supply 2 texture array sets (albedo and normal)

These are selected, along with other settings directly from the mesh UV data

So all of the texturing instruction is contained within the mesh itself.

UV3-UV5 are used for this purpose because:

- UV0 is the original texture map
- UV1 can be used by Unity for realtime lighting
- UV2 can be used by Unity for baked lighting

That leaves us to store various other settings regarding the mesh presentation in the remaining UV sets.

Each UV set is a list of values (Vector2, Vector3, or Vector4).

In each set there is one of these values for every vertex in the mesh.

With 3 UV sets, each having Vector4 values, that gives us 12 float values that can be recorded and adjusted to affect each triangle of the mesh.

The painter rebuilds any models that share vertices among their triangles so that each vertex is unique. That way we can access data from the UV coordinates by lookup when we know the triangle that we want to affect. Also, the shader can gather instructions for how to display each triangle within the mesh, simply by reading it's accompanying UV data.

So we store the following data as 12 floats

------ TABLE ----------

UV3.x ----- texture array index (slice)

UV3.y ----- tint Red

UV3.z ----- tint Green

UV3.w ----- tint Blue

UV4.x ----- scale in X direction of sampled texture

UV4.y ----- scale in Y

UV4.z ----- rotation angle

UV4.w ----- normal strength

UV5.x ----- shift in X direction of sampled texture

UV5.y ----- shift in Y

UV5.z ----- smoothness

UV5.w ---- metallic

UV3.x

----Texture Selection-----

This selects the textures that are pulled from the array. Each array can hold up to 256 textures and they are selected by indices 0-255. We actually save the index value as a fractional 0-1 and this is multiplied by 256 in the shader to select the desired texture

UV3.y, UV3.z, UV3.w

----Tint-----

RGB is represented by these 3 coordinates, each with a value from 0-1. The chosen tint color is added to the brightness setting and merged with the current texture by multiplication.

UV4.x , UV4.y

--- Scale X and Scale Y---

The original scale is baked into UV0. In the shader, these factors scale it up or down from there.

This might should really be called “Focus” instead of “Scale” because as you make the UV coordinates smaller, it zooms in and makes the texture look larger on your model.

The slider controls in SBZ_Painter only run from 0.0001 - 200, but you can manually input factors outside of these limits to further adjust if needed.

UV4.z

--- Rotation ---

The original rotation of the texture is baked into UV0. But the shader performs rotation based on this setting before displaying any individual triangle. We use 0-1 values here as well and the shader multiplies by 360 to get the actual degrees of rotation.

UV4.w

--- Normal Strength ---

The normal map arrays match 1:1 with the main texture arrays. They're used to simulate lighting effects giving the perception of graininess (depth) on object surfaces

UV5.x , UV5.y

--- Shift X, Shift Y ---

The original shift is baked into UV0. In the shader, these factors move it on X or Y from there. This is useful if there are parts of a texture that you want to align to specific locations on your model. In particular, this is handy if you use Seamless-X textures (ex: wall textures with stains dripping from top). In this case, you can adjust Shift-Y to align the stains to the top edge of the wall.

UV5.z

--- Metallic ---

Here we only use 0 or 1 settings.

0 = non-metallic (wood, carpet, wallboard, etc)

1 = metallic (door knobs, metal roofing, etc)

*** Caution: If you use this metallic setting, you will need to add and bake reflection probes into your scene (otherwise they will just look very dark because there are no reflections)

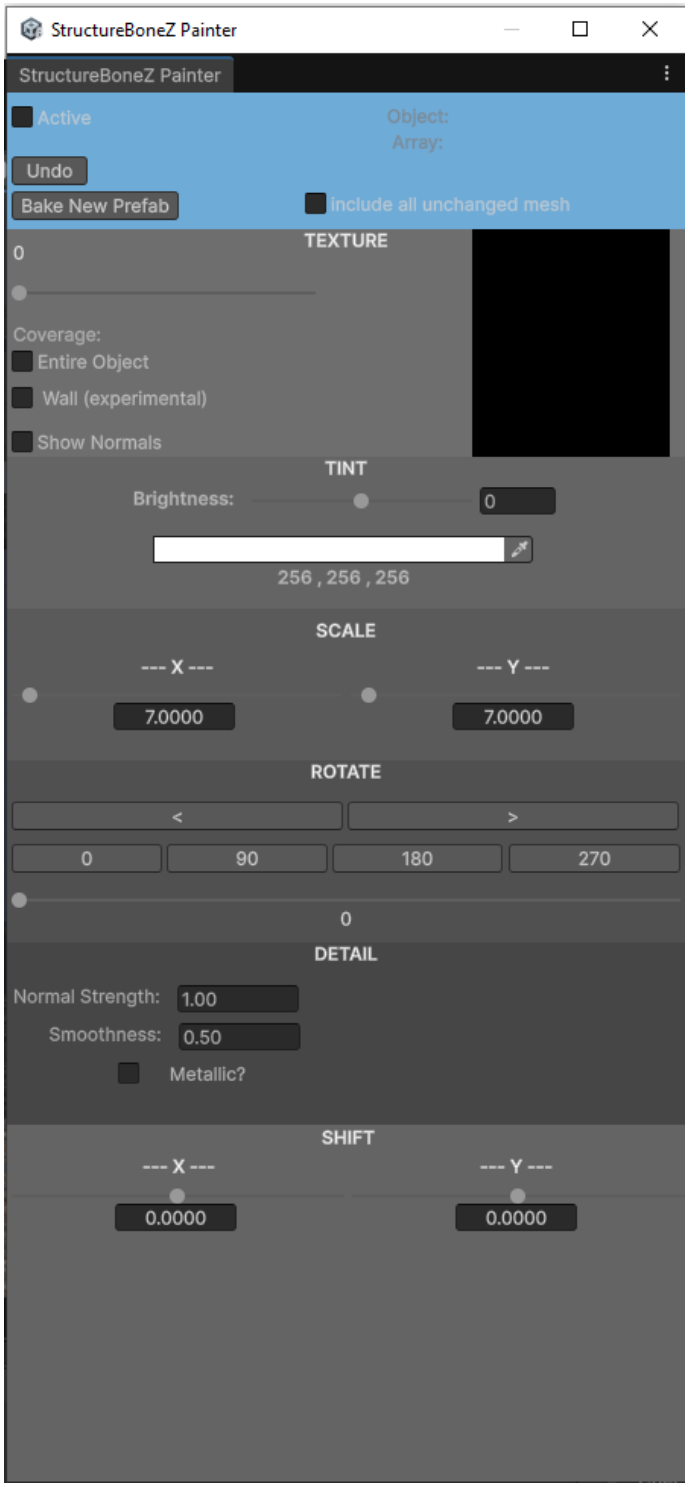
UV5.w

--- Smoothness ----

This is from 0-1 how smooth the object is. This affects light reflection on metallic or non-metallic objects. You set the value high for shiny floors and low for non-reflective surfaces like carpet. Shiny objects will only reflect light properly if you add and bake reflection probes nearby. The light that they reflect will be the light that the probe sees within it's clipping planes.

SBZ_PAINTER

Quickly and easily “paint” each mesh triangle using our custom SBZ_Painter. Being a Unity editor, there's no need to jump in and out of a modeler for UV editing. This works with most any mesh, but is intended to be used in conjunction with texture2Darray shaders.



It bakes all needed texture data into the mesh itself. Each building that we supply is made of multiple mesh gameobjects. A typical minimum would be Walls, Floor, Roof. There are normally also some detail objects (doorframes, windowframes, baseboards, window panes, window grilles, etc.) Each of these objects can also be textured with SBZ_Painter as long as they have mesh collider and an SBZ compatible material.

Window panes and glass doors do not use such material for several reasons. Namely: busting a few windows and glass doors out works better if they're not all combined together! We may in the future make glass doors that have the frame separate from the glass so that the frame could be “painted”.

Also, doorknobs normally have box colliders covering them that will be used for interaction. For this reason, and the fact that they are not going to be static batched anyway, we use standard materials on them as well instead of making them “paintable”.

ACTIVE checkbox

The SBZ_Painter is made for speed. This starts with the object selection process. Using the left mouse click for both object selection and “paint application” may seem a little disconcerting at first, but it makes for a quick workflow without extra keys and buttons involved.

The painter can only be made “Active” in editor mode (not at runtime). When it is “active”, Unity sceneview camera movement is disabled, and it looks for left-mouse clicks from the user to either “select a paintable object” or to paint on an already selected object.

If you click on an object and get no message but also no object selection is made, try clicking on another object first, then back to the target one. If that doesn't work, you may have to remove and then add again the mesh collider on that game object. We're investigating the cause of this behavior for potential cure in future updates. It seems to happen if you edit and replace some mesh objects within the structure but not all of them.

Once a paintable object is selected, you can left-click to paint it, or you can right-click to pull texture data from the hit triangle. So if you want to copy texture from another object that uses the same material, you can click the other object, right-click to get it's data, left-click to select the new object, then left-click again to paint that data into the new object.

UNDO button

Because, we're writing directly into the mesh data, there is no inherent undo functionality within Unity. For this reason, we backup all mesh data before changes and give you a way to apply previous data that was present on the currently selected object. This is particularly helpful if you accidentally paint over an entire object when you meant to just change a few triangles or walls instead. NOTE: if you exit the editor, or select another object, then all undo data is destroyed and cannot be recovered

BAKE PREFAB button

When you modify the mesh UVs, a new mesh is created. So any prefab must link to the new mesh (instead of the original FBX mesh) So when you bake a prefab from one of these structures, it writes the new mesh assets into a directory of your choosing. These prefabs can then be placed in other scenes, or spawned programmatically. So that SBZ_Painter can identify the root gameobject of the structure, you place an SBZ_ID component on it. This is only required for the prefab baking process.

By default, we only save new mesh from objects that have been changed. So, if you were to load up a building prefab and, lets say, just paint the roof. Then if you bake a new prefab, it will only save the new prefab and the new roof mesh (not every other mesh in the structure). So the new prefab is using the original mesh for all objects except the roof. Of course that original mesh could be from fbx files or another baked prefab, either of which may very well be in other folders scattered around in the project. So you should consider this when you choose where to store the mesh data.

This is also why there is the option:

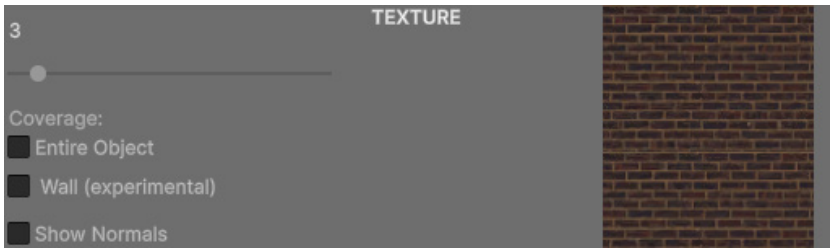
"include all unchanged mesh"

Use this if you want to make a complete single location repository of a structure's mesh objects

Once a prefab is baked, it can be placed as normal at design or run-time.

TEXTURE area

When a paintable structure object is selected, SBZ_Painter attempts to display the possible textures in the texture2D panel on the right. The index of that texture within it's array is displayed under a horizontal slider bar to the left. You can adjust this slider back and forth to display all textures available to this objects material. Normally, the color texture (diffuse/albedo) is shown. If you enable the “Show Normals” checkbox, then the associated normal map will be displayed instead.



COVERAGE

Normally, any single left-click will paint the current chosen texture to a single triangle in the selected object. If you hold left-shift down while moving the mouse cursor over other triangles within that object then they will be immediately changed as well.

Entire object checkbox

If the entire object checkbox is active, then all triangles contained in the selected object will be painted. BE CAREFUL: It's very easy to forget that this is active and paint over a lot of backside mesh faces.

Wall checkbox

To help with painting multiple triangles along the same face, we developed this novel approach. It only works on faces that are aligned within a slight tolerance along X, Y, Z local axes relative to the structure. It will not work on spheres, cylinders, or any other non-aligned faces. But this can be very helpful when painting interior surfaces of outer walls.

TINT

For tinting you can choose any color with the color picker. But this tint is blended with the original texture colors, so to further adjust it's affect, you can change the “Brightness” slider. This will always return to zero after use and the actual derived color is stored into mesh.

SCALE

As you scale the UVs smaller, the resulting image appears larger on the mesh. X and Y can be adjusted independently to stretch or shrink the image along it's original X or Y directions.

ROTATION

Each triangle can be rotated individually from 0-360 degrees. The original texture rotation is taken from UV0 which is the standard UVs that are typically edited in your modeling software.

DETAIL

Each triangle can be adjusted for Normal Strength and Smoothness. Metallic can be set to on or off.

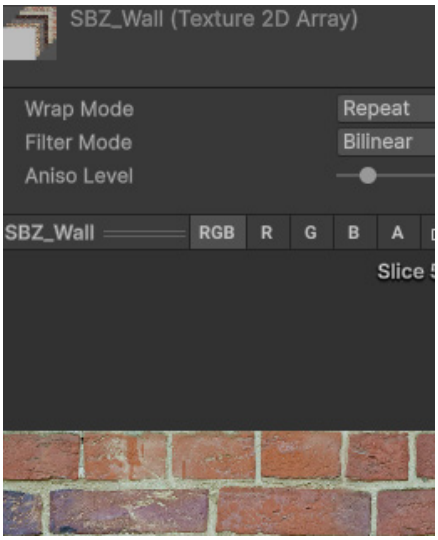
- Normal Strength This determines the “bumpiness” of the image. It alters the texturing based on the applied normal map to give the perception of depth.
- Smoothness This determines how lights will reflect from the object.
- Metallic We made this a bool rather than float so that you can choose simply is the object metallic or not.

If metallic is set true, then you will probably have to add reflection probes into your scene covering this object so that light will work correctly. Otherwise, the object will just look very dark because it is not receiving any reflections from the environment.

SHIFT

The original UV0 data in the mesh determines what part of the texture is displayed. To move them around on X & Y, you can use these shift sliders or enter shift amount directly into the input boxes. This is handy if you have a part of the texture that you want aligned to a specific part of your object.

TEXTURE ARRAYS



Texture arrays are, as you might expect, just an array of textures in a single file. They provide an easy way to access many different textures with a single material. They essentially serve the same purpose as texture atlases (tile sheets). The idea is that you can have a number of different textures combined into one and then access the ones that you want with an index. With a texture atlas, that index will shift the UV0 coordinates to the appropriate portion of image that you want. With arrays, it pulls the appropriate texture (which are also called slices) from it's internal set.

The arrays are built by processing a list of up to 256 texture2D images at design-time. We do not currently supply the software to accomplish this since there are already quite a few very capable ones available.

See some of the texture2DArray tools that we recommend here.

StuctureBoneZ currently ship with shaders that use albedo and normal maps. So two arrays are required for each material. All textures in either array must be of the same pixel size and will all be compressed into the same format. We recommend the albedo textures be compressed with DXT5 format and the normals to be compressed with BC5 format. This compression will be applied during the texturearray creation process so you do not need to adjust the Unity import settings beforehand.

We ship with all textures sized to 1024x1024 (1K). You can choose other sizes but be aware that the arrays can be quite large depending on how many textures you include in them.

It's best to use higher quality source textures to create the array. Lossless format (PNG, BMP, TGA, TIFF) are preferred. It is tempting to use .JPG for the much smaller file sizes but you will lose a lot of graphic fidelity right from the start. Also, keep in mind that Unity will decompress the files anyway when it creates the texture arrays and the file size will increase in that decompressed format. But when you build your project, those assets will be compressed into optimal form. So your original source textures will not be included in the built application (game), and the texture arrays will be much larger in your project folder than they end up in the standalone built game.

It is certainly possibly to make more elegant shaders for these structures and as always there will be trade-offs between visual quality, performance, memory usage, preparation time, and file size. We ship with basic shaders and will leave the more expensive shader production for future development. Ones that handle PBR, wetness, snow cover and such are all possible.

StructureBoneZ do not depend on texture arrays but default to them for extremely fast texture edition and their natural propensity to be performant. They also create a nice organizational aspect to your texturing workflow.

PREFABS

If you get a particular building the way you like and want to spawn it programmatically or move it into another project or scene, you just “Bake” the edited mesh data and prefab into a folder of your choosing. This is needed because meshfilters in prefabs must have access to mesh assets that are saved to disk. Simply dragging and dropping the structure parent object into Unity project folder will create a prefab as normal but it will lose the link to it's mesh data.

When you edit an SBZ mesh with our painter app, it creates a new mesh altogether for the object that is changed. The original FBX files are left alone, and the mesh in your scene are new ones. These are saved in the scene data and will persist through scene changes, application shutdown, and game builds. We provide an easy way to “Bake” the changed mesh data to disk while creating a prefab that links to them.

Just place an SBZ_ID component on your structure root object. This is only to point out to the editor what object actually is the root.

Then, use SBZ_Painter to select any one of it's objects. Click “BAKE PREFAB”, point it to a folder in your project, and it will create the prefab and mesh assets needed there.

Only the mesh objects that you change are saved as new mesh assets. For instance, if you have a building and change only the roof textures, then bake the prefab; only the new roof mesh will be added to disk. All other mesh filters in the structure will still link to their original mesh assets.

You can select “include all unchanged mesh” to bake every mesh anew and save them into the new prefab. This would be very handy for migrating the prefab into another project.

Once the prefab is baked, it can be placed into scene as normal at design or run-time.

IMPORT / EXPORT

IMPORTING FROM UNITY TO BLENDER

The original mesh assets that we supply are in FBX format. Accordingly, they can be edited with practically any modeling software. We primarily use Blender for this purpose so any tips that we offer will be Blender specific.

All objects share a common origin and are scaled to 1, rotated to 0, and located to 0. This makes it easy to import, edit, and export back into Unity. We do not include interactive objects (doors and lights) in our blender structures. They keep their own individual origins and are added to the structures in Unity.

We use Imperial Units (inches/feet) so the measurements may make more sense if you do the same. In Blender, choose Properties window->Scene->Unit System->Imperial

To edit a single structure object, you can import the FBX models into Blender as normal, make changes, and export back out overwriting the original or saving as a new model.

To import an entire folder from Unity project in one whack, there are brute force ways and more elegant ways:

BY BRUTE FORCE

The supplied FBX files can be drag-dropped into Blender one-by-one and will keep their relative transforms just fine. But this can be tedious, especially with organized collections in Blender and folders in Unity. Also, you have to drag them from Windows explorer instead of Unity. As of this writing the Unity drag and drop of FBX into Blender does not seem to work well. So importing every object into Blender can be very tedious.

For a more elegant solution, it takes some Python code or a blender add-on. Visit the DensRekr website or Discord channel for additional info.

EDITING IN BLENDER

Since all objects share a common origin, working on them individually may be awkward. In this case, you may want to switch the origin for a given object to it's own center while editing. In Blender, from Object mode, choose 'Object'->'Set Origin'->'Origin To Geometry'. Just remember to set it back to the common origin before exporting back to Unity. This is done from object mode by selecting the object, hit Ctrl-A, then 'All Transforms' to Apply all Transforms.

If you modify an object's mesh or add a new object, then you will need to wrap it's UVs (initialize it's UV0 coordinates)

In Blender:

in object mode

select the object

click on 'Object Data Properties'

select the first UVMap

switch to Edit mode

select all

then choose UV from the menu and unwrap the UVs using one of the given methods

we normally use 'Box Projection Unwrap'

to set the UV0 scale. We normally go into UV editing workspace, select all of the generated UV coordinates, hit 'S' to scale, and .1 <enter> to scale them to 1/10th of original. This is optional and will affect the base size of the textures applied to that object before they are scaled by SBZ_Painter.

If you edit or add a mesh object in Blender that you plan to use with SBZ_Painter back in Unity, then don't assign more than one material to it. This will create a submesh and SBZ_Painter will refuse to edit submeshed objects.

EXPORT BACK INTO UNITY

You can modify any of the FBX files which represent a single object (ex: Roof, Exterior Walls, Floor, etc), and export your changed file directly back into Unity. If that mesh object has already been spawned into scene, and hasn't been “painted” by SBZ_painter, then changes to the data will be reflected in scene.

Any objects in Unity that have already been “painted” will not be associated with the original FBX, so changes will not show up in scene automatically.

Exporting Individual Objects

If you have just edited a single object mesh and saved it back into Unity, then you need to apply the new mesh to your structure. To update the building you are working on, select the object in Unity that corresponds to the one you have edited, find it's meshfilter component, drag and drop the new mesh from the updated FBX mesh into it. The new mesh will have lost it's texturing information, so then you just re-paint it as desired with SBZ_Painter. ***NOTE when you swap out a mesh in this manner, also put the new mesh into the gameobject's meshcollider, or better yet ---> remove the mesh collider and add a new one.

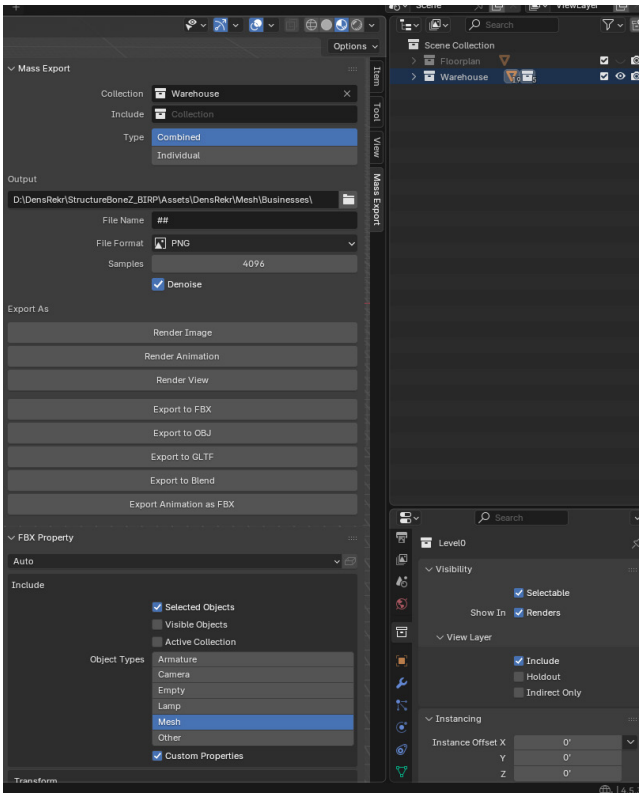
Exporting Entire Structures

For mass export, we use a Blender extension: Mass Export by ThreeDee.design

This add-on allows you to specify the folder where the mass export will take place and it will export into a folder structure based on the collections in your Blender project.

Since this folder info remains saved with the Blender file, you can make changes, then re-export directly back into the Unity project overwriting the original files with a single click.

If you add new objects, you will need to then select those files in Unity and set their import settings.



BLENDER EXPORT SETTINGS

For FBX export (whether individual or mass) choose the following settings:

Object Type: Mesh

Scale: 1.00

Apply Scalings: All Local

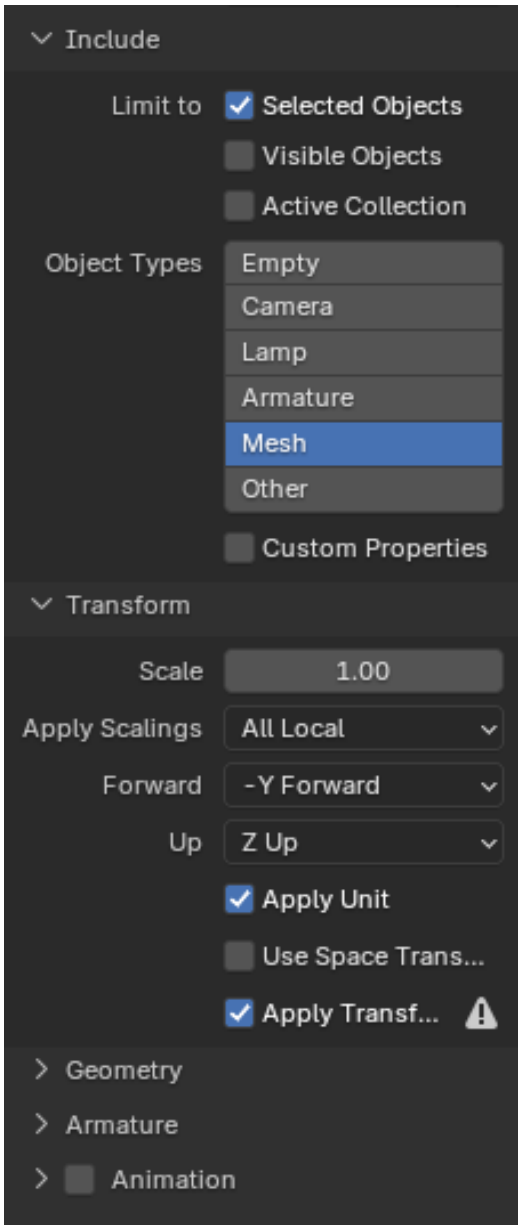
Forward: -Y Forward

Up: Z Up

Apply Unit: True

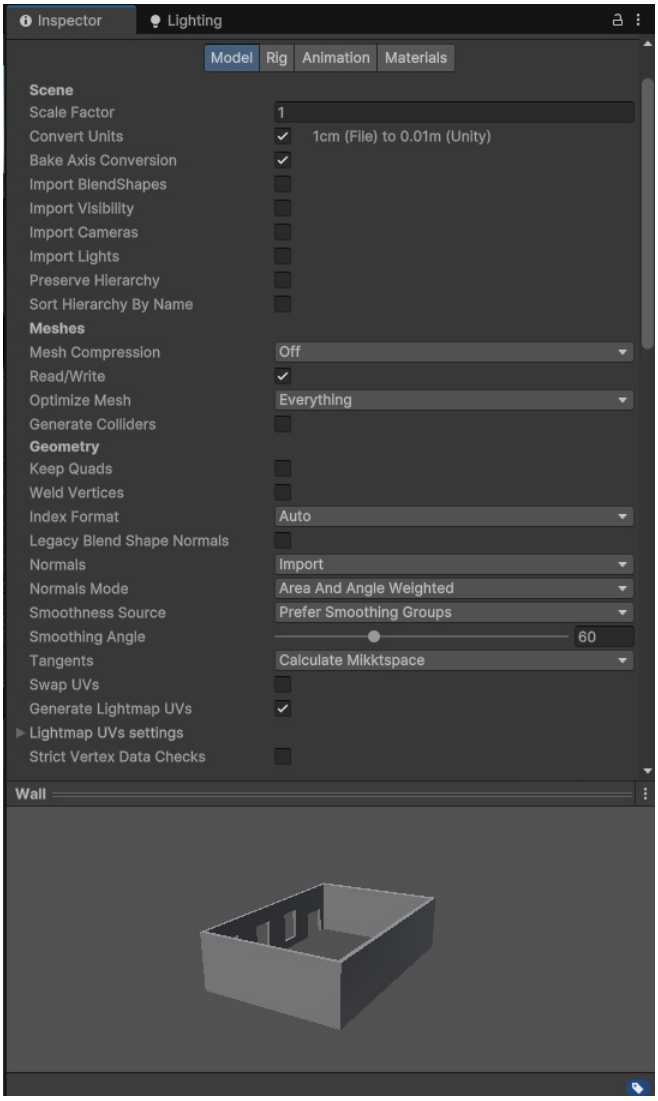
Use Space Transform: False

Apply Transform: True

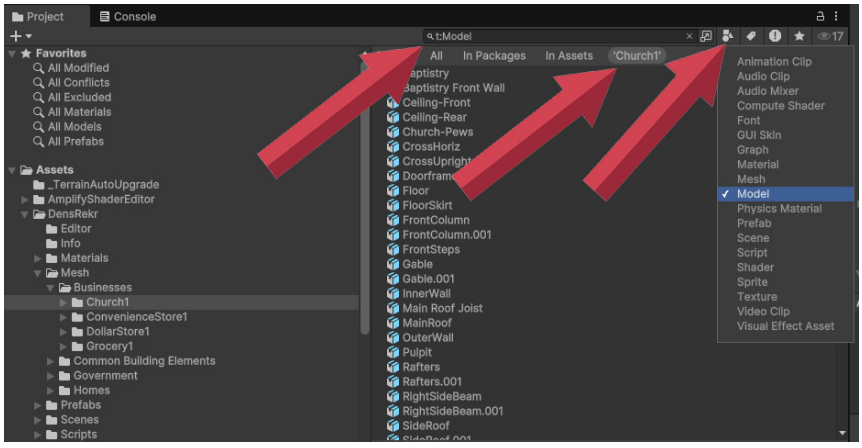


UNITY IMPORT SETTINGS

Here are the import settings we use to import from Blender to Unity. The most important one is “Bake Axis Conversion” which needs to be set to true. This is what orients the mesh correctly with it's transform rotation set to 0,0,0



If you do import a whole folder structure containing all of the various structure objects in organized folders, it can be handy to select them all at the same time. For this, you can use the project inspector search-bar as shown.



LIGHTING

To get the structures to respond well to scene lighting, we format our albedo texture arrays to DXT5 compression, and the normals texture arrays to BC5 compression.

While editing your structures in the sceneview, it may be difficult to see interior items because of shadows. In this case, go to <Window><Rendering><Lighting><Environment> and increase the “Intensity Multiplier” in the Environment Lighting section. You'll want to change it back for gameplay as this will wash away most of the scene's lighting effects.

Any mesh objects painted with metallic setting true will need to have a reflection probe in the scene covering it to provide reflection data. If you leave the probe set to “Baked” or “Custom” remember to hit the “Bake” button after sizing and positioning it the way you want. If you have metallic triangles or objects that are not covered by a reflection probe, they will probably appear very dark no-matter what lights are shining on them.

Our tint function does not limit the RGB values to a max of 255. So if there is errant data in the UVs (maybe written by a modeler) it can cause the tint to go beyond the standard 0-255 values and the mesh may appear super bright, like it is emitting extreme white light. If this happens just set the tint to a more normal range and paint the “Entire Object”.

If you zoom in very close to low light interior walls, and see shadow artifacts that seem to move with the camera, reducing the wall smoothness to zero will probably help. Also, in URP or HDRP, possibly increase the shadow resolution and decrease shadow normal bias and depth bias. These adjustments also help with bright lights leaking through corners of buildings (which is a well known lighting problem inherent to many game engines)

When two surfaces meet (for instance where a wall meets the ceiling) the human eye expects some darkness in that corner. This effect is Ambient Occlusion and can be turned on an adjusted in the Lightmapping settings of the Lighting inspector.

We are by no means Unity lighting experts so we hope to learn from our customers how to set these (and other aspects of Unity) up better and will continually amend these recommendations as needed.

In particular, configuring lights in HDRP to get interior/exterior transitions correct is very challenging and beyond the scope of what we currently offer.

OPTIMIZATION

With StructureBoneZ, we try to strike a good balance between visual quality, performance, versatility, intuitiveness, and edition speed. As always there are trade-offs. Here we address some of these optimization issues.

For LOD, frustrum, or occlusion culling, we try to combine and group all mesh in the most advantageous way. That's one reason we always separate the interior walls from exterior ones. It is a simple matter to place the interior walls, doorframes, baseboards and other interior details into LOD groups that cull all these renderers until the player gets close.

Sample windows and doors are supplied but not mesh combined. That's because you may want to open or break them individually and you may want various styles according to your game art. In super low poly style, you may choose to have no window panes or doors at all, whereas in realistic style, you may need to add reflection probes to cover the windows and have various style doors that swing in various directions. So for these, we provide only low cost samples to get you started and leave them each as separate gameobjects.

HELPFUL TOOLS AND RESOURCES

Texture Array Edition

Amplify Shader Editor (Amplify Creations)

This is a visual node-based, full featured shader creation program that also happens to ship with a texture array creator.

https://assetstore.unity.com/packages/tools/visual-scripting/amplify-shader-editor-68570

Texture Array Essentials (William Schack)

Tool specifically for texture arrays. We like the fact that this tool allows update of arrays without having to create them from scratch each time

https://assetstore.unity.com/packages/tools/utilities/texture-array-essentials-306888

MicroSplat, MegaSplat (Jason Booth)

These tools are typically used for terrains but have historically included generic texture array creation/ edition tools as well

https://assetstore.unity.com/search#q=jason%20booth

MeshBaker (Digital Opus)

Meshbaker specializes in optimization through the use of texture arrays, texture atlases, and mesh combination. It has tools to work with these and provides a wealth of info regarding asset optimization.

https://assetstore.unity.com/packages/tools/modeling/mesh-baker-free-31895

MODELING

Blender

https://www.blender.org/download/

Mass Export add-on by ThreeDee.design

https://threedeeshop.gumroad.com/l/mass-export

Folder Import addon by DensRekr (not commercially available. Contact via Discord for info)

ambientCG.com

website: https://www.ambientCG.com

CCO license: https://docs.ambientcg.com/license/

polyhaven.com

website: https://polyhaven.com

CCO license: https://polyhaven.com/license

SUPPORT

We welcome questions, concerns, suggestions, and comments regarding everything DensRekr is doing. Please use discretion on our product reviews as they will certainly affect sales. We hope to really do a great job here and offer ongoing improvements to the tools, information, and models that we provide. Please give us the chance to address concerns or problems on our Discord channel before posting bad reviews.

ROADMAP

We plan to build a large repository of buildings to populate and customize a modern era world. That is our main goal. We will seek contribution from a growing number of compensated artists to achieve this, most likely drawing from our customer base. If you would like to help, please contact us via Discord.

The SBZ_Painter is a tool that we created to make rapid customization of very performant buildings possible. Here are some of the possible coming improvements to it:

- Add feature rich shaders and adapt the painter to handle them
- Add function to initialize entire structure (set all UVs in every object to default values)
- Lookup and display Structure name in header if it has SBZ_ID attached
- Ability to reverse normals per triangle

Shadergraph versions of the materials.

Full featured “Interactive Detail” addons (to be sold separately) for each building.

- These may include:
- properly placed lights, light fixtures, switches
- various styled working doors
- cabinets, mailboxes, drawers, closets that open
- ladders and climb spots
- working security systems and cameras
- spring return and automatic doors
- audio effects

FAQ

Q: What if I would rather use individual materials?

A: In URP and HDRP it is not so important to minimize material count. Since they rely on SRP batching for optimization, you can just use minimal shader count. So , if you prefer, you can carve up each mesh to use different materials where needed and apply them as normal in unity inspector.

StructureBoneZ mesh are already carved as strategically as possible. So if you would rather use other materials and need to split up the surfaces (for instance, to apply different wallpaper or flooring to a specific room) then those triangles are already segregated from the other rooms and can be assigned without having to further split the mesh up. You would just use your favorite modeling software to assign new materials to those mesh faces and save back into project. Those mesh faces will be saved as a separate submesh, and allow the application of a different material in Unity. We do not pre-assign materials to the structures like this because we have no way of knowing which areas you will want to be distinct.

GLOSSARY

CC0 License: CC0 (aka CC Zero) is a public dedication tool, which enables creators to give up their copyright and put their works into the worldwide public domain. CC0 enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, with no conditions.

UV: texture coordinates that allow a 2D texture to be “mapped” onto a 3D object. The “UV” name doesn't stand for anything in particular, it is just used instead of X,Y to represent those 2D coordinates keeping it distinct from the mesh vertex positions

Shadergraph: Unity's node-based shader creation tool

SRP Batching: An optimization technique whereby Unity persists material data on the GPU and uses dedicated code paths to quickly update Unity engine properties in a large GPU buffer. This greatly speeds up rendering even when there are many different materials. To gain maximum benefit, though, as many materials as possible need to use the same shader. Mesh objects are not combined, but rendering is much faster. This is only available in URP or HDRP. Can be used alongside static batching.

Static Batching: An optimization technique whereby Unity combines mesh assets at runtime to reduce communication cost between CPU and GPU in order to render each frame. Gameobjects must be marked as static to join in on this technique and only mesh with exact same materials can be combined together