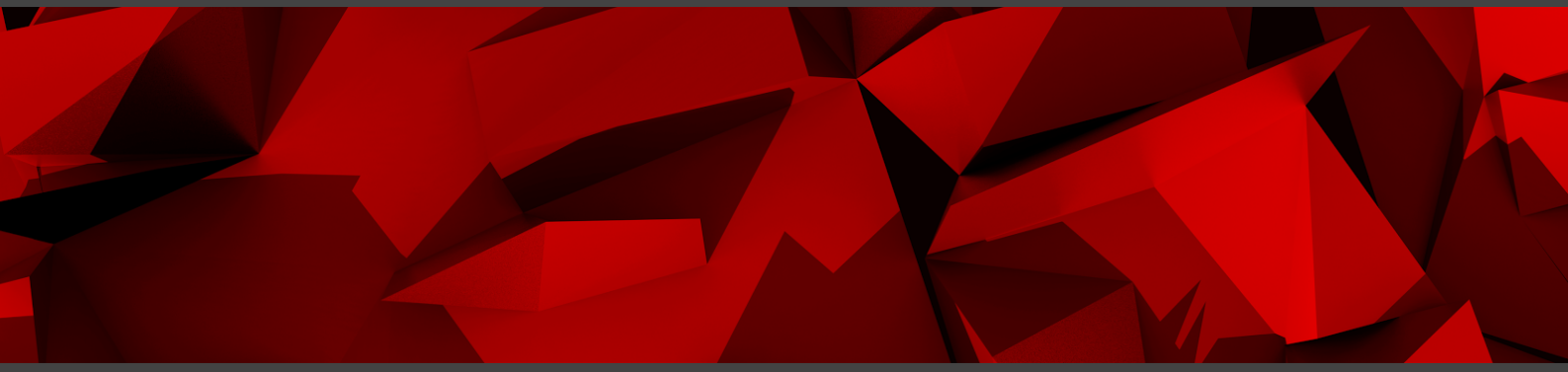
VertPaint - Documentation



Overview

VertPaint is a simple little tool for painting vertex colors onto the meshes in your scenes. You can paint, erase, modify, fill and invert the vertex colors of a mesh with intuitive controls and good performance. For more information about the controls and general usage of VertPaint, check out the help section foldout inside the VertPaint window. Furthermore: all VertPaint settings are equipped with a descriptive tooltip, which can be made visible by hovering your mouse cursor over the label for a few seconds.

The entire codebase is written in well commented C#, featuring detailed and extensive xml documentation compatible with IntelliSense.

If after reading through this documentation and watching the YouTube tutorial video series you still have questions, problems, bug reports, or any sort of need for a VertPaint related support, just feel free to write me an email with the details in it! I always try to answer asap. And by all means, if you think a specific feature should definitively be in this asset, do let me know! I always give my best at trying to implement user recommended features in future updates as good as I can :)

Getting started

To start painting vertex colors on a mesh, select the desired mesh inside the scene or hierarchy view and click on **Window/VertPaint**; you can also use the keyboard shortcut **Shift+Control+V** (or respectively **Shift+Command+V** on mac). The main VertPaint interface will show up. In this window you can customize your VertPaint brush properties such as radius, falloff, opacity and so on...

To find out more about what each setting does in detail, check out the corresponding sections in this documentation.

Now, if you move your cursor over the selected mesh, you can already see the circle brush appear on the model's surface (you can customize the brush style and alpha in the VertPaint brush settings). The default key for painting the specified vertex color is the left mouse button (this can be changed under the key bindings section).

If you want to change the mesh you're currently working on, keep down control (command on mac) and select the new mesh inside the scene view with your mouse. Note that any unapplied painted vertex colors on the previous mesh are going to be discarded, so apply the changes first before modifying your selection if you don't want to lose that data.

Closing the VertPaint window will also discard any unsaved changes.

The VertPaint Window

This section is dedicated to the VertPaint window itself and features a detailed list of all the available settings and controls.

Main toggles

- Enabled
    - This toggle enables/disables the VertPaint brush entirely; a disabled brush won't appear inside the scene view and won't do any harm.
- Toggle preview
    - Choose if the preview vertex colors key should be toggled on/off or held down to preview the painted vertex colors.
- Hide transform handle
    - Hide or unhide the transformation handle gizmo (the three XYZ arrows) inside the scene view.

Brush settings

- Radius
    - This is the brush radius in meters.
    - The float field right next to this slider is the maximum radius field, which determines the boundaries for the radius slider.
- Delay
    - The delay value determines the minimum amount of time (in seconds) that has to pass between paint strokes.
    - If you notice that the vertex colors are being applied too quickly and intensively when painting, try to increase this value and see if that helps the precision.
    - Usually, the default value of 0.05 seconds is pretty good.
- Opacity
    - The opacity controls the maximum intensity of the painted vertex colors.
    - Maximum opacity will result in fully opaque (full alpha) colors, whereas a value of 50% would halve the intensity of the colors (thus making each paint stroke blander).

- Falloff
    - The falloff curve controls the opacity of the painted vertex colors in relation to their distance to the brush's center.
    - t = 0 represents the center of the brush.
    - t = 1 translates to the outer edge of the brush area.
    - By default, this is a linear falloff that circularly fades out the opacity the further away you go from the brush's center.
- Color
    - The vertex color to paint.
    - Common values are full red, green or blue, since most shaders use a blend in between those to produce the material blending fx.
    - The buttons next to this field are shortcuts for the above mentioned common values (except for the 1-x button, which inverts the selected color value).
- Style
    - This is the esthetic appearance of the brush inside the scene view. It has no influence on the functionality of VertPaint's brush, it just allows you to choose how you want your brush to appear.
    - Sometimes it makes sense to change the brush style to the spherical style, like for instance when you're painting colors on concave meshes like tunnels, caves, etc...
    - It also makes sense to use the disc style with high alpha values for bad contrast scenarios where it's hard to see the brush on the underlying surface.
    - Having no physical appearance at all can also be great, for example when you want pin-point precision: if you want to apply only small amounts of color to single vertices it's best to enable the selection wire and set the brush style to "none".
- Alpha
    - The alpha of the used brush (0 = fully transparent, 1 = fully opaque). This only affects the appearance of the brush!

Key bindings

- Paint key
    - The key for painting vertex colors (hold down shift to erase). This can be either a mouse button or a keyboard button.
- Modify radius
    - The keyboard shortcut for adjusting the brush radius dynamically from inside the scene view.
- Preview vertex colors
    - Choose the keyboard shortcut for previewing the painted vertex colors.

Templates

- Autosave directory
    - Click on this path to select a new directory where VertPaint should deposit its autosave files.
    - Unless you want to move the root VertPaint folder around in your project, it's best to leave this setting alone.
- Save button
    - With this button you can save your current VertPaint configuration out to a VertPaint template file for later reuse.
    - VertPaint templates are standard, well-formatted .xml files.
- Load field/button
    - This is both a drag 'n' drop field and a button: if you click it, you can select the VertPaint template to load with the file selection dialog. If you don't want to select it like that, you can also directly drag the template file into this field and load it this way.
    - Loading a VertPaint template overrides the current configuration.
- Add favorite template field/button
    - The blue plus icon above the favorite templates list is also both a field and a button. Click or "drag 'n' drop" a template into it to add your favorite templates to a list that facilitates access and quick-loading VertPaint templates.

- Remove last favorite button
    - The minus button to the right of the above mentioned plus field removes the last favorite template from the list (from the bottom).
- Clear button
    - This button clears the favorite templates list (removing all its entries).

Mesh output directory

- This is the output directory where VertPaint will deploy the mesh assets containing the applied painted vertex colors.
- Each mesh asset is placed inside a subfolder named after the scene in which the paint action occurred.
- Click on the path to change it (like the autosave directory).

Quick-action buttons

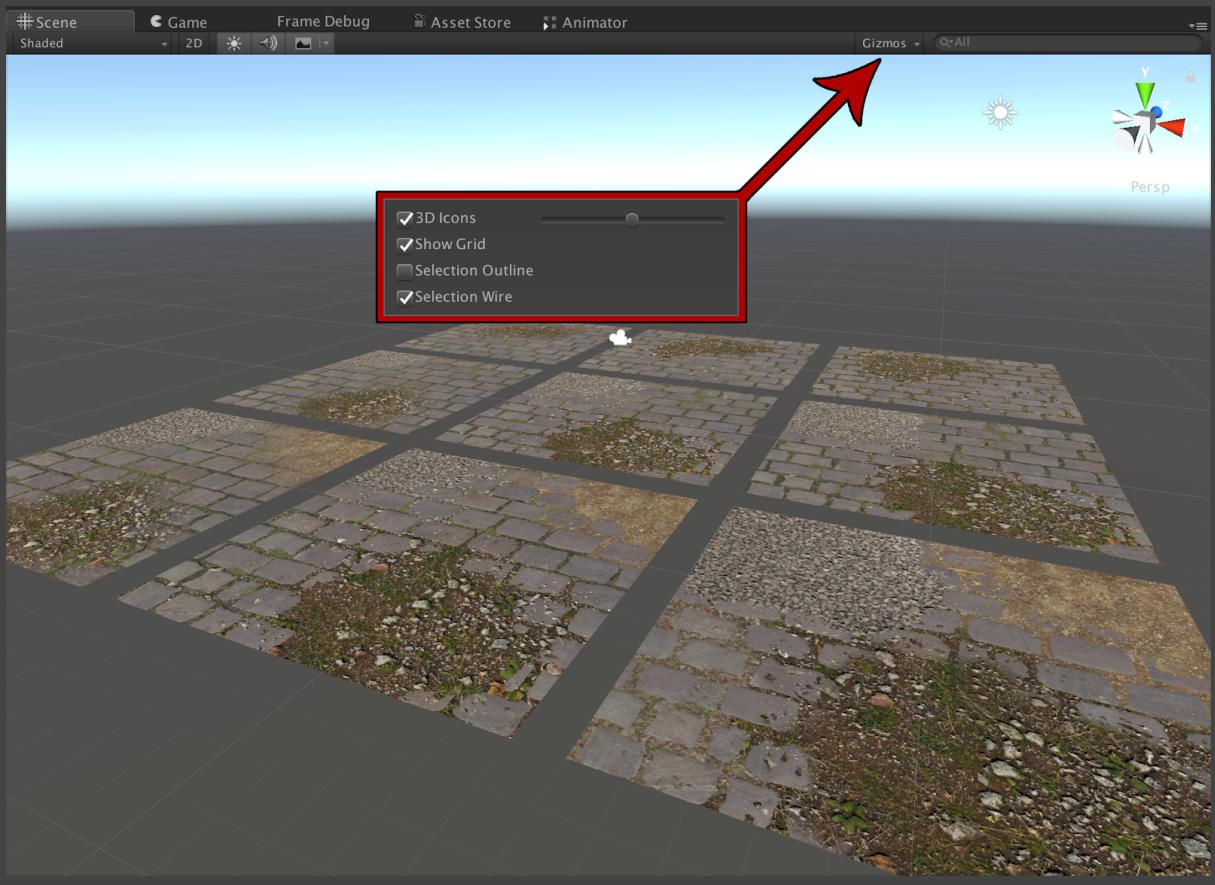
- Fill
    - Fills all vertex colors with the current brush color. Hold down shift to fill with clear black (0,0,0,0).
- Invert
    - Inverts all of the vertex colors on the selected mesh with one click on this button.
- Discard
    - Discards the changes made to the vertex colors of the selected mesh.
- Apply
    - Applies the painted vertex colors by saving them out to an asset on disk (inside the specified mesh output directory).
- Clean up
    - Automatically cleans up the mesh output directory by moving unreferenced and unneeded mesh assets in it into a subfolder called "_old".

- Reset
    - Reverts all VertPaint settings back to their default values, except for the favorite templates list (which you can clear with the C button if you want).

Painting vertex colors

To paint vertex colors on a mesh, select the mesh whose vertex colors you want to modify, open the VertPaint window, set up your brush and start painting (the default paint key is the left mouse button). The specified brush radius, falloff, opacity, etc... will be used to apply the selected vertex color.

If the Unity mesh selection wireframe or the orange outline disturb you, you can easily toggle them on or off in this menu here:



Note that this utility will only paint vertex colors, not materials or textures directly! If you want to blend materials with vertex colors, check out the "using the provided VertPaint shaders" section further down below.

Painting vertex colors on meshes that don't have a shader that actually uses vertex colors in some way won't yield any visual feedback whatsoever: you'd only see the vertex colors by previewing them in the scene view (default key for that is C).

Keep this in mind when painting on meshes with VertPaint and wondering why nothing happens so far.

To preview the painted vertex colors in their raw form inside the scene view, keep down (or press, depending on the state of your toggle preview setting) the preview vertex colors key. Make sure that the scene view is active (right-click on it to activate it) in order to preview vertex colors or modify the brush radius with the keyboard shortcut.

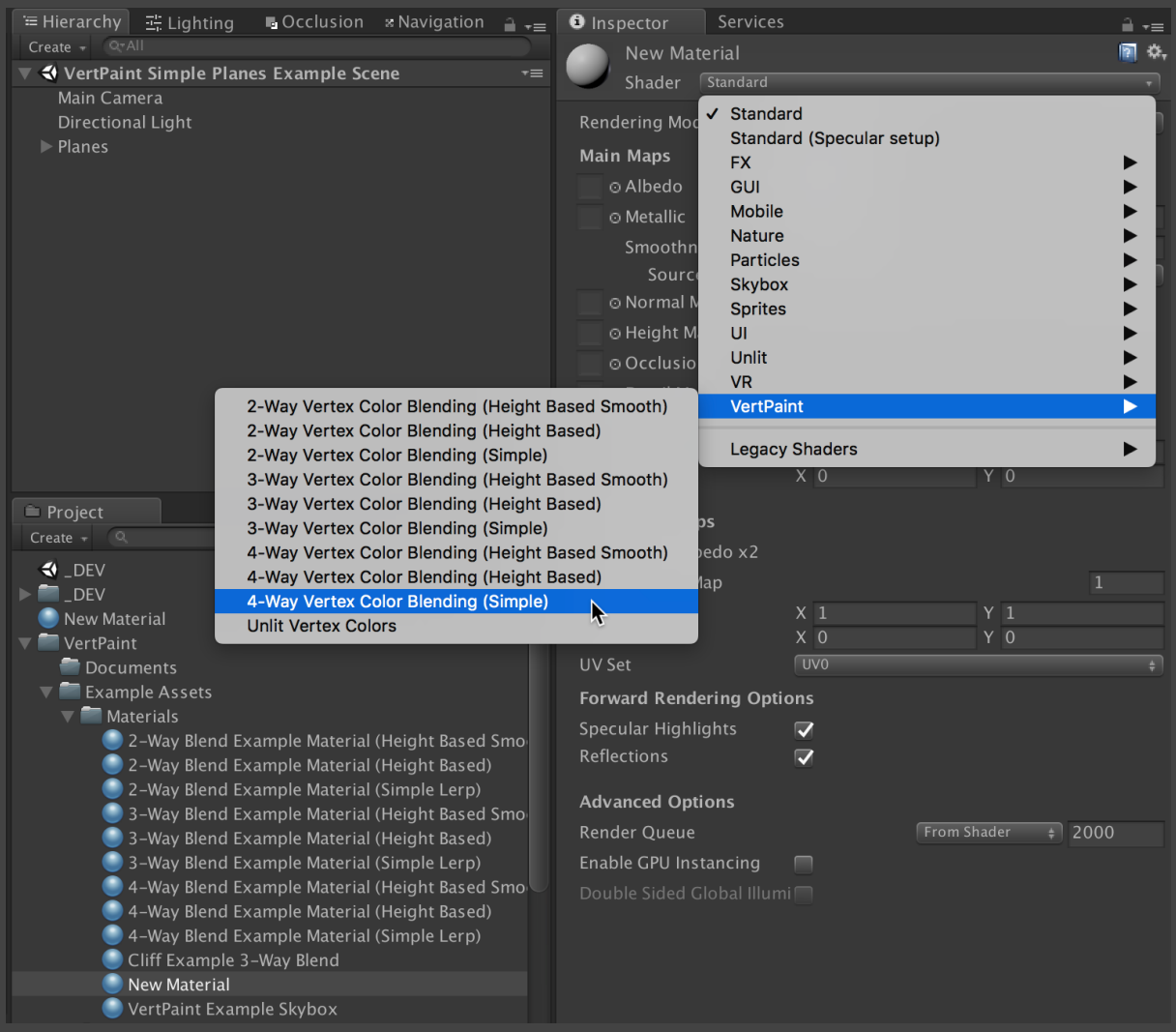
Speaking of brush radius modification: you can do that comfortably inside the scene view without having to switch over to the VertPaint window by holding down the radius mod key (default is X) and dragging the mouse cursor to adjust the brush radius.

When painting vertex colors, sometimes it happens that we exaggerate or just don't get the result we're looking for. Simply put: we need to erase what we've painted! This can be easily achieved by holding down **shift + the paint key**. Sometimes we even want to start from scratch, which can be done by using the fill button under the quick-action buttons: keep shift down when clicking on fill, which will result in clearing the vertex colors (setting their RGBA to clear black (0, 0, 0, 0).

Using the provided VertPaint shaders

So now you know how to paint vertex colors on your meshes with VertPaint. But why would you want to do that in the first place? If you're reading this, it's pretty obvious that there certainly is *some* reason why you want to do that. And the most common one is almost always blending together various materials on a mesh based on vertex colors. To achieve that material blend effect, we need shaders; custom shaders that read the meshes' vertex color data and apply albedo, normals, etc... in a fancy way.

You can find example materials using the VertPaint shaders in the *VertPaint/Example Assets/Materials* folder. To create a new material with a VertPaint shader, choose one of the included shaders from the VertPaint shader category.



If you take a look at each of the VertPaint-shaded materials' custom inspectors, there is always a "Background layer". That's the base material on top of which all the other layers blend in (from top to bottom => red to green to blue). If you combine the VertPaint painting utility with the usage of the VertPaint vertex blend shaders, you can see the effects of your painted vertex colors live on your active mesh (the textures are painted on the mesh in real-time). This makes level designing so much more fun and easy!

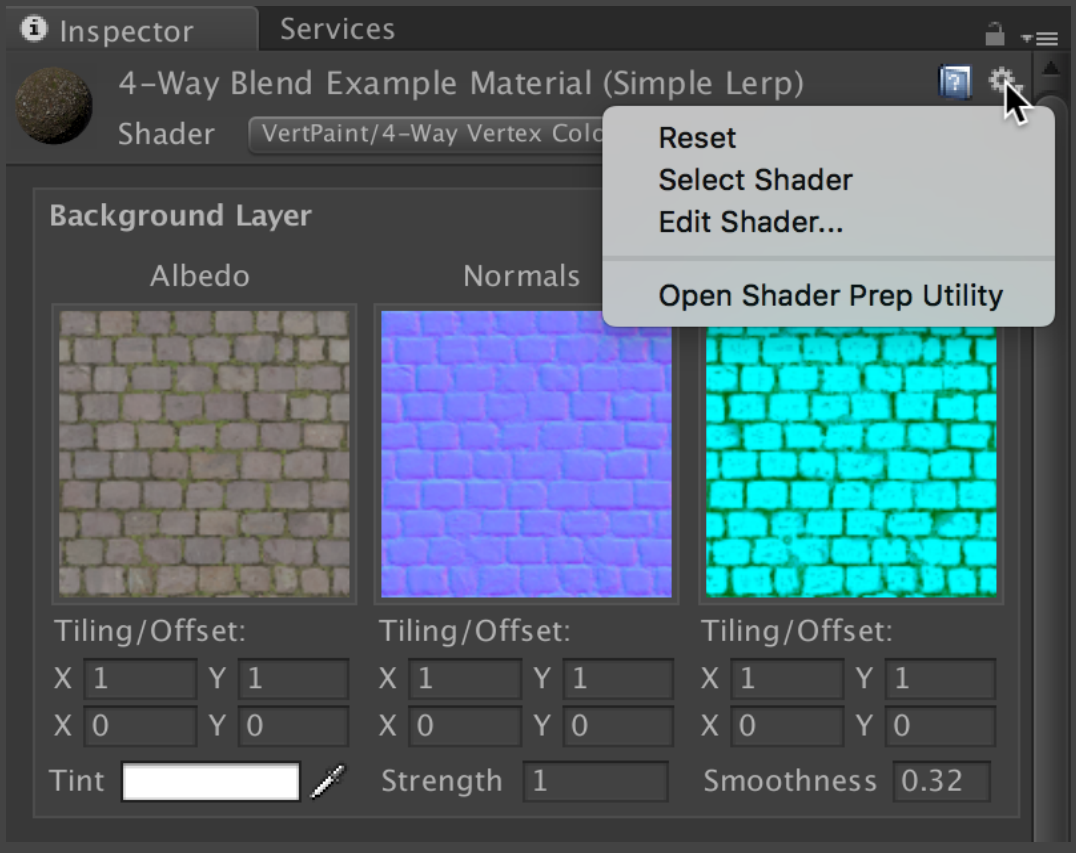
At the bottom of each VertPaint material inspector there's the "Open Shader Prep Utility" button. What that is and how it works is explained in the next chapter.

The Shader Prep Utility

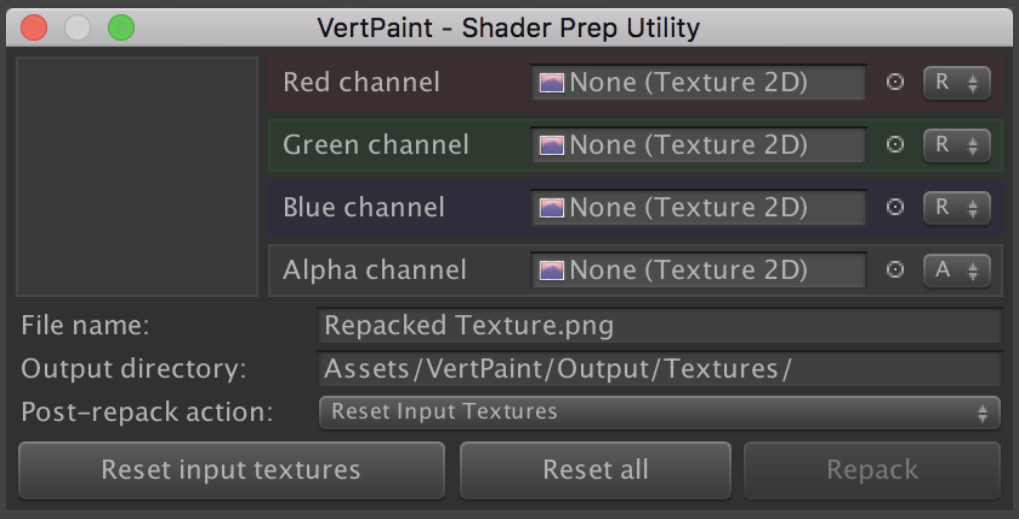
If you have your texture maps set up in a manner that is not compatible with the provided VertPaint vertex blend shaders (or any other custom shaders for that matter), don't worry! There's an included shader prep utility that preps your awesome textures and repacks them into the desired configuration to make them ready for any kind of shader.

You want to repack the metallic channel from one map into a new texture's red channel, and add the smoothness map from another source into the green channel? Too lazy to open GIMP or Photoshop and do the tedious work manually? No problem!

Just open the shader prep utility by clicking on the gear icon in any material in the project and → "Open Shader Prep Utility":



Another way to open the shader prep utility is to click on the "Open Shader Prep Utility" button that you can find at the bottom of all material inspectors that use one of the included VertPaint shaders.



The way this works is that you drag the textures whose channel(s) you want to remap into the red, green, blue or alpha channel fields respectively and select the source channel on the right. A perfect example for this would be the following:

- You have a set of textures that are configured for the Unity standard shader, ergo an RGB albedo map, a metallic(R)-smoothness(A) map, an RGB normal map, a height map (G) and maybe an ambient occlusion map (G).
- You want to have an MSHAO map that includes all of the above mentioned complementary texture maps in one single metallic-smoothness-height-ao map to use inside the VertPaint shaders (albedo and normal maps stay the same).
- You don't want to open an image editing program to repack the textures manually.

In this scenario, you'd drag the texture containing the metallic map into the "Red channel" field and select the metallic source channel (e.g. if the metallic map is in the red channel of the source texture, select R on the right hand side).

Then drag the texture containing the smoothness map into the "Green channel" and select the channel where the source smoothness is located (in the case of Unity standard shader textures, it's the alpha channel of the metallic map => select A).

If you have a heightmap, put it into the "Blue channel" field. Unity standard shader uses the green channel for that, so put the source channel selector to G.

The same goes for the ambient occlusion map (put that one into the "Alpha channel").

Now you can give a name to the final repacked texture file you're about to create: the repacked texture will be placed into the specified destination output directory.

Choose what happens after the repacking is done with the "Post-repack action".

Hit "Repack" and you're done! Note that this was just an example scenario, you can do anything you want with the shader prep utility and adapt it to your needs.

Structure

The VertPaint codebase is structured as follows:

- VertPaint (Namespace)
    - Classes
        - EditorWindows
            - ➢ VertPaintWindow
            - ➢ ShaderPrepUtilityWindow
        - GUI
            - ➢ Classes containing static icon texture variables and GUI-related methods
        - MaterialEditors
            - ➢ Custom inspectors for the included example shaders
        - EventArgs
            - ➢ Various EventArgs classes that are passed to the event subscribers upon invocation
    - Enumerations
        - BrushStyle
        - ColorComponent
        - PostRepackAction

Extensibility

VertPaint's source code is fully accessible and can be extended as much as you want. Feel free to experiment around, read through the code, see how everything works, etc...

You can derive from VertPaintWindow, subscribe to its public events, change its behaviour, and so on...

Nothing stops you from changing the code to make it work best for you and to integrate it into your project(s).

If you use Visual Studio, all the methods and fields have proper xml documentation that is fully IntelliSense compatible.