



The **PolyPaint Pack** allows you to paint directly on the mesh of your objects, without the need of bitmaps. Each vertex of the object will have a RGBA value associated with it. Yes, you can also paint Alpha information with PolyPaint.

It also allows you to load the RGBA information stored in a special kind of .OBJ and .FBX files.

It includes a set of tools to allow the creation and editing of all those RGBA values.

At the end, you can also render objects that carry that extra RGBA information.

PolyPaint Pack is made up of six plugins.

- **PolyPaint Tag** is a tag that you attach to any object that you want to add RGBA surface information to. It can be any polygonal object, be it modelled inside Cinema 4D or loaded from any other application or internet.
- **PolyPaint Shader** is a shader that will interpret the RGBA data stored inside the **PolyPaint Tag** allowing you to render that RGBA data.
- **PolyPaint Tool** is a painting tool for painting on the surface of your object.
- **PolyPaint Transfer** is a tool for transferring RGBA surface information of an object, stored in a **PolyPaint Tag** to another **PolyPaint Tag**, attached to another object.
- **PolyPaint From Material** is a tool for transferring RGBA information from a Material assigned to the object to be stored in the **PolyPaint Tag**.
- **PolyPaint Editor** is an “invisible” plugin that allows for fast display of RGBA surface information in the editor. Don’t even bother searching for this plugin. It performs its work “under-covered”.

This is a very complex set of tools so it is advised to read this manual in order to understand what are the limitations and how and why some features work the way they do.

PolyPainting or Vertex Painting is supported by some 3D applications. Examples of such applications are zbrush, 3D Coat or Blender. They associate RGBA values with each vertex and are capable of exporting the 3D meshes and their associated RGBA values in a specially formatted .OBJ or .FBX format.

When Cinema 4D opens these .OBJ or .FBX files, it ignores all the RGBA data and only loads the geometry information.

Nevertheless, being able to use that RGBA information and keep it associated with each vertex has some advantages.

For instance, no matter how much the mesh is distorted, the mapping of the color stays attached to the mesh. In fact, there is no mapping at all. There are no projection modes, no UVs, nothing. The document does become bigger, though. A list with RGBA values must be stored with an entry for each vertex of the object and, if the object is very dense, that list can take up a lot of space.

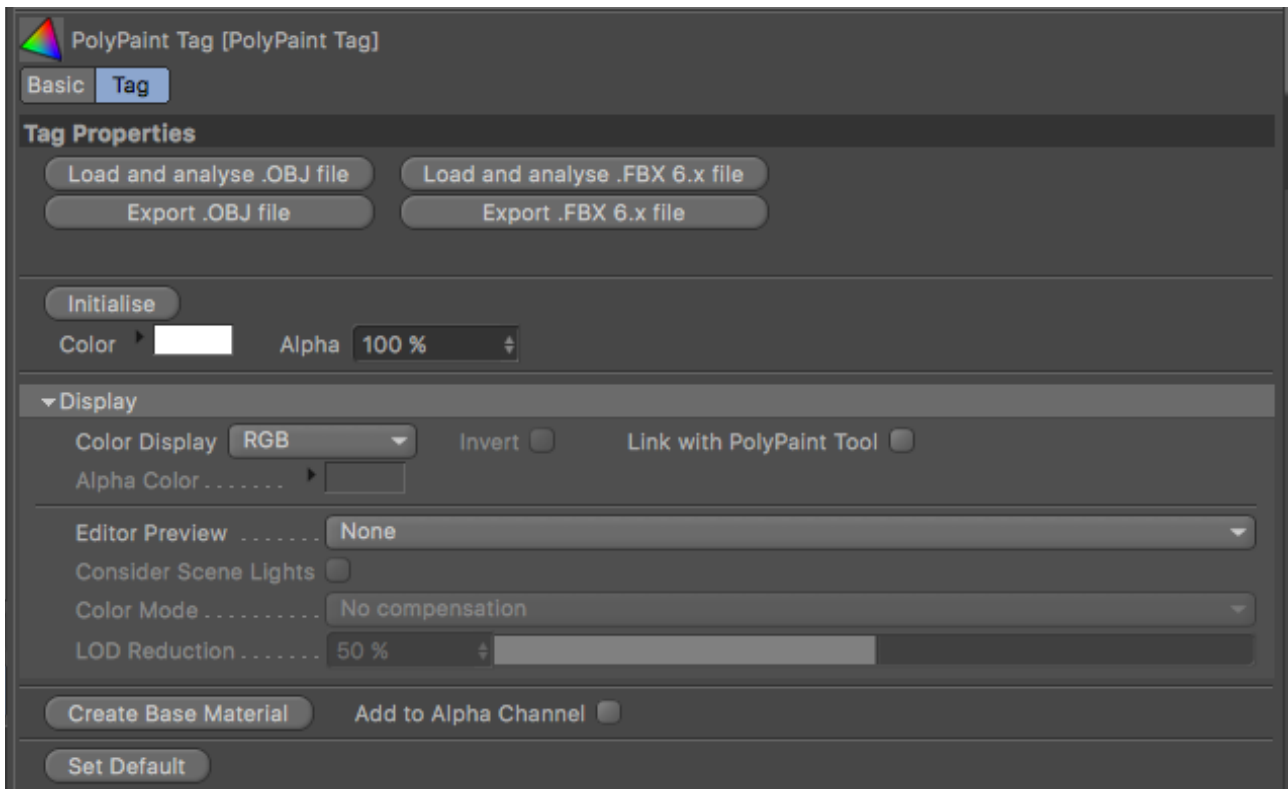
PolyPaint Tag

The **PolyPaint Tag** is the central point of the whole set of tools. This means that all other tools depend on this tag and, without it, no other tool will work.

Whether you create your own object in Cinema 4D or you import an object from elsewhere, the first thing you need to do, before using any other tool from **PolyPaint Pack**, is to add a **PolyPaint Tag** to the object.

It is only possible to add this tag to polygonal objects. If, by any chance, you add it to a parametric object, it will not work.

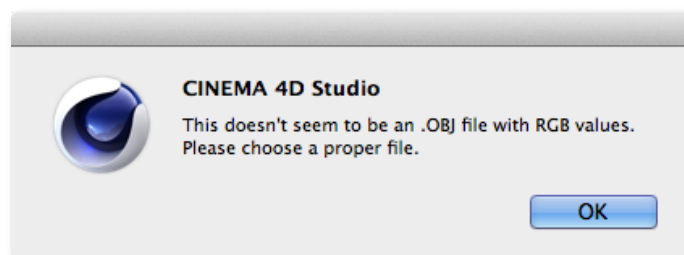
Once you add a **PolyPaint Tag** to a polygonal object, you get these parameters in the Attribute Manager:



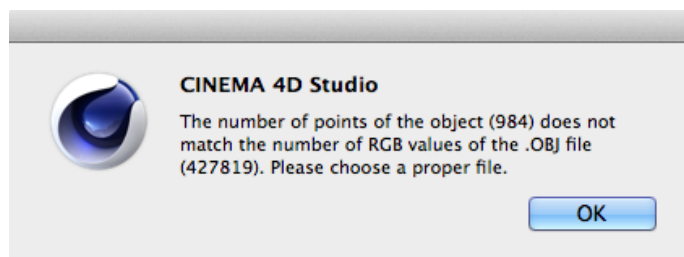
If the object with the **PolyPaint Tag** attached is an object loaded from a .OBJ file that was exported from an application that includes RGBA values for each pixel (ZBrush, 3D-Coat or Blender, for example), press the Load and analyse .OBJ file button.

After pressing the Load and analyse .OBJ file button, a dialog will open allowing you to choose a .OBJ file. You should choose the same file that was used to load the mesh into Cinema 4D.

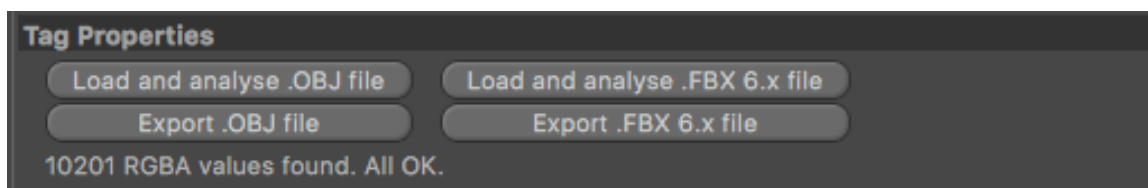
If the file you choose is not a .OBJ file, this message will be displayed:



If the number of RGBA values read from the .OBJ file does not correspond to the number of points in the object, this message will be displayed:



If all goes well, you will get something similar to this in the Attribute Manager, below the buttons:



Usually, it is advisable to import .OBJ files without optimising unused points (one of the options in the dialog that pops-up when importing .OBJ files). But, since **PolyPaint** provides information about how many RGBA values were read and how many vertexes the object has, it is usually simple to decide if importing the file again with different options is advisable or not.

PolyPaint is also smart enough to identify .OBJ files that have RGBA values coded in the two main formats and load them without the need of further information from you.

All the information given above about the importing of RGBA information from .OBJ files also applies to .FBX files.

However there are some limitations regarding .FBX files.

- The .FBX files **MUST** be in 6.x format.¹
- The .FBX files **MUST** be saved in ASCII or TEXT format.

Binary .FBX files or 7.x version .FBX files will NOT WORK (see Addendum).

Due to the way that .FBX files store the RGBA information, the **PolyPaint Tag** may have to deal with a huge amount of data. So, it is advisable that this file format be used with only simpler, smaller models.

Since the **PolyPaint Tag** will only affect the object it is attached to, with a .FBX file made up of multiple objects in a hierarchy the **PolyPaint Tag** will NOT work, therefore the .FBX file **MUST** be a single object in order to work. So, another limitation is:

- The .FBX files **MUST** contain a single object, not a group/hierarchy of objects.

¹ Very important!!! Check out the Addendum, at the end of this manual.

The inclusion of the .FBX import option is mainly for Unity users that have old .FBX files but any software that can export files in .FBX format (in version 6.x and in ASCII/TEXT) can be used to send files to the **PolyPaint Tag**.

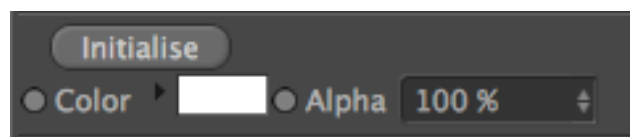
Below the **Load** and **analyse** buttons are the **Export** buttons. Each one will export, respectively, the object whose **PolyPaint Tag** is attached to, in .OBJ or .FBX format.

Remember that, if you decide to export in .FBX format, the file will always be exported in version 6.x and in ASCII/TEXT format. (see the Addendum at the end of this manual)

Also, the exporting in the .OBJ format will always occur in the format used by zBrush and 3D Coat (the one that codes RGBA values in lines prefixed with MRGB). Exporting in .OBJ format will also export a .MTL file. But this file could usually be ignored.

However, if your object is not derived from an .OBJ file or an .FBX file that carries RGBA information, don't worry. You can still initialise the **PolyPaint Tag** with a color and start using any of the paint tools provided with **PolyPaint Pack**.

To do that, choose a color from the color swatch, a value for the Alpha, and press the **Initialise** button.



The **PolyPaint Tag** is where all the RGBA values are stored. So, if you delete this tag, you will lose all the RGBA information for that object.

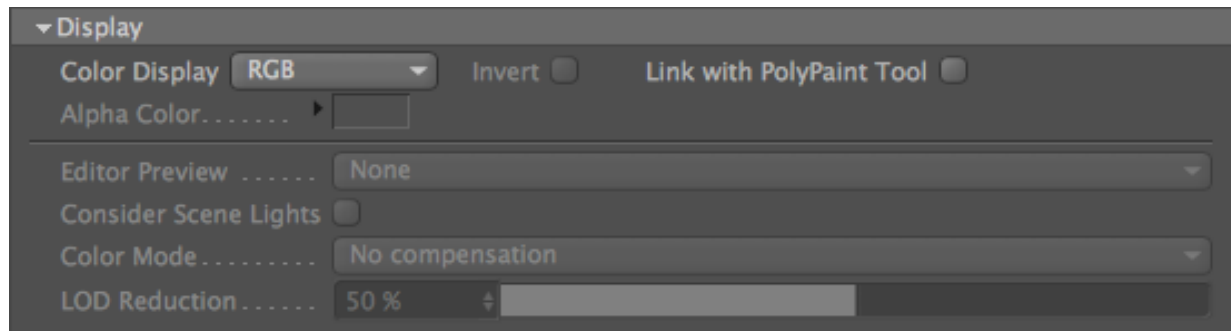
Also, you should not drag this tag from an object onto another because the RGBA values stored in the tag have a direct relation with the amount and index number of the vertexes of the object it is attached to. So, if you decide to drag the **PolyPaint tag** around, you are risking that something bad or unexpected happens.

For high density meshes, this tag can grow up in size quite a lot. This means that your document will get much heavier in size too. So, avoid importing or creating objects that are very dense.

If you import objects from other applications, try to export them in a single piece. Sometimes, objects exported from other applications, when imported into Cinema 4D, arrive separated in several objects. You can always select them all and perform a **Connect and Delete** but, sometimes, this messes up the point count. Performing an additional **Optimize** may, sometimes, mess up the point count even further. When you try to **Analyse .OBJ** file, the point count will not match the number of RGBA values, resulting in an error.

You can **ONLY HAVE ONE PolyPaint Tag** per object. That is a limitation imposed by the complexity and size of the data that is being stored and managed. Maybe in future releases multiple **PolyPaint Tags** may be added.

Once the RGBA values are initialised – be it through the **Analyse .OBJ**, the **Analyse .FBX** file or through the **Initialise** button – you can start doing stuff with that additional information.



In the **Display** section, you can choose what and how it is displayed. You can choose from:

- **RGB** - Only the color information is show.
- **Red** - Only the red component of the color information is show.
- **Green** - Only the green component of the color information is show.
- **Blue** - Only the blue component of the color information is show.
- **RGB+Alpha** - The color information is shown, blended with the Alpha information.
- **Alpha** - Only the Alpha information is show.

If the Color Display is set to **Red**, **Green** or **Blue**, turning ON the **Invert** option, will display the relevant channel inverted.

If the Color Display is set to **RGB+Alpha** or **Alpha**, turning ON the **Invert** option, will display the Alpha channel inverted.

If the Color Display is set to **RGB+Alpha** or **Alpha**, it is also possible to set the color that is used to show the Alpha information (defaults to black).

If the Link with PolyPaint Tool option is ON, when the Color Display is set to **RGB+Alpha**, **RGB** or **Alpha**, the correspondent paint option in the **PolyPaint Tool** will be adjusted accordingly. This can be turned ON or OFF in the **PolyPaint Tag** or in the **PolyPaint Tool**(see the **PolyPaint Tool**, below).

By default, no color is shown in the editor. To show any color choose an option different than **None** from the Editor Preview list. You have:

- **Slow - Always** - With this option selected the object will ALWAYS show the RGB values correspondent to each vertex, even if the object is not selected. However, this option has several limitations:
 - you **CAN'T** see the effect of painting operations while painting
 - you **CAN'T** see any Alpha information, just RGB, Red, Green or Blue.
 - you **CAN'T** select points, edges or faces of the object for editing
 Since this option is slower than other options, you have the option of displaying just a percentage of faces to speed up display (see below).

- **Fast - Any Selected** - With this option selected, the object shows the RGBA values correspondent to each vertex as long as the object is selected or ANY OTHER OBJECT with a **PolyPaint Tag** set to Editor Preview mode of Fast or Fastest is also selected.

As soon as no object is selected (or, at least, no object with a **PolyPaint Tag** set to Editor Preview of Fast or Fastest), the object will stop displaying the RGBA values.

This mode can be used for painting operations with the **PolyPaint Tool**.

- **Fastest - Active Only** - With this option selected, the object shows the RGBA values correspondent to each vertex, only when the object itself is selected. As soon as the object is deselected, the RGBA values associated with each vertex will stop being displayed.

This mode can be used for painting operations with the **PolyPaint Tool**.

When the Editor Preview is set to Slow - Always, the Consider Scene Lights and LOD Reduction options become active.

Turning the Consider Scene Lights option **ON**, will evaluate the scene lights and shade the faces accordingly. If turned OFF, the faces are shown as if the display mode is set to **Constant Shading**.

The LOD Reduction option is set to 50% by default. This means that only 50% of the faces are shown. Increasing this value will display even less faces and decreasing this value will display more faces. When LOD Reduction is set to 0%, all faces will be displayed (slower display). When LOD Reduction is set to 100%, no faces will be displayed (faster display).

The Color Mode options include:

- **No compensation** - No attempt is made to adjust the RGBA values for any specific color workflow. The colors are displayed as they are.
- **Compensate for sRGB** - The displayed RGBA values are compensated for sRGB. What this means is that if you previously painted your mesh in an application that does not work in Linear workflow, when previewing the painting in Cinema 4D, the color may appear too washed out. Setting the Color Mode to **Compensate for sRGB** will display the RGBA values already compensated to better match the color aspect of the original.
- **Compensate for Linear** - It is the opposite of the **Compensate for sRGB** options. If the displayed RGBA values are darker and more saturated than the original RGBA values were, in the application where you painted your mesh, set the Color Mode to **Compensate for Linear** and the displayed RGBA values will be compensated to better match the color aspect of the original.

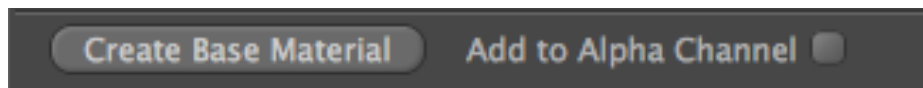
Due to the way Cinema 4D shows the objects in the Editor – in **Fast** and **Fastest** mode – the color of the RGBA values attached to the object is displayed **multiplied** by the solid color of the mesh. This means that, if your object is not shown in the Editor as plain white, the RGB values will show up darker or tinted because the default color of objects without any material is grey.

It is advised that you, at least, create a simple material with the Color channel set to white and attach it to the object containing the **PolyPaint Tag**.

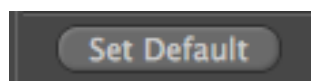
Just press the **Create Base Material** button and this will create a new material, set the Color channel to pure white, insert a **PolyPaint Shader** (see below) into the texture/shader slot of that material and assign the **PolyPaint Tag** to that **PolyPaint Shader**.

If the **Add to Alpha Channel** option is On, the same **PolyPaint Shader** is also inserted in the Alpha Channel of the material, the Alpha Channel is turned On and the **Use Alpha** option of the **PolyPaint Shader** is automatically turned On.

Finally, the material is added to the object, right after the **PolyPaint Tag**, all ready for rendering.



You can also set this up manually if you need to.



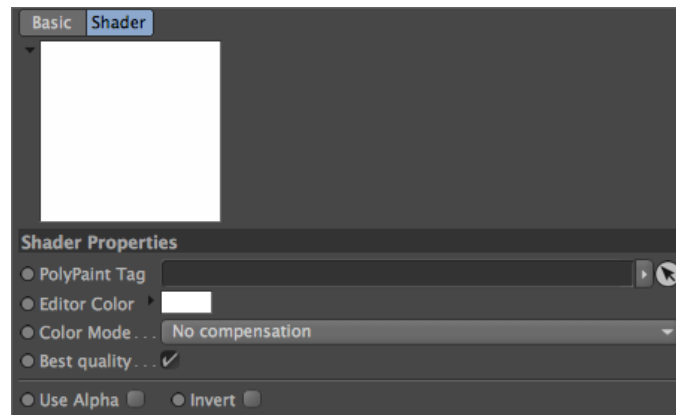
If you find yourself configuring the **PolyPaint Tag** to a specific set of options whenever you create a new one, maybe you should consider setting a new set of options as the default. To do that, adjust the options of the **PolyPaint Tag** to the values you prefer and press the **Set Default** button. From now on, every time you create a new **PolyPaint Tag**, the saved options will be already set as the default ones, until you save a new default set.

Note: To be able to save defaults on a Windows system, Cinema 4D has to be set to run as Administrator.

PolyPaint Shader

The **PolyPaint Shader** has the sole purpose of providing a render-time representation of the RGBA values stored in the **PolyPaint Tag**.

It can be used in any channel and in any slot of any shader that uses color or grayscale as input. The **PolyPaint Tag** field is where you must drag the **PolyPaint Tag** into. If no tag is there or the tag still has no RGBA values, the render will not return the expected colors.



So, to correctly render the RGBA colors, drag the correct **PolyPaint Tag** into this field.

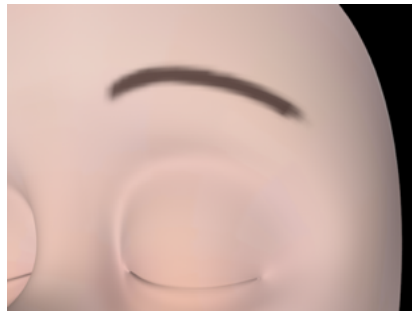
The **Editor Color** (defaults to white) is the color that is displayed/rendered when the RGBA values are not being showed. For example, when the **Fast/Fastest** mode is set and no object is selected.

The **Color Mode** parameters serves the exact same purpose as the **Color Mode** parameter in the **PolyPaint Tag** but, this time, for render.

The **Best Quality** option (ON by default) determines the type of interpolation used for rendering the RGB values. Turning off the **Best Quality** does not increase the render speed too much and choosing between having it on or off is more a matter of visual style required.



Editor View



Best Quality ON



Best Quality OFF

Turning ON the **Use Alpha** option will output the Alpha information instead of the RGB information. This is best used on channels like Alpha, Transparency or Bump.

Turning ON the **Invert** option will invert the Alpha information output.

PolyPaint Tool

The **PolyPaint Tool** is a multi-purpose tool for painting and editing the RGBA values stored in the **PolyPaint Tag**.

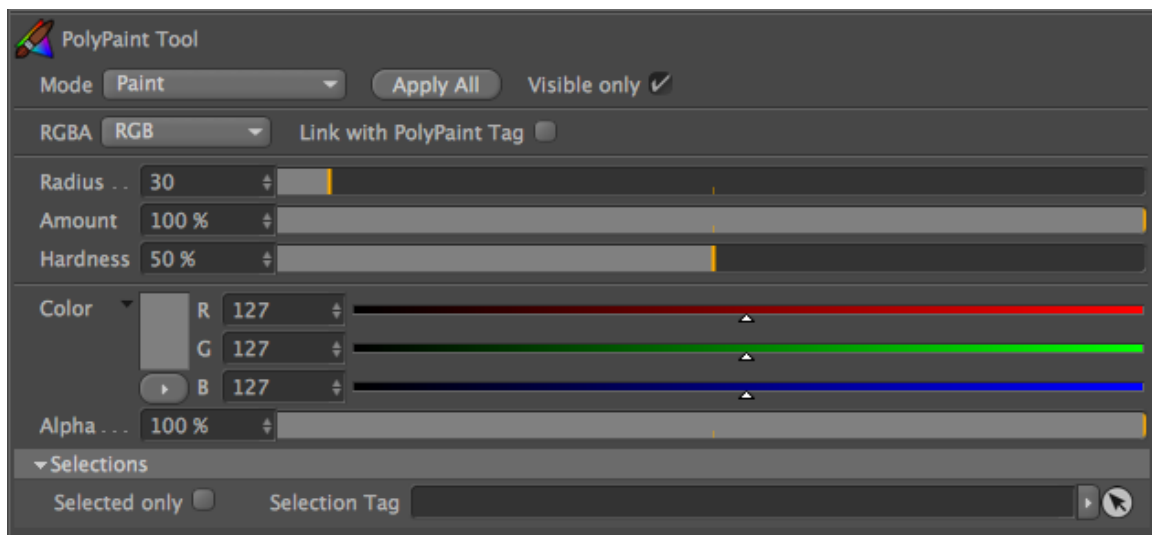
In order for this tool to work some requirements must be met:

- A polygonal object must be selected.
- The polygonal object must have a **PolyPaint Tag** attached.
- The **PolyPaint Tag** must contain RGBA values (loaded or initialised)
- The **Editor Preview** mode of the **PolyPaint Tag** must be set to **Fast** or **Fastest**.

If any of these requirements is not met, the **PolyPaint Tool** will NOT work.

So, make sure all requirements are fulfilled and, if necessary, change to any other tool (Selection, Move, Scale, Rotate, etc) and get back to the **PolyPaint Tool**.

While the **PolyPaint Tool** is active, the Attribute Manager shows this layout:



The **Visible only** option, when turned ON, will make that all paint/editing operations that are manually executed on the model (not using the **Apply All** button), will only affect vertexes that are not hidden by any part of the model.

The **Apply All** button will execute the operation defined by the **Mode** parameter on all the vertexes of the model.

The **RGBA** parameter defines if the painting occurs in the RGB only, in the Alpha only or in both, at the same time (this applies to the manual painting or the **Apply All** button).

If the **Link with PolyPaint Tag** option is ON, changing the **RGBA** parameter will adjust the **Color Display** parameter of the **PolyPaint Tag** accordingly. This way, you are sure that you are always seeing what you are painting. If the **Link with PolyPaint Tag** option is OFF, it is possible to paint on **RGB** and don't see the result, if the **Color Display** parameter of the **PolyPaint Tag** is set to **Alpha**, or paint on the **Alpha** and don't see the result, if the **Color Display** parameter of the **PolyPaint Tag** is set to **RGB**.

It is possible to turn ON/OFF the link between the **PolyPaint Tag** and the **PolyPaint Tool** both from the **PolyPaint Tag** or the **PolyPaint Tool**.

Sometimes, the parameters **Radius** and **Amount** will work in slightly different way whether the paint operation is done by hand, directly on the surface of the model or if it is applied by pressing the **Apply All** button. Both behaviours are described below, for each **Mode**.

The **Radius** option defines the size of the brush. This is an absolute pixel-sized value. This means that if, for example, the radius is set to 30, it will always be 30 pixels wide, no matter how zoomed in or zoomed out the editor is.

The **Amount** option defines how heavily the paint/editing effect is applied. Some modes work better with low values and this will be pointed out, individually, below.

The **Hardness** option defines the smoothness of the edge of the brush. Some modes work better with low values and this will be pointed out, individually, below.

In the following pages, a detailed explanation of each painting mode is provided.

Finally, the **Selections** group:



If the **Selected only** option is turned on, the painting operations (painting directly on the mesh or clicking the **Apply All** button) only affect the vertexes that are selected.

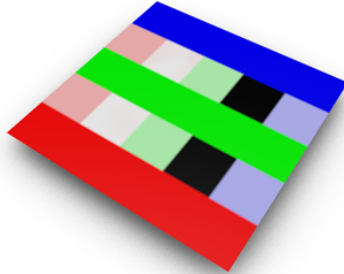
If the **Selection Tag** field contains a **Point Selection Tag**, the painting operations (painting directly on the mesh or clicking the **Apply All** button) only affect the vertexes that are marked as selected in that **Point Selection Tag** (the **Point Selection Tag** must belong to the object that contains the **PolyPaint Tag**).

If the **Selection Tag** field contains a **Vertex Weight Tag**, the painting operations (painting directly on the mesh or clicking the **Apply All** button) only affect the vertexes that have a percentage bigger than 0% in that **Vertex Weight Tag** (the **Vertex Weight Tag** must belong to the object that contains the **PolyPaint Tag**). The amount of paint/effect applied is a multiplication of the value of the **Amount** field and the value of the weight set in the tag.

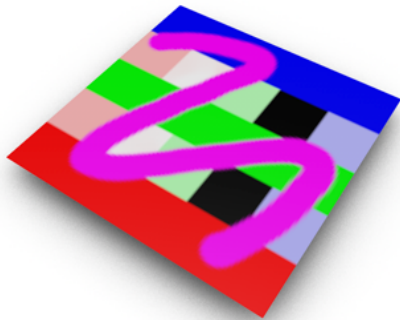
If the **Selection Tag** field contains a **Point Selection Tag** or a **Weight Map Tag** and the **Selected only** option is turned on, the painting only affects the points that are live-selected AND marked as selected (or weighing more than 0%) in the tag dragged into the **Selection Tag** field.

Paint Modes - Explained

This is the initial image over which all the paint/editing modes will be applied on. If anything works in a particular way in **Apply All** or when affecting Alpha, an explanation will be provided.



Paint



Painting directly on the mesh

This mode allows you to paint the vertexes with color.

The color is always applied in normal mode.

This means that no Multiply, Screen, Overlay, etc, modes are provided.

Clicking the **Apply All** button

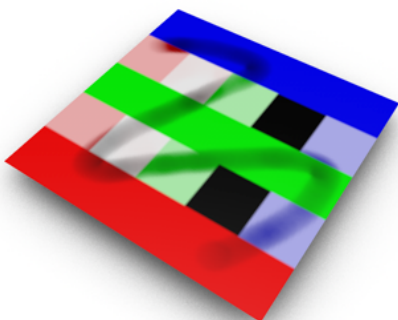
This will fill the whole mesh with the defined color.

To tint the all mesh with a color, lower the **Amount** value to something less than 100%.

The **Radius** and **Hardness** values are ignored.

The value that is painted in the Alpha (if **RGB+Alpha** or **Alpha** is set in the **RGBA** parameter) is set by the **Alpha** value slider, below the color sliders. The **Amount** value is ignored.

Darken



Painting directly on the mesh

This mode allows you to darken the color of the vertexes until they reach pure black.

The amount of darkening is controlled by the **Amount** parameter.

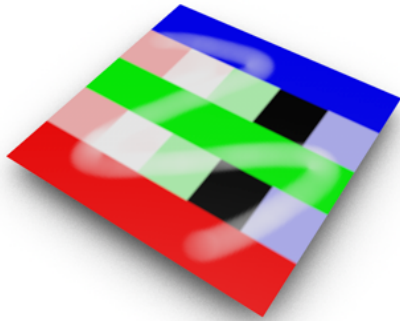
This works better with low **Amount** values.

Clicking the **Apply All** button

This will darken the whole mesh by the values set in the **Amount** parameter.

The **Radius** and **Hardness** values are ignored.

Lighten



Painting directly on the mesh

This mode allows you to lighten the color of the vertexes until they reach pure white.

The amount of lightening is controlled by the **Amount** parameter.

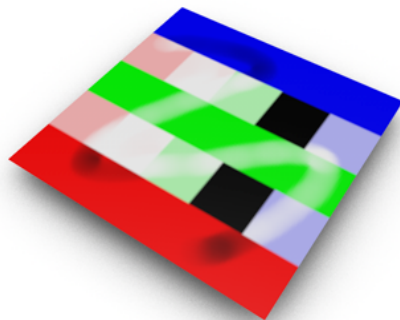
This works better with low Amount values.

Clicking the **Apply All** button

This will lighten the whole mesh by the values set in the **Amount** parameter.

The **Radius** and **Hardness** values are ignored.

More Contrast



Painting directly on the mesh

This mode allows you to darken the dark color and lighten the light colors of the vertexes.

The amount of darkening/lightening is controlled by the **Amount** parameter.

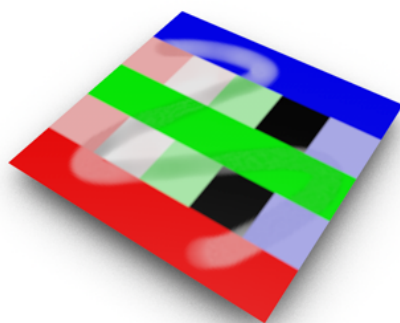
This works better with low Amount values.

Clicking the **Apply All** button

This will darken all the dark colors and lighten all the light colors of the whole mesh by the values set in the **Amount** parameter.

The **Radius** and **Hardness** values are ignored.

Less Contrast



Painting directly on the mesh

This mode allows you to darken the light color and lighten the dark colors of the vertexes.

The amount of darkening/lightening is controlled by the **Amount** parameter.

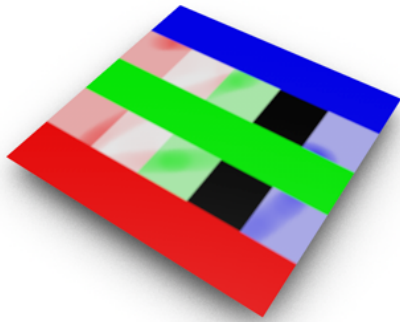
This works better with low Amount values.

Clicking the **Apply All** button

This will darken all the light colors and lighten all the dark colors of the whole mesh by the values set in the **Amount** parameter.

The **Radius** and **Hardness** values are ignored.

Saturate



Painting directly on the mesh

This mode will add color vibrance to the colors of the vertexes.

The amount of saturation added is controlled by the **Amount** parameter.

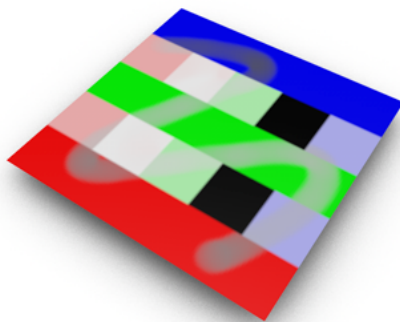
Clicking the **Apply All** button

This will saturate the whole mesh colors by the values set in the **Amount** parameter.

The **Radius** and **Hardness** values are ignored.

This mode will not affect the Alpha because the Alpha is always grayscale.

Desaturate



Painting directly on the mesh

This mode will remove color vibrance from the colors of the vertexes until they reach a grey value.

The amount of saturation removed is controlled by the **Amount** parameter.

Clicking the **Apply All** button

This will desaturate the whole mesh colors by the values set in the **Amount** parameter.

The **Radius** and **Hardness** values are ignored.

This mode will not affect the Alpha because the Alpha is always grayscale.

Pick Color Point

Picking directly on the mesh

This mode allows you to pick a color by dragging on top of the mesh. The picked color will be the one nearest the center of the brush.

Radius, **Amount** and **Hardness** are ignored.

Clicking the **Apply All** button

The resulting color is an average of all the colors of the mesh,

Pick Color Average

Picking directly on the mesh

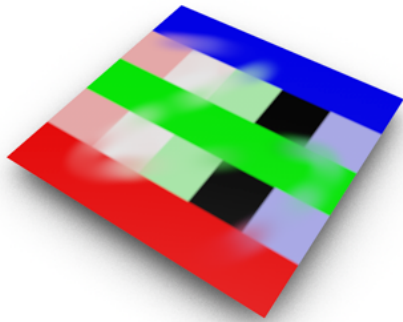
This mode allows you to pick a color by dragging on top of the mesh. The picked color will be the average of all coloured vertexes inside the brush radius.

Amount and Hardness are ignored.

Clicking the **Apply All** button

The resulting color is an average of all the colors of the mesh,

Smooth



Painting directly on the mesh

This mode will smooth/blur the colors inside the radius of the brush.

The amount of smoothness is controlled by the **Amount** parameter.

Works better with low **Amount** and low **Hardness**.

Clicking the **Apply All** button

This will smooth the whole mesh colors by the value set in the **Amount** parameter.

Only colors closer to each vertex than the **Radius** will be considered for smoothing.

Hardness values are ignored

WARNING!!

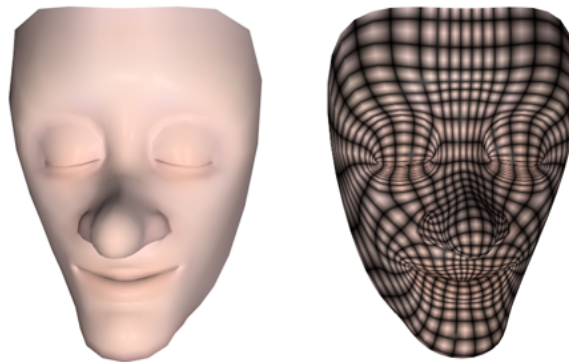
The **Apply All** button, when in **Smooth** mode, does not work in Cinema 4D for Windows 32 bits.

PolyPaint Transfer

PolyPaint Pack works better with meshes that are not extremely dense. **PolyPaint Pack** works faster when it does not have to deal with a lot of data as an RGBA value must be saved for each vertex of the mesh).

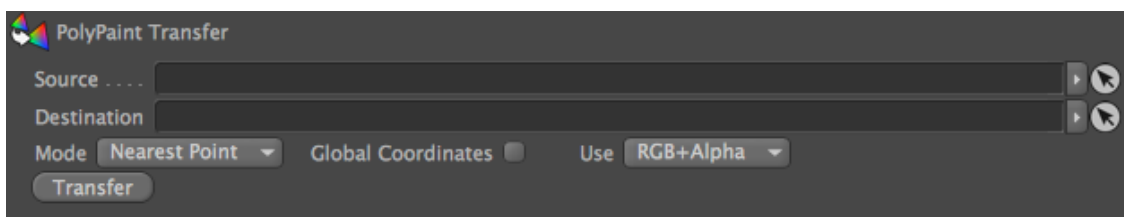
However, sometimes, higher density meshes must be used. Whether they are created inside Cinema 4D with the Subdivide command (usually with the HyperNURBS subdivide option ON), or simply importing a higher density version of the same mesh.

The same **PolyPaint Tag** can't work properly in objects whose point count increased. For example, if your original object has 794 vertexes, the **PolyPaint Tag** was storing 794 RGB values. As soon as the object has more vertexes, let's say, 3176 vertexes, only 794 vertexes can be painted with color and the rest will show up as black.



So, if you already have an object painted, it would be nice to be able to transfer all the color information to another object that has a different vertex count. And that is exactly what the **PolyPaint Transfer** tool does.

This is the dialog of the **PolyPaint Transfer** tool:



Into the **Source** field, you drag the object from which you want to copy the colors.

Into the **Destination** field, you drag the object onto which you want to copy the colors.

Each object must contain a **PolyPaint Tag** initialised with some color information.

The **Mode** parameter has two options:

- **Nearest Point** - Each vertex of the destination object will receive the color of the nearest vertex of the source object.
Each vertex is assigned the local coordinates based on the object's axis. This means that the object the RGB information is being transferred to can be in a different location within the scene without any loss of data or errors.
If the **Global Coordinates** option is **On**, the objects must be aligned and positioned in, roughly, the same location.
- **Ray Collision** - Virtual rays are fired from each vertex of the destination object. Wherever those virtual rays hit the surface of the source object, the color is sampled from that location and stored in the "ray emitting" vertex.
This mode **ALWAYS** works in local mode so the objects can be placed in different locations and with different orientations.

Some calculations take up some time, especially with meshes that are very dense. Look to the status bar (lower left) to check the progress of the tool, after pressing the **Transfer** button. There are no fixed rules for choosing any particular **Mode** but, usually, for transferring from a lower-density model (source) to a higher-density model (destination), the **Ray Collision** mode works a bit better. And, for transferring from a higher-density model (source) to a lower-density model (destination), the **Nearest Point** mode works a bit better.
If the transfer is taking too long, it is possible to press the ESC key to interrupt the operation. Only the colors transferred so far will be kept and everything else will remain with the previous color, if there was any.

The **Use** parameter allows you to define what gets transferred. If the RGB only, the Alpha only or the RGB+Alpha together.

PolyPaint from Material

The **PolyPaint from Material** command allows you to use regular materials, applied to objects, to create RGB values, without the need of painting by hand. Optionally, you can also transfer Alpha information from the material to the Alpha values stored in the **PolyPaint tag**.

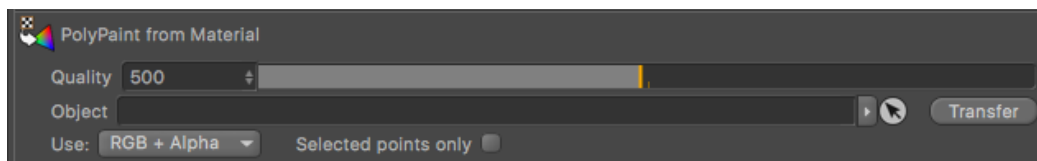
It is a very simple command and it has some limitations. But, used wisely, can produce excellent results and save lots of time and work.

To use this command, some requirements must be met:

- The object must contain a **PolyPaint Tag** already initialised.
- The object must have a UVW tag.
- The UVs should not be overlapping (use the BodyPaint UV Edit to edit them, if necessary)
- For better results, just one UVW tag should be present and, ideally, be the last tag of the object.

Create and apply as many materials you need. Adjust their projection, placement and any other material tag parameters, as required. Combine materials with Selections, with Mix Textures, with Alphas, turning on/off Tile, etc. When all the materials are placed and rendering yields the correct result, choose the **PolyPaint From Material** command.

Drag the object to the Object field and press Transfer.



If it all went well, you can now delete the Material tags and the UVW tag.

To see if it all went well, make sure you set the **Editor Preview** option of the **PolyPaint Tag** to anything other than **None**.

Adjusting the **Quality** parameter allows you to define how accurately the materials will be sampled. However, the final quality of the resulting transfer depends greatly on the amount and density of points of your mesh. Higher **Quality** values means more accuracy but also, longer time to calculate the sampled colors.

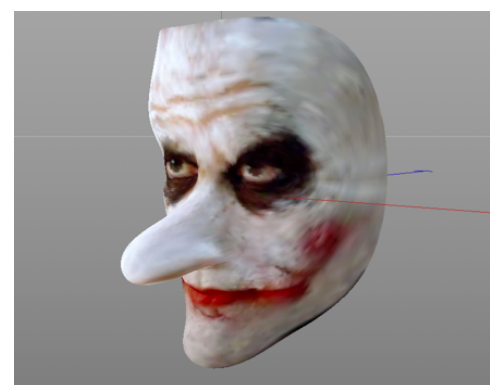
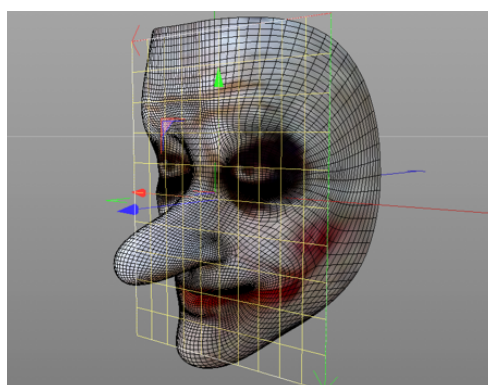
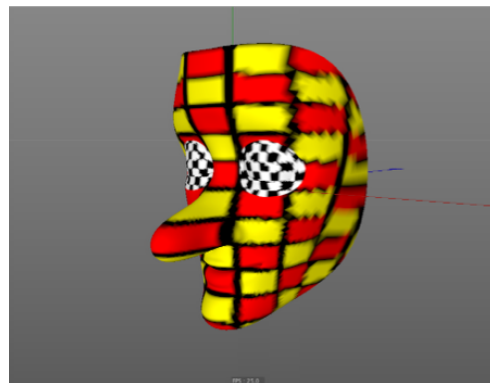
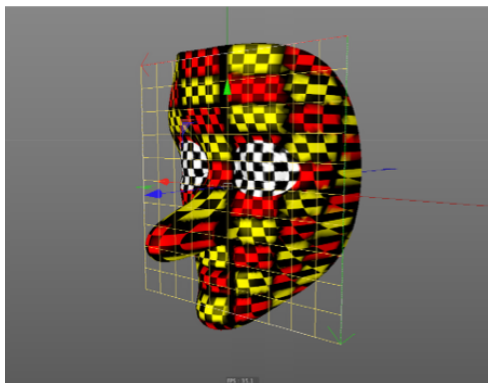
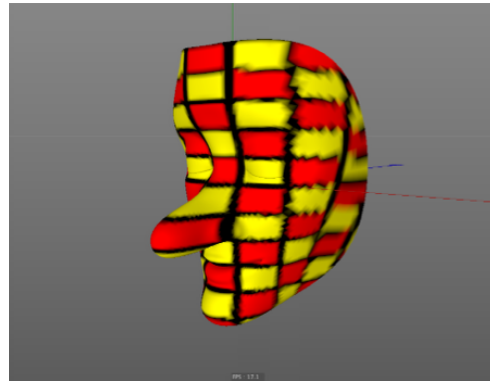
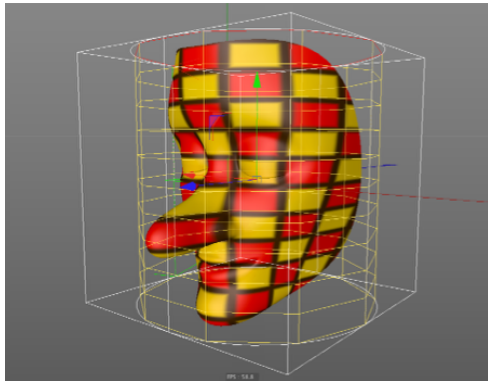
The **Use** option can be set to **RGB+Alpha**, **RGB** or **Alpha**. If the option includes **RGB**, the color in the **Color** channel is transferred. If the option includes **Alpha**, any information included in the **Alpha** channel of the material is transferred to the Alpha values contained in the **PolyPaint Tag**. Due to the way the calculations are performed, if there is any information in the **Alpha** channel of the material, that information will be multiplied with the **Color** channel information when stored in the RGB values stored in the **PolyPaint Tag**.

If you just want to transfer color information to some points, select the points and make sure you turn on the **Selected points only** option before pressing the **Transfer** button of the **PolyPaint from Material** command.

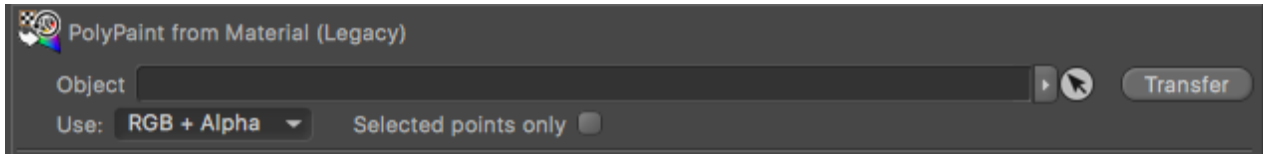
It is also possible to use several materials in different **PolyPaint from Material** sessions, by composing many color transfers. This is achieved by repeating all the steps described above, making sure that the **Selected points only** option is turned **On** and that different sets of points are selected each time.

With some artistic use of bitmaps, some very nice results can be obtained.

And, all the colors are now attached to the points of the object, so you can deform it as much as you want and you can also edit the colors with the **PolyPaint Tool**.



PolyPaint from Material



This new **PolyPaint From Material** command works fine for most cases but, for low-poly meshes, the old **PolyPaint From Material** command actually works better. So, to keep everyone pleased, the old command was kept but renamed to **PolyPaint From Material (Legacy)**.

However, the old command had a few extra limitations, besides the ones that were listed for the new command. And they are:

- The object must only have one material (the one that is to be processed).
- The object must have a UVW tag.

The material tag should keep all the values at their defaults, except for the Projection that must be set to UVW Mapping. This means that, for example, the tiling – if set – will be ignored. Although these seem to be lots of complicated conditions, it is quite easy to make sure they all are fulfilled. Do the following:

- Create a new material.
- In the **Color** channel, load a bitmap or a shader and adjust it as you want.
- Drag the material to the object (make sure there is no other material or UVW tag assigned to the object).
- Adjust the **Projection** to whatever type is best for you (Flat, Spherical, Cubic, etc).
- Adjust the position, scale and rotation of the material (select Texture mode, turn ON the Enable Axis modification icon and use the Move, Scale and Rotate tools).
- When happy with the result, right-click the **Material** tag and choose “**Generate UVW Coordinates**” from the menu list. The Material tag Projection mode will change to UVW Mapping and a new UVW tag will be created.
- Choose the **PolyPaint from Material (Legacy)** command.
- Drag the object to the Object field, adjust the Use option if necessary and press Transfer.

If it all went well, you can now delete the Material tag and the UVW tag.

To see if it all went well, make sure you set the Editor Preview option of the **PolyPaint Tag** to anything other than None.

Besides these small differences the **PolyPaint From Material (Legacy)** works just the same as the new **PolyPaint From Material**. Namely, the **Selected points only** option (see above).

Finally...

As a final piece of advice, since **PolyPaint Pack** deals with huge amounts of data, save often. All the tools were tested intensively but it is always better to be safe than sorry.

This set of plugins was extensively tested by me and a beta tester. However, not all possible scenarios could be evaluated. If you find any behaviour that you consider to be a bug, please report it to me, at rui_mac@ruimac.com

Addendum

PolyPaint AddOns

Three additional commands were added to version 1.5 of **PolyPaint Pack**: **PolyPaint FBX Import**, **PolyPaint FBX Export** and **PolyPaint Cleaner**.

PolyPaint FBX Import and PolyPaint FBX Export

The limitation of only being able to import and export RGBA information from FBX 6.0 files is finally gone. This is very good news for Unity users.

So, in version 1.5, two additional commands were added for exporting were added.

In your Plugins menu, there will be an extra entry, named **PolyPaint AddOns** and, inside it, two new import/export commands: **PolyPaint FBX Import** and **PolyPaint FBX Export**.

To import the RGBA information that is inside an FBX 7.2 file, first open the FBX file using the **Open** or **Merge** command of Cinema 4D.

Once the mesh is loaded, add to it a **PolyPaint** tag and **Initialise** it.

Make sure the object containing the **PolyPaint** tag is selected and choose **PolyPaint FBX Import** from the **Plugins->PolyPaint AddOns** menu and choose the same FBX file you used to load the mesh into Cinema 4D.

If it all goes well - error messages are displayed if something goes wrong - you can now display and edit the colors on the mesh (setting a **Editor Preview** to something other than **None** is required, of course)

To export your painted mesh to FBX 7.2 format, including the RGBA values in the file, make sure the painted mesh containing the **PolyPaint** tag is selected and choose the **PolyPaint FBX Export** command from the **Plugins->PolyPaint AddOns** menu.

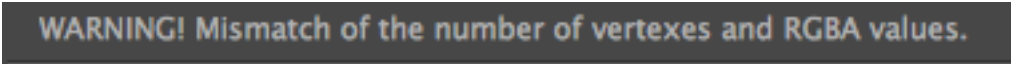
A **Save** dialog will appear, allowing you to choose the name and location of the FBX file.

When the FBX file is imported into **Unity**, a shader that reads the vertex color information is required. There are many free shaders or packages with shaders available for Unity that shade objects based on their Vertex Color information. You must use the correct one for interpreting the Alpha information also.

PolyPaint Cleaner

The number of points of the object that contains a **PolyPaint** tag must match the number of RGBA values stored in that **PolyPaint** tag. If the user deletes or adds points to an object after initialising a **PolyPaint** tag, those numbers will not match. This can lead to a number of problems, specially when exporting the object since the list of vertexes will not match with the list of RGA values.

If the number of vertexes and RGBA values does not match, the **PolyPaint** Tag will show a warning.

A dark gray rectangular box with a thin border containing the text "WARNING! Mismatch of the number of vertexes and RGBA values." in a light gray, sans-serif font.

WARNING! Mismatch of the number of vertexes and RGBA values.

This will also prevent you from exporting in .OBJ or .FBX format. If you try it, you will get a warning stating the same and prompting you to correct the problem before continuing.

Also, **PolyPaint** tags created with versions of **PolyPaint** prior version 1.5 will not include the Alpha information. So, if you open a file with **PolyPaint** tags created with **PolyPaint** prior version 1.5, you will not be able to paint on the Alpha or use the Alpha information.

To fix all these situations, select an object that contains a **PolyPaint Tag** and choose **PolyPaint Cleaner** from the **Plugins->PolyPaint AddOns** menu. This will check if the number of points matches the number of RGBA values. If it doesn't, it will add or delete RGBA values to or from the **PolyPaint tag** to make it all match.

Also, if there is no Alpha information, that layer of information is added, so that all Alpha operations become possible.

So, if you load an older file with **PolyPaint tags**, perform a **PolyPaint Cleaner** command on all objects containing tags so that they all get updated to version 1.5

Also, before exporting to .OBJ or .FBX, just in case, perform a **PolyPaint Cleaner** command on the object, just to make sure everything is fine.

After performing the **PolyPaint Cleaner** command, a window with information about what was done will appear.

Acknowledgements

I wish to thank Rodrigo Bitencourt Rodrigues (cyanographics@gmail.com), an animation movies graduate from the UFPEL (Universidade Federal de Pelotas - Brasil), who first contacted me about the possibility of creating a plugin to read and render RGB information from .OBJ files created in Zbrush.

I started coding in python, first to create the tag that would read and store the RGB values and then, to code the shader that would render those RGB values. The tag worked just fine but, due to limitations of the python libraries, the shader proved to be just too slow for efficient renders. That is why I finally decided that it was time to dive into the C++ world.

So, thank you very much, Rodrigo. Without you, **PolyPaint Pack** would not exist and I would not have started coding in C++. I would also like to thank you for the excellent beta-testing of this set of plugins and for providing great material for me to test the code on my own, as I was coding.

I would also like to thank everyone who bared with me at the Plugin Cafe (www.plugincafe.com) with all my constant questions and doubts about C++.

I would like to address a special thank you to Scott Ayers (ScottA) and Cactus Dan who were especially patient and helpful.

I would like to also thank “TheJimReaper” (Jim Field), James Leaburn and StCanas for helping me out with the proofreading of this manual.

Version History:

Version 1.0 (September 2014)

- Initial release.
- Only .OBJ files created with zBrush, 3D Coat and similar application, storing the RGB colors in MRGB blocks are supported.
- Only .FBX files saved in ASCII format in version 6.x are supported.

Version 1.1 (December 2014)

- Added importing of .OBJ files with RGB values coded directly in the vertex coordinates lines (like the ones created with meshlab application).
- Changed the algorithm to smooth colors locally and globally. It contained a bug and works much better now.
- Small speed improvements in the **PolyPaint Transfer** in Nearest Point mode.

Version 1.2 (May 2015)

- Added importing and exporting of .FBX files saved in ASCII format in version 7.2, mainly for Unity compatibility.

Version 1.5 (July 2015)

- Added the ability to work with Alpha information.
- All tools, modes and commands that could use Alpha were updated.
- New visualisation modes.
- New **PolyPaint From Material** command, allowing for more complex and flexible use of materials and projections to generate **PolyPaint** information.
- Added a new **PolyPaint Cleaner** command.
- Added the possibility of storing defaults for the **PolyPaint Tag**.
- Added a link between the **PolyPaint Tag** display and the **PolyPaint Tool** paint.
- Bugs cleaned from several tools, mainly in the manual **Smooth** mode in the **PolyPaint Tool**.

Version 1.5.2 (February 2016)

- Added the ability to work with Alpha information in the **PolyPaint From Material** command and **PolyPaint From Material (Legacy)** command.
- The .OBJ export now works in R17 also (it got broken after R16).
- Small internal tweaks to the **PolyPaint Tool**, mainly in the **Smooth (Apply All)** algorithm.