UNIVERSITY OF APPLIED SCIENCES
HOCHSCHULE MITTWEIDA

Angewandte Computer- und Biowissenschaften

Professur Medieninformatik

Bachelor Thesis

Analysis and improvement of Blender's texture painting functionality

Patricia Ließ

Mittweida, der 2. Juni 2019

Erstprüfer: Prof. Dr. rer. nat. Marc Ritter
Zweitprüfer: Manuel Heinzig M. Sc.

Ließ, Patricia
Analysis and improvement of Blender's texture painting functionality
Bachelor Thesis, Angewandte Computer- und Biowissenschaften
Hochschule Mittweida– University of Applied Sciences, June 2019

Abstract

This thesis is about the process of painting textures for 3D game models and the tools available in 2D and 3D software to do so with a focus on the Open Source 3D Suite Blender.

Name: Ließ, Patricia
Studiengang: Medieninformatik und Interaktives Entertainment
Seminargruppe: MI15w2-B
English Title: Analysis and improvement of Blender's texture painting functionality

Inhaltsverzeichnis

Abbildungsverzeichnis

[1]

[1]all the figures aside from Fig. 2.3 are either screenshots taken inside the respective applications or images composed of these; the images in Fig. 2.3 have been provided by Michael Kuwilsky

Tabellenverzeichnis

1 Introduction and Motivation

'Game Design is about creating an Experience.' [Sch08, p.10]

In the process of developing a game, a lot of decisions have to be made about what it should feel like to play and what has to be done to realize the vision the game developers have set out to achieve. And yet, the game itself is not the end goal but merely a means to enable the player to immerse themselves into its world and have an experience [Sch08, p.10].

A big part of creating said experience are the aesthetics. The visual presentation has an important role in getting potential players interested in the game in the first place. They are also essential to making the world feel genuine and believable [Sch08, p.347]. At the same time though, developers have to constantly think of ways to keep performance up and stable for the game to run smoothly on the end user's hardware. These two goals often stand in conflict with each other. To combat this, many techniques and tools have been developed over the course of game development history in an attempt to find solutions [Ahe17, p.3].

One such technique is the texturing of 3D objects. Texturing allows for more surface detail [Ahe17, p.38] on the model without impacting the performance too much since there are no additional polygons that need to be displayed by the engine. This may not necessarily make a difference when there is only one object, but game worlds are made up of a lot of assets and not every hardware setup can handle the task of rendering them all in real time.

These textures can be created in many ways: Materials can be photographed or made from pre-existing images. There are also a number of online services such as Poliigon that provide textures ready to be used for rendering photo-realistic images. Of course, textures can be procedurally generated or painted by hand, too. One application that may be used to do the latter is Blender.

Unlike other applications typically employed in the industry, Blender is a free and continuously developed Open-Source 3D Suite supported by an ever-growing community of users. At the time of writing, Blender is in the beta-phase of version 2.8 which brings with it significant changes such as a new Render Engine, an overhauled User Interface and - most importantly - a quite sophisticated tool for drawing and animating 2D elements in 3D space, showing what Blender can be capable of in regards to creating graphics. For the future, the development team has already announced that it will be working on further extending texturing capabilities for both procedural and manual painting methods.

Being able to create textures manually enables artists to dictate the appearance of a 3D object in the medium they are most comfortable with and have control over: drawing and painting. Developing the knowledge and know-how needed to realize a visual concept takes years of practice to make the right decisions about when to use which tool with which color in which configuration. This way, they can directly dictate the final look, unlike with procedural methods and pre-made images. That is why it is an important tool to have in the belt for any texture artist.

The goal of this thesis is to find out which options an artist has at their disposal to paint textures in Blender and what can be done to improve these conditions for working in a production environment in terms of what tools could be added to further enable them.

1.1 Reasons for using Blender

Blender is a 3D-Suite. Unlike other applications - such as ZBrush or Substance Painter which are more specialized in terms of texture painting and will be scrutinized later on in this thesis - Blender aims to become an all-around solution for the entirety of the production steps that go into developing 3D creations for VFX, movies, games and animation. This makes it well-suited for a production team as they do not have to switch applications as often as they do when needing to work with a whole set of software for all the various stages in the pipeline, meaning they do not have to worry about problems between file formats as much. Another advantage is that it is easier to help each other out when working with the same software.

Blender is also a free program. This means that anybody with the minimum hardware requirements and an internet connection can download and run it without having to pay a lot of money to be allowed to use it. This also lowers the barrier of entry. Beginners can dabble into it as a hobby without having to commit by paying a price they may not be able to afford or even justify as they may not even know if it is something they want to continue doing yet.

Lastly, Blender is an Open Source software. It has a big community behind it that use it for creating art and request features or customize it by developing their own add-ons when something is missing. They are also very active in reporting bugs, making it possible to advance the program more rapidly.

1.2 Texture Painting and other Creation Methods

As has been stated before, textures can be created in a lot of ways, e.g. through procedural generation, painting or using images found on the web or by photographing real world objects. These are all viable ways of texture image creation, though each of them has its up- and downsides.

The easiest and quickest method would be taking an image found on the internet. The benefit of this is that there is a wide variety to choose from though depending on the context in which the object is displayed later on, this may or may not have legal consequences in case the licence for the image does not allow this use. On a visual level, the image may not be as well-suited to the object which may cause *seams* to show - spots on the object where the texture is not continuous that break the suspension of disbelief - or the resolution is not high enough. Some of these issues may be fixable though using image manipulation tools.

Photographing real-life objects eliminates the problem of legal consequences of using an existing image - in most cases - while still being a quick way to create textures. It also allows for more control of the lighting as well as the contents shown. Seams and the resolution may still be an issue though, whereby the latter depends on the camera and settings. These photographs are also not as suited for creating non-realistic textures.

Procedural Content Generation is a major topic in the games industry. Which is not surprising, considering its many advantages: A well-made system is able to generate a lot of content in a very quick and cost-effective manner. Especially when it comes to making a number of variations of the same asset for games with an open world, they are very useful.

However, this method of content creation does have some disadvantages: For one thing, these systems are only worth it if they are used on a large scale since the development of one that is of good quality requires a lot of hours and resources [Gre17, p.6f.]. Even though the content is generated by algorithms, the developer behind it still has to define the parameters that are needed to do so and what edge cases may appear that will have to be resolved. The ability to directly influence the later presentation of the game content is also entrusted to the algorithm - and with that the chance to create exactly the experience that has been envisioned.

Establishing a workspace for texture painting, on the contrary, is done significantly quicker. The artist has direct control over the look of the texture on the 3D model and can look at the latter at any moment while working. As long as non-destructive methods are used there are also a lot of possibilities for quick changes, like color correction.

Above all the artist has a lot more control over the final result and can gauge at any time, how well the object fits into the game with the texture applied by constantly asking themselves questions and making decisions, like how many details should be included, how strong the contrast in a specific area should be, which color should be used, etc. until the outcome matches the intended vision.

1.3 Objective and Structure of the Thesis

The aim of this thesis is to examine the possibilities for hand-painting textures in the Open Source software Blender [Tea19a] which involves looking for what the application already does well while also examining the ways in which it falls short and finding and designing solutions that can be integrated into the interface.

To do that, this work will start off with an explanation of some of the general key concepts in Texturing and Surfacing. These involve going into different types of textures, which systems are used to composite them into a coherent final image and how they are then applied to 3D models. It will then move on to go over what constitutes a workflow for 'traditional 2D' Digital Painting as well as which requirements should be met to work efficiently in an asset production pipeline for a video game.

Chapter 3 will be an analysis of the texture painting options in Blender. Weaknesses as a result of insufficient or missing functionality will be noted. Alternate 2D and 3D software used in texture painting workflows will then be scrutinized using these findings to explore how they approach those issues.

Chapter 4 again establishes some basics, this time in preparation for add-on development in Blender. Subject to scrutinization is the philosophy of the Blender User Interface and some general usability design principles as well as how Blender enables developers to write their own scripts using Python 3.

Chapter 5 constitutes the practical part of this thesis, in which various ways of extending the texture painting capabilities of Blender will be explored as well as choosing one of them to develop further and implement.

Chapter 6 will be an examination of how well this integration works in context of the actual texture painting work process by looking at how much faster an artist can paint with and without the add-on.

Finally, Chapter 7 concludes this thesis with an outlook on future prospects of painting textures in Blender as well as providing some more examples of what can be done to offer more options to artists.

2 Basic Concepts

For a better understanding of the terms and processes described in this thesis, a basic comprehension of some of the fundamental concepts for texture painting is needed. This chapter will go over what textures are, by which methods they can be created and how they are applied to 3D models. It will also explain how artists in general work to create textures in a video game production environment and what is important for such a workflow.

2.1 Defining the Surface of a 3D Object

To gain an understanding of what goes into detailing the appearance of a 3D model beyond its geometry, this section will go over some of the concepts involved in texturing - or better yet - surfacing an object. Texture Painting itself, as a matter of fact, is only one method of *Surfacing*, a form of *Digital Compositing*. This is the 'process of integrating images from multiple sources into a single, seamless whole' [Bri08, p.2]. In this case, the afore-mentioned images are the textures, though they can be manipulated with various methods and tools like filters, color correction, etc. which will be explained at a later point. For now, this is just mentioned to emphasize the point that texture files are not the only elements to go into the surfacing process.

All these elements have to be organized though. Two systems have been developed to fulfill said task: one that is based on *Layers* and another one that uses *Nodes*. Node-based systems operate on the principle that one node performs a specific procedure on the composition and then passes it on to the next one (cf. Fig. 2.1). All of these individual nodes add up to a node graph. This way of working is used in several places across Blender, namely when it comes to post-processing and surfacing. At the time of writing, this is being extended to be used in other areas as well such as animation in a project called 'Everything Nodes', though.

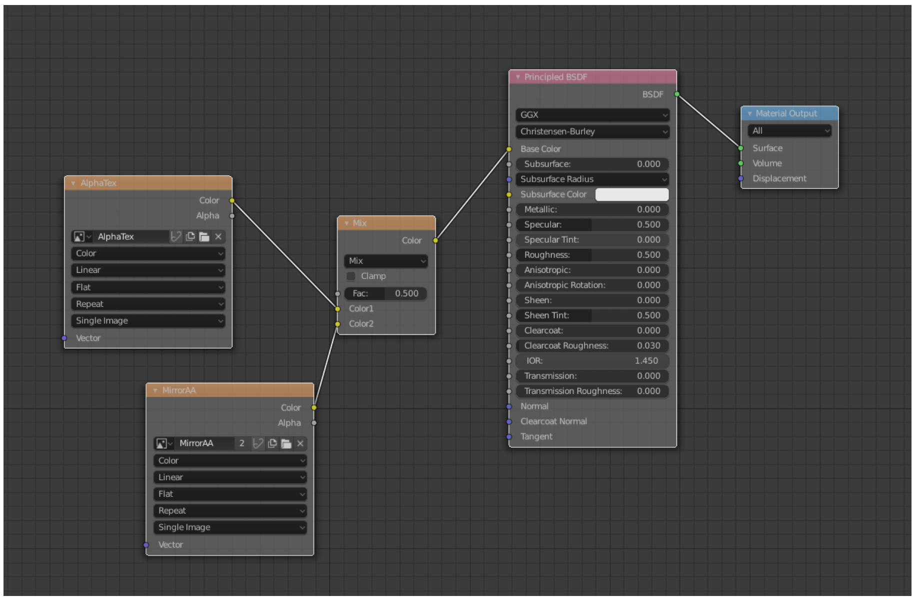
Layer systems, meanwhile, work in the way that the image is split into several parts which are stacked on top of one another. The upper segments either obscure the ones beneath or interact with them, depending on the mode that has been chosen for them. These modes will be elaborated on in section 3.3, though basically, they determine which formulas are used to calculate the color of the current's layers pixels with the input of the pixels beneath.

It is also possible to represent a layer-based system as a node graph. In essence, layers are a form of node as well, though they are very limited. In a node tree, it is possible to branch out and only affect certain parts of the image at a time, whether that be a specific color channel or texture. One node may also take in more than one input and return multiple outputs. The output of one node can then be reused as input for several other operations. Layers, meanwhile, can only be stacked. They are also limited in the operations they can perform on the composition.

The interfaces to manage the layers of an image have been designed to make interaction relatively quick, though, which can not necessarily be said for node systems. That way, it is possible to select multiple layers to merge them into one and reduce the amount of memory needed to store them separately. The originals may also be kept as the new merged layer can be manipulated to experiment with different effects. Adding and deleting layers is also a quick affair and they can be switched around per drag and drop.

The textures that are manipulated in these systems can be created through different means which are roughly divided into two groups: procedural and non-procedural methods. These are employed depending on what the production context is like, how many textures of this sort are needed, what lies in the developer's capabilities, what kind of online resources are available etc.

With *procedural* methods, the texture is created using code segments or algorithms which use parameters entered by the user to 'calculate' the desired texture [Ebe03, p.1]. Because of this, the size of these files is relatively small (in the kilobyte range). They are evaluated at run-time which makes it possible for the textures to not be resolution dependent. However, the computation of the textures may cost performance [Ebe03, p.14f.]. Because the textures are generated at run-time, developers have less control of how they look exactly in game.


AlphaTex
Color
Alpha
AlphaTex
Color
Linear
Flat
Repeat
Single Image
Vector

MirrorAA
Color
Alpha
MirrorAA 2
Color
Linear
Flat
Repeat
Single Image
Vector

Mix
Color
Mix
Clamp
Fac: 0.500
Color1
Color2

Principled BSDF
BSDF
GGX
Christensen-Burley
Base Color
Subsurface: 0.000
Subsurface Radius
Subsurface Color
Metallic: 0.000
Specular: 0.500
Specular Tint: 0.000
Roughness: 0.500
Anisotropic: 0.000
Anisotropic Rotation: 0.000
Sheen: 0.000
Sheen Tint: 0.500
Clearcoat: 0.000
Clearcoat Roughness: 0.030
IOR: 1.450
Transmission: 0.000
Transmission Roughness: 0.000
Normal
Clearcoat Normal
Tangent

Material Output
All
Surface
Volume
Displacement

Abbildung 2.1: Node System in Blender [Fou98]

On the other side, there are image based textures. These can be images or even videos that are projected onto the surface of the 3D object. As they already exist as files, they don't have to be calculated at run-time. However, they are of much larger size than procedural textures and, depending on the number of textures used, the draw calls needed to load them onto the graphics card can slow down performance. These textures may be i.e. downloaded from dedicated online material libraries, created from photographs or with the use of 2D or 3D Software.

This way of texturing is especially used if the setup of a system for procedural texture generation is not worth the time and resources. This may be the case when the textures needed are only applicable to a small number of assets. Some textures can also be too difficult to generate procedurally [Ebe03, p.14].

As they are image files that have to be mapped onto a 3D object, the question arises how the 2D coordinates translate to 3D space. Depending on the texturing method, this varies. There is Vertex painting, which just assigns each vertex a certain color, negating the need to map anything. The downside to this is that the mesh has to have a certain resolution of vertices. This is detrimental to performance when working on the model in a 3D application and can slow down the painting process considerably when the hardware is unable to handle the load.

With texture painting and images, the coordinates get translated from 2D to 3D through a so-called *UV-Map* or *UV-Layout* (cf. Fig. 2.2). The term is derived from the fact that 'U' and 'V' are used to label the axes in 2D or the local axes for the texture on the object and therefore differentiate them from the ones in the global 3D Space called 'X', 'Y' and 'Z' [Mac00, p.115]. UV-Maps are similar to fold layouts used in paper crafting or sewing patterns [Bec15, p.367]. They are generally made up of several parts called *UV Islands* or *UV shells*. These maps describe how the surface of the 3D object is laid out on a 2D plane. Depending on how well this is done, though, there may be some 'stretches' as the segments do not always flatten out. 3D artists use special test textures made up of different colored, numbered squares with which such spots become very apparent as the texture appears noticeably warped on the model.

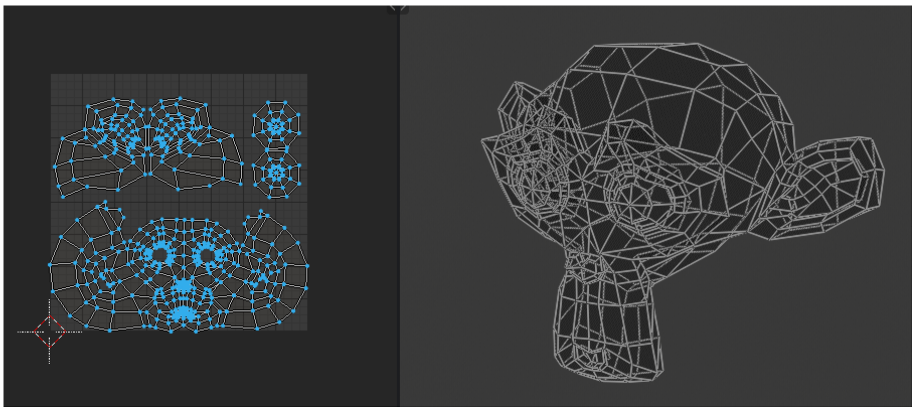


Abbildung 2.2: 3D Model (right) with UV Map (left)

Textures are used to manipulate various aspects of the surface material (cf. Fig. 2.3). This is why, depending on what the object is made of and in which style the artist is working, they come in many versions. There is no exhaustive lists of all of the types there are, but common ones include Albedo Maps, Normal Maps, Roughness Maps, AO Maps, and many more.

Albedo-, Diffuse- or Color-Maps add detail to the surface, though they do not influence how the object is shaded [Bec15, p.372]. They mainly convey the local color of the surface, but may be used as a guide for other maps to determine where recesses or bumps might be located. The only differentiation between Color and Diffuse Maps becomes apparent when dealing with metallic materials in *PBR* workflows. PBR, short for Physically Based Rendering. This is because metallic surfaces behave differently from non-metallic ones when it comes to how they reflect light. With metal, light either gets reflected or absorbed. There is no diffuse reflection or subsurface scattering.

Normal Maps influence the shading of the surface without changing the surface geometry. The *normal* is the vector that runs orthogonal to a plane, or in the case of 3D objects, to a specific face or poly. The texture works in that the angle of said normal is encoded into the three color channels (RGB). The angles of the new surface normals are calculated by taking the original normal vectors and combining them with the 'vectors' stored in the texture [Bec15, p.373f.].

Ambient Occlusion Maps (or AO Map for short) determine how much a given point on the object is exposed to ambient light. It is 'based on the observation, that shadows in reality appear in the places in which objects leave little room for the incidental light to disperse' [Bec15, p.319].

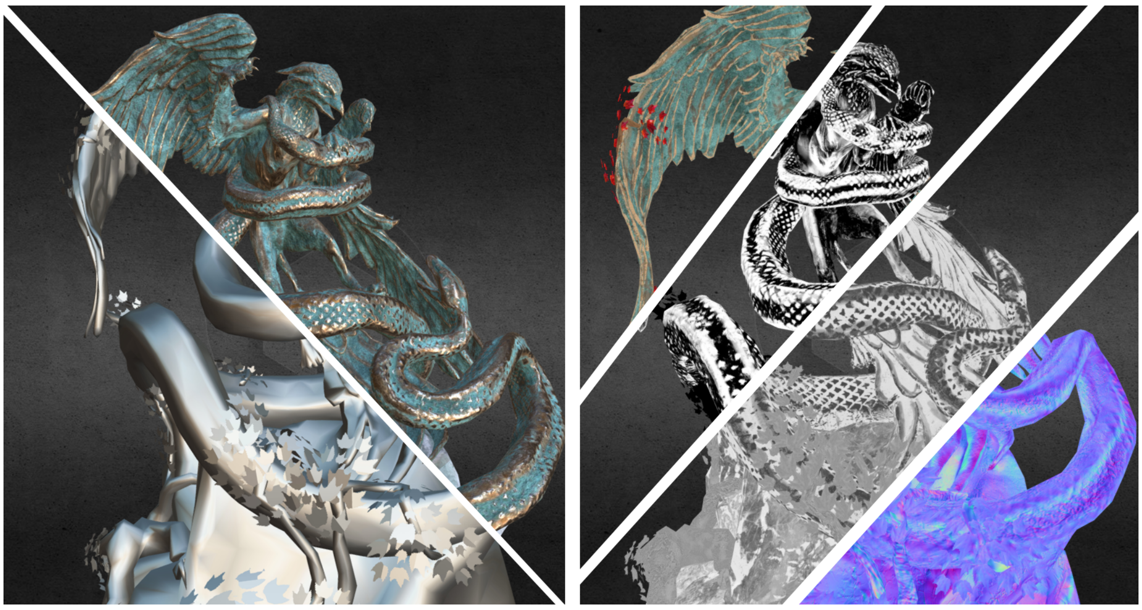


Abbildung 2.3: from left to right: model without textures and fully-textured; Color Map, Metallic Map, Roughness Map, Normal Map; Sculpture provided by Michael Kuwilsky

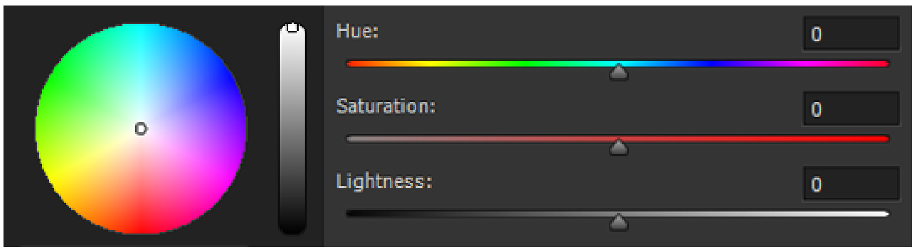


Abbildung 2.4: Color Wheel in Blender with Value Slider (left) [Fou98] and HSV Sliders in Photoshop CC (right) [Ado90]

2.2 The Basics of Color

Textures encode information in the form of color. To understand how textures themselves work, it is important to know how colors work in the digital space. This information is also useful for getting into the mindset of a painter.

Images are made up of color channels, the number of which depends on their color mode. The most common one used is RGB (cf. Fig. 2.4, left) - short for Red, Green and Blue, the three primary colors of the additive color spectrum or the colors of light. For each of the primary colors, there is a channel, in which the appropriate value (between 0 and 255) of each pixel in this color is stored [Mac00, p.93]. The 'images' of these channels are therefore displayed in graytones. There may also be a fourth channel that is called 'Alpha Channel' which stores the opacity of each pixel as a value between white and black.

For painters, it is generally easier to think of color in the terms of Hue, Value and Saturation or HSV (cf. Fig. 2.4, right) for short which is another color mode used in graphics applications. The color is defined through its hue - or its place on the spectrum of light -, its saturation - which defines how bright or dull a color appears - and its value - how light or dark the color is.

2.3 Requirements of a Video Game Production Environment

In order for the game to meet the requirements of immersion and a smooth gaming experience mentioned initially, the assets that are part of it also have to adhere to them. That is why not just the programmers but everyone who is part of the development process and makes assets is responsible for optimizing them to do so [Gra09, p.3].

This means for textures that they have to work efficiently while also looking good in the game. Factors that influence this are e.g. the file size, the amount of textures in a scene and texture resolution. Beside the textures themselves, other techniques have also been developed over time to tackle these issues. These are in example: MIP-Mapping, Atlas Textures, Texture Splatting and Multitexturing [Ahe17, p.6].

MIP-Mapping is a technique used to texture objects that are at a distance to the camera. The further away they are, the less screen space they take up and the less resolution is needed to depict them. Higher detail makes the texture look worse as it can not be distinguished at this distance and only serves to let the object appear more noisy and unclear. Unlike procedural generated textures, image-based ones can not be adjusted in real time. To avoid this problem, artists make multiple versions of them at various levels of detail. Algorithms are then used to interpolate between those stages for a smoother feel in-game.

Atlas textures contain the surface information for multiple 3D objects. This reduces the number of files that has to be loaded into the graphics memory. It does require though that the layout of the texture is well thought out, so that the artist can find all the UV shells that belong together quickly while the space as still exploited as efficiently as possible.

Texture Splatting is a method to get more variety out of a small number of textures. By themselves, small textures can not provide the necessary variety to make a large surface interesting to look at without the files taking up too much space. That is why the individual textures are combined to avoid repetition. *Multi-texturing* is used to achieve a similar effect for a single object: It describes the use of multiple textures that add details like scratches or dirt to the surface to break up the surface detail and add more interest.

It should be possible for the work to proceed quickly. Especially frequently used tools should be easily and readily accessible. Since pressing single keys or key combinations is significantly quicker than navigating through multiple menus, the use of hotkeys or shortcuts is heavily encouraged. Even macros and automating often used procedures or building customized menus can speed up working substantially [Ahe17, p.47].

Another aspect of working quickly is the use of non-destructive tools. They provide the possibility for a margin of error and quick changes without rendering the previous work unnecessary.

It is also important that the workflow allows for quickly establishing a rough approximation of the final result. This way, feedback can be received rather quickly before too much work is put in.

2.4 Texture Painting Workflow

Since Texture Painting is a form of Digital Painting, this section will examine some of the techniques of in this domain to gain a basic understanding of common work practices among artists.

Depending on the software and its configuration, there may be some steps to set up the workspace first to be able to paint. E.g. in Blender, the user has to switch to texture paint mode and make sure that the model is prepped by unwrapping it and adding a material with a texture to it to paint. In ZBrush, the artist has to switch to RGB mode for the brush and disable any options that add to or take away from the sculpt itself as well as apply a base color to the object or else it will continuously swap color whenever a new one has been selected.

After setting up the workspace, the first step is to establish the base color. This can be achieved by filling everything with a single color or a gradient. The previously created masks and clipping layers ensure that the artist does not have to be careful about drawing in areas that are not meant to be colored.

The second phase is 'Color Blocking' in which very broad strokes are used to establish an initial shading of the object. In this stage, the tools and colors are switched frequently and the size, opacity and hardness of the brush are often adjusted on a whim. That is why it is important that those changes can be made quickly and easily.

In the last phase, the main goal is to tighten up the work, meaning the final details are added. Besides that, the whole texture is continuously scrutinized for any inconsistencies so that the object looks more uniform in the end.

2.5 Tool requirements for painting in 2D

After examining these different workflows, a few principles can be distilled of what makes a good environment to paint textures: The tools should enable the artist to work quickly. The artist should be able to employ non-destructive techniques, so that, if any errors occur or changes need to be made, not all work should be lost.

For working quickly and efficiently, there need to be tools to quickly select large areas and fill them in. Depending on the on the process of the painting, demand may be higher for rudimentary but quick tools in the early stages to get the idea down rather quickly and gather feedback to either make the necessary changes and course-correct or abandon the work and start anew without losing too much time.

In image manipulation software such as Photoshop or GIMP, various selection tools can be utilized to swiftly mark an area that can then be blocked in with the fill bucket or a gradient. Depending on the context, there are a few options to choose from that very from quick and easy to use (Marquee Tools in various shapes) to requiring more control, but allowing for more flexibility as well (Lasso Selection).

Of course, the more the image is specified, the more polishing is needed to bring out the details and emphasize its appeal. At this time, tools to make small, precise changes are of more importance. This phase takes a lot more time to complete, though, which is why the artist should take care to keep all areas of the image to a similar level of completion and avoid putting in too much time when there still might be changes that need to be made. It is crucial, therefore, that the artist is not further slowed down by the tools. Parameters that are changed frequently, such as color, size and opacity, should be easily and quickly accessible and adjustable.

In this scenario, applications like ZBrush and Krita offer a handy palette that can be summoned at cursor location with regularly required elements such as brush selection, color palettes, a color wheel as well as slider to adjust size and opacity of the tool. Photoshop in example offers the option to hold a key and move the pen when using a tablet to adjust the size and opacity, depending on the direction the pen is moved. This negates the need to precisely hit a slider to move around and adjust the parameter.

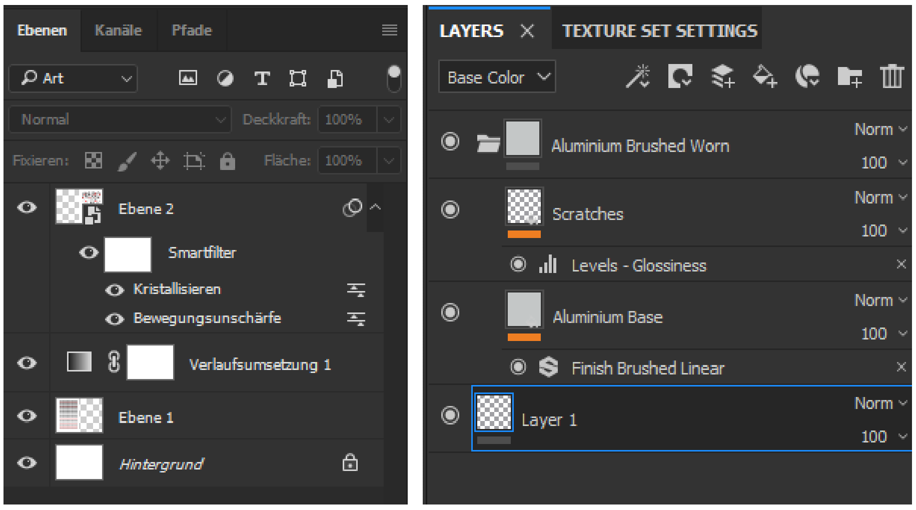


Abbildung 2.5: Layer Systems in Photoshop and Substance Painter

Lastly, to work non-destructively, data needs to be preserved. A mistake made should be quickly reversible. Experimenting with the color to look for a better fit may not impede the work already done up to this point.

By using *layers* (cf. Fig. 2.5), the artist can work on one 'level' of the image without compromising the work done in the other ones. Layers are comparable to transparencies that can be drawn on [Wol12, p.353]. They are used to separate parts of the image, and can be shifted around as needed. It should be noted that the pixels on each layer will obscure the image data on the lower levels, depending on its opacity. The final image is the sum of all its layers [Wol12, p.354]. How the layers interact with each can be manipulated through the use of layer modes, masks and adjustment layers.

With masking, it is possible to hide the pixels on a layer without erasing them, so that they can still be used later on if needed. This is achieved by controlling the visibility of the layer through a separate 2D plane that is only painted in tones ranging from black to white, where black areas are hidden from view and expose the layer underneath while the white areas are fully opaque.

Clipping Layers can be described as a special form of masking as they use the pixel information from the layer beneath to limit the area that can be shown when painting. This is especially useful when applying shadows and highlights to a base color as there is no need to care about 'drawing over the lines'.

Adjustment Layers may be used to adjust the color of the pixels of the layers below. They are special in that they themselves do not necessarily contain pixel data with the exception of using a Layer Mask to only affect certain areas of the image. Since they are separate from the other layers, they can still be deleted without losing too much time should they prove to not deliver a satisfying result.

Layer modes work in a similar manner, only that they affect a layer directly. They provide the formula to calculate how the color of the pixels said layer contains are interacting with the pixels of the layers underneath. Depending on what is selected, the result may darken, lighten or intensify the colors of the image.

3 Tool Analysis

Now that a basic understanding of the workflow requirements for production environments as well as the tools used for such has been established, it is time to take a look at how well this problem is approached in Blender.

To recap, in the last chapter, it has been determined that a production workflow for hand-painted textures has to ...

1. enable the artist to work quickly and comfortably.
2. allow for a fast approximation of the final result, so that feedback can be given as early as possible.
3. leave a margin of error for quick changes that may be needed after feedback has been given.
4. be as non-destructive as possible.

Besides analyzing Blender, a look at other applications is also needed to see how they manage the challenges of texture painting in their own ways. On the 3D side, there are Substance Painter and ZBrush to examine. Unlike Blender though, both of them are specialized applications: Substance Painter was created for surfacing objects with a focus on texture painting and procedural texture generation while ZBrush is known for its sculpting capabilities and the fact that it is optimized for working with a pen tablet.

Additionally, 2D graphics software will be scrutinized too: specifically Krita (Open Source) and Photoshop (Industry Standard).

Last but not least, there are also plugins used in conjunction with applications. In this case, these are BrushBox and Lazy Nezumi Pro. BrushBox is an addon specific to Photoshop which makes it easier to manages the brushes used in the software. Lazy Nezumi Pro is a plugin that can be hooked onto any program with painting capabilities and manages the stabilizer functionality of the brush.

3.1 Texture Painting in Blender

As a 3D Suite, Blender provides the option to work in 3D as well as 2D. This has some significant advantages when painting textures. For one, **the artist can see at any time how the asset will look like with the texture applied** and is able to turn the object to spot possible mistakes. This makes it easier to get feedback as well.

There is also the advantage of no further guesswork in which UV shell belongs where on the object [Gra09, p.547]. Any distortion present in the UVs is also accounted for as the 3D software automatically makes the appropriate adjustments to compensate for this. Additionally, because the UV shells are directly beside each other on the model and not separated as they are in the UV Map, it is easier to paint across multiple seams to hide them [Gra09, p.548].

For setting up a non-destructive working environment, Blender offers a Node Editor. This can be used to build systems that function similar to layers and layer masks, complete with different layer modi. These make it possible to combine different textures in a way that they can be 'stacked on top of each other'.

To keep the Node Tree in order, Frames or Groups are used to create more visual clarity on which nodes are responsible for which effect. All in all, it is important to maintain a consistent naming convention to avoid unnecessary confusion further down the line.

After setting up the workspace, the next step is to fill in the base color as well as rough in some initial shades and lights to quickly gain a clearer picture of the later result.

For adding the base color, the face selection mode along with the various selection modi and the fill tool is a quick way. With this, it is possible to lay down one solid color as well as different kinds of gradients.

When blocking the first rough brush strokes, it is possible to quickly switch between active and secondary color. There is also a color picker to select the color directly from the 3D View itself as well. The radius (or size) and strength (or opacity) can be adjusted comfortably too by pressing the appropriate hotkeys and moving the cursor left to reduce the size and increase the strength while moving it right does the opposite respectively (cf. Fig. 3.1). The hardness of the brush can be indirectly

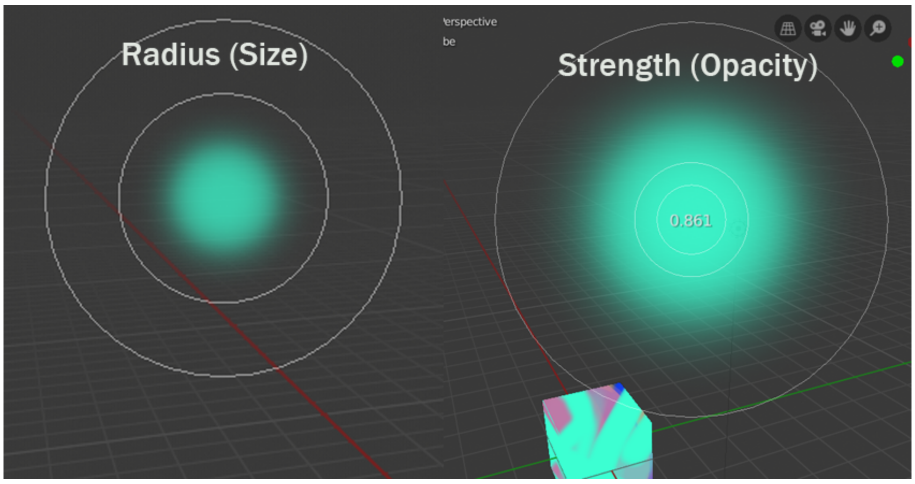


Abbildung 3.1: Adjusting the Radius (Size) and Strength (Opacity) in Blender

adjusted as well with the help of a curve diagram which depicts the opacity of the brush from its center to the edge of the circle, though this may be used to generate other effects as well like decreasing the opacity in the center or alternating its strength to draw rings instead of circles.

With the help of Face Selection Mode and several selection options as well as the Fill Tool, a base color or gradient for the different components of the texture can be quickly established.

3.2 What Blender is lacking in terms of Texture Painting

For all the features listed in the previous section, Blender still has several shortcomings when it comes to its texture painting capabilities. These weaknesses will be described in the following.

Context-Sensitive Palette (customizable)

Blender could employ more context-sensitive menus that open at mouse position. In 2.8 there is already the option to open a small menu with sliders for adjusting the radius (or size) of the brush as well as its strength (or opacity). There is a menu to switch between tools as well (draw, soften/sharpen, smear, clone, mask) that can be opened per hotkey depending which option is chosen in the user preferences.

Making the menu context-sensitive ensures that only the currently viable options are shown and keeps the interface simple. Additionally, there could also be an option to further simplify or customize the menu through check-boxes or drag and drop functionality.

Tools

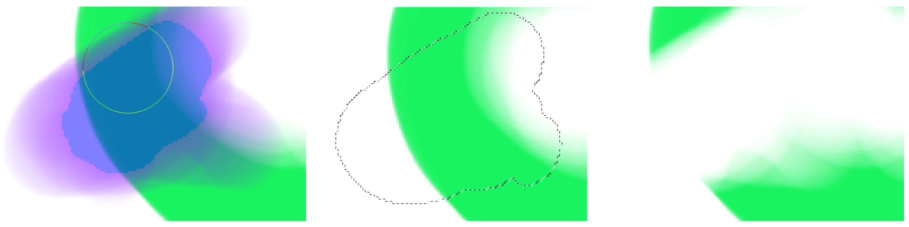


Abbildung 3.2: Masking in Paint Tool Sai

The area that the tools can manipulate at once is of a circular shape by default and cannot be changed. This means that any alpha that is loaded into Blender should be optimized for that, or else the artist has to work around any unwanted edges that are caused by the shape of the alpha as images are of a rectangular shape by default. This is especially noticeable when combined with the curve function to adjust the opacity at different distances from the center of the brush.

Some of the tools also do not work as well as equivalents in other application. I.e. the mask brush is painted on the model in ZBrush by keep CTRL pressed while painting and can be deleted by selecting an area beside the object that is empty. In Paint Tool Sai, it is also possible to paint in the mask which is easily discernible from the rest of the picture because of its blueish purple color in mask mode which disappears while painting and gets replaced by the dashed outline that typically appears around selections in graphic software (cf. Fig. 3.2).

The clone tool in Photoshop requires the user to first choose a source to copy from before painting. In BlackInk, the artist can import an image as a color source but this can also be used as a source for the clone tool. Another possibility would be to select an area on a mesh and save that in a kind of cache with a preview of the selection and use that as a source.

Node System

Blender already has a very sophisticated node system when it comes to surfacing an object. For texture painting though, it is not suited very well. For one, managing layers does not work as well as it does in applications like Substance Painter or Photoshop which are specially made for this task. To add a layer in Blender, the user has to manually add two nodes - a MixRGB and an Image Texture Node - , add the right file to the latter one and then connect them in the right manner to the existing nodes. Deleting or shuffling layers around

3.3 Alternative solutions in similar applications

While Blender does already contain most of the basic features needed for texture painting with a very versatile node system that enables layer management as well on top, it does still have some considerable deficits compared to other software that is used for painting textures. These applications are therefore worth taking a look at to see how they approach the challenges of this task and what can be done to improve Blender in this regard.

Brush Systems

Blender does have some options to make a set of brush varieties, though when compared to 2D graphics application in particular, it is still quite lacking. In this regard, Photoshop and BlackInk will be looked at as both of these programs demonstrate what is possible when it comes to brush customization in conjunction with the use of pen tablets.

Standard Graphic Tablets nowadays are able to support pen pressure, with some models even being able to recognize the tilt and rotation of the pen tip. These parameters can influence different settings of the brush and are generally used to emulate how “real” brushes lay down color, but do not have to be, as they can control other, unconventional variables as well. This is mostly done in applications like BlackInk which does not follow the philosophy of trying to imitate traditional painting mediums, but rather embraces the computer and graphics tablet as its own, unique medium with other ways of using tools and painting images.

Within Photoshop's 'Brush' Panel, there lies a wide assortment of choices to customize brushes for painting digitally. Beginning with the category of Shape Dynamics, these include options to alter general properties like the size of the tip or how blurry its edges are as well as settings to adjust its shape and orientation. Scattering introduces ways to randomize the manner in which the stroke is put down when painting, by adding variety to the size of each single 'stencil' or, as the name implies, 'scattering' them beside the area where the cursor actually is moved to paint the stroke. This haphazardness gets transferred to color attributes through integrating gradients and adjusting how the current fore- and background color are mixed into the stroke. With the possibility to also introduce textures and even the ability to combine two brushes with one another, the application allows a quite delicate control over the manner in which 'paint' is applied onto the digital canvas.

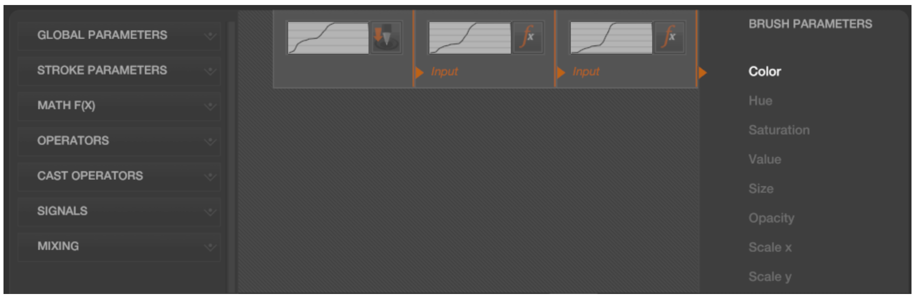


Abbildung 3.3: BlackInk Controller Editor

BlackInk goes a different route by employing a node system (cf. Fig. 3.3). For each attribute of the brush, a new node tree can be created which uses inputs of the pen tablet such as pressure or drawing speed and combines them with mathematic functions to exaggerate or downplay their effect. These cluster may then be mixed with any number of additional Through keeping each attribute of the brush stroke seperate, the resulting trees are kept relatively manageable and are thus easier for the user to interact with. It is also possible to go even deeper and directly edit the code itself. The brushes are developed in BSL (Brush Shading Language) which is based on HLSL (High-Level Shading Language). This enables the user to add their own settings and determine how they interact with each other to create the desired effect with the Interface adjusting in real time when making changes.

Working with Graphic Tablets

ZBrush is a 3D application with a focus on sculpting. Since this kind of work is faster and more comfortable when using a stylus, ZBrush is optimized for a workflow that relies heavily on the use of a pen tablet and keyboard shortcuts which can be mapped to the tablet as well if it features shortcut keys.

This is useful for texture painting too. Brushes can for example be picked from a dedicated menu using key combinations to select the right one. Tools like Masking, Smoothing and 'Erasing' are activated by keeping the corresponding key pressed while painting. A seperate brush can be selected for them as well, so that there may be four brushes to use at once.

Curve Mode in ZBrush is another way to work non-destructively. Like the name states, it allows the user to put down a curve first before applying color to it. Even after this step, this path can still be manipulated by moving it to another part of the model or changing the settings to create another type of stroke like dotted lines or drawing a thicker or differently colored line.

Presets

Having the option to customize tools in so many ways gives the artist a lot of control, but it is also a very time consuming process depending on the amount of settings that can be adjusted. Through the use of presets, this can be partially circumvented. Software that comes with premade brushes enables the user to start working instantly without having to set up the tools first. They can also be reused on other projects when saved, so that the setup process does not have to be repeated time and time again and the settings are assured to stay the same unless changed willingly. Another benefit of this is the possibility to export these files and share them with other artist, allowing for outsourcing of the setup process.

Brushes are not the only tools that can be made into presets, though. The same goes for palettes, Smart Objects and Smart Filters in Photoshop as well as Smart Materials and Smart Masks in Substance Painter. In addition to being reusable, they have the benefit of being a non-destructive way of working.

While working on projects and creating presets to use again later on, this collection can grow quite large. At this point, asset organization becomes quite important in order to profit from the benefits of both the amount of presets as well as quickly accessing the right tool at the right time.

Brush Box is a third party add-on that organizes Photoshop brushes into various subsets, providing the option to sort them e.g. by project, production stage or medium. This saves time when working on a project that takes a longer time or requires multiple assets with the same stylistic language, both in the tools employed and the colors chosen. Substance Painter comes with a "Shelf" and a selection of presets categorized by materials, textures, alphas etc.

Combining Hand-painting and Procedural Workflows

Substance Painter is the go-to software when it comes to surfacing an object, as it provides tools both for working procedurally as well as hand-painting, even enabling the artist to take a hybrid approach to the process. Often this results in a workflow that relies heavily on procedural generation for the establishing of the look, with hand-painting used sparingly to finish up the surface as well as touch up any bad decisions the algorithm may have made. This way, the artist can work quickly and non-destructive for the majority of the time and gets the option of using materials for other texture sets as well while still being able to make the call on how the object looks like in the end.

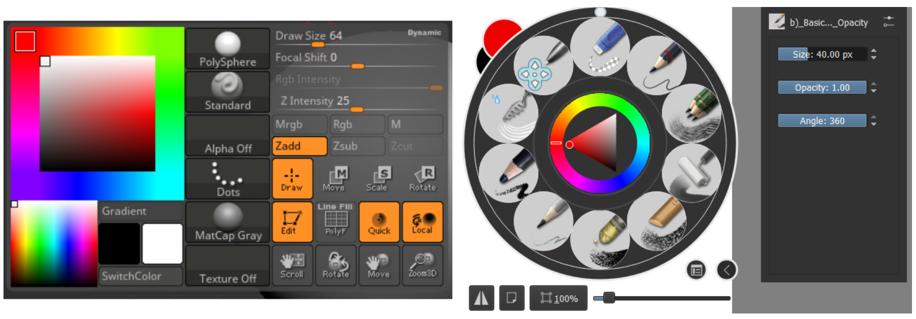

Draw Size 64
Focal Shift 0
Z Intensity 25
Size: 40.00 px
Opacity: 1.00
Angle: 360

Abbildung 3.4: Tool Palettes in ZBrush (left) [Pix99] and Krita (right) [KDE05]

Adjusting Brush Settings

In both Krita and ZBrush (cf. Fig. 3.4), the user can summon a menu with options to change brush settings at any time. These tool palettes are especially useful as they negate the need to have to navigate to the appropriate menu first in order to manipulate a certain slider. Even though this may not take that long, such actions are taken quite frequently and can add up over the course of the working process.

Especially when it comes to brush selection in Krita, the user is not boggled down by the task to locate a certain tool amongst a collection of hundreds of other brushes,

Pressing the SHIFT or CTRL-Key in ZBrush activates the Smooth or Mask Tools respectively. Many applications also have the option of activating the Color Picker per Hotkey to select a certain color that already exists in an image quickly.

Stabilizer

Working with a stylus does not always go smoothly. Especially when the pen is moved rather slowly, the strokes that are put down may render as very uneven. This is a problem as lines should go down as the artist intended and not have to be redrawn time and time again to achieve an adequate result.

Stabilizers take care of this problem by slowing down the stroke and making it feel more ‘weighted’ to the user. This means the mark is not as affected by small tremors, but rather follows an approximation of the movement of the hand, depending on how strong the influence of the stabilizer is set. At that point, the functionality of most in-software variants of this tools ends, but Lazy Nezumi also has additional features to offer.

Lazy Nezumi is a third-party software that works by attaching it to the application that is used to draw. It comes with a selection of presets and a lot more options to choose from when creating a stabilizer setting for a brush. It allows for the use of rulers and other guidelines to restrict the user to drawing straight or circular lines when needed. There are also settings for various perspective guidelines and fractal patterns. In addition to this, Lazy Nezumi further allows users to write scripts themselves.

4 Requirements and Specifications

Now that the basic concepts of texture painting for video games have been explained and a look at the capabilities of various software packages has been taken, it is time to look into the possibilities of creating an add-on to implement one of the features that could improve the workflow for texture artists in Blender.

For this, it is essential to become familiar with the Blender Philosophy and what makes a good User Interface in general and how these principles can be applied to the appearance of the add-on. Furthermore, a first look will be taken at how the Blender Python API is constructed.

4.1 Blender Philosophy

For the Add-on to integrate well into what already exists, especially regarding its usability, it has to follow the overall philosophy of the software. First and foremost, Blender is a tool for artists. This means that the main focus of any of its features should be to support the creative process. In order to achieve this, a lot of the functionalities are developed in the process of making the so-called Open Movies or other 3D productions. Of course, it is also important that it can be expanded and that the code base is stable. However, should these goals conflict with Blender's main focus, the latter takes priority.

This abstract vision of the software is further broken down into more concrete principles that are applied when designing the interface. These guidelines are: simplification, consistency, customizability and accessibility [Bec15, p.25f.].

Simplification and consistency are expressed in how the elements are arranged and in their behavior [Bec15, p.25f.]. Any buttons and features are grouped in a sensible manner to make them easier to locate. There are presets for different workspaces depending on what the current task is, be it animation or UV editing, in example. In case a feature can not be found, it is also accessible via the central search function. This is also useful in case the user forgets the shortcut for a specific function, as the hotkeys are noted just beside the feature they correspond to. Of course, this way of searching may also be chosen to locate a specific action more quickly, as the menu spawns at cursor location and can be manipulated by just using the keyboard.

Context sensitivity is another pillar of the application's operation. As such, the contents of the menus change depending on the current object interaction mode (Object Mode, Edit Mode etc.). Actions are also performed depending on where the mouse cursor is currently located. This is combined with consistency as the same hotkey performs similar, but not equal actions depending on the context.

As an Open Source software, Blender is very customizable. This is not only expressed in that the code is freely available and the Python editor that comes with the program, though. The GUI itself can also be freely manipulated depending on which kind of editors are needed at the moment. Workspaces can be added and customized and the current state of the project is saved with all these configurations to reduce friction when the project is opened up the next time. These changes may be saved as a startup file, as well.

While working, Blender is conceived to be non-modal [Bec15, p.39]. It will not hinder the progress through dialogues requiring additional input, time to save or execute. With the way the many editors are handled, there are no more than one window at any time. Consequently, time-consuming rearrangement of overlapping menus is not necessary. All relevant options are displayed at once in the GUI [Bec15, p.39f.] and may only be hidden when not essential for the task. Dialogues, too, are almost completely redundant. Any action taken is promptly executed, requiring no further data input. This is achieved with sensible presets that can still be adjusted afterwards. The scene is thereby updated in real time to instantly show changes.

The more familiar the user becomes with the software the more they will come to rely on the hotkeys to speed up their process. Working like that has the added benefit that the UI elements used prior to this can be concealed to free up screen space[Bec15].

In this way, the software is being optimized to make creating for the user as fast and comfortable as possible. The learning process for new features is also made easier by a consistent behavior and look of the user interface. Once the user has understood how an element responds, that knowledge can be applied 'globally' which further reinforces said knowledge.

4.2 Principles of Good Usability

A human interacts with the software via the User Interface. The User Interface is only part of what constitutes the User Experience (UX).

Usability - all software has usability. Some are more usable, others less so. Usability design is based on the same principles as good interpersonal relationships, namely: Respect, Confidence, Support, Understanding and Communication [Flo15, p.11]. These factors have to be considered for every aspect of the User Interface and often, a compromise between two or more of them has to be found. Failing to do this may lead to a User Interface that is frustrating, unreliable, ...

A well designed User Interface leads to a reduction in learning time and fewer mistakes. It takes less time to complete tasks and overall user satisfaction is increased.

'Respect is achieved through expertise, appropriate behavior and knowledge about one's own and others strengths and weaknesses.' [Flo15, p.16] Humans treat each other in a respectful way when nobody is mistreated through actions taken or words said. For software, this means that problems are taken seriously

'Trust' is another important factor when it comes to software. The user has to rely on the program to work as expected when performing an action. Moreover, the application should crash as little as possible, if at all, to avoid losing progress. Especially in a production environment. easy to lose and hard to gain [Flo15, p.17]. It is build by demonstrating reliability and being congruent in what is said and what is done.

4.3 Add-on Usability

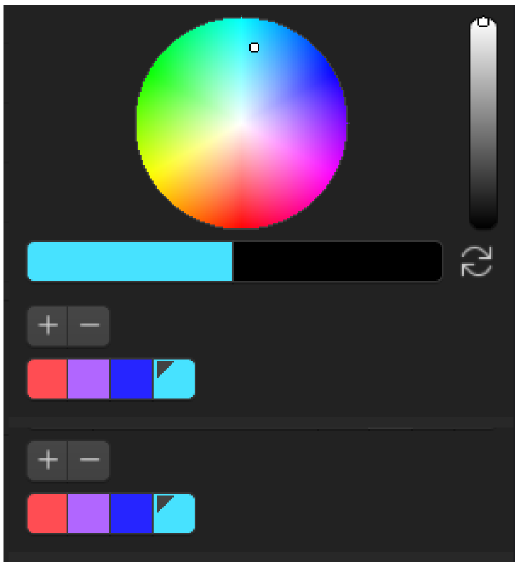


Abbildung 4.1: Addon UI sketch

For the add-on that is to be developed, the author has decided to keep it relatively simple. Of course it would be possible for the menu to contain a lot more options, however this would only end up convoluting the space too much and slow the user down because they would be confronted with a wide palette of choices when they only want to change a certain setting. In this case, the author has decided to go with a menu to change the color that opens at the current position of the cursor (cf. Fig. 4.1).

As it is a menu that opens up when a certain button is pressed, it is very important that the context is considered. It would not make much sense to open up an interface like this while the user is manipulating a 3D object in Edit Mode because they do not have a need for color to do that. Blender also relies heavily on the use of hotkeys and has a lot of features that need to be covered by them, so the context should be as specific as possible to not block keyboard shortcuts unnecessarily.

On the topic of hotkeys, there is another reason besides ease of use and speed why they are a more favourable way of working than using menus: They free up screen space. Blender may allow the user to split up the interface to their liking but when trying to work on the model and depending on the screen size they have at their disposal, the more space available to work on the piece itself, the better. A menu that is summoned per hotkey is only there when it is needed and thus does not get in the way of the painting process itself. Of course, this also makes it possible to use the properties window to display another set of options while working.

The menu should also respond to the user in a way that they know what they are doing. It should synchronize with what is happening in the properties panel when the user is changing the color, meaning the pointers on the color wheels should move together, the colors of the brush should change accordingly, the color should change when another brush is selected (except for the case when unified color is activated) and the color palettes should be manipulated in the same ways as well.

Lastly, the UI elements are arranged in a way that they relate to each other and make sense. For this menu, it means that the primary and secondary color are bundled together with the button to switch between them. At the same time they are separated from the options that pertain to the palettes beneath, like the palettes themselves as well as the options to add to and delete colors from them.

4.4 Python in Blender

To develop an add-on for an application, it is crucial to gain insight into how the software's architecture is structured and what methods are available to realize a feature. Blender itself is coded in C, but provides the option to extend and customize the software providing an API for Python 3 - called Blender Python or bpy for short.

Python has the advantages of being a more high-level and thereby easier to understand language than C, making it quicker to pick up for users to start writing their own scripts and automate some of their work processes. Unlike other programming languages, Python does not rely on parenthesis to separate code blocks but instead uses indentations - which are commonly used to make code more readable - as a way for the compiler to determine where the body of a function begins and where it ends [Tro12, p.64f.]. C, on the other hand, is much more efficient as a low-level programming language since it does not have to go through an interpreter first like Python. This way, Blender has both the advantage of a powerful code base that handles the parts most critical to its performance while allowing users to write their own scripts and keep them up-to-date more easily without having to dive into the source code [Kai12, p.903]. Like Blender, Python is also an Open Source language and available for free [Wal07, p.14].

Some systems are provided to assist with the development of customized scripts and add-ons. Beside the Blender Python API documentation [Tea19b], these are the Python Tooltips, the Info Panel, the Text Editor, the Python Console, the Edit Source Function as well as a number of templates.

The Tooltips and Info Panel are helpful in that they provide the command that is executed when interacting with an UI element or that a certain hotkey triggers when pressed. This is convenient for automating tasks or executing operators from the Python Console, as a large number of objects can be manipulated at the same time without being manually selected and having a method executed on them.

The Text Editor is, as the name applies, the main component for writing the scripts themselves, while the Python Console provides a way to quickly test out single lines of code. While the bpy module is already imported by default into the Console, in contrast to the Text Editor wherein each time a script is executed, it is an independent session that does not import the bpy module by default [Con17, p.15].

The process of coding may be sped up further through Auto-Completion which lists all of the available methods for the input that has already been given. With 'Edit Source' and the Templates, Blender offers a way to directly take a look at the python code that is already integrated and analyze it to learn how the Python API works in this application. Any additional information can be found in the documentation.

These tools all help to grasp how the Python API is structured. It is made up of a number of submodules, most notably bpy.ops, bpy.context, bpy.data and bpy.type. There are several other modules as well, though these are the ones that will be the main focus in add-on development.

The Blender Python API is made up of a number of submodules that are called to access various parts of Blender, the current .blend file as well as the methods and variables provided by Python 3 itself.

bpy.data gives access to the internal data of Blender - it contains all the data that determines an objects shape and position [Con17, p.12].

bpy.context is used to access the objects that the user is interacting with. It is a reflection of the Blender Philosophy of 'Context matters' and works by creating references to bpy.data based upon the object's current state within Blender. Because of this, all the context values are read-only. They can be modified though by accessing them through the bpy.data module or by running operators [Tea19b].

bpy.ops contains the operators and primarily functions for manipulating objects [Con17, p.11] including ones written in C, Python or macros [Tea19b]. They do not really have return values but rather specify upon finishing how the task ended, if it was successful or cancelled.

bpy.types is first and foremost interesting when it comes to registering UI elements.

5 Implementation

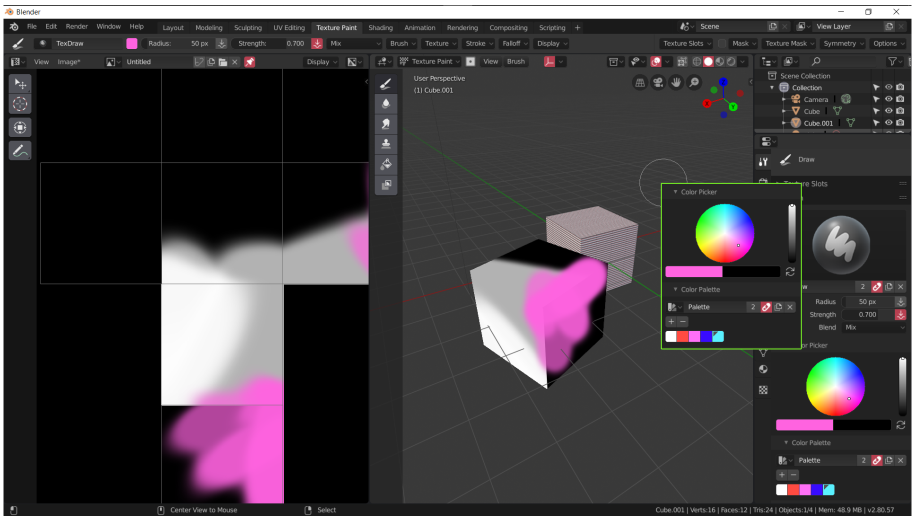


Abbildung 5.1: Blender Color Palette at Cursor Position

The plan is to create a palette similar to the ones that exist in ZBrush and Krita, that opens up at the current location of the cursor when a certain hotkey is pressed (cf. Fig. 5.1). This way, the artist does not have to seek out the menu where the settings are usually situated anytime they need to change them. This saves the time of moving the cursor to said location and does not break the flow more than necessary.

Since there is already a menu to adjust the size (or 'radius') and opacity (or 'strength') integrated in Blender 2.8, this add-on will focus on providing a Menu to change the color while painting by containing a Color Picker with a wheel and value slider as well as the current primary and secondary colors and the palettes of the current project. The latter is normally only shown one at a time, with a drop-down menu acting as a way to switch between them. These settings should of course synchronize with the ones located in the Properties View and change depending on which brush is selected. The menu should also only be accessible while the user is in Texture

Paint Mode and has either the ’Draw’ or ’Fill’ Tool activated, as these are the only tools that use this setting.

Of course, this add-on could be expanded by integrating more options, however this would only unnecessarily clutter up the space and make the menu less comfortable to use as the artist would have to pick the right option they want to change first.

5.1 Add-on Code

Listing 5.1: Add-on head and info

```

import bpy

class ColorPalette(bpy.types.Panel):

    bl_info = {
        ”name”: ”Color␣Palette”,
        ”author”: ”Patricia␣Liess”,
        ”location”: ”View3D␣>␣UI”,
        ”version”: (1,0,0),
        ”blender”: (2,8,0),
        ”description”: ”Summon␣a␣color␣palette.”,
        ”category”: ”TESTING”
    }

    bl_label = ”Color␣Palette”
    bl_idname = ”PAINT_PT_layout”
    bl_space_type = ’VIEW_3D’
    bl_region_type = ’UI’
    bl_context = ”imagepaint”
```

In order for the script to be able to work in the first place, the required modules have to be imported before the class is even defined. At the very least, the Python API (bpy module) is needed to work with the Blender-specific functions. After that, the class itself is defined starting with its name *'ColorPalette'* and the class it inherits from *bpy.types.Panel*.

The first lines of any Blender add-on should also contain the variables that blender needs for internal purposes which are tagged with the prefix 'bl' to distinguish them from user specified variables that are used in the code proper to describe the operations performed. Part of this is the bl info Dictionary which must appear at the start of the file as Blender parses it from the first 1024 bytes of each file to fill in the meta-data about the add-on in the User Preferences Menu [Con17, p.69]. This info contains the name of the add-on itself as well as that of the author while informing the user of what the add-on does, where to find it within the application and what version they need of Blender to be able to run it. Lastly it states whether this is an official blender add-on, one which is supplied by the community but not yet included in the release or if it is still in the testing phase.

Beneath this info-block, a few more blender variables are specified to define the context in which the interface for the add-on is to be found. In this case, the panel appears in the 3D View at the cursor location when in Texture Paint Mode and is labelled 'Color Palette'.

Listing 5.2: poll function

```

    @classmethod
    def poll(cls, context):
        mode = context.mode
        cont = context.tool_settings.image_paint
        tool = cont.brush.image_tool
        cond1 = mode == 'PAINT_TEXTURE'
        cond2 = (tool == 'DRAW' or tool == 'FILL')
        conditions = cond1 and cond2
        return conditions
```

The *poll function* is a class method that takes in the variables cls which refers to the class itself and the current context which refers to the bpy.context and looks at the applications state at the moment the script is executed. The purpose of this function is to check whether or not the feature should be available at the moment. In this case, the conditions to be met are that the application should be in Texture

Paint Mode and either the 'DRAW' or the 'FILL' tool should be selected as they are the only ones that allow the user to choose a color for them. Should these apply, this method will return true.

Listing 5.3: Color Palette Add-On

```
1
2      def draw(self, context):
3          layout = self.layout
4          imgp = context.tool_settings.image_paint
5          layout.prop(imgp.brush, "color",
6              text='Main␣Color')
7          layout.prop(imgp.brush, "secondary_color",
8              text='2nd␣Color␣Color')
9          row = layout.row()
10         row.operator("paint.brush_colors_flip",
11             text='Flip␣Brush␣Colors')
12         row = layout.row()
13         row.separator(factor=2.0)
14         col = layout.column()
15         col.template_ID(imgp, "palette",
16             new="palette.new")
17         layout.template_palette(imgp, "palette",
18             color=True)
```

The *draw method* defines the palette menu itself. It takes in the variable self that is used to reference an instance of the class as well as context like the poll method did before. It starts of by defining a few variables to shorten the commands in the brackets later on. There is self.layout which takes an instance of the ColorPalette class that inherits its layout from panel to display the properties and palette later on. The variable imgp is short for the reference to the current properties of the texture and image painting mode [Tea19b].

The prop method takes in the current brush as a datablock from which to take the properties 'color' and 'secondary color' as well as the appropriate labels for them to display the color selection for the two colors of the brush in the interface (cf. l.5-8).

row.operator establishes a new row with a button to exchange the primary and secondary colors of the brush (cf. l.10) while the template ID method inserts a drop-down menu to switch between the palettes that are displayed underneath with template palette.

Listing 5.4: Color Palette Add-On

```
 1
 2 def register():
 3     bpy.utils.register_class(ColorPalette)
 4
 5 def unregister():
 6     bpy.utils.unregister_class(ColorPalette)
 7
 8 if __name__ == ”__main__”:
 9     register()
10     #bpy.ops.wm.call_panel(name=ColorPalette.bl_idname)
11     bpy.ops.wm.call_panel(name=”PAINT_PT_layout”)
```

The register and unregister functions are also part of each Blender add-on. They enable the user to freely decide whether or not they want to enable the feature within the Preferences. *'Call Panel'* (l.12) is an operator that is used to display the menu when the script is executed.

6 Evaluation

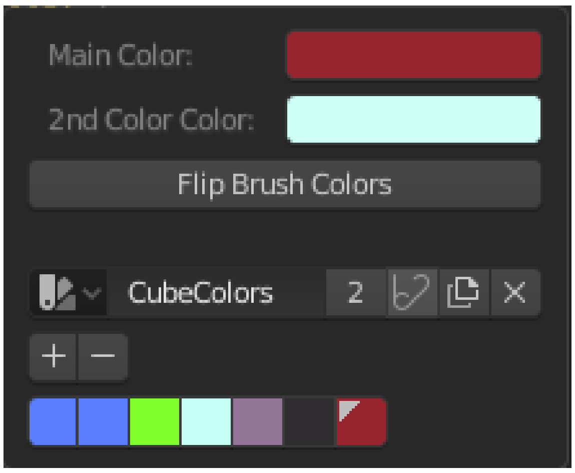


Abbildung 6.1: Final Add-on UI

The add-on fulfills all the goals that had been set for it in that the user can adjust both the primary and secondary color of the brush as well as flip their colors by clicking on the button beneath. Instead of showing all the palettes available though, the author decided to include a drop-down menu to select one active palette and have it displayed beneath. This way, the interface will not become to cluttered with palettes should there be more than a certain number of them (cf. Fig. 6.1). Another way of solving this issue would be to only select three to five palettes to show. Depending of how much space they take up, this could be a viable alternative. However, the user would still need to be able to exchange the palettes in some way.

The Color Wheels for the primary and secondary color are only shown when clicking on the color fields beside the labels. It is unclear at this point whether or not this is beneficial, as it does add one more click that has to be performed to access the color wheels, though on the other hand, less space is needed for the menu and when clicking on a color, the user knows which one they are modifying.

All the UI Elements work as expected in that they synchronize with the corresponding settings in the Properties Editor. The colors of the palettes can even be selected and deleted. New palettes may be added and named at any time. Though there has been an issue with the menu not displaying correctly - while still being fully functional - as a result of having the class inherit from bpy.types.Menu (cf. Fig. 6.2). This has been changed and adapted to fit bpy.types.Panel - that are also used in the Properties Editor to display the color wheel and palettes - which solved this issue.

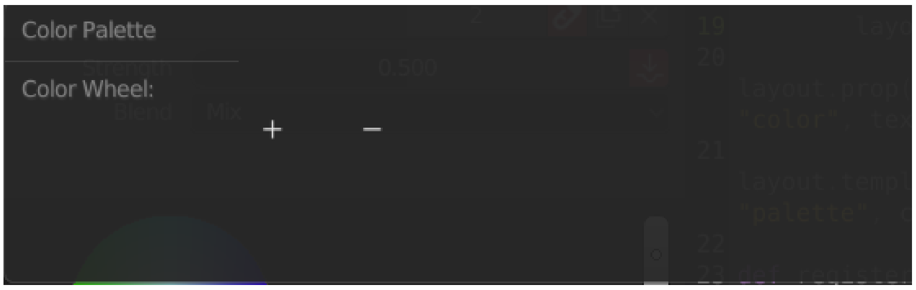


Abbildung 6.2: faulty Palette Interface

Developing the add-on took more time and effort than previously estimated. This is because the author of the thesis was neither acquainted with Python 3 itself nor the way it worked in Blender and had to familiarize herself first with the API. The various tools provided by Blender had lead her to the assumption that this task would be relatively straight forward which it was not.

The add-on did not quite achieve the intended effect of reducing the time for switching colors because the color wheel itself still has to be accessed by clicking on a button first. Depending on the layout of the application, there are no significant variations in the time it takes to set the primary brush color either in the new context menu or in the properties panel. This is also due to the fact that the add-on has not been fully implemented yet and can only be executed as a script right now and not yet used as effectively while painting.

However, ZBrush and Krita both offer the feature of a color palette that can be accessed through pressing a hotkey, so they will be tested in terms of how many times the process of changing the color and drawing a stroke can be repeated within a minute for both the color wheel in the context menu as well as the one in the side menu. [1]

Conditions	Number of Color Changes	
	Side Menu	**Context Menu**
Self-Test (ZBrush + Tablet)	33	28
Self-Test (Krita + Tablet)	29	20
Subject 1 (ZBrush + Tablet)	45	1st Try: 37 (2F) 2nd Try: 58
Subject 2 (Krita + Tablet)	13	6

Tabelle 6.1: Context Menu vs Side Menu Evaluation

After running these tests, some new insights could be gained (cf. Table 6.1). For one, the Context Menu in ZBrush was a faster way to change the color on second try then the side menu, but the first try shows that it is also more prone to making mistakes as the context menu in ZBrush contains a lot of other options beside the color wheel. One potential factor for the added speed may also be the fact that the Color Wheel in the Context Menu is a lot bigger than the one in the Side Menu.

Meanwhile, the experience for the second subject in Krita proved to be more frustrating, as the two color wheels acted in different ways. While the one in the in the side menu was stationary, the one in the context menu turned depending on the hue that was selected on the outer ring. Another confounder lied in the fact that the color could be picked from the wheel in two ways: Either by touching the pen

[1]Note: Color Wheel is hereby used to relate to the color selection tool used in graphics applications, which does not always take the form of a wheel necessarily. In ZBrush, it is a square surrounded by a border to change the hue, in Krita, it is a triangle that is surrounded by a hue ring

to the tablet's surface and moving it or by right-clicking (mapped to the pen) and only hovering the pen above the tablet while moving. The problem with this is that the menu also opens and closes through right-clicking by default, meaning the artist either has to work with the right-click option or deal with the fact that the color selected will switch the instance the menu is closed. The first option would hereby be more beneficial as the context menu is closed upon releasing the button, reducing the clicks needed to switch the color.

In general, the side menu was quicker to use than the context menu, though this is not only due to the fact that the menu has to be summoned first. The implementation of the context is just as important, as seen in the experience of subject 2. In the self-tests, another insight that has been gained when it comes to context color wheel in Krita is that when trying to change more than one things about the color - like shifting the hue and changing the saturation - it is better to first move the pen normally and then use a right click to make the last change while simultaneously closing the menu. This can take some time to get used to, as both the author and subject 2 do not have much practice in using Krita.

Through out-sourcing these settings, though, they are easier to get to for the user as they do not have to search in the Properties Panel first for the right area which also leaves the latter open to use for displaying other settings. There is the added benefit that the panel does not have to be open for painting. It is therefore optional and can be closed to free up more screen space for painting, meaning the artist is one step closer to working in full-screen mode with only having to call on the menus when needed.

7 Conclusion

A single menu for changing colors on the fly is not all that can be done to improve the texture painting process in Blender. There are several other deficits that can still be addressed and following are some ideas and concepts for further add-ons which mainly focus on designing for a more pen tablet supportive environment.

To conclude, the last section of this thesis further gives an outlook on how the development of Blender will continue as well as how texture artists may combine different workflows to use as their tools when surfacing an object.

7.1 Alternative Add-Ons

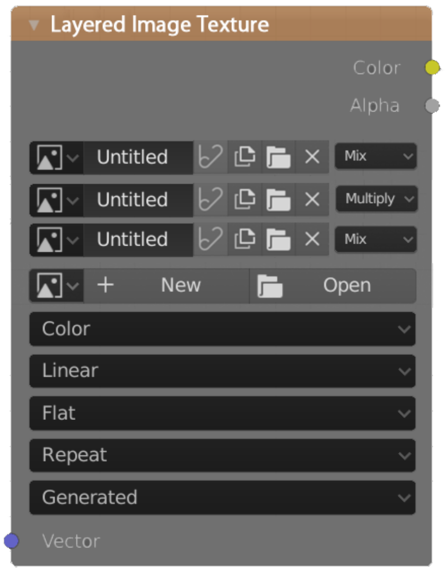


Abbildung 7.1: Layered Image Node

Layer Management

One potential way to enhance the layer management capabilities of the node editor while still keeping its flexibility of use in tact would be to introduce a new node that works the same as an Image Texture Node (cf. Fig. 7.1) , only with the added bonus of the option to add multiple textures in a simple layer system and even changing their blend modes.

This would eliminate some of the complexity for texture painting node trees as the textures can all be stored in one node and there are no further MixRGB Nodes needed for stacking the layers. The disadvantage of this concept though is that the user can not add more nodes between layers or between a texture node and a mix node which makes this concept somewhat stiff. Still, it is a possible alternative to consider and extend upon.

Another possibility would be to have some sort of simplified node editor management system or view where the nodes could be shifted around more easily while the more detailed view would still be there to adjust parameters or make other adjustments to fine-tune the graph. A mode such as this could also be used to move groups of nodes around more easily and connecting them automatically. Layers could be added, merged and deleted by the press of a single button instead of having to add them manually every time.

Managing layers could also be made easier by simply providing a small preview of the texture itself in the node so that the user knows which texture they are dealing with.

Navigation around the 3D Space

A big improvement would be a navigation system for the workspace that is more suited to work with a pen tablet rather than a mouse, like in ZBrush, where an artist can right-click (or holding the assigned button on the pen) and optionally keep CTRL or ALT pressed to move or rotate around an object or zoom in on it. This negates the need to switch between a pen and a mouse for painting and navigating - which has to be done a lot in 3D space to look for any discrepancies in the model or texture. Offering the option to translate around the point which has last been edited further removes unnecessary navigation to get back to the area of interest.

Hotkeys and Context Menus

The more screen space an artist has at their disposal for painting digitally, the better. This needs to be balanced with the ability to change their tool settings as they go though. Context menus that contains the options to do so and are accessible through hotkeys are one possible solution to this, like the color palette add-on developed for this thesis. Other options would be e.g. a layer menu to switch between textures to draw on.

These hotkeys can also be used directly too, for example to switch between or cycle through tools. A feature like the one in ZBrush could be implemented wherein the user holds down CTRL, SHIFT or ALT to employ the brushes for Masking, Smoothing or 'Erasing' (does the opposite of the currently selected brush for painting, in this case it draws with the secondary color)

In painting, value is the most important part of color [Cas13, p.15]. This is because the shape of an object is defined by how it interacts with light and where its shadows are cast. For this reason, digital artists often view their images in gray scale to see if there is anything to be changed about the values without the distraction of color. One way to implement this is to assign a color mode that displays only the values of every color to a hotkey so that the artist can quickly switch between both views.

7.2 Prospects

At the time of writing, the software is currently nearing the end of its beta phase for the new version 2.8. This new edition features a more sophisticated version of the 'Grease Pencil', a tool formerly used only to make annotations to the 3D scene or node editor. Now, it has been transformed into a whole palette of functionalities used for 2D animation that can be created within 3D space. The development team has already announced that some of its new projects beyond the finalization of 2.8 will be, among others, to add more texturing tools and make painting and sculpting improvements. (cf. Blender Developers Blog, 24.12.2018)

With its wide variety of use cases and emphasis on enabling artists to work quickly within an environment customizable to their needs, Blender is definitely a viable alternative to many of the more costly 3D creation suites. As of now though, there are still quite some tools that could be added to Blender to give the artists more freedom.

Texture Painting is, of course, just one way to make textures for games. Particularly as this can be a time-consuming process, depending on the amount of objects that need to be textured with different materials. Still, it enables the artist to directly manipulate the look of the game instead of letting algorithms do the work which is very important when creating the vision of the game the developer has in mind.

Another possibility is to use procedurally generated, painted and photographed textures in conjunction, which can speed up the texturing process. It may even lead to entirely new aesthetic styles in video games, depending on 'how' the image data is used.

Still, games with a more stylized aesthetic tend to not profit quite as much from the online collections of photo-realistic materials. Though there are a number of examples for procedural stylized textures as well, it may be not quite as easy to integrate them. This is due to the fact that they are stylized, and much more an interpretation of the artist than a somewhat accurate depiction of the natural world. Because of this, one premade texture may be more recognizable for its stylization and might not fit quite well with the look of the game.

Literaturverzeichnis

[Ado90] Thomas Knoll; John Knoll (Dev: Adobe): *Photoshop*, 1990, raster graphics editor.

[Ahe17] Luke Ahearn: *3D Game Environments. Create professional 3D Game Worlds*, CRC Press, Boca Raton, 2 Aufl., 2017.

[Bec15] Thomas Beck: *Blender 2.7. Das umfassende Handbuch*, Rheinwerk, Bonn, 1 Aufl., 2015, 1. korrigierter Nachdruck.

[Bri08] Ron Brinkmann: *The Art and Science of Digital Compositing. Techniques for visual effects, animation and motion graphics*, Elsevier/Morgan Kaufmann, Amsterdam, 2 Aufl., 2008.

[Cas13] Gilles Beloeil; Andrei Riabovitchev; Roberto F. Castro: *Art Fundamentals. Color, Light, Composition, Anatomy, Perspective and Depth*, 3DTotalPublishing, 2013.

[Con17] Chris Conlan: *The Blender Python API. Precision 3D Modeling and Add-on Development*, Apress, New York, NY, 2017.

[Ebe03] David S. Ebert: *Texturing and modeling*, Elsevier Science, San Diego, 3 Aufl., 2003.

[Flo15] Alexander Florin: *User-Interface Design. Usability in Web- und Software-Projekten*, Books on Demand, Norderstedt, 2015.

[Fou98] Blender Foundation: *Blender*, 1998, open Source 3D Software.

[Gra09] Andrew Grahan: *Game Art Complete. All-in-one*, Elsevier/Focal Press, Amsterdam; Boston, 2009.

[Gre17] Darren Grey: *When and Why to Use Procedural Generation*, in *Procedural Generation in Games* (herausgegeben von Tanya X. Short; Tarn Adams), Kap. 1, S. 3–12, CRC Press, Milton, 2017.

[Kai12] Johannes Ernesti; Peter Kaiser: *Python 3. Das umfassende Handbuch*, Galileo Computing, Bonn, 3., aktualisierte und erw. aufl. Aufl., 2012.

[KDE05] KDE: *Krita*, 2005, free open-source raster graphics applications.

[Mac00] Rüdiger Mach: *3D Visualisierung. Optimale Ergebnispräsentation mit AutoCAD und 3D Studio MAX*, Galileo Design, Bonn, 1. aufl. Aufl., 2000.

[Pix99] Pixologic: *ZBrush*, 1999, 3D digital sculpting tool.

[Sch08] Jesse Schell: *The Art of Game Design. A Book of Lenses*, Elsevier Inc., Amsterdam, 2008.

[Tea19a] Blender Documentation Team: *The Blender 2.80 Documentation*, 2019, URL: `https://docs.blender.org/manual/en/dev/index.html`, besucht am 07.02.2019.

[Tea19b] Blender Documentation Team: *The Blender Python API Documentation*, 2019, URL: `https://docs.blender.org/api/master/index.html`, visited: 02.04.2019.

[Tro12] Adam Mechtley; Ryan Trowbridge: *Maya Python for Games and Film. A complete reference for Maya Python and the Maya Python API*, CRC Press, Amsterdam, 2012.

[Wal07] Peter Walerowski: *Python. Grundlagen und Praxis*, Addison-Wesley, München, 2007.

[Wol12] Jürgen Wolf: *Gimp 2.8. Das umfassende Handbuch*, Galileo Press, Bonn, 1 Aufl., 2012.

Appendix

Glossary

Term	**Explanation**
Nodes	elements that perform a single operation on a composition and are connected to each other to form a graph
Blender Python API	(application programming interface) Interface between Python and C, to provide methods for accessing Blender's methods and writing scripts to extend the application or automate certain procedures
GUI	graphical user interface - enables the user to communicate with the system via the screen
Procedural Textures	use of algorithms and mathematic functions to define color and texture of an object's surface [Ebe03, p.1]
Textures	image files or algorithms that are used to help define the surface of a 3D object

Selbstständigkeitserklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Mittweida, den 2. Juni 2019

Patricia Ließ