Universität Bremen
Fachbereich 3: Mathematics and Computer Science
Institute of Computer Graphics and Virtual Reality
Digitale Medien B.Sc.

Bachelor Thesis

Recreating the Valles Marineris -

Modeling and Texturing

Josephine Ortgies
<ortgies@uni-bremen.de>
Matriculation No. 4138129

21st May, 2019

Primary Examiner: Prof. Dr.-Ing. Gabriel Zachmann
Secondary Examiner: Prof. Michael Beetz Ph.D.

Universität Bremen

Nachname		Matrikelnr.	
Vorname/n			

Diese Erklärungen sind in jedes Exemplar der Bachelor- bzw. Masterarbeit mit einzubinden.

Urheberrechtliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.
Alle Stellen, die ich wörtlich oder sinngemäß aus anderen Werken entnommen habe, habe ich unter Angabe der Quellen als solche kenntlich gemacht.

Datum	Unterschrift

Erklärung zur Veröffentlichung von Abschlussarbeiten

Die Abschlussarbeit wird zwei Jahre nach Studienabschluss dem Archiv der Universität Bremen zur dauerhaften Archivierung angeboten.
Archiviert werden:
1) Masterarbeiten mit lokalem oder regionalem Bezug sowie pro Studienfach und Studienjahr 10 % aller Abschlussarbeiten
2) Bachelorarbeiten des jeweils der ersten und letzten Bachelorabschlusses pro Studienfach und Jahr.

○ Ich bin damit einverstanden, dass meine Abschlussarbeit im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.

○ Ich bin damit einverstanden, dass meine Abschlussarbeit nach frühestens 30 Jahren (gem. §7 Abs. 2 BremArchivG) im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.

○ Ich bin nicht damit einverstanden, dass meine Abschlussarbeit im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.

Datum	Unterschrift

Abstract

This thesis addresses the creation of a realistic Mars terrain in Unreal Engine 4, consisting of a landscape, different textures and 3D models for bigger stones and clutter created with a variety of suitable software. It aims to complement the VaMEx-VTB project by providing a useful terrain for testing out robot navigation driven by the visual SLAM algorithm. The algorithm's performance is tested on the finished terrain to determine the influence of prominent landscape features on its accuracy.

Contents

1 Introduction

This chapter serves as an explanatory section to highlight the motivation and goals of this bachelor thesis, as well as introducing the project that this thesis is part of.

1.1 Motivation and Goals

The Technology Center Informatics and Information Technology (TZI) of the University of Bremen is developing a virtual testbed for the Valles Marineris Explorer (VaMEx) in Unreal Engine 4. For this, a realistic Mars terrain is needed because the project explores the behavior and orientation of Mars Robots by means of landmarks.
The goal of this thesis is to create a realistic terrain, including a terrain model, different textures and 3D models used for terrain clutter and different rock types and formations. These components are then set up in Unreal Engine 4 (UE4) to form a landscape in which the virtual Mars robots can roam and interact. Furthermore, some tests are run on the performance of the visual SLAM algorithm approach used in this project, which helps the robots with landmark based orientation, to determine how a more realistic terrain serves to increase the algorithm's performance, opposed to utilizing a blank terrain with no specific landmarks.

1.2 Project VaMEx-VTB

The Project VaMEx-VTB is part of the bigger VaMEx Project run by the DLR (Deutsches Zentrum für Luft - und Raumfahrt). It's a joined project of different German companies and universities which deals with the development of concepts, algorithm and hardware that may be used for the exploration of the Valles Marineris on Mars. A swarm of robots, consisting of drones, mars rovers and quadrupedal robot that can stand up on two legs (*Charlie*), is supposed to be deployed, which communicate with each other and are able to navigate autonomously. Since real hardware testing on-site is very expensive, VaMEx-VTB provides a virtual testbed that makes it possible to test swarm behavior and navigation of mars robots in a cost-efficient way. This way, tests under different environmental circumstances are possible as well. In addition to that, the virtual test bed provides tools to analyze the performance of algorithms and concepts the swarm units are based on to enable optimization at an early stage. The VTB consists of a part of the Valles Marineris landscape with realistic lighting and swarms with different robot types, wich interact individually with the landscape and with each other. It also provides access to simulation data and has the possibility to change environmental parameters to simulate different conditions. The sensory input received and the unit's interaction with the environment can be measured and observed as well[3].

1.3 Structure

In this Bachelor Thesis, the process of terrain creation in UE4 with the help of different material creation - and 3D modeling software is treated. The goal is to compare the performance of VSLAM-based swarm navigation on the resulting map with a less varied map.
To discuss previous work and fundamental knowledge, in chapter 2.1 the VSLAM algorithm, its working conditions and the ORB-SLAM approach used in the VaMEx project are explained.
In the following chapter 2.2, texture maps relevant for material creation in this project are discussed, separated into sections about color maps, normals maps, ambient occlusion maps, roughness maps and height maps. This is followed by outlining the different noise generation functions which have been utilized inside different software used.
Chapter 3 deals with the process of material creation and 3D modeling and illuminates which design decisions have been made. To do so, chapter 3.1 describes the inspirational process and idea generation to set on a final look for the terrain. The process of creating procedurally generated and tileable materials in Substance Designer is explained in chapter 3.2, starting with the *Sand* material in chapter 3.2.1, two organic *Rock* materials in chapter 3.2.2 and 3.2.3 and *RiverBed* material in chapter 3.2.4.
In the following chapter 3.3 the modeling process of clutter and rocks is outlined. This chapter is separated into subchapter 3.3.1, which describes the process of easily generating clutter in Maya, subchapter 3.3.2 which consists of the sculpting of rock models in Mudbox and voxel-sculpting in 3DCoat in subchapter 3.3.3. The mesh decimation of the created high poly model is described chapter 3.3.4. Furthermore, the normal baking in Maya onto the decimated mesh is explained in chapter 3.3.5.
Texturing the model is elevated further in chapter 3.4, starting with the creation of a diffuse - and roughness map by texture painting in Maya (chapter 3.4.1), followed by rendering out ambient occlusion and edge highlights in CrazyBump (chapter 3.4.2) and finishing up the diffuse map in Photoshop, which is illuminated in chapter 3.4.3.
Afterwards, the landscape creation process for the final terrain in UE4 is explained in chapter 4. Chapter 4.1 consists of the landscape generation based on a height map. To be able to apply different materials to the landscape, a master material is needed, whose creation is described in chapter 4.2.

How obvious tiling of seamless materials can be prevented is explained in chapter 4.2.2. This chapter mainly deals with the creation of a material and supporting material functions, the actual texturing of the terrain with the UE4's terrain painting tool is described in chapter 4.3.
The previously created 3D models are placed into the terrain in chapter 4.4 and 4.5. Lastly, chapter 4.6 deals with the distribution of clutter on the terrain, which is done by using UE4's foliage painter.
After finishing up the terrain, tests to determine the performance of the VSLAM algorithm are run, as explained in chapter 5. The finished terrain and results are shown and discussed in chapter 6.
The last chapter 7 evaluates the work that has been done and rates the usability of models and textures for further projects.

2 Fundamentals and Previous Work

2.1 Visual SLAM

Visual SLAM is a visually based Simultaneous Localization And Mapping Algorithm (VSLAM) widely used in the area of mobile robotics utilized in various fields, such as space - or military appliances. These applications require the robots to complete tasks while navigating in unknown environment without manual input by humans [32]. To accomplish this, they need to be able to localize themselves accurate and drift-free in this environment. VSLAM proposes an approach where robots can locate themselves inside an environment without prior information, using visual sensors or cameras, while simultaneously creating a map of their surroundings. This is done by recognizing landscape features and determining whether these features have been seen before from another point of view. The discerning of previously mapped features and therefore re-visited areas makes the algorithm globally stable and and reduces the non-conformance in estimated trajectories and positions [32]. In the following sections, the key concepts of the visual SLAM algorithm are discussed.

2.1.1 Camera Measurement

As a first step, the cameras capture the environment. Using cameras or visual sensors, as it is done in this visual based approach, is cheaper than using laser - or sonar sensors but results can be noisy and require higher computational effort. However, due to the advances in technology, processing the input images in real time is possible by utilizing more complex algorithmic approaches [32].

2.1.2 Feature Extraction

The recorded image is processed and landmarks or features are extracted. Feature extraction methods range from recognizing single points, such as corners, edges and larger areas to even recognizing specific features such as doors [32]. One of the most efficient feature extraction methods is the Features From Accelerated Segment Test (FAST) method proposed by (Rosten and Drummond), which uses edge - and point based tracking. This method places a circle of 16 pixel around each target pixel and compares the brightness values of the surrounding pixels to the center pixel. If the surrounding area has a certain percentage of pixel values corresponding with the center pixel, it is determined as either a corner, an edge or a uniform region [18]. Another option to describe features is the BRIEF descriptor proposed by (Calonder et al), but it can only be used in conjunction with a feature detector and classifies images by setting key-points with the help of intensity comparisons.

2.1.3 Data Association

The elevated data needs to be processed regarding its association with previously elevated data of features. This is crucial for building a reliable map of the surroundings, false data association can lead to inconsistent maps [11]. A way to determine whether a feature has previously been recorded is the Normalized Cross Correlation (NCC). This compares two images by adding up the product of intensity of corresponding pixels and normalizing the sum. The score values can reach from 1 to -1, the higher the value, the greater the resemblance of the processed images [32].

2.1.4 Refinement

Due to cameras being more prone to creating noisy images, feature matching and data association can create faulty results. These can be mostly eliminated by using the Random Sampling Consensus (RANSAC) originally proposed by (Fischer and Bolles), which estimates the transformation happening between two pictures by checking for a random number of features, applying the estimated transformation to all other points of the image and computing the distance between the corresponding points. If those pixels have a distance smaller than a pre-defined threshold, they are considered *inliers*. These steps are repeated several times on the same image, the more iterations, the more accurate of a result it produces. The more *inliers* a result contains, the more likely it is to be the correct transformation [32].

2.1.5 Loop Closure

Loop Closure detection is dedicated to the problem of recognizing pre-visited locations in non-follow up frames. When a robot recognizes it has visited a certain area before, it reduces inaccuracy in map creation. This is most commonly done be selecting only certain keyframes of different areas and comparing the current image to those keyframes [29]. The higher the number of keyframes that have to be compared, the higher the computational effort [14].

2.1.6 ORB-SLAM

The SLAM framework used for the in the VaMEx project is based on the ORB-SLAM2 library by (Mur-Artal et al) and it uses ORB feature detector and descriptor, which combines elements of the FAST feature detector and the BRIEF feature descriptor. ORB-SLAM is capable of operating in large and small indoor - and outdoor environment in real-time and is resistant to motion clutter. The features in use are similar to other SLAM approaches, ORB-SLAM uses feature-tracking, mapping, re-localization, and loop closing. To select keyframes used for re-localization a „survival of the fittest“ method is proposed, which inserts keyframes as quickly as possible and removes redundant ones later on [14].

The Collaborative ORB-SLAM2 Framework in use is capable of running on several different agents at once, creating one common map when overlapping of environment is detected, making it suitable for robot swarm units. The robots' sensors extract environmental features and only relevant features are send to a central processing unit, which runs the SLAM algorithm and merges overlapping maps, making it possible for the robots to localize themselves on one shared map while contributing to said map [29].

2.2. Texturing

Textures are 2D images that are mapped onto a 3D object to provide a more interesting surface appearance. The three main approaches for texturing 3D objects are painting textures by hand, processing scans or photographs and procedural texturing. In procedural texturing, an algorithm is used to generate textures [7], which improves productivity in the texture creation process and reusability of textures by allowing dynamic adjustment of texture resolution and altering textures by simply changing parameters, as opposed to having to re-draw the whole texture, as (Deguy) states. Furthermore, complex textures can be created by overlaying different noises or patterns while still maintaining small file sizes, unlike the limited creative freedom coming with simply using scans or photographs and large resolution image files.

Different texture maps are used together to create materials for 3D objects or landscapes to make them appear in a desired style. There are a variety of texture maps used and created in the context of this thesis to put together a realistic Mars environment, which are elaborated further in the following sections.

2.2.1 Color Map

The most commonly used color map is the diffuse map, which simply displays the color of an object’s surface. This map determines the final color of the object without the influence of other maps such as the normal map and can include ambient occlusion information as well [*Fig i*].

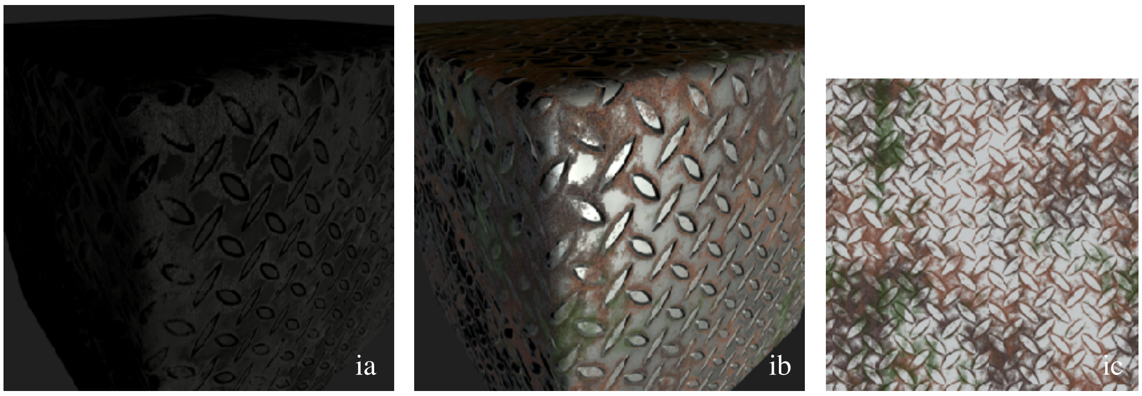

Fig i: *Color maps in texturing: object without color map[ia], object with color map applied[ib], 2D color map[ic]*

2.2.2 Normal Map

When using multiple instances of assets in a 3D environment, it is crucial to minimize the poly-count of objects. A sculpted 3D model can easily have hundreds of thousands of polygons, which makes it very costly to compute at runtime. Utilizing normal maps for texturing creates an illusion of surface detail by providing information how the scene lighting needs to be applied to a mesh, thus allowing the usage of low polygon (low poly) models while keeping a realistic look to them, which is illustrated in [*Fig ii*]. Normal maps store the angle and slope of each pixel and display these normals with red, green and blue values, red storing the x-axis, pointing the normals left or right, green storing the y axis, pointing the normals up or down and blue storing the z-axis, pointing the normals outwards or inwards in respect to the surface. They can be baked from a high poly model onto a low poly model, in this process the normal map stores the high poly model’s surface slope and maps it to the respective pixel on the low poly model, which creates the effect of more detailed surfaces without changing the actual polygons of the model. Photographs or other bitmaps can be converted into normal maps as well. The color (128, 128, 255) defines perfectly flat surfaces on a normal map and can also be referred to as the *normal base color* [13].

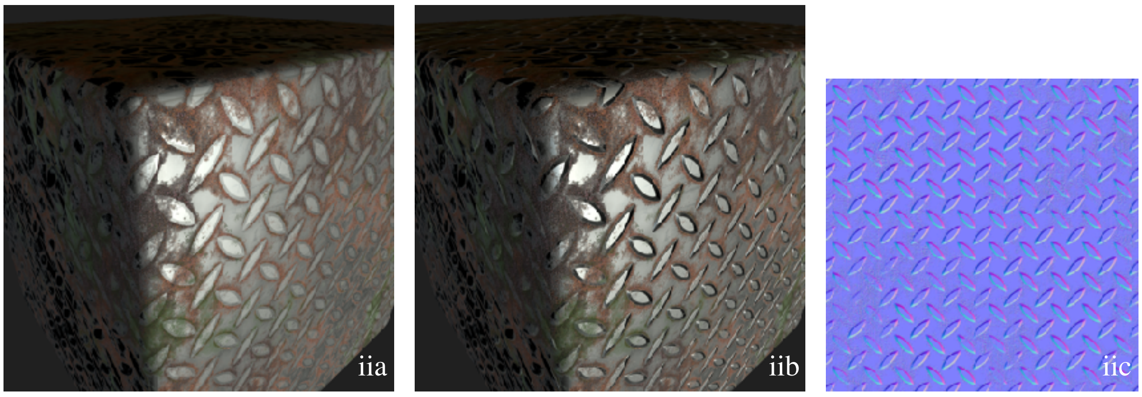

Fig ii: *Normal maps in texturing: object without normal map[iia], object with normal map applied[iib], 2D normal map[iic]*

2.2.3 Roughness Map

A roughness map determines how light is scattered and refracted off a textured surface, enhancing variety in a material. It defines the roughness of an object and where its surface is smooth or matte [*Fig iii*]. Roughness maps contain greyscale values, ranging from black (0), which is going to be displayed as smooth, to white (1), which is computed as a totally rough (or matte) surface [25].

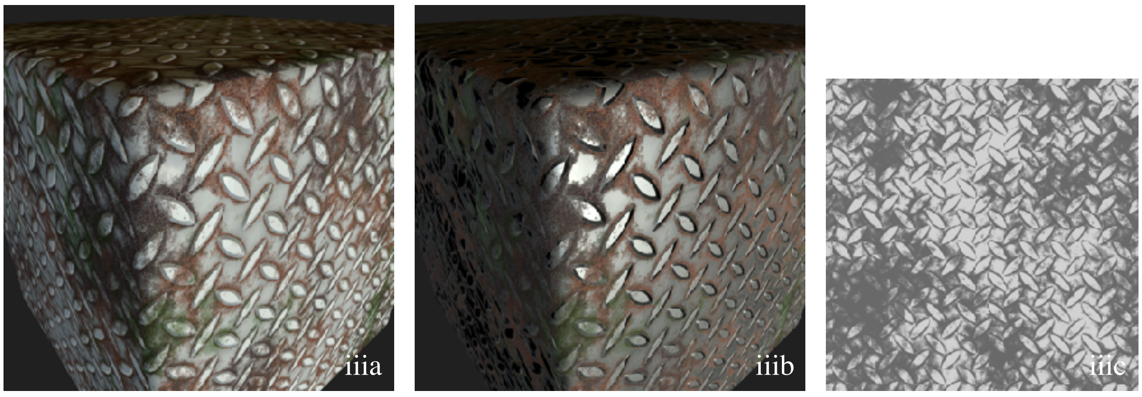

Fig iii: *Roughness maps in texturing: object without roughness map[iiia], object with roughness map applied[iiib], 2D roughness map[iiic]*

2.2.4 Ambient Occlusion Map

Ambient occlusion maps can be used to simulate pre-computed lighting conditions, specifying ambient light reflecting off the surface of an object based on visible structures occluding the surface [1]. It makes occluded areas appear darker, simulating shadow and light and therefore can make a texture look more detailed by providing perceptual clues of curvature and depth [*Fig* *iv*]. In Substance Designer, the Horizon-Based Ambient Occlusion (HBAO) algorithm proposed by (Bavoil et al) is used to compute ambient occlusion maps [19]. This computes ambient occlusion by using the horizon angle and the tangent angle of a height field based on eye-space normals.

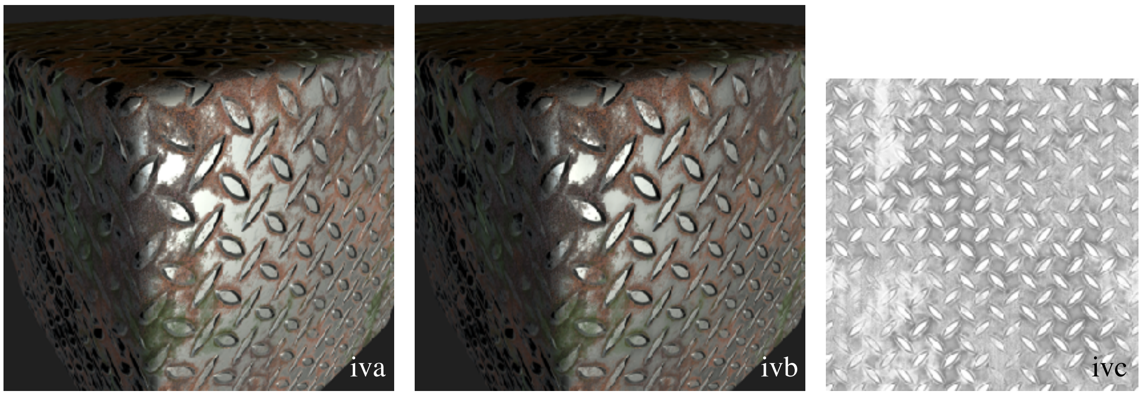

Fig iv: *Ambient occlusion maps in texturing: object without ambient occlusion map[iva], object with ambient occlusion map applied[ivb], 2D ambient occlusion map[ivc]*

2.2.5 Height Map

To determine the topography of an object or landscape, height maps are utilized. These are greyscale maps, the lighter pixels defining higher points of the surface and darker pixels the lower points [*Fig v*] [7]. Usually, a 50% grey value represents a perfectly flat surface. These can be used for tessellation-induced altering of surfaces to make models look more realistic. In UE4, height maps of landscapes in the format of 16-bit, grayscale PNG files [24] can be imported to quickly create a detailed terrain.

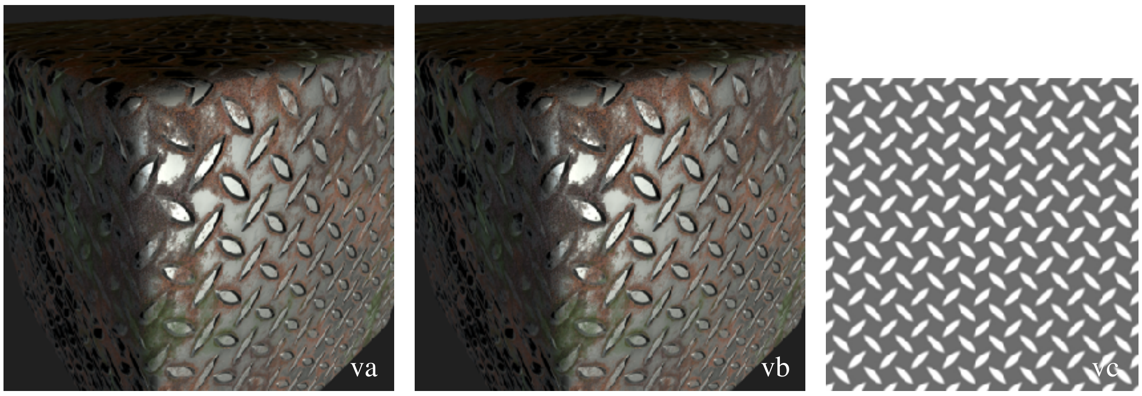

Fig v: *Height maps in texturing: object without height map[va], object with height map applied[vb], 2D height map[vc]*

2.3 Noise Generators

For creating diverse materials, Substance Designer, which was used for creating landscape - and object materials, offers a variety of bulid-in noise generators. Noise is widely used in procedural generation, it can create natural looking patterns fast, as opposed to drawing texture patterns by hand. Noise functions take in an n-dimensional point in space and assign a pseudo-random value, pseudo-random meaning that the same input always produces the same output [12]. The most commonly used noise generators in this project, Perlin Noise, White Noise and cellular noises are discussed in the following sections.

2.3.1 White Noise

White Noise is the most basic noise function, a uniform distribution of random numbers which can be generated in a random physical process, it has the same amplitude in all frequencies. Because of that, the sampling rate can never be high enough to capture all detail in White Noise. It can be produced by utilizing a pseudo-random number generator, but is hard to control because its visualization has a different outcome each time. For this reason, it is sparsely used in procedural texture creation [7]. An example of White Noise can be seen in [*Fig vi*].

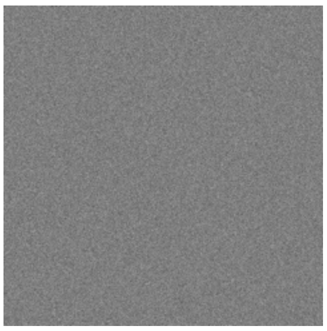

Fig vi: *An example of White Noise (generated in Substance Designer)*

2.3.2 Perlin Noise

Perlin Noise was originally developed by Ken Perlin for textures of the Disney film ‚Tron' to make organic surfaces look less plastic by distorting them. According to (Hyttinen), it is cheap to compute and can be used to create a variety of natural looking patterns, making it the primary tool used in procedural texturing [30]. Perlin Noise is a Gradient Noise function, which uses a lattice with generated integer values and interpolates between those. As opposed to Value Noise functions, which use raw randomized integers, Perlin Noise computes a random gradient vector for each point, resulting in less artifacts. The noise is computed by calculating neighboring lattice coordinates and gradients for each input point [8]. In Substance Designer, Perlin Noise is generated with softer transitions and can be scaled from 1 to 256 by the user, as well as distorted to create small variations [*Fig vii*] [21].

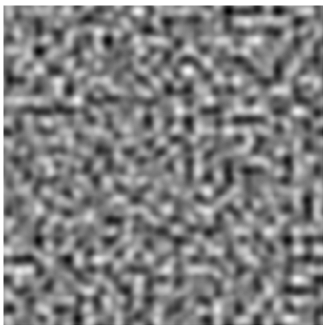
Fig vii: *An example of Perlin Noise (generated in Substance Designer)*

2.3.3 Cellular Noise

Cellular noise generators in the form of *crystals* [*Fig viiib*] and *cells* [*Fig viiia*] are used in the material creation process of this project. These are a common choice when it comes to texturing rocks and other organic material because they resemble *Voronoi* patterns [*Fig viiic*] which are often found in nature. The most commonly used cellular approach is presented by (Worley) and computes round regions randomly distributed in space with soft transitions, resembling cells, by scattering so-called *feature points* through space and computing a scalar function based on this distribution. Poisson distribution is used for scattering the points, which provides points with their location independent of surrounding neighbors. For every point, the distance to the *nth*-closest feature point is computed to generate the noise value, in most cases, the closest feature point is taken into consideration [7].

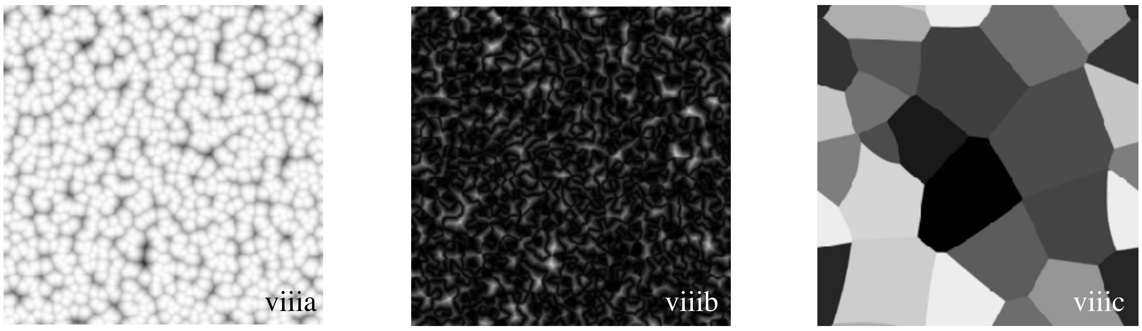

Fig viii: *Different examples of Cellular Noises: cells[viiia], crystal[viiib], Voronoi pattern[viiic] (generated in Substance Designer)*

3 Modeling and Texturing

3.1 Inspiration

To get a good feeling for the design of the terrain, some pictures from a variety of websites, as well as a height - and color map [*Fig 1.1*] of the targeted part of the Valles Marineries are considered, which can be found on the HiRISE (High Resolution Imaging Science Experiment) website of the University of Arizona. HiRISE is the most powerful camera that has been send to orbit another planet to date. In this case it is located on board the Mars Reconnaissance Orbiter and delivers pictures with an accuracy of up to 30cm per pixel [23]. This source's colormaps can only be used as a reference to a limited amount, the color range contains more shades of grey and therefore does not look ‚martian' enough to be used in a demo of Mars terrain. Hence, other pictures are used regarding the terrain's color, especially pictures taken by NASA's mars rover „Curiosity" [*Fig 1.3 and Fig 1.4*]. A good collection of those can be found on geology.com [10] and contains several pictures of different rock formations. Aerial photographs of the Valles Marineris [*Fig 1.2*] and mars terrain by other 3D projects [22] are considered as well.

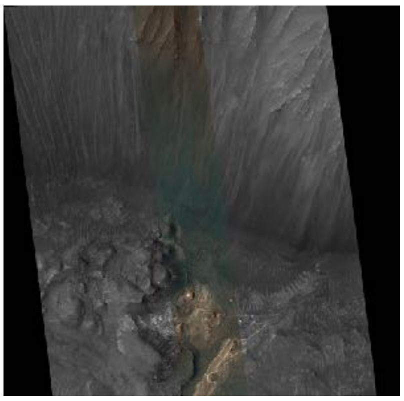
Fig 1.1: *Possible Evaporites Near Fan in Coprates Chasma*

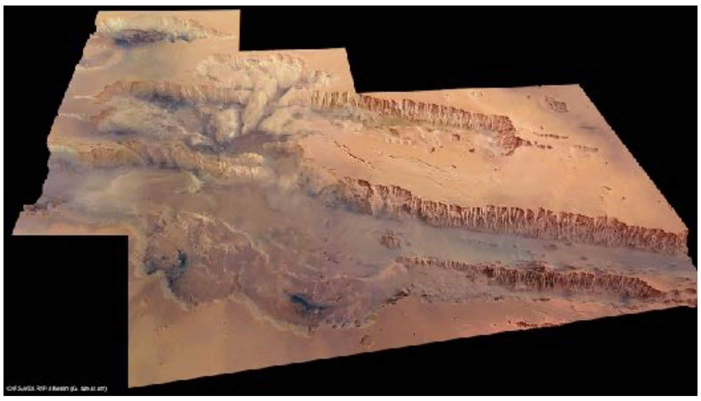
Fig 1.2: *Valles Marineris on Mars*

Fig 1.3: *Curiosity on Mars*

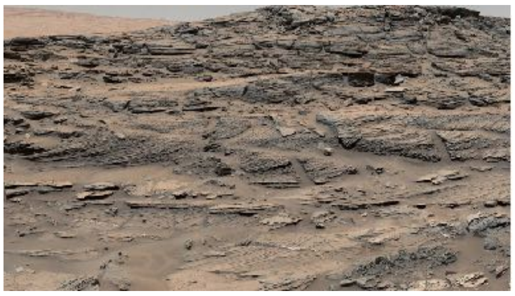
Fig 1.4: *Sandstone on Mars*

3.2 Material Creation in Substance Designer

Substance Designer is a software by Allegorithmic, which offers the opportunity to create procedurally based materials in a node-based environment by using the method of Procedurally Based Rendering. It has build-in noise - and pattern generators, as wells as options to convert outputs into normal maps, ambient occlusion or height maps. This, and the fact that it has plugins available for major 3D modeling software, such as Maya, and game engines, such as UE4 or Unity, makes Substance Designer an industry standard. The creation of the four materials used in the VaMEx-VTB are explained in the following chapters. By exposing parameters of some nodes in Substance Designer, the plugins make it possible to edit these parameters inside a project or change resolution and thus easily creating different looking material instances or fitting one material to the environment it is needed in. In addition to this, because the materials are procedurally generated, the material files are quite small, only a few kilobyte instead of mega - or gigabyte, making Substance Designer the perfect choice for this project. Here, the working resolution for materials is *2048px * 2048px* because this resolution is also used in UE4 later on, creating a high resolution material that is still cheap to compute. Each material needs to include at least one graph which contains all the nodes and outputs, but more graphs can be added to allow better readability of the main material graph. It is practical to create separate graphs when the feature being created is re-used in different materials. All nodes have their own graph, which is also accessible from the outside to make custom changes.
The nodes in use that were not authored by Allegorithmic are clearly marked in the following paragraphs and can be obtained on Allegorithmic's *Substance Share* platform.

3.2.1 *Sand*

As a starting point, the material *Sand* is created, with the main graph ‚sand without dunes', further referred to as ‚main graph', to serve as a solid base for designing a Mars terrain by imitating the typical red-orange sand associated with Mars. This material includes the option to add or remove small pebbles and sand impurities and is explained in detail in this chapter, to serve as an example for procedurally based material creation. In the following chapters, the other materials are explained in their differences compared to the *Sand* material.
First, the pebbles are created: This starts off with two *polygon_2* nodes, which generate a 5-sided and a 6-sided polygon via a build-in pattern generator, which can be seen in [*Fig 2*]. To get the base shape of a pebble, the pixel values of a *cell-pattern,* which was created via a build-in noise generator for cell patterns, are subtracted from the polygons, creating the look of

roughened up edges. Afterwards, the result is run through a *non-directional warp* node (*author: Daniel Thieger*), warping it based on a Perlin Noise. This is now branched out into four different shapes which have their contours sharpened by a *auto-levels* node and are further altered by a *slope blur,* again based on a Perlin Noise.

All this aims to create more variation, and therefore some diverse, natural looking shapes, despite starting out with two simple polygons. This shows how it is possible to create various shapes from a small number of base shapes by utilizing and re-using noises and branching to get different and detailed results. In the following paragraphs, the altering of one polygon is described, the three other ones are done in exactly the same way.

To create a more sloped top, the polygon is blended with a linear gradient. The resulting shape is shown by running the polygon through a *histogram scan* and enhancing the contrast. This shape is multiplied back on top of the polygon to sharpen the edges. In one case, this step is done repeatedly with a rotated and stretched shape to produce more edges on top of the stone. As a last step, some cracks, whose graph is discussed later in this chapter, are added by multiplying them on top to create even more variation.

The created pebbles are distributed on top of the sand base by utilizing a *tile sampler* and masking the input with a distribution mask generated from a *cell-pattern* and a *grunge map*. By adding in a *custom input*, it is possible to add in a custom distribution mask when using the material in UE4 later on.

The pebbles vary in orientation, size and rotation, which can be set inside the *tile sampler*. Smaller sand impurities are created by a different *tile sampler* by only using one of the pebbles shapes and a smaller scale.

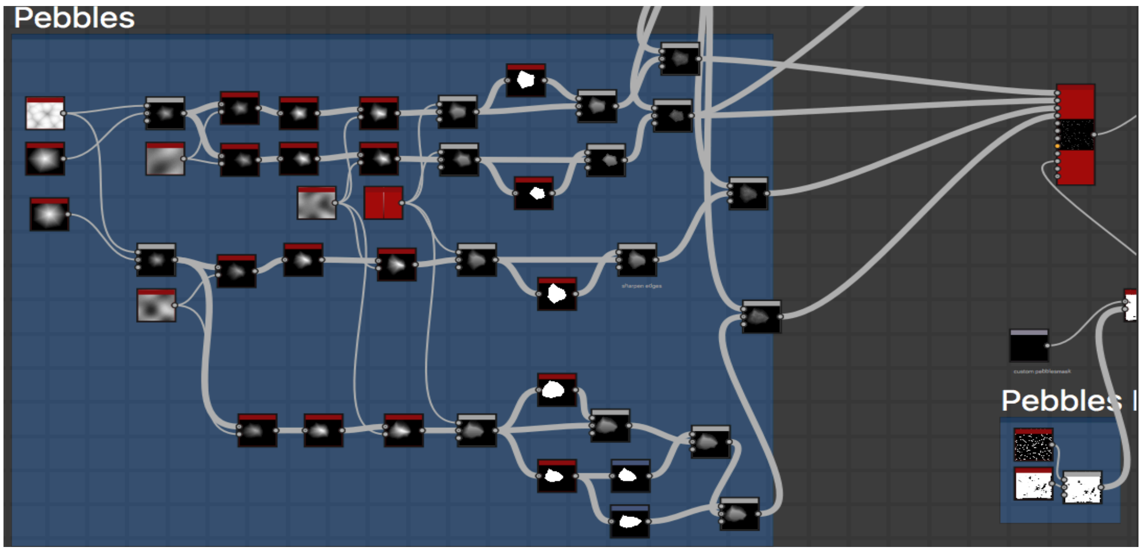

Fig 2: *Creating Pebbles inside the Sand Material in (Substance Designer node graph)*

To roughen up the pebbles' surfaces, cracks are generated in a different graph called ‚Cracks' *[Fig 3]*. This graph's output is not computed when using the material, so its exposed parameters won't be visible from outside Substance Designer.
At first, a disc pattern is generated by a *tile_random* node. By running it through a *histogram scan* and increasing the contrast, the disc pattern is colored in white and can be used as a mask to create a *Voronoi* pattern by running the original disc pattern through a *distance* node. This pattern appears in nature often, making it a nice base for the cracks. To extract the pattern's edges, it is run through an edge detect and these edges are warped based on a Perlin noise to get variation on the otherwise straight edges. The same operations are performed on the original Voronoi pattern, which is then returned as a mask named *pattern*. To finalize the edges, they are blurred and multiplied on top of themselves to make them less hard but prevail a certain degree of sharpness. These are returned as a mask named *cracks* as well. In addition to this, some parameters are exposed, such as the *edge width* of the *edge detect* node and most parameters of the *tile random* node. This serves the purpose that instances of the *cracks* graph inside the main graph can be altered easily to create differently shaped cracks.

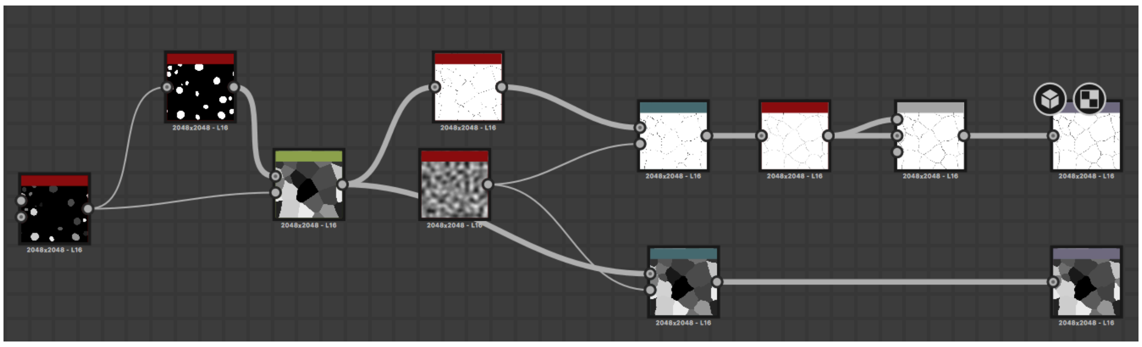

Fig 3: *Cracks graph (Substance Designer node graph)*

Inside the main graph, the *cracks*-output is run through a *slope-blur* node, using a *cell-pattern* as the slope, to create sloping at the edges at various points. The result is blurred to make it look more natural and less sketchy and multiplied with a *Gaussian noise*. To finalize the cracks, their contrast is lowered to make them appear less deep and they are multiplied on top of the previously created pebbles.
The sand is based on a single gray input color and the sand impurities and pebbles are added via a *blend* node by multiplying them on top based on the opacity. This *opacity* parameter is exposed to include the possibility of adding in pebbles and impurities to the users liking. If a single color value is used as an input, it is possible to scale it down to *16px * 16px* to save memory space. This is the smallest size that Substance Designer can work with properly. If this

is done, the input of the following node needs to be set to ‚relative to parent‘ to make sure the same resolution as the rest of the project is kept and prevent resolution issues later on.

The ambient occlusion output [*Fig 8c*] is derived directly from this blend node, using an *hbao* (Horizon-Based Ambient Occlusion) node to compute the ambient occlusion. Everything else is derived from this blend node as well, like the height [*Fig 8e*] - and the normale map. It is generated by a normal node and later blended with the *Sand*'s normal map, because it only contains the pebbles and sand impurities at this point. Next, a sparkling effect for the sand is created *[Fig 4],* as well as the corresponding normal map. This starts off as a *white noise* being branched into two versions, rotated by 90° clockwise and anti-clockwise to create some variation. One of these is run through a histogram scan to reduce the number of black pixels significantly. Since the roughness output displays black as shiny and white as completely rough, the remaining black pixels will determine the sparkling of the sand. By adjusting the exposed *position* parameter of the histogram scan, the sparkle count can be controlled in UE4.

The *Sand*'s normal map is now computed from the white noise's other branch with a *normal map* node. The result ist sharpened and blended with a *normal color* to tone down the effect the normal map will have and is blurred afterwards to reduce its pixelated look. The sand normal is removed from the pebbles' location by blending in another normal color based on a black and white mask of the pebbles' distribution.

In the next step, a normal combine node is used to simply combine the sand normal and the pebble normal. The result is returned as the material's normal map [*Fig 8b*].

To blend the pebbles in with the sand, a *dust* node is used, which generates sand in the pebbles' crevices if pebbles are added to the material and can later be used as a mask when it comes to coloring the material.

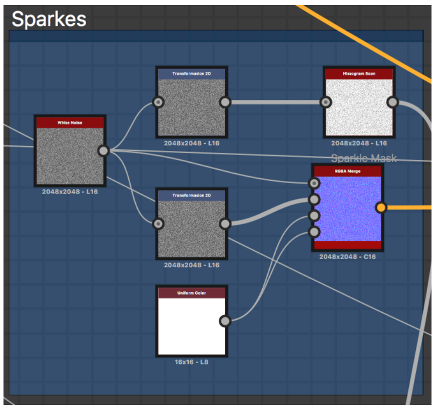

Fig 4: *Creating a sparkling effect (Substance Designer node graph)*

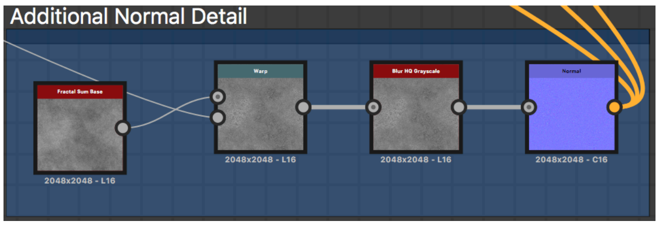

Fig 5: *Creating a additional normal detail (Substance Designer node graph)*

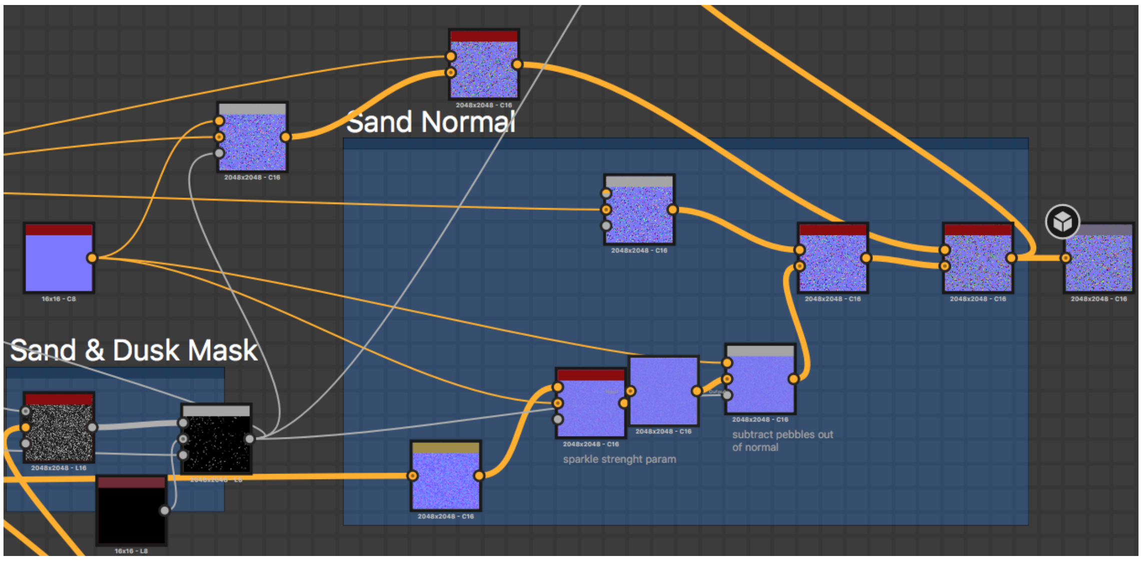

Fig 6: *Computing the Sand's normal (Substance Designer node graph)*

The roughness output [*Fig 8d*] is finalized by adding in the pebbles, based on the pebble mask, as completely white, meaning completely rough. This is returned as the material's roughness.
The coloration of the material represents the last step of creating the *Sand* material and can be seen in [*Fig 7*]. First, a fractal sum is *vector warped* to generate a less monotonous structure with softer transitions. This pattern is taken into a *gradient map* node and a gradient picker is utilized, taking as many values as possible from a close-up image of red sand to color the pattern. Because the color appears to be too bright to be natural, the saturation is lowered slightly by a *levels* node. To give the color map more variation and grouped colors, a *grunge* pattern is colored in by two more gradient maps and by picking gradients of images of actual Mars terrain - one darker and one lighter -, which is overlaid onto the red sand pattern. The base color is blurred to reduce the extreme sharpness and noise and multiplied by itself to keep definition.
The pebbles's color is created in the same manner as described above, the only difference is that the pattern is derived from the dust map, which includes pebbles and some sand. To make the color of the pebbles and sand adjustable and therefore ensure reusability on a broad range, a *hsl* (Hue Saturation Lightness) node is added after the pebble color and the sand color each. By exposing this node's parameters, the hue, saturation and lightness of the texture can be adjusted separately. Unfortunately, Substance Designer doesn't offer the possibility to expose a gradient picker to adjust each gradient separately, inserting a *hsl* node is a usable alternative.
The pebble color is blended on top of the sand color based on the pebble mask. In addition to this, the material's curvature which was derived from the normal map with a *curvature smooth* node, is layered on top of the color and returned as the *base color* [*Fig 8a*].

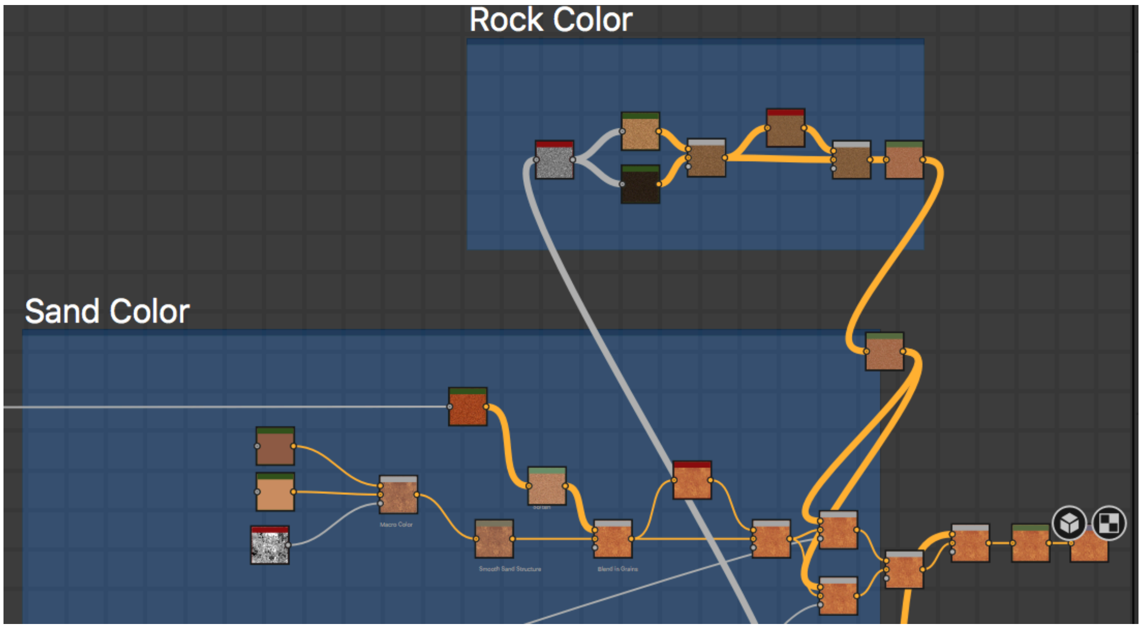

Fig 7: *Sand coloration (Substance Designer node graph)*

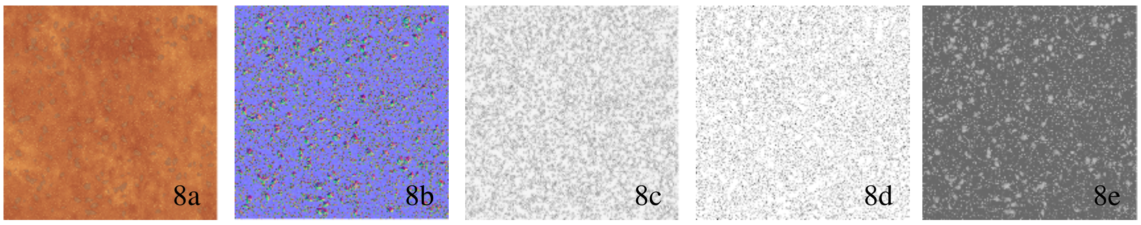

Fig 8: *The Sand material's outputs with active pebbles and sand impurities: base color[8a], normal[8b], ambient occlusion[8c], roughness[8d], height[8e] (Substance Designer outputs)*

3.2.2 *RockM*

Two different materials are created to represent rocks, which are very similar in composition. The one treated in this paragraph is inspired by sandstone [*Fig 1.4*] and consists of a reddish-brown stone with some red sand inside its crevices.

This material is started in a similar fashion to the *Sand* material's pebbles, by generating a 5-sided polygon with a *polygon_2* node and subtracting a *cell-pattern* [*Fig 9*]. The result is altered by two *slope blur*s, one based on a Perlin Noise and one based on a *crystal* pattern (which generates a Worlye Voronoi - type noise [20]) to make the shape resemble a rock, the *slope blur*s produce jagged edges. To angle the rock's top, the result is multiplied by a linear gradient. It is altered further by using a *transformation2D* to stretch the shape and make it appear less round. To add more variation to the edges, the shape is scaled down (because Substance materials are always tileable, downscaling the shape leads to new shapes appearing at the edges of the render window) and subtracted from the original shape. This is multiplied by a *cloud-pattern* to break apart the rock's surface.

The *cell-pattern* that has been used in the beginning is now warped nondirectionally (*author: Daniel Thieger*) to create larger cracks that are multiplied on top of the stone shape. This shape is plugged into a tile sampler to distribute it evenly and cover the entire surface of the rendering window.

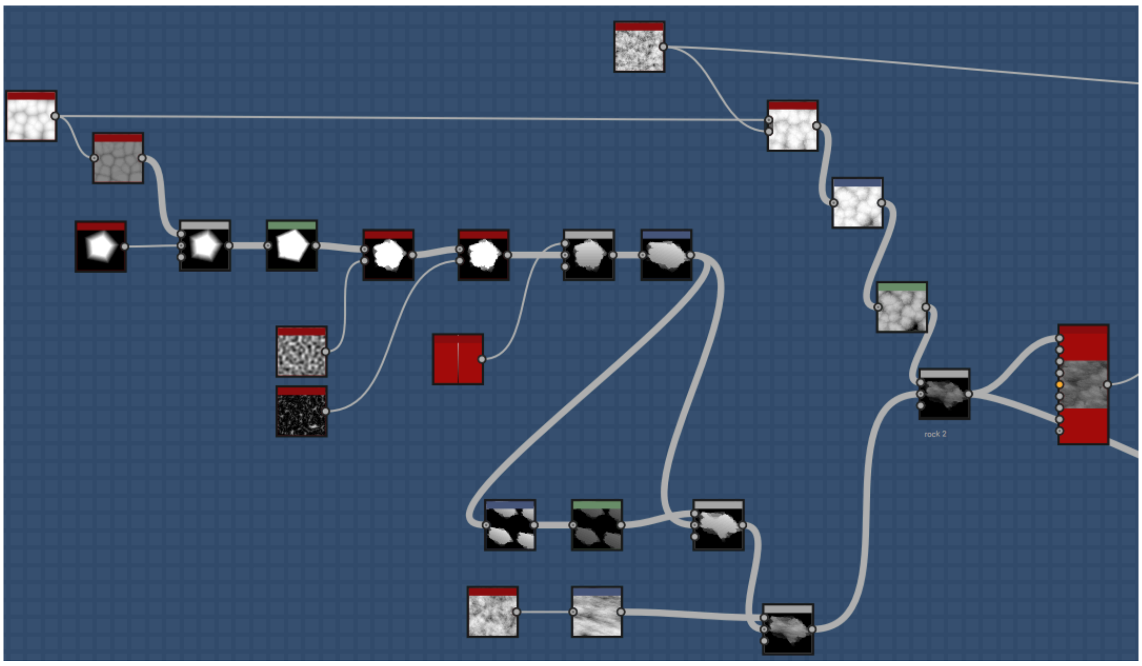

Fig 9: *Creating a basic rock shape for the RockM material (Substance Designer node graph)*

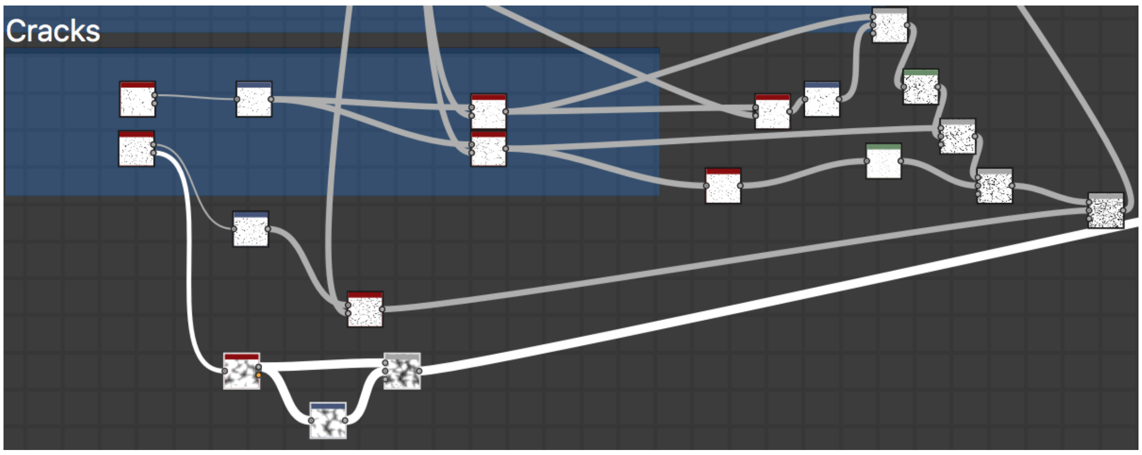

Fig 10: *Adding in cracks to break up the rock's surface (Substance Designer node graph)*

Smaller and more precise cracks are produced by a *cracks* node [*Fig 3*]. Two instances of the cracks graph are included in the material, whereas one is scaled smaller and both are altered by stretching and rotating them [*Fig 10*]. Different slope blurs are applied, using the rock structure as a slope. This ensures the cracks match the underlying rock in their sloping. Several different cracks are branched off of this, varying in scale, intensity and rotation, the variation being achieved by *2D transformations*. These cracks are added on top of each other to generate a more random and diverse looking cracks pattern. This is added on top of the stone pattern by multiplying the two nodes to create an effect of deep erosion. A sand mask, which is responsible for adding sand in the deeper areas of the stone pattern, creating an effect of weathered material, is created. By using a *bevel* node, these sandy areas get a natural fall-off, being higher on the rock's edges and sloping down towards the ground. This is subtracted from the stone shape, deepening these areas.

The ambient occlusion is obtained directly form this, being build up of one strong and one lighter *hbao* node, which are multiplied with each other, creating a strong and deep ambient occlusion output, especially around the rock's edges [*Fig 13c*].

To generate the roughness output, the stone pattern is fed into a *get slope* node (*author: Quantum Theory Entertainment*) to extract its slopes and the result is run through a *sharpen* node to break up the slopes a bit [*Fig 11*]. This way, the rock can display a more diverse roughness, being completely rough along the crevices and smoother on top, like it would be the case in a natural weathered rock. Since the *get slope* node extracts only black and white values, and black in roughness is computed as glossy, this effect is dampened slightly by multiplying a single gray color value on top. The roughness output [*Fig 13d*] is finalized by multiplying the sparkle mask for the sand in the crevices on top, which is created the in same way as discussed before in the sand material's section (chapter 3.2.1). The distribution of the sparkles is driven by a dust map, which is crated partly from an *hbao* node that takes the stone shape as an input and some additional normal detail, wich is explained in detail in the next section.

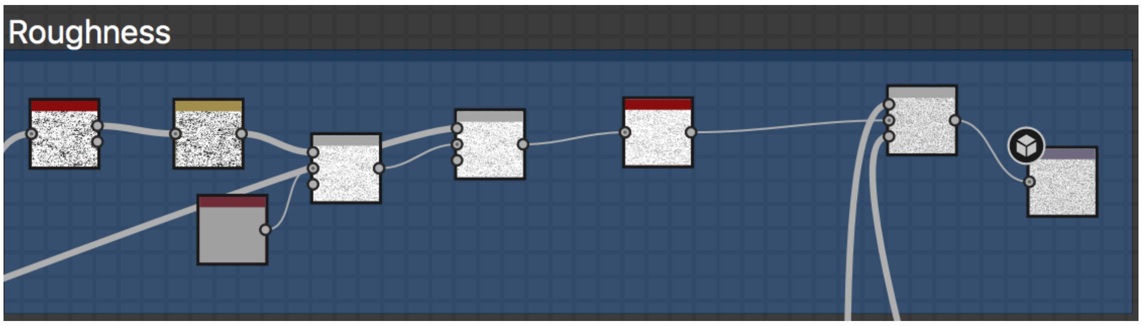

Fig 11: *Computing the rock's roughness output (Substance Designer nodegraph)*

The material's normal map is obtained in a similar way to the S*and* material. By using a normal node, the rock shape is converted to normal colors and is complemented by a *bevel*'s normal map by combining the two normals. This *bevel* node is derived from the *pattern* output of one of the *cracks* nodes, resulting in a more interesting distribution of high and low rock parts throughout the shape.

The previously mentioned additional normal detail [*Fig 5*] is created from a simple *fractal sum,* which is warped based on the stone structure to fit it to its shape, blurred to reduce the pixelated effect and converted into a normal map. This way, some detail normal that aligns nicely with the rock's structure is set up. It is combined with the main normal map and transformed into the material's height output by inserting a *height* node [*Fig 13e*]. The following normal creation resembles the S*and* material's normal creation process, as a first step, the sand's normal map and the previously mentioned sparkle map are generated from a *white noise,* which is optimized in several steps. Driven by the dust mask, the sand's normal information is combined with the stone's normal and returned as the material's normal map [*Fig 13b*]. The curvature needed for correct coloring later on is derived from the normal map as well by using a *curvature smooth* node.

The coloration of this material [*Fig 12*] is nearly identical to the *Sand* material. In addition, the material's slope and ambient occlusion are considered when distributing the sand color and rock gradient maps. More gradient maps are used to set a more diverse color scheme [*Fig 13a*].

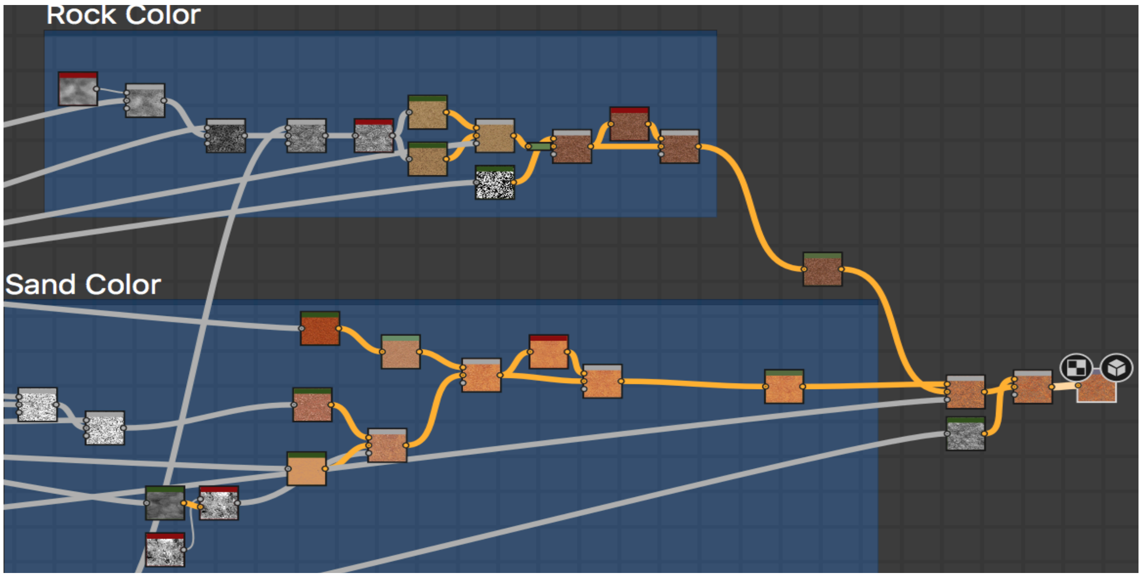

Fig 12: *Creating the rock's base color (Substance Designer nodegraph)*

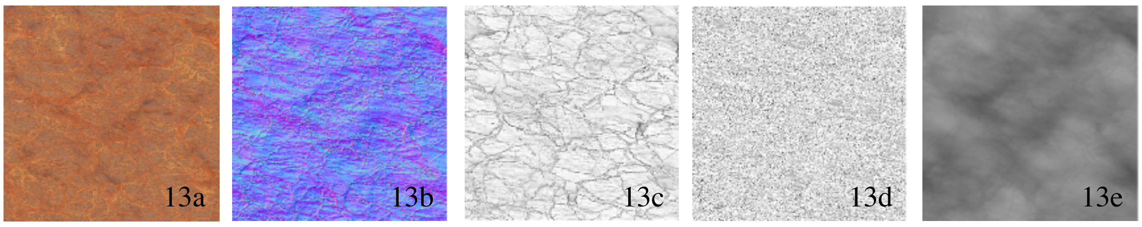

Fig 13: *The RockM material's outputs: base color[13a], normal[13b], ambient occlusion[13c], roughness[13d], height[13e] (Substance Designer outputs)*

3.2.3 *Rock_02*

The *Rock_02* material is supposed to give off a more eroded look than the *RockM* material. It differs in color, deepness of the crevices and has less sand added inside of these.

The main graph varies in this: the normal map derived from the *beveled* cracks node is not used, in fact, the cracks themselves are deeper to create the weathered look. Some parameters inside the tile sampler that produces the stone pattern are changed, such as the *scale* being doubled to set a thicker look of overlaying stone shards. The dust map's usage as a mask is modified in a way that makes it have less white space and therefore allows less sand to be rendered on top of the stone shape. The *Rock_02* material outputs can be seen in [*Fig 14*].

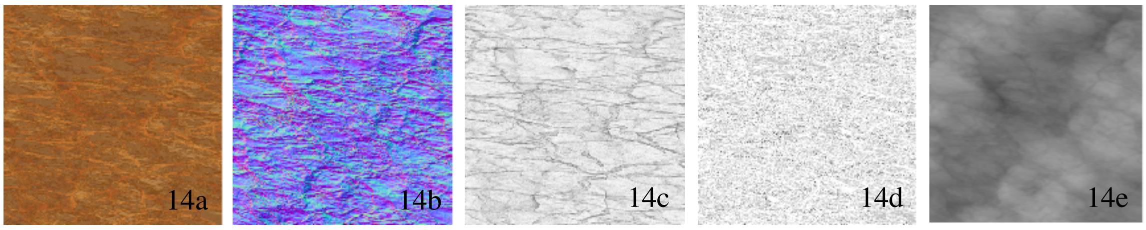

Fig 14: *The Rock_02 material's outputs: base color[14a], normal[14b], ambient occlusion[14c], roughness[14d], height[14e] (Substance Designer outputs)*

3.2.4 *RiverBed*

Lastly, it was decided to include another material largely based on the *Sand* material, which resembles a dried and cracked riverbed.

The cracks that are used to roughen up the pebbles included in the S*and* material are used to crack up the actual sand and some smaller cracks are added in to give it a more weathered look [*Fig 15*]. These cracks, being inverted and strengthened by a *histogram scan*, serve as a mask for distributing the pebbles, to have pebbles mainly along the edges of the cracks.

The cracks are then multiplied on top of a single gray value, which serves as the sand base. To create a sloping effect from the cracks downwards, as it can often be found in real riverbeds, the largest cracks are run through a *bevel* node, which is inverted and blurred to have smoother transitions between slopes. This is multiplied on top of the sand and cracks. The result is treated exactly the same way as the *Sand* material, the sand impurities and pebbles being added and all outputs derived from there. The coloring of pebbles and sand is adjusted slightly by picking gradients of an image [*Fig 1.3*] that shows the Mars Rover Curiosity on this type of material on Mars. When used later on, the parameters that can be altered are the same as inside the *Sand* material.

These materials serve as a base for texturing the 3D Models and landscape in UE4. They are exported as a *.sbsar* file, which can be processed by the Substance Designer plugin in Maya or UE 4. Before exporting, the material's resolution needs to be set to ‚relative to parent' to make sure the resolution can be adjusted later on.

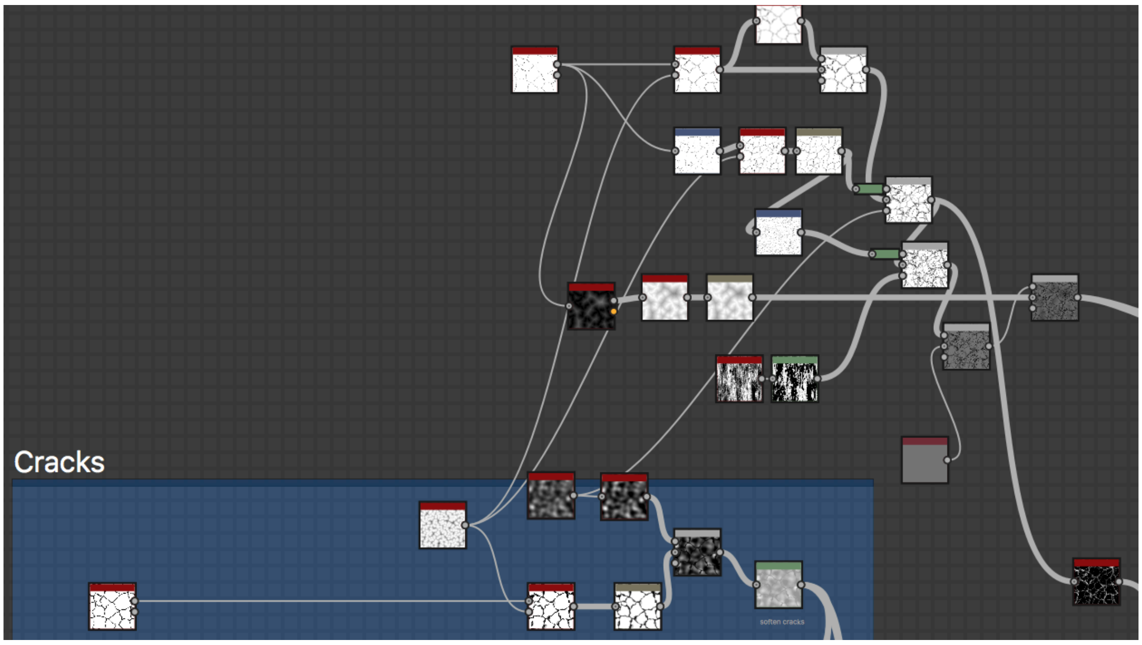

Fig 15: *Variations of cracks inside the RiverBed material*

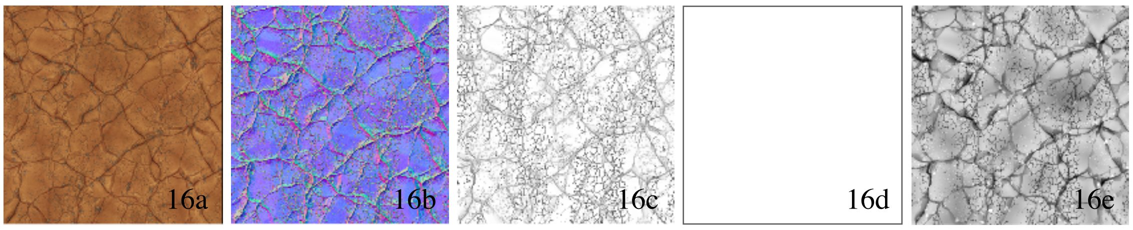

Fig 16: *The RiverBed material's outputs: base color[16a], normal[16b], ambient occlusion[16c], roughness[16d], height[16e]*

3.3 Rock Modelling

In the process of modeling, 4 different detailed rock models are created, two rocks, with one looking more like a meteor and the other more smooth and slope-like, and two different rock formation, one forming an arc and one pointing upwards. These are later on used multiple times with different transformations applied to enrich the Mars terrain. The modeling process is described in detail in this chapter.

Different software was used to accomplish this: Maya by Autodesk is used as the main 3D modeling software, but Blender could have been used as well. Since Autodesk's products are more widely used in the creative industry and easier to understand than Blender, and Substance offers a plugin for Maya which makes the material handling more efficient, Maya was chosen instead. For sculpting, Mudbox, which is also an Autodesk product, was used because it offers a wider range of sculpting options and a much simpler interface than Maya. Mudbox scenes can be directly send to Maya and vice versa, which provides a smoother workflow. Texture painting and previewing different layers of texture maps is also possible in Mudbox. Substance offers its own software to do this, Substance Painter, which would possibly work better because Substance materials are utilized anyway, but since Mudbox was already in use, it was used for this purpose as well.
The other 3D software in use is 3DCoat because it allows voxel sculpting, offering more options to create rock formations out of a single sculpted model. For mesh decimation only, MeshLab is utilized, its quadric edge collapse decimation algorithm results in consistent low poly models.
To finalize the texture maps created in Mudbox for each individual model, CrazyBump is used to render out normal and diffuse maps and finally, Photoshop is utilized for color correction and blending corresponding texture map layers.

3.3.1 Clutter in Maya

Maya has a helpful built-in shatter option to create rocks for clutter fast and easy. A sphere is created in Maya and the subdivision level is doubled to have a 40 x 40 subdivided sphere. This sphere is slightly deformed by pulling out vertices in *soft selection* mode to make it appear less round. Now, inside FX mode, the shatter effect can be applied to the sphere. It is possible to set the shard count and edit the edge jaggedness. After applying the effect, the original shape can be deleted and the created shards can be pulled apart. These now can be edited individually to improve weird-looking shapes.
For texturing the clutter, it is sufficient to unwrap the UV coordinates automatically and create a new material using the Substance plugin. The clutter will be scattered throughout the scene and won't be scaled up significantly, so any seams that might be appearing after texturing won't be too noticeable. The shards can be exported individually as *.fbx* files, since the *.fbx* file format stores texture information as well. Before exporting a shard, it is important to snap the model to the center of the grid (0, 0, 0), otherwise it will be off-center when reimporting it into UE4 and this will make transformations harder.

3.3.2 Sculpting in Mudbox

To start sculpting in Mudbox, a basic shape in needed, which has to be created in Maya first. To do so, a polygon cube is added in Maya and scaled up on the z-axis by a factor of 2 to get the basis for a long rock [*Fig 17a*]. Afterwards it's important to make sure the faces of the model are squares to ensure that sculpting works properly and any further subdivisions have equal proportions, so the cube is subdivided 8x in length and 16x in width. This cube is then exported as an *.obj* file because these are readable by most 3D software and imported into Mudbox.
In Mudbox, the mesh is divided up to subdivision level 4 or 5, which is approximately 160000 polygons to ensure smooth sculpting and some parts of the top and bottom are pulled out a bit using the sculpt tool to deform the mesh slightly. The main sculpt brush in use is the scrape brush with one of Mudbox's default stamps (*bw_cliffFace*) used as a stamp image, the stamping distance set to 0 to prevent repeating patterns and with randomize selected. With this brush the mesh's edges are scraped off and simply by varying the brush's size a rock shape is sculpted. To detail the surface even further, the same stamping image is used as an actual stamp and stamped on some parts of the rock mesh until it looks natural [*Fig 17b*]. Afterwards, it is exported as an *.obj* file and imported into 3DCoat for Voxel Sculpting.

3.3.3 Voxel Sculpting in 3D Coat

Voxel sculpting is a very good method if a model only needs surface detail added. This way it's easy to add or remove material, just as if the model was made out of clay. The imported mesh is added to the scene and then the mesh is added again, but this time scaled and rotated differently. Now it is possible to add this to the original mesh multiple times with different scales and rotations to create a more diverse rock in a way that does not make if obvious that the same model is used over and over again [*Fig 17c*]. A ‚fill voids' command is run on the final mesh to ensure that the model won't contain any artifacts on the inside. This is the finished high poly mesh for the rock model which will be used later on to bake normals onto the low poly mesh.

3.3.4 Mesh Decimation

To make the the model usable in UE4 without significant loss of performance it is important to lower the mesh's polygon (poly) count. The high poly mesh has about 250000 polygons after voxel sculpting and because the models are going to be used multiple times in UE4, it is more performant to keep the poly count for decorative objects as low as possible.

To compute the low poly model from the high poly model, a software called MeshLab is used. Theoretically, it’s also possible to reduce the poly count when exporting an object from 3DCoat, but the algorithm 3DCoat uses takes most faces off of one side of the mesh, resulting in a very uneven mesh with one huge flat surface and very sharp edges which is not usable for baking normals onto it later on. The *.obj* file exported from 3DCoat is imported into MeshLab and the Quadric Edge Collapse Decimation algorithm is run. With this filter it is possible to specify the number of target faces, which is 800 for the rock models for the VaMEx-VTB [*Fig 17d*]. By having this very low number of polygons it is also possible to use the models as foliage for the Terrain in UE4 and having tens of thousands of instances without decreasing the frame-rate significantly.

3.3.5 Baking in Maya

When reimporting the mesh into Maya for UV unwrapping and baking, it is possible that non-manifold geometry occurs as a result of voxel sculpting. Here, it often shows in the form of one edge connecting 3 faces instead of 2. These problematic areas are highlighted automatically when importing the mesh, it is best to delete the affected faces and close any occurring holes.

To create the illusion of a higher poly mesh, normal baking in Maya is utilized. First, the low poly mesh’s UVs need to be cut. When cutting and unwrapping a mesh’s UVs, a 2D plane is created from the 3D object, that way it is possible to texture an object without the texture stretching. It is best practice to cut UVs that are in the crevices of the object because seams that might occur at the cuts later while texturing can easily be hidden with ambient occlusion. Because of the previously done voxel sculpting, there are a lot of crevices where two meshes were added together, it’s best to cut the mesh there. After the cutting is done, the mesh can be unwrapped in the UV editor. Maya has a special shader to show in which areas the texture might be stretched, after turning it on the affected areas can be cut and unwrapped again. Another option is to move vertices manually to achieve the same effect. After reorganizing the UVs, the high poly model which was made in 3DCoat and Mudbox can be baked onto the UVed low poly model.

To do so, the high poly mesh is imported and placed on top of the low poly mesh and the ‚transfer maps‘ action in the rendering menu is selected. Inside the ‚transfer maps‘ window, the low poly mesh can be set as the target mesh and the high poly mesh as the source mesh, meaning the high poly meshes normals are baked onto the low poly mesh. To make sure the normals are baked correctly, the envelope size needs to be adjusted to determine the area where Maya shoots rays inwards to apply the source meshes normals to the target mesh, in the best case enveloping the target mesh completely so no spots are missing. Afterwards, the normal

map is selected as an output map and the output file is set to be a *.tga* file. The *targa* (Truevision Advanced Raster Graphics Array) file format is used for all graphics in this project because it works well with most 3D software and can utilize lossless compression. It is a standard for working with textures, that is the reason it is used, but *.png* would be a usable format as well. The other settings are left at default settings, the only thing that needs to be specified is the output resolution, which is set to 2048 x 2048 pixel, because this is the standard resolution for all image data used while creating the terrain. Lastly, the output quality can be specified, which is best set to ‚high' and the normal map can be computed. This is saved in the specified location and automatically applied as a material to the low poly mesh in Maya [*Fig 17e*].
Afterwards, the UVed low poly model is exported as an *.obj* file once more.
Crazy Bump is a software that makes it possible to render out different types of maps from normal maps or photographs with variable settings, which makes it a good choice for aiding the texturing process. In this case, the ambient occlusion and a diffuse map are rendered out from the baked normal map. This diffuse map serves to purpose of providing edge accentuation to make the edges in the final color map look more highlighted and will therefore be referred to as ‚edge highlight map' to avoid confusion with the base color diffuse map.

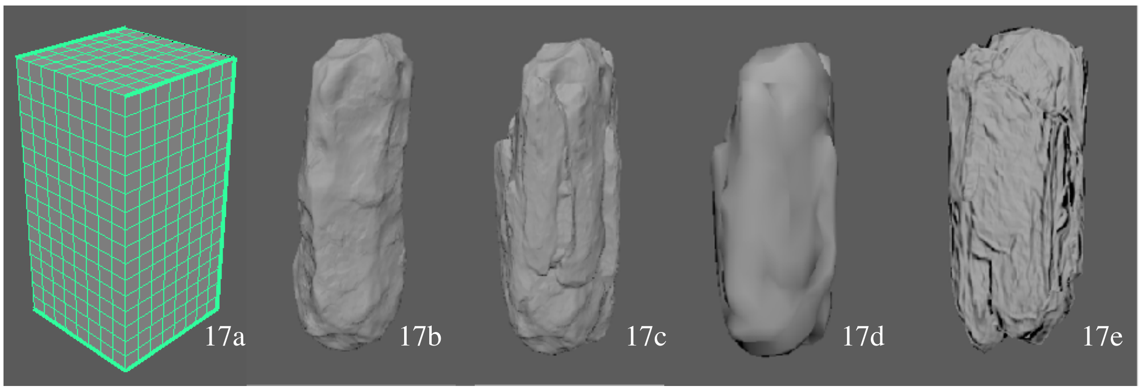

Fig 17: *The Process of modeling Rock_01: base shape[17a], sculpting[17b], voxel sculpting[17c], mesh decimation[17d], normal baking[17e]*

3.4 Texturing the Model

3.4.1 Texture Painting in Mudbox

The UVed low poly model is imported into Mudbox to paint on a texture that has previously been created in Substance Designer. The two textures used for the models are *RockM* and

Rock_02 [*Fig 18*]. Since Substance Designer provides a plugin for Maya, it would also be possible to directly drag-and-drop a substance texture onto a 3D model in Maya, however, the models don't have a uniform direction in tangent space, which makes very strong seams visible and zigzag patterns at some smaller UV parts appear when using a tileable material. By painting on the texture in Mudbox and rendering out the specific texture maps in CrazyBump, it is easier to control the outcome of the final texture map and refine small details. Because the *.obj* file holds the mesh's UV information, painting on a texture will return a texture map to use as the base color that fits the model and the previously created maps perfectly.

First, the previously created ambient occlusion and edge highlight maps are imported as paint layers (diffuse) and the baked out normal map is imported as a normal paint layer. The blend mode of the two diffuse layers is set to multiply, this serves as a nice preview what the low poly model without any further texturing would look like. It's also helpful to adjust the opacity of the layers, making the edge highlight map less opaque to make sure it won't tamper with the final color of the object. The base color of the substance material is imported as a diffuse paint layer as well and set as a stamping image, as well as the material's roughness map. Now the model is simply painted with the stamping image to get a base texture, alternating between using the base color and roughness over each other to create a corresponding roughness map [*Fig 18c*] for the base color map. Because the texture is stretched where the faces of the model don't face the camera, the same stamping image is used to refine the painted on texture and to paint over any seams or unnatural looking transitions. Afterwards, the painted diffuse map and the roughness map are exported as *.tga* images.

3.4.2 Rendering out Texture Maps

CrazyBump is used to render out the detail normal and combine the model's two normal maps. For this, CrazyBump uses calculations in 3D space, preserving a lot of detail when combining normal maps. First, the base color map is imported into CrazyBump, its normal information is displayed, sharpened and only the medium detail is rendered. This now serves as the detail normal map of the rock, which is basically the normal map of the texture. It is combined with the model's second normal map, which displays the high poly model's normals [*Fig 18b*]. The finished normal map is exported as a *.tga*.

3.4.3 Finalizing the Texture in Photoshop

The finalizing of the color map is done in Photoshop because it offers an easy way to combine layers and color correct these. For this purpose, the ambient occlusion map, the edge highlight

map and the diffuse color map are imported as separate layers into a Photoshop project. The diffuse map being the top layer and the ambient occlusion the bottom layer, the blending mode of the layers is set to multiply and the opacity of the edge highlight map is lowered to the same value that looked natural when previewing the maps on the model in Mudbox. Because the edge highlight map and the ambient occlusion map might darken the texture slightly, it is possible to adjust lightness and other parameters, taking the original substance material's base color as a reference, to make sure the textured model will blend well with the textured environment in UE4. The final result is exported as a *.tga* and serves as the model's color map [*Fig 18a*].
The final look of the model can always be previewed in Mudbox by simply importing the color map and the normal map.

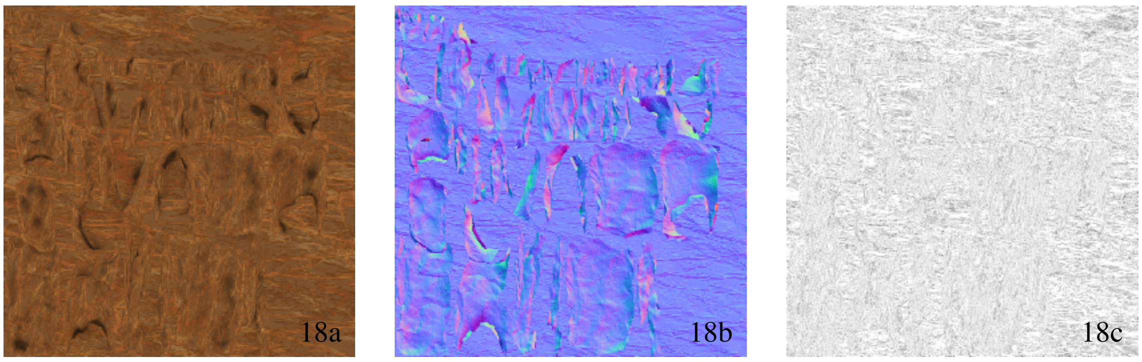

Fig 18: *The three final texture maps for the Rock_02 material for the Rock_01 model: base color, including edge highlight and ambient occlusion[18a], normal map[18b], roughness map[18c]*

4 Styling the Terrain in UE4

In this chapter, the assembling of the final terrain in Unreal Engine 4 is outlined, starting with creating the base landscape and then painting it and distributing the 3D models.

4.1 Generating the Landscape

Unreal Engine 4 has a build-in landscape generator that can take a custom height map as an input. The *new landscape* tool is used to generate a small part of the Valles Marineris. The height map in use was provided by HiRise [*Fig 1.1*] and depicts the site that was tagged as „Possible Evaporites Near Fan in Coprates Chasma". To make sure the landscape is displayed correctly, the scale needs to be set before importing. The *x* and *y* scale need to be set to the resolution of the height map in *m/pixel* and the *z* scale to (*maximum hight - minimum height*) divided by 512. In UE4, one unit equals 1cm, so the calculated results need to be multiplied by 100 to get the correct measurements. In this case, the maximum height is 3213.117m and the minimum height is 181.644m, meaning the import scale of the height map needs to be set to 25 x 25 x 597, because the initial scale of the height map was *2000px * 2000px*. The landscape is imported into the existing project as a new level to prevent interfering with the development of the virtual testbed itself, it is later used as the main map in the project.

4.1.1 Terrain Sculpting

UE4 offers several tools for altering terrain, such as landscape sculpting, landscape painting or painting on foliage. These three options are used for the creation of the Mars terrain, landscape painting being explained in chapter 4.3 and the addition of foliage in chapter 4.6. First, the landscape sculpting option is used to smooth out some surface details that display artifacts from the height map. When looking at the digital terrain model on HiRISE, it becomes apparent that the rising side of the terrain consists of landslides coming down. Because these are not visible anymore on this *2000m * 2000m* segment, they are sculpted into the terrain manually by using the *smooth* tool. Some slopes are added at the bottom of the small mountain range as well to improve the transition between mountains and sandy areas by flattening the terrain and eroding the edges. These also provide a more diverse and challenging virtual environment for the rovers to be tested in.

4.2 Setting up a Master Material

Applying several different materials to a landscape via landscape painting is made possible by setting up a master material consisting of several different material layers. This is applied to the landscape and the layers can be painted on top of each other later on.

4.2.1 Including different Textures

To import Substance materials into Unreal Engine 4, the Substance Plugin needs to be installed inside the engine. It is a free plugin which can be found in the UE4 marketplace and enables resolution control and the possibility to import different maps, such as the material's height map which is not imported by default, as well as an interface for altering exposed parameters. When importing a material, all maps that the material outputs, are imported into the targeted assets folder. A so-called ‚substance graph instance' serves as the interface for altering parameters and the ‚substance factory' gives the user the option to re-import a complete material or delete all instances of it.
After the different materials have been imported and their corresponding height maps have been imported via the substance graph instance, the resolution is set to *2048px * 2048px* and a master material needs to be set up that includes all materials. In this case *Sand* serves as a master material, simply because red sand is the base look for this Mars terrain, but it would be possible to choose any other material or an empty one. Two other instances of the *Sand* material are created, one includes pebbles and sand impurities and the other differs in color, being more of a brown sand. The *Sand*'s base color, normal map, ambient occlusion map and roughness map are already hooked up correctly by the plugin, now the *RockM*'s, *Rock_02*'s, *RiverBed* and the two other *Sand*'s instances' maps can be dragged into the node graph or selected as texture samples. To be able to blend and paint different textures on top of each other, *layer blend* nodes are utilized, one for each map type [*Fig 19*]. Inside the *layer blend,* the new layers can be added and named after the targeted material, in this case there are 6 different layers. The blend type is set to *height blend,* which creates another input for the material's height map and the preview value needs to be greater than 0 to ensure the layer is displayed when painted on. In this case, a value of 0.5 is used for each layer. As a next step, the different maps simply need to be plugged into the layer inputs and the *layer blend*'s output is plugged into the corresponding output of the material. The material itself only has 16 texture samplers and this limit is reached pretty quickly when utilizing 5 different texture maps per incorporated material. To prevent faulty rendering, the sampler source, which controls where each texture lookup comes from, of each texture

sample is set to *shared wrap,* which means that the sampler source does not occupy a single sampler slot, but uses a global sampler. By setting it up this way, one master material can deal with up to 128 texture samples [27].

The textures are tiled very small at this stage, the user would not be able to identify the specific details on one texture tile. To scale them up to a realistic size, a *landscape coordinate* node is plugged into the maps' UV inputs and the mapping scale is altered, it is increased to 400 for the rock textures and 50 for the river bed texture, this value is determined by experimenting until a good looking, not too blurry texture scale is found. A disadvantage of this procedure is the textures becoming blurry when scaled up too high. This is especially problematic with the *Sand* texture's grains becoming larger and visible. In the next chapter, a way to prevent this is discussed.

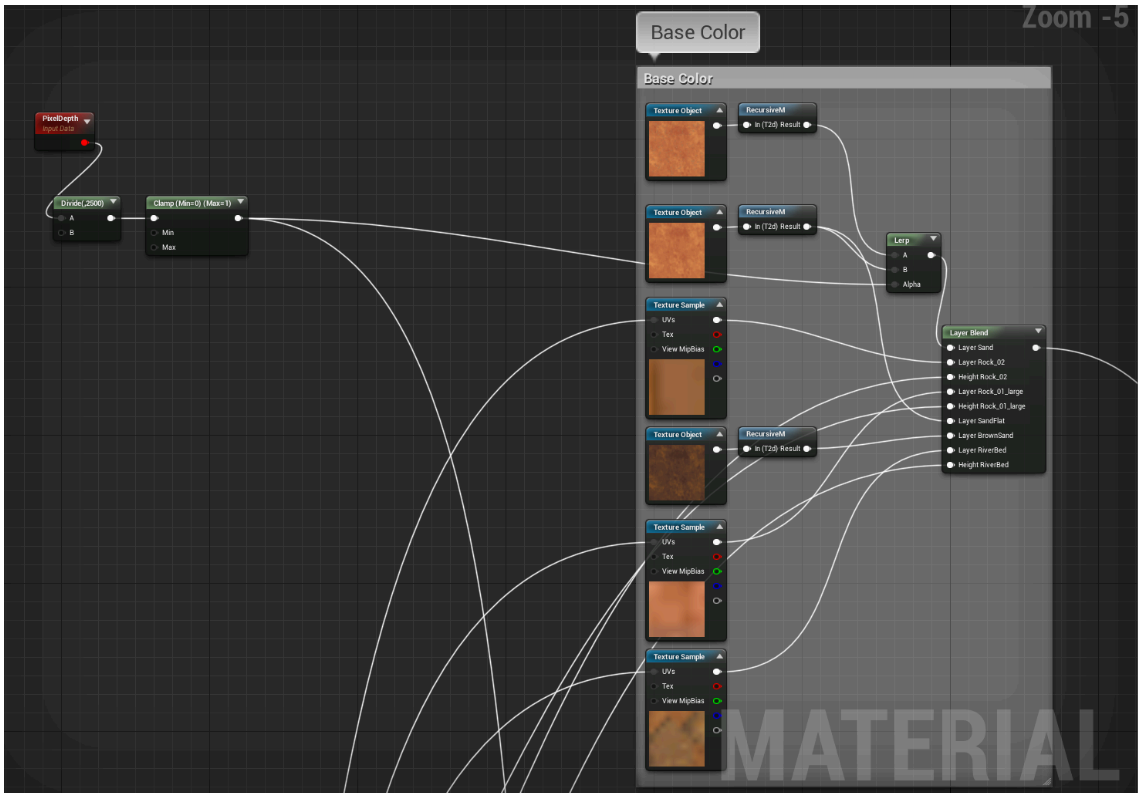

Fig 19: *Setting up the layers for the master material's base color output (Unreal Engine 4 material)*

4.2.2 Obscuring visible Tiling

The terrain needs to look good from different angles and distances without displaying obvious repeated patterns. When texturing a terrain for a game, usually only one viewing distance is going to be used, in most modern games it is the camera distance of the player. This virtual test bed needs to look good from the perspective of the robots but also from above when used for presentations or demonstrations. One way to achieve this would be scaling up the landscape coordinates so the texture does not repeat itself visibly when looking at the terrain from above. Although the texture has a high resolution, this leads to a stretched and pixelated look when viewing the texture up close.

Another way of achieving a good look from all angles is utilizing texture blending, as proposed in a tutorial by YouTube channel Worstplayer. Here, different scales of the same texture are used depending on the camera distance to the plane and being blended with each other when altering the camera distance to achieve a smooth transition. This is done with a material function that gets a 2D Texture for an input. The function can later be added as a node after the desired texture map into the master material to create a multi-level texture. Empty texture samples define the layers that are blended on top of each other, the input texture is set a the texture sample's texture. These get *texture coordinates* as the UV input, but the *texture coordinates* are divided to scale up the texture by e.g. 10 for the second smallest scale, going up the 2000 for the overall largest scale that spans across the whole landscape when viewed from the top. The layers are then crossfaded using linear interpolation based on the distance of pixels to the camera in cm, the layer of lager scale is used when the camera moved farther away from the landscape. Because the pixel depth is in UE4 units, it needs to be divided to increase the blending distance. The first two layers are blended when the camera is about 2m away, so the pixel depth is divided by 200. The result needs to be clamped between 0 and 1 to prevent the alpha value for linear interpolation to exceed 1. This can be repeated with as many layers as possible to get nice transition between the differently sized textures, in this case the material function contains 9 layers, each doubling in scale compared to the previous one [*Fig 20*].

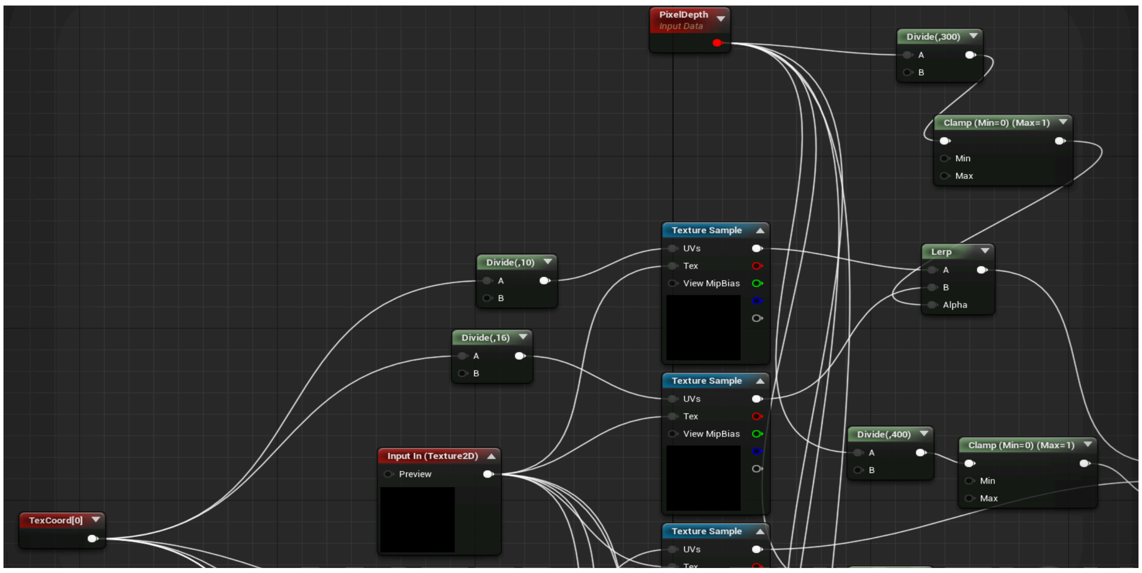

Fig 20: *Setup of a material function to avoid visible tiling by blending texture samples on top of each other, based on the camera's position (Unreal Engine 4 material function)*

This technique only makes sense when it comes to very diffuse textures, such as sand or moss. If it was used on the rock textures, the layers would not blend very well when altering the viewpoint because these textures have a certain directionality due to the visible cracks and other detail. Also, a problem might arise when it comes to the robots' navigation, the mountains serve as landmarks and if the appearance of a landmark would change depending on the robots' distance, this might interfere with the VSLAM algorithm. In this project, it is used only in combination with the *Sand* texture, which is quite diffuse and is mainly responsible for the overall look of the terrain, the *Sand* being the main texture in use. Since the *Sand* has pebbles that can be added on top of it, it would look very weird if these were scaled up all the way to the largest texture scale in use. These pebbles are only blended in for the two closest camera distances by linear interpolating between two S*and* instances, one containing pebbles and one being perfectly flat, both stored as texture objects. The interpolation is based on the camera's distance to the landscape, this being achieved by getting the *pixel depth*, as explained above, dividing it by 200 and clamping it down between 0 and 1 to prevent glitches.

4.2.3 Tessellation

Tessellation is used to get more realistic materials by splitting up triangles into smaller triangles to display height differences, as opposed to only simulating height differences by using normal maps. This way, it increases the surface detail of a mesh, but the triangle splitting is done at runtime in UE4, which might decrease the performance of a program significantly [26]. It is a DirectX11 feature and can only be enabled if this is supported. Tessellation in UE4 also does not work without a dedicated graphics card, which was detected while working on this project on an older Macbook Pro.

Inside the material, the tessellation mode can simply be changed to ‚flat tessellation', which splits up the triangles without smoothing them. This enables the world displacement - and the tessellation multiplier outputs. A height - or displacement map can be used as an input for the world displacement. This is multiplied by a factor to set the distance of the displacement / the strength of the effect. In this case, it's set to 125 for materials of larger scale (*RockM* and *Rock_02*) and to 3 for the S*and* material to keep a realistic look to it. The result is multiplied with a VertexNormal in worldspace and added as the world displacement [*Fig 21*], depending on the paint layer. The tessellation multiplier is a simple constant value which can be used to strengthen the effect. In this case 2 is enough.

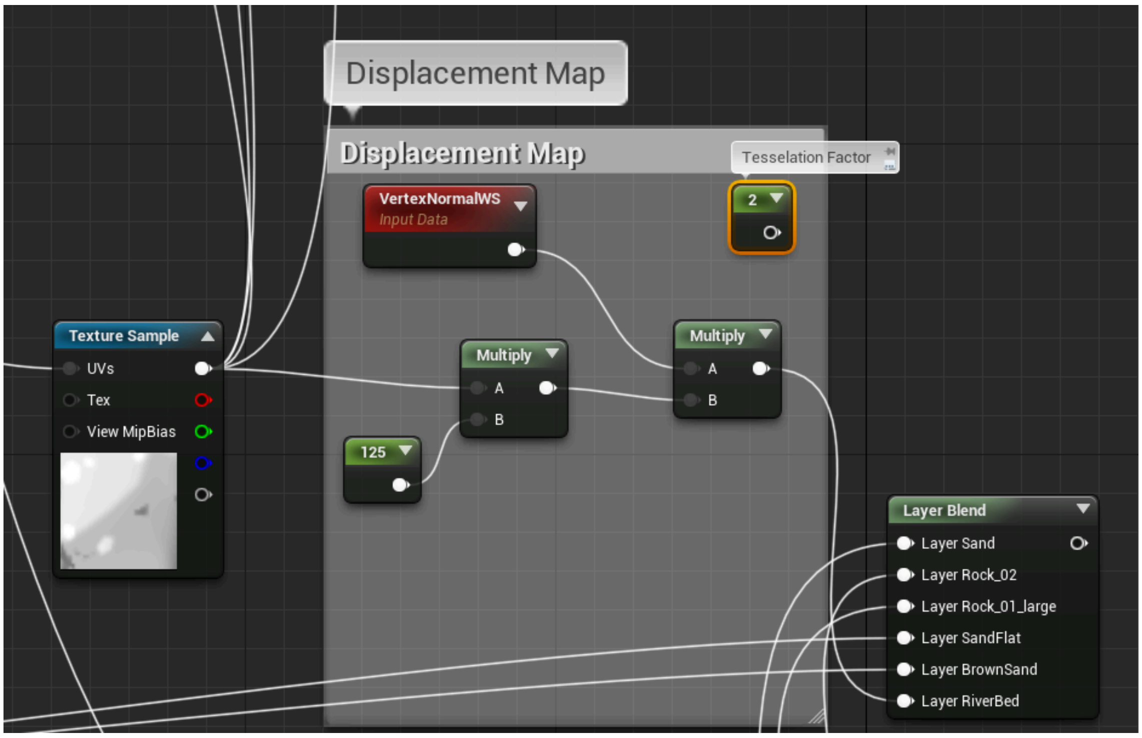

Fig 21: *Setting up tessellation for the landscape material (Unreal Engine 4 material node graph)*

4.2.4 Hole Material

To give the Mars robots a diverse environment and explore reactions to different conditions, it was decided to add in a cave. This requires creating a hole inside the terrain, where rock models can be put later on to form a cave that reaches into one of the mountains. To do so, the master material is simply copied and its blend mode is set to masked. Furthermore, a *landscape visibility* node is plugged into the opacity mask output. The hole material has to have the same components as the master material, otherwise there might be rendering issues. After setting it up, it can be attached to the landscape as a *landscape hole material*.

4.3 Terrain Painting

UE4 offers a terrain painting tool, where the layers that have been set up inside the master material can be managed. First, a layer information for each painting layer needs to be created by simply clicking the *plus* symbol in the righthand corner of each layer and storing the layer information inside the *layers* folder as a *weight blended layer (normal)*. Inside the painting tool, the brush size, falloff and opacity can be adjusted. When a layer is selected, it can simply be painted onto the terrain. The texture in use to paint the hills is *Rock_02* with some red sand in-

between to create smoother transitions. The top part of the landslides are painted with the *RockM* texture and the wall itself is painted in brown sand. Red sand is put into the crevices of the landslides, imitating the part sliding down the cliff wall, to create a transition between the brown and red sand. The slopes are painted in the *Rock_02* material as well and some brown sand is added into the area with lowered opacity to create a better transition between rock and sand. The landscape already has some sharp rock-like parts, which are painted with the *Rock_02* material as well. In some flat ground areas, the *RiverBed* material is added to create the look of a more eroded, more detailed ground, similar to a photo taken by curiosity mentioned before [*Fig 1.3*].
As a last step, a hole is painted onto one of the mountains by using the *landscape hole* material, to remove the ground from this particular position to make adding a cave in later on possible.

4.4 Placing Models

As a finishing step, the previously created 3D models are added into the terrain. The exported files are imported into the models folder. Because they all have individual texture maps, a folder for each rock and rock formation is created for organization purposes. The models are imported and the texture maps are added into the matching material folders, *rock01* (which matches the *RockM* material) or *rock02* (which matches the *Rock_02* material). The materials need to be set up individually this time because they don't come as a combined *.sbsar* Substance file. To do so, an empty material is created and the texture maps are plugged into the corresponding material outputs. Afterwards, one of the two materials is assigned to the model.
When the model is properly textured, it can be placed inside the scene and transformed to fit the landscape. To create some steep cliffs, the *rockFormation_02* is rotated and scaled to resemble a cliff and is added to one of the mountains, repeating this process multiple times until a cliff range is formed. To break the shape up, some *rock_01* models are added in-between.
On the lengthy bumps that come with the terrain, some *rock_01* models and rock formation models are placed to create slopes that *Charlie* can climb. Some other rock models of all kinds are placed into the map to create a diverse environment with obstacles in it. The texture used on these rock models is the *rock02* texture, which equals the *Rock_02* material mostly used in terrain painting, to create a sense of the assets belonging to the terrain. The four models in use can be seen in [*Fig 22*].

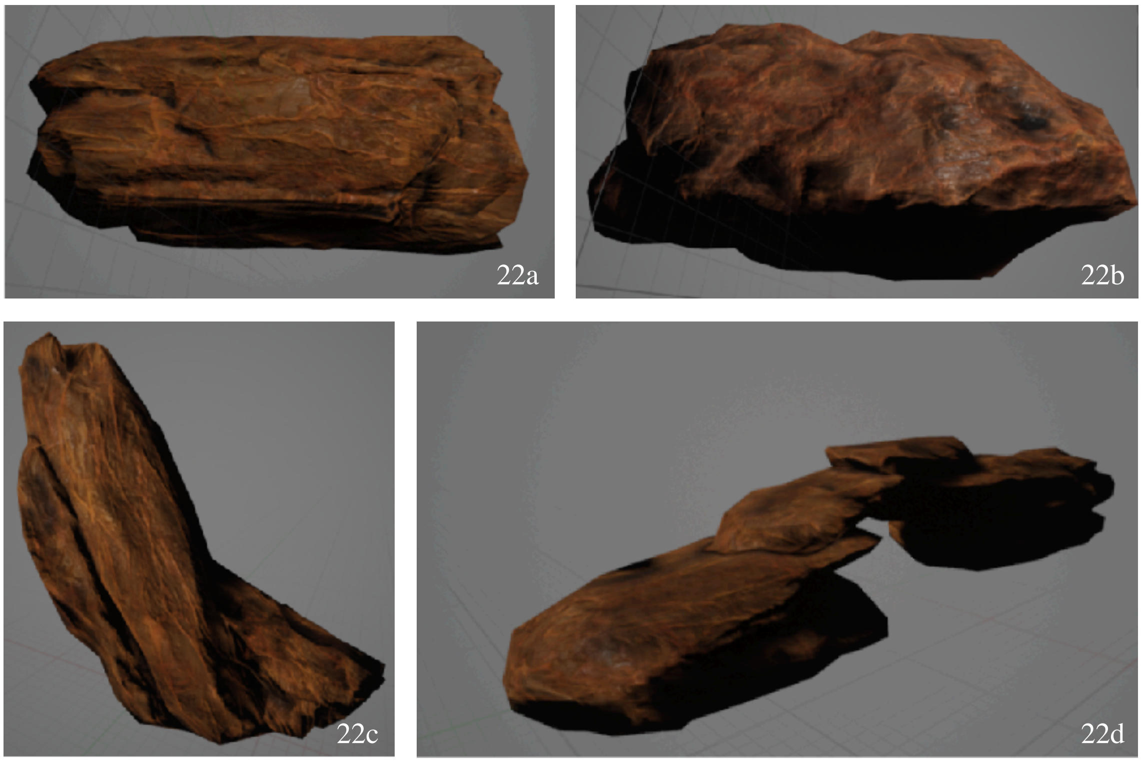

Fig 22: *The 4 different kinds of rock modeled for the terrain, textured in Unreal Engine 4: rock_01 [22a], rock_02 [22b], rockFormation02 [22c], rockFormation01 [22d]*

4.5 Cave

A cave offers the possibility to test out robots in different lighting conditions and crammed spaces. The cave is added on one side of the hill by applying the hole material to the selected area. Now, five black planes in the shape of a cube open at the front are added to define the space the cave should occupy to make sure the light really is blocked out and no robots can accidentally fall off the terrain. Afterwards, the space is filled with rock - and rock formation models of various shapes and sizes, first creating the cave mouth and then creating two small antechambers with entrances still large enough for the robots to fit through. The created space is improved by adding in some smaller rocks for details. The cave's entrance can be inspected in [*Fig 33*].

4.6 Foliage

UE4 provides the ability to paint on so-called *foliage instanced meshes* in the foliage painter, which makes it possible to quickly add a large number of static meshes to a landscape or remove them again.

Foliage is added in the form of small rocks near the cliffs and rock formations to give the effect of weathered, crumbled rock. The meshes used here are meshes of the *rock_01* and *rock_02* models to show their affiliation with the bigger models used. To make the sand plaines more interesting and create a more challenging path for the robots, foliage in the form of previously created clutter [*Fig 23*] is added in some areas around the robots' spawning point. The clutter models that have been created in Maya are imported into UE4 and the material applied is the *Rock_02* material.

To set this up, the 3D model which should be used as foliage needs to be set up as a new foliage actor by simply creating an empty foliage actor and adding in the 3D model as a mesh. The material assigned the model stays the same as well. Inside the foliage tool, the mesh created can be added into the list of foliage and can be painted onto the terrain if selected. Each foliage can have different settings altered, the scale can be randomized between two values which helps create variation when adding multiple instances of one object and the collision can be set to multiple options. It is set to *block all* to make small rocks added in this way serve as obstacles for the robots by blocking all other actors. In the overall brush settings, the paint density and brush size can be set, the paint density determining the space between individual meshes. To erase meshes gradually, the erase density can be increased and meshes can be erased by pressing *control* and *shift* while painting over the target area.

By adding tens of thousands of foliage instances, all to be rendered at once, the performance of the Unreal projects decreased significantly, even though foliage is a performance-friendly way of adding multiple meshes to a terrain. The loss of performance can be prevented by setting up a culling distance for the meshes to be rendered, being between 1500 and 2000 in this case, to make sure not all meshes need to be rendered even when being invisible to the camera.

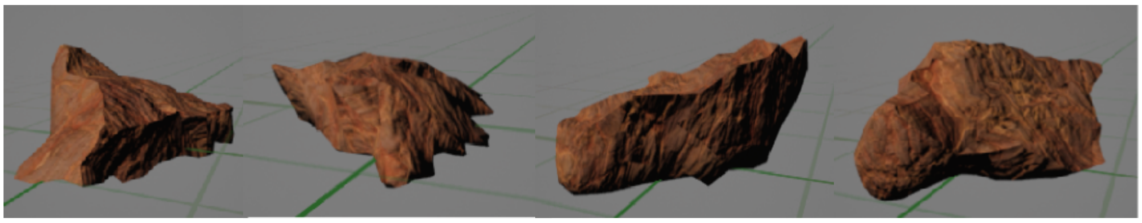
Fig 23: *Different examples of textured clutter used as foliage in Unreal Engine 4*

5 VSLAM Tests

The performance of the visual SLAM algorithm is tested on different variations of the final terrain:

- terrain with no tessellation applied, spawning point at the original position and some rocks cluttering the terrain
- terrain with tessellation applied to the landscape material, spawning point at the original position and some rocks cluttering the terrain
- terrain with no tessellation applied and no material function to prevent obvious tiling added to the landscape material, spawning point at the original position and some rocks cluttering the terrain
- terrain with no tessellation applied, spawning point at a position with larger rocks very close by
- terrain with tessellation applied to the landscape material, spawning point at a position with larger rocks very close by
- initial bland landscape without features added [*Fig 24*]

The tests are run with one rover (UGV) for a duration of 10 minutes each, until the rover has completed a full circle on the terrain. The number of features is set to 2000, meaning that 2000 corners are extracted using the FAST method (chapter 2.1.2). Every run, the rover followes a pre-defined route.
To determine the algorithm's performance, the numbers of successful and failed tracking on each map are measured, as well as the number of inliers matching and if the RANSAC (chapter 2.1.4) could compute a model.

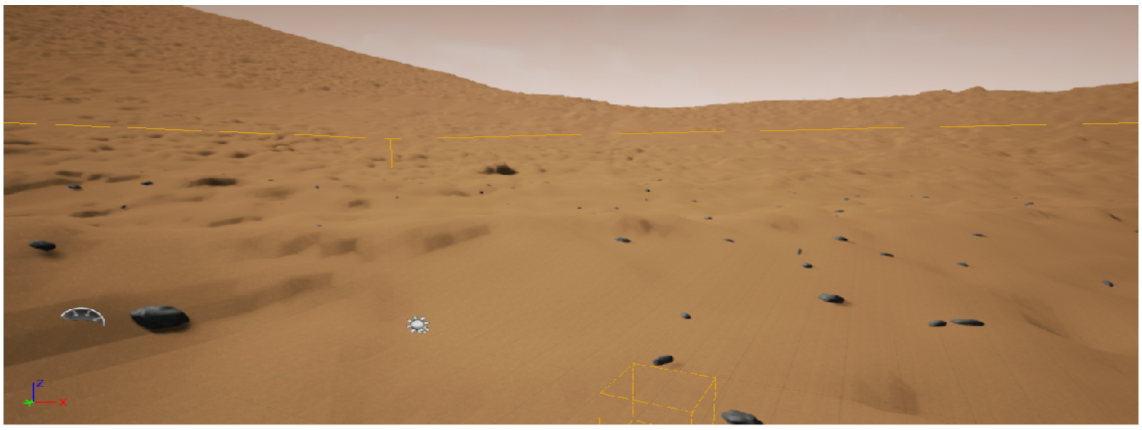
Fig 24: *initial featureless terrain in UE4*

6 Results

6.1 Substance Materials

In the process of this thesis, four different Substance Designer materials were created. These materials can be widely re-used, the *Sand* [*Fig 25*] material allowing to adjust parameters, such as sparkle intensity and - count for the small reflections on the sand, adding and removing small pebbles and sand impurities [*Fig 25b*], adjusting the hue, saturation and lightness of its color - thus allowing universal use in all kinds of terrains - and also adjusting the pebbles' coloration in the same way, as well as their distribution by providing a custom pebble mask. The *RiverBed* [*Fig 28*] material allows for the same kinds of changes, except having a fixed alignment of pebbles. The two *RockM* [*Fig 26*] and *Rock_02* [*Fig 27*] materials have each the option to adjust color in terms of hue, saturation and lightness of rock and overlying sand separately and also customizing the rock's roughness. With every material, it is possible to change its resolution and random seed. A separate C*racks* graph is also included in each material, which can be re-used when creating a new Substance material in need of this effect.

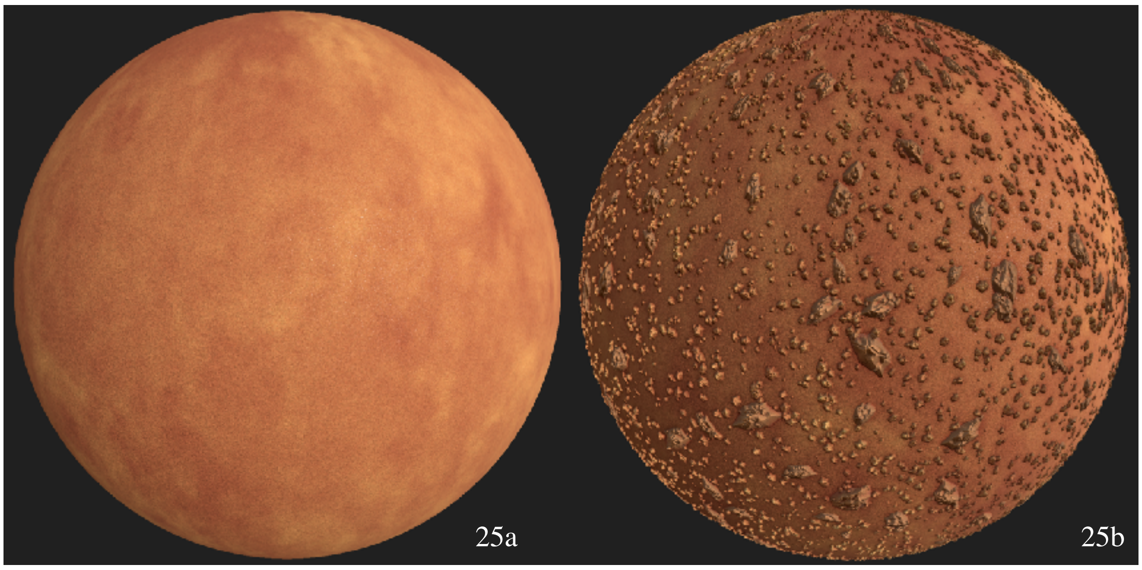

Fig 25: *Renders of the final Sand material, [25a] without pebbles and impurities, [25b] including pebbles and impurities*

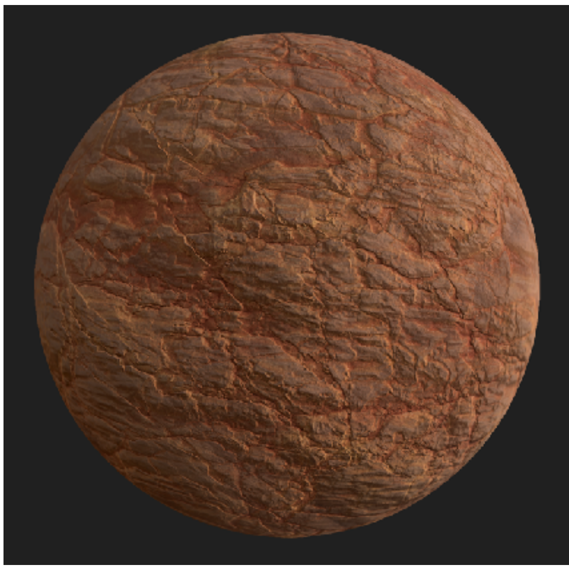
Fig 26: *Render of the final RockM material*

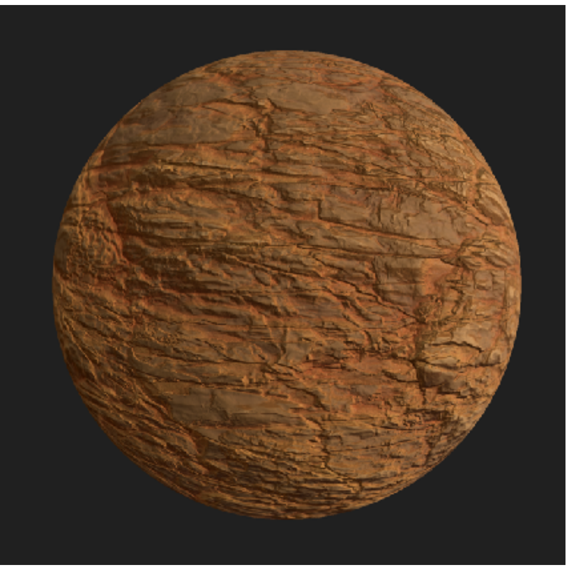
Fig 27: *Render of the final Rock_02 material*

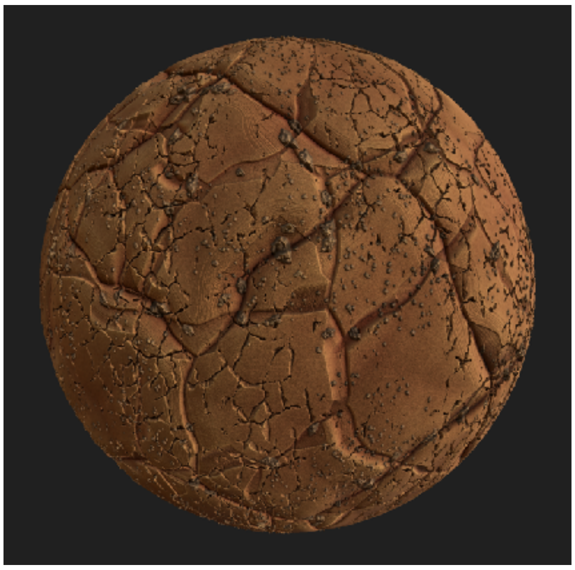
Fig 28: *Render of the final RiverBed material*

6.2 Mars Terrain

A part of the Valles Marineris was recreated in Unreal Engine 4, depicting an area of *2000m * 2000m,* derived from a height map of the original terrain. This area includes high sand slides, a hilly mountain range sloping down into some terraces, a dried river bed and plain and rocky sandy expanses, all created by painting on different materials [*Fig 29 & Fig 30*]. To include more landmarks and physical obstacles into the VTB, 4 different kinds of rocks with 2 different kinds of specified textures were modeled. These consist of a large rock which is more flat and can be used to simulate slopes, another rock, which looks more eroded and has the shape of a meteorite, a rock formation which forms an arc and another formation pointing upwards. Since the *Charlie* robot has the ability to use its arms independently, the arc formation and the modeled slopes can be used as a feature it can explore closely. These rocks were placed around the terrain, one rock formation being used as a line of cliffs and the single rocks to simulate slopes coming out of the ground [*Fig 32*]. Another feature that was added to the landscape is a cave inside one of the mountains [*Fig 33*]. It consists of rocks and has two smaller sub-caves and can be used for navigation in the dark, transitions from dark to light and navigation inside small spaces.

Clutter in the form of small rocks was distributed sporadically onto the terrain, especially around larger rock formations to give the illusion of broken off pieces and to provide more landmarks and obstacles for the robots.

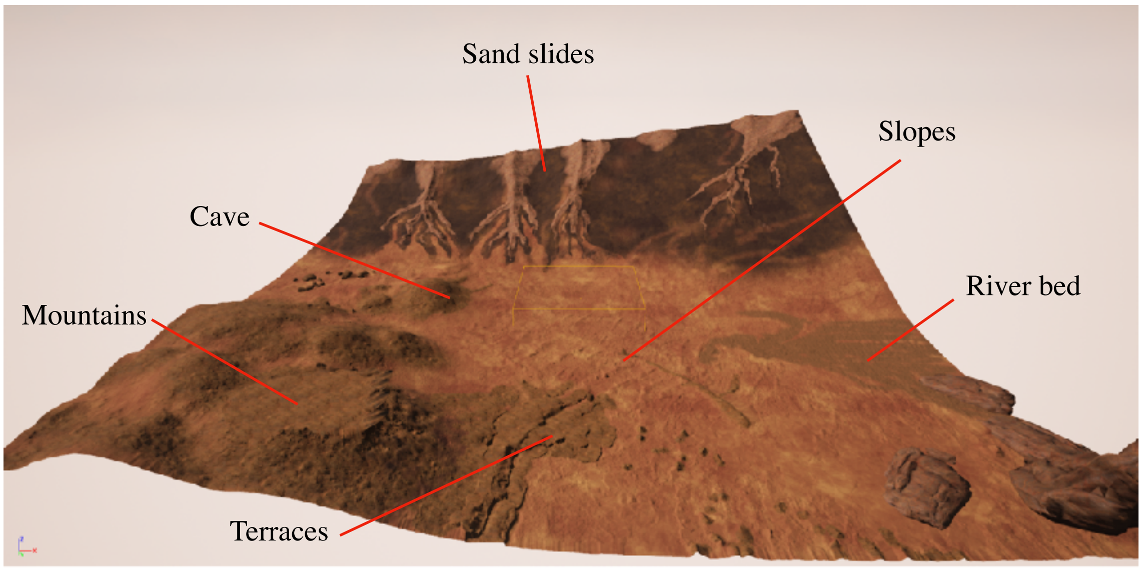

Fig 29: *Overview of the painted landscape in Unreal Engine 4*

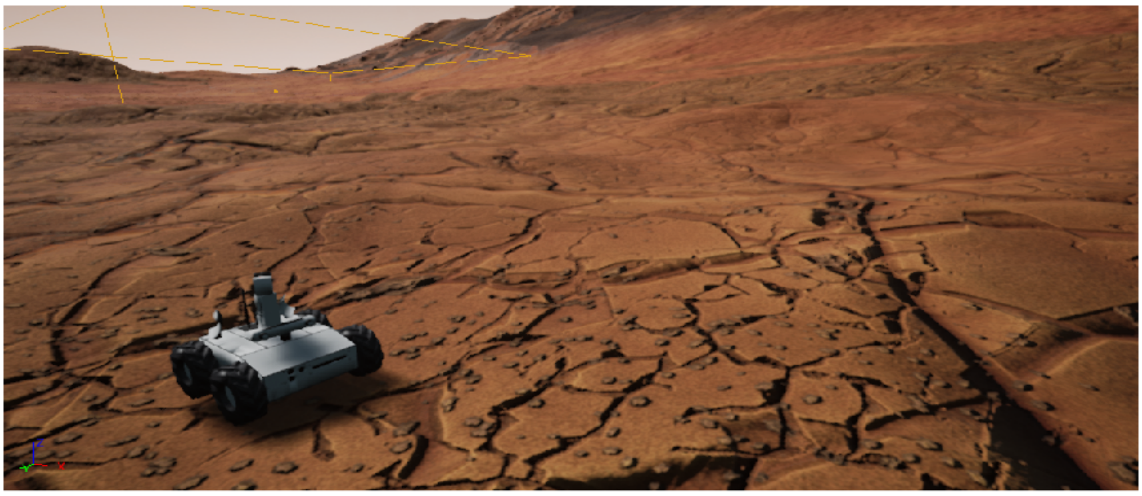

Fig 30: *Dried RiverBed section in Unreal Engine 4, including Mars Rover*

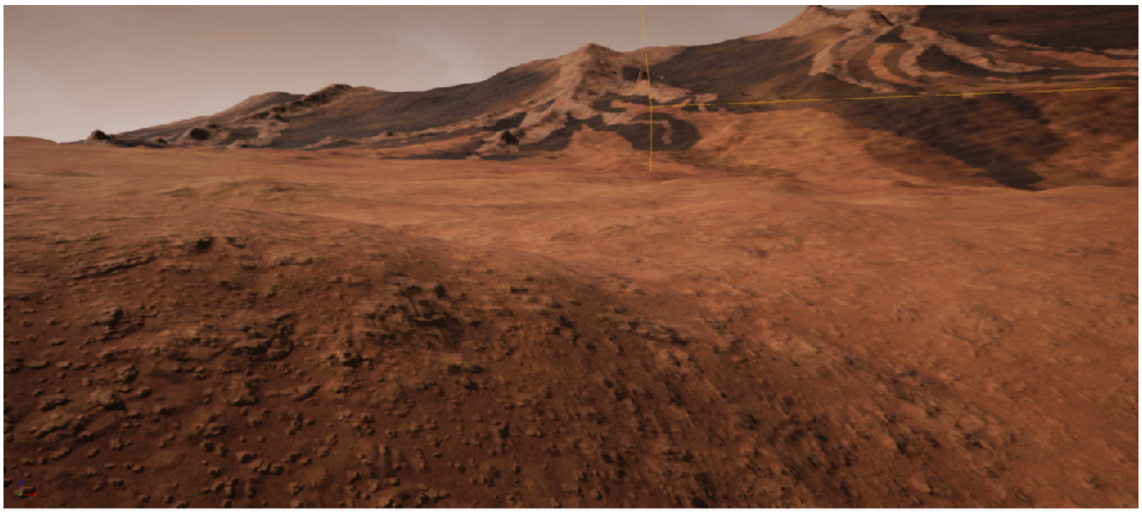

Fig 31: *Sand material with pebbles included, sand slides in the background*

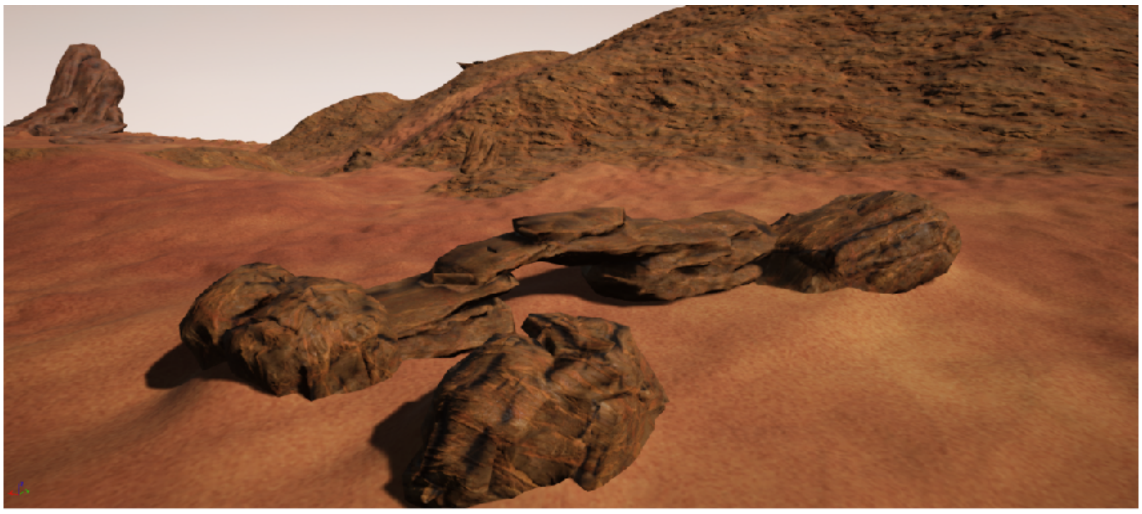

Fig 32: *Rocks and rock formations included in the terrain, mountain range in the background*

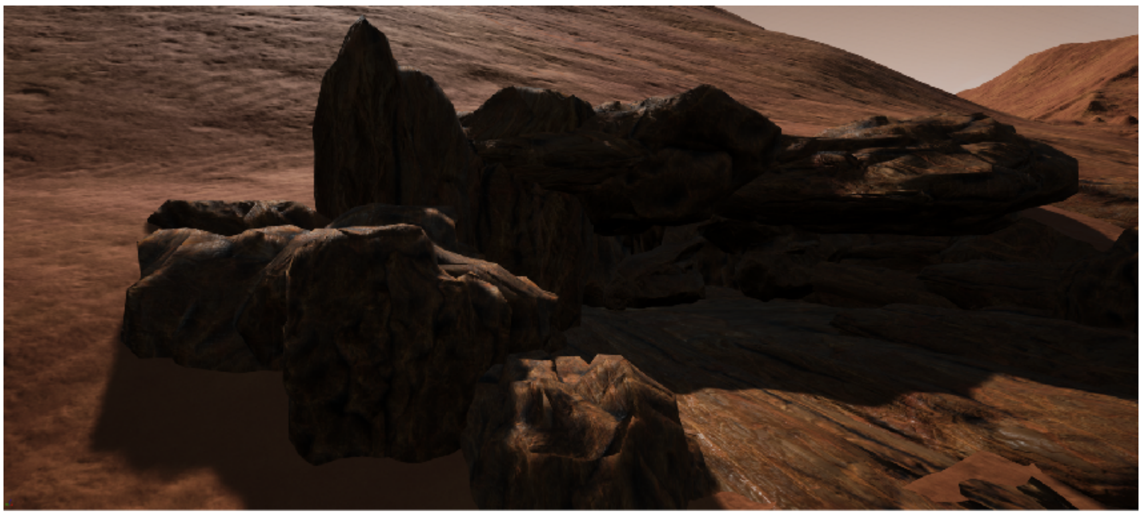

Fig 33: *Entrance to the cave going deeper into the mountain*

6.3 VSLAM Performance

While testing the visual SLAM's performance inside the project, several problems occurred. For once, the colliders of the clutter had to be disabled because the rover used for testing would not recognize any objects in its way as obstacles, tailgate these and get stuck. This also happened when crossing the path of other spawned robots that were not used for testing at that time. This specific problem is possibly rooted in the implementation of Unreal's pathfinding algorithm, which does not consider small clutter. If it did, the algorithm would have trouble finding enough possible paths to follow. Another factor is that the current rover model is not suitable for off-road usage.

The tests run on the algorithm itself resulted for the most part in failed local map tracking on all terrains, the terrain without tessellation having a failure rate of 82,16%, the terrain with tessellation failing in 76,92% of the cases and the untessellated and tiled version having a failure rate of 96,5%. When moving the spawning point to an area with larger features close by, the failure rate of the terrain without tessellated material was 95,74% and the rate when having tessellation enabled was 75,82%. On the initial map, the local map tracking failed in 95,38% of the cases [*Fig 34*]. The results may show a small increase of performance when comparing the initial bland map to the final terrain, however when changing the spawning point, the final terrain's result is worse than the initial map's result. While testing out the algorithm, there was always a point were the feature recognition would stop working completely and new information could not be published anymore. This might be an error in the implementation of the algorithm, which causes the process to terminate early. The stage of loop closure was not reached in the testing phase on any terrain, apparently not enough features were sampled to refine them with the RANSAC (chapter 2.4.1), a maximum of 1 feature being recognized in the process.

The visual SLAM algorithm produced no reliable results in the test. The cause of this might be the very large terrain, a rover can only explore a small part of this in the given time. Its sensors might also not reach the landmarks that are on the terrains's edges, although they are clearly visible when activating the rover's camera view. Another reason for the algorithm's failure could be its implementation, which might not be working correctly yet. It only records one panoramic picture per second, which is too long of a time gap between pictures to find matches when comparing sampled pictures.

The only thing that could be concluded is that enabling tessellation improves the usability of the terrain because more diverse features very close to the rover's sensors might improve its navigation. On the contrary, having the landscape texture tiled very small and therefore having color features appearing repeatedly on a small portion of the terrain might counteract the algorithm's performance.

But since the results obtained by the tests are inconclusive, these two things can not be stated as factual outcome. In the end, it could not be concluded wether the new terrain has increased the visual SLAM's performance or not.

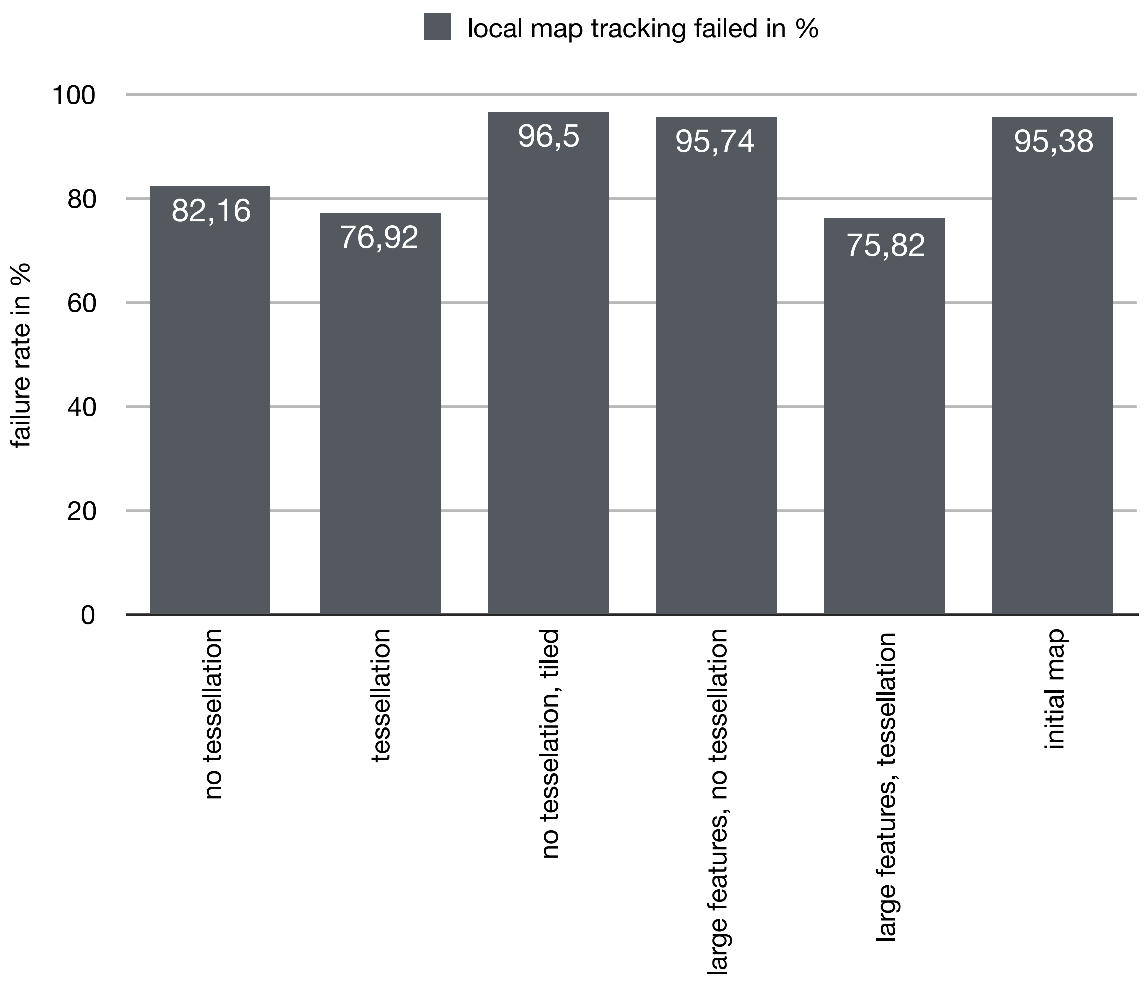

Fig 34: *Different terrain versions and local map tracking failure in %*

7 Conclusion and Future Work

The goal of this thesis was to create a Mars terrain, consisting of a landscape in UE4, different rock models and models for clutter, textures for these models and textures for the landscape. During the process of working on this task, four different, scalable rock models were created, with two different individual materials each, that were all used in this project and can be re-used in other projects or utilized as a base for Voxel sculpting to create new rocks and rock formations. The rock models have a very low poly count (about 800 polygons each), therefore are fast to compute, even on less performant hardware, and can have multiple instances.

The four Substance Designer materials that were generated are versatile and therefore sufficient to texture a Mars terrain in UE4. Because features such as resolution, color, roughness and random seed can be changed inside of 3D software or gaming engines, these textures can be re-used in a wide range of other projects by adjusting their parameters. These textures and models were applied onto a landscape generated from a height map of the Valles Marineris in UE4 and a visually pleasing terrain with differently challenging sections was put together, that is used as an environment in the VaMEx-VTB.

The tests run to determine if the terrain had increased the visual SLAM's performance were inconclusive, it can only be suspected that the feature enriched terrain leads to a better performance, however, a slight improvement is showing in the tests. The problem for this might be rooted in the implementation of the algorithm, since the VaMEx-VTB is still a work-in-progress project.

A next step would be to expand the material collection that has been created to cater to environments needed in the future based on the materials created during the thesis. It is also possible to expand the line of created assets or customize more materials and add them to the already existing ones to have a wider range to choose from when re-using these.

When the VaMEx-VTB's implementation is complete and running, the tests could be repeated to see if there is a difference to the tests held prior and determine to what extend the created terrain helped to improve the visual SLAM's performance.

Glossar

Deutsches Zentrum für Luft- und Raumfahrt (DLR) - is the national space research centre of Germany. Its research extends to aeronautics, energy, transport, digitalization and security and part of its explorer-initiatives is the VaMEx (Valles Marineris Explorer) project [5].

Mesh decimation - is a class of algorithms that reduce the faces, edges and vertices of a given polygonal mesh to lower its polygon count and make it cheaper to compute.

Normal baking - transfers detail from a high poly model onto a low poly model. To bake the high poly meshes normals onto the low poly mesh, rays are cast inwards from a certain distance through the two models and record the surface detail of the high poly model. These are saved as a normal map and can be applied to the low poly model to give the illusion of surface detail [16].

Procedurally Based Rendering - is a method to render light interaction accurately to the real world. It considers energy conversation, absorption, scattering, diffuse and specular reflections and computes lighting in linear space. Therefore, an object can be used in all sorts of lighting conditions, still looking correct [4].

Unreal Engine 4 - is a powerful game engine presented by Epic Games and was used to create the VaMEx-VTB.

UV coordinates - are texture coordinates used to map a 3D object onto a 2D plane, which is called UV-mapping. To texture an object, the UV coordinates need to be unwrapped [7].

Valles Marineris - is the largest canyon system in our solar system and spans for over 4000 kilometers with a depth of 7 kilometers, located along the martian equator. It is mostly unexplored [6].

Virtual Testbed - are mainly (bur not exclusively) used in the field of robotics and provide an option to test new concepts in a save and cheap virtual environment and combine complex development projects by utilizing efficient simulation algorithms. This results in an increase of development speed and robustness, because units can be tested in every environmental condition imaginable and data can be collected and analyzed to improve performance [13].

Voxel sculpting - has no topological restraints, as opposed to geometrical sculpting, it mimics sculpting with clay. It uses voxels, which can be described as volumetric pixels, points on a three-dimensional grid with a position relative to the surrounding voxels. The most commonly used Software for voxel sculpting is 3DCoat.

Bibliography

[1] Bavoil, L., Sainz, M., Dimitrov, R.: *Image-space horizon-based ambient occlusion*. ShaderX7 Advanced Rendering Techniques (2008). http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.214.6686&rep=rep1&type=pdf (visited on 29/04/2018)

[2] Calonder, M., Lepetit, V., Strecha, C., Fua, P.: *BRIEF: Binary robust independent elementary features*. In: ECCV on Computer Vision (2010), 778–792.

[3] CGVR Universität Bremen. *VaMEx-VTB*. http://cgvr.informatik.uni-bremen.de/research/vamex-vtb/index.shtml (visited on 29/04/2018)

[4] Deguy, S.: *The New Age of Procedural Texturing*. (2015). https://academy.allegorithmic.com/courses/the-new-age-of-procedural-texturing (visited on 04/05/2018)

[5] Deutsches Zentrum für Luft- und Raumfahrt. *DLR im Überblick*. https://www.dlr.de/dlr/desktopdefault.aspx/tabid-10002/#/DLR/Start/About (visited on 29/04/2018)

[6] Deutsches Zentrum für Luft- und Raumfahrt. *VaMEx*. https://www.dlr.de/kn/desktopdefault.aspx/tabid-4309/3222_read-35897/admin-1/ (visited on 29/04/2018)

[7] Ebert, D., Musgrave, F. K.: *Texturing & Modeling: A Procedural Approach* (2003).

[8] Fischer, R.: *Prozedurale Generierung von Multi-Biom-Landschaften*. Master's thesis, Universität Bremen (2019). http://cgvr.informatik.uni-bremen.de/theses/finishedtheses/multibiom/Thesis_Fischer.pdf (visited on 29/04/2018)

[9] Fischler, M., Bolles R.. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. (1981) Comm. ACM. 24 (6): 381–395.

[10] Geology. *Rocks on Mars*. https://geology.com/stories/13/rocks-on-mars/ (visited on 29/04/2018)

[11] Gil, A., Mozos, O.M., Ballesta, M. et al.: *A comparative evaluation of interest point detectors and local descriptors for visual SLAM*. Machine Vision and Applications (2010) 21: 905-920. https://doi.org/10.1007/s00138-009-0195-x (visited on 29/04/2018)

[12] Hyttinen, T.: *Terrain synthesis using noise*. Master's thesis, University of Tampere (2017). https://tampub.uta.fi/bitstream/handle/10024/101043/GRADU-1494236249.pdf (visited on 29/04/2018)

[13] MMI RWTH Aachen. *Virtuelle Testbeds*. https://www.mmi.rwth-aachen.de/forschung/virtuelle-testbeds/ (visited on 29/04/2018)

[14] Mur-Artal, R., Montiel, J.M.M., Tardos, J.D.: *ORB-SLAM: a Versatile and Accurate Monocular SLAM System*. IEEE transactions on robotics (2015) 5/31: 1147-1163. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7219438 (visited on 29/04/2018)

[15] Mur-Artal, R., Montiel, J.M.M., Tardos, J.D., Galvez-Lopez, D.: *ORB-SLAM2*, https://github.com/raulmur/ORB_SLAM2/blob/master/README.md (visited on 29/04/2018)

[16] Polycount. *Texture Baking*. http://wiki.polycount.com/wiki/Texture_Baking, (visited on 29/04/2018)

[17] Polycount. *Normal Map*. http://wiki.polycount.com/wiki/Normal_map (visited on 29/04/2018)

[18] Rosten, E., Drummond, T.: *Machine learning for high-speed corner detection*. Proceedings of the 9th European Conference on Computer Vision, (2006), Springer Berlin/Heidelberg, pp: 430-443

[19] Substance Designer Documentation. *Ambient Occlusion (HBAO) (Filter Node)*. https://support.allegorithmic.com/documentation/sddoc/ambient-occlusion-hbao-filter-node-159450550.html (visited on 29/04/2018)

[20] Substance Designer Documentation. *Crystal 1*. https://support.allegorithmic.com/documentation/sddoc/crystal-1-159450702.html (visited on 29/04/2018)

[21] Substance Designer Documentation. *Perlin Noise*. https://support.allegorithmic.com/documentation/sddoc/perlin-noise-166363420.html (visited on 29/04/2018)

[22] Turbosquid. *3D Mars Surface Full Scene by 3D Multimedia*. https://www.turbosquid.com/3d-models/3d-mars-surface-scene-terrain-landscape-1215082 (visited on 29/04/2018)

[23] University of Arizona HiRISE. *About us - Principal & Co-Investigators*. https://www.uahirise.org/epo/about/ (visited 07/04/2019)

[24] Unreal Engine Documentation. *Creating and Using Custom Heightmaps and Layers*.https://docs.unrealengine.com/en-us/Engine/Landscape/Custom (visited on 29/04/2018)

[25] Unreal Engine Documentation. *Roughness*. https://docs.unrealengine.com/en-us/Resources/ContentExamples/MaterialNodes/1_4 (visited on 29/04/2018)

[26] Unreal Engine Documentation. *Tessellation*. https://docs.unrealengine.com/en-us/Resources/ContentExamples/MaterialProperties/1_8 (visited on 29/04/2018)

[27] Unreal Engine Forum. *Texture samplers limitation is still present in 4.9?*. 09/28/2015, 09:07 PM https://forums.unrealengine.com/development-discussion/rendering/57394-texture-samplers-limitation-is-still-present-in-4-9?85868-Texture-samplers-limitation-is-still-present-in-4-9= (visited on 29/04/2018)

[28] Van Opdenbosch, D., Steinach, E.: *Collaborative ORB-SLAM2 Framework*. https://github.com/d-vo/collab_orb_slam2/blob/master/README.md (visited on 29/04/2018)

[29] Van Opdenbosch, D., Oelsch, M., Garcea, A., Aykut, T., Steinbach, E.: *Selection and Compression of Local Binary Features for Remote Visual SLAM*. IEEE International Conference on Robotics and Automation (ICRA) (2018) 7270-7277. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8463202 (visited on 29/04/2018)

[30] Worley, S.: *A cellular texture basis function*. In: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques (1996), 291–294. http://weber.itn.liu.se/~stegu/TNM084-2017/worley-originalpaper.pdf (visited on 29/04/2018)

[31] Worstplayer. *UE4 Tutorial - simple trick to fix repeating textures*. https://www.youtube.com/watch?time_continue=4&v=bxJHTlP64BE (visited on 29/04/2018)

[32] Yousif, K., Bab-Hadiashar, A. & Hoseinnezhad, R.: *An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics*. Intell Ind Syst (2015) 1: 289-311. https://doi.org/10.1007/s40903-015-0032-7 (visited on 29/04/2018)

List of Figures

Appendix

Data volume

The data volume enclosed contains the following files:

- Substance Designer material files
  - *.sbs* files for each material, containing the full graph, that can be opened in Substance Designer
  - *.sbsar* files for each material, ready to be used in 3D software that supports Substance materials
  - *.png* files of rendered materials

- Textured models
  - *.obj* files of high - and low polygon models
  - *.tga* files of corresponding color, roughness and normal maps for 2 rock materials each
  - *.fbx* files and corresponding texture files of clutter

- Terrain screenshots
  - *.png* files of high resolution screenshots of the terrain

- VSLAM results
  - *.rtf* files containing the VSLAM test results for each test case

- Unreal Project „VaMEx-VTB“ (can also be found on *https://gitlab.informatik.uni-bremen.de/cgvr/VaMEx-VTB* on the landscape2 branch)