Laura Oravakangas

Game Environment Creation:

Efficient and Optimized Working Methods

Bachelor of Business Administration

Autumn 2015

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

TIIVISTELMÄ

Tekijä: Laura Oravakangas

Työn nimi: Peliympäristön toteuttaminen: Tehokkaat ja optimoidut työskentelymenetelmät

Tutkintonimike: Tradenomi (AMK), Luonnontieteiden ala

Asiasanat: peli, grafiikka, 3D, ympäristö, 2D, tausta, kenttä, suunnittelu, optimaalisuus

Opinnäytetyö käsittelee peliympäristöjen toteutukseen liittyviä työskentelymenetelmiä, miten tehokkaita ne ovat ja ovatko aikaansaadut tuotokset tarpeeksi optimaalisia pelin, pelialustan ja muiden huomionarvoisten osien kannalta. Lisäksi työ käsittelee peligrafiikan perusteita ja kenttäsuunnittelua, syventyen niihin kumpaakin sekä yleisellä että tekniikan tasolla.

Pääsääntöisesti kappaleissa olevat esimerkit on havainnollistettu joko Autodesk Maya -mallinnusohjelmalla, Photoshopilla tai Unreal Engine 4 -pelimoottorilla. Opinnäytetyö painottuu kuitenkin yleisiin käsitteisiin, joita voidaan tavata eri nimikkeillä useissa samantapaisissa ohjelmistoissa.

Peliympäristöjen kehittämisessä, sekä 2D ja 3D grafiikassa, on omat haasteensa. 2D -peliympäristöt näyttävät litteiltä ja elottomilta, kun taas 3D peliympäristöt tarvitsevat enemmän optimointia suurien tiedostomäärien lataamisen ja prosessoimisen vuoksi.

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Author: Laura Oravakangas

Title of the Publication: Game Environment Creation: Efficient and Optimized Working Methods

Degree Title: Bachelor of Business Administration (UAS), Business Information Technology

Keywords: game, graphics, 3D, environment, 2D, background, level, design, optimal

This thesis deals with the working methods related to the creation of the game environments, how effective they are and whether or not the created assets are optimal enough for the game, the gaming platform or other aspects that seek attention. Additionally this work presents the basics of game graphics as well as level design, while focusing on them both at the general and technical level.

The examples in the paragraphs are generally presented with either modelling software Autodesk Maya, Photoshop or the game engine Unreal Engine 4. However, this thesis focuses on general concepts and information, which can be found as different terms in similar software.

The development of game environments, both in 2D and 3D graphics, has its own challenges. 2D game environments tend to easily look flat and lifeless, while 3D game environments require more optimization because of the downloading and processing of the large amounts of assets.

FOREWORD

In spring 2015, this thesis started as an exploration on what to do after graduation, even though I already had a clear vision about creating game graphics as a career choice. I wanted to specialize in something and realised that environments, exploration and a dash of level design were the things that I craved for.

First, I want to give special thanks to my fiancé and Game Producer Ilkka Vallo for being a patient and calming person in times when I had dived too deep into work, but also for energizing and keeping me going when life seemed too dull. I would also like to thank my friend and Taiga Studios Chief Executive Officer Janne Remes for uncovering new ideas and generally creating a more positive atmosphere every time we met. Additionally, my friends around the globe deserve a praise for their supporting personalities.

Last but not least, I would like to express my deepest gratitude to my family. Even though the learning process will never end, they have guided me through this journey of education and survival, but have also given me the freedom to choose my own path. They are the greatest stronghold I can always come back to during my travels.

Espoo 27.11.2015

Laura Oravakangas

TABLE OF CONTENTS

LIST OF SYMBOLS

Albedo	Other name for diffuse or base colour.
Ambient occlusion	Value of non-directional light accessibility for surfaces, which darkens the enclosed areas of an object with no clear shadows.
Foreshortening	The illusion of an object appearing shorter when it is angled towards the viewer.
Negative space	The visible space of a silhouette filled with other than the object itself.
Node system	A system that uses individual units of scripts, math and other inputs to create visual algorithms.
Physically based rendering	(PBR). Technology that tries to mimic the light and surface properties as presented in real life.
Placeholder graphics	Graphics composed of usually rough looking assets that will be replaced with more polished assets.
Procedural generation	A technique that uses algorithms instead of manual work to create new data, assets, etc. Extremely useful for generating enormous amounts of randomized content.
Rendering	Creation of an image or frame to the screen.
Texel	A texture region of a 3D object.
Texel resolution	The size of texture pixels in the 3D world.
Triple A title	A term for successful games that have high commercial and development budgets as well as rating.

1 INTRODUCTION

Psychologists call the lasting effects of first impression as the “halo effect”, which is determined by the brain’s fast judgement of the milliseconds of visible information (Hopkin, 2006). First experiences that a player will get from a game will be from its art, regardless of whether it is from a video, screenshot or the game itself. With every next game published, players are demanding more content and better looking graphics than the previous games had, which causes a lot of pressure to the game developers. And because of the demand, the game environments are becoming one of the most expensive entities when making game graphics.

This thesis focuses on the aspects of level and asset design, asset production, optimization and overall construction of both the 2D and 3D game environments. It is aimed to be a collection of essential information about the general game environment development, ranging in details from small independent game studios to big, self-publishers of triple A titles. However, software and tools generally have the same principles between each other, so there is only a tiny amount of information how to use which software. Additionally, some terms under the basics of 3D or 2D graphics can usually be applied to each other.

2 GAME ENVIRONMENTS

Game developers try to seek the all main elements of the real life that contain beautiful and interesting details, and compress them to the much smaller worlds of games. And by correctly exaggerating these elements of interest the entertainment value can increase tremendously. A developer wants the player to experience something new every time he is going to a new area. (FZDSCHOOL, 2011, 2012)

2.1 Different environments

There are basically two types of player exploration in games: linear and non-linear. Non-linear environments give the player more freedom to choose where to go, whereas linear environments can be simply explained as a straight path from point A to point B, sometimes giving players a choice to select a path. Both of them have their benefits and disadvantages. Linear environments are easier to combine with a narrative and minimize the problems of navigation, from which the player can get easily frustrated. However, linearity does not offer much re-playability. Non-linear games have problems with storytelling, but offer a lot of different activities. (Williams, 2014)

Sandbox games are not the same as open world games. They are about the player creating his own experience in the game world, like finding a new way to play it. There are basically two types of open world environments, which have been seen in games through history. First are the seamless worlds or areas which smoothly transition to each other with no clearly seen theme, onto which some smaller parts like cities, dungeons, buildings, etc. are then scattered (See figure 1). This can be done fairly easily and fast when designers do not have to communicate about every single decision they might make. They are basically making the same area, so it is easier to check each other’s work. (Extra Credits, 2015c)

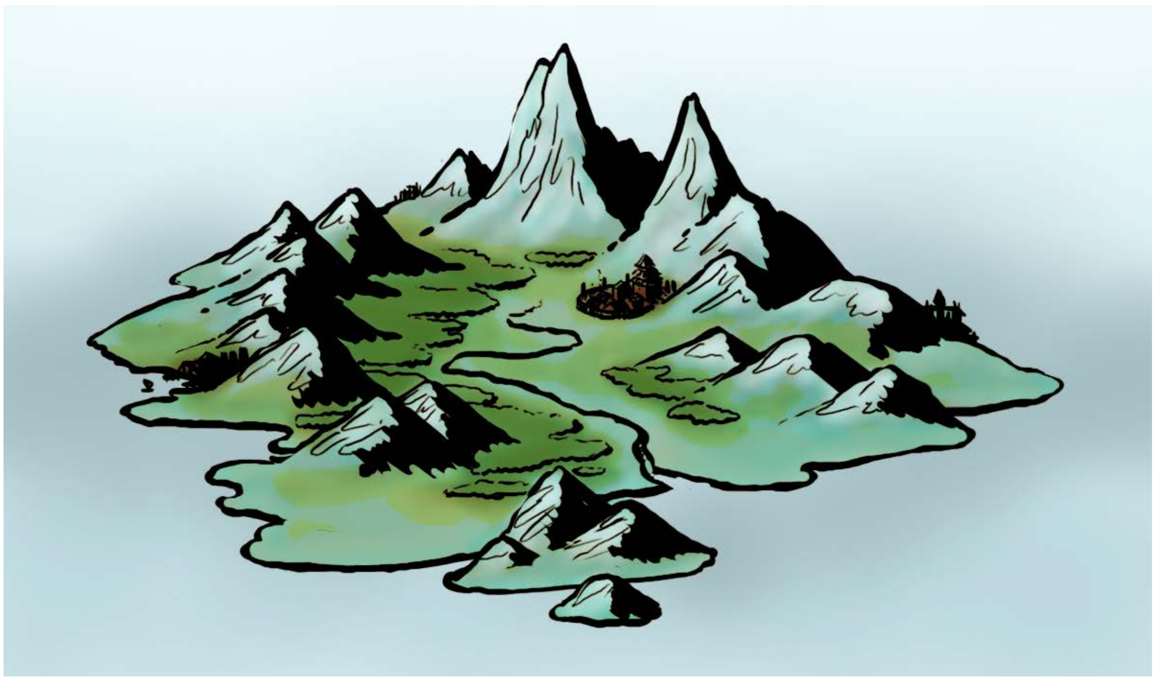


Figure 1. Seamless worlds.

The second type of open world environment is the world made of different playgrounds that have borders with each other and might have a theme like an icy tundra or a hot, lava spewing volcano (See figure 2)(Edge Staff, 2015). This is great for story-telling and player experience. The problem with the playgrounds, shortly modules, is that they are more specific than one huge world, and require more attention and communication in the design apartment. This is because not everyone can make the same quality work, or to make an area or a zone complete in a time limit which is acceptable for the rest of the team. Also the feeling of having completely differently themed zones does not usually feel natural. (Extra Credits, 2015c) For example typical MMO roleplaying games have several different areas with different scenarios like the desert, swamp, mountains, etc. which the players can explore. (FZDSCHOOL, 2012)

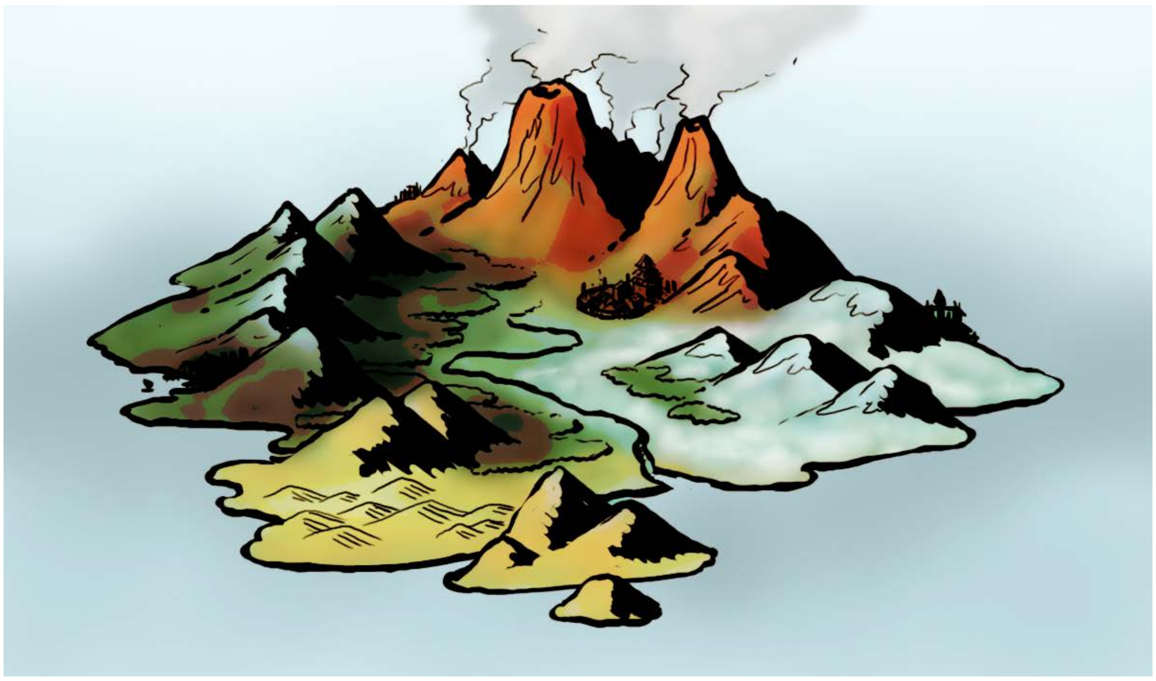


Figure 2. World made of modules.

2.2 What affects the environments?

There are several limits that game developers come across when making environments for games, be it because of performance or gameplay. (Cheng, 2014a) The balance between the quality graphics and the time required to make them is the key to recreating a believable world. (Maxinow, 2009) Generally a larger environment such as open landscape is harder to make because of their huge field of view and requirement of skillful level designers and artists. Even though small environments are easier to control and they are not so memory and performance restrictive, they can too become a huge task if the amount of desired visuals is large. (Burkart, 2015)

2.2.1 Desired frame rate

Frame rate means how quickly images are rendered on screen, and higher the frame rate, the smoother and crispier the movement or the illusion of it. How many times a second the computer updates the game's image on screen is usually set to 30 frames per second, shortly FPS, or 60 FPS, which purely depends on the game the developer wants to make. Frame rate under 30 FPS feels bad to control and can put off the player’s experience of the game. However, the 60 FPS can too seem odd at games which has a lot of movement. This is because of the 24 FPS frame rate which has been used in the film industry for a long time. Shooting a film in 24 FPS creates a lot of different artefacts in the illusion of movement like motion blur. The audience, players and movie watchers alike, have already accustomed to this so much, that the game developers try to add these artefacts to their games manually, even though the frame rate is at 60 FPS. (Extra Credits, 2015a)

When the game seems laggy, meaning the screen is stuttering on certain frames, it means that the computer is having a hard time processing all the data and rendering it on the screen. This can be caused by multiple factors, and the solution to fix those without reducing the frame rate is usually making the graphics, lighting etc. less complex. This is not always the best solution as better graphics can get better results in sales than better frame rates. (Extra Credits, 2015a)

2.2.2 Platform

The choice of the platform defines how much detail the developer can put to his game. Each of the platforms have their own limitations and capabilities over the other, like mobile phones’ ability to go anywhere with the player due to their small size. In a matter of hardware which the console, PC or a mobile phone has, consoles are relatively easy to handle because of the same minimum technology what they have in every console. PCs can be huge powerhouses or small laptops, which do not have the same internal memory capacity, graphics cards and so on. But PCs are still pretty standardized as there are only a handful of hardware manufacturers and suppliers, at least when compared to mobile phones. But with each new generation of hardware, there will be less limits than before (See figure 3). (Lecky-Thompson, 2008, pp.79-86)

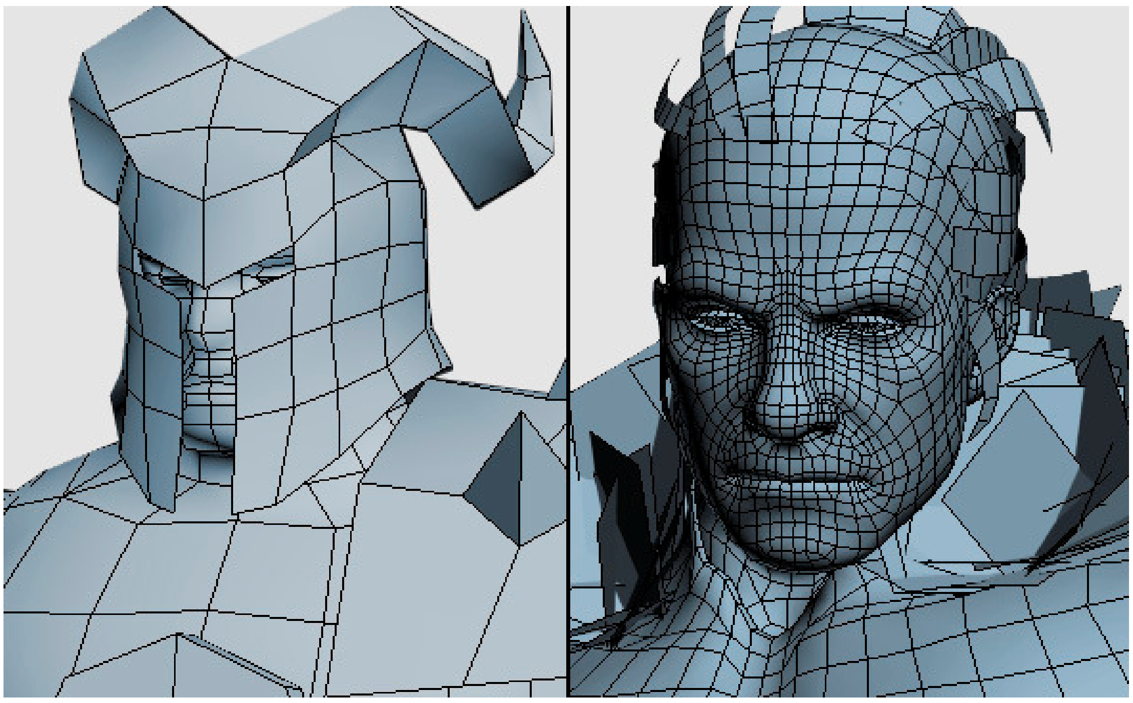


Figure 3. Comparison of Playstation and Playstation 3 models. Adapted by author from (Masters, 2014).

2.2.3 Genre and camera view

Choosing a certain game genre will dictate the platform choice to a certain point, just like the platform choice dictates the game's graphics. (Lecky-Thompson, 2008, pp. 77-78) And choosing a certain camera view can define a certain genre, because of the earlier games that have already used the same view. For example horror games have been historically divided into two sections: puzzle games and action oriented games, which both of them need the most out of the horror atmosphere. (Reed, 2015) There are several camera types regardless of how many dimensions the game has: Side-view, top-down view, third-person view, first-person view and everything between like pre-set locations for the camera, which could over-the-shoulder aiming, lock-on targeting, etc. Shifting from one camera angle to another can affect the immersion of the game greatly. For example Batman: Arkham Asylum uses this in cases where the player goes in to an air duct and changes the camera view from third person to first person to give that claustrophobic feeling. (Anhut, 2011)

In 2D games like side-scrollers, it is easy to see the environment and the relevant information for the player to progress, because the player can see the levels only from one direction. However, because of this the two dimensional games are generally more reliable to control, and cause less frustration and confusion (Totilo, 2010). 2D-games unfortunately fail at providing long distance information, at least when the camera view is from the side. (Anhut, 2011) This is not always the case for 3D-games where the camera is floating around the player and usually, but not always, could be controlled so that the player can adjust the view to his liking. (Gamesradar_US, 2010)

Having the view from a certain view also defines some aspects of the production of the environment. For example when making a 3D side-scroller game, there is no need to have back-facing polygons in the 3D-objects. (Malmqvist, 2001) In 3D multiplayer games the constraints of making an immersive environment increases greatly. This is because if the polygonal budget in a game is for example one million polygons in view at a time, and one player character is about 50 000 polygons, there will not be much polygons left for the environment if the maximum player count is designed at ten. (Masters, 2014d)

The player needs to be able to measure the distances in the environment and the hazards there might be so that he could navigate through them easily. (Anhut, 2011) But generally speaking, the closer the camera is to the surface of the environment, more there should be details. And in cases where the player can look at objects like in first-person, the objects should be polished so that the immersion of the game is more accurate. (HD Admin, 2012)

With the recent growth of interest in virtual reality and especially Oculus Rift, the needed amount of polish to the game environments can, and will increase into enormous amounts. Virtual reality means that a virtual world is filling the player’s vision, which can give a better immersion of the world, at least when compared to a limited, ordinary monitor screen that takes only a portion of the total field of vision. Because of this, the players have a greater view of the environment, and can easily get closer to objects and structural elements. (Oculus VR, 2015)

3 BASICS OF 2D GAME GRAPHICS

Two dimensional presentation of the game can solve issues both in the gameplay aspect as well as in the budget, but only if it is decided early. 2D game development generally requires a smaller skillset than 3D game development, and thus is widely used in the independent game studios that do not have large teams. Artists have a lot of control over the graphics in 2D games, as environments are more predictable because of the fact that they convey information more easily than 3D environments. As an example, the stealth game Mark of the Ninja's environments convey information excellently in regards to its two dimensional gameplay (See figure 4) (Extra Credits, 2014). Graphics of certain styles, such as pixel, vector or anything else the development team desires, are also easier to develop for 2D games because of the locked camera placement. (Ohlew, 2014) However, it does not mean that simple, clean graphics would not be less attractive than fully detailed drawings. (Lovato, 2015)

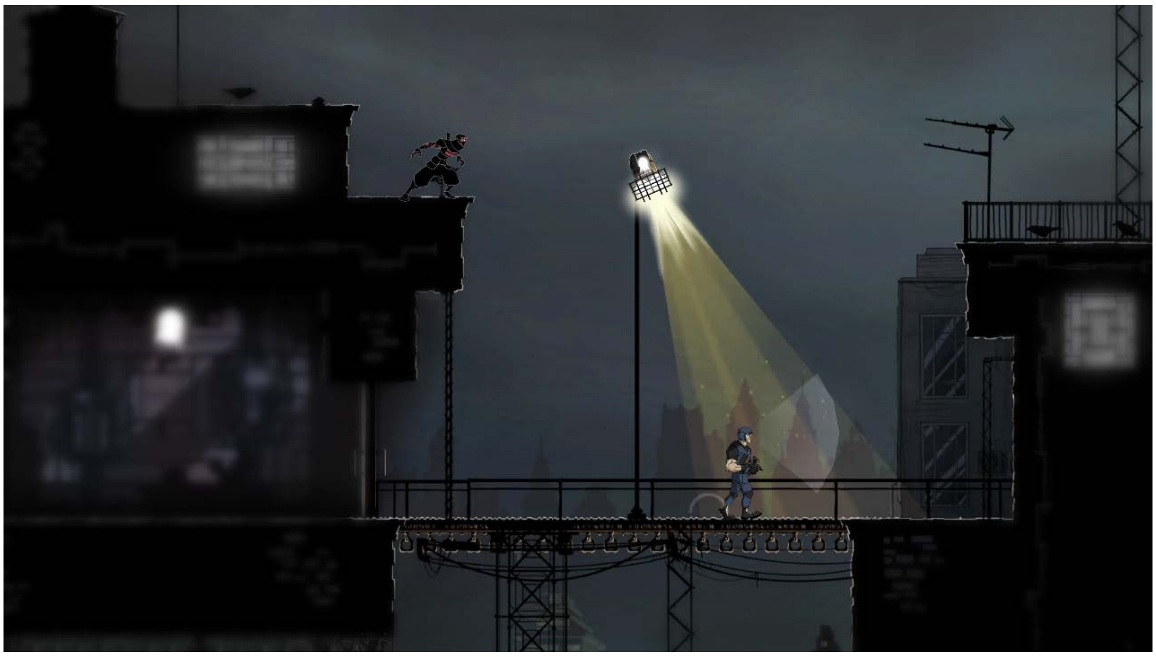


Figure 4. Mark of The Ninja screenshot. (Kei Entertainment, 2015)

3.1 2D-graphics

Monitors are composed of pixels, small square shaped dots that contain colour information. (Curtin, 2011) There are basically two ways to create digital images, either by using raster or vector graphics. Raster graphics, for example photographs, are based on pixel grids, and offer a lot more editability in details. Vector graphics are based on mathematical shapes, curves and lines that define areas for colour to come in, which offers infinite scalability without losing any details (See figure 5). (PsPrint, n.d.)

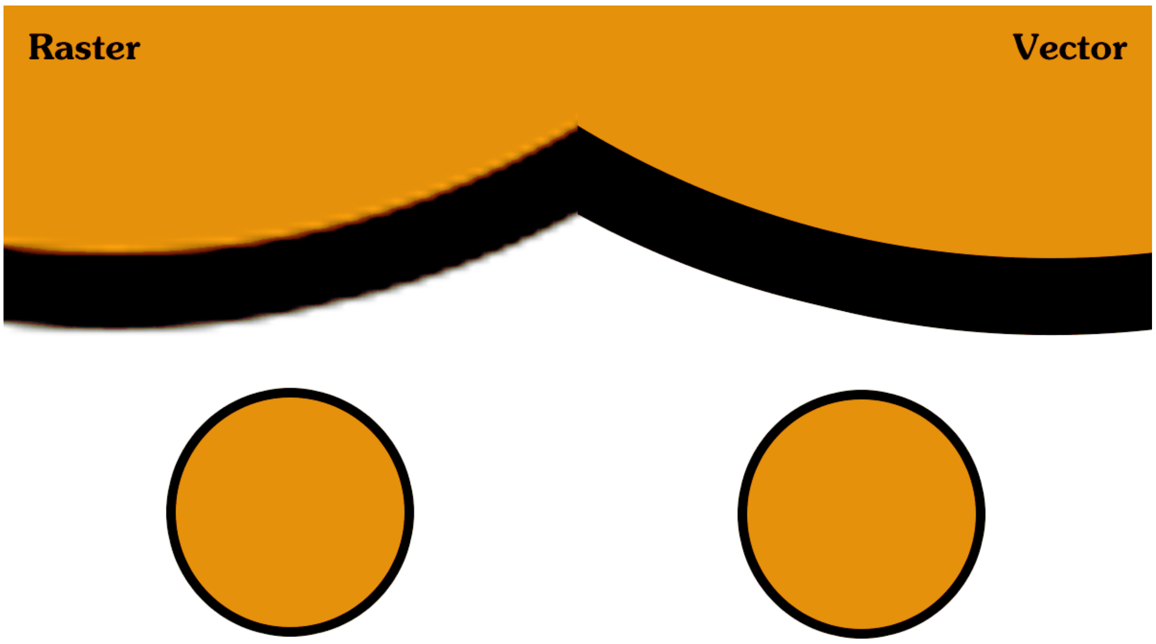


Figure 5. Comparison of identical balls both in raster and vector graphics, which are then scaled six times bigger.

Unfortunately, both the vector and raster graphics have their downsides. Vector graphics do not display small, natural nuances without requiring extreme detailing, and are generally used with printable media like logos. (PsPrint, n.d.) The raster graphics create generally larger files and tend to get blurry when re-sized, because the computer tries to guess the missing pixels. (PhotoBiz, 2013)

Still, most of the 2D games use raster graphics, in short bitmap images. This is because vector graphics need to be processed in real-time, which can get really heavy for the processor when there are several shapes and lines to calculate.

Bitmap images are processed ahead of time, thus they are faster to render on screen. (GameBuilder Inc, 2013) Vector graphics can also be seen as a style definition with its crisp and clean look (Li, 2015). And to create that look, graphics software like Adobe Illustrator and Photoshop allows the rasterization of a vector file, meaning that the created file is in form of bitmap. (PhotoBiz, 2013)

Regular 2D games use the technology of texture atlases called spritesheets, which are a collection of several individual images, sprites, in one file. By duplicating these repeatable sprites of all sizes, the environment can be created fast, easy, and require less memory space than for example a fully painted environment would require (See figure 6). (Joel Burgess & Purkeypile, 2013) Individual sprites are also easier to edit and manipulate in the game engine, which means for example bending or rotating a sprite of leaves to create interesting animations. (Lambert, 2013)

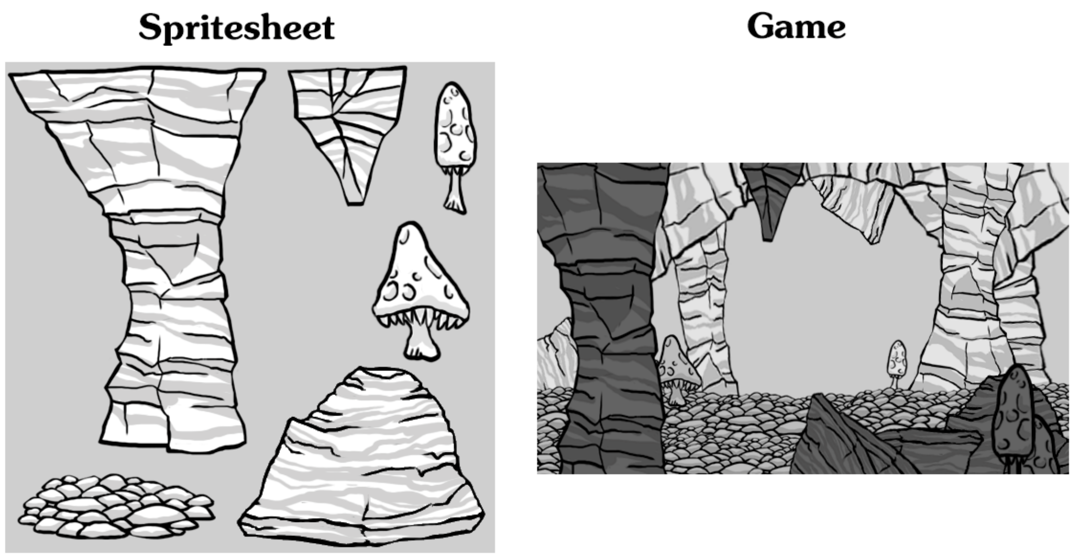


Figure 6. A regular usage of a spritesheet of different assets.

But why not use separated images in place of spritesheets? This is because the amount of individual images can grow easily from tens to hundreds, and rendering them individually eats up a lot of processing power. It is generally faster to display sprites from a single file. (Lambert, 2013)

3.2 Perspective

Actual cameras and eye rely on the perspective projection, which has a focal point and the objects in the scene recede to that point in the distance. This is not a problem with general 3D games, but with 2D games, which require shifting the camera, it presents certain problems. Perspective in 2D games is about making an illusion of depth, because the player is usually locked to 2 dimensions and the background is rarely done as an actual, realistic projection of three dimensions. (Koncewicz, 2009) For example traditional 2D side-scrollers are depended on the differences between background and foreground. (Lux, 2012)

3.2.1 Linear perspective and single screen games

Linear perspective can be seen as the real-life presentation of the view, in which it relies upon the horizon line, in short the viewer's eye level, and the objects in the view converge along the z-axis to a single vanishing point on the horizon line. This can also be called as a one-point perspective. Placing that one vanishing point in the dead centre of the view causes the environment's walls, roof and floor to be visible, like it is seen as a room (See figure 7). This creates the illusion of expansiveness, and can be used to creating single screen games. (Lux, 2012)

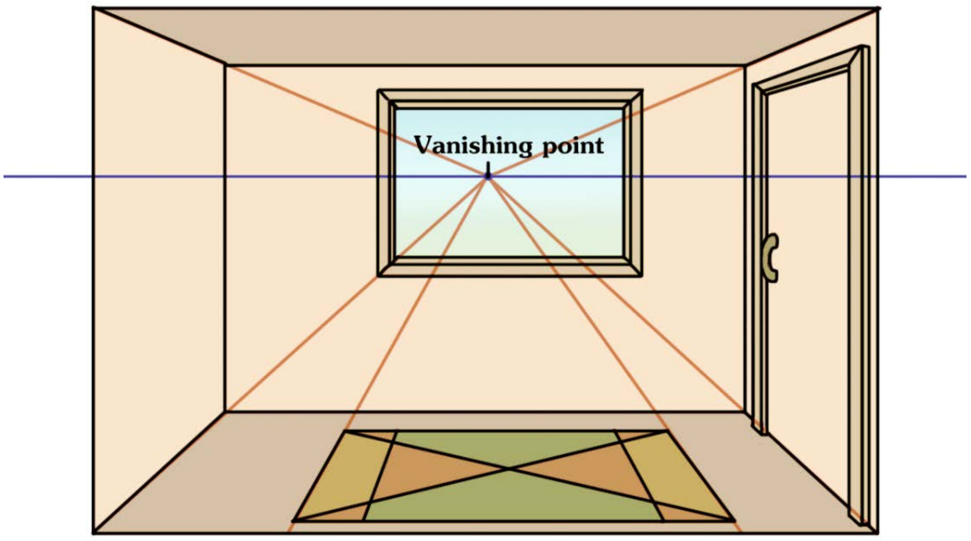


Figure 7. Linear perspective used for a room.

However, linear perspective is only useful for single screen games. This is because in reality, when the viewer changes place, so does the vanishing point of the view and thus the appearance of the objects. Because of this, the problem of making every possible appearance of an object in 2D games as an individual image is almost impossible. Single screen games' view shows only a segment of the entire space, and thus the objects in the view can made only once. The transition between the different segments is usually made with loading screens so that the change between different segments perspective would not be so obvious. (Lux, 2012) However, breaking the flow of the game by a loading screen can create a break in immersion, so it should be thought carefully. (Courrèges, 2015)

3.2.2 Axonometric projection

Axonometric projection is a technique for creating all three dimensional 2D images, meaning that it tries to show all the sides of an object. This can seem unnatural, as the eye does not function like that, and it can also present issues like elevation blending together unrealistically. There are several types of axonometric projections, which rotate an object's axes differently, and can be differentiated from values of the angles between axes and their symmetry. (Koncewicz, 2009) One of the axonometric projection's good perks is the non-existent vanishing point (Lux, 2012).

Isometric projection

In isometric projection, all the angles between axis is at 120 degrees, meaning that the axes scale is identical and for example a cube's surface areas have the same amount of used space in an image (See figure 8). This projection however is not useful for pixel style graphics. This is because of the 30 degree angled lines that are drawn for isometric projection. A line of 30 degrees has a width and height ratio of 2:1, but using this calculation to draw a pixel line does not result in a 30 degree angle. To overcome this result, pixel style graphics can use dimetric projection. (Koncewicz, 2009) For example Supergiant Game's role-playing game Bastion is in isometric view (See figure 9).

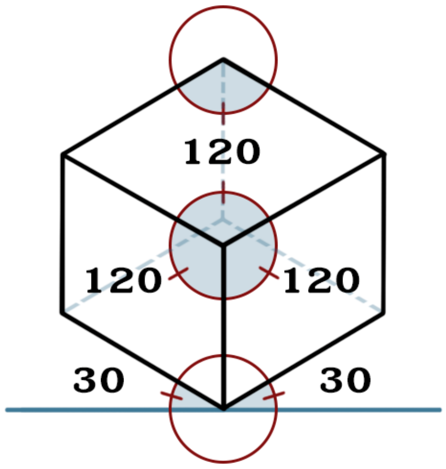


Figure 8. Isometric projection.

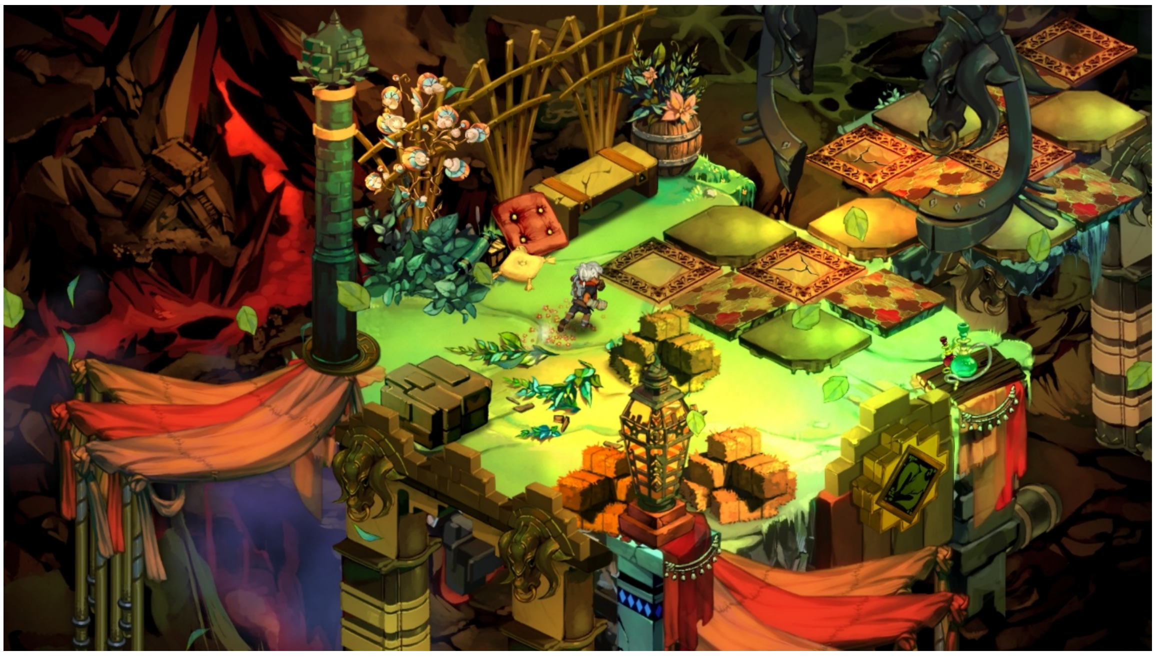


Figure 9. Screenshot of Bastion. (Supergiant Games, n.d.)

Dimetric projection

In dimetric projection, only two of the axes' angles are identical, meaning that this projection can be done in multiple ways, because the values of the angles do not really matter. However, one of the most useful perks in this projection is the ability to make 2:1 height and width ratio, which is needed for the pixel style graphics, and the angle used for this is 26,6 degrees from the horizontal plane (See figure 10). (Koncewicz, 2009)

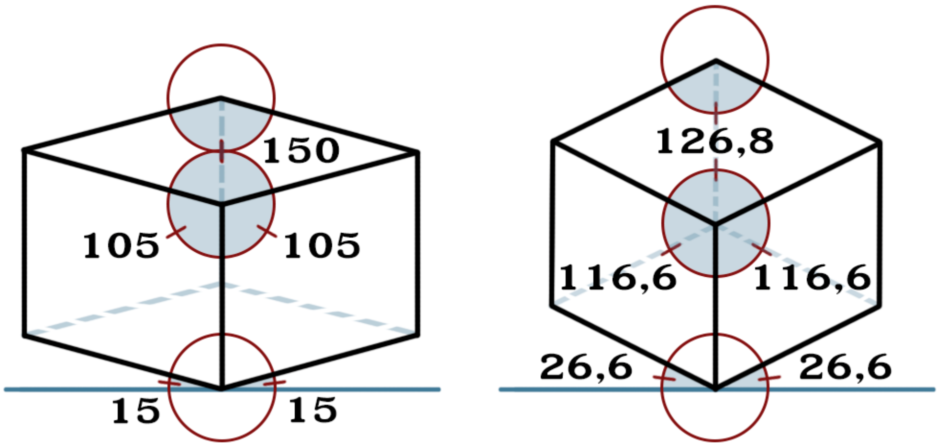


Figure 10. Dimetric projection and dimetric projection with 2:1 ratio.

Trimetric projection

Trimetric projection retains some symmetry in an object's surfaces, even though all the three axes are completely differently angled, resulting different foreshortening of the axes (See figure 11). Even though its roots are in the isometric and dimetric projections, it offers quite a different look. As an example, a city building game SimCity 4 uses trimetric projection (See figure 12). (Koncewicz, 2009)

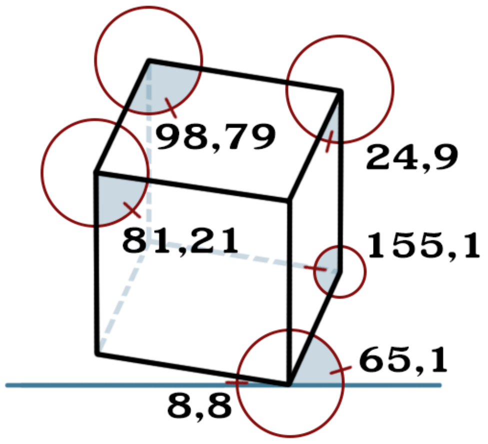


Figure 11. Trimetric projection.

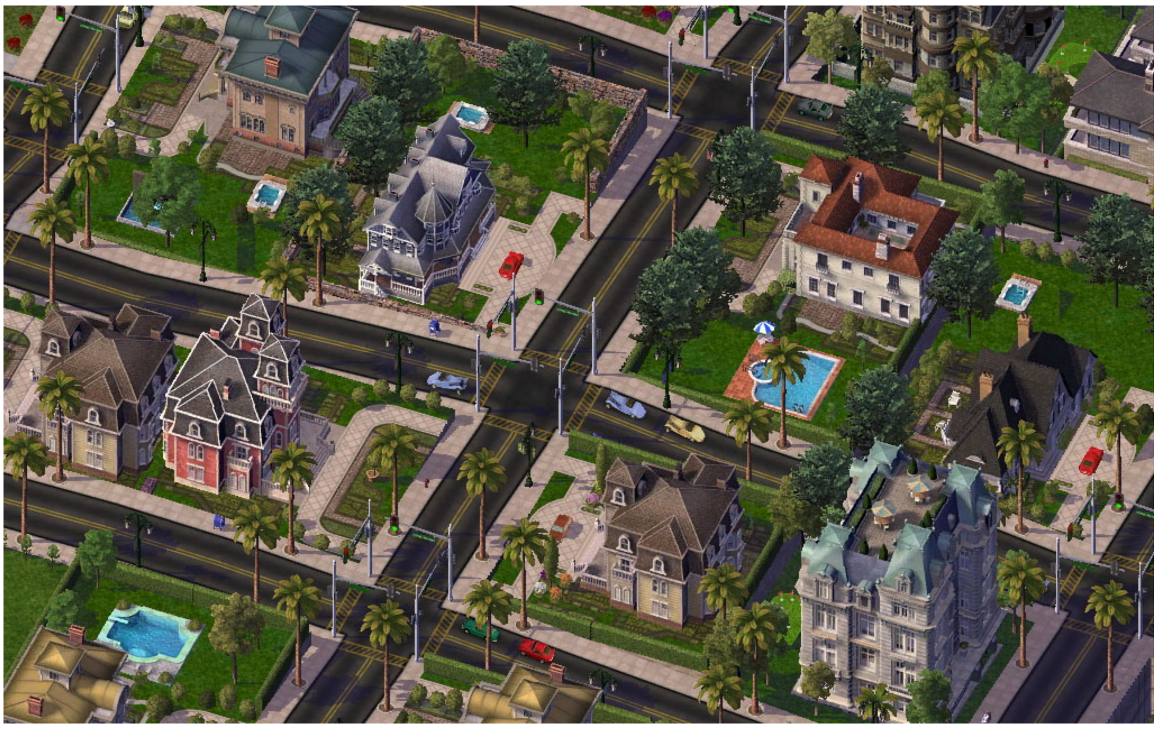


Figure 12. SimCity 4 Deluxe screenshot. (Electronic Arts Inc. 2011)

Oblique projection

In oblique projection, the surface area to which the camera is mainly projected creates an image to the screen of that surface with 90 degrees corners (See figure 14). This however creates awkward looking scale and angle proportions, and the object usually seems distorted, for example a sphere will look oval. Because of this, the oblique projection is not generally used for games. (Koncewicz, 2009)

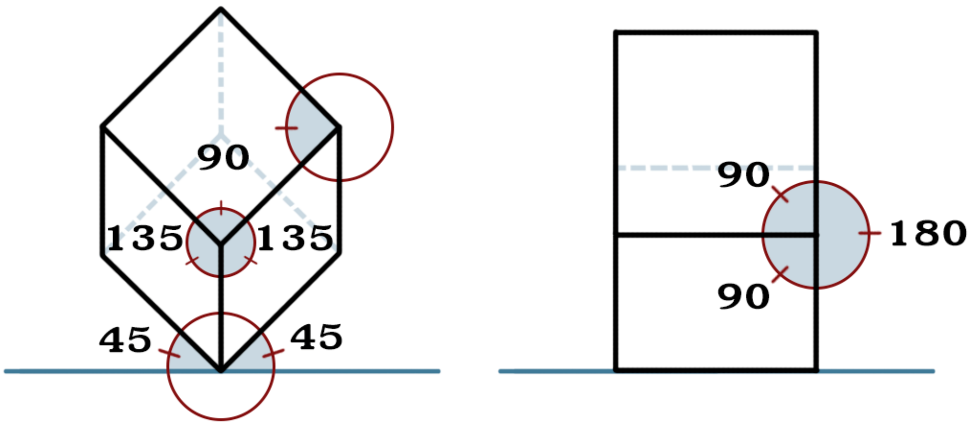


Figure 13. Oblique projection.

Cabinet projection is a more used sub-type of oblique projection that uses a fairly easy ruleset. One side of a cube is drawn as it would be seen straight from the front, and the other sides are extended from it, usually at an angle of 45 degrees (See figure 15). For a better sense of depth, the length of the other sides are also decreased to half. (Koncewicz, 2009)

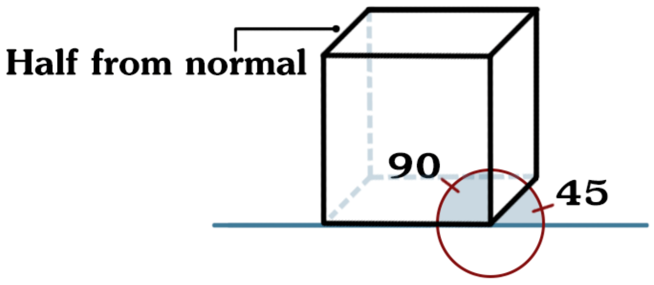


Figure 14. Cabinet projection.

3.2.3 Orthographic projection

In orthographic projection, the camera can be locked to three views: side view, top-down view and third person view that can also be called as a bird's eye view, and it does not have a vanishing point. For platforming side-scrolling games like Super Mario Bros, the orthographic projection is widely used because of their certain set of rules that are easy to follow. However, due to visual reasons, most of the 2D games do not display the orthographic projection correctly. This is because in orthographic projection all the axes have a consistent relationship between each other, so no receding of objects happen because there is not basically any horizon line where to recede (See figure 13). In practise, for example in Super Mario Bros, this would be seen as extremely large clouds in the view because there is basically no distance between foreground and background. Also the axes are foreshortened equally, meaning that in top-down view pyramid-shapes can look a little awkward. So to add more depth, the objects are scaled smaller as their imagined distance increases. (Koncewicz, 2009)

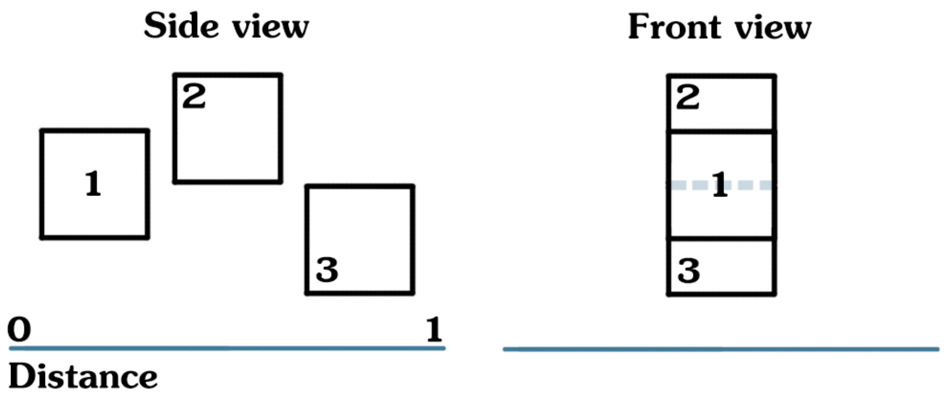


Figure 15. Orthographic projection with identical boxes. (Author's note: numbers do not identify the orientation of the boxes, just their placement.)

3.2.4 2.5D: The experience of two dimensionality

The term of 2.5D is usually associated with games, which have 3D graphics, but the gameplay itself is restricted to two dimensions or otherwise fixed-perspective view (See figure 16). By limiting the player’s field of view and play area, the designers have a greater set of controls when creating visually good looking environments. 2.5D also reduces the needed amount of processing power when compared to regular 3D games that have a free view. (Galamoth, 2015)

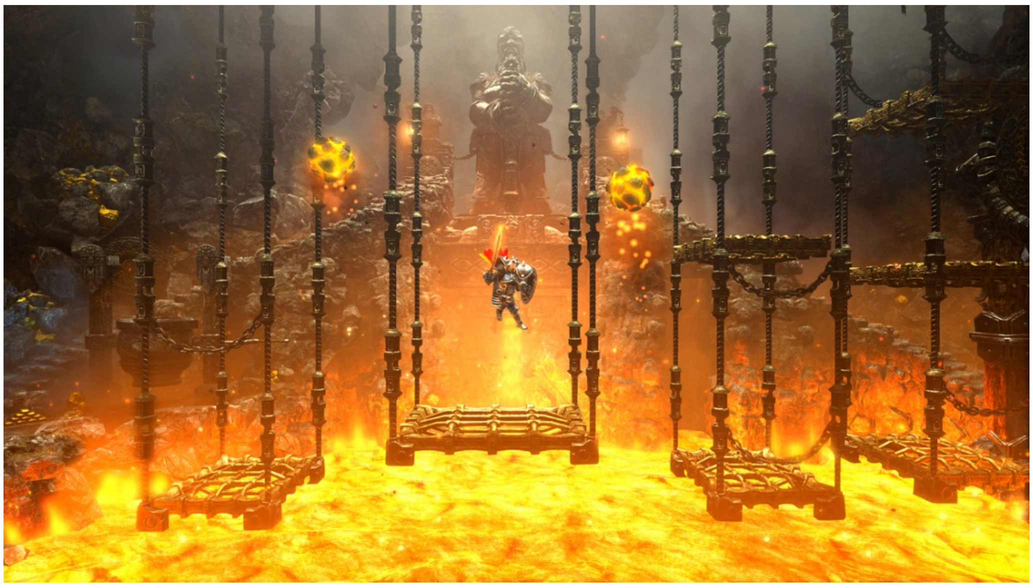


Figure 16. 2.5D in Trine 2 screenshot. (Frozenbyte Inc. 2011)

3.3 Creating the illusion of depth

In 2D games, it is important to have contrast between the foreground and background to give a better illusion of depth. In order to compensate the limits of two dimensional assets and locked camera, there are more tricks for creating depth than just varying the size and placement of the assets related to the distance of the player’s character. (Lux, 2012) Some of the most general techniques are used on the layers of the foreground and background, such as overlapping and parallax scrolling. Parallax scrolling illustrates depth nicely in 2D-games with different speeds of motion between the layers of foreground and background. (Gamesradar_US, 2010) In practise, the foreground scrolls faster than the background, creating different levels of focus. (Forman, 2001) Overlapping basically means that the graphical elements, such as sprites, are overlapped on top of each other, creating an illusion that the overlapping part is closer to the viewer (See figure 17). (Lovato, 2015)

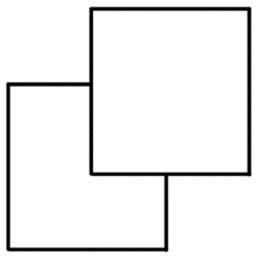


Figure 17. A simple example of overlapping elements.

3.3.1 Focus and detail

Details are affected by an object's importance to the gameplay, which means that the non-interactive background is the last in line to get the details (Lux, 2014c). An object's details can also be seen as a noise that draws the viewer's eye to it. When the whole 2D background has an equal amount of details everywhere in the view, it will be hard to read, which means focusing on the elements that matter the most gets harder. (Hawkins, 2014, p.50) By removing unnecessary details, and focusing on the biggest factors such as composition, silhouette, etc. the background is easier to read and does not distract the viewer anymore. (Masters, 2014j)

The viewer is only able to focus his eye on one object or an element at a time, meaning that the surrounding objects get blurred. The capabilities of viewer's eyes are also limited at perceiving details that are further in the distance. Great amount of light and shadow also decreases the visibility of details of an object. By replicating these natural effects of the eye, the 2D backgrounds can easily get more depth. The visual phenomena of focus can also be applied to 2D graphics that has outlines. When an object is further away in the distance, the width of the outlines also gets smaller. Also the object's importance to the gameplay can be signified by a thicker outline (See figure 18). (Lux, 2014c)

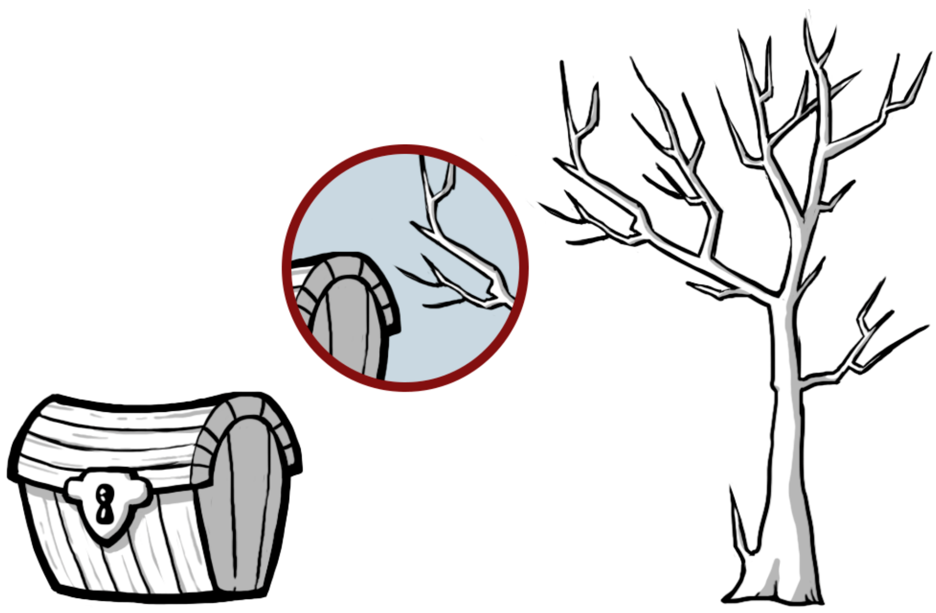


Figure 18. Differences in line width.

3.3.2 Atmospheric perspective

Objects in the distance are hazy in details and shifted in colour not only by the limitations of the eye, but also because the air and colour of the sky affects them. (Jansson, 2012) The air surrounding the objects is filled with small particles such as dust, water, smoke, pollution, etc. which affects the light's scattering in the air (Riverman Media, n.d.). The more distance there is between the viewer and the object, more there are particles obscuring the light's path and affecting the view. In the figure 19, the background seems really blue in contrast to foreground's more vibrant colour. (Lovato, 2015)

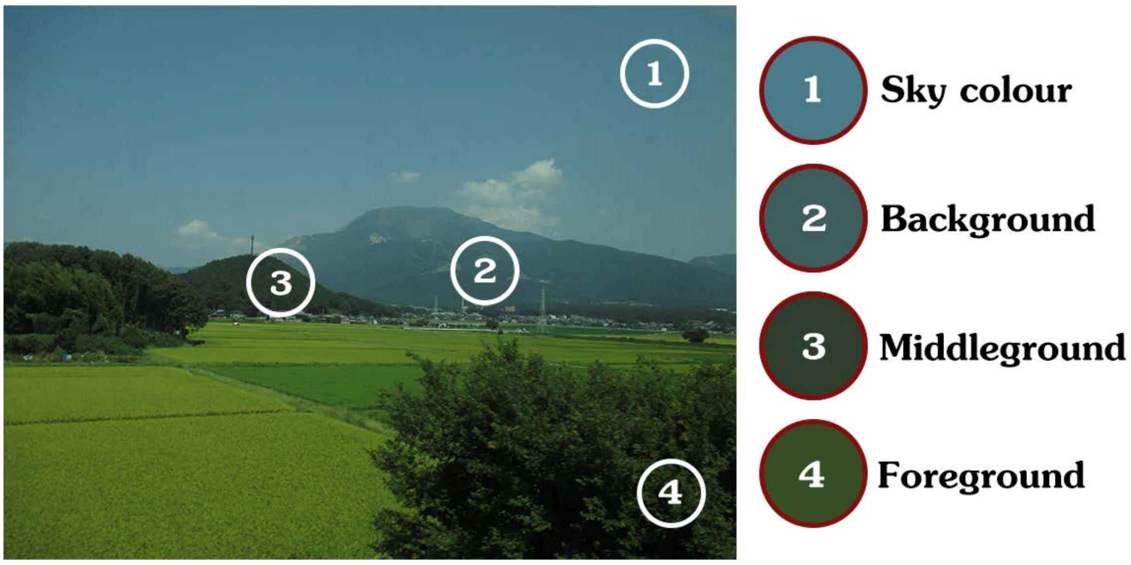


Figure 19. Atmospheric perspective.

3.3.3 Colour theory and contrast

Traditional colour theory focuses on the colour wheel and the colour relationships to one another. Both of them are useful when choosing the colour composition, but also the colour proportions and distributions are as important when making 2D backgrounds or 3D environments (See figure 20). (Lux, 2014a) The use of limited colour palettes that have a small set of strong saturated colours and a larger set of weaker colours, will make unified, harmonious looking environments. Wide range of different colours tends to only confuse the viewer, taking away the immersion of space. (Gurney, 2010, pp.104-105)

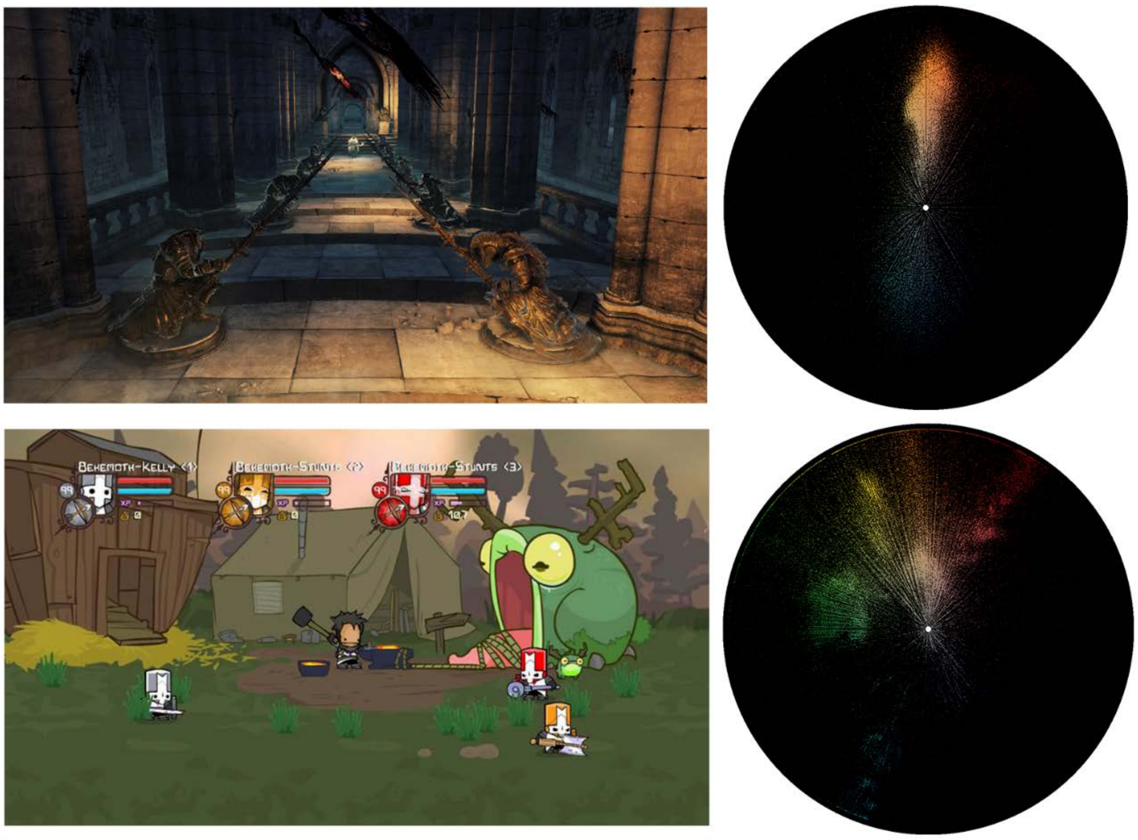


Figure 20. Comparison of two sets of colour palettes: Upper screenshot is from Dark Souls 2 role-playing game, and the bottom one is from Castle Crashers side-scrolling game. The colour palettes were calculated with Joost van Dongen's Colour Wheel Visualizer. (Dark Souls 2 by From Software, 2015; Castle Crashers by Behemoth, 2003; Colour Wheel Visualizer by Van Dongen, 2015)

Understanding the colour theory well is extremely important when making 2D backgrounds, because the properties of colour: hue, saturation and value affect each other in very different ways. There should also be a large contrast in colour properties between the player's character and the environment, so that the character would stand out. To test the contrast, it usually means placing the character in different environments that have varied lighting and colour. (Lux, 2014a, 2014c)

Value, how bright or dark an element is, can be seen as the most important of the colour properties, because it affects the perceived depth the most with lighting. A good example of a 2D game which used only values for depth would be Limbo, which uses black and white values and blurring of the objects extremely well (See figure 21). (Lux, 2014a)

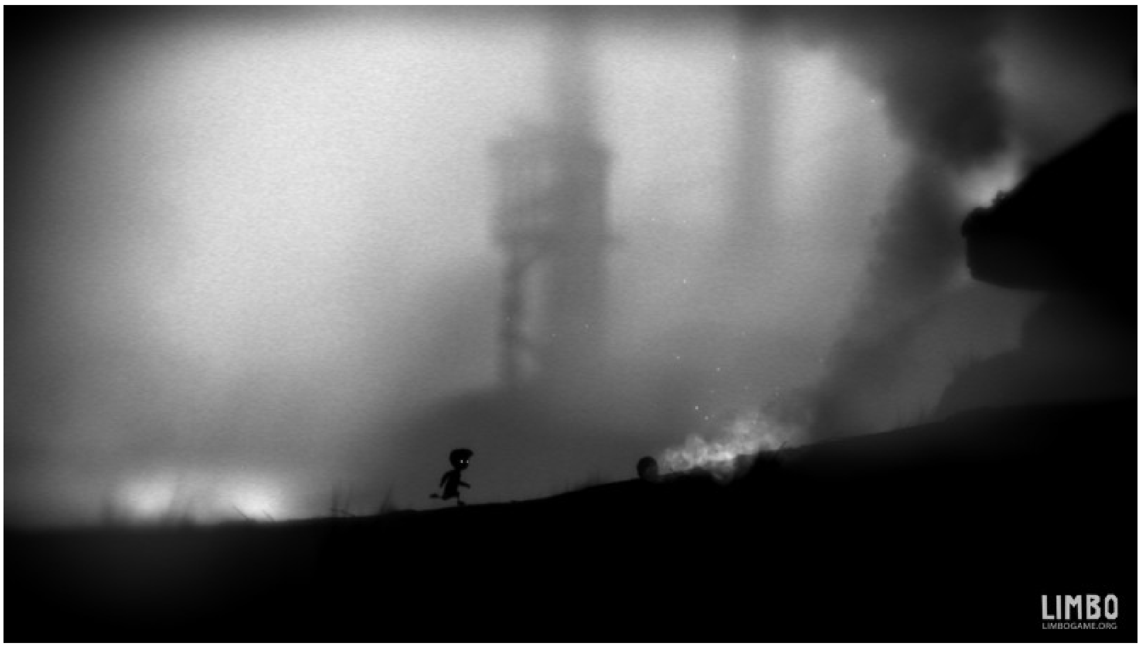


Figure 21. Screenshot of Limbo. (Playdead, 2014)

Hue means an object’s differences in colours. An object’s colour can be affected by internal or external sources, such as blood in the cheeks making them red. Two differently coloured objects reflect colours to each other, such as sky reflecting blue on the surfaces that point up. Using only the same hue with objects makes them look dull (See figure 22). (Jansson, 2012)

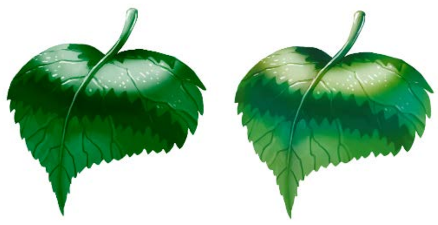


Figure 22. Comparison of the same object that have different hues: Left object uses only green hue, whereas the right object uses also blue and yellow with the original green.

Saturation calculates how much colour an object has. For example greys are desaturated, meaning that there is no colour. (Jansson, 2012) Viewer’s eyes are naturally drawn to bright, saturated colours, which is why the less important backgrounds should be more desaturated, even in games that are generally colourful (See figure 23). (Riverman Media, n.d.)

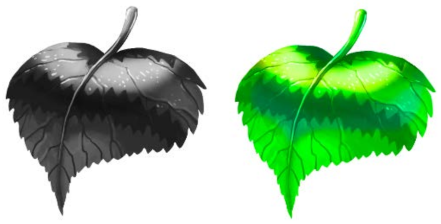


Figure 23. Desaturated and extremely saturated object.

Shadow and light affect the saturation a lot, because for example in very light or dark areas the colours can appear almost white or black. However, the shadows' or lights' colour should not be just added black or white, because it will make the object look lifeless (See figure 24). (Lux, 2014b) Thin objects that have certain materials, such as cloth and leaves, are subject to light penetrating them, which creates an effect called sub-surface scattering. The light bounces in the material, and exits from it, which will affect the saturation of the object making it look more saturated and illuminated from the exit side. (Jansson, 2012)

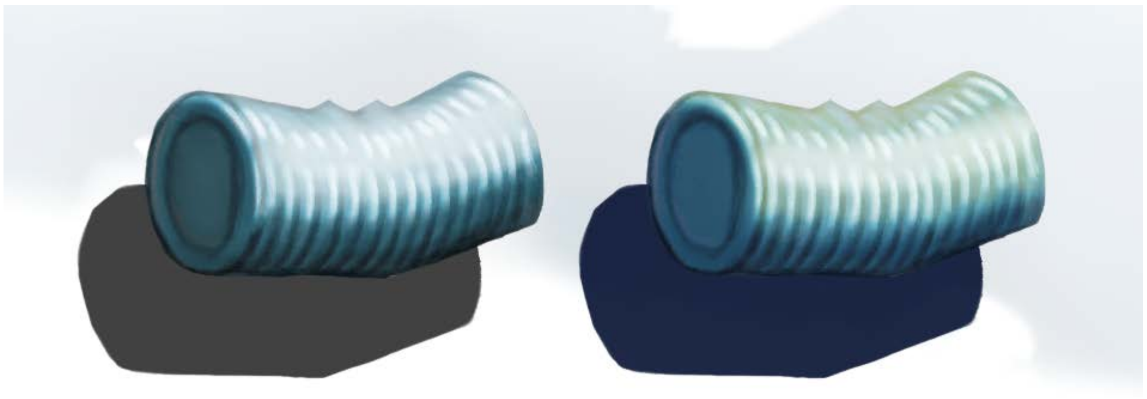


Figure 24. Comparison of light and shadow colours.

4 BASICS OF 3D GAME GRAPHICS

3D Game models vary from cinematic models both in polygon count and texture resolution. In cinema, rendering a single frame for a movie can take from several minutes to several hours because of the millions of polygons and other details. In games rendering needs to be done in real-time, meaning there is only a fraction of a second for the computer to render a single frame. (Cheng, 2014a) Because of this, game models and their associated textures are much smaller, so it is best to make and keep details where the player will notice them first. (Masters, 2014d)

4.1 Modelling and sculpting

Modelling is the action of creating model in a virtual three dimensional space with a 3D-modelling software. Models are created from polygons, multi-sided planes that are connected to each other to create a form. (Kennedy, 2013, p.14, p.28) A simple 3D-object, for example a cube, is comprised of several planes, edges and vertices (See figure 25). Planes are the polygonal surfaces, edges are simply edges of the planes, and vertices are the ends of the edges. (Mayden, 2015)

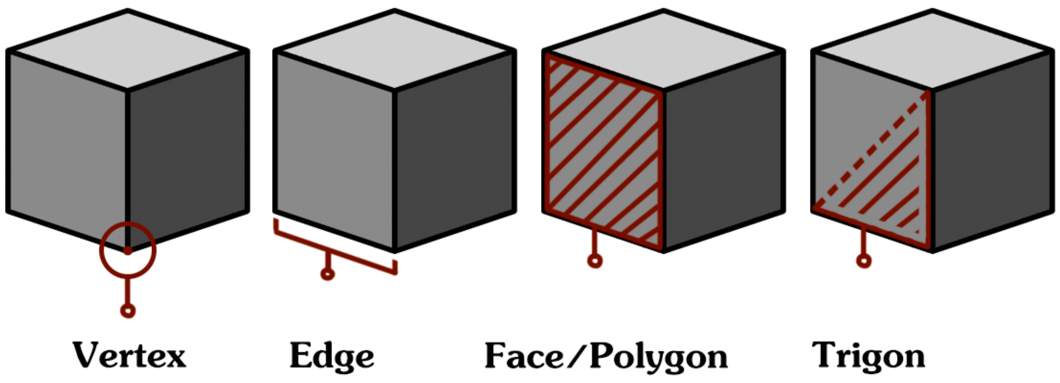


Figure 25. Properties of a 3D-cube.

Polygons should usually be made of four sided planes, which can also be called as quads. This is because quads ensure clean topology with looping geometry, to which animation can be applied easily (See figure 26). Quad loops are also generally easier to modify when modelling and are more predictable when sculpting.

Triangles are the simplest polygons that can be created, but should be generally avoided as they create issues in for example animations and while sub-dividing polygons for sculpting. An artist can not avoid triangles in organic shapes, so it is important to hide them or put them to places where there are a minimal amount of deformation because of animations. (Mayden, 2015)

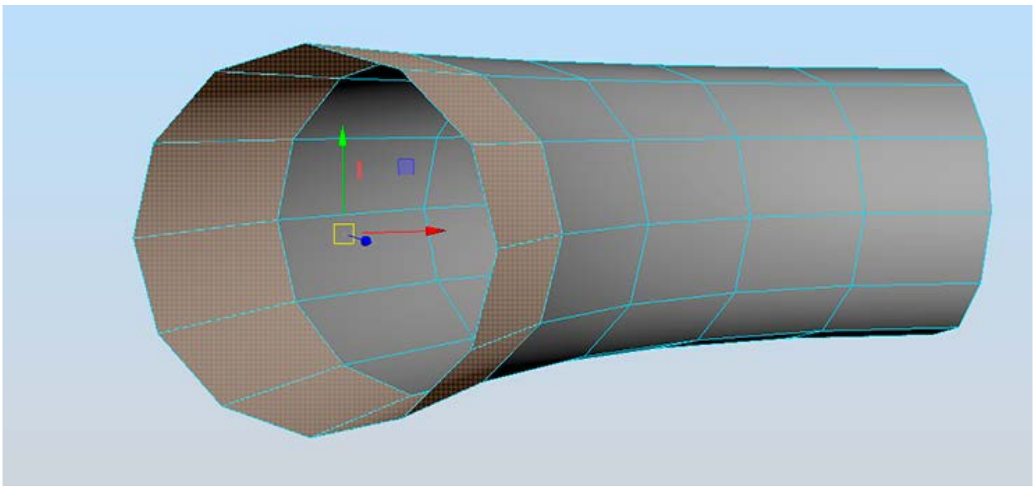


Figure 26. A simple quad loop.

N-gons, which are polygons with five or more sides, should be avoided at all costs. When importing a ready 3D-mesh in to a game engine, the engine will automatically triangulate the polygons, which can cause unwanted looking shading and animations. This is because the n-gon has so many options for triangulation (See figure 27). (Mayden, 2015)

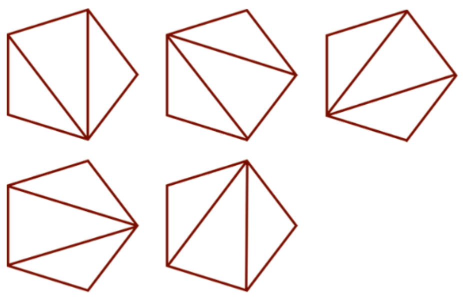


Figure 27. Possible triangulations of a 5-sided n-gon.

A 3D-model starts usually as a primitive shape and goes through several stages of refinement (Solarski, 2012). There are many different modelling techniques, for example box modelling, image-based modelling, edge modelling (See figure 28), digital sculpting, etc. Even a classical art technique called silhouette paper cutting can be used to model an object by cutting a 3D-plane based by a good side view image (See figure 29). (Cheng, 2015b) Geometry's silhouette, meaning outer edges of an object, should also have most of the object's polygons, because it will give an impression of a highly detailed object (McGrath, 2008).

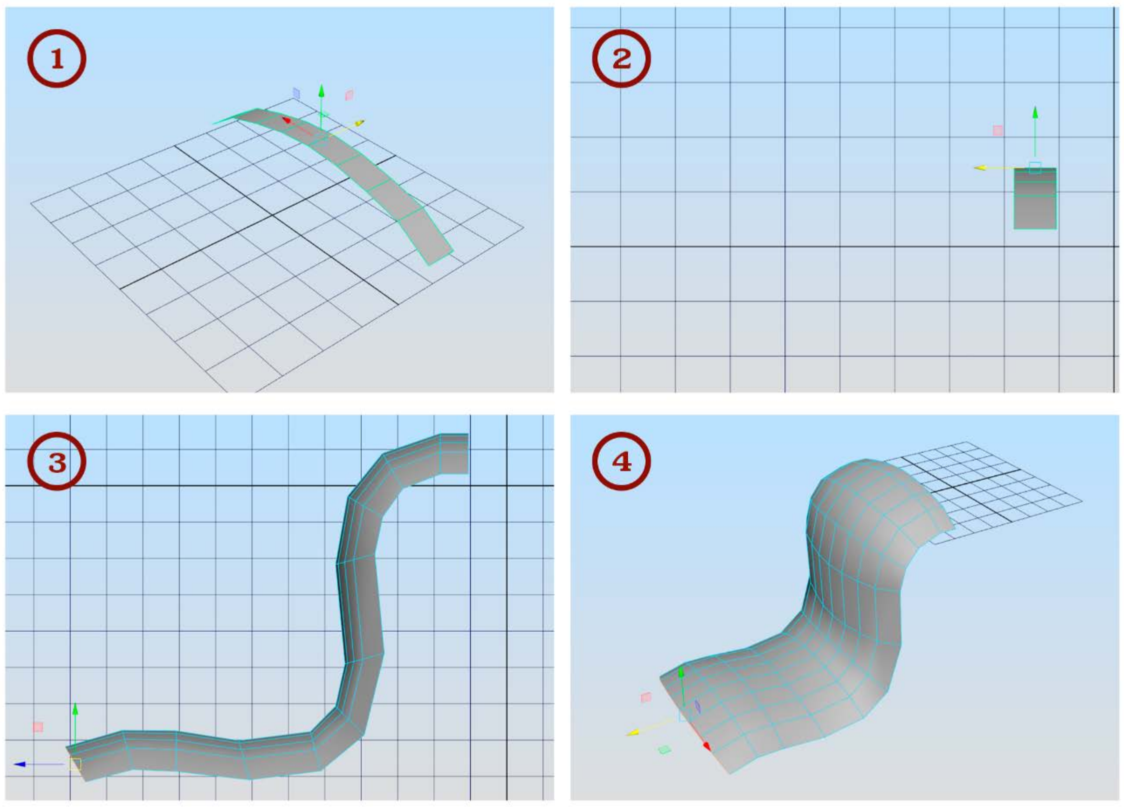


Figure 28. Edge modelling: Extruding polygon's edges in the side view (phases 2 and 3.) to create shapes in a short period of time.

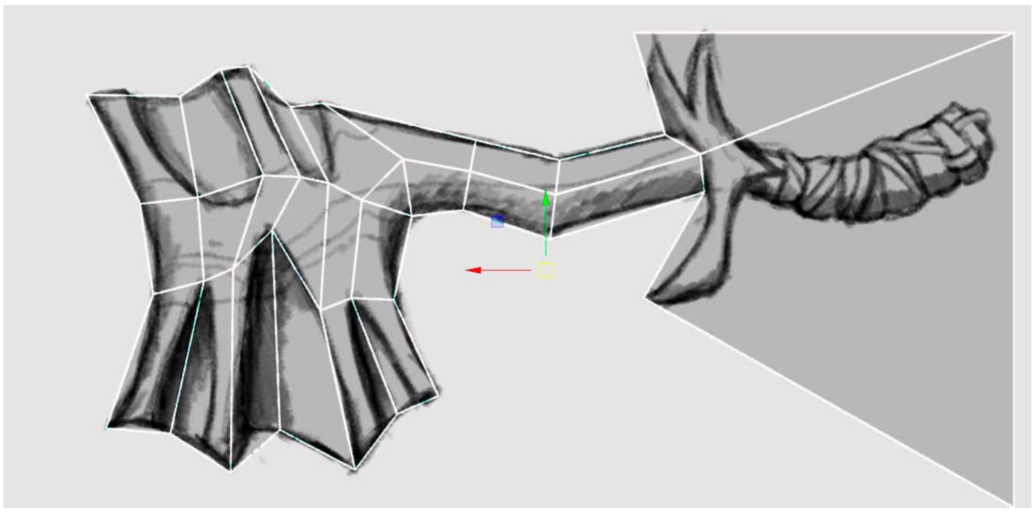


Figure 29. Silhouette cutting.

The specifications of a model describe its use in the game, and allow the artist to better see on what details he should spend more time and what not. This way the artist does not waste time on modelling unwanted details. (Hawkins, 2012, p.106) Not every aspect of the object has to be fully detailed, because textures can do a lot of faking (Cheng, 2014a). Some smallest polygonal details might even cause visual artefacts when viewed from far (McGrath, 2008). Things that will be in the dark or really far away can have less geometry, and surfaces that the player will not see in the game at all can be deleted (See figure 30) (Masters, 2014d).

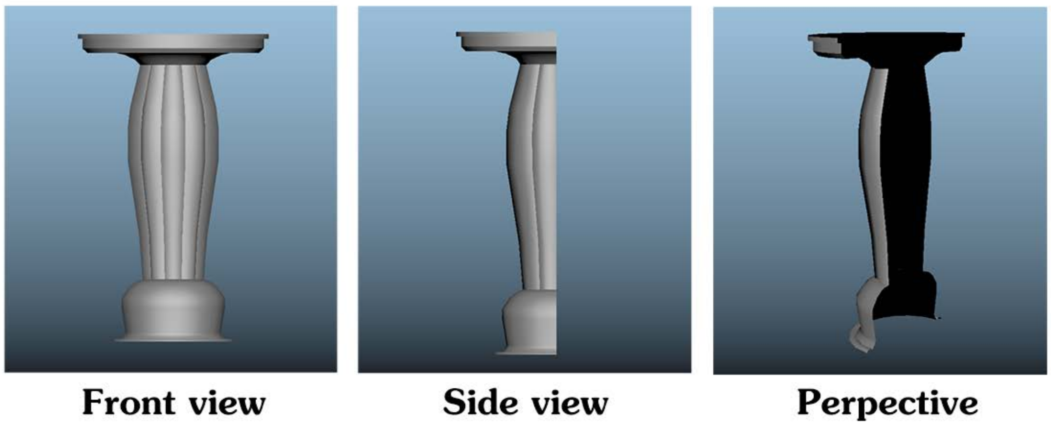


Figure 30. A 3D-model of a pillar which bottom, top and back sides are supposed to be touching floor, roof and wall.

In probable need of lower resolution models, it is also best to model objects' quad loops using multiples of four, because this way every second edge loop can be easily deleted without further thinking or reworking the model from scratch (See figure 31) (Kuzminova, 2009). This rule of multiples also applies to symmetrical and cylindrical objects that need to be easily combined together in the game editor. For example a seven-sided pipe will not combine seamlessly with another identical pipe, if the original pipe is rotated 90 degrees (See figure 32). (Perry, 2002)

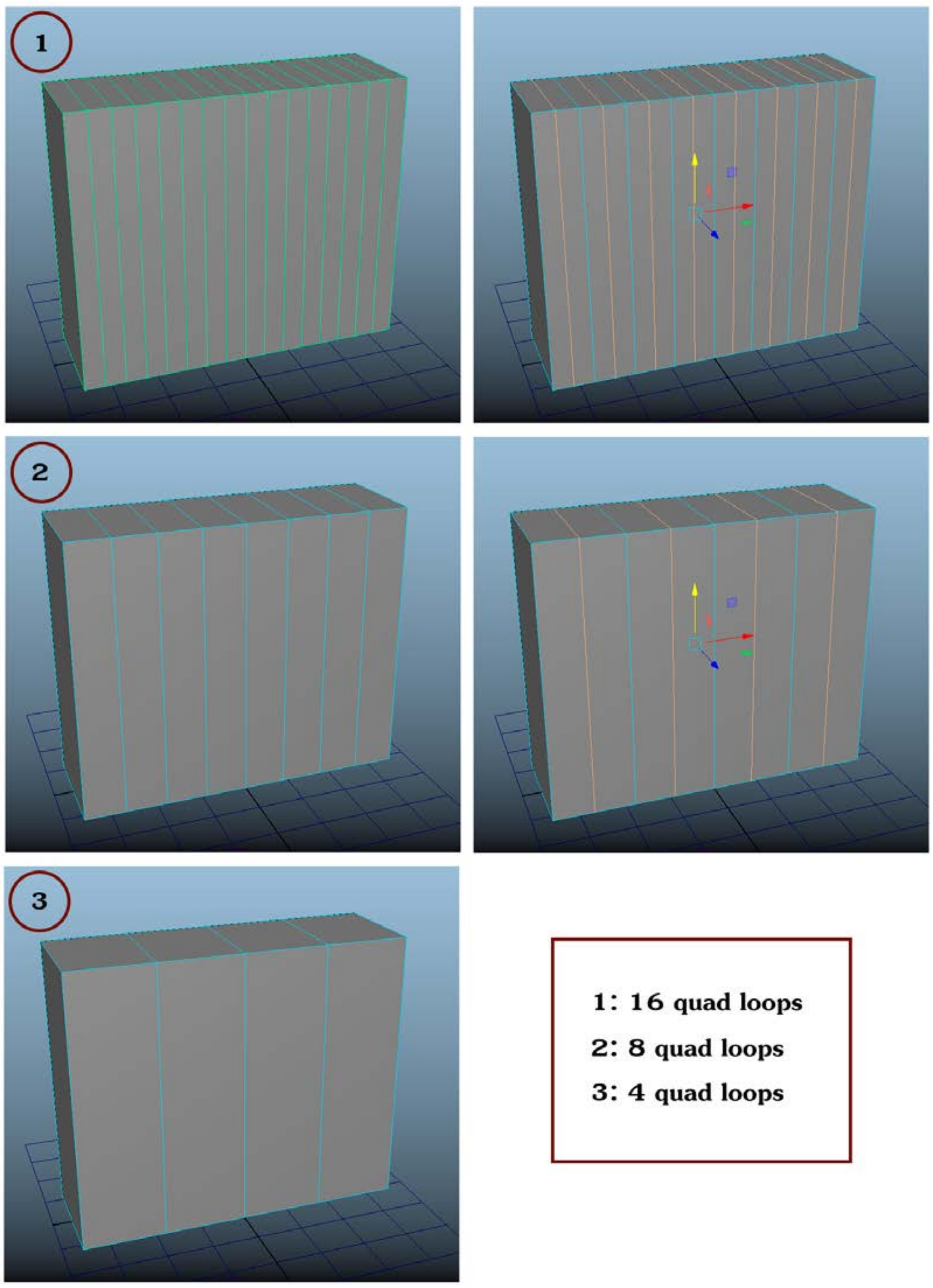


Figure 31. Geometry with multiples of four.

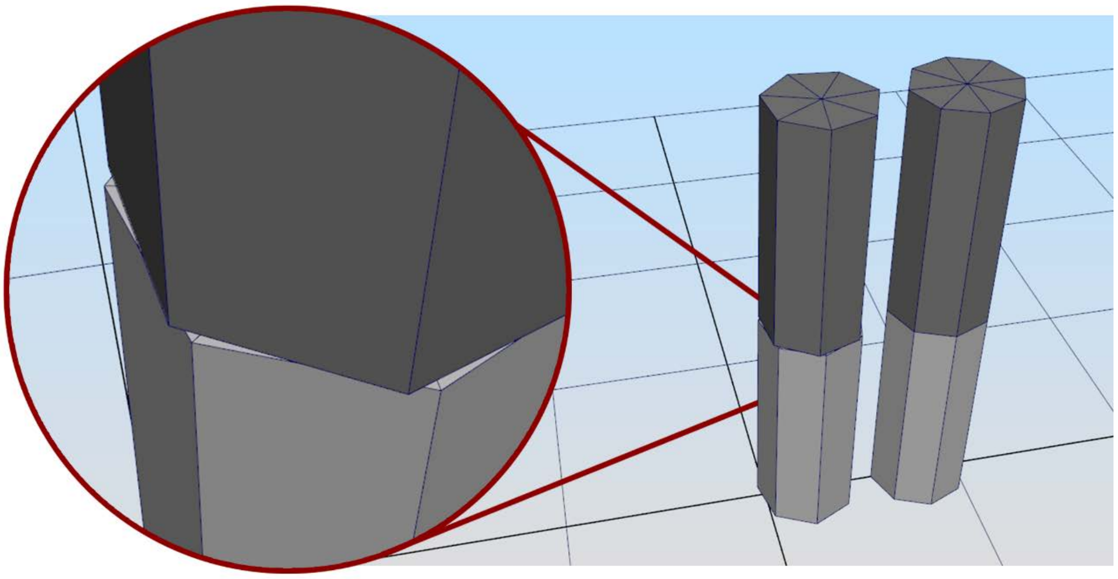


Figure 32. A seven sided cylinder and eight sided cylinder rotated 90 degrees: the seven sided cylinder does not snap to the bottom cylinder correctly.

Sculpting is creating and manipulating a 3D-object the way an artist would with clay or other real world sculpting materials. A regular 3D-modelling software can support about hundreds of thousands of polygons, whereas a sculpting software can support millions of them. (Sam. R. Kennedy, 2013, p.29) Sculpting is especially useful for making small and organic details, because modelling them in a traditional way simply takes too much time (Masters, 2014b).

Sculpting building blocks like beams, planks, tiles, etc. which have unique sides, will create a tool kit that can be used to assemble all kinds of structural pieces like walls, pillars, etc. It is pretty familiar to going to a hardware store and assembling the objects by hand. Creating the building blocks sides unique gives more variety in the final asset’s surface and saves production time. It can also create better results than fully sculpting the object from scratch, for example a wall that has a large visible surface. (Almost Human Ltd, 2014a)

To create a lower resolution model of the sculpted object, which could be put in to the game, it goes through a process called retopology. Retopology can be automatic, which means a software calculates all the new polygons' placements, but can present issues like bad topology and wasted polygons. It can also be manual, which means placing vertices on top of the sculpted model's surface, which then create the new polygons. Even though manual retopology requires more time to do, it gives more control for the artist as he can decide how optimized the new model will be. (Marshall, 2014b)

4.2 Normal direction and smooth shaded foliage

There is a vector called normal for every vertex in the model, which can be visualized as a ray that uses the same orientation as the surface of an object (See figure 33). The normal's direction can be manipulated in a 3D-modelling program, which then affects the shading, in short how the light is applied on the surface of an object in the game engine. (Light, 2015) 3D-modelling programs group these vertex normals under the name of hard or smooth edges and smoothing groups to separate polygons' shades from each other or uniformly shade them together (See figure 34). (Backus, 2013)

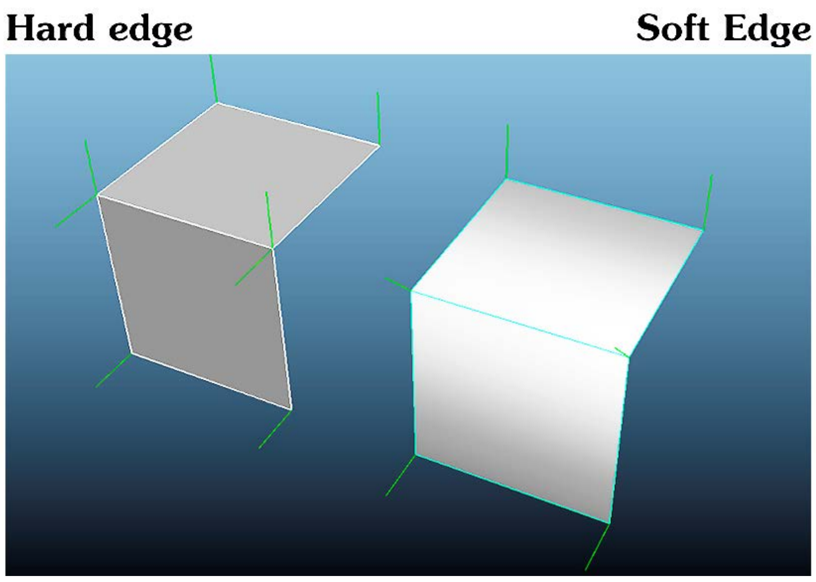


Figure 33. Normals' directions in identical 3D-objects.

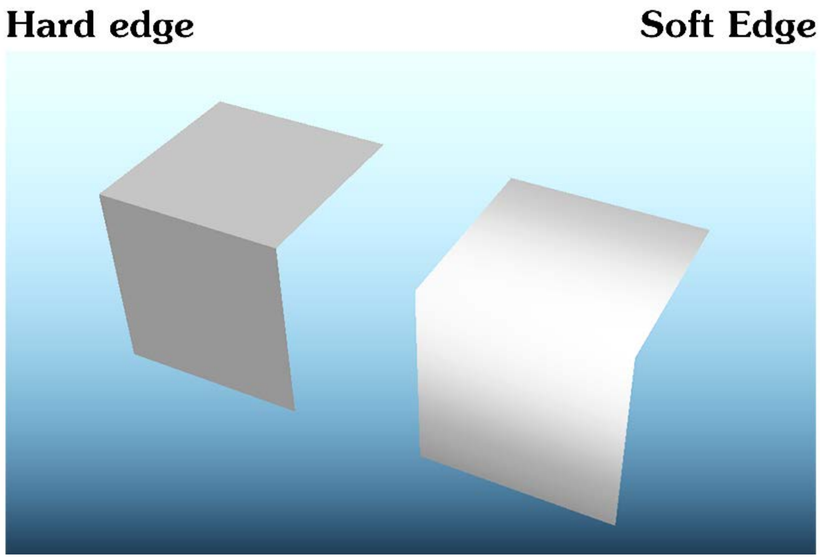


Figure 34. Hard edge shading versus smooth edge shading.

Trees' leaves and other foliage are usually composed of several individual planes. If the planes are not greatly edited, the normals' orientation of these planes faces are at the default 90 degree from the surface (See figure 35). This will make the whole object look unnaturally shaded, as shown in the first screenshot in the Figure 36. By editing the normals' orientation to the object's centre, it will create a better transition in the shading of the planes. The smoother transition can be achieved in many ways, which can differ between 3D-modelling programs. For trees and bushes it is best to use a sphere as a source for the normal direction, and for grass and shrubs a half sphere (See figure 36). Because of the source, normals' directions are now pointing outward of the object. (Light, 2015)

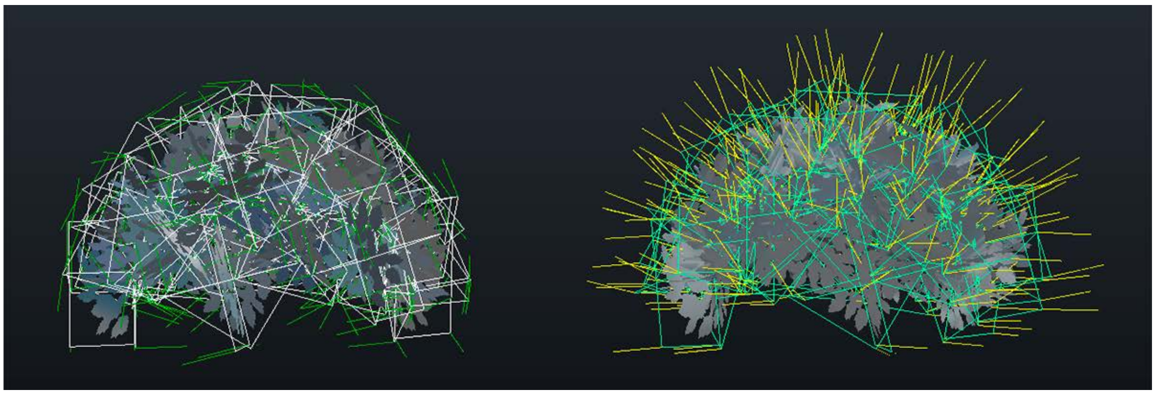


Figure 35. Not edited bush and normal manipulated bush: The left bush's normals are coloured green and the right bush's normals are coloured yellow.

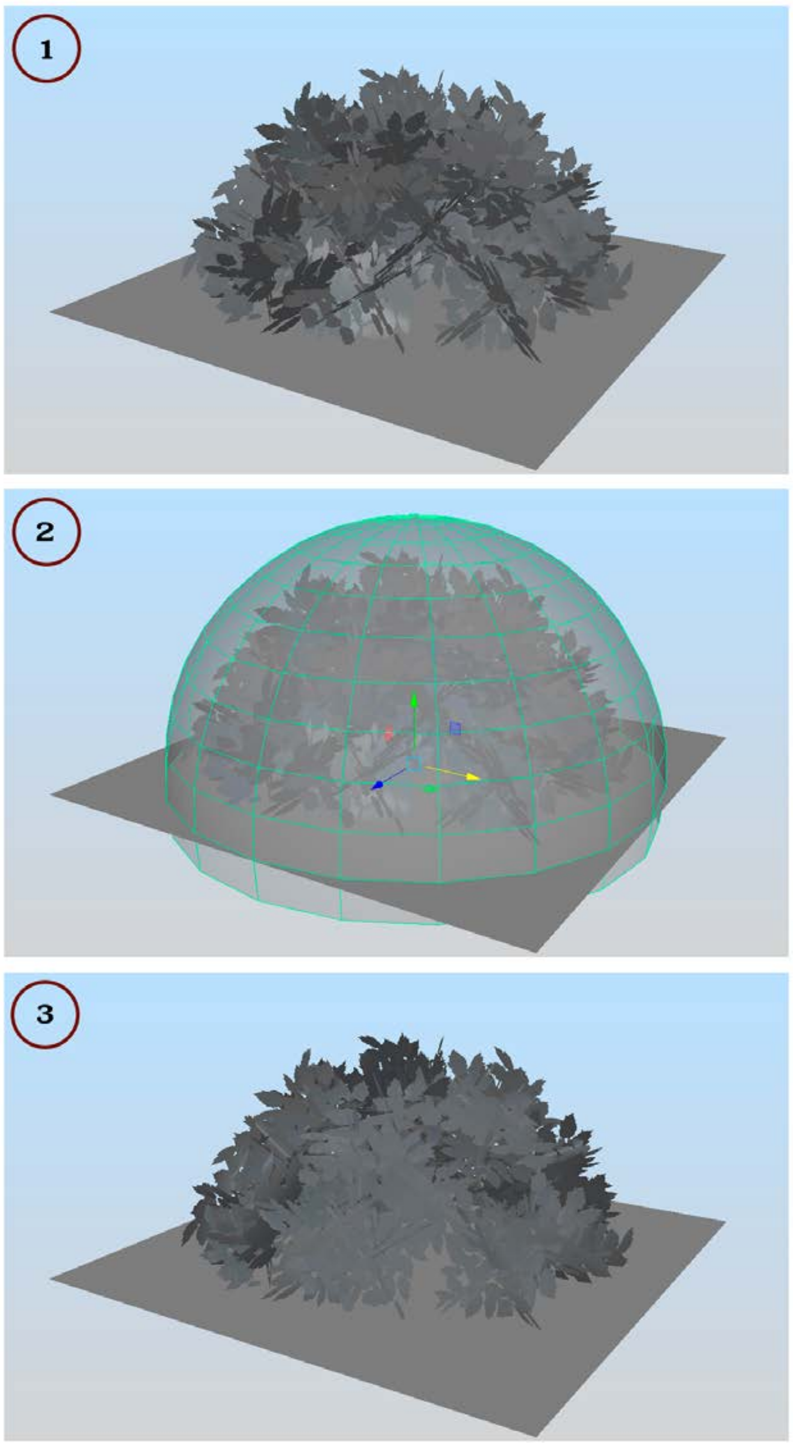


Figure 36. Using a primitive shape as source for transitioning the source’s normal directions to a shrub.

4.3 UV-mapping

UV-mapping, also named as unwrapping, helps the computer to understand to which parts of an object the texture is applied to. The idea of it is to flat out the 3D-object's polygons to a two dimensional surface, so that the placing or painting a 2D texture would be easier. Each polygon of the object are represented as faces in a UV-map, which then can be taken as a simple screenshot, or in another name as a snapshot, and moved to a painting software, after which the texture is applied to the model to check how the texture works. (Kennedy, 2013, p.77)

The faces which are collected together in the UV-coordinates to create a larger, seamless shape, have several names between different software: UV shell, island or cluster. Most of the 3D-softwares have an automatic solution to create UV shells, which is really useful to start unwrapping complex 3D-objects. However, manual editing is usually needed later to get the best results. (Masters, 2014m)

However, UV-maps can not be fully seamless, and need proper adjusting for example by moving the seams to places where the player will not see them in the game. (Masters, 2014d) Stretching certain parts of UVs bigger is also possible, as it creates a visually more detailed object. Still, it should be avoided at places which have text or regular forms, as they will easily look distorted. It should also be avoided at places that have already been resized, as the hard differences in texture resolution can look too obvious. Non-uniform stretching can also cause issues with rendering, so it should be planned carefully at times when it is really necessary (Provost, 2003a). Cylindrical texture details are generally hard to make, because unwrapping an object with a curvy surface is hard to unwrap to look like the original, thus the cylindrical details can look pixelated when texture is applied to a 3D-object. (Kuzminova, 2009)

Saving UV space is important to make the most details visible. Texture atlases, where several objects' UV shells are located, are useful at maximizing the usage of a UV-space (Klafke, n.d.). Even though it is possible to mirror symmetrical UV shells together to save UV-space, not every game engine supports correct lighting of mirrored UV shells. (Kuzminova, 2009) Especially in development teams that have several artists, the UV shells should also be grouped together in an orderly manner as well as placed upright, because then the other artists can easily see how the UV-mapping is formed (See figure 37). It is generally a good rule to have as few UV shells as possible (Hawkins, 2012, p.120).

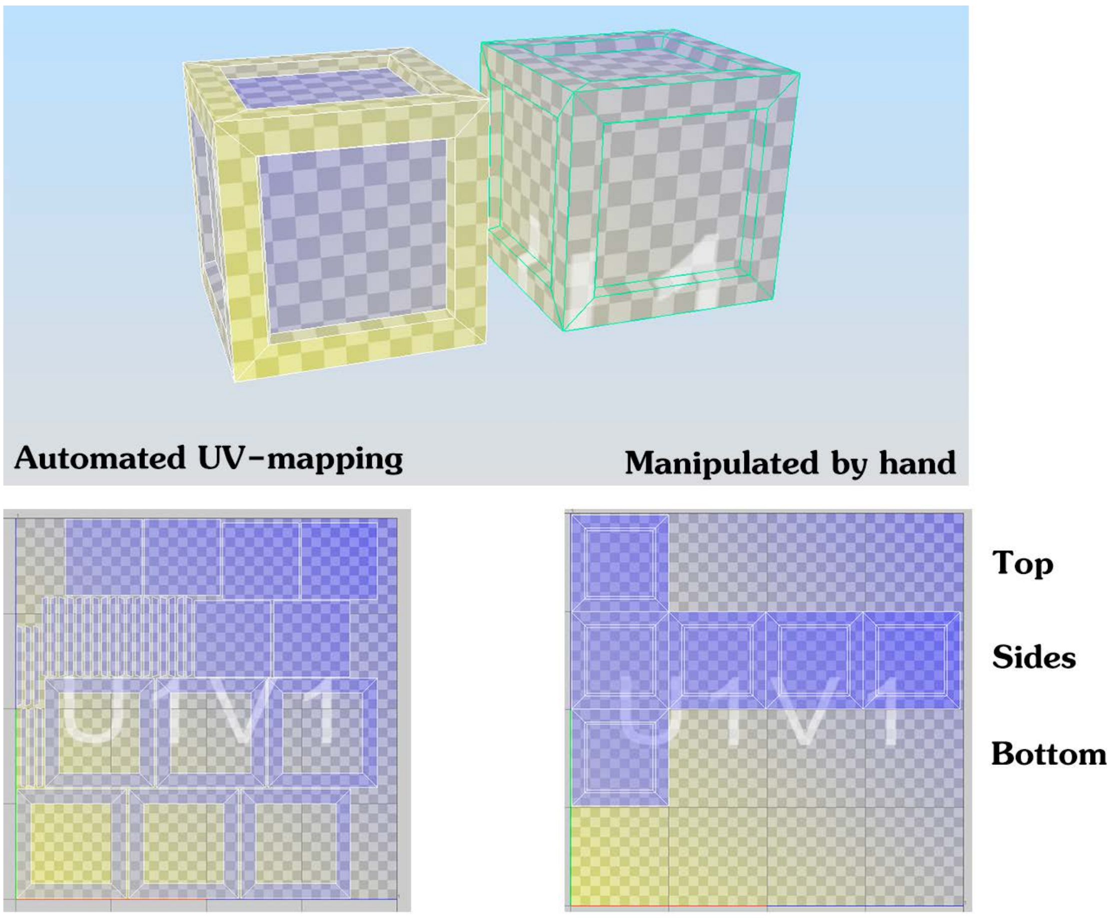


Figure 37. Comparison of two different UV-maps of the same box: There is not a large difference how much UV surface both the boxes use, but the right box's UV-map can be combined with additional UV-maps of other objects. The right box also has less UV-seams, which is also lightweight for the processor, which is further discussed in chapter 6. Optimization of assets.

If the game uses LODs, meaning level of detail assets, it probably also uses mip-maps, which are lower resolution textures that are changed in the model when the distance between the player and an object increases (Provost, 2003b). However there is a problem with mip-mapping that can be called as the "bleeding" effect. When an asset's texture is changed to lower resolution, the UV-map does not change and the pixels next to the UV-seams bleed to the UV shells, creating visible seams on the asset. This can be prevented by the use of edge padding, which spreads the colours outward from the UV seams (See figure 38). (Unity Technologies, n.d.a) To spread the colours nicely, the UV shells should have at least 4 to 10 pixels worth of space between each other and have similarly coloured UV shells placed together. (Kuzminova, 2009)

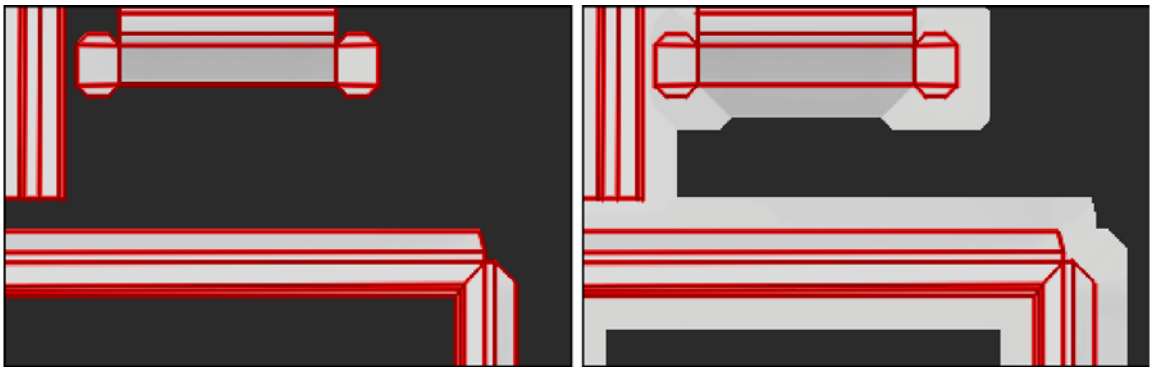


Figure 38. A close up of a texture without and with edge padding.

4.4 Baking

Baking is a process of taking the details of a high resolution asset and generating a texture map of those details for the low resolution asset's UV-map, so that the low-resolution model could appear more detailed than it really is (See figure 39) (Kennedy, 2013, p.29). Modelling and then sculpting a high resolution mesh is not always the best choice if there is a certain vertex budget for the game, but at least making one will ease the generation of nice looking textures and materials. (McGrath, 2008)

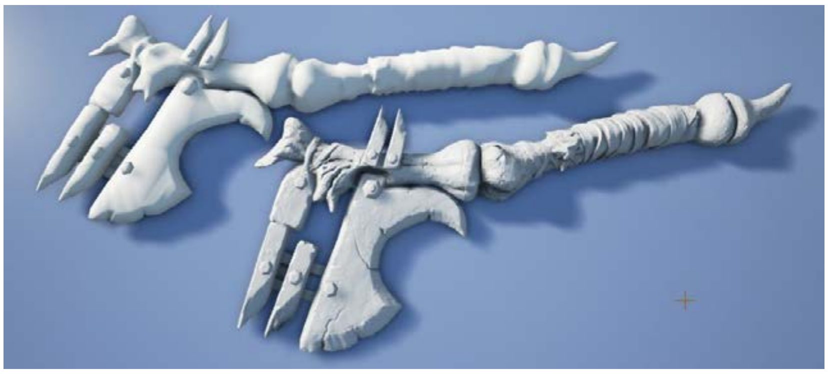


Figure 39. The same object without and with a baked normal map. (Epic Games Inc. 2015)

One of the most important maps for high detailed assets are normal maps that are used to create an illusion of depth without increasing the processing time (Masters, 2014m). They are usually baked, because they use a certain colour palette that is not easy to draw by hand (Trümpler, 2013). Normal maps are heavily used in physically based rendering systems, because they need a good normal map to simulate the lighting correctly. (Van Spronsen, 2014)

4.5 Texturing

Texturing is using a painting software to create a 2D picture based on the 3D-object's UV-map, which is then applied to the surface of a 3D-model. It can also mean creating more depth and details to a surface through a normal map, which fakes the lighting of bumps and dents on surfaces by remembering depth information from a similar but a higher polygon model (Kennedy, 2013, p.14).

Textures are also referred as texture maps, and can be used to tell the game engine a lot more than the surface's colour like reflectivity, roughness, transparency and so on. They can add a lot of detail to the surface of the model. (Kennedy, 2013, p.61) Textures must be able to hide low resolution geometry, and if done right, the low polygon model can only be seen when viewing it up close. (Masters, 2014g)

Transparency maps, also called as opacity maps, are used to give the illusion of separation in leaves and other foliage, nets, fences, water, etc. and they use grayscale values to determine which areas are transparent and which are not. Ambient occlusion, or shortly put AO, map simulates details caused by a fake, indirect lighting. Those details are soft shadows that enhance the separation between objects and add an extra level of realism, because the eye can easily detect otherwise unnoticeable surface details. (Masters, 2014i)

Other grayscale textures are roughness, metallic and cavity maps. Roughness defines how blurry or sharp the reflection will be and metallic map also affects the reflections by literally controlling how "metal-like" the material of the surface is.

They both can also be calculated in the Unreal Engine 4 –game engine as numbers scaling from 1 to 0, which is especially useful for uniform objects that do not have complex structure or UV-maps. Cavity map can be seen as an AO map that has all the smallest details which the normal map might not present well enough. (Epic Games Inc, n.d.e)

Bump maps and normal maps are pretty similar to each other when creating detailed surfaces. Bump maps use greyscale textures to create fake depth to a surface, and are generally lighter for the renderer than normal maps. Normal maps use certain RGB values to designate the Y, X and Z coordinates of the surface, and typically create more detailed surfaces than bump maps. (Masters, 2014h, 2014m) However, a displacement map can also be used to create more surface detail, which affects the geometry's placement of an object. Even though it uses a greyscale map to create detail, which is generally easier to make than a full RGB valued texture like normal map, it is still pretty heavyweight for the renderer as it needs additional geometry to be moved in real time. (Russel, 2014)

Creating some simple light variance in surfaces can be done more easily than editing the high polygon model and then baking a new normal map from it. By taking a normal map of a sphere, it can be used as a colour palette to be colour picked and dropped (See figure 40). This is extremely useful trick to make several different variations of for example stone floors that have slightly tilted tiles that already has a normal map. The normal map works as a base for the colour dropping, to which the new gradients of colours are combined by the use of area selection tool and the overlay blending (See figure 41). (Hawkins, 2014, pp.63-64)

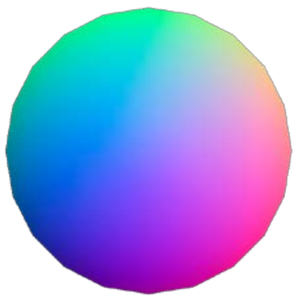
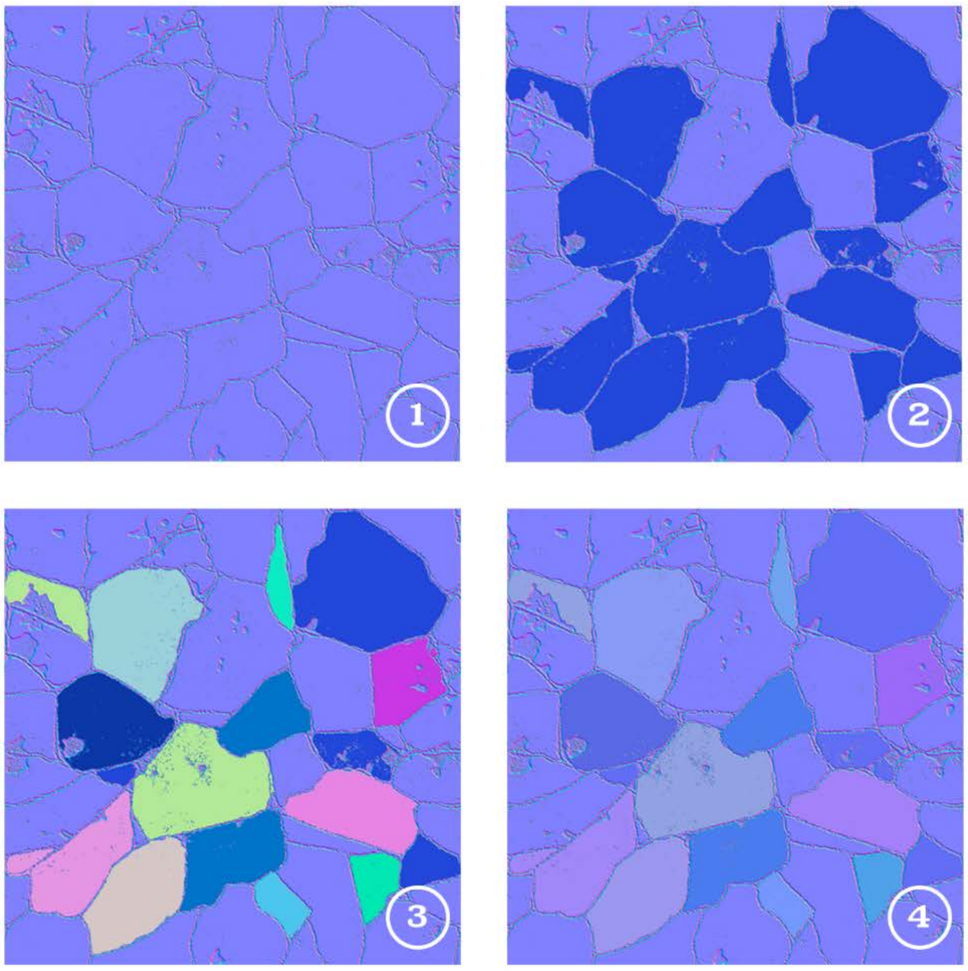


Figure 40. Colour palette from a sphere's normal map.



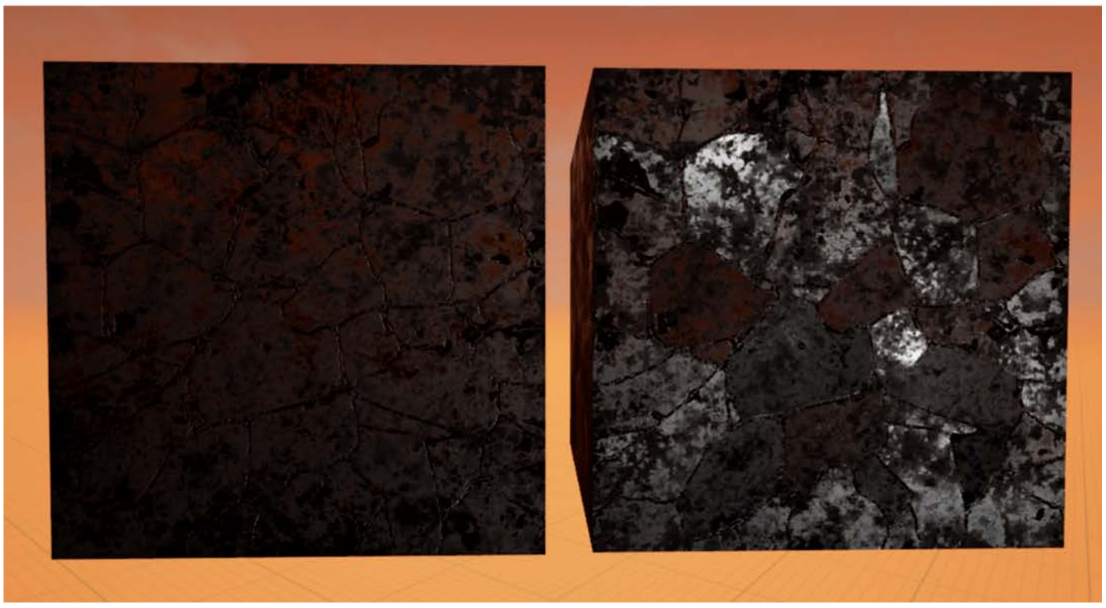


Figure 41. Using the normal map’s colour palette for tilting tiles in textures and comparison of regular and generated normal maps. 1: A regular normal map. 2: Selection of tiles. 3: Colour dropping. 4: Lowered opacity of selected areas.

To get the most details of an environment is to plan the texture library and the layers' naming convention, as well as making the textures a bit larger than first intended, because it prevents recreating them when they are needed bigger than before. (Hawkins, 2014, p.51) For large areas like floor, walls and roof, it is important to use tiling textures to lessen the required production time and memory space. (Masters, 2014g) Designing and then creating a base set of textures which work together at the start of the game development makes sure that the environment looks good without any added details like set dressing. This also prevents the possibility of starting over because of a badly composed environment. (Hawkins, 2014, pp.50-51)

Painting textures should be done in a manner of using the already painted materials for previous assets, like wood grain, cloth, etc. by copying them and then adjusting them a little paint over. This saves time from the overall asset production and is also helpful with consistency between different assets. (Hawkins, 2012, p.59) Creating a realistic texture by hand can be a bit time consuming. However, making a diffuse map based on the baked AO map, or any other baked map, is great for making materials that have a lot of noise. (McGrath, 2008) The baked map reminds of the forms that the high resolution mesh has, so it is easier to add more details on top of the AO layer in the painting software. (Hawkins, 2012, p.123)

It is worth noting of a texturing technique called diffuse painting. It was born as a solution to technical limitations, in which all the details such as reflections, depth, etc. was painted onto the diffuse map to save processing power. Usually it has been used as a stylistic choice of art style, because it gives more responsibility for the artist to make assets that need look good from all angles. Diffuse only models also give the opportunity to mirror and stack a lot of UV shells together, because the shading relies very little on the renderer (See figure 42). It is also useful for reusing the textures between different assets, making quick variants, editing both the existing assets' model and UV-maps and also moving the texture around generally. (Hawkins, 2012, pp.118-123)

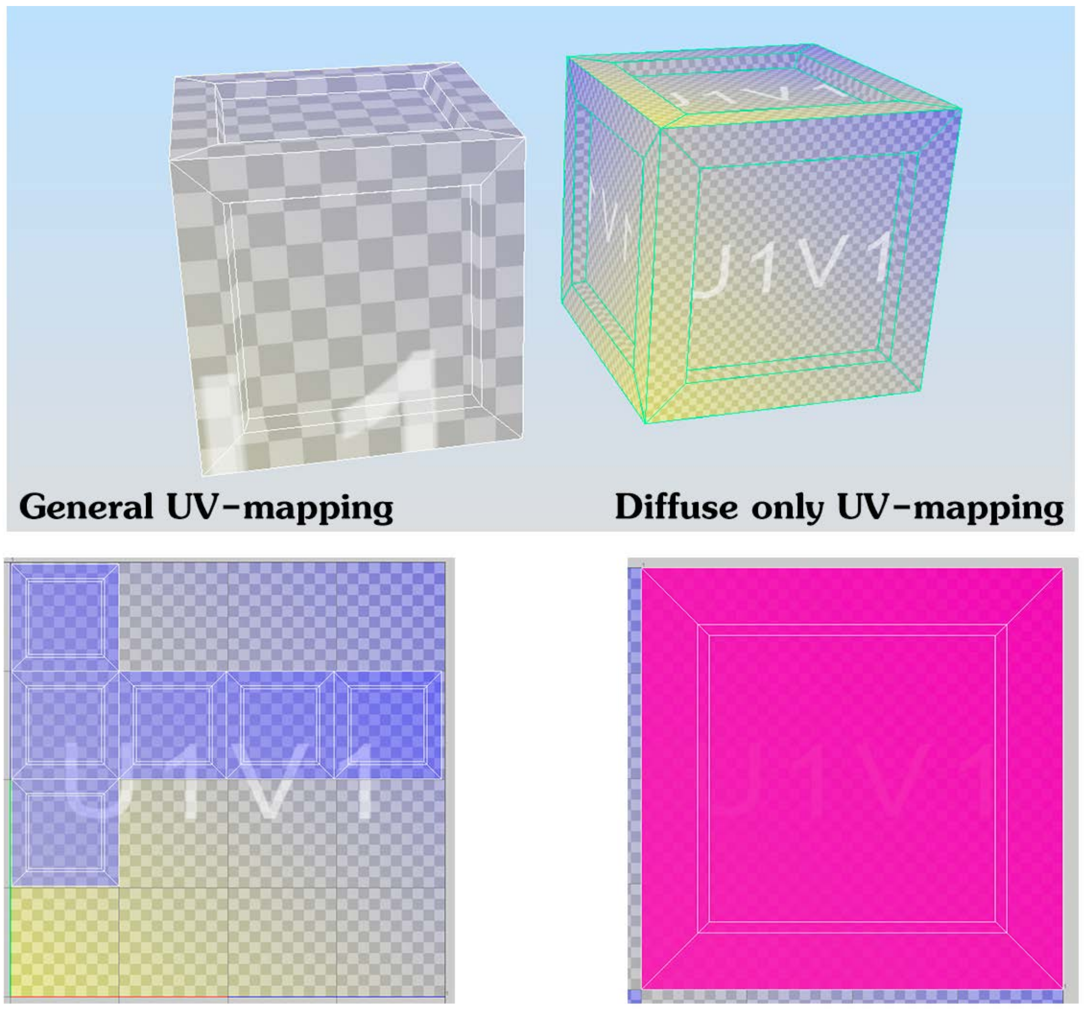


Figure 42. A comparison of a general model and an UV stacked, diffuse painted model. In this case, the pink colour on right indicates of overlapping UV-shells.

Textures can also be of use when modelling 3D-objects. Rather than starting with modelling a surface and then making the texture, a texture can be first applied to a simple plane and then modelled by following the details of the texture (See figure 43). To get more visual variety, it is important to make some trim textures for example for buildings. Reutilizing textures and trims in several different buildings creates a consistency in the whole environment. (Klafke, n.d.) Various props of individual buildings like doors, windows, etc. should be combined in to a single texture atlas. (Hawkins, 2012, p.59)

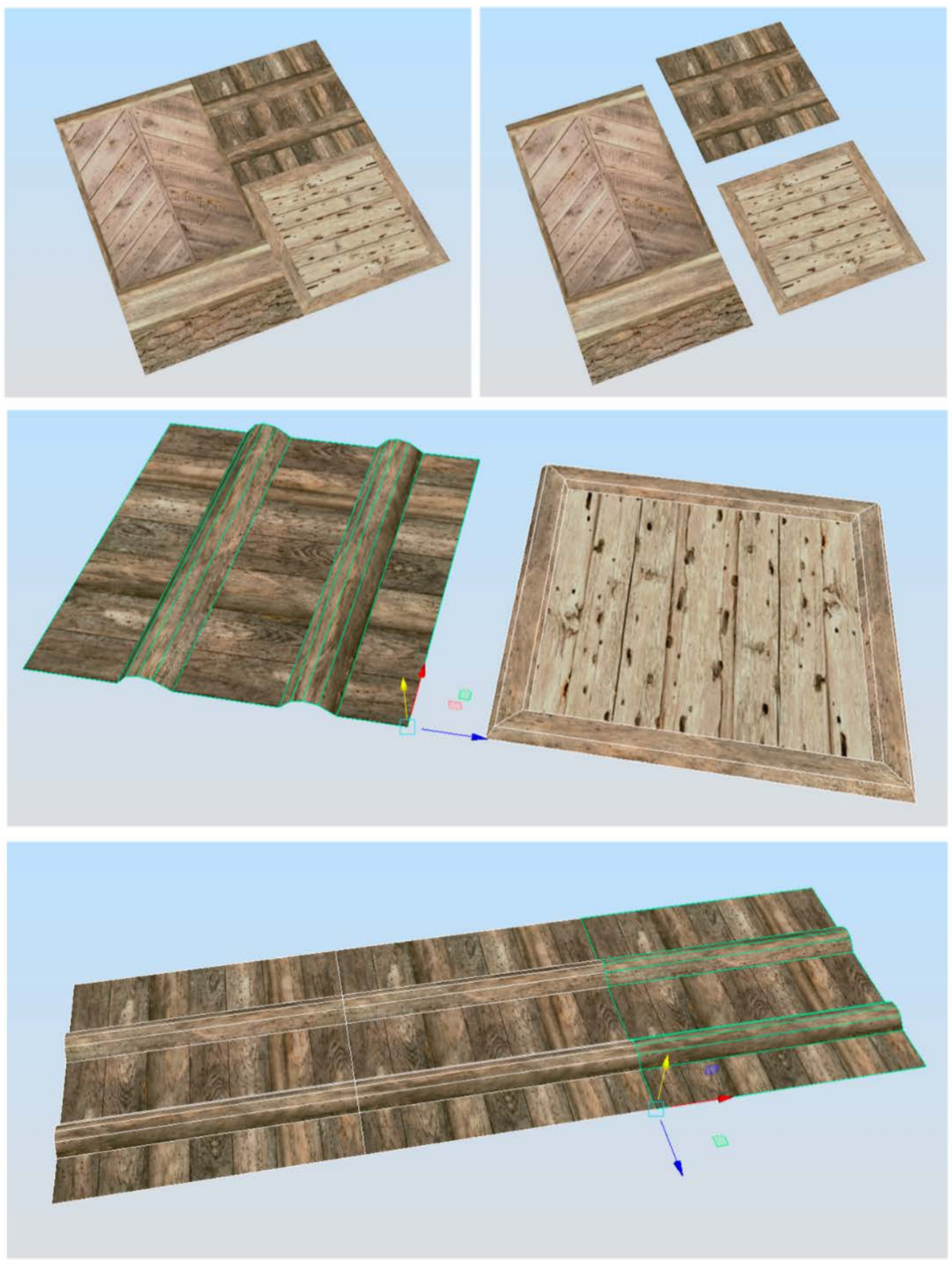


Figure 43. Using a texture for modelling tiling assets.

4.5.1 Diffuse brightness

For photo-realistic games, making of a regular colour texture also has some typical problems with how bright or dark the texture can be created. This is because the diffuse map does not have any light data, for example shadows or light reflections, as that data will be rendered in the game. If the original texture is too dark or has too much contrast, the shadows and lights in the game will not look realistic, especially with indoor environments that will only look extremely dark. The solution to this is not to increase the intensity of lights in the game engine as that will only increase the overall contrast in the environment, but to adjust the textures brighter (See figure 44). But what texture brightness is the right one? The idea of a correct texture brightness is when a surface is lit by a 100 % bright white light from any angle. (Epic Games Inc, n.d.b)

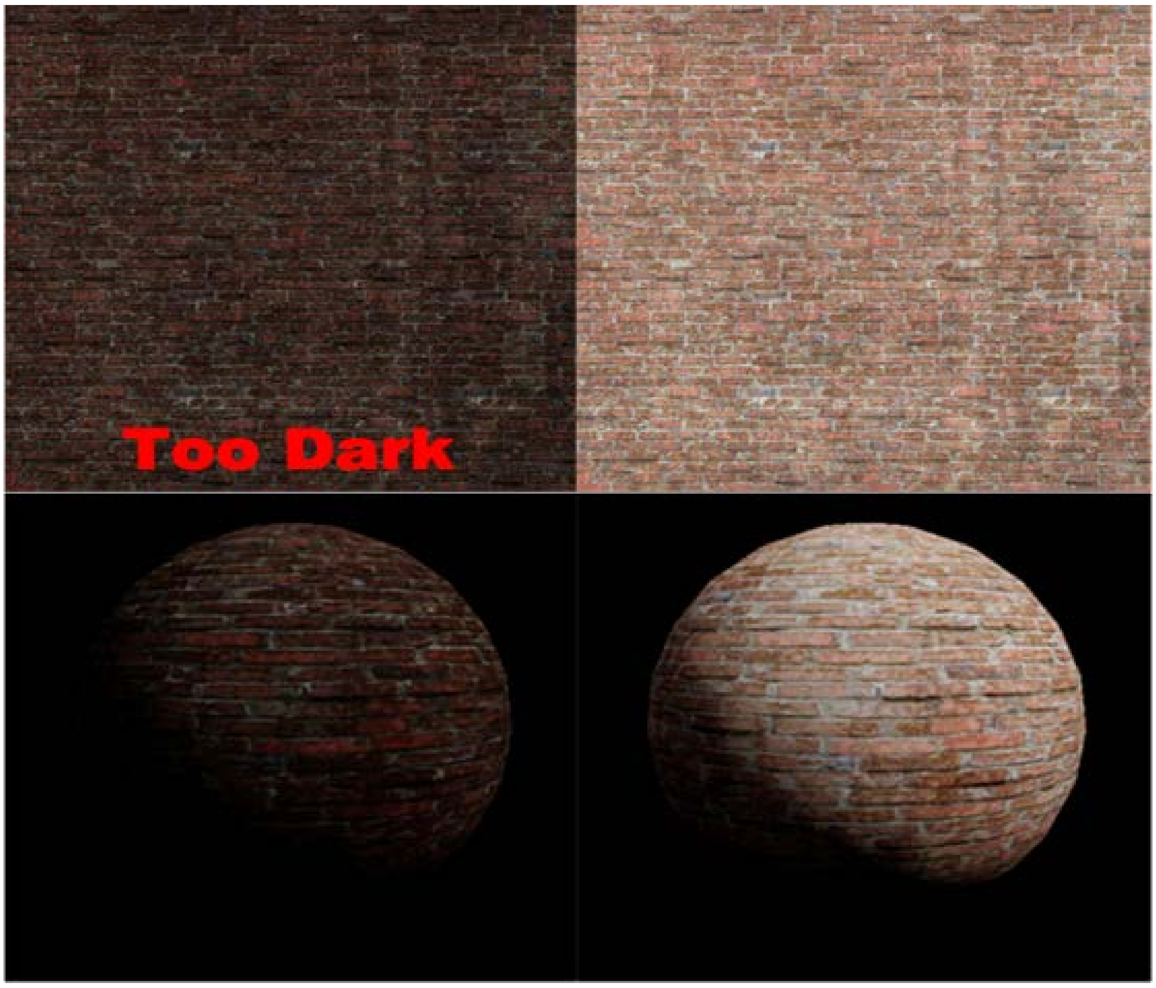


Figure 44. Comparison of same diffuse map that have different brightness. (Epic Games Inc, n.d.b)

Regular monitors’ light intensities are close a certain mathematical curve called gamma space. Textures are normally created using sRGB colour profile, which stores data in gamma space. But when importing the texture to a game engine, the texture will be converted to a different curve called linear space, with which the renderer calculates the lighting. Because of the different curves between the renderer and the monitor, texture can look okay in the artists monitor, but in a 3D-game and in different monitors it may appear different. (Gritz & d'Eon, 2007)

In gamma space the values are generally darker than in linear space. The mid-grey in gamma space as RGB value is at 127, 127, 127, whereas the mid-grey in linear space is at 187, 187, 187. (Epic Games Inc, n.d.b) The quick way to check accurate values is by comparing the accurate real-life values with Photoshop's histogram panel. If the values are off from the correct ones, the histogram's sliders can be adjusted to change the values correctly. Several material libraries have collected a lot of RGB value data, which can be used for making realistic textures. (Viscorbel, 2015)

4.5.2 Gradient mapping

Textures require a lot of memory and processing time, and to get the most details as possible, it is beneficial to optimize textures too at a certain level. (Provost, 2003a) Material's diffuse texture can be made by gradient mapping them with a texture called gradient mask, if the other details like wear and tear are presented correctly and the material does not generally have a lot of need for colour (See figure 45). (Maxinow, 2012)

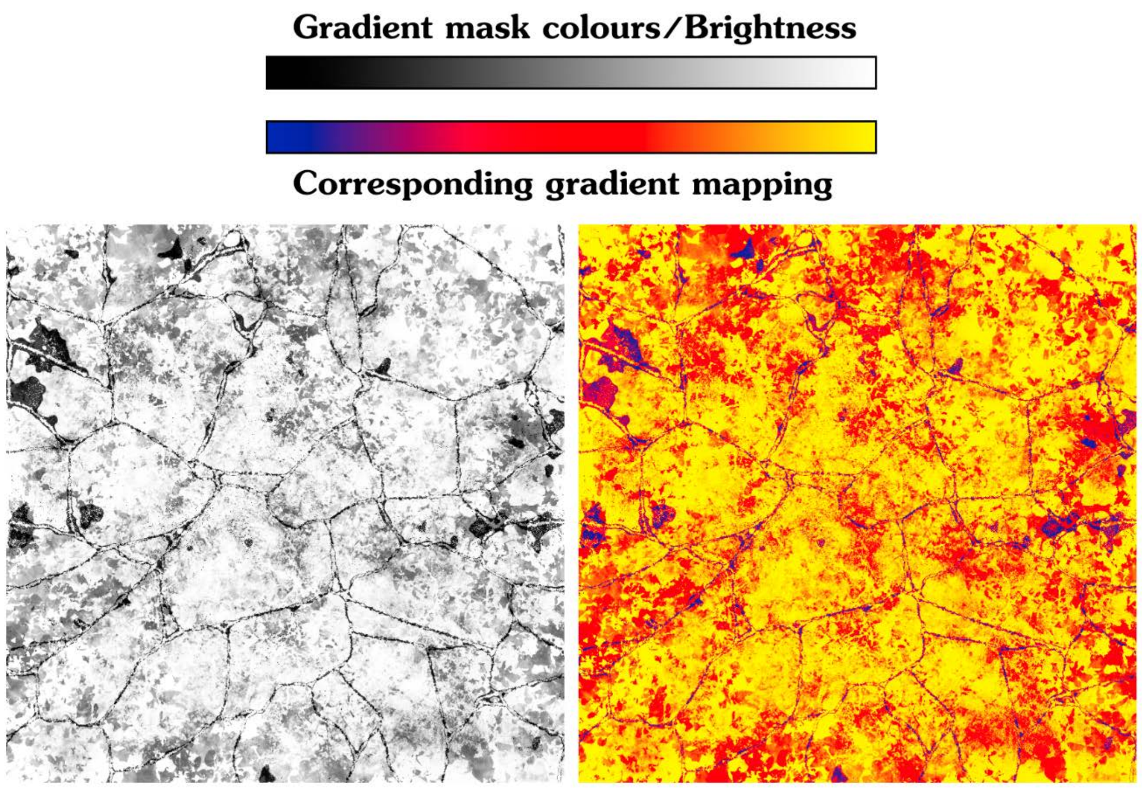

Gradient mask colours/Brightness
Corresponding gradient mapping

Figure 45. Multiplying the grayscale mask with gradient mapping.

The gradient mask can be created individually by hand or by combining three to four grayscale maps together by the simple use of RGBA, in short for red, blue, green and alpha channels (See figure 46). The textures that use grayscaling can be dynamically used in the game engine to create an enormous amount of different materials, especially with tiling textures (See figure 47). This saves a lot of memory space and also gives the design team a great tool set to work with, because they can tweak a simple slider of gradient values. And because the normal map's blue channel rarely stores any information, the normal map can also be combined with another grayscale map. (Maxinow, 2012)

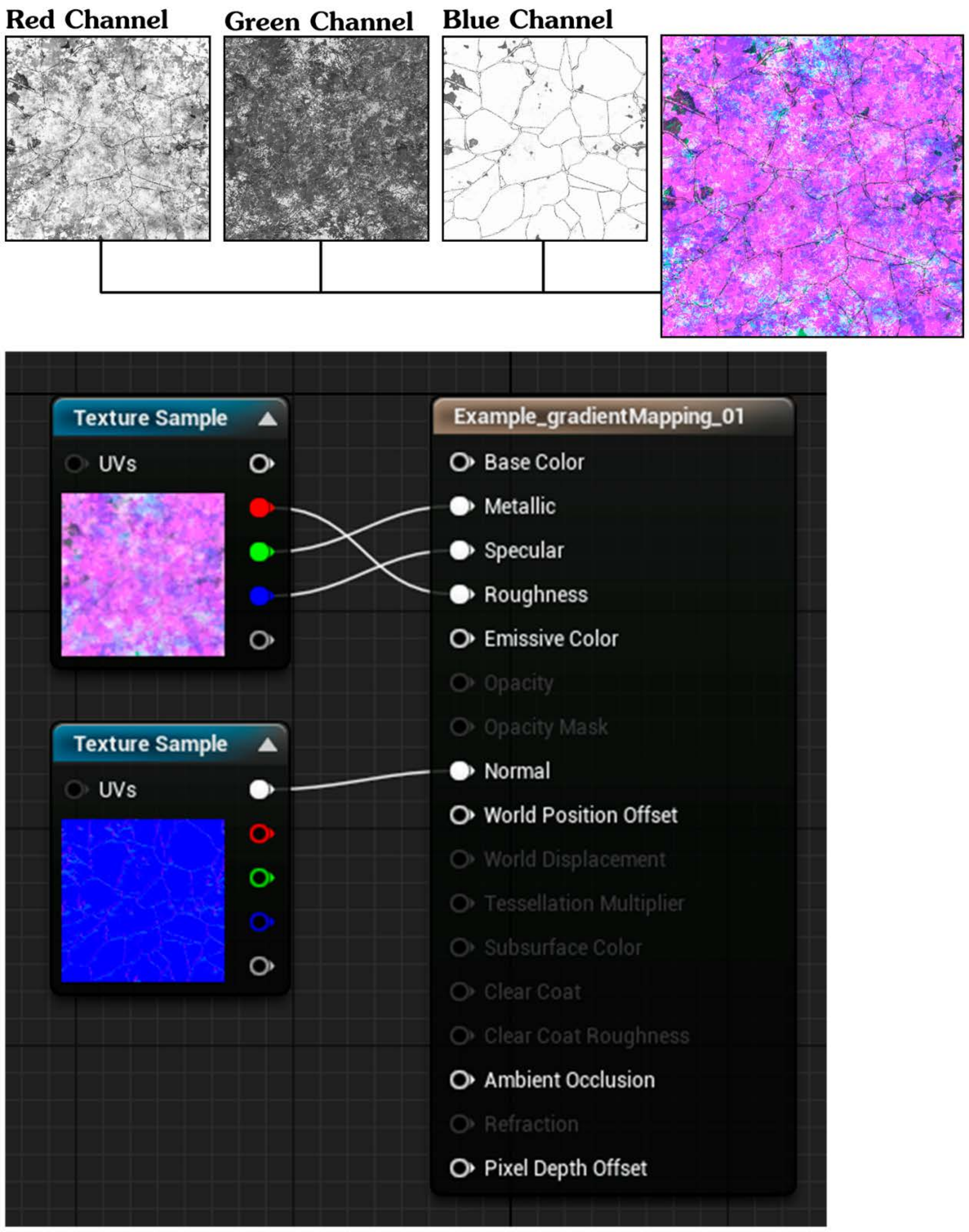


Figure 46. Combination of grayscale maps into one file and using the generated file with a normal map for a material in Unreal Engine.

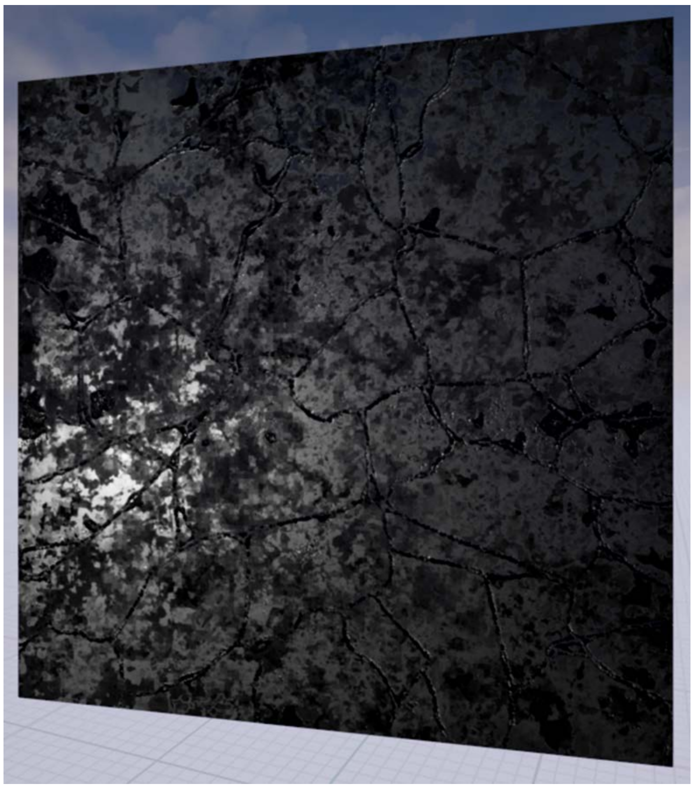


Figure 47. The generated tiling material from figure 45.

4.5.3 Using matte painting with environments

Matte painting is a regular technique of making compelling environment concept art and backgrounds for movies, which are usually so well made that there is no way to tell the difference (Masters, 2014j). It can also be used in games, because the player can not see the difference between a 3D-asset and a 2D image after a certain distance (See figure 48). However, to use this technique, the player's ability to move around has to be calculated and tested, as well as how expensive the matte painting will be because they tend to be quite large in the view. For example a 2D-image of a small city can be less expensive to process than a really wide mountain range. Matte paintings should be done in the last phases of the game development and are relatively easy to make as they are just simple 2D images. (Hawkins, 2012, p.67)

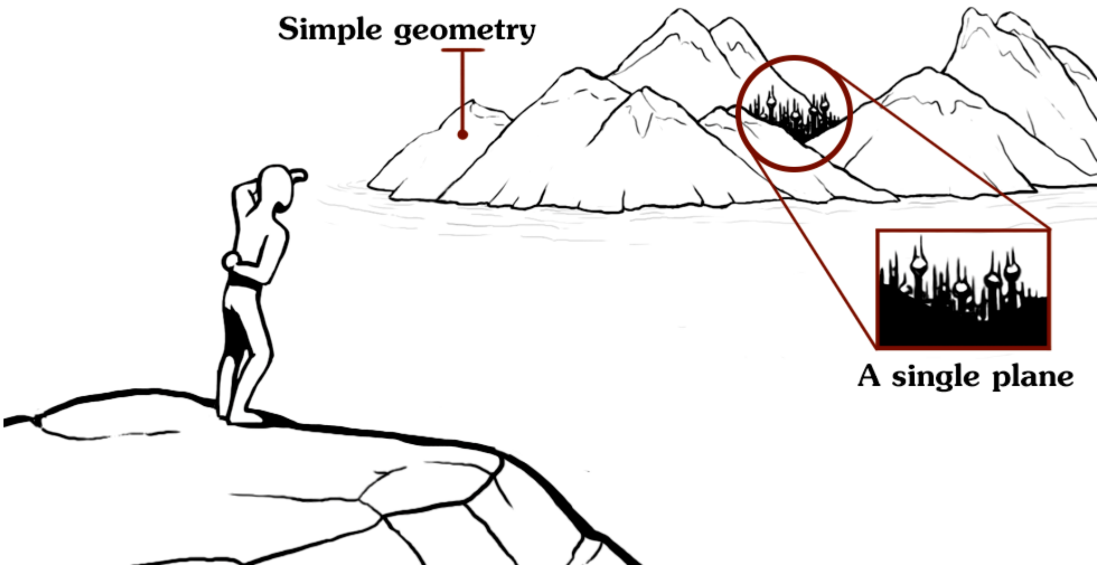


Figure 48. Matte painting used in an environment.

5 ANIMATIONS FOR ENVIRONMENTS

To start with the basics of animation, 2D spritesheet animation is basically just changing images in quick succession to create a continuous movement (Lambert, 2013). Good 3D animations require a lot more technical prowess than 2D animations. In 3D games the objects are expected to act as they would be in real life, and to create that effect, the objects are usually rigged, especially the more complex characters. Rigging means creating an object's skeleton that the animator can manipulate to bend said object. (Masters, 2015b) Animation for both 2D and 3D can also be done with puppet animation. This means that the animated object is split into clear, separate elements that are moved, rotated, scaled, etc. to create and edit motions easily. But as the term implies, the puppet animation does not look natural in all cases. (Fessler, 2014)

Animation for foliage is usually done with procedural vertex animation, meaning the foliage's swaying and bending is calculated by the computer in real-time. For example the development team behind Crysis took this further, and generated animations for the individual leaves to give a better illusion of realism. They divided the animation into main bending and detail bending, and blended those two animations together. (Sousa, 2007)

For materials and effects like water, smoke, clouds and fire, the animation can be done with changing the UV-maps coordinates, which also means UV panning. This is basically moving the coordinates horizontally and/or vertically, which works best with tiling textures. UV panning can also be combined with objects that are not just straight planes, but have curves. (Epic Games Inc, n.d.d)

Animating, for example a river or a stream, starts first with modelling a straight plane, UV-mapping it as a square with no curving, then continuing the modelling of that plane in to a curved stream leaving the UV-mapping intact. After that it is textured as a tiling texture and then UV panned in the game engine (See figure 49). This way the animation looks more fluid and does not require much work. (Epic Games Inc, n.d.d) 2D games can also be filled with animated planes like this to save texture memory and time instead of painting the sprites as individual images. Putting several layers of planes together also makes the animation look more organic. (AssemblyTV seminars, 2015)

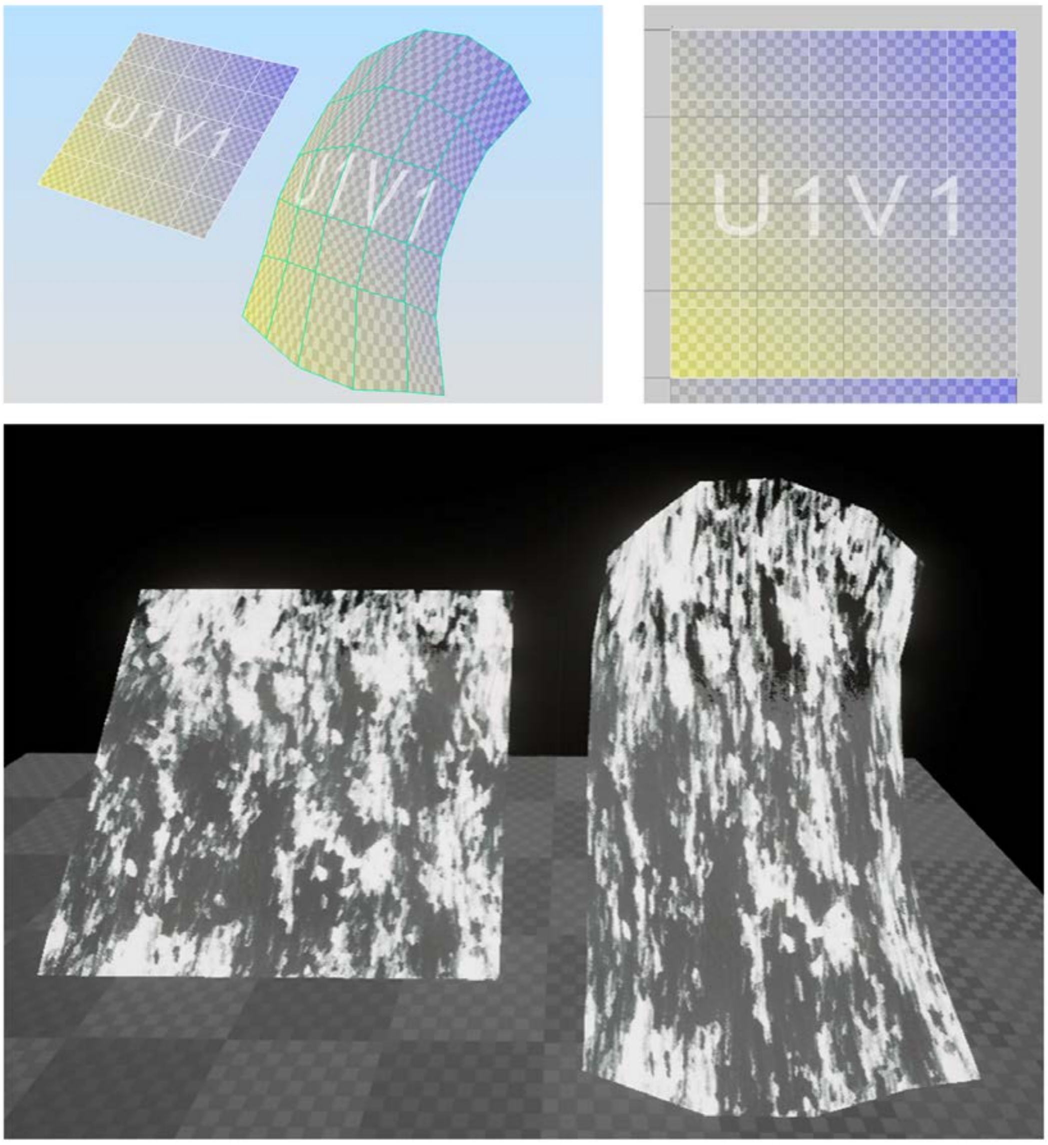


Figure 49. Creation of a water stream: creating a plane, modifying it without touching the UV-map and then applying a tiling texture with UV-panning.

6 OPTIMIZATION OF ASSETS: WHEN IS IT NECESSARY?

To avoid worst-case scenarios it is best to know what to optimize in an environment. It is easier to make mistakes on performance with level design than with graphics. The objects that are in view most of the time are generally the most detailed, so distributing them across the level evenly is a better solution than taking away the details. But in situations where correct level design does not help, objects' optimization needs are calculated. (Provost, 2003b)

6.1 Transform-bound objects

If an object's transform time, in short for how fast the renderer processes the vertices, is large, then the object is said to be "transform-bound." This usually happens with surfaces that have a lot of vertices or characters with complex animations combined with a large amount of vertices. Transform complexity is affected also by lighting, displacement maps and the distance between the object and the player. Distance affects the screen vertex density a lot, and the further away the object is, the vertex density rises as well. (Provost, 2003b)

6.2 Vertex knowledge

The amount of vertices changes a lot in the same object when travelling down the pipeline from a 3d-modelling software to a game engine and then finally to the game itself. In practice vertices get split at UV-seams, smoothing group boundaries and material boundaries, and can double or triple the vertex count, which then affect the transform time of the renderer (See figure 50). (Provost, 2003a) This is why it is not useful to measure the cost of a 3D-object by its polygon or triangle count, because the renderer only processes vertex count, which can vary in very many ways (Goldberg, 2013).

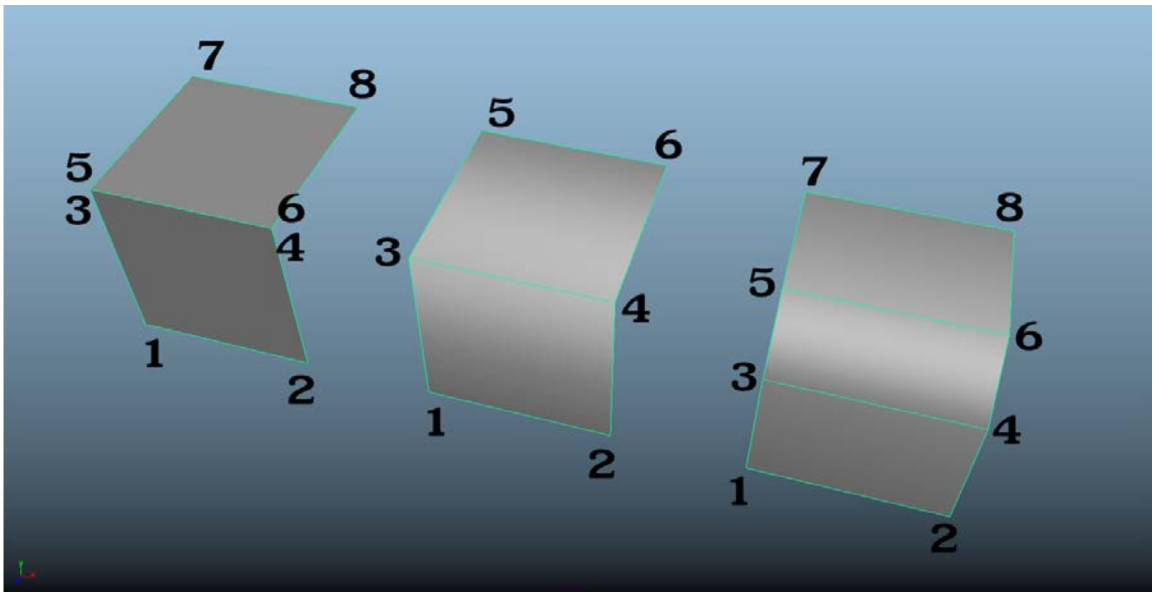


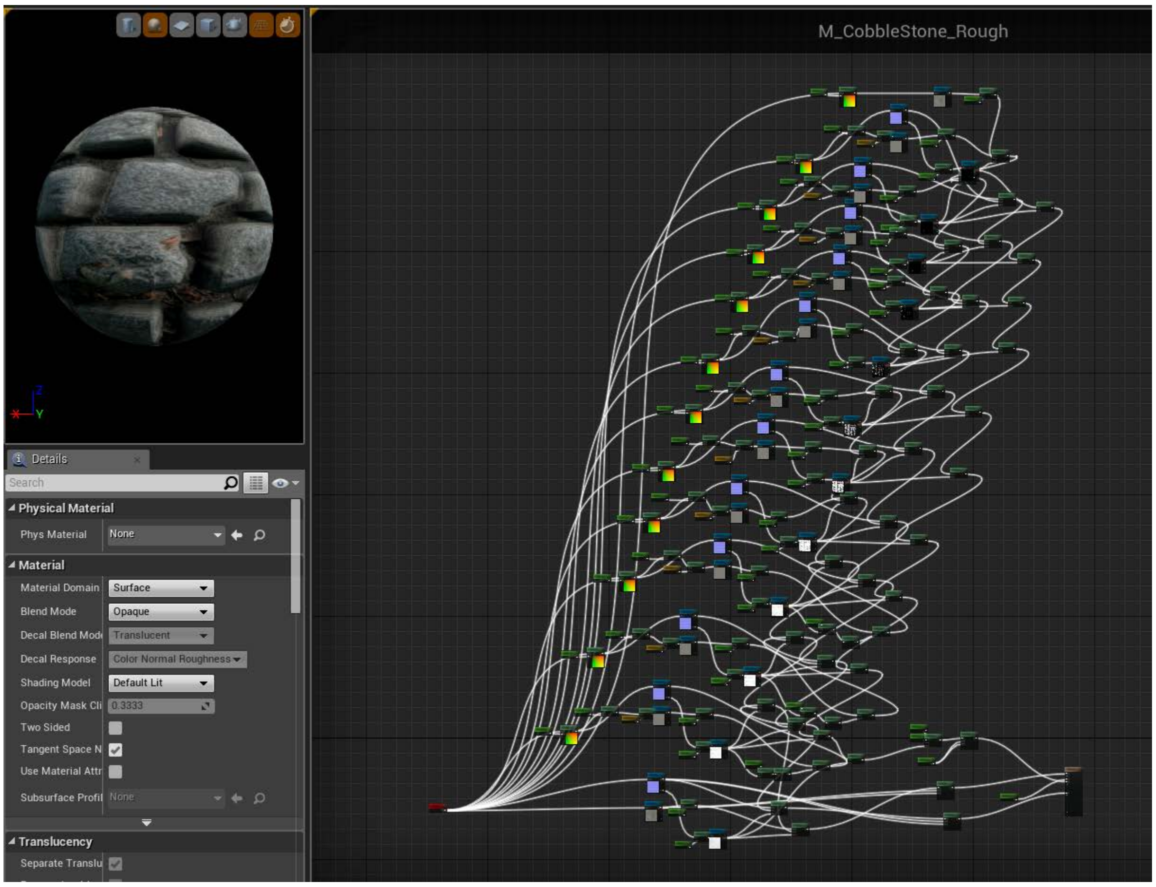
Figure 50. Examples of vertex splitting by smoothing groups from left to right: two polygon model with a hard edge, same model with a smooth edge and a three polygon model with a smooth edge.

To avoid vertex splitting, the amount of boundaries between smoothing groups should be reduced manually, UV-areas should be stitched together so that a single vertex is not shared by several triangles, and an object's textures should be combined together to minimize material based splits. Optimizing the UV-shells to cover the UV space smartly will also simplify the rendering process. In some cases normal maps are more useful than smoothing groups because the normal maps are usually mapped to the same UV coordinate set as the diffuse map so they do not cause any vertex splits. (Provost, 2003a)

UV-space boundaries, smoothing group boundaries and material boundaries can occupy the same set of faces to reduce vertex splits. This is because the renderer allows one smoothing group and material for each vertex, so it will only get split once. The same goes with the UV-space boundaries that occur at boundaries of smoothing groups. (Provost, 2003a)

6.3 Fill-bound objects

When the renderer takes a lot of time to render a surface on screen, it is said to be “fill-bound.” Big surfaces like walls, floors, etc. tend to become fill bound, which is caused by the large texture size, complex material properties and texture density on screen. The renderer is also usually fill-bound in low-density vertex areas. Unfortunately the surfaces can not get any smaller on a polygonal level, but at least the complexity of a material can be calculated by the amount of textures it needs to blend together before sending the compiled data to a renderer (See figure 51). In contrast to vertex density which rises with the distance, the rendering time of the surfaces decreases. (Provost, 2003b)



Physical Material

Phys Material	None

Material

Material Domain	Surface
Blend Mode	Opaque
Decal Blend Mod	Translucent
Decal Response	Color Normal Roughness
Shading Model	Default Lit
Opacity Mask Cli	0.3333
Two Sided	☐
Tangent Space N	✓
Use Material Attr	☐
Subsurface Profil	None

Translucency

Separate Translu	✓

Figure 51. A screenshot of a cobblestone material node system in Unreal Engine 4.7.

6.4 Techniques to optimizing assets

3D assets can shift from being fill-bound to being transform bound, when the distance increases, and from transform bound to fill bound when the distance decreases, if the vertex density of an object is non-uniform. If both of them are low in an object, it is save to skip that object and start checking the next object for any optimization needs. (Provost, 2003b)

There are several techniques to optimize an object. Two of the mostly used are mips-maps and level of detail meshes, shortly LODs, which both use distance to calculate when to switch the original texture and mesh to a smaller or higher resolution file. These two techniques are especially useful in outdoor environments where the field of view is large. (Provost, 2003b)

There is also a reason why objects, especially large ones, should not be merged into a single object. This is because when a small surface area comes into the view, be it how tiny as possible, the renderer still needs to process all the vertices and the texture files that are assigned to that object. Splitting the object in roughly equally same sized pieces, while thinking about both of the texture and mesh size, balances the rendering a lot (See figure 52). (Provost, 2003b) As an example, when making a racing game, the size of the pieces can be as big as city blocks. In a horror game that has only small corridors and rooms, the size of the pieces could be really small. (Perry, 2002)

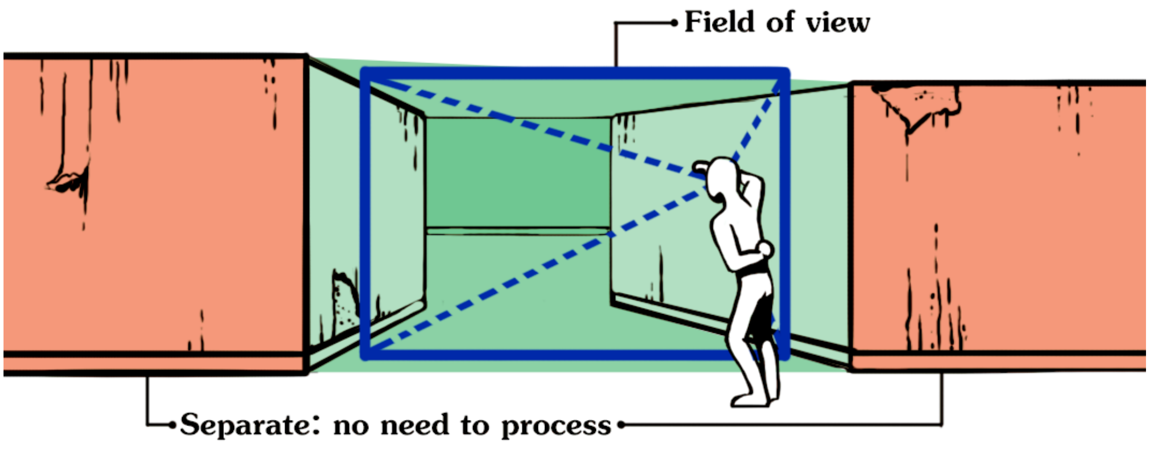


Figure 52. Field of view and processing of the objects.

Like with distributing vertex density equally across the mesh to avoid large transform time, UV-maps of meshes should also be tweaked to distribute the texture density uniformly across an object’s surface. Materials that have for example transparency should not be used on large surfaces, because they create several layers of geometry which needs to be rendered. Same goes with the double sided materials, because they can easily cause fill time related problems. (Provost, 2003b)

6.4.1 Texture knowledge

Texels are needed to be retrieved from an object’s associated texture every time the renderer is drawing the object’s surface. Processor does this through a texel cache on which the texture regions are pasted. Texel cache can also be called as a scratchpad. Before the processor can start drawing the pixels on screen, it checks the texels from the texel cache. If the texels are already in the cache, the processor proceeds with the drawing. But if they are not in the texel cache, the processor first needs to load the new texture regions, put them in the texel cache and then start drawing the pixels. This can be simply called as a texture cache miss. The more there are texture cache misses the more time it takes to draw a surface. (Provost, 2003a)

As an example, non-uniform texel density will cause texture cache misses. This can be avoided by tweaking an object’s UV-space roughly to the same proportions as they are in the 3D-model. (Provost, 2003a)

Many hardware processes data in limited “chunks” to be more efficient and works with the power of two numbers: 8, 16, 32, 64, etc. Because of this, it resizes textures that does not have the dimensions in that scale. The problem in this might not be that large with individual objects, but in large game environments where the assets amount is much larger, it causes longer rendering times on useless correcting. It is important to use power of two dimensions for textures for quick loading, and it also minimizes the effects of automatic resizing of a texture, for example blurring, fuzziness and corruption. (KatsBits, n.d.)

6.4.2 Cubemaps and selective reflection rendering

To help the rendering process of reflections, a cubemap is created in the game engine. In practise, it can be seen as a camera pointing to every direction of 90 degrees; right, left, top, bottom, front and back, and then creating a panoramic picture from them. That picture is then rendered in the reflections, and is usually very low resolution. It can also be converted to a spheremap to get better performance. The difference between a cubemap and spheremap is that the cubemap has six texels, whereas the spheremap only has two texels. (Courrèges, 2015)

Large areas like puddles, lakes and sea reflect an enormous amount of surrounding lights and objects, which can cause the frame rate to drop as there can be a lot of objects to render. But not every reflection has to be rendered, for example things that are really far away from the reflective surface. Almost Human Ltd (2014b), the team behind Grimrock 2, has made the rendering objects' reflections automatic by choosing a certain rendering mode for each of the objects by marking them as “never”, “always” or “cell”. “Never” and “always” should be quite obvious, and the “cell” mode is for the level designer to paint in the level editor which cells have their reflections enabled. (Almost Human Ltd, 2014b)

7 GAME ENGINES

There are several different game engines released to the public ranging in price, power and complexity. Some of them can suit better for making 2D games, 3D games, photorealistic or stylized graphics, etc. so it is important for the developer to carefully calculate his needs before choosing one. (Masters, 2015a)

7.1 Instancing

Instancing is a basic system of referencing several duplicates of assets from a single file into the game, which helps with the memory management of the hardware. (Epic Games Inc, n.d.c) It is also useful in general asset management and replacing old assets with a new one, because the game engine automatically changes the instances based on the file change. This saves time from manually placing all the assets again to their original places. (Marshall, 2014a) Developer can also use the original file, for example a material, as a master instance, creating several visually different materials that use the original material's parameters as a base. Whenever a change is made for the original material, for example optimization, all the "child" materials are also optimized in that change. (Van Spronsen, 2014) In many situations, an entire file can be replaced with something else as long as the new data is functionally compatible with the old one (Perry, 2002).

7.2 Level of Detail

Level of detail (LOD) system retains visual details using a hierarchical distance relative change of assets, without increasing the workload for the hardware. In practise, the game engine changes the asset, 3D-model, textures, etc. with a lower quality asset when the distance between of an object and player increases. The amount of different LOD assets is depended of the game, and the percentage amount of detail, for example vertex count and texture resolution, can be calculated to be 100% in a high quality asset, medium quality at 50% and low quality at 25% (See figure 53). (Cheng, 2015a)

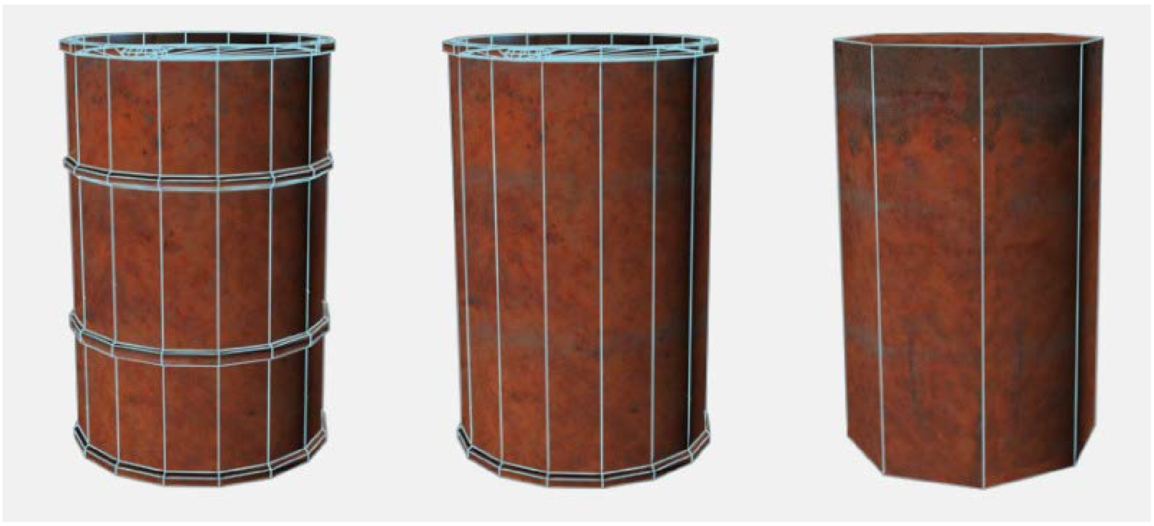


Figure 53. Three LOD levels of a barrel. Adapted by author from (Masters, 2014).

An ugly popping effect between assets can be seen in a regular LOD system, which is usually minimized with a crossfade or dissolve effect between the LOD assets. (Almost Human Ltd, 2014b) Unfortunately terrain needs another solution for this, which can be called as a continuous method of LOD (Cheng, 2015a). In this method the hardware procedurally decreases and increases the amount of detail in real time and blends the details of a high and low quality terrain assets that are next to each other (Hoppe, 2004). Even though the method provides a steady frame rate, memory efficiency and a better looking terrain, it is still hard to implement (Cheng, 2015a). As an example of different techniques in one game, in Just Cause 2 there are twelve LOD assets for the different sized terrain areas, it uses “geomorphing” to align the vertices of a high quality mesh with the low quality mesh and has procedurally generated data for the inaccessible, distant game world, as the visible distance can change because of the various player locations in the world (Blomberg, 2013).

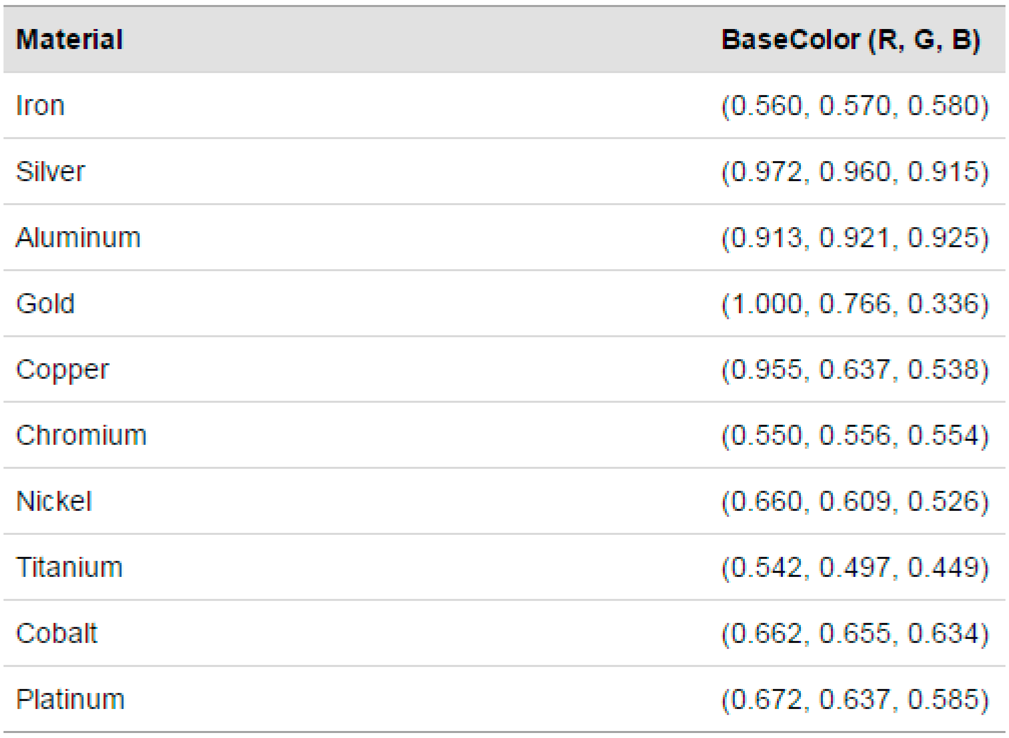
7.3 Materials and shaders

The idea behind materials is that they literally define the surface’s properties like roughness, transparency, etc. in addition to its colour, and how the surface reacts to light. Materials use input such as images, math and other various settings as their base to create the calculations for a surface’s light reaction. (Epic Games Inc, n.d.e) One of the good perks of materials is that they also can be instanced for several variations. For example a master material of wood could be instanced for new, old and mossy wood, which uses only few more calculations or textures for the new surface details. This saves a lot from memory space and is also light-weight for the processor. (Van Spronsen, 2014)

Shaders are algorithms that work in a close relationship with materials. The material contains all the data that a surface needs, but the shaders actually combine the material’s data with the object and lighting to create the final shading. Shaders also affect how many options materials can have, and can be especially made for foliage, liquids, etc. (Unity Technologies, n.d.c)

In need of realistic environments, more and more game engines and developers are slowly shifting to a rendering system called physically based rendering (Orsvärn, 2015). This rendering system needs physically based materials and lighting to work accurately. The great thing about physically based rendering is that the values used in the materials are taken from real-life values, so they work equally in all lighting types and environments and look more natural (See table 1). (Epic Games Inc, n.d.e) Using the real-life values increases consistency in the whole environment and the system is generally easier to use between several different game projects. Artists no longer have to guess which values work best with which material and lighting, and creating textures is easier as the same rules apply for all of the textures. (Wilson, n.d.)

Table 1. A screenshot of a colour set of RGB values for certain materials in the game engine. Adjusted by author from (Epic Games Inc, 2015).



Material	BaseColor (R, G, B)
Iron	(0.560, 0.570, 0.580)
Silver	(0.972, 0.960, 0.915)
Aluminum	(0.913, 0.921, 0.925)
Gold	(1.000, 0.766, 0.336)
Copper	(0.955, 0.637, 0.538)
Chromium	(0.550, 0.556, 0.554)
Nickel	(0.660, 0.609, 0.526)
Titanium	(0.542, 0.497, 0.449)
Cobalt	(0.662, 0.655, 0.634)
Platinum	(0.672, 0.637, 0.585)

However, the materials can still become a problem if they are not organised. This is because of games that require a lot of artwork, the materials list can become enormous. Designers might not find materials fast enough and end up making their own for example. This is why there should be only a certain set of materials which to work with. Even the calculation count in the material can become a bottleneck for performance. To prevent this, for example the multiplied or overlaid textures with math could be added to the texture in the painting software. (Hawkins, 2014, pp.154-156)

7.4 Decals

Decals are 2D images projected on to 3D-objects' surfaces in a game engine to for example break tiling textures on large surfaces with an image of for example moss, ivy, posters, the list can be endless (See figure 54). They do not add any extra geometry, so they are pretty lightweight to use. (Masters, 2014g)

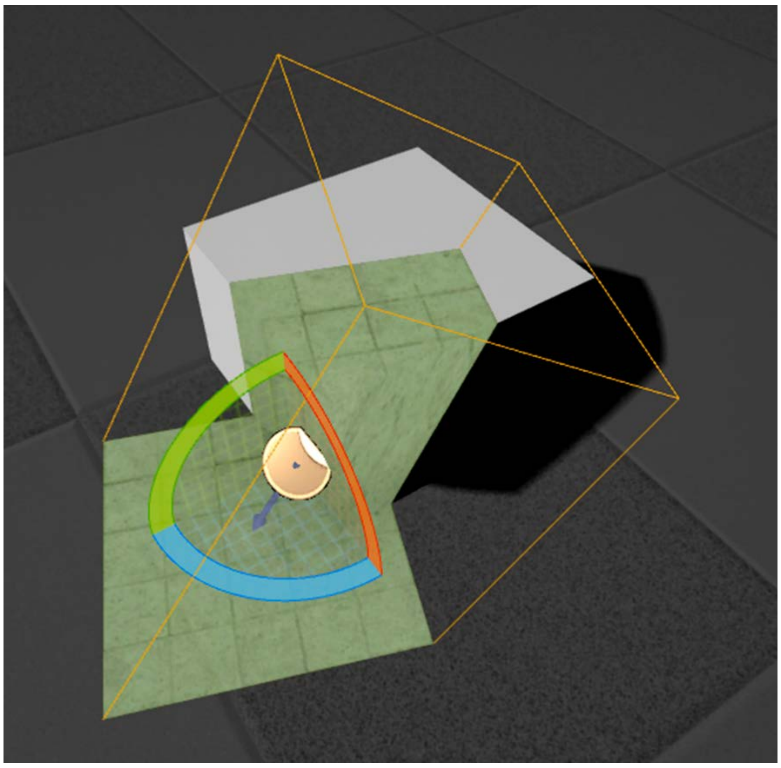


Figure 54. Example of a decal, which in this case creates a green tile pattern on two objects.

7.5 Particles

Things like smoke, explosions, snow, fire, etc. can make the map look more alive with motion (Becerra, 2015). They can be created with particles, or in another name, particle systems. Usually a single particle is a small polygon plane that has a texture map with partial transparency. So when for example simulating a snow-fall, there could be thousands of planes in the view at a time. A single snow flake can also trigger another particle system when it hits the ground, creating the effect of snow breaking up into new snow flakes. Because of this, the particle system should be as lightweight as possible for the renderer. (Van der Burg, 2000) Also the transparency in particles creates a lot of layers for the renderer to process, which can also be called as overdraw. To decrease the processing time of particle systems like fire, the textures should have more detail in them to decrease the amount of needed particles, and with particles like shrapnel and pebbles, the particles can be solid, lightweight geometry. (Ericson, 2009)

7.6 Lighting

There are several editable types of lighting tools in regular game engines, which can be used to create the exact mood for the environment. Lighting between the levels should be different, so that it provides the player with new experiences. However, the lights need to be calculated with each frame in the game, and if done wrong they can cause the rendering time to increase enormously. (Masters, 2014f, 2014m) The lighting effects like a dynamic day and night cycle, accurate refractions and interactive lights need a lot of hardware processing power to work properly. (Stuart, 2015)

There is a method called lightmapping a scene, which basically saves all the light and shadow data directly on the texture of objects, which is also called as baking (See figure 55). The created textures can also be called as lightmaps, and after which the actual lights can be deleted from the scene, saving processing power. (Unity Technologies, n.d.b)

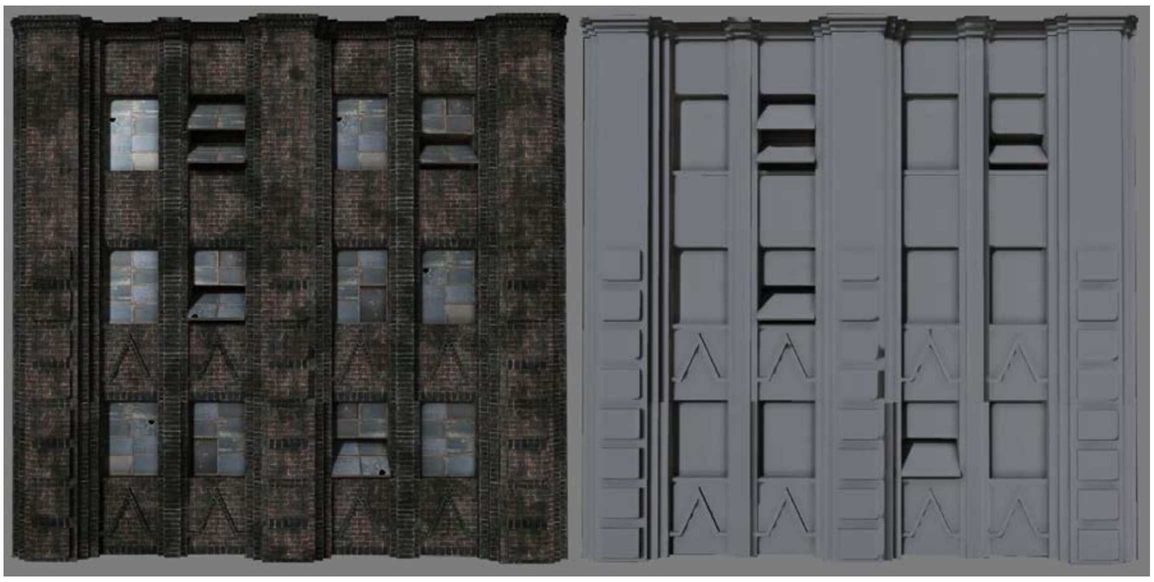


Figure 55. An example of a lightmapped asset. (Epic Games Inc, 2015)

The lightmapping technique has to be carefully thought of, because not always the light bake looks good on an environment (Georgio, 2012). This is because the lightmaps' are generally so small that the UV shells colours bleed easily to each other and cause other artefacts like pixelated shadows, so they also need extra care with edge padding. (McGrath, 2008) Lightmaps also do not make any dynamic effects to moving objects, for example to a character walking under a streetlamp. This is why it is important to know what lights to bake and not to bake. (Masters, 2014f) As an example, no dynamic lights but a single 1024x1024 lightmap was used for the War Games maps in Halo 4 because of the performance budget (HD Admin, 2012).

Different lighting tools work differently from another, for example a spot light can be self-explanatory, but there are also lights called point light, area light, directional light, volume light and ambient light. Additionally, the light hardness or softness is usually determined by the distance and size of the light source. (Masters, 2014k) Different lights create different scenes, for example point lights can be placed near wet areas, so that the light reflection bounces off from it (Van Spronsen, 2014)

Spot and area light both have a form in them, where the spot light conveys the light in a cone shape and the area light has either a rectangle or a circle as its source shape. They are both great for creating directional, eye catching lighting, for example the area light can be used to light the scene through a window. Point light illuminates an area in the shape of a circle, and is useful at illuminating the scene softly if desired. It can be used to create light for candles and light bulbs. (Masters, 2014k)

A directional light emit light in to a single direction to an infinite distance, which is used to create the illusion of a sun into large, open environments. Volume light is similar to the point light, but it can be changed in to primitive shapes for example like sphere or a cube, which are useful to illuminate objects in their volume. It is useful at creating beams of light in fog and other environmental light patterns. Ambient light does not create any shadows or shading, as it does not have any directionality, but is great for filling a scene that does not have enough light. (Masters, 2014k)

Lighting 2D games can be tricky, because the sprites are originally just flat planes. To bring out more depth and mood to 2D games, normal maps can be used to create more dynamic lighting. This can be done in three ways. The first method is by modelling the objects and then baking the normal maps from them, which can be then combined with the diffuse maps. The second method uses photos as a source to generate normal maps automatically, for example by using a software called Crazy Bump. The third method uses hand-painted normal maps, which is generally much harder to do. However, a software called SpriteLamp offers a simple solution to this. It uses hand-painted grayscale maps called "lighting profiles" to generate not only a normal maps but also other maps without the need to model anything. However, it can be time consuming to draw all the lighting profiles for individual assets, but at least it offers another option to create texture maps. In game engines, it is also possible to add other texture maps like depth maps to add lighting variety to the environment. (Morgan, n.d.)

7.7 Post-Processing

Adding screen effects is called post-processing. The available effects are depended on the game engine, but usually there are several effects to adjust a scene. They can make the scene look very unique and interesting if the artist spends enough time tweaking them. (80.lv, 2015) Some effects might work better for example in indoor environments than in outdoor environments and vice versa. (Becerra, 2015) As an example of post-processing effects, some of the most common are anti-aliasing, which smooths an object's jagged edges on screen, motion blur which blurs the scene based on the viewer's motions, and depth of field which blurs the objects based on their distance to the viewer. (Epic Games Inc, n.d.f) Using the post-processing effects correctly is basically a balancing act of visuals (HD Admin, 2012).

8 DESIGN PROCESS

Every kind of environment has a huge amount of objects in it, and every one of them has a story to tell, be it large or small. Planning them is an enormous process, and should be one of the most refined production process so that precious time is not wasted on pointless iterations on details. (Masters, 2014e) Selecting, researching, developing and then refining are the steps to make a successful design whether the design is for a single object or a vast environment. All of the successful artwork are based on the same visual grammar, which have evolved through the classical art techniques. It should be noted that presenting a person’s work to other members of the team is extremely crucial for feedback, because not everyone can think the same. (Solarski, 2012) The following sections will not focus on a specific design or story, but will discuss matters on a general level. Because of this they can be applied to other graphics development such as character creation.

8.1 Designing single objects and entities

Several environment assets and other props should be designed in unison, rather than making them one at a time. This is because of the easier judgement whether or not the assets look good together and to save time from tedious reworks. (Solarski, 2012) Even though every asset bears a certain visual strength in a composition, objects that are supposed to attract the eye should have high enough contrast from their surroundings, such as colour and shape. (Lovato, 2015)

8.1.1 Brainstorming and high concept

Developing a high concept, a short paragraph which defines the design goal clearly, is important for the development team as it provides a common reference point. It defines the desired emotional experience of an object in a short summary, from which the art and design teams can start their creative process. (Solarski, 2012)

To start brainstorming a high concept, keywords are needed. They are small adjectives, which define emotion, colour, shape, size, etc. The reason for they needing to be adjectives rather than nouns is because they describe emotions better. There should also be a design opposing factor in the keywords, which will also result in a deeper meaning and emotion. These selected keywords are then formulated into a high concept (See figure 56). (Solarski, 2012)

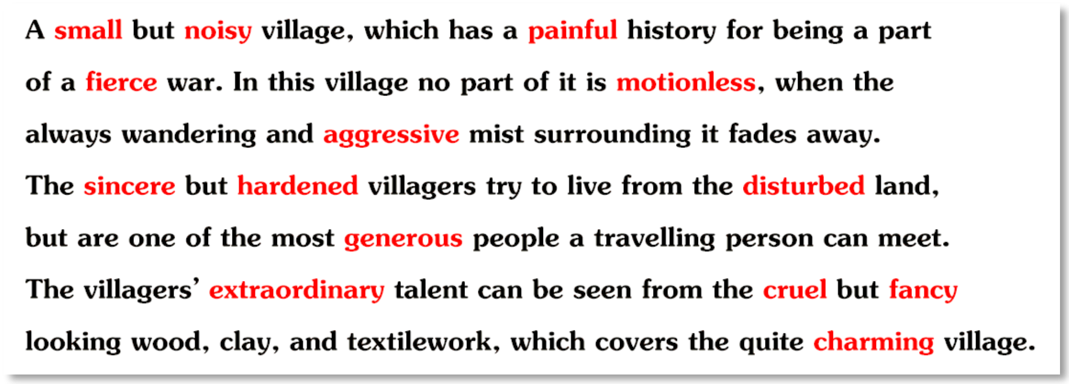

A small but noisy village, which has a painful history for being a part of a fierce war. In this village no part of it is motionless, when the always wandering and aggressive mist surrounding it fades away. The sincere but hardened villagers try to live from the disturbed land, but are one of the most generous people a travelling person can meet. The villagers' extraordinary talent can be seen from the cruel but fancy looking wood, clay, and textilework, which covers the quite charming village.

Figure 56. A high concept of a village with adjectives highlighted.

8.1.2 Research

Based on the high concept created, art and design department should start gathering reference images. By creating a mind-map of the keywords and then further diving into them uncovers an array of not before seen material. This can be relatively easy, as the researchers can use the keywords described in the high concept, rather than just randomly seeking with words that might conflict with the high concept. (Solarski, 2012) Because of this, the first idea, which every other game developer would also have thought of, is avoided, and other strong ideas will emerge. (Tudor, 2010).

In the end, having a full arsenal of great reference images may take hours, days, even weeks to complete, which depends on the game development schedule. (Solarski, 2012) Sometimes an idea of the high concept can lose its impact as it moves through the production phase, so it might be a good idea to pick the best or most extreme reference to prevent it from happening. (Hawkins, 2012, p.31)

After gathering of the reference images, it is time to start drawing some small sketches based on them. Sketches of the first seen key features can be really rough, as they are later used as their own reference images (See figure 57). (Solarski, 2012) Keeping what is interesting and amplifying the sketches further with the keywords should help to convey the true form of an object that the artist or designer wants to replicate from their minds. (Cheng, 2014b)

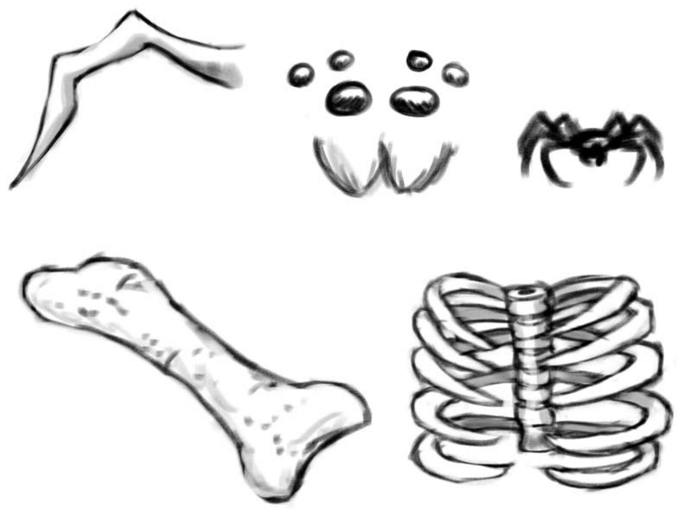


Figure 57. Example sketches of key features.

8.1.3 Developing thumbnails

It is relatively easy to seek new ideas that come up from the researched images and the key point sketches. Next up is to create an object’s design by the use of thumbnail sketching: starting very simply with the mix and match use of primitive shapes’ silhouettes and the previous research (Kennedy, 2013, p.36). The idea behind a thumbnail sketch is to keep the drawing really small, so that the primary elements of form and shape can be seen immediately and so that they could be quickly iterated (See figure 58) (Tudor, 2010). The primary shapes, circle, square and triangle each create certain emotions, so it is good to consider which shape works best with the high concept. Using the research too explicit in the design is not a good idea, as it will easily take away the emotions caused by the primitive shapes. (Solarski, 2012)

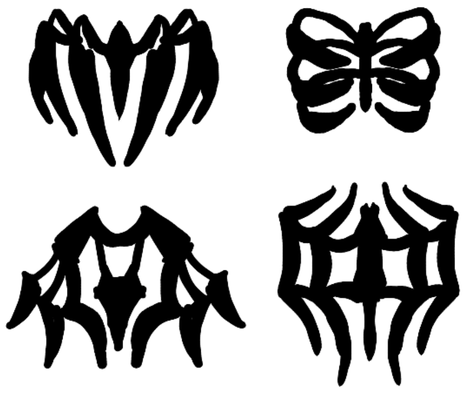


Figure 58. Thumbnail sketches.

Some problems can occur with an object’s visual clarity. If there are angles in the silhouette that are too subtle and disappear with the rest of the objects’ from, especially when viewed far, it is best to exaggerate them a bit (See figure 59). For example, to define two elements separate, it is good to exaggerate contacting points in the silhouette. (Solarski, 2012)

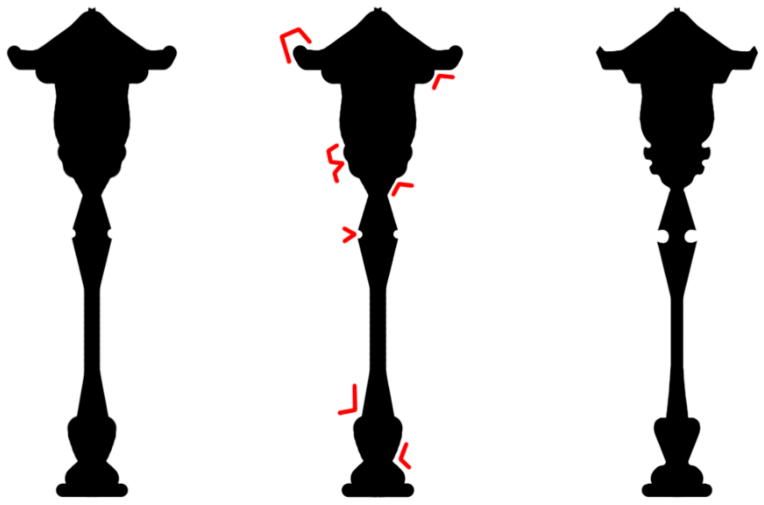


Figure 59. An example of exaggerating silhouette to clarify form.

At this point it is also best to develop other objects’ and environment’s silhouettes to get a visually good overview of several objects together and to compare them with each other. (Solarski, 2012) Making of multiple pictures from which to choose from helps the visual process of the design, because usually the team or the client does not know what they want but rather know what they do not want. Having multiple choices helps to convey the best solution out of the rest. (FZDSCHOOL, 2012)

8.1.4 Refining

After successfully making of a silhouette that satisfies everybody in the team, it is time to make it as a full detailed drawing. In this phase, silhouette's outline is traced again multiple times onto a new paper or layer, and the artist can start adding details into the design, and compare which detail works best with each other (See figure 60). The added details should not change the already approved silhouette. (Solarski, 2012) It is worth noting that randomly placed lines create random details that will only distract the player. (Jansson, 2012) Adding details can also be done with an easy value painting method, where black, white and various shades of grey are added onto the silhouette. When started simply, grayscale painting works great for creation of a pleasing design, and also creates a base onto which the colours are easy to put on. (Kennedy, 2013, p. 38)

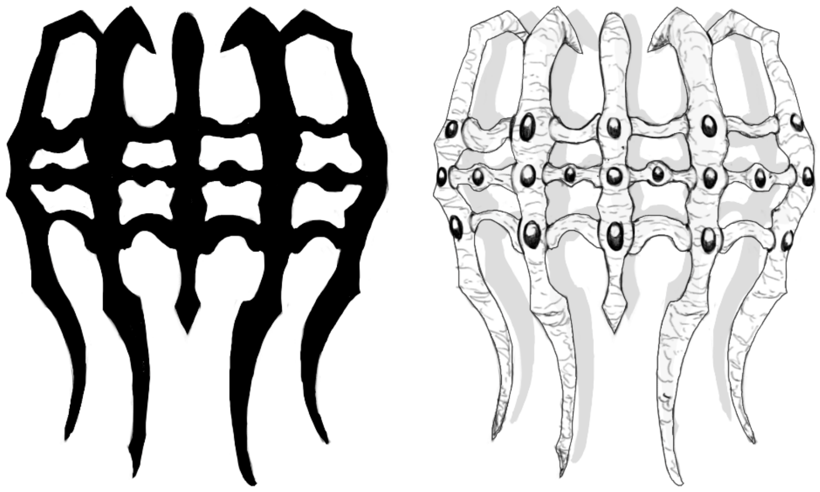


Figure 60. Adding details within the silhouette.

The final design is then put to the game or a model sheet, which also has other references like materials' textures and separately cut details, so that the obscured details are also shown. The model sheet works as a general reference document, and can be used for both 2D and 3D work. (Solarski, 2012)

8.2 Level design

Planning out how and where the player goes in a level or a certain area of a map is essential to keep him satisfied, regardless of how unique or beautiful looking game’s graphics are. There are things to watch out for and playtest early, for example player getting lost or confused, not enough action in a multiplayer game or simply the level design being too linear, where there are no different routes to take. However, good level design will not give all the answers for the player up front, thus keeping the player interested in his surroundings and figuring things on his own. (Mark Masters, 2014h)

There is a possibility that the player wants to play a certain area or a level again, just to see if he missed something. (Masters, 2014h) Because of this, it is important to get as much as possible from an area, rather than just placing objects here and there randomly. A well-made level gently forces the player to keep attention for anything important, thus making him value and treat the level scenery more than just a mere background. (Harris, 2007)

Keeping a good contrast between shapes, lighting and colours early is what makes the most of the level and thus guides the players eye where intended. (Hawkins, 2014, p.50) This can be achieved by making a blockout of placeholder objects, textures and basic lighting, which will set the mood for the level. By testing the level for action, the level undergoes several iteration processes to get the best gameplay. (Hawkins, 2012, p.43) This saves time from the graphics team, and they can then focus on their tasks without making extra mistakes. (Masters, 2014a)

8.2.1 Mood

Mood is a basic concept of emotions that the player can experience from playing a game. Excitement and fear can be considered as moods, but for example love and anger are more about certain emotional events which will pass away more quickly. Calculating and then aligning the mood with certain parts of the level creates a holistic experience for the player, and there is rarely only one mood to experience in the game. For example horror games mood changes between building up tension, horror and upright terror. (Gard, 2010a)

To create a good mood flow from another is to generate a mood map for the level (See figure 61). It can be very simple or very detailed, but the main purpose of it is to see if there are any problems between how well the level content and the mood order goes with the emotional experience. (Gard, 2010a) It is worth noting that different people feel different emotions even though the mood might be generally familiar and understandable, so it is important to playtest the level with several people. (Nutt, 2011)

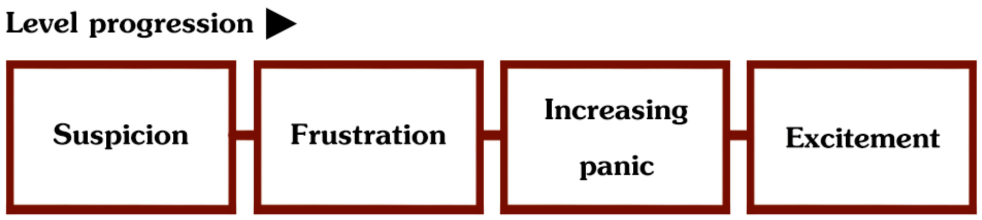


Figure 61. A simple example of a mood map.

8.2.2 Silhouette

A silhouette means that an object filled with only one colour, usually black so its interior details will not be seen, which is then projected against a white background. Making the silhouette first when designing objects or levels is a good way to see the general form, scale and proportions of the object. (Kennedy, 2013, p.36) Varying these three right creates an interesting, understandable silhouette even when seen far away (See figure 62). When making stylised graphics, it is also okay to exaggerate some features parts when needed. (Hawkins, 2012, p.119) Important elements, such as a tower in a large castle, should have some negative space around it, so that it looks clear and distinguishable from the rest of the object, which is also shown in the figure 62. (Lovato, 2015) A good silhouette is memorable, unique and give off more information than its size, for example a building can feel a lot more dangerous by just having a spiky silhouette. (McEntee, 2012)

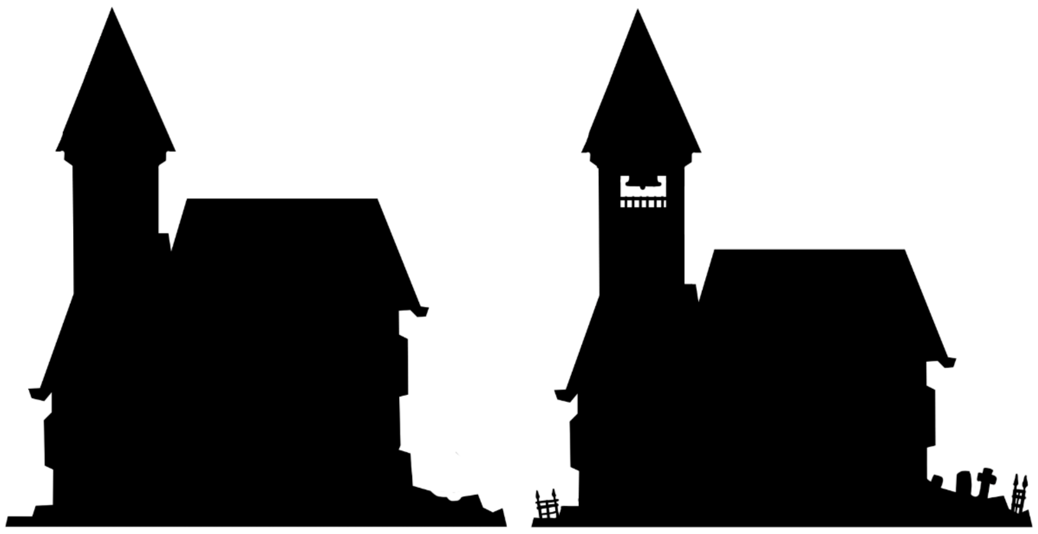


Figure 62. Comparison of two different silhouettes: The right silhouette of a church seems more understandable than the left one, and the tower also has more negative space around it.

But what silhouette, or basically a shape can be considered dangerous or welcoming? As a classical standpoint on aesthetics, each of the three primitive shapes, circle, square and triangle, are associated with different psychological characteristics (See figure 63). Circle shape is soft and welcoming, but it can also mean dynamic as a round shape tends to roll around on flat surfaces. Square shape is stable and strong, and strongly suggests of a sturdy, safe place if seen as a building. Lastly, triangle shape is aggressive and sharp, meaning it can hurt when touched. When making a silhouette, using these three shapes right will largely benefit for a game’s level design. (Solarski, 2013a)

Figure 63. The most basic shapes.

8.2.3 Light

Rather than giving instructions where to go, the player can be gently directed towards the goal by the correct use of lighting. For example a spotlight can illuminate a doorway or an object, which will then catch the player’s eyes and he will hurry to see if there is anything important in the light. (Masters, 2014h) In all cases though, the light should have some sort of source, because lights without particular source will only confuse the player. (Masters, 2014f)

Lighting affects the mood the most, because an environment with only little lighting feels menacing because of all the dark shadows, whereas the same environment with bright day light is not scary at all. It is easier to make a lighting for a movie to set the right mood, but for games lighting needs more planning because for example of the multiple camera views and the surface reflections certain environments might have in 3D-games. (Masters, 2014f)

It is recommended to place a basic lighting in an early stage of a level development, because it helps greatly in asset production. Textures and shapes against light are seen correctly, which saves production time from the graphics team when they are polishing them. The lighting does not have to be final, but should at least convey the basic feeling of the mood. (Hawkins, 2014, p.48)

8.2.4 Colour

If used correctly, colours can be used to guide the player through a level, even if the colour scheme is realistically desaturated. Placing colourful items or details into the level attracts the eye, if the overall contrast is high enough between the environment’s colour and an object’s colour. (Lovato, 2015) The colours used in the level should be uniformly chosen with each other, as a bad colour scheme is disturbing to the eye. (Maxinow, 2012)

Colour palette is one of the first things that a game development team should make for a certain level and stick with it the whole production process. The palette only needs some adjustments when it does not feel authentic with the level. (Masters, 2014c) It defines the right mood for the whole level, enhances the storytelling and evokes a desired emotional impact through visuals. (Hawkins, 2012, pp. 82-83)

For example in the development of Uncharted 3 action-adventure game, which is heavily based on the desert environment, the development team was concerned about the final levels' colour palette. Because the game already had so many environments with a warm palette, the team was given a note that they could not use any more warm colours. In the end, they went with blue, gold, warm white and some cooler red tones to add some dynamics to the textures. The correct use of colours that work harmoniously with the rest of the palette and a dash of unexpected colours gives the level a kick it needs. (Hawkins, 2012, pp. 82-83)

8.2.5 Points of interest

An environment is a hierarchy of details, where difference and contrast between shapes, colours, details, size, etc. to create a composition to attract the player to the focus point of it. The environment will look flat, if there are details all over place. (Jansson, 2012) Because of this, points of interest, in short for the locations that the developer wants the player to visit, should not be lumped together. The environment designer should not also put them too far away from each other, as the player might feel bored when going huge distances. (Van Spronsen, 2015)

8.2.6 Verticality versus horizontality

One of the many problems which can be encountered when making a level is that it becomes too flat. Those kind of levels lack authenticity both in visuals and gameplay. This can be avoided by creating a bit more height difference into the level, so that for example the player has a possibility to see how many enemies there might be ahead or where he should go next. (Hawkins, 2014, p.101) This way the player can plan which route to take, if there is any cover, etc. In multiplayer games adding height differences will keep the player in action because he has to keep looking up and down constantly in addition to horizontal vision (See figure 64). (Hawkins, 2012, p.19) Adding verticality can also pose some unexpected problems, like the player reaching to a place he is not supposed to. Because of this it is important to take care of the level's locations from both design and artistic aspect so that they would make sense. (Van Spronsen, 2015)

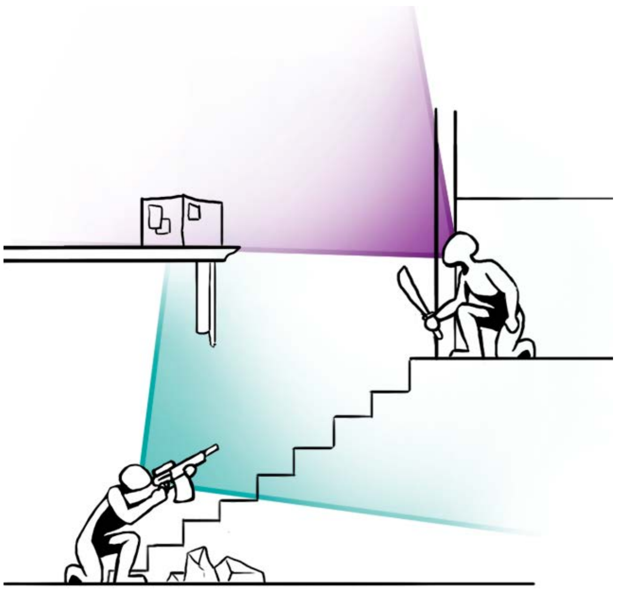


Figure 64. Player's field of vision in different levels.

8.2.7 Composition

A pleasing arrangement is created by a correct composition of shapes, colours and light, which relationships are disparate from each other. This gives the player a meaning to compare this imbalance, which requires the player to move their eyes. Controlling that eye movement, what will and will not attract the eye, is the goal of a good composition. (Santos, 2015)

Optionally a composition could be made based on the Golden ratio or by the rule of thirds, because they offer an easy, mathematical grid which to follow (See figure 65). Rule of thirds is loosely based the Golden ratio, and is one of the easiest to use (Creative Bloq, 2014). The image is divided into thirds vertically and horizontally, and placing details into the created lines or crossings of them in an unbalanced way creates an eye attractive arrangement. (Santos, 2015) Placing key points to the dead centre of the view creates a composition from which the player is hard to move their eyes from. (Solarski, 2013b)

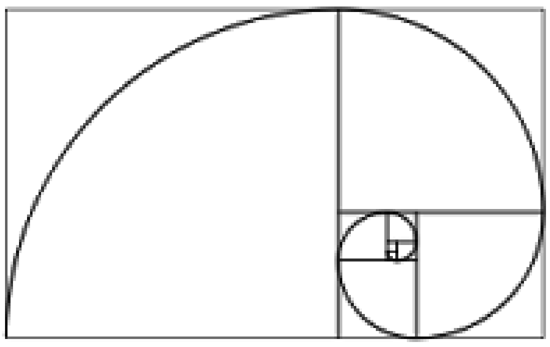


Figure 65. The golden ratio. (Gurney, 2010)

Leaving some space around assets also makes them easy to read, look unique and somewhat important to check out. (Lovato, 2015) This uniqueness of a focal point can also be enhanced by the use of implied lines. Implied lines are not actual lines, but the contrast of different values and colours that the eye will naturally follow. For example an image could have several lines, which all end to a single focal point. These implied lines can be created for example by the use of sharp objects that point to the focal point (See figure 66). (Santos, 2015)

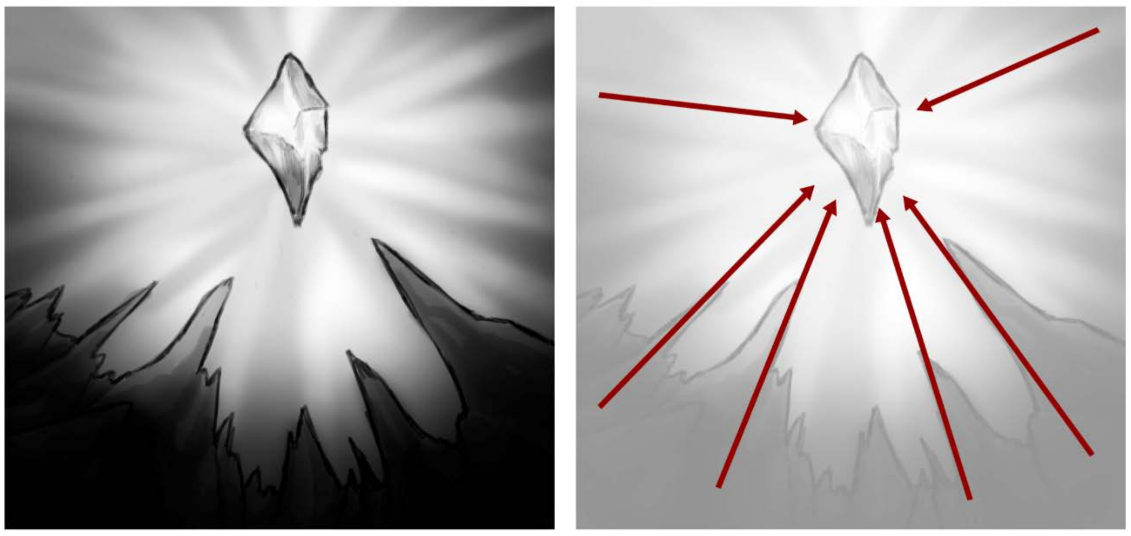


Figure 66. Usage of implied lines.

Objects like huge rocks can look out of place it there are no supporting objects around them. A simple rule of placing huge, medium and small sized objects together creates good compositions of small areas into a level. An example of an areas' objects, also called as props, debris could be seen as a small props, chairs and furniture as medium sized props and then the buildings and trees as large sized props. (Hawkins, 2014, p.105)

8.2.8 In-level storytelling and schema

Levels have a great potential to tell many background stories and to enhance mood, regardless of the main story of the game. (Gard, 2010b) In-level storytelling does not disrupt the player's gameplay for example with a cut-scene, but communicates the story through a smart placement and design of the objects and environment. (Solarski, 2013b)

Schema is an object's most general concept or a representation of reality that is already presented in the mind. No-one has the same schema as the other. This concept has to be extremely clear and visible, when making environments for games. This is because of the cases where the player is already familiar with a real life place that is presented in a game, he has his own schema of that place. (Gard, 2010b)

Inaccuracies like American dumpsters in European city really feel off to the European player, just because he does not recognise dumpster as something that still might have the same first level schema. The objects in an environment should be context-relevant, buildings architecturally correct like shown in the figure 67 and the level already lived by somebody if the game is supposed to be realistic. (Gard, 2010b)

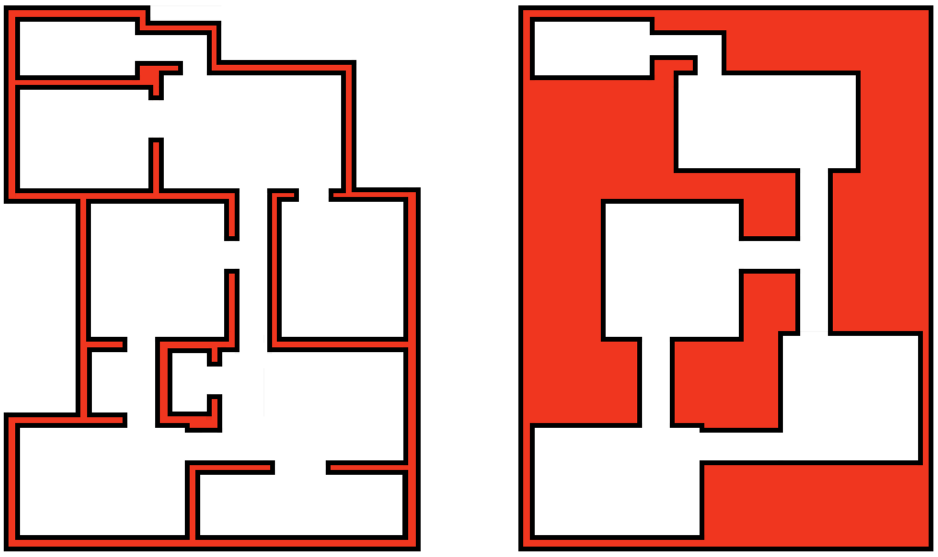


Figure 67. Comparison of a good and bad architectural levels' layouts. Red colour indicates that the structure is solid.

8.2.9 Level flow

A player should have some kind of motivation through the whole game, or he will easily get bored. Additionally a game that is only about action and where there is no time to take a breath will only exhaust the player, so it is a matter of balancing the level flow. To calculate the level flow, the developer can make a flow chart that for example simply describes the places of the level how they are played linearly and the amount of events, such as action and puzzle (See figure 68). Even the mixture of gameplay should not be the same through the whole game, because it can get pretty dull quickly. For example changing the movement of the player from walking to driving is a good, new experience. (Gard, 2010a)

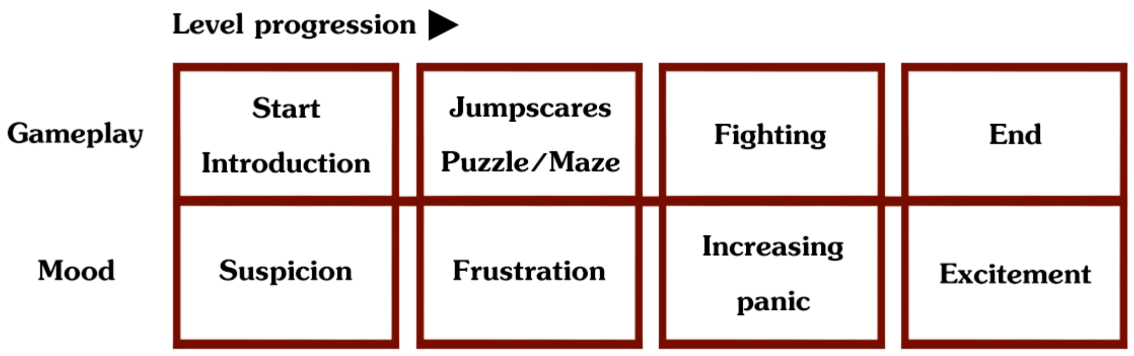


Figure 68. An example of a flow chart for a level.

The flow chart can be as complex as the developer wants it to be and it can be for example combined with a mood map. But it should never be too long, which depends totally of the game genre. The flow of the level and player’s actions can be manipulated by multiple actors. The level can show critical locations or other information at the start of the level to show what the player needs to know or where to go. There could be problems for which the solution can be somewhere else, for example a locked door that needs several keys to open it so that the player can progress further. Limiting the player’s progress can also be called as player gating. (Gard, 2010c)

The challenge in designing a good level flow is not making it 100% obvious to the player. (Gard, 2010c) For example putting an exit point of the level hidden from the sight will force the player to look around for it in the level. If the level is full of monsters, the player would not just run around aimlessly looking for the exit as it is too risky. Any area of the game should be fully utilized. There are situations where the player will not commit to a new area appropriately, but use the previous, already checked zone as their battleground when fighting new enemies. (Hawkins, 2012, p. 19)

9 ASSET PRODUCTION

Based by the planning, researching and sketching of the individual objects, making of the level layouts, colour palettes, etc. it is still best to start simply when developing the in-game assets and playtest them through the development process. Each detail the graphics artist adds has to be carefully added, and if the playtesting shows some problems whether they are about gameplay or graphics, it is best to take a step back and calculate what went wrong. (Hawkins, 2014, p.30)

9.1 Rules and logic

The idea of rules exist for a reason, and they should not be bent easily after they have been decided. In big game companies rules should be used mutually between all development teams, so that there will not be any problems when those teams shift with each other or the development process will not stop when starting a new project. (Burgess & Purkeypile, 2013) However, when making a new set of rules for a project, these newly set standards should be tested as early as possible of the game development, because it will be only harder to change the rules and the assets based on them later. (Hawkins, 2014, p.25) Having as many “doors” open as possible in the game development is not the way to get assets done. (Hawkins, 2012, p.31) However, the same thing goes for finding a one rule that would work with all the assets. (Van Spronsen, 2014)

9.1.1 Naming convention

A good naming convention is about making assets’ names readable to a complete outsider and then consistently using the system everywhere and anytime of game development, regardless if there is a new game to work on. This is because in game development studios the team sizes can suddenly change and the level designer must have an ability to look a certain asset in a timely manner from a collection of hundreds, even thousands of assets. (Burgess & Purkeypile, 2013) Naming convention is also applicable to folder structures. (Klafke, n.d.)

Planning and naming the pieces in a way, which will line them up nicely in an alphabetical list, can be a tedious task when the names start to have more than three different codes in them. Naming should not be based on generality like “building” or “house,” because they get easily duplicated and are hard to separate from each other (Kuzminova, 2009). Joel Burgess (2013) gave a good example in his blog about a not-so-obvious naming convention about naming hallways: “1way”, “2way”, “3way” and then “4way” line up in a list extremely well, and because of this, it is justified to do if understood completely. Adding a suffix to the end of the assets name such as “01”, “02”, “03”, etc. is wise when making pieces that have the same footprint and purpose, but are visually different. Because of this, they can be easily swapped in the game editor just by changing one number. (Burgess & Purkeypile, 2013)

9.1.2 World dimensions

Planning out the scale and dimension of the assets early ease the general game development. It is important to figure out like how far and high the player can jump, are there any covers in the game, from which height the player will die if he falls, etc. because they will determine the dimensions of assets. There are some general points to know about sizes to prevent later problems (See figure 69). Doors should have the same uniform size through the game, because combining rooms together becomes much easier and the team behind animation and AI, meaning artificial intelligence, have a fixed size to work from. The narrowest space in the game should also be a minimum of two characters, so that the AI pathing works nicely. The maximum steep of an incline is basically determined by the look of the character’s animation. (Burgess & Purkeypile, 2013) Animations should be generally kept in mind when making environment assets, because of the multiple ways a character can interact with them. For example, the development team might suddenly decide to add the ability to sit on chairs. (Perry, 2002)

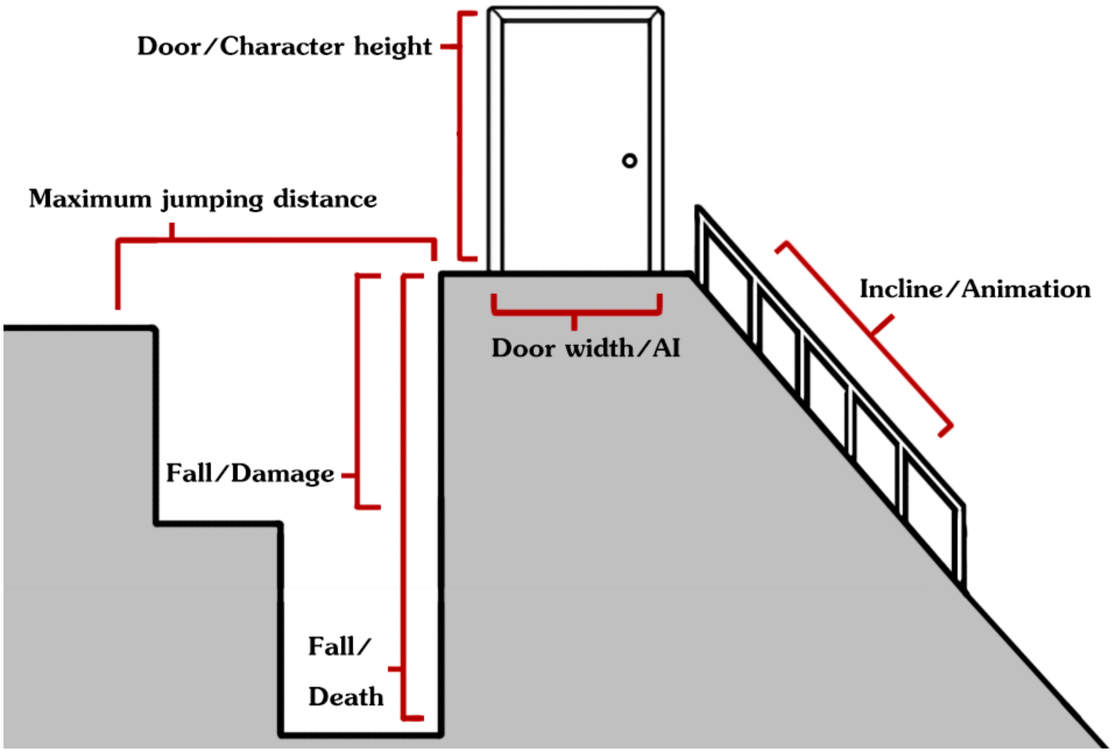


Figure 69. Dimension planning for a game.

To prevent problems like intersecting assets and bad proportions, there is also a technique called whiteboxing or in other name, grayboxing. Whiteboxing is like sketching, but in 3D space, where the simple shapes are combined together like they would be seen in the game. The created combination is then then put to the game engine to see what kind of dimension it occupies and how it might intersect with other assets, after which the object goes through several polishing passes. (Almost Human Ltd, 2014a)

9.1.3 Footprint

Footprint is the space and full bounds of an object. It is widely used in measuring correct grid and asset snapping when making kits and other tileable pieces (See figure 70). Having the same footprint, or at least multiples of it between different pieces prevents issues like gaps, overlapping walls, etc. A tileable piece’s seams can be on the edge of the footprint, but otherwise it is not recommendable to build at the footprint’s edge to prevent overlapping issues. In short, pieces should be structurally believable and have thickness both horizontally and vertically. (Burgess & Purkeypile, 2013) Using the same footprint between pieces adds the possibility to mix and match different looking objects to get more variety in an environment, because an artist is basically free to do anything inside the footprint (Van Spronsen, 2014)

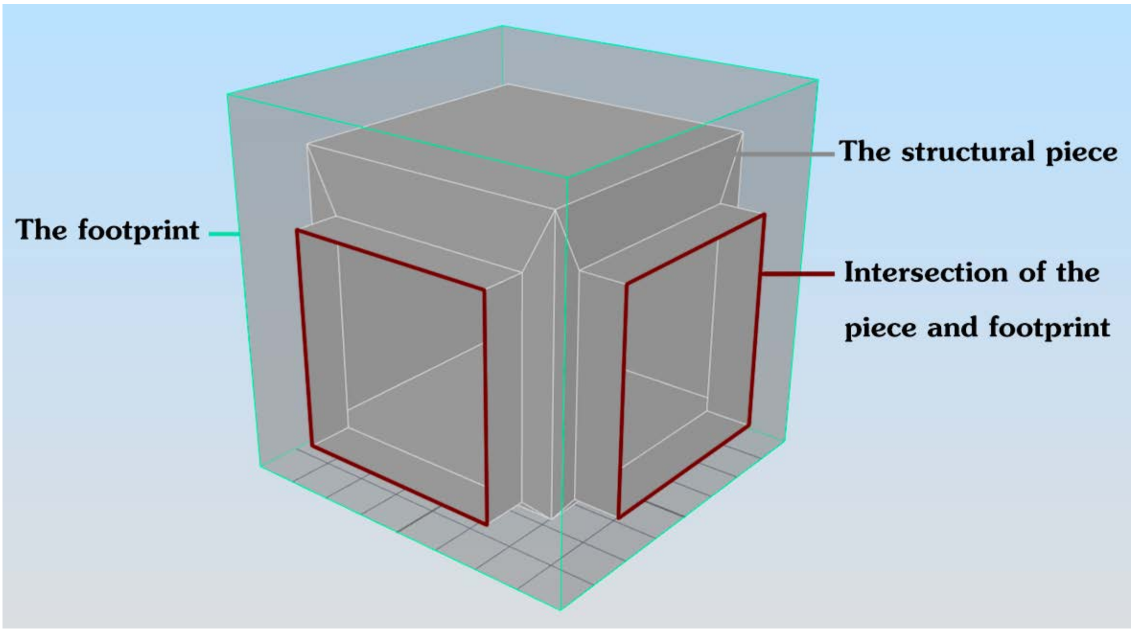


Figure 70. An example of a footprint.

9.1.4 Grid Snapping

Grip snapping is a great way for the designers to quickly create new levels, because at 90 degree angles room pieces like walls, floors, etc. line up quickly and easily. Grid snapping settings should be as large as possible and at the power of two, because level designers might also need smaller sized grids to work with. This working method also shows obvious errors like gaps between assets. (Burgess & Purkeypile, 2013)

Unfortunately grid snapping does not work well with organic looking environments. To make it a bit easier, there are things like snapping to a reference, in which the randomly rotated object will make a new grip which to work with (See figure 71). These two different grids unfortunately create gaps between each other, and create an extra problem of capping them. (Burgess & Purkeypile, 2013)

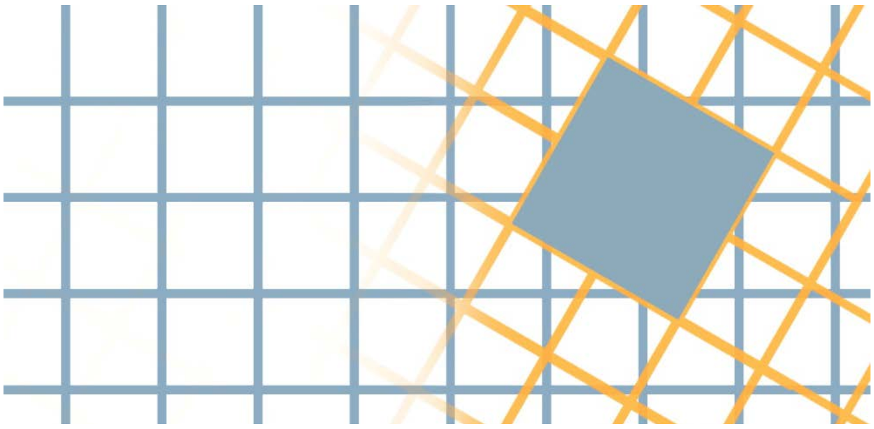


Figure 71. On the left a general grid, and on the right a new grid generated from the object's orientation.

9.1.5 Pivot point placement

Pivot point of an object is the origin from which the software calculates how the object is rotated, scaled and moved. (Govil-Pai, 2006, p.105) For example a windmills wings are rotated from the connecting point of the wings and the engine. Having a correct pivot point in an asset will decrease the level designer's work, because he does not have to spend time on rotating and placing objects by hand.

Objects throughout the game should also have a general rule for the pivot point placements. (Hawkins, 2014, pp.27-28) It prevents reworking the pivot points of the previously made assets, which can be a huge problem because in a heavily instanced project the already finished levels need manual editing after editing existing pivot points. (Burgess & Purkeypile, 2013)

Not all the objects have the same optimal pivot point placements. Things like pipes should have their pivots at the edge, walls at the bottom corner and individual assets like set dressing objects at the bottom centre. Walls' pivot point at the bottom corner is because they are generally easier to rotate and place on the grid. (Burgess & Purkeypile, 2013) Generally speaking a pivot point placement should be at one of the plane extremities. (Klafke, n.d.)

9.2 Modularity

Modularity is good for games that need tremendous amounts of details and need to be performance friendly at the same time. (Hawkins, 2012, p.65) However, modularity can also be seen as the opposite from uniqueness. Rather than making a whole custom level of geometry and textures, the level is built from several duplicates of the same assets, which saves from time, need of reworking, performance and creates consistency through the whole level. The ability to edit the assets and to see the results almost immediately in the game is also one of the great perks in modularity, because the game development team usually has strict deadlines to follow. (Perry, 2002) Modularity is not the only working method available, and can and should be used with the unique assets to create a better illusion of realism (Klafke, n.d.).

Modular pieces are the most seen objects and structures in the game, and thus should be planned, created first and spent the most time with when making iterations and visual variations. Reusability is one keyword for modularity, for example in organic systems like caves, the floor assets could be used as a ceiling and a fence could be used as a strengthening structure for walls. (Perry, 2002) Modular pieces should be compared with other assets from time to time as they might look good as in individual pieces but not good as in groups with other assets (Klafke, n.d.).

9.2.1 Shell-based building system

Shell-based building system relies heavily in the assets free placement, and breaks up the otherwise standard, similar looking environments. If for example a cave uses a modular kit system, the cave can easily get a rectangular shape which does not look natural. Using shell pieces like pillars and balconies to shave off the 90 degree corners gives the cave a more organic feeling. This could also be done with fantasy-like buildings, where the proportions does not always have to be 100% accurate. Visually it is hard to get a similar looking areas with a shell-based system, because of the infinite possibilities these kind of assets could be combined. Also the amount of time and resources needed to make shell pieces is minimal, and with cases like bad looking intersections and seams, it is best to use some concealment pieces. (Joel Burgess & Purkeypile, 2013)

9.2.2 Concealment pieces

Creating some concealment pieces helps the overall production of the environment, especially with organic looking areas that need transitioning between each other. They are helpful at places like walls or ground assets that are usually placed at odd, natural angles, which can cause seams and gaps between the objects. Concealment pieces also work well with streams of water, which need a natural looking starting and ending points (See figure 72). Covering up various kinds of level work is generally done by using for example large stone slabs, foliage, fallen tree trunks and other generic objects. (Perry, 2002)

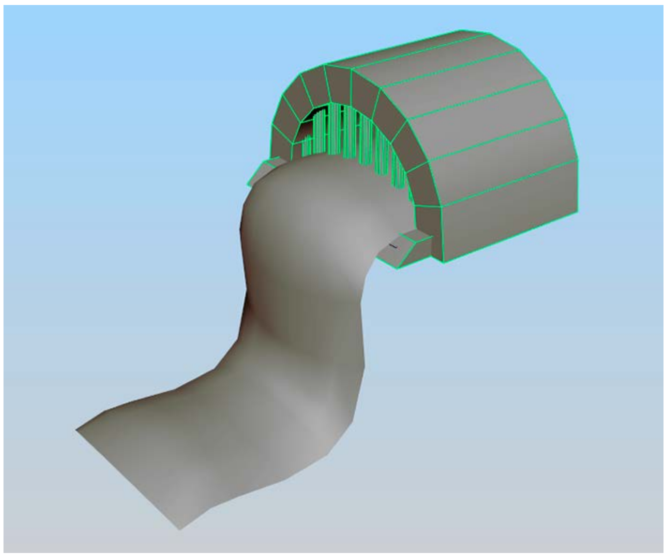


Figure 72. An example of a concealment piece for a water stream.

9.2.3 Hero pieces

Hero pieces are fancy looking accessories that the player might see only once or twice in a game. At first they might seem impressive and important for the player's experience, but which will be forgotten later in the game. Artists should not be too focused on hero pieces when making environment assets at an early phase of the development, because for example a regular wall asset can be seen hundreds of times and thus should have more of the polishing time. (Joel Burgess & Purkeypile, 2013)

9.3 Kits

Kit is a system which has multiple parts that can be used together to create vast amount of environments and also gives an opportunity for fast iteration (See figure 73). An example kit could be a collection of pieces to create a cave or a house, and the parts can be of multiple sizes (See figure 74). It acts as a great toolkit, because everything is basically standardized. Unfortunately they are complicated to make, because artists are required to have a deep understanding of the technical properties behind kits. (Burgess & Purkeypile, 2013)

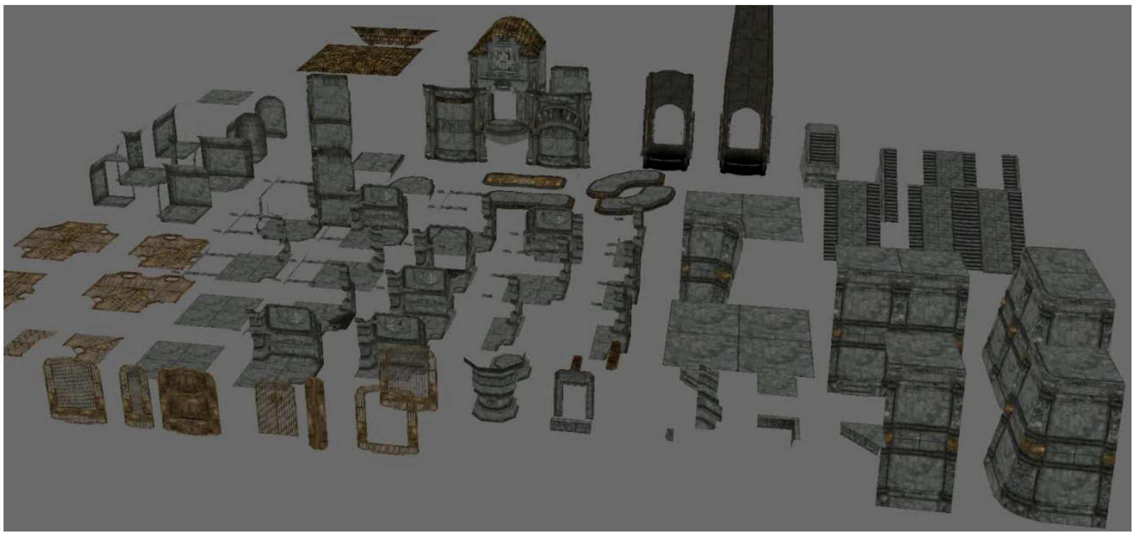


Figure 73. A dwarven kit for Elders' Scrolls V: Skyrim. Adapted by author from (Joel Burgess & Purkeypile, 2013)

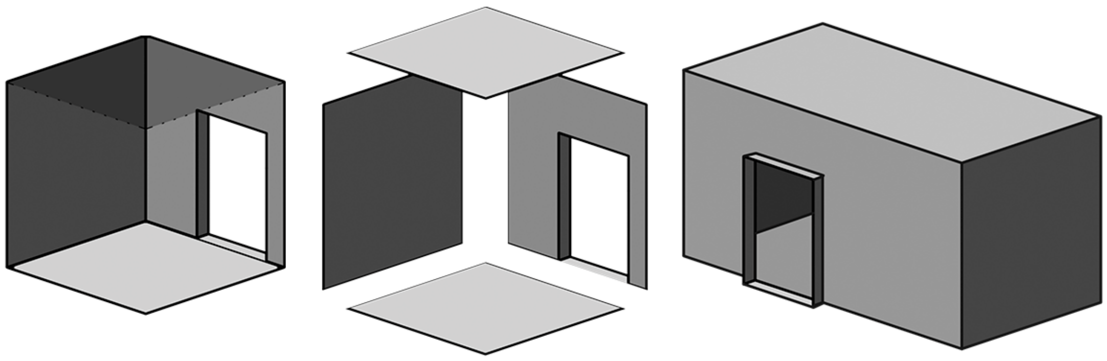


Figure 74. Different kit sizes: half-open boxes, planes and rooms. Each of them have their own advantages and disadvantages in regards of footprints and grid snapping.

The process of identifying problems and fixing them within a kit is also time consuming, as they usually present new problems in a heavily instantiated environment that uses kit pieces as its core. Because of this, and for getting the designers something to play with, it is important to get the kit into a functional state in an early process of development. Further updating of the kit is not rushed, is mainly artistic and does not interrupt the design process as the basic functions of the kit pieces stays the same. (Burgess & Purkeypile, 2013)

9.3.1 Overall creation of a kit

There are basically five steps to create a working kit. First, the kit is planned and its functionality is calculated based on the environments' and level design's needs. No art assets are yet created, but rather questions related to the kit are answered by both the design and art team. For example visuals and re-usability are decided and reference content is gathered in this phase. (Burgess & Purkeypile, 2013)

In the second phase, the decided functionality gets tested so that possible reworking is avoided. First, very basic kit pieces are created for the level designers, who then play around with them to calculate rules like naming conventions, proportions, etc. Because of this, the kit pieces do not have any textures and the geometry is very simple, almost box-like (See figure 74). By quickly testing, solving problems and iterating the rules, this phase is completed relatively quickly. Testing the kit pieces has many methods. For example the kit pieces can be looped back on itself or stacked on top of another to see if the footprint of the kit pieces work correctly. Problems with the core kit should be solved without exceptions, because they will only increase the workload of the overall level design when for example a "patch-up" piece is used for the gaps between certain wall pieces. (Burgess & Purkeypile, 2013)

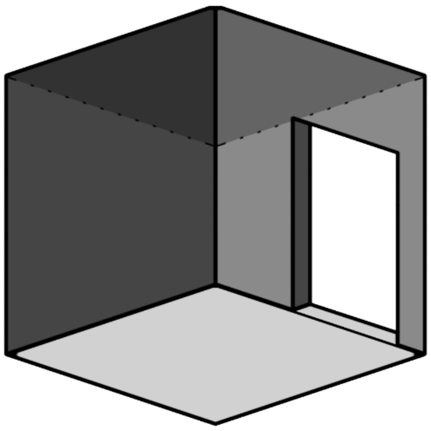


Figure 75. A rudimentary kit piece.

The most common, primary kit pieces are figured out and edited in the geometry level in the third phase. This means by adding and iterating the most impactful, tiling visual elements, for example the wooden beams on walls, so that their new footprint seams could be tested (See figure 75). Their final pivot point placements should also be tested and accepted by the rest of the team. (Burgess & Purkeypile, 2013)

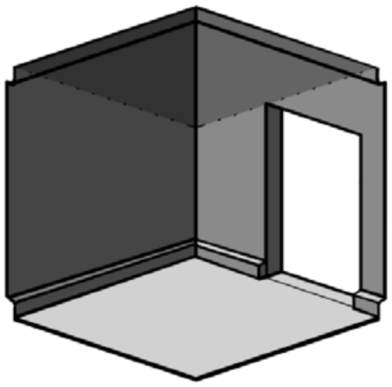


Figure 76. Adding the impactful details on a kit piece.

Fourth phase is about the kit's development visually. While avoiding to affect the functionality of the kit, less-critical pieces are added, visual variants are created and finally the textures for the assets are generated (See figure 76). Creating one final kit piece and to get that accepted early, prevents reworking the whole kit. This is because developing the visuals for the whole kit at the same time poses the possibility to waste time when the kit's visuals do not get accepted. To put it shortly, this phase takes a lot of development time as actual graphics will be made, iterated and tested by the rest of the team. (Burgess & Purkeypile, 2013)

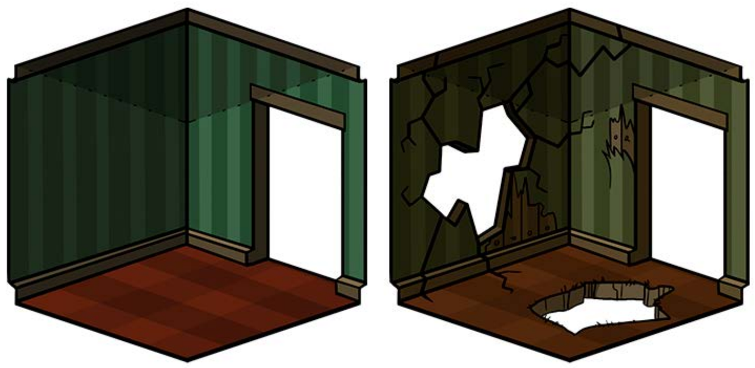


Figure 77. Adding the textures and making of visual variants.

The last phase is the polishing phase, which lasts through the rest of the development process. The visuals are polished, artists test the levels themselves, fix issues and add additional kit pieces when needed. However, the additional kit pieces should be thought carefully, as there might be a solution in the kit already. (Burgess & Purkeypile, 2013)

9.3.2 Kit Knowledge

There are some terms about kits that need describing. Most of the artists do not like making kits at all because it can mean less time on making the assets and more time thinking about technical details. (Burgess & Purkeypile, 2013)

Kit-bashing

Kit-bashing can be called as a method in which the assets are combined in a yet unpresented way, to create something totally new. (Almost Human, 2011) It is a very organic process, as there are not many limitations other than common sense. (Sheffield, 2013) Kit-bashing can create a very specific, new identity to an area, which normally would be seen for example only as a military area that has rubble, tanks, medic tents and such. Mix and matching said military area kit with for example an Asian kit can result in a totally different mood. (Burgess & Purkeypile, 2013)

A sub-kit

Supporting sub-kits are needed to make a compelling kit. Their functionality and the amount of time spend on making a sub-kit is chosen by the designer and the artist, so that the overall kit would not get bloated with extra kit pieces. The scope of the sub-kit can vary greatly: some general and most used sub-kits can have as much as over 50 pieces where as a smaller, not so much used kits can have only a handful. As an example, a regular building kit could have sub-kits that are specifically for a small room, large hallway, etc. (Burgess & Purkeypile, 2013)

Small scope kits

Small scope kits contain many pieces, usually small to medium size, to create visual variety, especially to organic environments. They are cheap to make and usually work well with any kind of surface that has the same visual theme. (Burgess & Purkeypile, 2013)

Glue kits are small-scope kits that answer to needs like transitioning different visual themed kits and sub-kits, covering up hallways that are not snapped together and other similar situations. To add more verticality in to an environment, another small scope kit called a platform kit is quite useful because it does not need any complex thinking or logic. The kit contains pieces for example of stone slabs of all sizes that can be placed on top of other surfaces to create interesting areas. (Burgess & Purkeypile, 2013)

Kit tiling, directionality and asymmetry

When talking about 3D-assets, an all axis tiling kit is very challenging to make, even though it is beneficial to use not only horizontally but also vertically. Room kits usually tile on two axis and hallways on one. (Burgess & Purkeypile, 2013) A directionally restricted kits are especially useful with organic environments, but require a lot of extra work. To compare it to a traditional kit, which usually can be rotated to any direction, a directionally restricted kit needs some extra pieces to work properly. They can be used to make asymmetrical halls and hallways, an all-around room of cardinal directions, etc. However, hallways that do not snap together due to having a different direction will need a kit piece that bridges the gap between them. This can be also called a “de-twist” piece, because it flips the sides of the meeting pieces. (Burgess & Purkeypile, 2013)

9.4 Procedural Generation

In procedural generation, the game’s software calculates the placement of the assets in the game. It can calculate infinite amount of environments randomly, so there is always something new for the player to look for. Game developer like Mojang, which made Minecraft (See figure 77), has done this extremely well with its game environment. Procedural generation gives the environment team more time to work on something else and important, for example creating more assets. This kind of generation does not have to be, and should not be, fully random, as the designer can give the software a set of rules which to follow. There is also always the possibility to make a procedural environment with hand-crafted areas, be they large or small. (Extra Credits, 2015b)

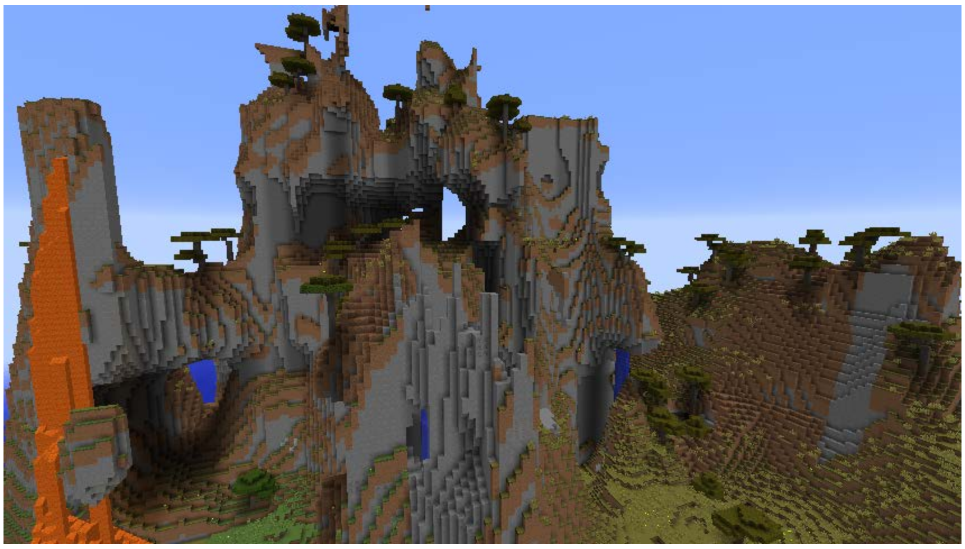


Figure 78. Minecraft screenshot. (SeargeDP, 2015)

However, creating the code behind the procedural generation is very time consuming and can present more bugs and issues than in an environment created by a level designer or an artist. The general issues are about generating an environment where the player can not progress having for example no accessible exit points. There is also the issue about making an environment which is really unique and has a hand-crafted experience. Procedural generation tends to be repetitive in time, which is why game developers, who has a certain story or narrative in mind for the game, do not use it at all. Additionally, procedural environment generation is not efficient when making a small scale of levels. (Extra Credits, 2015b)

10 BUILDING THE ENVIRONMENT

There are generally three steps in building a game environment: blockout, build and then polish. They can also be called as art passes, because each of them brings the development closer to the final result (Van Spronsen, 2015). Blockout and building passes can shift together a lot, as there are two types of scenarios on how to start working. In the first scenario the designers start making the level without any actual artwork but with primitive shapes that the game editor provides, after which it is passed to art team for building and polish. This is also called as grayboxing (Nutt, 2012). In the second scenario the designers work with the artists' asset set, which is not that different from game editor assets set at first, but which then evolves throughout the game development. (Provost, 2003b)

10.1 BBP: Blockout, build and polish

Based on the design process before the asset production, the designers start blocking out the elements using primitive assets and shapes (Provost, 2003b). While starting large, it is important to add simple lighting to bring out a pleasing composition of shapes and silhouette, and to check out that the lighting is right for the desired mood. Rather than spending an enormous time on every detail, it is best to work quickly and focus the attention on the most visible elements. (Hawkins, 2014, p.48) While the designers are working on the blockout, the artists can and should spend time on making standalone assets, tiling textures and materials that the designers can use in their rough blockout (McGrath, 2008).

Once the base blockout has been made, the level is then passed several times between designers and artists, in which they iterate the placements of in-game items, playtest the level and generally optimize everything from visuals to gameplay. (Provost, 2003b) Big elements like buildings and foliage are slowly built on top of the blockout by the mixed use of modular and unique assets. (Van Spronsen, 2014)

Polish is the last beautification and little tweaking that bring out the visuals to their whole glory. Polish is usually made in several iteration passes, which contain things like set dressing, post-processing and light adjustments, last small iterations of models and textures, tiny bits of optimizations, etc. (Christoph, 2015)

10.2 Dividing areas and scene complexity

Performance is not always about optimizing game's individual assets. When there are several objects in the player's view, performance tends to go down. The visibility spectrum is all the visible space from the player's location at all given time, and the larger the space in the view, harder it gets for the computer to render it. Rather than reducing the polygons of the models or the size of the textures, dividing areas to a certain visibility spectrum to reduce the scene complexity is a good way to bring the performance back up (See figure 78). (Provost, 2003b)

Dividing of areas can be done in several ways: doors to close off rooms, transition zones and tight curved-up spaces between larger areas to block out the view. There is also fog, depth-of-field and other post-processing effects to limit the view of the player, if the performance still tends to go down. (Provost, 2003b) Blocking the line of sight is also a method of designing levels, so that the player is for example actively keeping track of the surrounding enemies (Hawkins, 2012, p.18).

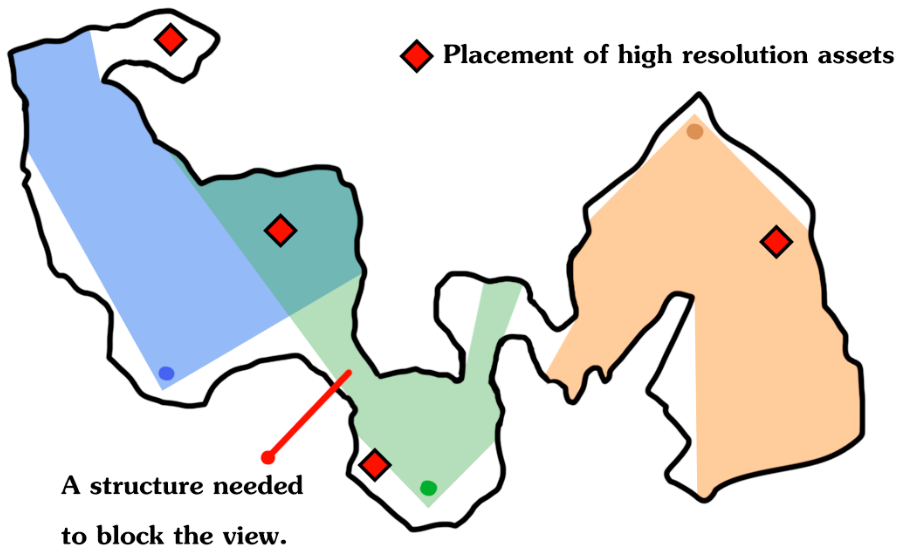


Figure 79. Dividing areas by the visibility spectrum.

Other matter about scene complexity is about vertex and texture density. It is performance friendly to distribute high-resolution models evenly across the space and/or placing them in small visibility spectrum areas. Same matters with the texture density, where the developer has to calculate how much texture memory he can use in a given area, and also distribute it evenly. (Provost, 2003b)

10.3 Asset placement

Developer should avoid placing arbitrary architecture and objects that are only designed for the player rather than the inhabitants of the game (Gard, 2010b). It is also important to start with the biggest elements, because players are going to see them first in an environment. After that it is just a balancing act of which smaller details to work on. (Nutt, 2012).

10.3.1 Set dressing

Set dressing is about decorating a certain set with objects so that it looks like it has been lived-in. All the large and small accessories like furniture or vegetation are what break the appearance of emptiness, and fills the set with a background story. Not all areas can look the same, as the purpose and the inhabitants vary from set to set: for example a child’s room is usually more messy and full of toys than an adult’s room and in a hospital a storage room could have medicine where the other storage room has only patients’ files. (Marshall, 2014a)

Unless the set is about a clean, organised environment, objects are rarely aligned perfectly. Every object should be kilted off just a little to give a bit more realism to the set. (Marshall, 2014a) In some game engines, there is a function which lets the developer to “drop” an object from the air by the use of physics, which will then set and stay on that fallen position. This saves a lot of time from the general hand placing items randomly. (Hawkins, 2014, p.105)

Set dressing is also very useful for covering up tiling walls and floors, and two identical areas can be totally different with the accessories placed in them, if they have enough visual impact. Set dressing pieces also should be created in a manner of modularity, so that they could be combined in multiple ways and thus giving more diversity to the environment. (Perry, 2002)

The players do not usually look directly at set dressing, as the elements are not directly gameplay related. Some set dressing does not need collisions at all, for example the objects that are outside the playable area. But in cases where the player can bump into objects, it is best to give them at least a simple collision box, so that the physics or player collision can be simulated on them. Having a collision data to move certain small objects provide more visual interest than static, non-movable objects. (DeLeon, 2012)

10.3.2 Diversity

A player does not generally accept repetitive looking environments, as it gets easily boring to look at. Other than in technological sci-fi environments, this can be a huge problem to conquer. (Sheffield, 2013) With assets that are going to be duplicated in a level several times to get a better performance, especially buildings, this can be avoided by making all the sides different and then displaying the building from different angles to create some visual variety. (Masters, 2014h)

Unfortunately in almost all cases, the player is bound to see the obvious repetition in the environment, which then decreases the authenticity of the world. This can also be called as art fatigue, which is hard to minimize because changing the visuals and gameplay to keep things fresh does not always work. There is also a possibility that the experimentation in to something new will confuse the player. (Burgess & Purkeypile, 2013)

11 CONCLUSION

Game environments are coming to an era, where everything can be physically constructed and destructed by the player, while still having a certain feel of authenticity. The ability to simulate accurate environments is coming closer with every invention of technology and because of this, level designers and environment artists will also have their hands full of work. Doors and gates are no longer locked, creating worlds of limitless details and ability to explore where ever the player might want to. Challenges in modularity and diversity will probably stay for a long time, but they will get smaller and smaller as time passes further.

This thesis could have dig deeper in already presented details to give a better sense of feeling how big and important the game environments creation can be from the artists' and designers' point of view. In addition to that statement, there are also a huge amount of other details about sound affecting the immersion of environments and programming that is required to make functional, processor lightweight environments. Saving time in the overall creation of games has become the first priorities in this demanding world of players. But with enough practise, creating game environments efficiently will become a natural process of planning and production.

REFERENCES

80.lv. (15.9.2015). World building and art direction of Destiny. Retrieved 16.9.2015 from: http://80.lv/articles/world-building-and-art-direction-of-destiny/

Almost Human Ltd. (8.9.2011). Building the dungeon. Retrieved 12.10.2015 from: http://www.grimrock.net/2011/09/08/building-the-dungeon/

Almost Human Ltd. (28.8.2014a). Making of Grimrock 2: Level building blocks. Retrieved 12.10.2015 from: http://www.grimrock.net/2014/08/28/making-of-grim-rock-2-level-building-blocks/

Almost Human Ltd. (7.3.2014b). Performance optimizations. Retrieved 12.10.2015 from: http://www.grimrock.net/2014/03/07/performance-optimizations/

Anhut, A. (4.10.2011). A look inside – Evaluating camera angles for immersion. Retrieved 2.11.2015 from: http://www.gamasutra.com/blogs/AnjinAn-hut/20110410/89304/A_Look_inside__Evaluating_Camera_Angles_For_Immer-sion.php

AssemblyTV seminars. (2015). ARTtech: Tricks used in background animations of Shadow Bug. [Video]. Retrieved 27.10.2015 from: http://archive.assem-bly.org/2015/seminars/arttech-tricks-used-in-background-animations-of-shadow-bug-by-assemblytv-seminars

Backus, K. (5.11.2013). Go beyond retro pixel art with flat shaded 3D in Unity. Retrieved 28.10.2015 from: http://gamedevelopment.tutsplus.com/articles/go-be-yond-retro-pixel-art-with-flat-shaded-3d-in-unity--gamedev-12259

Becerra, A. (11.9.2015). Creating medieval atmosphere in the level. Retrieved 21.10.2015 from: http://80.lv/articles/creating-medieval-atmosphere-in-the-level/

Blomberg, L. (16.5.2013). Sponsored: The world of Just Cause 2 – Using creative technology to build huge open landscapes. Retrieved 23.9.2015 from: http://www.gamasutra.com/view/feature/192007/spon-sored_the_world_of_just_cause_.php

Burgess, J., Purkeypile, N. (19.4.2013) Skyrim’s modular level design – GDC 2013 transcript. Originally presented at Game Developers Conference 2013. Retrieved 19.10.2015 from: http://blog.joelburgess.com/2013/04/skyrims-modular-level-design-gdc-2013.html

Burkart, B. (2.7.2015). 8 Secrets of great multiplayer maps from Ben Burkart. Retrieved 21.10.2015 from: http://80.lv/articles/8-secrets-of-a-great-multiplayer-map/

Cheng, L. C. (15.2.2015a). About level of detail (LOD) in game. Retrieved 17.9.2015 from: http://blog.gameartworkbook.com/game-art-theory/about-level-of-detail-lod-in-game/

Cheng, L. C. (18.1.2015b). Base mesh creation workflow with silhouette for games. Retrieved 17.9.2015 from: http://blog.gameartworkbook.com/3d-game-art-modeling/base-mesh-creation-workflow-with-silhouette-for-games/

Cheng, L. C. (1.5.2014a). How to reference a reference? Retrieved 17.9.2015 from: http://blog.gameartworkbook.com/game-art-theory/how-to-reference-a-reference/

Cheng, L. (28.4.2014b). Principles of 3D modelling for games. Retrieved 17.9.2015 from: http://blog.gameartworkbook.com/game-art-theory/principles-of-3d-game-art-modelling/

Courrèges, A. (2.11.2015). GTA V – Graphics study. Retrieved 13.11.2015 from: http://www.adriancourreges.com/blog/2015/11/02/gta-v-graphics-study/

Christoph. (24.9.2015). How a map for Albion Online is designed. Retrieved 21.10.2015 from: https://albiononline.com/en/news/map-design-albion

Creative Bloq. (9.6.2014). The designer’s guide to the golden ratio. Retrieved 22.10.2015 from: http://www.creativebloq.com/design/designers-guide-golden-ratio-12121546

Curtin, D. P. (2011). Sensors, pixels and image sizes: Pixels and screen display. Retrieved 12.11.2015 from: http://www.shortcourses.com/sensors/sensors1-11.html

DeLeon, C. (31.12.2012). Non-essential level art is essential. Retrieved 23.9.2015 from: http://www.hobbygamedev.com/int/non-essential-level-art/

Edge Staff. (3.9.2015). Why Skyrim’s most barren environment makes for a bountiful world. Retrieved 21.10.2015 from: http://www.gamesradar.com/why-skyrims-most-barren-environment-makes-bountiful-world/

Epic Games Inc. (n.d.a) Creating and using LODs. Retrieved 17.9.2015 from: https://docs.unrealengine.com/latest/INT/Engine/Content/Types/StaticMeshes/HowTo/LODs/index.html

Epic Games Inc. (n.d.b). Epic Games texturing guidelines. Retrieved 6.11.2015 from: http://udn.epicgames.com/Three/TexturingGuidelines.html

Epic Games Inc. (n.d.c). Foliage instanced meshes. Retrieved 5.11.2015 from: https://docs.unrealengine.com/latest/INT/Engine/Foliage/index.html

Epic Games Inc. (n.d.d). Material editor – How to animate UV coordinates. Retrieved 28.10.2015 from: https://docs.unrealengine.com/latest/INT/Engine/Rendering/Materials/HowTo/AnimatingUVCoords/index.html

Epic Games Inc. (n.d.e). Physically based materials. Retrieved 10.11.2015 from: https://docs.unrealengine.com/latest/INT/Engine/Rendering/Materials/PhysicallyBased/index.html

Epic Games Inc. (n.d.f.). Post-processing content examples. Retrieved 10.11.2015 from: https://docs.unrealengine.com/latest/INT/Resources/ContentExamples/PostProcessing/index.html

Epic Games Inc. (n.d.g). Unwrapping UVs for lightmaps. Retrieved 6.11.2015 from: https://udn.epicgames.com/Three/LightMapUnwrapping.html

Ericson, C. (2.1.2009). Optimizing the rendering of a particle system. Retrieved 9.10.2015 from: http://realtimecollisiondetection.net/blog/?p=91

Extra Credits. (23.9.2015a). Frame rate - How does frame rate affect gameplay? - Extra Credits. [Video]. Retrieved 27.10.2015 from:

https://www.youtube.com/watch?v=zL5kOyHWI_E&in-dex=8&list=PLhyKYa0YJ_5DZUWTC25vSZbJ6OCk1XB1p

Extra Credits. (22.7.2015b). Procedural generation - How games create infinite worlds - Extra Credits. [Video]. Retrieved 27.10.2015 from: https://www.youtube.com/watch?v=TgbuWfGeG2o&in-dex=24&list=PLhyKYa0YJ_5CciOENqQoB_QuZJ41r9sML

Extra Credits. (10.6.2015c). Exploration in games - Four ways players discover joy - Extra Credits. [Video]. Retrieved 27.10.2015 from: https://www.youtube.com/watch?v=FE7lDFAcb4Y&in-dex=18&list=PLhyKYa0YJ_5CciOENqQoB_QuZJ41r9sML

Extra Credits. (17.7.2014). Design club – Mark of the Ninja – Stealth games and visual cues. [Video]. Retrieved 27.10.2015 from: https://www.youtube.com/watch?v=6vJNqseX-rs&in-dex=4&list=PLhyKYa0YJ_5CH8BA8XcqReieXLFf4afI0

Fessler, D. (19.2.2014). Thoughts on modular animation. Retrieved 9.11.2015 from: http://danfessler.com/blog/thoughts-on-modular-animation

Forman, C. (6.2001). Isometric views in Director: Theory and game application. Retrieved 21.10.2015 from: http://director-online.dasdeck.com/buildArti-cle.php?id=952

FZDSCHOOL. (15.4.2011). Design cinema – Ep 40 – Fantasy Landscape. [Video]. Retrieved 4.11.2015 from: https://www.youtube.com/watch?v=xCmz3XuS6TI&in-dex=46&list=PLvNv1kRvuSwLYS2CkHTDS6-zVKSoUYzJO

FZDSCHOOL. (22.8.2012). Design cinema – Ep 56 – MMORPG Landscapes. [Video]. Retrieved 4.11.2015 from: https://www.youtube.com/watch?v=mmff8bBCYW4

Galamoth. (25.5.2015). The 2.5D. Retrieved 10.11.2015 from: http://www.gi-antbomb.com/25d/3015-660/

GameBuilder Inc. (2.7.2013). Vector or bitmap images in your game, why choose? Retrieved 12.11.2015 from: http://gamebuilderstudio.com/blog/91/Vector-Or-Bit-map-Images-In-Your-Game,-Why-Choose%3F

GamesRadar_ US (8.10.2010). Gaming's most important evolutions. Retrieved 22.9.2015 from: http://www.gamesradar.com/gamings-most-important-evolutions/

Gard, T. (27.5.2010a). Action adventure level design III: Pacing, content, and mood. Retrieved 23.9.2015 from: http://www.gamasutra.com/view/fea-ture/133829/action_adventure_level_design_.php

Gard, T. (7.5.2010b). Action adventure level design: Kung Fu Zombie Killer. Re-trieved 1.10.2015 from: http://www.gamasutra.com/view/feature/4413/action_ad-venture_level_design_.php

Gard, T. (20.4.2010c). Action adventure level design, part 1. Retrieved 1.10.2015 from: http://www.gamasutra.com/view/feature/4326/action_adventure_level_de-sign_.php

Georgio, P. (16.4.2012). CG101: Lighting. Retrieved 6.11.2015 from: http://www.digitaltutors.com/tutorial/719-CG101-Lighting

Goldberg, M. N. (2.6.2013). Polycount vs vertex count. Retrieved 3.11.2015 from: http://distantsoulsdev.blogspot.fi/2013/02/polycount-vs-vertex-count.html

Govil-Pai, S. (2.8.2006). *Principles of Computer Graphics: Theory and Practice using OpenGL and Maya.* Springer Science and Business Media. ISBN-13: 978-0-387-25479-1.

Gritz, L., d'Eon, E. (12.2007). The importance of being linear (Chapter 24). *GPU Gems 3.* Boston, MA: U.S. Corporate and Government Sales. ISBN-13: 978-0-321-51526-1.

Gurney, J. (2010). *Color and Light – A guide for the realist painter.* Kansas City, USA: Andrews McMeel Publishing, LCC. ISBN: 978-0-7407-9771-2. Retrieved 13.11.2015.

Harris, J. (26.9.2007). Game design essentials: 20 open world games. Retrieved 22.9.2015 from: http://www.gamasutra.com/view/feature/130319/game_design_essentials_20_open_.php

Hawkins, R. (8.2012). *Vertex.* Retrieved 8.10.2015 from: http://artbypapercut.com/

Hawkins, R. (1.2014). *Vertex 2.* Retrieved 8.10.2015 from: http://artbypapercut.com/

HD Admin (18.12.2012). In case you missed it: The environment art of Halo 4. Originally published at Polycount.com. Retrieved 21.8.2015 from: http://www.halodiehards.net/in-case-you-missed-it-the-environment-art-of-halo-4/

Hopkin, M. (13.6.2006). Web users judge sites in the blink of an eye. Nature Publishing Group. Retrieved 30.9.2015 from: http://www.nature.com/news/2006/060109/full/news060109-13.html

Hoppe, H., Losasso, F. (2004). Geometry clipmaps: Terrain rendering using nested regular grids. ACM Trans. Graphics (SIGGRAPH), 23(3). Retrieved 5.11.2015 from: http://research.microsoft.com/en-us/um/people/hoppe/proj/geomclipmap/

Jansson, N. (2.5.2012). Art tutorial. Retrieved 22.10.2015 from: http://www.androidarts.com/art_tut.htm

KatsBits. (n.d.). Make better textures for games, ‘Power of Two’ & Proper image dimensions. Retrieved 6.11.2015 from: http://www.katsbits.com/tutorials/textures/make-better-textures-correct-size-and-power-of-two.php

Kennedy, S. M. (2013). *How to become a video game artist: The insider’s guide to landing a job in the gaming industry.* New York, USA: Watson-Guptill Publications. ISBN: 978-0-8230-0809-4. Retrieved 2.11.2015.

Klafke, T. (n.d.). Creating modular environments in UDK. Retrieved 12.10.2015 from: http://www.thiagoklafke.com/modularenvironments.html

Koncewicz, A. (11.4.2009). A layman's guide to projection in videogames. Retrieved 26.10.2015 from: http://www.significant-bits.com/a-laymans-guide-to-projection-in-videogames

Kuzminova, G. (7.7.2009). Game art creation and reviewing requirements. Retrieved 19.10.2015 from: http://www.gamedev.net/page/resources/_/creative/visual-arts/game-art-creation-and-reviewing-requirements-r2648

Lambert, S. (26.11.2013). An introduction to spritesheet animation. Retrieved 28.10.2015 from: http://gamedevelopment.tutsplus.com/tutorials/an-introduction-to-spritesheet-animation--gamedev-13099

Lecky-Thompson, G. W. (1.1.2008). *Video game design revealed.* UK: Charles River Media. ISBN-13: 978-1-58450-562-4. Retrieved 2.11.2015.

Li, J. (29.7.2015). The maps of Jelly Splash: The beauty in simplicity. Retrieved 23.9.2015 from: http://www.gamasutra.com/blogs/JunxueLi/20150729/249780/The_Maps_of_Jelly_Splash_The_Beauty_in_Simplicity.php

Light. (29.7.2015). Let's talk about normals. Retrieved 28.10.2015 from: https://blendernpr.org/lets-talk-about-normals/

Lovato, N. (17.9.2015). 3 steps to improve your game's graphics. Retrieved 23.9.2015 from: http://www.gamasutra.com/blogs/NathanLovato/20150917/253939/3_steps_to_improve_your_games_graphics.php

Lux. (18.12.2014a). Color relativity: Color theory beyond the wheel. Retrieved 11.11.2015 from: http://purplepwny.com/blog/color_relativity_color_theory_beyond_the_wheel.html

Lux. (3.11.2014b). Pixel art for beginners, programmers, and everyone else. Retrieved 11.11.2015 from: http://purplepwny.com/blog/pixel_art_basics_for_beginners_programmers_and_everyone_else.html

Lux. (20.2.2014c). Visual hierarchy for game developers: A practical guide to making important stuff seem important. Retrieved 11.11.2015 from: http://purplepwny.com/blog/visual_hierarchy_for_game_developers_a_practical_guide_to_making_important_stuff_seem_important.html

Lux. (16.7.2012). Using perspective to convey depth in 2D games. Retrieved 11.11.2015 from: http://purplepwny.com/blog/using_perspective_to_convey_depth_in_2D_games.html

Malmqvist, N. (19.6.2001). Creating low polygon game models. Retrieved 19.10.2015 from: http://www.gamedev.net/page/resources/_/creative/visualarts/creating-low-polygon-game-models-r1404

Marshall, J. (20.5.2014a). Dress it up! The importance of set dressing. Retrieved 9.10.2015 from: http://blog.digitaltutors.com/dress-importance-set-dressing/

Marshall, J. (23.4.2014b). Prioritize your character design by using retopology tools. Retrieved 5.11.2015 from: http://blog.digitaltutors.com/prioritize-character-design-using-retopology-tools/

Masters, M. (5.3.2015a). Unity, Source 2, Unreal Engine 4, or CryENGINE - Which game engine should I choose? Retrieved 5.11.2015 from: http://blog.digitaltutors.com/unity-udk-cryengine-game-engine-choose/

Masters, M. (27.1.2015b). How to get started with 3D. Retrieved 9.11.2015 from: http://blog.digitaltutors.com/how-to-get-started-in-3d/

Masters, M. (27.12.2014a). Remastering a legend: Interview with a Halo 4 Anniversary artist. Retrieved 9.10.2015 from: http://blog.digitaltutors.com/halo-2-anniversary-environment-artist-interview/

Masters, M. (8.12.2014b). Do I need to know how to sculpt as a 3D modeler. Retrieved 5.11.2015 from: http://blog.digitaltutors.com/need-know-sculpt-3d-modeler/

Masters, M. (2.11.2014c). DT Exclusive | Building the world for Middle-earth: Shadow of Mordor. Retrieved 9.10.2015 from: http://blog.digitaltutors.com/dt-exclusive-building-world-middle-earth-shadow-mordor/

Masters, M. (24.10.2014d). DT Exclusive: Game environment tips from a Halo artist. Retrieved 9.10.2015 from: http://blog.digitaltutors.com/dt-exclusive-game-environment-tips-halo-artist/

Masters, M. (27.4.2014e). Setting the stage for a 3D world with prop and environment modelling. Retrieved 9.10.2015 from: http://blog.digitaltutors.com/setting-the-stage-for-a-3d-world-with-prop-and-environment-modeling/

Masters, M. (15.3.2014f). Light up your world: How lighting makes all the difference for games. Retrieved 9.10.2015 from: http://blog.digitaltutors.com/understanding-the-importance-of-lighting-for-games/

Masters, M. (26.2.2014g). Texturing for games – Maintain a high level of detail without extra geometry. Retrieved 8.10.2015 from: http://blog.digitaltutors.com/texturing-games-maintain-high-level-detail-without-extra-geometry/

Masters, M. (6.2.2014h). Keeping your players engaged – Tips for great game level design. Retrieved 9.10.2015 from: http://blog.digitaltutors.com/keeping-players-engaged-tips-great-game-level-design/

Masters, M. (4.2.2014i). Understanding ambient occlusion. Retrieved 6.11.2015 from: http://blog.digitaltutors.com/understanding-ambient-occlusion/

Masters, M. (2.2.2014j). Things every artist should know to paint appealing environments. Retrieved 9.10.2015 from: http://blog.digitaltutors.com/things-every-artist-know-paint-appealing-environments/

Masters, M. (22.1.2014k). Understanding different light types. Retrieved 9.10.2015 from: http://blog.digitaltutors.com/understanding-different-light-types/

Masters, M. (19.1.2014l). Understanding UVs – Love them or hate them, they’re essential to know. Retrieved 6.11.2015 from: http://blog.digitaltutors.com/understanding-uvs-love-them-or-hate-them-theyre-essential-to-know/

Masters, M. (16.1.2014m). Start mastering important 3D texturing terminology. Retrieved 5.11.2015 from: http://blog.digitaltutors.com/cover-bases-common-3d-texturing-terminology/

Maxinow, A. (2012). Ditching diffuse maps. Retrieved 21.8.2015 from: http://www.artisaverb.info/DitchingDiffuse.html

Maxinow, A. (2009). Efficient art production. Theory and practise. Retrieved 21.8.2015 from: http://www.artisaverb.info/Hygiene.html

Mayden, A. (30.9.2015). Modeling with quads or triangles – What should I use? Retrieved 5.11.2015 from: http://blog.digitaltutors.com/modeling-with-quads-or-triangles/

McEntee, C. (27.3.2012). Rational Design: The core of Rayman Origins. Retrieved 30.10.2015 from: http://www.gamasutra.com/view/feature/167214/rational_design_the_core_of_.php

McGrath, T. (1.1.2008). Creating efficient next gen environment art. Retrieved 19.10.2015 from: http://www.gamedev.net/page/resources/_/creative/visual-arts/creating-efficient-next-gen-environment-art-r2446

Morgan, F. (n.d.). Sprite Lamp. Retrieved 13.11.2015 from: http://www.snakehillgames.com/spritelamp/

Nutt, C. (28.11.2011). Building a fantasy world – the art direction of Kingdoms of Amalur: Reckoning. Retrieved 23.9.2015 from: http://www.gamasutra.com/view/feature/134924/building_a_fantasy_world__the_art_.php

Nutt, C. (6.2.2012). Building the world of Reckoning. Retrieved 23.9.2015 from: http://www.gamasutra.com/view/feature/135059/building_the_world_of_reckoning.php

Oculus VR, LCC. (2015). Field of view and scale. Retrieved 13.11.2015 from: https://developer.oculus.com/documentation/intro-vr/latest/concepts/bp_app_fov_scale/

Ohlew, T. (10.10.2014). For the love of 2D games: Why developers still make them. USgamer. Retrieved 11.11.2015 from: http://www.usgamer.net/articles/for-the-love-of-2d

Orsvärn, L. (2.10.2015). Physically based rendering. Retrieved 10.11.2015 from: http://blog.wolfire.com/2015/10/Physically-based-rendering

Perry, L. (11.2002). Modular level and component design. *Game Developer Magazine, 30-35.* Retrieved 19.10.2015 from: https://udn.epicgames.com/Three/rsrc/Three/ModularLevelDesign/ModularLevelDesign.pdf

PhotoBiz. (19.12.2013). What is the difference between raster and vector graphics. Retrieved 9.11.2015 from: http://help.photobiz.com/kb/index.php?View=entry&EntryID=370

Provost, G. (7.2003a). Beautiful, yet friendly part 2: Stop hitting the bottleneck. Originally published by Game Developer Magazine. Retrieved 21.8.2015 from: http://www.ericchadwick.com/examples/provost/byf2.html

Provost, G. (6.2003b). Beautiful, yet friendly part 1: Maximizing efficiency. Originally published by Game Developer Magazine. Retrieved 21.8.2015 from: http://www.ericchadwick.com/examples/provost/byf1.html

PsPrint. (n.d.). What’s the difference between raster and vector? Retrieved 9.11.2015 from: https://www.psprint.com/resources/difference-between-raster-vector/

Reed, A. (18.9.2015). The creators of Amnesia want to fix what Resident Evil and Silent Hill broke. Retrieved 22.9.2015 from: http://www.gamesradar.com/creators-amnesia-want-fix-what-resident-evil-and-silent-hill-broke/

Riverman Media. (n.d.). Easy atmospheric perspective in Photoshop. Retrieved 21.10.2015 from: http://rivermanmedia.com/easy-atmospheric-perspective-in-photoshop/

Russel, E. (14.8.2014). Elliminate texture confusion: Bump, normal and displacement maps. Retrieved 6.11.2015 from: http://blog.digitaltutors.com/bump-normal-and-displacement-maps/

Santos, D. D. (18.3.2015). 12 pro tips to improve your artistic composition. Retrieved 22.10.2015 from: http://www.creativebloq.com/digital-art/tips-composition-31514496

Sheffield, B. (3.6.2013). Inside the striking art and design of Hawken. Retrieved 23.9.2015 from: http://www.gamasutra.com/view/feature/193316/inside_the_striking_art_and_design_.php

Solarski, C. (4.10.2012). Sponsored feature: Drawing basics and video game art: Character design. Retrieved 20.8.2015 from: http://www.gamasutra.com/view/feature/178656/sponsored_feature_drawing_basics_.php

Solarski, C. (30.1.2013a). The aesthetics of game art and game design. Retrieved 30.10.2015 from: http://www.gamasutra.com/view/feature/185676/the_aesthetics_of_game_art_and_.php

Solarski, C. (20.7.2013b). Framing and centering: Directing player attention in open 3D worlds. Retrieved 15.9.2015 from: http://www.gamasutra.com/blogs/ChrisSolarski/20130621/194798/Framing_and_Centering_Directing_Player_Attention_in_Open_3D_Worlds.php

Sousa, T. (12.2007). Vegetation procedural animation and shading in Crysis (Chapter 24). *GPU Gems 3.* Boston, MA: U.S. Corporate and Government Sales. ISBN-13: 978-0-321-51526-1.

Stuart, K. (12.2.2015). Photorealism – The future of video game visuals. Retrieved 14.9.2015 from: http://www.theguardian.com/technology/2015/feb/12/future-of-video-gaming-visuals-nvidia-rendering

Totilo, S. (2.7.2010). The revenge of 2D. Retrieved 1.10.2015 from: http://kotaku.com/5577352/the-revenge-of-2d

Trümpler, S. (4.12.2013). Handmade normal maps. Retrieved 5.11.2015 from: http://simonschreibt.de/gat/handmade-normal-maps/

Tudor, L. (31.8.2010). An artist’s eye: Applying art techniques to game design. Retrieved 21.10.2015 from: http://www.gamasutra.com/view/feature/134398/an_artists_eye_applying_art_.php?print=1

Unity Technologies. (n.d.a). Applying edge padding to alpha textures. Retrieved 5.11.2015 from: http://docs.unity3d.com/462/Documentation/Manual/HOWTO-alphamaps.html

Unity Technologies. (n.d.b). Lightmapping in depth. Retrieved 6.11.2015 from: http://docs.unity3d.com/460/Documentation/Manual/LightmappingInDepth.html

Unity Technologies. (n.d.c). Materials and shaders. Retrieved 26.11.2015 from: http://docs.unity3d.com/Manual/Shaders.html

Van der Burg, J. (23.6.2000). Building an advanced particle system. Retrieved 10.11.2015 from: http://www.gamasutra.com/view/feature/131565/building_an_advanced_particle_.php?page=3

Van Spronsen, J. (8.9.2015). The Witcher 3: Designing the environments (Developer interview). Retrieved 12.10.2015 from: https://www.mapcore.org/articles/interviews/the-witcher-3-designing-the-environments-developer-interview-r69/

Van Spronsen, J. (29.8.2014). Interview with Thiago ‘Minos’ Klafke, Blizzard environment artist. [Interview]. Retrieved 12.10.2015 from: https://www.mapcore.org/articles/interviews/interview-with-thiago-39minos39-klafke-blizzard-environment-artist-r42/

Viscorbel, S. (27.3.2015). Vray materials – part 1 – diffuse. Retrieved 6.11.2015 from: http://viscorbel.com/vray-materials-part-1-diffuse/

Williams, M. (10.22.2014). Linear, story based games aren’t dead yet. Retrieved 4.11.2015 from: http://www.usgamer.net/articles/linear-story-based-games-arent-dead-yet

Wilson, J. (n.d.). Tutorial: Physically based rendering, and you can too! Retrieved 10.11.2015 from: http://www.marmoset.co/toolbag/learn/pbr-practice

LIST OF FIGURES