Tampereen ammattikorkeakoulu

On Production of 3D Character Models for Games

Mika Yli-Pentti

OPINNÄYTETYÖ
Toukokuu 2021

Tietojenkäsittely
Pelituotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn tutkinto-ohjelma
Pelituotanto

YLI-PENTTI, MIKA:
On Production of 3D Character Models for Games

Opinnäytetyö 35 sivua, joista liitteitä 1 sivu
Toukokuu 2021

Tietokonegrafiikka on kehittynyt kiihtyvään vauhtiin. Reaaliaikaisen renderöinnin teknologian kehityksen myötä tavat luoda kolmiulotteisia malleja ovat uudistuneet. Opinnäytetyössä oli tavoitteena tutkia videopeliin soveltuvan kolmiulotteisen hahmon nykyaikaista tuotantolinjaa sekä silmäillä mahdollisia tulevaisuuden mullistuksia. Opinnäytetyön tarkoituksena oli käytännön työn kautta saavuttaa syvempää ymmärrystä ammattimaisesta työnkulusta. Käytännön työnä toteutettiin Unity -pelimoottoriin viety animoitu hahmo.

Kolmiulotteisen hahmon tuottamisessa on useita vaiheita. Isommissa tuotannoissa samaa hahmoa saattaa edistää usea artisti vuorollaan. Hahmon tuotannon vaiheet vaikuttavat toisiinsa, ja siksi jokaiselle hahmoartistille on hyödyllistä ymmärtää tuotannon kaikkia vaiheita. Opinnäytetyöstä syntyi englanninkielinen yleiskatsaus hahmon tuotannon teknisiin vaiheisiin.

Reaaliaikainen tietokonegrafiikka on mielenkiintoisessa vaiheessa sukupolven vaihdosta. Innovaatioita rakennetaan soveltamalla elokuvateollisuuden ratkaisuja realismia tavoiteltaessa. Hahmoartistin työnkuva jatkaa muovautumistaan, ja niin alalla vaadittuja ominaisuuksia ovat sopeutuminen uuteen ja alituinen halu kehittää taitojaan.

Asiasanat: 3D-hahmon kehittäminen, 3D-mallintaminen

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Option of Game Development

YLI-PENTTI, MIKA:
On Production of 3D Character Models for Games

Bachelor's thesis 35 pages, appendices 1 page
May 2021

Computer-generated imagery has developed at an accelerating pace. The process of authoring three-dimensional models has changed with the advancements in real-time rendering technologies. The objective of the thesis was to explore a complete pipeline on the production of a three-dimensional character model for games and glimpse into procedures recently adopted by the industry. The purpose was to acquire deeper insight into different stages of character creation process and professional workflows. An animated game character was developed as the practical portion of the thesis.

The development of a three-dimensional character consists of several stages. In larger productions many artists can be working on the same character in turn. The stages of production affect the subsequent ones; therefore, it is beneficial for artists to have a general knowledge of the pipeline as a whole. The thesis is a synopsis of the production stages from a technical standpoint.

Real-time graphics are at an interesting point in time of a generation leap. Innovations are being built by adopting solutions from the movie industry in the pursuit of realism. The job of a character artist continues to reshape, and so adaptivity and a continuous appetite for improvement have become essential qualities for them.

Key words: 3D character development, 3D modeling

TABLE OF CONTENTS

1 INTRODUCTION 6
2 3D MODELS 7
    2.1 Real-time graphics 7
    2.2 High-poly model 8
    2.3 Low-poly model 9
    2.4 Hair and fur 10
    2.5 Normals 11
    2.6 UV mapping 12
    2.7 Map projection 13
    2.8 Level of detail 14
3 MATERIALS AND TEXTURES 16
4 RIGGING 19
    4.1 Skeleton 19
    4.2 Forward and inverse kinematics 20
    4.3 Skinning 20
    4.4 Control rig 21
    4.5 Facial rig 21
    4.6 Deformation 22
    4.7 Automatic rigging solutions 22
5 MOTION 24
    5.1 Keyframe animation 24
    5.2 Motion capture 24
    5.3 Blend shapes 25
    5.4 Animation blending 26
6 CHARACTER CREATORS 27
7 CASE STUDY 28
    7.1 Production 28
    7.2 Results 29
8 DELIBERATION 30
REFERENCES 31
APPENDICES 35
    Appendix 1. Software products 35
    Appendix 2. Animations video 35

GLOSSARY

UVs	Texture coordinates
Texel	Texture element
High-poly	High polygonal amount
Low-poly	Low polygonal amount
LOD	Level of detail
Quad	Quadrilateral; a polygon with four sides
PBR	Physically based rendering
GPU	Graphics processing unit
CPU	Central processing unit
FPS	Frames per second
FK	Forward kinematics
IK	Inverse kinematics
Mocap	Motion capture
NPC	Non-playable character

1 INTRODUCTION

A complete 3D character model pipeline consists of several stages that affect one another and sometimes even overlap. There is no one true way of making a model. What matters is the consistency of the end result and that it follows some optimization practices in terms of performance to match the target framerate of the application. Understanding a little of the technical side of real-time graphics can aid the artist in reaching those goals.

In larger productions multiple artists typically work on a character and the work is eventually handed to someone else. In addition to conventions and communication, having a general knowledge of the complete pipeline helps to avoid a situation where previous work is not readily understood.

This thesis explores the stages of a complete 3D character model pipeline for games and addresses some technical points in them which can help an artist perceive what is not immediately obvious about the techniques. However, every case is generalized and simplified to some extent, for considerably more can be encompassed in these topics. A few recently adopted techniques in video games are also glimpsed at to speculate what the future may hold for real-time character model authoring. The subjects of the thesis are written from a general aspect and apply to most of the relevant programs, but it should be noted that a portion of the terms and descriptions used may be different in some software packages.

2 3D MODELS

A 3D model is a structure made of polygonal surfaces. Surfaces are a collection of connected points called vertices (Petty n.d.). Video games use an application programming interface (API) such as DirectX, OpenGL, Vulkan, or Metal, to interact with a graphics processing unit (GPU). The model data is provided to a graphics API as an array of vertex data in a particular format that can be referenced by an additional indexing array. The format defines what information is contained in a vertex. Simplified, these arrays specify the color, normal, and texture coordinates of a vertex, as well as the position in its own model space and the other vertices it is connected to. Together, they form the rendering primitives, e.g., triangles (Akenine-Möller et al. 2018, p. 702, 14.)

If a model is in a scene and inside the frustum of the camera used for rendering, its vertices will be passed to a vertex shader where the data associated with each vertex is processed. The results are interpolated and outputted to a fragment shader that determines final color of the pixels of the rendered surfaces. (O'Conor 2017.) Shaders are more thoroughly explained in Chapter 3. In effect, the process of rendering is much more complex and out of the scope of this thesis.

A polygonal model can be created or generated in several ways (Akenine-Möller et al. 2018, p. 682). This thesis mostly surveys the application of modeling programs to build the models, but also touches on a few specific cases where a different method is used.

2.1 Real-time graphics

Video games are an interactive medium. As such, they must be responsive and function in real-time, including graphics. For mobile devices or cinematic games on consoles 30 frames per second (FPS) refresh rate could be considered acceptable. The modern standard is that fast-paced games are rendered at 60 FPS, or higher. To achieve that, graphics need to be well optimized in comparison to movies where rendering a single frame can take hundreds of hours. Graphics hardware has seen rapid increase in computing power in the past decades. In turn, advancements in rendering techniques require and consume more of the finite resources the hardware has to offer. (Akenine-Möller et al. 2018, pp. 1-2.)

The process of creating models keeps transforming as new and improved software products surface. This means that digital artists are required to be adaptive as the job will fundamentally change when the technology advances.

2.2 High-poly model

Creating a modern character model often includes the step of building a high-definition model that will not be put into the game engine. Instead, it is to be used as a reference from which much of the defined details will be written on texture maps via projection. The maps can be used to author other textures which are later applied to a material for the low-definition model that is imported to the game engine. This is done to optimize the model to be compatible with real-time rendering.

A traditional way to create a high-poly model is to use a modeling program, such as Maya (Appendix 1), edit sub-object components (i.e., vertices, edges, or faces), and apply sub-division to the mesh that is the collection of the surfaces of the model. Sub-division divides the surfaces of the model using an algorithm. Curved surfaces are smoother with more edges to define them, if divided with central tendency as depicted in Figure 1. During high-poly modeling the surfaces are generally kept as quads as they are easier to shape than other polygons and produce predictable results when sub-divided.

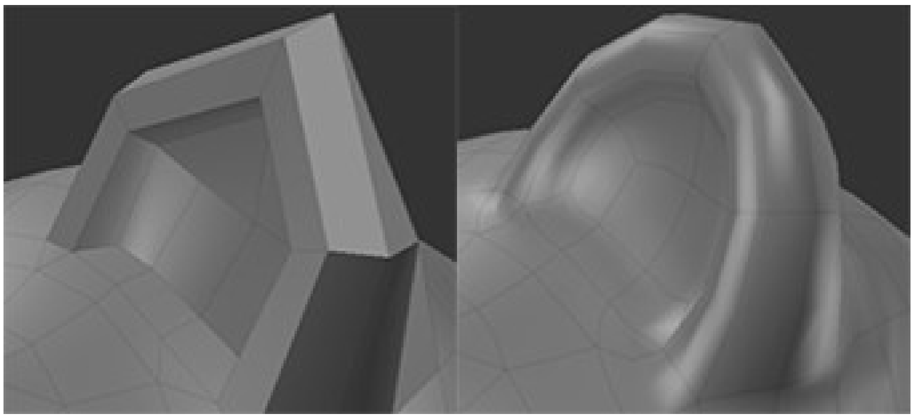
FIGURE 1. No subdivision applied on the left. The mesh is subdivided once using the Catmull–Clark algorithm on the right. (Foundry n.d.)

Each vertex is connected to preferably four other vertices, but that is not entirely possible in every case. A set of five or more edges that merge into a single vertex is referred as a pole. Poles may produce pinching in sub-divided models that can be seen as artifacts in the maps created from them. (TurboSquid n.d.)

Another method to make a high-poly model is to digitally sculpt one in a program such as ZBrush (Appendix 1). Sculpting also uses sub-division to redefine the model, but instead of selecting and editing sub-object level components directly, the model is shaped with brushes of various behaviors that affect multiple vertices at a time. It is possible to create extremely fine detail at higher sub-division levels with millions of polygons. Sculpting is well suited for organic subjects as it is easy to sculpt imperfections. (Spencer 2010.)

Creating high-poly models from photographs is also possible. Akenine-Möller et al. (2018, p. 682) define photogrammetry as the reconstruction of surfaces from one or more photographs. Software used for photogrammetry attempts to find matching points between images. Coordinates are calculated from the found matching points and after many iterations, a point cloud that represents the form of the object is presented as a result. (Moody 2017.) The models formed from this data are usually very dense with millions of polygons and demand optimization to meet real-time requirements suitable to game engines. While photogrammetry requires a dedicated pipeline and is computationally expensive, the resulting quality of the assets is substantially high. (Statham 2020, pp. 289–307.)

2.3 Low-poly model

Low-poly model is the version of the model that is textured, rigged, animated, and then imported into the game engine. The low-poly model should reflect the shape of the high-poly model as closely as possible while having only the essential number of polygons. Instead of creating the low-poly model from scratch, remeshing the previously made high-poly model into a lower definition is often preferred. (Marshall 2020.) Contrary to its name, the most important low-poly character models can consist of tens, if not hundreds, of thousands of polygons in larger modern productions.

Since models are completely triangulated when they are imported to a game engine, artists can build low-poly models with both quads and triangles. Faces with more than four sides are called N-gons and should be avoided as they can cause errors in smoothing, deformation, and rendering. Thin and lengthy, or tiny triangles are also a problematic issue as they cause excessive overdraw on GPU (Persson 2009). The triangulation can be done manually to verify its consistency across programs. At this point, the artist must also begin considering how the character will be animated and how the animations might deform the model. Areas that bend considerably need special attention regarding their topology and often additional supporting geometry (constitution and number of polygons in a mesh).

2.4 Hair and fur

Traditionally, real-time hair and fur have been created using a card-based approach, or with basic modeling. Cards form the approximate body of a larger number of individual hairs as seen in Figure 2. (Epic Games n.d.-a.) Cards are an efficient means in terms of performance if overdraw can be avoided (Jiang 2016). Nevertheless, they are far off from effortless authoring, have limited styles, lack realistic simulation, and are bound to simplified shading (Tafuri 2019).

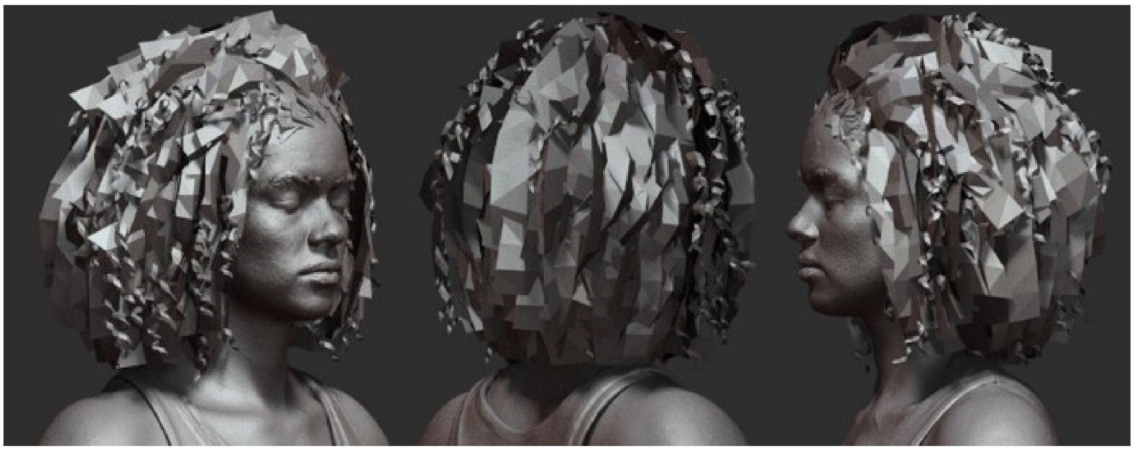

FIGURE 2. Hair cards (Jiang 2016).

Consequently, the industry has started implementing strand-based workflow for hair and fur into real-time rendering. Strand-based hairs have formerly been used only in movies. This is because rendering each hair as an individual strand has a severe impact on performance (Epic Games n.d.-a). Currently, this technology is feasible only on high-end graphics hardware. Nevertheless, it clearly is a major step forward in realistic real-time character rendering.

2.5 Normals

> A normal is a theoretical line, perpendicular to the surface of a polygon. – – normals are used to determine the orientation of a polygon face (face normals), or how the edges of faces will visually appear in relation to each other when shaded (vertex normals). (Autodesk 2019)

A vertex has as many normals as the faces it connects. Vertex normals can be kept separate or set to be unified. Separated vertex normals point in parallel directions to their faces. In this case, surfaces are presented as facets in the shading of the model. Technically there are as many vertices located at the same place as there are vertex normals, even though they may be represented as a single vertex. Thus, vertices with separate vertex normals increase the number of vertices the game must process when rendering, making it less performant. If the vertex normals are unified on a particular vertex, they point at the same averaged direction, and the shading of the faces appears smooth. Vertices with a shared vertex normal are typically considered to be the same single vertex. (Autodesk 2019.)

Vertex normals are an important variable in the calculation that determines how the lights in a scene affect the shading of the surfaces. Therefore, editing the normals has a massive impact on how the model is lit when rendered. This can be exploited in various ways. For example, to smooth the shading of overlapping polygonal hair cards by transferring the vertex normals from a proxy mesh with unified vertex normals as visualized in Figure 3. In a more specific case, the vertex normals can be manually modified to control the shadow threshold in cell-shading (Motomura 2015). Vertex normals also play an important role in the process of baking a normal map (Wilson n.d.).

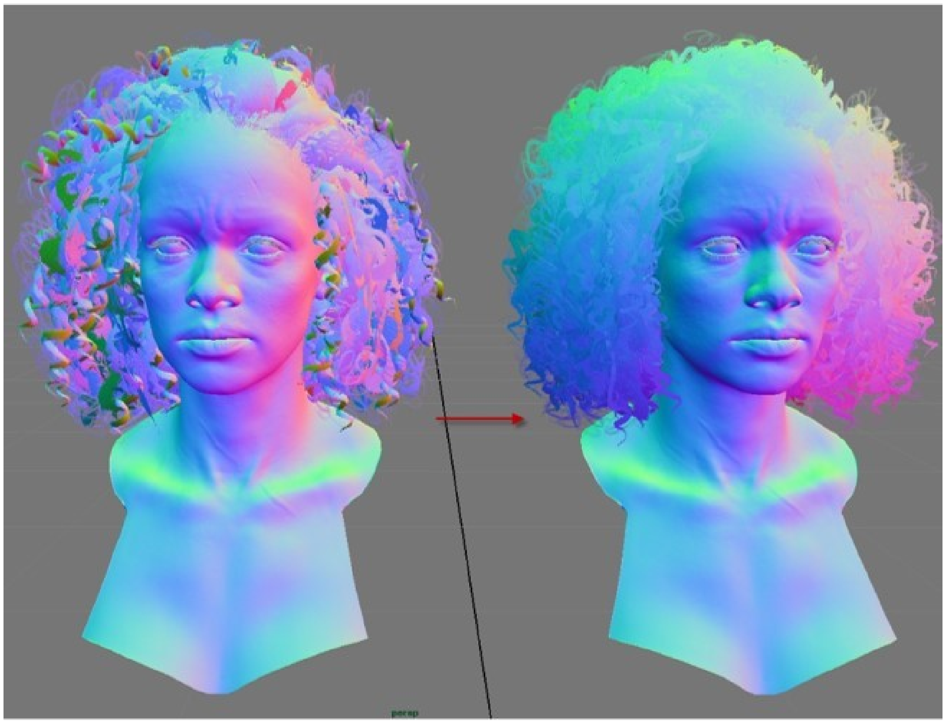

FIGURE 3. Unified vertex normals on hair cards (Jiang 2016).

2.6 UV mapping

UVs are two-dimensional texture coordinates that are one of the attributes included in the vertex data. They exist to define a system called UV texture space and correspond with vertex information to map images onto surface mesh. UV mapping is the process of creating a 2D representation of the 3D mesh. (Autodesk 2018; Russell 2020.) It is done by marking seams and then cutting the mesh into separate 2D shells in an UV editor. The layout of the shells is important. Empty space results in wasted memory and larger file sizes on the disk. However, watertight UV layouts are also impractical. Leaving a small padding of space between the shells helps to avoid bleeding from adjacent shells at lower mipmap levels (see Chapter 3). Figure 4 illustrates a model that has properly mapped UVs.

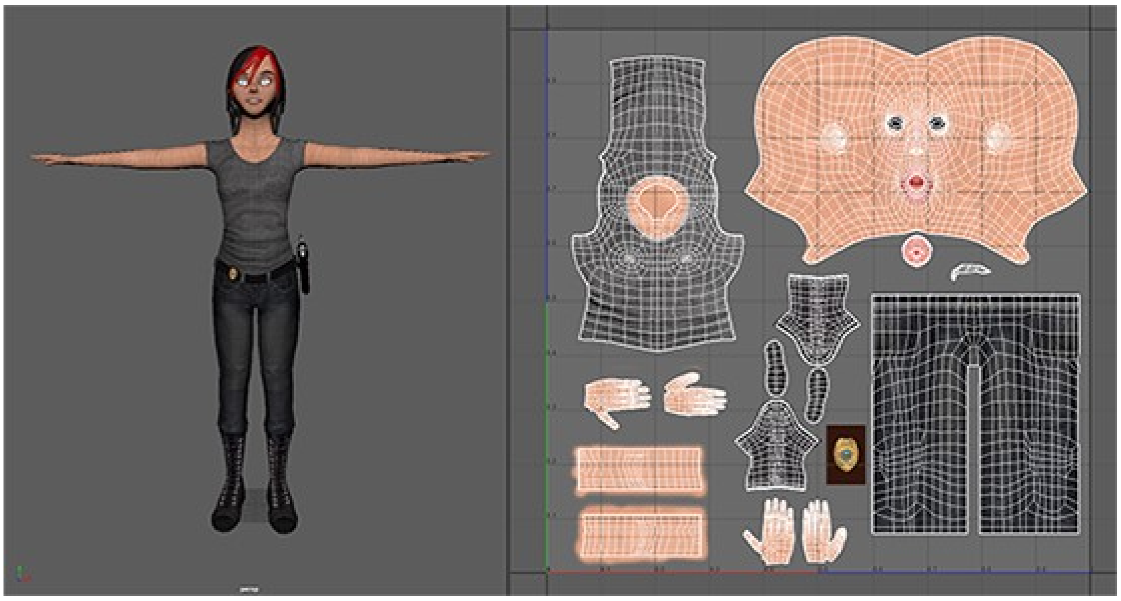

FIGURE 4. UV mapped model with textures. The UV shells are laid out on the right. (Autodesk 2018.)

Important to note as well is that when the mesh UVs are cut, the vertex normals are split along the cut. This is unavoidable, and so the vertex normals of those vertices can be explicitly set separated at no additional cost.

Texel density is an essential concept to digest in conjunction with UV mapping. Texture elements, or texels in short, are the smallest unit of texture space, i.e., a pixel in a texture image, and they are mapped to a single dot in 3D object space. Texel density in turn, is the texture resolution on a mesh. Mesh size, camera distance, UV shell sizes, and the resolution of the texture image are the factors that impact texel density. (Russell 2020.) To give an example, a model with high resolution textures that is far away from the camera has a high texel density if rendered to only a few pixels on screen. Texel density can be calculated and set to scale the UV layouts accordingly across the models of the same importance in a scene to keep texture quality on them consistent (Iezzi 2016). In conclusion, creating UVs require meticulous planning.

2.7 Map projection

Once the high-poly and low-poly models have been created, and the low-poly model has been UV mapped, they can be put through a process called baking. In baking, the surface details from the high-poly model are projected onto the low-poly model and written (or baked) on texture maps. The projection is done by using the vertex normals of the low-poly model and a cage to find surfaces of the high-poly model as demonstrated in Figure 5. Therefore, it is critical that both models are similar by shape and that vertex normals are set split where needed. There are several maps that can be generated from projection to be used in texturing. A commonly baked map is the normal map. (McDermott 2015.)

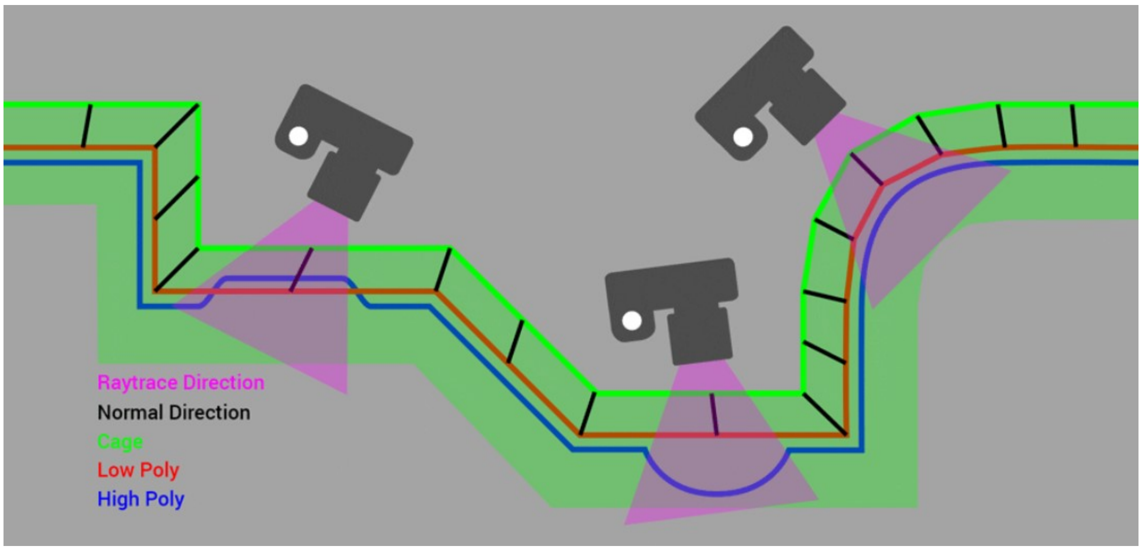

FIGURE 5. How projection works (Wilson n.d.).

A normal map is a special 2D texture that can be utilized in the fragment shader. With a normal map applied, the surfaces of a low-poly model are perceived as if it was the high-poly model when interacting with the lights in a scene, as depicted in Figure 6. Because normal mapping does not change the actual shape of the low-poly model, the effect works best on surfaces facing the view direction of the camera. (Unity Technologies 2021a.)

2.8 Level of detail

Level of detail (LOD) is a technique that reduces the workload of GPU to render distant models by swapping them for simplified meshes with less geometry. A range of LOD levels can be designated and explicit meshes are made or generated for them. LOD 0 is the level with the most geometry and is used when the model is close to the camera. The distances from the camera for other LOD levels are specified. (Unity Technologies 2021b.)

The LOD level change when a mesh is swapped might be visually distracting. To remedy this, both meshes can be cross-faded in and out simultaneously with dithering or alpha blending to make the transition appear smoother. (Takahashi 2017.)

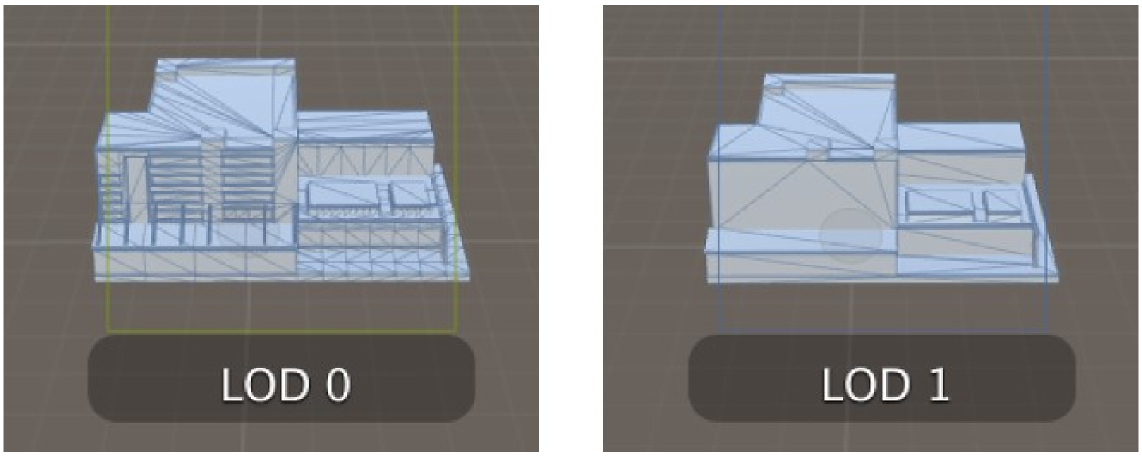

FIGURE 7. LOD group of two levels (Unity Technologies 2021b).

3 MATERIALS AND TEXTURES

Materials are what describe the appearance of the surfaces of a model. Materials reference a shader and can be interpreted to be instances of it. Exposed shader properties are inputs for the shader that can be accessed and set by the material. (Unity Technologies 2021c.)

A shader is a program that specifies instructions on how the vertices are processed and the color of the resulting pixels. They are customarily written in a shading language such as HLSL and traditionally run on the GPU in the manner shown in Figure 8. Some game engines, such as Unity and Unreal Engine (Appendix 1), also provide a way to create shaders through a node graph editor. The structure of a shader depends on the shading language and the rendering pipeline, but all typically have a property block and at least one pass. Inside a pass reside the vertex and fragment shaders. The vertex shader takes vertex data as input and handles per-vertex calculations. It outputs a processed struct (a composite data type) of information to the fragment shader. A fragment denotes a potential pixel. Calculations are done per-pixel in the fragment shader. (O'Conor 2017.)

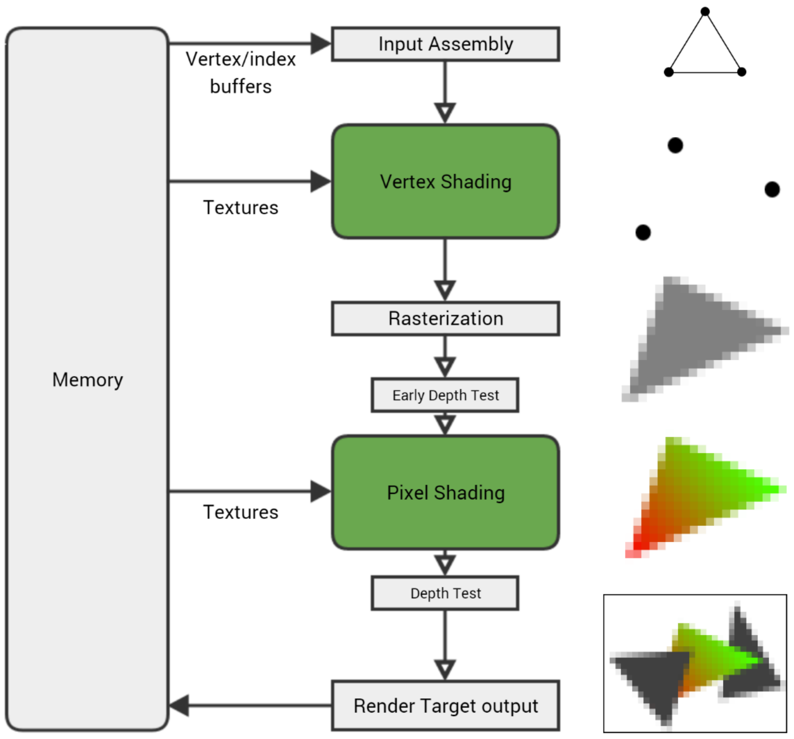

FIGURE 8. Traditional shader flow on GPU (O'Conor 2017).

When the central processing unit (CPU) commands the GPU to render a model, a draw call is issued. GPU state is set according to the provided material and vertex data. GPU state changes are expensive operations in terms of performance. Thus, batching and merging similar draw calls matter to a great extent. Using a single uber-shader for everything makes that more trivial to set up. Enabling depth buffer is also beneficial as it reduces overdraw with depth testing. (O'Conor 2017.)

Promising novel shader pipelines are being developed. Compute and mesh shaders provide potentially higher performance and more flexibility over the traditional pipeline. (Kubisch 2018; Torabi 2019.) Microsoft is also in the process of bringing DirectStorage, a DirectX-based API that will allow modern GPUs to fully utilize the high transfer-speeds of solid-state drive data storages, to Windows personal computers (Yeung 2020).

Assets in video game rendering workflows have traditionally been textured and shaded using diffuse and specular maps that approximate light interaction with materials. For artists, that induces guesswork and inconsistency. Physically based rendering (PBR) model was introduced into real-time rendering in the past decade. It follows principled approaches of representing assets from a physically accurate standpoint. In PBR, the shader handles the rules of physics, while the artist authors maps that conform to guidelines of a PBR workflow. This makes it possible to efficiently create assets that imitate reality and look accurate in all lighting conditions. PBR is more of an ambiguous concept than a standard, so there are different implementations. The two most popular PBR workflows are metallic/roughness and specular/glossiness. (McDermott 2018.)

Material maps are created by referencing measured surface values observed in the real world in both workflows. A good starting point is to figure out whether the surface is metal or not. With programs such as Substance Painter (Appendix 1), materials can be masked and layered on top of each other building up a combined material to be used in a game engine. They also come packed with access to growing libraries of predefined and scanned materials that can be helpful in laying a foundation for a custom material. (McDermott 2018.)

While this thesis does not explore the theory behind PBR or how to create materials in practice, there are a few practical things to note concerning the use of bitmap textures in a game engine. Some maps in a PBR workflow are required to be computed in linear color space while others are gamma corrected. The conversion between the two is usually handled automatically within the texturing software but the textures may need to be flagged with corresponding color settings when imported into the game engine. Gamma-encoded textures include color, diffuse, specular, and emissive maps. Linear textures include roughness, ambient occlusion, normal, metallic, and height maps. (McDermott 2018.) Another vital caution to take is a technique called mipmapping. By enabling mipmaps when importing the bitmaps, prefiltered copies of the image are generated as depicted in Figure 9. The size of every subsequent copy is progressively reduced by half. Smaller versions are used when the model is farther away from the camera. Albeit mipmaps increase the memory footprint of textures, they greatly decrease memory access bandwidth requirement by improving the effectiveness of the cache when the GPU fetches the texture. Their use greatly helps to avoid aliasing and shimmering as well. (Unity Technologies 2021e; O'Conor 2017.)

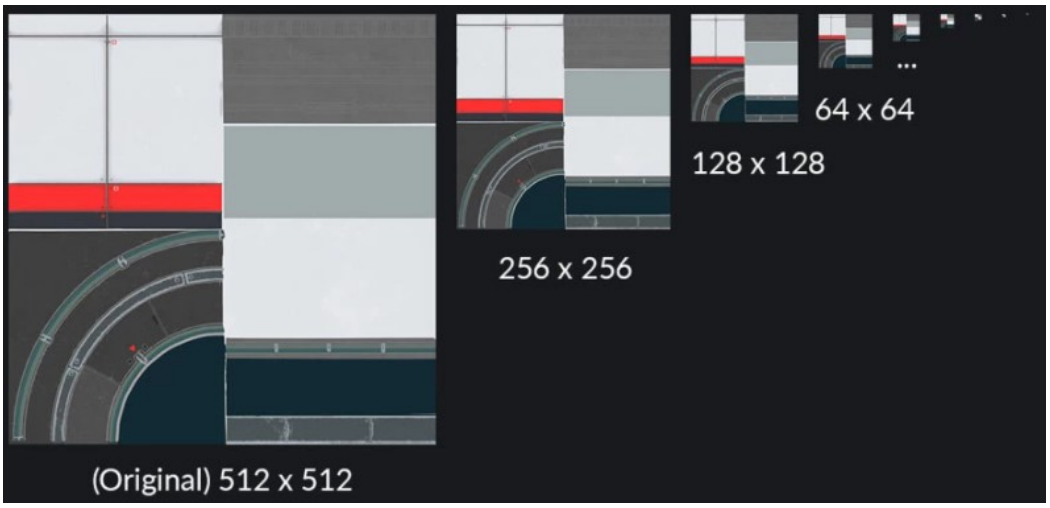

FIGURE 9. Mipmapping (Unity Technologies 2021d).

4 RIGGING

Rigging is the process of creating a skeleton with joints, binding the mesh to it via skinning, and lastly making a set of controls that drive the underlaying skeleton. The rig is the component that the model is animated with. (Unity Technologies 2020.) The real-time nature of video games induces some technical limitations in rigging that the rigger must be aware of (McLaughlin, Cutler & Coleman 2011).

4.1 Skeleton

A skeleton is a hierarchical construction of joints. The hierarchy of a skeleton is a transform-based relationship between multiple joints where one is known as the parent and the others as the children. The children inherit the transform of the parent. For example, when the parent is translated to a position the children follow while keeping their relative offsets to the parent. The child has a local transform which does not affect the parent. (Zeman 2015, chapter 1.) Typically included in the hierarchy of a skeleton are joints for hips, spine, arms, legs, neck, and head. For a more complex rig, fingers and facial joints may also be required. The number of joints in a skeleton hierarchy is unlimited, but the run-time performance of a rig declines with additional joints. (McLaughlin et al. 2011.)

Two rotation calculation systems are used in 3D graphics: Euler rotations and quaternions. Eulers are easy to comprehend but are occasionally unpredictable and cause problems, most notably the gimbal lock. Euler rotations are hierarchically structured, meaning joints have an adjustable rotation order in where the rotation axes are parented accordingly. Gimbal lock happens when two of the axes point in the same direction because of rotation, causing a loss in degree of freedom. Quaternions on the other hand are precise but difficult to understand and animate with. Game engines convert Euler rotations to quaternions for calculations behind the scenes. To have a chain of joints animate correctly, joint orientations must be configured so that the joints in the hierarchy point to proper directions while retaining zero Euler rotation values. (Zeman 2015, chapter 2; Unity Technologies 2021f.)

4.2 Forward and inverse kinematics

Kinematics are the mechanics of movement, and they are used to animate the skeleton. Forward Kinematics (FK) is the principle that rotating a joint will not change its own position but will change the world position of its children in an arc around it while keeping the distance between the two the same. FK are suitable when the character body is freely floating. (Zeman 2015, chapter 4.)

To animate a character in an interaction with another entity, e.g., walking on ground, Inverse Kinematics (IK) are used. IK inverts the forward kinematic equation so that the parent is rotated based on the position of the child. An IK chain can be established between joints. An effector, a handle, and a solver are included in an IK chain. The effector is placed at the position of the child joint, such as wrist. The handle is an object that the artist selects and animates the chain with. The solver calculates and controls the rotations of the joints, omitting the rotation of the effector joint. Because there is more than one correct solution in IK, a pole vector is needed to tell the solver which one to use. A common method using a rotate-plane IK solver is to create a locator object and constrain the handle to aim at it with a pole vector constraint. Additional parameters to influence IK solver behavior might be included in the attributes of the joints depending on the digital content creation program used. (Zeman 2015, chapter 4.)

4.3 Skinning

After the skeleton has been created, the model will then be bound to it with skinning. When a model has been skinned it reacts to the transformations of the joints using skin weights that are assigned to vertices. Multiple joints can have influence on the same vertex. Depending on the target platform and the game engine used, the number of influences may be limited. In addition to the number of joint influences, the precision of the skin weight values also may affect the overall run-time performance. (Autodesk 2020a; McLaughlin et al. 2011.)

Linear blend skinning is the only currently supported skinning method by Unity and Unreal Engine. The skinning can be improved by painting weights or by selecting vertices and editing them on the component level after initial automated skinning. It is generally a good idea to model the character in a T- or A-shaped rest pose as it will make the process of skinning easier. Making test animations of extreme poses is also helpful in identifying misbehaving deformations.

4.4 Control rig

The control rig is a set of controller objects that are constraining skeleton joints or other controller objects of the rig. The appearance of the control rig depends upon the rigger's preferences, but commonly only simple shapes are used as controller objects.

Constraints are similar to parenting, except that they work in a reverse fashion. A transform of a constraining object is connected to that of the constrained object. Commonly used constraints are point, orient, aim, parent, and pole vector constraints, which all have unique behavior. (Zeman 2015, chapter 9.)

Custom attributes can be created to make animating parts of the model more convenient (Zeman 2015, chapter 9). For example, a hand could be set to clench into a fist and then unclench with clamped driven keys mapped to an attribute attached to the controller of the hand.

4.5 Facial rig

A simple facial rig can be set up just like any other rig; a few joints to control head and jaw movement with aim constrained eyes. But that kind of facial rig cannot produce emotions. The muscles in our face work in tandem to form the facial expressions, so to achieve realistic facial animation a more sophisticated rig is also needed. For instance, a rig that compounds driven blend shapes into joint-based animation. Such rigs can become incredibly complex to assemble.

Speech is an important issue that the facial rig must address if the character is able to talk. Syncing mouth animation to sound is difficult to get right. Phoneme-based blend shapes have been traditionally used in computer generated-imagery to lip sync a character. They are the shapes a mouth makes with the sounds that form up speech. An improved approach is to use visual phonemes called visemes. (Osipa 2010, chapter 1.)

More recently, the video game industry has started adopting a muscle-based blend shape workflow for facial animation and lip syncing. Facial Action Coding System (F.A.C.S.), as lead character technical artist Axel Grossman states, “basically breaks down the form changes in the face as they relate to individual or combined muscle actions.” The benefit of using a muscle-based F.A.C.S. rig over a phoneme-based one is that it retains actor performances more closely. (Pinto & Grossman 2018.)

4.6 Deformation

Deformation can be defined as the adjustment of geometry in response to an external object or action (Zeman 2015, chapter 7). The joints and skinned geometry are generally the primary source of deformation in games for being optimized on the hardware (McLaughlin et al. 2011).

Modeling programs such as Maya may come with deformer tools. Game engines may only support skin clusters and blend shapes as imported deformation (Unity Technologies 2020). Therefore, unsupported deformer animation must be converted into blend shapes and baked prior exportation.

There can be cases in which a character needs to have extreme deformation that the primary skeleton is unable to support, such as transformation animations. A model swapping trick can be used where a completely different model and skeleton are loaded to replace the primary ones. (New Frame Plus 2020.)

4.7 Automatic rigging solutions

Rigging is a time-consuming process and take a large amount of knowledge. Different types of figures require different types of rigs. Biped humanoid characters with realistic proportions are a well-researched topic and advancements are brought to rigging them every year. New tools and techniques continue to speed up the workflows.

Maya has introduced a Quick Rig tool that can be used to automatically create a simple rig for a mesh that can be refined further with Autodesk HumanIK system. Mixamo is an online service that automatically rigs compatible models that are uploaded to it. Unreal Engine and Unity have both recently added run-time animation rigging capabilities that can be used to create procedural IK animation. A skinned model can also be rigged and keyframe animated inside the Unreal Engine with Control Rig system. Machine learning and neural networks are being utilized in rigging as Liu et al. (2019) demonstrated in their presentation Neuro-Skinning: automatic skin binding for production characters with deep graph networks. Skinning decomposition is a set of automated techniques that extract skinning models and joint transformations from input animations. For instance, it can be used to efficiently transfer motion capture data into the game engine and is recently being augmented into pipelines by industry-leading studios. (EA n.d.; Le & Deng 2012; Pinto & Grossman 2018.)

5 MOTION

5.1 Keyframe animation

Animations in 3D graphics are traditionally done with keyframe animation. The animator poses the character joints at desired frames to form a sequence. Markers called keys are set that indicate the timing and the value of selected attributes of the animated object. The motion in between keys is smoothly interpolated. (Autodesk 2020b.)

Keyframe interpolation can be influenced with easings and key weights. Game engines usually compress imported animation for performance reasons by reducing the amount of keyframes.

5.2 Motion capture

Motion capture, or mocap in short, is the practice of translating actual motion from real world into digital form and applying the data to a model. Presently, mocap is done in studios using state-of-the-art optical systems for larger productions. The actors perform in special mocap suits that are embedded with reflective locator markers. (Nogueira 2011.)

Facial motion is nowadays captured without the bulky locators as they hamper the actor's performance. Instead, their facial movement is recorded in a video stream. A software algorithm recognizes the forms of the face and tracks them. (Faceware Technologies n.d.) The captured motion data is used to control a facial rig, as depicted in Figure 10.

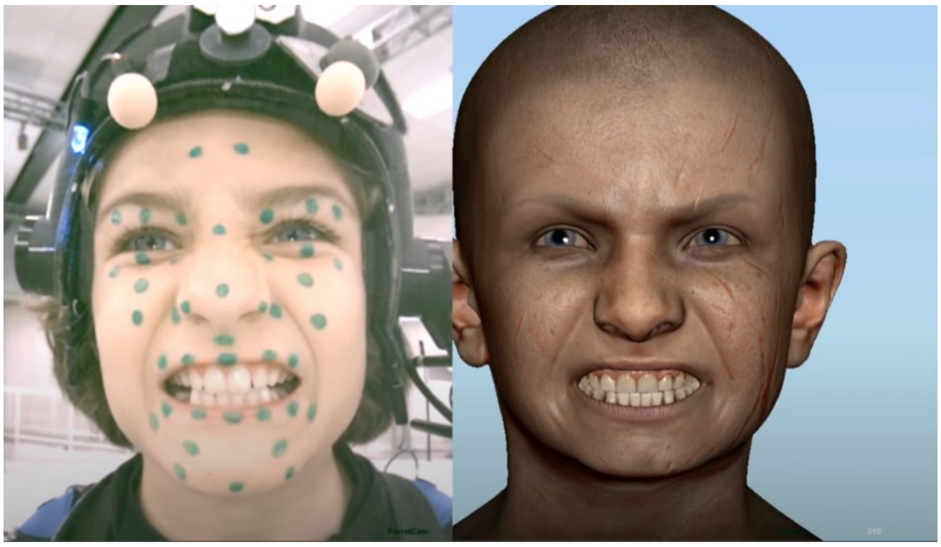
FIGURE 10. Motion capture data controlling a facial rig (Pinto & Grossman 2018).

There are alternatives to the expensive mocap studio setup. Microsoft Kinect sensor is an affordable markerless motion capture solution. (Nogueira 2011.) Epic Games provides an iOS application that can capture facial animations with newer iPhones and animate the face of a character inside Unreal Engine in real-time (Epic Games n.d.-b). Additionally, many facial motion capture desktop software exist that use a webcam or a pre-recorded video as a source.

5.3 Blend shapes

Blend shapes are modified copies of the base mesh geometry, reshaped to a unique expression. They can be used to animate a character by interpolating the geometry between the original and the reshaped meshes. (Orvalho et al. 2012.) Blend shapes are commonly used to animate a face but can also be created to fix specific deformation problems on the base mesh as a corrective measure (Autodesk 2017). The hard rule with blend shapes is that the number of vertices and their exact order must remain the same to the original mesh. Some game engines only support static blend shapes that are not driven by any deformers (Unity Technologies 2020).

Sophisticated modern facial rigs can contain hundreds or even thousands of blend shapes some of which are bilateral. It is a huge undertaking and takes much organizing because each scanned expression requires cleaning up before they can be used as blend shapes. (Pinto & Grossman 2018.)

5.4 Animation blending

To achieve seamless high-quality animation, animation blending is required in games. Transition animations can be made for switching character states, but they might reduce the responsiveness of player input. Layering and overriding parts of animation is an efficient way of dynamically increase variety in animations. (McLaughlin et al. 2011.)

Materials can also be set to react to animations. For example, in Figure 11 is shown how a wrinkle map can enhance the magnitude of a compressed expression of the character considerably (Valve Software 2019).

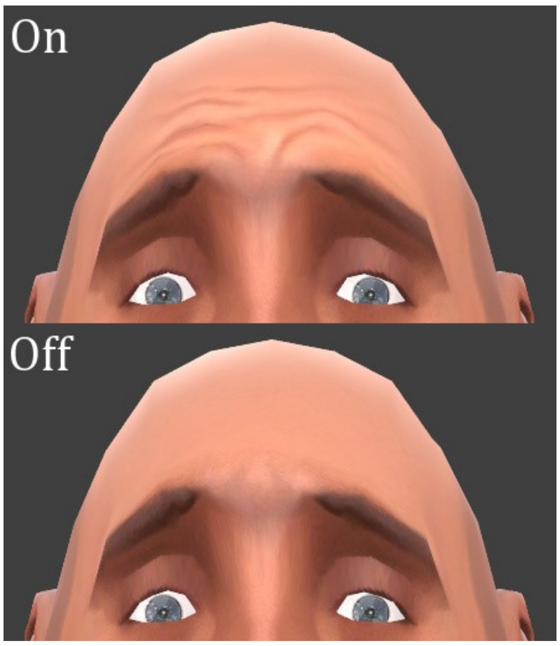

FIGURE 11. A wrinkle map adds expressive detail to the forehead (Valve Software 2019).

6 CHARACTER CREATORS

To hasten the process of character creation, there are solutions that provide character models in varying degrees. Character creators build models from lists of adjustable feature presets using blend shapes. They can be used as is or as a starting point for a custom model. They may come packed with a compatible rig.

For example, MakeHuman is an open-source character creator that can be used to prototype humanoid models (MakeHuman Community n.d.). Epic Games is developing ambitious MetaHumans, which are scan-based photorealistic characters that feature ready-made LODs and a rig to be used in Unreal Engine (Epic Games 2021). Non-photorealistic character creators also exist. For The Legend of Zelda: Breath of the Wild, Nintendo made the NPC characters using their proprietary Mii format (Polygon 2021).

7 CASE STUDY

To accompany the thesis, a practical case study of an animated 3D character model was created using many of the researched techniques. The production stages are shown in Figure 12. The Legend of Zelda: Link's Awakening was used as inspiration to the style of the character.

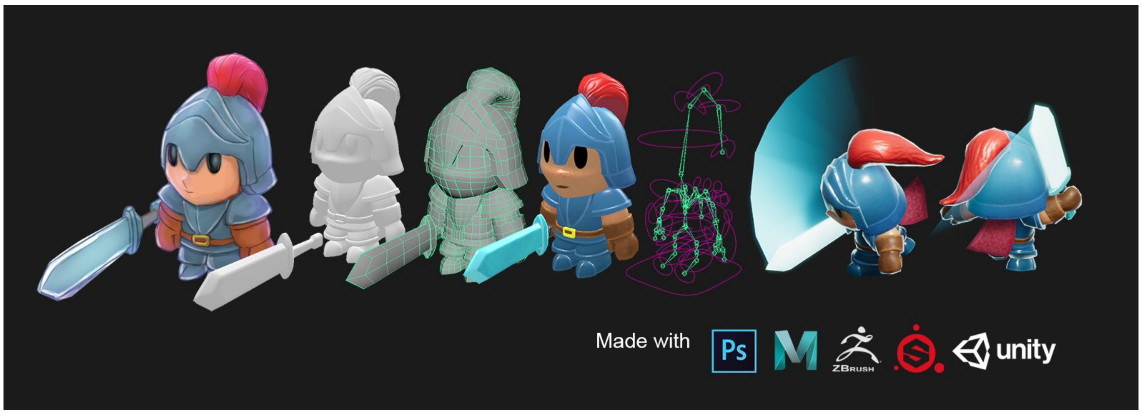

FIGURE 12. Production stages.

7.1 Production

Concept art for the character was made by first drawing a rough sketch on a paper and then polished digitally in Photoshop (Appendix 1). A 3D model was blocked out next in Maya to get the proportions right. The model was refined further using basic modeling to a subdivided high-poly version. The brush of the helmet was then exported to ZBrush where moderate details were sculpted onto it. A low-poly model was essentially already done from the undivided mesh. UVs were then made in two sets. Since the model was designed to be completely symmetrical, only one half of the model had to be unwrapped. The other half could then be mirrored and offset to bake a normal map correctly. The projection was done in Substance Painter where the model was also textured. At that point, the model was imported to Unity to ensure consistency in the semblance between Painter and in-game version of the model.

Rigging in Maya was the next step. A skeleton hierarchy stemming from the hips was created and the mesh bound to it with skinning. Some excessive manual weight editing had to be done in component level due to suboptimal pose and overlapping parts of the mesh. A rig was then constructed with simple shapes and constrained to the skeleton with a couple of custom attributes. IK was experimented with but did not produce satisfactory results so only FK were used. The finished rig has limitations, but they can be disregarded since the model will always be viewed from a fixed top-down angle from which imperfections in animations are less visible. The character was also animated in Maya. In Unity, Universal Render Pipeline was used to access Shader Graph and create shaders for the character.

7.2 Results

The finished character has animations for seven different states with custom-made materials and effects, ready to be used in a game (Appendix 2). For further improvement, facial impressions could be added using flipbook texture animation, which switches between cropped regions in a texture to display the desired segment. For example, a flipbook texture could contain multiple different eye or mouth shapes.

In retrospect, if the character had been modeled with different topology and in a better pose, the process of rigging would not have been as puzzling. It was also discovered that Maya (2019 version) does not natively provide a function to mirror animation during animating the walk cycle. The only way to do that is to painfully copy and paste transform values across to the other side of the rig one by one. That is not a job fit for a human, even when there are only a couple of mirrored animations in the production. A third-party tool called animBot (Appendix 1) saved the day and did all that repetitive heavy lifting with only a few button clicks. The character was named “Brave” and will probably feature in a turn-based mobile puzzle game that is in development.

8 DELIBERATION

Production sizes continue to grow as more and more data is required to be processed to achieve photorealistic real-time quality in games. Time becomes an invaluable resource as a result. The advances made with the cutting-edge technologies will eventually become accessible to independent studios and users. Scan-based workflows are well established in texturing today due to the advent of PBR. With MetaHumans, creating realistic characters will be facilitated without the massive amount of labor that is associated with photogrammetry.

Will character artists be needed anymore with these prospective tools that can do much of the work automatically? Absolutely! The tools speed up the workflows, allowing the artist to concentrate more in the creative part of making characters. As prominent as MetaHuman character creator seems, it probably cannot make stylized figures, monsters, or other creatures of imagination. Just like motion capture did not kill the job of an animator, neither will character creators kill the job of a character artists.

While the modern tools ease and streamline the production pipelines, having some technical knowledge can help character artists to troubleshoot performance issues with their models, and ultimately create better games. Even when specializing in certain stages of the production, the benefit of comprehending it as whole is potential increase in efficiency. This thesis provided a condensed overview of a complete production pipeline that can serve as a gentle start line for further study.

REFERENCES

Akenine-Möller, T., Haines, E., Hoffman, N., Pesce, A., Iwanicki, M., Hillaire, S. (2018) Real-time rendering. Fourth edition. Boca Raton: CRC Press, Taylor & Francis Group.

Autodesk 2017. Create post-skinning corrective shapes. Documentation. Referenced 14.5.2021. https://knowledge.autodesk.com/support/maya-lt/learn-explore/caas/CloudHelp/cloudhelp/2017/ENU/MayaLT/files/GUID-B8755F24-7C24-4066-B6F8-2DE6954DC2BF-htm.html

Autodesk 2018. UVs. Documentation. Referenced 8.5.2021. https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2020/ENU/Maya-Modeling/files/GUID-FDCD0C68-2496-4405-A785-3AA93E9A3B25-htm.html

Autodesk 2019. Polygon normals. Documentation. Referenced 7.5.2021. https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2019/ENU/Maya-Modeling/files/GUID-9C257D44-924D-4B3F-ADEF-C71FAA98EAB1-htm.html

Autodesk 2020a. Skinning your character. Documentation. Referenced 13.5.2021. https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2020/ENU/Maya-CharacterAnimation/files/GUID-EFE68C08-9ADA-4355-8203-5D1D109DCC82-htm.html

Autodesk 2020b. Keyframe animation. Documentation. Referenced 14.5.2021. https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2020/ENU/Maya-Animation/files/GUID-66ED4510-CC1B-4E11-918B-B7DC447E38A7-htm.html

EA n.d. Introducing Dem Bones: an Open Source Library for Skinning Decomposition. Website. Referenced 14.5.2021. https://www.ea.com/seed/news/open-source-dem-bones

Epic Games n.d.-a. Hair Rendering. Documentation. Referenced 9.5.2021. https://docs.unrealengine.com/en-US/WorkingWithContent/Hair/Overview/index.html

Epic Games n.d.-b. Recording Facial Animation from an iPhone X. Documentation. Referenced 14.5.2021. https://docs.unrealengine.com/en-US/AnimatingObjects/SkeletalMeshAnimation/FacialRecordingiPhone/index.html

Epic Games 2021. MetaHuman Creator. Website. Referenced 14.5.2021. https://www.unrealengine.com/en-US/metahuman-creator

Faceware Technologies n.d. Intro to Faceware Studio. Documentation. Referenced 14.5.2021. http://support.facewaretech.com/studio-intro

Foundry n.d. Subdividing Objects. Documentation. Referenced 8.5.2021. https://learn.foundry.com/mari/Content/user_guide/multiple_objects/subdivid-ing_objects.html

Iezzi, L. 2016. Figuring Out Texel Density. Article. Referenced 8.5.2021. https://80.lv/articles/textel-density-tutorial/

Jiang, Y. 2016. The Process of Creating Volumetric-based Materials in Un-charted 4. Presentation, Siggraph 2016. Referenced 9.5.2021. http://ad-vances.realtimerendering.com/s2016/

Kubisch, C. 2018. Introduction to Turing Mesh Shaders. Article. Referenced 30.5.2021. https://developer.nvidia.com/blog/introduction-turing-mesh-shaders/

Le, B. & Deng, Z. 2012. Smooth Skinning Decomposition with Rigid Bones. Presentation, Siggraph Asia 2012. Referenced 14.5.2021. https://binh.graphics/papers/2012sa-ssdr/

Liu, L., Zheng, Y., Tang, D., Yuan, Y., Fan, C. & Zhou, K. 2019. NeuroSkinning: automatic skin binding for production characters with deep graph networks. Presentation, Siggraph 2019. Referenced 13.5.2021. https://dl.acm.org/doi/10.1145/3306346.3322969

MakeHuman Community n.d. Software. Referenced 14.5.2021. http://www.makehumancommunity.org/

Marshall, J. 2020. Retopologizing Game Characters in Maya. Video. Refer-enced 7.5.2021. https://learn.unity.com/tutorial/setting-up-for-new-topol-ogy?uv=2018.1&courseId=5d0b72eaedbc2a001ffed2b2&pro-jectId=5d06177fedbc2a2d85d21849#

McDermott, W. 2015. Substance Paint Tutorial – Fundamentals 05: Baking tex-tures. Video. Referenced 8.5.2021. https://www.youtube.com/watch?v=ePnL-TuzRAbg

McDermott, W. 2018. The PBR Guide – Part 1 & 2. Article. Referenced 6.10.2020. https://academy.substance3d.com/courses/the-pbr-guide-part-1

McLaughlin, T., Cutler, L. & Coleman, D., 2011. Character rigging, defor-mations, and simulations in film and game production. In ACM SIGGRAPH 2011 Courses (pp. 1–18).

Moody, A. 2017. Tutorial: Photogrammetry Rubble Prop. Blog. Referenced 7.5.2021 http://www.digitalmistakegames.com/2017/07/30/280/

Motomura, J. 2015. GuiltyGearXrd’s Art Style: The X Factor Between 2D and 3D. Video, GDC 2015. Referenced 7.5.2021. https://www.gdcvault.com/play/1022031/GuiltyGearXrd-s-Art-Style-The

New Frame Plus 2020. How Do Inkling Transformations Work? Video. Refer-enced 13.5.2021. https://www.youtube.com/watch?v=-oMTHqC5GxI

Nogueira, P. 2011. Motion Capture Fundamentals – A Critical and Comparative Analysis on Real-World Applications. Referenced 14.5.2021. https://pagi-nas.fe.up.pt/~prodei/dsie12/papers/paper_7.pdf

O'Conor, K. 2017. GPU Performance for Game Artists. Article. Referenced 9.5.2021. http://fragmentbuffer.com/gpu-performance-for-game-artists/

Orvalho, V., Bastos, P., Parke, F.I., Oliveira, B. & Alvarez, X. 2012. A Facial Rigging Survey. In Eurographics (STARs) (pp. 183–204).

Osipa, J. 2010. Stop staring facial modeling and animation done right. 3rd ed. Indianapolis, IN: Sybex.

Petty, J. n.d. What is 3D Modeling & What's It Used for? Article. Referenced 7.5.2021. https://conceptartempire.com/what-is-3d-modeling/

Persson, E. 2009. Triangulation. Article. Referenced 10.5.2021. http://www.hu-mus.name/index.php?page=News&ID=228

Pinto, E. & Grossman, A. 2018. Character Rigging & Cinematic Animation in God of War with Erica Pinto & Axel Grossman. Video. Referenced 6.10.2020. https://youtu.be/kCpdVm2bWEI

Polygon 2021. Breath of the Wild's NPCs are actually Miis, modder confirms. Article. Referenced 14.5.2021. https://www.poly-gon.com/2021/1/5/22215263/breath-of-the-wild-npcs-are-miis-nintendo-legend-of-zelda-switch

Russell, E. 2020. UV Mapping Game Characters in Maya. Video. Referenced 7.5.2021. https://learn.unity.com/project/uv-mapping-game-characters-in-maya?uv=2018.1&courseId=5d0b72eaedbc2a001ffed2b2

Spencer, S. 2010. ZBrush Digital Sculpting Human Anatomy. Hoboken: John Wiley & Sons, Incorporated.

Statham, N. 2020. Use of Photogrammetry in Video Games: A Historical Over-view. Games and culture. [Online] 15 (3), 289–307.

Tafuri, S. 2019. Strand-based Hair Rendering in Frostbite. Presentation, Sig-graph 2019. Referenced 9.5.2021. http://advances.realtimerender-ing.com/s2019/index.htm

Takahashi, K. 2017. Cross-fading LOD shader example. GitHub repository. Ref-erenced 12.5.2021. https://github.com/keijiro/CrossFadingLod

Torabi, P. 2019. Skeletal Animation Optimization Using Mesh Shaders. Game and Software Engineering. Blekinge Institute of Technology. Thesis.

TurboSquid n.d. Poles. Article. Referenced 7.5.2021. https://resources.turbos-quid.com/training/modeling/poles/

Unity Technologies 2020. Intro to Working with Control Rigs. Article. Referenced 13.5.2021. https://learn.unity.com/tutorial/intro-to-working-with-control-rigs#5fe5216bedbc2a7ca4374c4e

Unity Technologies 2021a. Normal map (Bump mapping). Documentation. Referenced 8.5.2021. https://docs.unity3d.com/Manual/StandardShaderMaterialParameterNormalMap.html

Unity Technologies 2021b. Level of Detail (LOD) for meshes. Documentation. Referenced 8.5.2021. https://docs.unity3d.com/Manual/LevelOfDetail.html

Unity Technologies 2021c. Materials introduction. Documentation. Referenced 8.5.2021. https://docs.unity3d.com/Manual/materials-introduction.html

Unity Technologies 2021d. Arm & Unity Presents: 3D Art Optimization for Mobile Applications. Article. Referenced 12.5.2021. https://learn.unity.com/tutorial/mipmapping-9873?uv=2019.4&courseId=5f6baa23edbc2a0020abb1c0&projectId=5f6ca0ecedbc2a0020808586

Unity Technologies 2021e. Importing Textures. Documentation. Referenced 12.5.2021. https://docs.unity3d.com/Manual/ImportingTextures.html

Unity Technologies 2021f. Rotation and Orientation in Unity. Documentation. Referenced 13.5.2021. https://docs.unity3d.com/Manual/QuaternionAndEulerRotationsInUnity.html

Valve Software 2019. Wrinkle maps. Documentation. Referenced 14.5.2021. https://developer.valvesoftware.com/wiki/Wrinkle_maps#:~:text=Wrinkle%20maps%20are%20a%20component,Source%202007.

Wilson, J. n.d. The Toolbag Baking Tutorial. Article. Referenced 9.5.2021. https://marmoset.co/posts/toolbag-baking-tutorial/

Yeung, A. 2020. DirectStorage is coming to PC. Article. Referenced 30.5.2021. https://devblogs.microsoft.com/directx/directstorage-is-coming-to-pc/

Zeman, N. B. 2015. Essential skills in character rigging. Boca Raton, Florida: CRC Press.

APPENDICES

Appendix 1. Software products

- Adobe. Photoshop. https://www.adobe.com/products/photoshop.html
- Adobe. Substance Painter. https://www.substance3d.com/products/substance-painter/
- Autodesk. Maya. https://www.autodesk.com/products/maya/overview?term=1-YEAR
- Epic Games. Unreal Engine. https://www.unrealengine.com/en-US/
- Pixologic. ZBrush. http://pixologic.com/features/about-zbrush.php
- Unity Technologies. Unity. https://unity.com/
- Animbot Inc. animBot. https://animbot.ca/home/

Appendix 2. Animations video

https://www.youtube.com/watch?v=oF5A3xW9yxQ