Pipeline for Modeling and Texturing Realtime Graphics

Øyvind Lien

Luleå University of Technology
Bachelor thesis
Computer graphics
Department of Skellefteå Campus
Division of Leisure and Entertainment
Pipeline for modeling and texturing realtime graphics

Øyvind Lien

Luleå Tekniska Universitet – Skellefteå
Computer graphics
Preface
This Bachelor Thesis is written by Øyvind Lien. It completes my 3 year bachelor study in Computer Graphics at Luleå Tekniska Universitet. During 10 weeks in April/May of 2009 I have been located at Turbo Tape Games in Bergen, Norway. My assignment has been to develop and research a pipeline for producing 3D content for Naval War: Arctic Circle, an RTS game for the PC platform. I choose this topic because I have always been interested and fascinated by videogames and game art, which is the reason I choose this education in the first place.

I would like to thank my supervisor Fredrik Sundt Breien for helping me during my stay, giving valuable feedback on my work and patiently answering all my questions. The thanks also goes to the rest of my co-workers at Turbo Tape Games, Jan Haugland, Espen Th. Sæverud and Tor-Inge Jenssen. I would also like to thank Bug Norway for helping me out with office space.

Bergen, May 2009
Øyvind Lien
Abstract
My goal for the thesis was to develop a time and economically efficient pipeline for modeling and texturing content for realtime graphics. Keep a high paced production speed, and focus on simple solutions. At the same time I wanted to avoid the use of extra plug-ins and software for texture baking, UV mapping and such. The pipeline was made to fit the time schedule, resources and graphical profile Turbo Tape Games had for their current project Naval War: Arctic Circle.

The thesis was developed by following the principles of real-time graphics. It focused on simplicity and resulted in a time and cost efficient pipeline, usable in a professional production environment. I found that planning all the steps strictly, using good reference images and re-using as many elements as possible was an important factor.
Appendix

8/16/24 bit image Indicates the number of colors per pixel. 8 bits give 256 colors, 24 bits give 16.7 million colors per pixel. Also known as True Color, the amount of colors the human eye can distinguish. (3)

Adobe Photoshop Photoshop is a graphics editing software developed by Adobe Systems. It is the current and primary market leader for commercial bitmap and image manipulation. It is industry standard for graphics professionals. (4)

Alpha channel The portion of a four-channel image that is used to store transparency information. (3)

Ambient Occlusion A CG shading method that uses a type of global illumination to better compute self-shadowing objects. Often used in computing as part of a multiple-pass rendering workflow, or baked out and added onto a UV space. (4)

Artifact A (usually undesirable) item in an image or on a mesh that is a side effect of the process used to generate or modify that image. (1)

Autodesk Maya A high-end 3D computer graphics and 3D modeling software package, developed by Alias Systems Corporation, but now owned by Autodesk. (6)

Bump Mapping Bump mapping is a computer graphics technique where at each pixel, a perturbation to the surface normal of the object being rendered, is looked up in a texture map and applied before the illumination calculation is done. Normal and parallax mapping are the most commonly used bump mapping techniques. (7)

Diffuse Color Map The base 2D texture image on a 3D mesh. Also referred to as Color Map. (1)

FBX file FBX is a format for storing 3D models. It allows storing of motion data along with texture and mesh information. FBX is supported by most 3D applications and graphics engines. (8)

Game engine A game engine, also referred to as graphics engine, is a software system designed for creating and development of videogames, containing physics calculations and interactions. It imports polygon objects and renders them in realtime. (9)

MB file Short for Maya Binary. A native output file format for Autodesk's Maya. (10)

Mental Ray Mental ray is a production-quality rendering application. Using global illumination, ray tracing, photon emission and caustics to simulate realistic light and shadow casting. Included in the Autodesk Maya 3D application. (11)

NURBS Non-Uniform Rational B-Splines. A class of piecewise parametric curves or surfaces where each curve segment or surface patch is described by a ratio of Non-Uniform B-Spline polynomials. (12)

Pipeline A well-defined set of processes for achieving a certain result. (3)

Pixel A pixel or picture element is the smallest piece of information on a digital image. They are normally arranged in a 2D grid represented by dots, squares or rectangles. For colors a pixel typically consists of a RGB or CMYK scheme. (14)
Plug-in
In computing, a plug-in consists of a computer program that interacts with a host application to provide a certain, usually very specific, function on demand. (13)

Polygon
A plane figure which is a closed contour of straight lines. A basic primitive in the graphical representation of 3D objects. A triangle, also referred to as a tri-gon, is the simplest polygon which can exist in the Euclidean plane. (25)

PSD file
A proprietary format developed and used primarily by Adobe Photoshop, it includes support for many application specific features such as layers. (15)

Rasterization
The task of taking shape based vector graphics info and converting it into a raster image consisting of pixels or dots. This is then printed or displayed on a screen. (16)

Raytracing
A method in computer graphics for calculating waves and particles for realistic lights and shadows. (17)

Resolution
The amount of data that is used to capture an image. The term is typically used to refer specifically to the spatial resolution of a digital image. (3)

RGB
A method of defining colors using Red, Green and Blue. (3)

RTS
RTS or Real time-strategy is a genre of computer games. It does not involve turns, and has a strategical and action based gameplay. (18)

Specular Map
An image determining the brightness and size of specular highlights on a mesh. (1)

Texture
A 2D image represented on a 3D model, often represented on a UV space. (3)

TGA file
Truevision's TGA File Format, often referred to as TARGA. Contains support for Photoshop layers and alpha channel information. The format can store image data with 8, 16, 24, or 32 bits of precision per pixel, maximum 24 bits of RGB and an extra 8-bit alpha channel. (19)

Triangle
The simplest polygon form, also referred to as a tri-gon. (2)

Turtle
A Plug-in for Maya developed by Illuminate Labs. It combines advanced rendering technology and baking functionality. Often used to bake normalmaps, lightmaps or ambient occlusion to a UV space. (20)

Unfold 3D
UV mapping software supporting .OBJ files. Suitable for UV mapping organic surfaces. Has features like path finding and Texture UV matching. (21)

Unity 3D
A multi-platform game development tool. The editor runs on Windows and Mac OS X and can produce games for the Windows, Mac, Wii and iPhone platforms, including browser games. (22)

UV mapping
UV mapping is a 3D modeling process of making a 2D image representing a 3D model. The map transforms the 3D object onto an image known as a texture. In contrast to "X", "Y" and "Z", which are the coordinates for the original 3D object in the modeling space, "U" and "V" are the coordinates of the transformed object wrapped to 2D “UV space”. (1)

Z-buffering
The management of image depth coordinates in 3D graphics. (23)
**Introduction**

Following is the result of my 10 week bachelor thesis research, during this period I was located at Turbo Tape Games in Bergen, Norway. The goal was to create a pipeline for modeling and texturing for realtime graphics. In the dictionary a pipeline is described as: “A well-defined set of processes for achieving a certain result”.

The research involved finding solutions to create 3D models of various ships, airplanes and missiles, and come up with an efficient pipeline for producing them. This involved deciding what maps was necessary, the texture resolution, experiment with bit-depth and what level of detail was suitable. Everything based on the in-game view distance and graphical style for Naval War: Arctic Circle.

For the 3D models I used Autodesk Maya 2009. The texturing was done in Photoshop CS4. Implementation to realtime was done in the game engine Unity 3D, this was a new software experience for me.

This introduction continues with the following topics:

- Background
- Intention
- Approach to the problem
- Turbo Tape Games
- Naval War: Arctic Circle
- Unity 3D
- About Realtime graphics

**Background**

I have been interested in videogames for as long as I can remember. It has always been a goal for me to get into this line of work. I read about Naval War: Arctic Circle in the newspaper and found it very interesting. I sent my portfolio to Turbo Tape Games, they quickly replied and told me they liked what they saw, and was interested in assisting me with my thesis. After a few phonecalls and e-mails I was ready to go. Getting into a real production environment and working with realtime graphics was a new challenge for me, since we only scratched the surface of this topic in school.

**Intentions**

The intention for this thesis was to create a pipeline usable for Turbo Tape Games, in their production of Naval War: Arctic Circle. To achieve this I wanted to use the competence I have learned during my studies, get experience from a professional production environment and improve my technical and artistic skills. I wanted to solve the many problems that appeared during the process and explore the principles of modeling and texturing for realtime graphics. My goal was to use simple solutions, and focus on production speed as a base in the pipeline.
**Approach to the problem**

An independent game developer often has limited amount of manpower due to a low budget. It is important that the production pipeline for 3D graphics is simple and cost efficient. The cost of buying licensed software applications should be kept as low as possible. This means that the pipeline should include as few applications and plug-ins as possible, and only use the ones that are necessary to archive the desired result. In the case of this thesis, one goal was to build the pipeline around using only Adobe Photoshop for textures, and Autodesk Maya for the 3D models. Avoiding the use of software like *Unfold 3D* for UV mapping or *Turtle* for texture baking.

**Turbo Tape Games**

Turbo Tape Games is an independent game developer founded in February 2008. They currently consist of 4 people. Their early work is ranging from small games, interactive museum installments and learning software for schools. Naval War: Arctic Circle is their first major game production. No final release date is set, but it will probably be sometime in spring/summer 2010 for the PC platform.

**Naval War: Arctic Circle**

Naval War: Arctic Circle is an RTS game set in the year 2030. The story is build on presumed historical facts, with superpowers fighting for world domination and resources. The gameplay is set on the Arctic and North-Atlantic oceans.

The game has a realistic approach with physics and sonar data implemented into radar and sensor systems. It also includes a global weather system. The main focus during development is on gameplay. The framework for the game is well into development, with working physics and interactions. But lacks most of the graphical content and some features.

**Unity 3D**

Unity 3D is a multi-platform game development tool. The editor runs on Windows and Mac OS X and can produce games for the Windows, Mac, Wii and iPhone platforms. It can also produce browser games. A good feature for 3D artists is the integrated support for native formats from Maya, 3Ds Max, Blender and such. When a model is edited, it can be refreshed in Unity, skipping the steps of having to re-export and import it again. The license for Unity pro costs 1500 dollars.
About Realtime graphics
Realtime computer graphics involves generating images in real time. The term is most often used in reference to interactive 3D computer graphics, using a processor (GPU) to render 3D objects seen in videogames and similar interactive mediums. (24)

Rendering 3D objects in real time using a graphics engine, is different from rendering non-realtime in a 3D application like Autodesk Maya or 3Ds Max. In non-realtime the application often uses techniques like raytracing to calculate light for realistic purposes, but these can take many hours to render a single image frame. (24)

In realtime graphics, the system has less than 1/30th of a second to calculate an image. In order to do that, the processor cannot afford shooting millions or billions of rays, instead it relies on the technique of z-buffer triangle rasterization. (23)

In a rasterization process every object is decomposed into individual primitives, the most popular and common one is the triangle. These triangles are then rendered onto the screen one at a time. They get positioned, rotated and scaled on the screen, and a special hardware called rasterizer generates the pixels inside each of these triangles. The triangles are then decomposed into small screen units called pixels, that are suitable for displaying on a screen.

The pixels are then rendered on the screen using a particular color. A texture can be used to paint onto the individual triangle, which is deciding what color to output at each pixel based on a stored picture. (16)
Production and methods

Description
Following is the accumulation of the research done before the production started. Looking closer on the game at its current stage, asking questions about the goals for the product, and exploring the necessary principles of modeling and texturing for real time.

Project research
Before starting the graphics production it is important to get a good general overview, look at the style and see what level of detail the game demands. In Naval War: Arctic Circle the player controls units over a very large area, much of the gameplay will be in satellite mode view, with a schematic overlook. This means that the level of detail on the units can be quite low. One of the goals was to be able to run the game smoothly on older computer, not necessarily up to date with next-gen hardware.

Figure 1  Screenshot from the Unity 3D engine, the models are from turbosquid.com

During a battle at sea with massive military units, the number of units in view are few, estimated to be 10-15 at maximum. Ships, airplanes and missiles are non-organic shapes, in most cases with clean textures consisting of paint, and with little color variations.

Because the maximum zoom level is quite far from the model, an estimated 6000 triangles should be enough for the boats and airplanes. If 10 models are in view, 60000 triangles will be rendered in the engine. This is not a huge number in a game. Many games today have 2 million triangles or more rendered in the scene at one time.

A base for the texture resolution was set to 2048x2048 pixels. Textures can easily be scaled down but not up. It is also easier to work in Photoshop at a higher resolution, selections are easier to handle and less pixel bleeding occur.

The missiles however should have a much lower triangle count, since a larger number of units would be rendered on screen. An estimate of 500 triangles per unit, and a texture resolution of 512x512 pixels seemed reasonable. A number of 30 missiles on screen would add 15000 rendered triangles to the scene.

Based on the information presented so far, LOD objects doesn't seem necessary. This was also mentioned in the design document. Creating LOD objects for far, medium and close view is a common technique to reduce the number of triangles rendered by the engine at one time. Since the number of triangles rendered at one time will be limited in this game, including LOD's in the pipeline would increase the production time, and not serve any purpose.
NURBS and Polygons

Some game engines have support for NURBS objects and curves, but polygon objects are commonly used, mainly because they are less demanding to render. (1)

A polygon is a plane figure which is a closed contour of straight lines. A triangle, also referred to as a tri-gon is the simplest polygon which can exist in the euclidean plane (25). The lines are called edges and the edgepoints are called vertices (vertex in singular).

Various types of graphics engines have different ways of outputting 3D object information. Some count polygons, also referred to as quads, while others count triangles as standard (2). The important thing here is that the modeler and programmers use the same term. It is not good if the programmers order an object with a triangle count of 2000, but the modeler misunderstands and creates it with a poly count of 2000. In this case the model will be twice as complexed, since 2000 polys consists of 4000 triangles.

When working with polygons in a 3D application, you can by default see the polygon faces from both sides. This two-sided feature is there to help the artist visualize the object. In realtime, the engine has the capability to do the same thing, but this will cost twice the performance and will create problems when used with per-pixel and per-vertex lighting. It is also important that the face normals is facing the correct way, in most cases toward the camera. If the normals are the wrong way, the object will be invisible, since only the backside is rendered. (2)

NURBS meaning Non-Uniform Rational B-Splines, is a class of piecewise parametric curves or surfaces where each curve segment or surface patch is described by a ratio of Non-Uniform B-Spline polynomials (12). A NURBS objects does not contain edges, faces or vertices. Instead it has editable isoparms, surface patches and surface vertices. (12)

![Polygon and NURBS primitives](image)

Figure 2  Polygon and NURBS primitives
Smoothing groups
Since objects in real time should have a limited triangle count, smoothing groups are used to give the objects a seamless look. This is achieved by assigning the edges as soft or hard, changing the vertex normal rotation angle, and telling the engine what direction the vertex should have. (2)

Avoiding modeling mistakes
It’s a good idea to model with the triangle faces visible in the 3D application, this makes it easier to spot problems early on. Overlapping faces is a common problem. This is when a face has been accidentally duplicated, making two faces lie on the same spot. This may cause visual problems like z-fighting and lightning problems. (2)
Z-Fighting is when overlapping faces are drawn at the same position, and are “fighting” with each other in the graphics card. This creates flickering and artifacts on the object. It is also expensive for the engine to handle. (1)

In modern game engines T-junctions or Stray Vertices are becoming less of an issue. A T-junction occurs when an edge does not share a vertex with another edge (See Figure 4). In a worst case scenario it doesn't only cause visual problems, but also collision and surface detection issues. Stray Vertices is when a vertex is floating in space, this can open up an edge on the model (See Figure 5). The best way to avoid these issues is being careful when modeling. (1)
Optimizing geometry
The basic idea when optimizing an object, is to remove all the edges that doesn't contribute to the shape. Often the edges in-between two end points on a shape is unnecessary. Two identically shaped models can have a very different triangle count (see Figure 6). Figure 7 shows the unwanted edges marked in orange. Figure 8 shows the wireframe when the edges are deleted. After the edges are deleted, the vertices still remain. This is solved by going to vertex mode, selecting the entire model and delete the remaining “floating” vertices.

![Figure 6](image1.png) ![Figure 7](image2.png) ![Figure 8](image3.png)

**Figure 6** Notice the difference in triangle count. Tris = Triangles  
**Figure 7** Deleted edges  
**Figure 8** Improved

Another aspect of optimizing is to remove the faces that are invisible in-game. This can for instance be faces on the underside of an object (see Figure 9).

![Figure 9](image4.png)

**Figure 9** The brown faces above are invisible in-game, and can be deleted
**Ambient Occlusion**

Ambient Occlusion is a shading method which helps add realism to local reflection models, by taking into account attenuation of light due to occlusion. It approximates the way light radiates in real life, creating shadows in cracks and dents, while not affected by the light sources in the scene. In Maya this is calculated using Mental Ray (see figure 10). (5)

Ambient Occlusion can be rendered as a shader and assigned to the object, or baked out in a separate file matching the current UV coordinates of the model (see figure 11). This baked image is added as a separate layer on the texture file. In Photoshop it is blended with the diffuse color map using the multiply blending mode.

Using Maya's *Batch Bake* for this kind of baking can take many hours if the scene is complexed. In those cases using a separate render plug-in like Illuminati's *Turtle* will greatly reduce the render time. In the case of this pipeline however, an external plug-in wasn't necessary, since the Maya baking only took 20 minutes for a 2048x2048 ambient occlusion pass (see Figure 11). Figure 10 shows Ambient Occlusion on the Kirov Battlecruiser produced for the game.

---

**Figure 10**  *The Kirov model produced for the game*

![Figure 10](image)

**Figure 11**  *Baked Ambient Occlusion blended with the multiply blending mode in Photoshop*

![Figure 11](image)
Bump mapping
Bump mapping is a technique where at each pixel, a part of the surface normal of a rendered model is looked up in a heightmap. This is then applied before the illumination calculation is done. This creates a more detailed and realistic surface, similar to the real world. (7)

In realtime graphics normal mapping and parallax bump mapping are commonly used. Normal mapping tracks the surface normals of an object, and encodes the high detail normal information to every pixel in the normal map, that is later applied on a low-poly mesh. When light falls across the object, it appears to be very high in detail (1). Instead of grayscale information a normal map uses RGB values, providing X,Y,Z information to calculate light and surface properties (see figure 12).

Parallax mapping is an enhancement of the bump mapping techniques commonly used in game engines. It is implemented by displacing the texture coordinates at a point on the rendered polygon by a function of the view angle in tangent space (the angle relative to the surface normal) and the value of the heightmap at that point. At steeper view angles the texture coordinates are displaced more, and gives the illusion of depth due to the parallax effects as the view changes. (7)

Displacement mapping in contrast to bump mapping, normal mapping and parallax mapping, use a procedural texture or heightmap to cause an effect where the actual geometric position of points over the textured surface are displaced, adding geometry to the 3D object. (7)

Figure 12 A copy of the Diffuse color map generated as a normal map in the Unity 3D engine.
**Specular map**
Specular mapping represents the amount of specular reflectivity a surface has. It determines the brightness, shininess and size of specular highlights. The is controlled by an image, where dark and light values creates a mask for reflectivity. In Unity 3D this is controlled in the alpha channel, and reads only a 2 channel greyscale image. However in some cases an RGB image can be used, this also determines the color of the highlight.

![Grayscale specular map](image)

*Figure 13  Specular map inside the alpha channel*

**Engine testing**
When working with an unfamiliar game engine it is a good idea to do various tests with the models first, maybe even break some of the “rules” to see how the engine handles it. This will give an overview of possible errors, that might avoid artifacts when the models are rendered in the engine later on. Unity 3D handled soft and hard edges well, this was important to establish early since it was a key factor in how the geometry on the models was planned. Exported animation keyframes also worked great in the engine. Even if Unity 3D handles bad topology, it is still a good idea to avoid it.

*Figure 14  T-junction edges and soft/hard edges  Figure 15  Inside the engine*
The development process
During the development of the pipeline, a total of 5 military units and 4 missiles were modeled and textured. These includes the boats; KNM Nansen, Admiral Kuznetsov, Kirov Battlecruiser, KNM Skjold and the airplane Tupolev TU-95. The general production process were similar for all units, but many improvements to the pipeline was made underway.

The production process is following the principles and techniques described in the pages above.
Result

Foreshortening
The production time and effectiveness evolved during the process. The first model was produced in about 6 days, counting 7 hours effective work each day. This time-frame gradually decreased for each model attempted. The final model took 2 days to produce, illustrating a 70% decrease in production time.

Getting more experience with Maya and learning shortcuts was a factor in this. But the main reason for the increase in productivity was mainly due to improvements in the pipeline. The time spent on UV mapping was drastically decreased, since sewing all the parts together wasn't necessary. Also finding the correct settings for operations like lofting and conversions, and learning about techniques like curves projection on a NURBS object contributed greatly.

The rest of the models in the game are very similar to those made in this pipeline, this means that the same pipeline can be used, and the production time for each model is easy to calculate.

Creating the pipeline

Unit 1: KNM Nansen
The first unit I was assigned to produce was the frigate boat KNM Nansen. What follows below is the process in chronological order.

Reference
Reading about the history of the boat, what missions it was involved in and finding info like measurements and available on-board equipment, helped to map out the general design approach. I was only able to find a partly schematic image for the side view, but not the front and top. I solved this problem by using the measurements, then creating a box matching the length, width and height in reference to each other.

Modeling the basic shape
After some testing I came to the conclusion that using curves to loft the shape of the hull was the most effective approach. Using a polygon box primitive and editing it was time consuming and didn't give a satisfying result. When lofting I found that using the Linear setting worked best for a non-organic shape.

Lofted objects can have polygons as output geometry, but I found that when editing the curves, the topology got out of control and became unnecessarily complexed. A better approach was to output it in NURBS, edit the curves until the shape was correct, then convert the object to polygons.

![KNM Nansen reference](image1)

**Figure 16** KNM Nansen reference

![NURBS Curves when lofted](image2)

**Figure 17** The basic shape lofted
Optimizing
When the basic shape of the boat was ready, I needed to optimize it by removing unnecessary edges and faces. This is done to keep the triangle count as low as possible. First all the faces on one side of the model was deleted, now I only had to optimize half the model. When this was completed, the model was duplicated, mirrored, combined and the overlapping vertices in the center was merged.

![Mirrored and merged](image)

Figure 18  The side reference was assigned to a background plane in side view

Modeling the details
When the main shape was ready, details like antennas and turrets was made out of simple primitives, matching the design from the reference images. To keep the triangle count low, it was important to only create the objects extruding from the boat. Windows and other flat details were added later in the texturing stage.

To be able to rotate antennas, cannons and such in the engine, the smaller objects was parented to the main boat object.

![Optimized and with details](image)  ![Wireframe](image)

Figure 19  Unwanted edges were removed  Figure 20  A wireframe sample
Smoothing edges
To achieve the correct smoothness of the hull, certain edges were smoothed. This is an excellent technique for real time objects, since it creates the illusion of more geometry on the model.

Figure 21  What it looks like with all hard edges  
Figure 22  The side of the hull was softened

UV Mapping
The big areas on the model is of most importance, for this reason they should be assigned more room in the UV space. I made one UV space for these important areas, and a second one for the smaller details. The resolution of the small texture could be scaled down by 50 %.
In some cases identical geometry can share the same coordinates on the UV space, being a good way to save space. If the model requires a normal or lightmap however, the geometry needs individual coordinates. In the case of KNM Nansen, I realized that spending 30 % of the time on UV mapping was unnecessary.

The edges I sewed together didn't make a difference, since the view distance in game was too far away. For a closeup model however sewing is necessary. I also did a mistake by flipping some of the UVs, making them invisible in the engine. I had to go back at a later stage and redo the entire UV mapping. Figure 24 shows the new and improved UV space.

Figure 23  Old UV space with sewn edges  
Figure 24  More areas with automatic mapping
Texturing
Ambient Occlusion was baked with Maya's *Batch bake*. This was added to the UV space in Photoshop with a multiply blending mode. The Diffuse Color Map was created with photo textures from cgtextures.com, combined with filters, blending modes and handpainting. The sharpness of the texture was edited by having a copy of the entire texture set to a overlay blending mode, with a high-pass filter applied (see Figure 25). While setting up the Photoshop layers, I was cautious in the way I sorted and named everything. This can really save time at a later stage, if I have to go back and edit the texture. It is also important to have a structure if another artist continues the texturing sometime later in development.

A grayscale version of the texture was added to a alpha channel, to create a specular map for the engine to read. The texture was downsized to 1024x1024 and stored as .TGA, since this format has low compression and supports alpha channels. Inside the engine, both resolution and bit-depth can be assigned, so there was no reason to change this in the source file.

![Figure 25](image1.png) *The UV space textured*

![Figure 26](image2.png) *Rendered in Maya*

Triangle count
The estimated triangle count from the research phase was 6000 for each model. In the case of KNM Nansen a satisfying result was reached at only 2454 triangles. This serves the goal of keeping the triangle count as low as possible. KNM Nansen due to its stealth capabilities has very little deck details. The triangle count for the other models produced later generally increased, as more details was necessary to keep the shape realistic.

![Figure 27](image3.png)

<table>
<thead>
<tr>
<th>Verts:</th>
<th>1369</th>
<th>1369</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edges:</td>
<td>2564</td>
<td>2564</td>
<td>0</td>
</tr>
<tr>
<td>Faces:</td>
<td>1269</td>
<td>1269</td>
<td>0</td>
</tr>
<tr>
<td>Tris:</td>
<td>2454</td>
<td>2454</td>
<td>0</td>
</tr>
<tr>
<td>UVs:</td>
<td>2483</td>
<td>2483</td>
<td>0</td>
</tr>
</tbody>
</table>

*Faces = Poly or quad count; 1269
Tris = Triangle count; 2454*
Determining scale

A grid scene was set-up to get the correct scale on the boat, and make sure it fitted with the units in the game engine. In the engine 1 metric unit was set to be equal to 1 meter. This was solved by creating a grid in Photoshop, then apply it to a background plane in Maya's orographic sideview. One square on the grid equals 10 meters. For this to work in Maya, the scene's metric units has to be set in such a way so that 10 meters equals 1 grid on the background plane.

The boat was imported and moved so the back part was placed at 0 meters. With the pivot point on the model also placed at 0, the correct scale could be archived by scaling the entire model up. In the case of KNM Nansen it should have a length of 134 meters.

Figure 28  The boat scaled according to the grid

Finishing touches

Now the pivot was centered, and the object snapped to the center of the scene grid. This was done so the model would be centered to the scene when imported to the engine. Transformations also had to be reset, so the correct Scale was set to 1x, 1y and 1z.

At first the model was exported as a .OBJ file, but this created cretin shading problems and UV bugs when imported to the engine. So after experimenting with other formats, .FBX seemed like the best option. FBX allows storing of motion data along with texture and mesh information, everything included in one file. The production time for this model was 6 days, counting 7 hours work each day.

Figure 29  KNM Nansen imported to the engine and rendered
Unit 2: Admiral Kuznetsov
The next unit was the Russian aircraft carrier Admiral Kuznetsov. I initially followed the same process as the previous model, but some changes were made underway. I will only mention the changes done to the pipeline on the next models.

Reference
For this and the next models I managed to find schematics for side, top and front view. The source was The-Blueprints.com, a free website with schematics of military units. This was a big help and increased the speed of the modeling allot.

UV Mapping
On this stage I learned from the mistakes on the previous model. This time I only planar mapped the top and sides of the boat, nothing was sewn together. The rest of the details were mapped with automatic mapping.
Modeling & texturing
This stages was mostly unchanged from the previous model. I had to redo the hull a few times, but learned from the mistakes and got it correct in the end. The schematics really made a difference. The triangle count ended at 4426, mainly because the control tower was done quite detailed.

The landed airplanes and other details on the deck wasn't included, mainly because when these kind of ships are in battle, it doesn't contain a huge number of airplanes on deck. The production time of the entire unit was 4 days, counting 7 work hours each day.

I paid more attention to the texture here, making scratches and wear on the decks asphalt. This gave more life to the model.

![Figure 32](image1.png)  
*The wireframe*

![Figure 33](image2.png)  
*Textured version from the Maya viewport.*
Unit 3: Kirov Battlecruiser
Next in line was another Russian boat, the Pyotr Velikiy, a Kirov class Battlecruiser.

Figure 34  Reference image of the Kirov Battlecruiser

Modeling & texturing
For this unit I decided to add many of the deck details as geometry. This could have been done in the texturing stage, but would have increased the production time, since they had to be painstakingly aligned. These were simple and repeating primitive shapes, not much extra time on the modeling stage was used. It only involved duplicating, scaling and moving the primitives. Since the triangle count on the previous models were lower than expected, I decided that a little extra geometry could be afforded. I also did a change to the UV layout, using only 1 UV space this time. I felt the details on the tower was so small, that an extra space wasn't necessary.

The production time was 2 and a half days, counting 7 work hours each day. The final model counted 6324 triangles.

Figure 35  Ambient Occlusion
Figure 36  Deck details are mainly geometry
Unit 4: KNM Skjold
The next unit was KNM Skjold, a small and superfast stealth missile craft.

![Reference image of KNM Skjold](image)

**Figure 37**  *A reference image of KNM Skjold*

**Modeling & texturing**
This unit had a very simple shape, so I basically made it as a high-poly version, capturing the shape realistically. While researching I found that it had some variations in the camo-texture, based on what area it is operating. Because of this I made two versions, one blue/gray and one brown. The biggest challenge here was the camo-texture. This was solved by closely studying the reference photos, and make filled selections in Photoshop matching the pattern. The various colors were created on separate layers, so they could be changed easily at a later stage.

The production time including both units were 1 and a half day, counting 7 work hours each day. The triangle count ended up at 1979.

![Blue/gray version wireframed](image)

**Figure 38**  *Blue/gray version wireframed*

![Brown version](image)

**Figure 39**  *Brown version*
Unit 5: Tupolev TU-95
To get a new challenge I wanted to try the pipeline on an airplane. The steps would be generally the same, but some new challenges appeared in regard of a more round shapes. The goal here was to keep the triangle count within a reasonable limit.

Figure 40  A dark green version of the Tupolev Tu-95 (Bear)

Modeling & texturing
I had to redo the body of the airplane a few times, the first versions were to simple and sharp. I came to the conclusion that projection a NURBS curves onto the body was the most effective way to create the wings (see figure 41). I also had to redo some of the UV map at a later stage, since the cylindrical mapping approach didn't give a satisfying result, by creating to much texture distortion. A better way was an automatic mapping with the seam “hidden” at the bottom of the body.

Figure 41  Using the project curve on surface tool in Maya

Animations and result
A total of 8 propellers, some contra-rotating, were animated with a 360 degrees rotation, all working perfectly inside the Unity 3D engine. The final result ended up with a triangle count of 7226. The main reason for the high count is all the propellers, and the smooth shape of the body.

Figure 42  Wireframe in Maya  Figure 43  Textured version
Missiles
The final task was to create 4 missiles. These should be very low in triangle count, since maybe as many as 30 units would be visible in-game at the same time. At the same time they should have some detail, since an in-game tracking camera feature was planned.

The process
I used a cylinder polygon primitive for the basic shape, and UV mapped everything with automatic mapping. The texture resolution was set to 256x256, more than this would have been unnecessary based on the scale in-game. Since the basic cylinder shape and wings were very similar on all 4, they were re-used in most cases.

Time and triangle count
The production time was between 2 hours to 4 hours for each, it varied based on complexity. The triangle count was generally around 250. For instance the BGM.109 Tomahawk ended up at 178 triangles (see Figure 44). The RBS-15 MK3 was a bit higher at 352 triangles (see Figure 45).
Exporting to Realtime
Using .OBJ files as first intended, turned out with problems like artifacts, tangled up UV space and other rendering issues in Unity 3D. After some testing .FBX files showed to be a good option without errors. This format allows storing of motion data along with texture and mesh information, everything included in one file (If the *embed media* function is used) (8). A box for smoothing groups and animations can be checked before exporting.

![Figure 46](image)

The FBX exporter in Maya has many options
Graphics engine implementation
This importing to realtime process was quite straight forward. I found that the textures got sharper when the *aniso level* was increased, but this also increases the rendering load. A value of 2-3 seemed reasonable as a start. The *Filter mode*, *texture size* and *bit-depth* could also be changed at this stage (22). Since the final watershader for the game wasn't completed yet, these settings in regards to the current *draw cycles* has to be tweaked at a later stage.

Creating bump maps
In the engine the textures are listed in the assets editor. These texture were duplicated and generated into a normal map, giving bump and texture details to the models. The shader type was set to *bumped specular*. The strength of the bump could be changed with the *bumpiness* slider.

It is hard to notice much of a change in the texture due to the far view distance, but since a increase to the render load wasn't seen, I decided to include it. This feature didn't add production time to the pipeline, since it only took a couple of minutes to set up.

![Inspector settings inside Unity 3D](image)

*Figure 47* The inspector settings inside Unity 3D
Creating specular maps
The specular maps were created in Photoshop, by duplicating the entire texture into a new layer. This layer was desaturated, and the levels were changed so the whites and blacks got more contrast. This copy was then assigned to a new layer in the alpha channel. Since the final lighting in the engine scene wasn't completed, the specular maps in the case of this game has to be tweaked at a later stage, matching the final in-game lighting. This can easily be done at a later stage in development.

Figure 48  The regular texture of the Tupolev TU-95 airplane
Figure 49  The specular map
The final result
Following are renders and real time screenshots of all the models created during the development of the pipeline.

Engine renders

Figure 50  All the units lined up inside Unity 3D

Figure 51  Pyotr Velikiy: The Kirov class Battlecruiser
Figure 52  Topview of the Tupolov-TU 95

Figure 53  Daytime render of the frigate KNM Nansen. 
KNM is short for: Kongelige Norske Marine
The Individual Units

KNM Nansen

Figure 54  *KNM Nansen from Maya. Rendered with Maya Software and default lighting*

Admiral Kuznetsov

Figure 55  *Close-up Maya Software render of Admiral Kuznetsov's steering house*
Pyotr Velikiy: Kirov class Battlecruiser

**Figure 56**  Maya Software render

**Figure 57**  Side view Maya Software render
Tupolev TU-95 (Bear)

**Figure 58** Tupolev TU-95 Maya Software render

**Figure 59** Topview render of Tupolev TU-95, Maya Software
KNM Skjold

**Figure 60** KNM Skjold; brown camo edition, Maya software render

**Figure 61** KNM Skjold; blue/gray camo edition, Maya software render
Missiles

Figure 62  From left: Naval Strike, Boeing Harpoon, RBS-15 MK3 and BGM-109 Tomahawk

Figure 63  Maya software render of the missiles
Discussion
Strengths of the pipeline
The strength of this pipeline is that it has a clear overlook on all the production aspects, and can easily be adapted and used on the remaining models in the game. With a production schedule of 2 days (14 working hours) per model, they are cost efficient to produce. The fact that the pipeline evolves around using only Autodesk's Maya and Photoshop CS4, contributes to minimizing the production cost, keeping the licensing cost to a minimum.

Before the production was started, an estimate of 6000 triangles for the boat and airplane models were made. The actual number turned out lower for most of the units. The missiles were estimated to have 500 triangles, the result here was about 50% less.

Possible improvements
All the models were created from scratch, the only exception was a ship cannon that was reused a couple of times. A possible improvement to the pipeline would be to also re-use the hull of the ship, since this shape is quite similar on many of the models this game demands.

A weakness in the pipeline is that it relies on schematic drawings of the units. If these cannot be obtained, the production time will increase. Being more accurate at the reference stage might have decreased the development time further. Some of the images gathered through sites like Google image search were named wrong, this resulted in an inaccurate design, since pictures of slightly different boats were used as modeling reference. These involve minor changes in the design that was hard to spot at first glance. In these cases the model had to be changed, and certain areas remodeled.

Conclusion
During this bachelor thesis I have attained experience in planning and developing 3D models for a game. I have improved my technical and theoretical knowledge of polygon & NURBS modeling aimed for realtime graphics. I have also received experience from a professional production environment. This involved receiving a certain production task, using and evolving my competence and making changes to the models based on feedback from the rest of the development team. Due to the theme of the project I have gained considerable knowledge of the history, use and design of military units.

When working in a small company, you get involved in almost all the aspects of the production in some way or the other. It has given me a clear view on the many aspects of creating an entertainment product, in terms of time schedules, finances and game design. Working close with programmers has granted me a better view on how they think, and how they work with a realtime graphics engine, and what kind of requirements they demand from the artist.
References
The literature and Internet links are numbered to show where in the thesis they are used.

Literature

ISBN 0-7357-1409-6


ISBN 978-0-12-370638-6

Internet

4 Wikipedia – Adobe Photoshop
http://en.wikipedia.org/wiki/Photoshop

5 Wikipedia – Ambient Occlusion
http://en.wikipedia.org/wiki/Ambient_occlusion

6 Wikipedia – Autodesk Maya
http://en.wikipedia.org/wiki/Autodesk_Maya

7 Wikipedia – Bump mapping
http://en.wikipedia.org/wiki/Bump_mapping

8 Wikipedia – FBX format
http://en.wikipedia.org/wiki/FBX

9 Wikipedia – Game engine
http://en.wikipedia.org/wiki/Game_engine

10 Wikipedia – MB format
http://www.autodesk.com/

11 Wikipedia – Mental Ray
http://en.wikipedia.org/wiki/Mental_ray

12 Wikipedia – NURBS
http://en.wikipedia.org/wiki/NURBS

13 Wikipedia – Plug-in
http://en.wikipedia.org/wiki/Plug-in_(computing)

14 Wikipedia – Pixel
http://en.wikipedia.org/wiki/Pixel
Wikipedia – PSD file
http://en.wikipedia.org/wiki/PSD_file#File_formats

Wikipedia – Rasterization
http://en.wikipedia.org/wiki/Rasterization

Wikipedia – Raytracing

Wikipedia – RTS Game
http://en.wikipedia.org/wiki/RTS_game

Wikipedia – TGA format
http://en.wikipedia.org/wiki/Truevision_TGA

Illuminati Labs - Turtle
http://www.illuminatelabs.com/

Unfold 3D
http://www.polygonal-design.fr/e_unfold/

Unity 3D
http://unity3d.com/

Wikipedia – Z-buffer

Wikipedia – Real-time graphics

Wikipedia – Polygon
http://en.wikipedia.org/wiki/Polygon

Textures
http://www.The-Blueprints.com
http://www.cgtextures.com

Other related
http://www.navalwargame.com