Linköpings universitet – Campus Norrköping
TNM032: Modelleringsprojekt
Supervisor: Anna Lombardi
2005-01-16

TANXIM^{2k5}

Peter Asplund, *petas874@student.liu.se*, 821028-8975
Henrik Hedman, *henhe894@student.liu.se*, 820506-0034
Jonas Hellsten, *jonhe289@student.liu.se*, 820106-1952

Abstract

The course “TNM032: Modelleringsprojekt” is a project where one is supposed to make a simulation of a physical system. This report describes the methods used to simulate a main battle tank. The main part of the simulation is the engine and the suspension. The simulation is implemented using the programming language C++ and OpenGL. The simulation is represented by a 3D-visualisation where the user can control the movement of the tank. The necessary simplifications needed to make the simulation and some of the problems encountered are described.

Table of Contents

List of Figures

1. Introduction

Physical simulation combined with 3D-graphics is a popular area for both games and scientific visualisation. This report describes the way in which we choose to work during our project.

1.1. Background and Aim

The main object of the course "TNM032: Modelleringsprojekt" is to make a simulation of a physical system. The aim of the project is to simulate the Swedish Main Battle Tank: Stridsvagn 122. The included parts are making a 3D model of the tank and simplified physical models of the engine, the suspension and the turret. Then we used the programming language C++ with the help of *OpenGL* to implement the system in 3D. This report will illustrate how the project was made and which parts are included in making a simplified simulation of Stridsvagn 122. The primary goals we wanted to achieve were to have a tank with an engine and a suspension. The secondary goals were to make the turret rotate, be able to fire a projectile and adding sound.

1.2. Method

Initially a plan of how and when the different parts of the system would be implemented were constructed. Then with consideration to the group-members different experience with C++, the work was divided depending on the ability of each member of the group. Data about the tank has been gathered from soldf.com and *AerotechTelub*. Estimations have been used for the data not available. *AerotechTelub* were kind to receive us and answer questions about the tank. The programming language used is C++ with *Irrlicht Engine* as the rendering engine. We have used *Irrlicht Engine* as it simplifies the animation and allowed us to spend more time on the simulation. *Irrlicht Engine* uses the *OpenGL* Application Program Interface (API). An FTP-server was set-up to make the latest version of the source code and information available to the members in the project.

1.3. Structure

In this document the physical models and implementations used are explained, beginning with the limitations of the simulation and the description of the models used. Then continuing with the implementation of the simulation and animation and ending with a discussion. Finally an appendix is included showing the source code and constants given by *AerotechTelub*.

2. Simulation Simplifications

Due to the lack of information and the limited time, the simulation is simplified and some approximations of the real world have been made. To have a more accurate simulation would demand more of the computer and would be more time consuming to implement.

- The constants used in the simulation were received from *AerotechTelub* (see Appendix) or approximated to give a life-like result.
- There is no collision detection, and dynamics for the wheels are left out. The tank can move through anything and its movement is limited to one plane (XZ-plane). It cannot leave the ground nor can it go up an incline.
- The weapon is not elevated above the back of the tank, when the turret is rotated around.
- Only internal friction in the models is present. The ground does not effect the movement of the tank.
- When turning the tank there is no loss in momentum. The turning is simplified which results in an inaccurate centripetal force.
- The lack of data for the suspension forced us to make approximations of the spring and damper constants.
- The engine is implemented using a simplified bond graph and the constants are approximated.

3. Description of models

The model consists of several different parts, and each takes an input and computes an output. The following parts represent the different sub-models we have chosen to divide the tank into.

3.1. Engine

The data *AerotechTelub* presented us with was very sparse, which forced us to make approximations and to find our own data. We knew the power of the engine, but not the curve of acceleration or the engine efficiency depending on different revolutions per minute (RPM). First we sat down and drew up a couple of ideas how it could work, and made bond graph representations of those ideas. We assumed that there was internal friction and internal momentum, a transmission, and then ground friction and external momentum, for example the weight of the tank. Because of the causality, we had to insert a C-element after the transmission in the bond graph, which felt correct, since the track of the tank must be elastic to some extent. The model looked like the following:

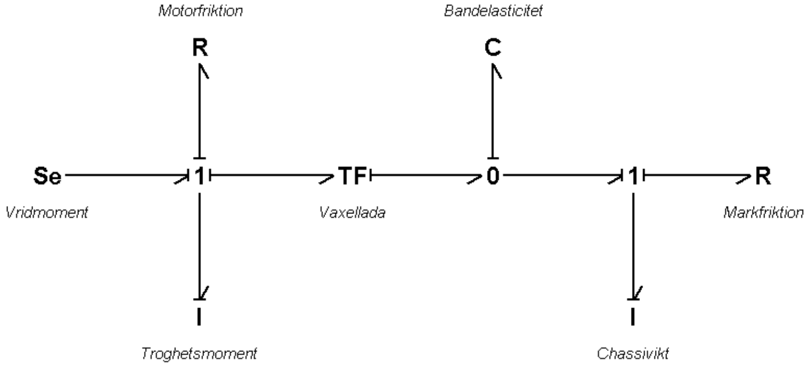

Figure 1. Bond graph for the initial engine

After a week of work, when the model had been tested and we supposed it was complete and finally implemented it into the graphical environment, we discovered it was inaccurate. The C-element made the tank behave like it was driving on a big rubber band even though our simulation in 20-Sim had been correct. The only thing we could do was to increase the spring constant, up until the simulation became stiff, which made the simulation unstable. Back to the drawing board: we had to rethink the model. Thankfully, we came up with a new model, only a bit simpler than the previous. Instead of three state-of-space descriptions, we only had one. When we validated the model in 20-Sim it was almost exactly the same as our old one. Our course book says, "Don't fall in love with your models", and it is a very good rule. After implementing the new model, the engine was working the way it should. The model looked like the following:

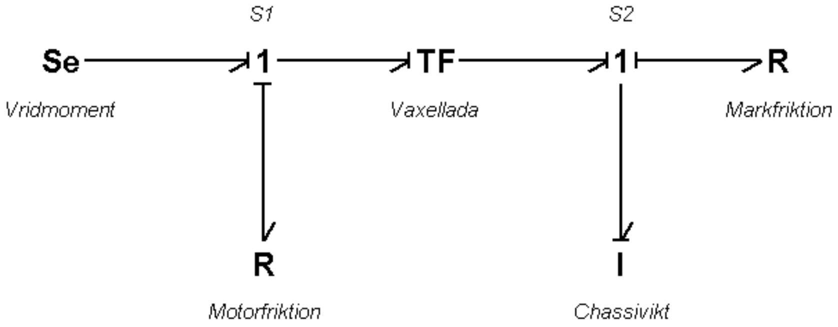

Figure 2. Bond graph for the final engine

The transmission is like a transmission of a car, with two gears in reverse instead of one. It has one neutral and four forward gears. There are limits in the transmission; it is not possible to go from forward to reverse if the speed is not lesser than 0.5m/s, and the other way around. The brakes are applied as an R-element to the S-junction that represents the exterior of the tank. The brake force is multiplied with the speed, which makes the tank break more the faster it goes. This leads to a behaviour that is not quite realistic; for example the tank never comes to a complete stop. To represent the efficiency/RPM we approximated a curve by multiplying the torque with a small equation. The force starts at 50% and grows linearly to 100%, depending on RPM.

3.2. Suspension

We had little knowledge about how the suspension worked on the tank. The information accessible was that it included a damper and a torsion bar. The torsion bar was connected with the opposing wheel and the damper was tilted some degrees. No data about the damper or the torsion bar was available. We also found out that the dampers were not present at all the wheels. After some discussion we decided to simplify the suspension by removing the torsion bar and adding a spring instead and also have a non-tilted damper on all the wheels.

We started by setting up a bond graph for the suspension using a spring and a damper, both with and without a mass on the wheels. We tested several bond graphs in 20-sim, but no one worked correctly when implemented. Finally we left the bond graphs and used basic physics, while using the constants from the 20-sim tests. The suspension is calculated using Hook's law in combination with a damper. Hook's law: $F = -k*x$, where F = *Force exerted by the spring*, k = *spring constant* and x = *displacement from equilibrium position.*

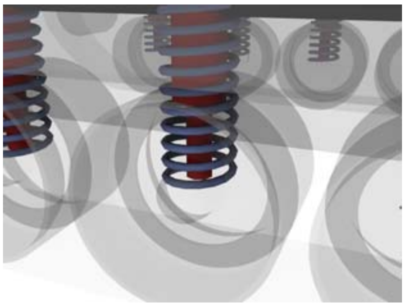

Figure 3. Suspension: including spring and damper

3.3. Forces acting on the tank

Several different forces affect the tank; these are forces from the engine, the springs, gravity and centripetal forces. The forces contribute to the translation and the rotation of the tank. A special case is the propulsion of the tank; the velocity is calculated separately and is always applied in the direction of the tank. This is not physically correct, but a correct steering of the tank was not the main goal of the project.

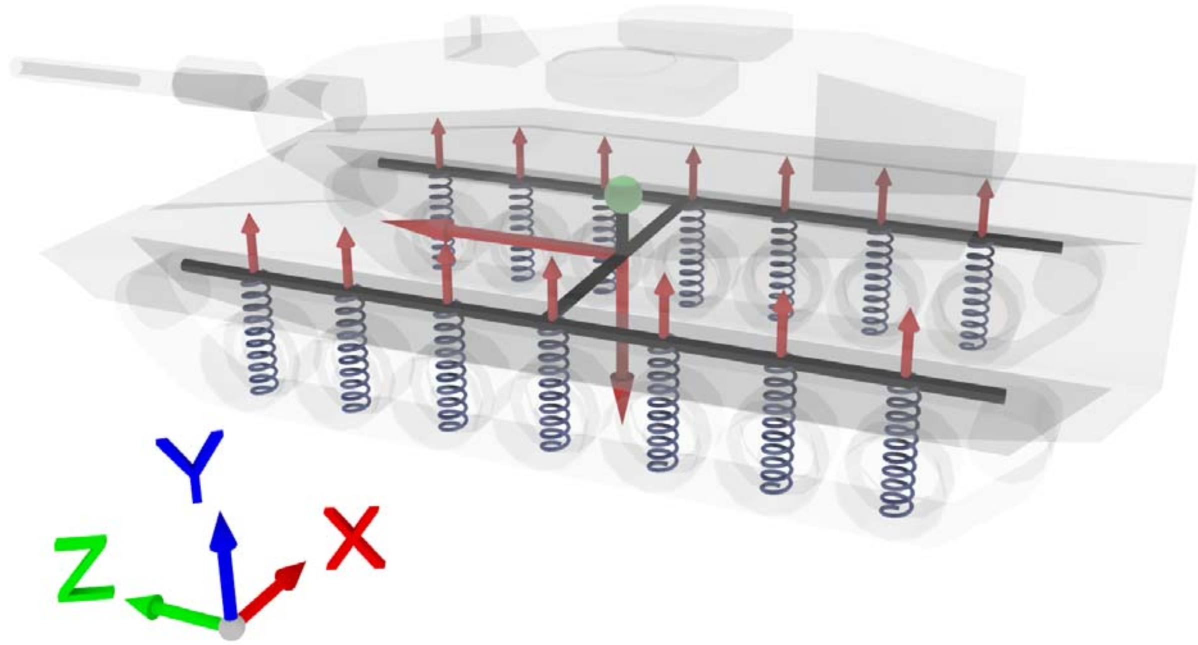

Figure 4. Schematic representation of the forces acting on the tank

To get correct suspension a spring was placed at a point above every wheel, and the distance to the XY-plane was measured to get the elongation of the spring. The forces were then calculated by the suspension function. The sum of all the forces from the spring is calculated and the gravitational force is added. The resulting force is then used to translate the tank by calculating the acceleration according to Newton's second law. The acceleration was also integrated twice to give the new position of the tank.

To get a correct rotation of the tank, the moment of inertia was calculated by approximating the tank as a box with homogenous mass distribution. Next the moment from the engine was added. The moment arm was at first calculated as the height difference of the first fixation point for the suspension and the Centre of Mass (CM). This was later corrected by averaging the height of the fixation points that where in the middle of the length on each side. The moment from the springs was simply calculated by using the local position coordinates as a moment arm. This was possible as the CM was located at the origin. At first the rotation was only calculated in the X-axis, but was later extended to the Z-axis. A jump function was added by adding a large force to one randomly selected suspension. The purpose was to give more visual information how the suspension is working. This showed that the rotation was very slow and long lasting. To get a faster response the moment of inertia was divided by a constant. Last, a centripetal force was added by using the *Force = mass * velocity * angular-velocity* equation which was derived from equations from *Physics handbook*. The centripetal force uses the same method as the engine to calculate the moment arm. The first test showed that it worked, but the centripetal force got so strong that it rolled the tank over. The reason for this is mostly due to the speed the tank is able to turn in. To keep some of this effect but to make it harder to roll over the force was divided by 50.

Turning the tank is handled separately from the rotation in the X- and Z-axis but in a similar way. A simple model was built in 20-sim with an intensity source, an I-element, an R-element and an S-junction. The moment of inertia was already calculated, and thus it was only necessary to estimate the parameters for the source and the R-element to get the right final value and a reasonable step response. This was done for all the different gears as the turning radius differed. The implementation was very straightforward; a damper is added to the moment force, which is then used to accelerate the mass. The turning of the tank is highly simplified. A correct model would probably take into account the different velocities of the tracks to calculate the rotation. Another difficulty was the lack of information of the detailed workings of the steering system.

3.4. Turret

The rotation of the turret is done the same way as the turning of the tank. A force is applied to a simple system with inertia and a damper. The elevation and dumping of the gun is achieved by rotating the gun with a fixed angular velocity. A non-linearity is added to limit the rotation to 20 degrees up and 10 degrees down. In reality, the turret is highly regulated to achieve precise control over the direction and thus the aiming of the weapon. Another feature that is missing is the elevation of the weapon when the turret is rotated to the rear of the tank. This is done to keep the weapon from hitting the backside of the tank.

4. Numerical method

The physics for the simulation are recalculated for every frame. To make this possible in real time, a numerical method must be used. While working on the models for the engine and the suspension in 20-sim several numerical methods were tested. We tested both the Euler and Runga-Kutta differential equation solver. We implemented improved Euler, but the increase in complexity was not necessary. The method we choose to implement is Euler. This method is sufficient for our simulations and it showed no noticeable difference in stability when decreasing the steps per second to as low as 20.

We used a variable step size in the simulation to keep the simulation going in synchronization with the animation. Thus the step size is equal to the time it takes to render one frame of the animation. The step size is calculated by dividing one with the number of frames per second (FPS). The FPS is fetched from the rendering engine.

5. Animation

We decided early to use an open rendering engine to relieve us of the burden of programming an advanced rendering engine in a low level language which none of us had any experience of. The first candidate was the popular and advanced open source rendering engine *Object-Oriented Graphics Rendering Engine (OGRE)*. Then no one of us was able to compile and start the engine after following the tutorials and guides on the engine's home page, the engine was abandoned in favour of *Irrlicht*, an advanced rendering engine almost as popular as *OGRE*. *Irrlicht* was considerably easier to use and thus it was decided to use it for the project. *Irrlicht* can use either the *OpenGL* or the *Direct3D* API; we choose to utilise the *OpenGL* mode as it has superior support for different software platforms. *Irrlicht* is used like most other scene graphs; nodes are arranged in a hierarchy where a transformation of a node affects every node beneath it. Most nodes have a model attached; these models are what we actually see, others represent cameras or the environment. Some nodes have no function associated to them and are only used to simplify the animation.

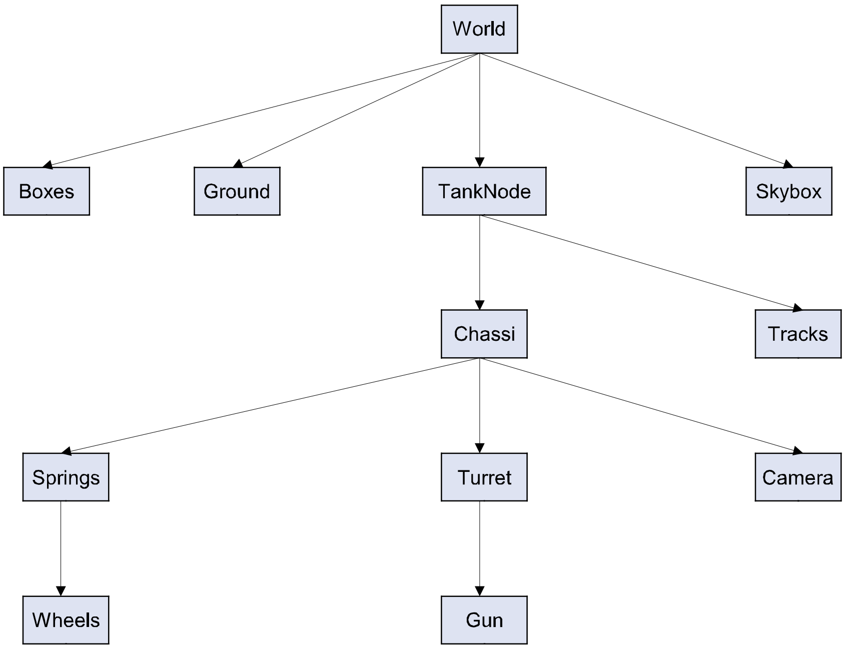

Figure 5. Scene graph of the animation

The models were created in *3ds max* and the textures were either made in *Photoshop*, *3ds max* or were free to use. Models were exported to the 3ds-format and were then imported by *Irrlicht*'s internal importer.

6. Software and Requirements

This was the software used in constructing Tanxim and an estimation of the software and hardware required to run it.

6.1. Software used

3D-models created with *3ds max 7*
Adobe Photoshop 7
Compiled with *Bloodshed Dev-C++* 4.9 for Windows and *g++ (GCC)* 3.3 for
GNU/Linux.
Scene graph done with *Irrlicht Engine 0.7*
Estimation of system and models done in *20-sim*

6.2. Requirements

The simulation is tested on both *Windows XP/2000* and *GNU/Linux* running latest stable *Gentoo* with 2.6.X kernel. The required hardware (and software):

Windows (XP/2000) or *GNU/Linux*
1.5 GHz processor or similar
256 MB RAM
64 MB video card with OpenGL 1.2 support
Irrlicht Engine 0.7 (included)

7. Running the simulation

To compile Tanxim for *Windows*, the easiest way is to use *Dev C++*, and load the Tanxim.dev-file in the src-directory, and press compile. In *GNU/Linux*, use the makefile (See README for more details). The simulation is started with the executable file in the "bin" folder; either "tanxim.exe" or "./tanxim", depending on your operating system. The keyboard controls the tank and turret and the mouse is used to control the camera.

Movement:

W	Forward
S	Backward
A	Left turn
D	Right turn

Gear:

Q	Gear up
E	Gear down

Turret rotation:

I	Downward
K	Forward
J	Left
L	Right

Miscellaneous:

Space	Pause
C	"Jump" (Put a force on a random suspension)
ESC	Quit
R	Reset position of tank

8. Conclusion and Discussion

We have created a tank simulation, with an engine and a suspension, where you can drive the tank and control the turret. We are all satisfied with the outcome of the project. All the primary goals set up for the projects were accomplished. The secondary goal controlling the turret was also implemented. We did not have time to finish the ballistic subsystem or adding sound.

The parameters of the simulation are rough approximations. We consider this to be accurate enough especially as we have no way of validating the system.

There are a few things that can be improved in the program. By adding collision detection we can interact with the world, and adding friction to the track against the ground would add more realism. The weapon's elevation and dumping could be improved to behave as the real tank. There are also a lot of graphical improvements to be done, to enhance the visual aspects of the simulation.

Simple physical system can be used to great effect to enhance the realism in visualisation, games and other graphical applications. This is a quickly advancing area, and one of the most popular features in computer games right now.

In the beginning our ambitions were too high. We started out with a model that was too complex, which made the numerical errors intolerable. By simplifying the system we realised we could maintain the accuracy, dispose the errors and enhance the performance. This is something we should have thought about earlier in the development.

References

Interview with Mikael Axelsson at *AerotechTelub*, 1 November 2004.

Literature

Nordling, Carl & Österman, Jenny (1999). *Physics handbook*. Studentlitteratur. 6th edition.
Glad, Torkel & Ljung, Lennart.(2004). *Modellbygge och Simulering*. Studentlitteratur 2th edition.
Magnus Merkel m.fl. *Rapportskrivning. En lathund för studenter* (2002). (kompendium)

Internet

These references have been used continually during the whole project.

Irrlicht Engine 0.7 API [www] Hämtat från <http://irrlicht.sourceforge.net/docu/index.html> 25 October 2004 - 16 January 2005
Irrlicht Engine Tutorials [www] Hämtat från <http://irrlicht.sourceforge.net/tutorials.html> 25 October 2004 - 16 January 2005
C++ Reference Project [www] Hämat från <http://www.cplusplus.com/ref/> 25 October 2004 - 16 January 2005
Strv122 Stridsvagn 122 Leopard 2 [www] Hämtat från <http://www.soldf.com/strv122.html> 25 October 2004 - 16 January 2005

Appendix

Data supplied by AerotechTelub

Velocity at gear:

Forward 1	max 15km/h
Forward 2	max 30 km/h
Forward 3	max 45 km/h
Forward 4	max 70 km/h
Reverse 1	max 15 km/h
Reverse 2	max 30 km/h

Max break momentum 20kNm

Smallest turn radius at gear:

Forward 1	10 m
Forward 2	15 m
Forward 3	20 m
Forward 4	25 m
“Centrum turn”	0 m

Chassis length	8 m
Chassis width	3,5 m
Turret length	5 m
Turret width	3 m
Barrel length	4 m
Barrel diameter	20 cm
Total height	3 m

Max elevation of weapon	20 degrees
Max dumping of weapon	10 degrees

Total weight	60 ton
Turret + Cannon	20 ton
Chassis	40 ton

Estimated Constants

Engine:

Internal friction	105
Internal momentum	100
Ground friction	40

Transmission at different gears

Reverse 2	-1/0.1923
Reverse 1	-1/0.0962
Forward 1	1/0.0962
Forward 2	1/0. 1923
Forward 3	1/0.2885
Forward 4	1/0.4487

Suspension:

Spring coefficient	1/0.000004
Damper	20000

Chassis:

Moment of Inertia

X-axis	38125 (adjusted to look better)
Y-axis	68450
Z-axis	32720 (adjusted to look better)

Turning moment at different gears

Reverse 2	15E6
Reverse 1	10E6
Neutral	48E6
Forward 1	10E6
Forward 2	15E6
Forward 3	24E6
Forward 4	28E6