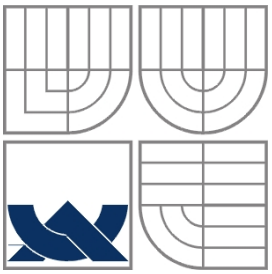
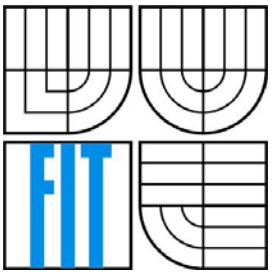


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

INTERAKTIVNÍ EDITOR A PROHLÍŽEČ ANIMACÍ

INTERACTIVE ANIMATION EDITOR AND VIEWER

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

BC. MICHAL DUCHÁČ

VEDOUCÍ PRÁCE
SUPERVISOR

ING. JAN PEČIVA

BRNO 2007

ZADANIE

LICENČNÁ ZMLUVA

Abstrakt

Príchodom výkonných grafických staníc počítačová animácia rýchlo vytlačila tradičné metódy animácie. V súčasnosti, počítačová animácia má mnoho aplikácií, napr. video hry, kinematografia, média, predpoveď počasia a mnohé iné. Táto diplomová práca sa zaoberá rôznymi technikami požívanými pri vytváraní počítačových animácií. Najrozšírenejšou technikou je metóda kľúčových snímkou. Táto metóda je podobná technike používanej pri tradičných animáciách vytváraných rukou. Základné princípy animácie pomocou kľúčových snímkou sú popísané v tejto práci. Práca ďalej obsahuje návrh a popis implementácie interaktívneho editora animácií založenom na tejto metóde. Editor používa Kochanek-Bartelsové spliny pre interpoláciu hodnôt medzi jednotlivými kľúčovými snímkami.

Kľúčové slová

Počítačová animácia, pohyb, kľúčové snímky, interpolácia, tradičná animácia, priama kinematika, spätná kinematika, CGI, Open Inventor, Coin3D, VRML, Slerp, Squad

Abstract

Since the introduction of high end graphical workstations, computer animation has quickly replaced the traditional means of animation. Nowadays computer animation has many applications e.g. video games, motion picture industry, media, weather forecasting and many others. This master thesis discusses various techniques used to created animations using computers. Keyframing, is the most common approach in computer animation. Borrowing its name from the concept of traditional hand animation, the workflow process remained the same. Basic principles of animation using key-frames are explained and an Interactive Animation Editor solution based on keyframing is proposed and the implementation of this editor is described. Editor uses the Kochanek-Bartels interpolation of values between each key-frame.

Key words

Computer animation, motion, parametric key-frames, interpolation, traditional animation, forward kinemics, inverse kinematics, CGI, Open Inventor, Coin3D, VRML, Slerp, Squad

Citácia

Ducháč, Michal. *Interactive animation editor and viewer*. Brno University of Technology. Brno, 2007, Master thesis.

INTERACTIVE ANIMATION EDITOR AND VIEWER

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením Ing. Jana Pečivu.

Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

……

Michal Ducháč

22.5.2007

Poďakovanie

Chcel by som poďakovať svojmu vedúcemu Janovi Pečivovi za neoceniteľné rady a pomoc pri vypracovávaní tejto práce.

Table of Contents

Table of Figures

1 Introduction

1.1 Background

Computer graphic, as a subfield of computer science, is concerned with digitally synthesizing and manipulating visual content. It plays key role in different areas of today's life. Designers, architects, technologist use various kind of CAD tools in order to make their work more effective, and more efficient. Graphical representation of complex simulations enables more intuitive analysis of acquired results. This method is used by well known institutions such as NASA to visualize results of their simulation of newly discovered space phenomenon (e.g. collision of two black holes), but also by common organizations such as your local hydro-meteorological institute. In the field of medicine computer graphic is used to help surgeons better and safer perform complicated surgeries, but perhaps the most well-known application of computer graphic is entertainment in form of animated movies and video games. Entertainment business is good example of very rapid development in computer graphic. While CGI effects were used only in highly budgeted movies and only in few shots in the past, very few today's movies get by without them; some do not even contain a single shot, which was not altered using CGI effects.

There are several major sub-problems in computer graphic including:

- Description of the shape of an object (modeling)
- Description of the appearance of a surface (shading)
- Description of the motion of an object (animation)
- Creating an image of an object (rendering)

Modeling describes the shape of an object. The two most common sources of 3D models are those created by an artist using some kind of 3D modeling tool, and those scanned into a computer from real-world objects. Models can also be produced procedurally, physical simulation, or by using other techniques.

Shading is the process of describing surface appearance. This description can be as simple as the specification of a color in some color space or as elaborate as a complex shader program which describes various appearance related attributes across the surface.

Rendering converts a model into an image either by simulating light transport to get physically-based near photorealistic images, or by applying some kind of different method in order to create non-photorealistic images, often artistic images.

Animation refers to the temporal description of an object, i.e. how it moves and deforms over time. There are several ways to describe changes. Popular methods include key-frame based methods, inverse kinematics, and motion capture. Physical simulation is another way of specifying motion.

This master thesis looks closely at computer animation, with the goal to create a simple interactive animation editor.

1.2 Problem description

The objective of this master thesis was to create a simple interactive animation editor and animation viewer. For the purpose of implementation, the Coin3D graphics toolkit was used for the 3D visualization part, and Qt toolkit for the application interface part. This project was also trying to determine and evaluate animation capabilities of VRML standard.

The goal of this master thesis was to:

- Get a general understanding of representing 3D scene using Open Inventor an VRML formats
- Read through animation capabilities of VRML format.
- Design simple animation editor based on Open Inventor and VRML
- Propose advanced methods for animation editing
- Implement these methods
- Publish the project

1.3 Disposition

This report consists of eleven chapters and two appendixes. Chapter 1 and 2 describe the general concepts of computer animation and the most common methods used in computer animation. Current software, which is most commonly used to create computer animations today, is described in chapter 3. Chapter 4 discusses the animation capabilities of open inventor and VRML formats. Chapter 5 presents the proposed design of the animation editor. Chapter 6 and 7 describes the implementation of interpolation of position and rotation using Kochanek-Bartels splines. Chapter 8 describe in more detail the implementation of the animation editor and chapter 9 describes the basic usage of the application. Chapter 10 concludes the whole report and chapter 11 presents the list of references, which were used in this report. Appendix A contains screenshots of the animation editor and appendix B contains the list of application shortcuts.

1.4 Abbreviations

CAD	Computer-aided design
CGI	Computer generated Imagery
CT	Computed tomography
IAnimEd	Interactive Animation Editor
NASA	National Aeronautics and Space Administration
OpenGL	Open Graphics Library
SGI	Silicon Graphics, Inc.
Slerp	Spherical Linear Interpolation
Squad	Spherical Cubic Interpolation
UI	User interface
VRML	Virtual Reality Modeling Language

2 Computer animation

2.1 From traditional animation to computer

In general concept, animation is a set of images, which, when displayed sequentially at a given rate, give us feeling of motion. Animated images are an outstanding way to capture one's imagination. They are a powerful tool, through which people are able to tell various kinds of stories, they can bring to life odd, and in real life often nonliving things or they can visualize images, which exist only in our imagination. What book could do through a detailed description taking up several lines or even pages of text, animation can show in one single image. But creating this illusion of life is rather complicated. In order to create smooth and continues motion, each image in animation, often also called as frame, has to naturally blend with the other images in the animation sequence.

Traditionally, animation was created by hand; each frame of the action was drawn separately. Traditional animation, like many other form of animation begins with the storyboard. This is a kind of script for animation, written in form of images, which show sequence of major action and illustrates the expressions of the characters. It is often supplemented with words describing the action, expressions or phrases that the characters are supposed to say. The sound track of the animation was usually already finished and animators use it to determine timing of the animation. In older animations, background scenery was often stationary and the characters and other moving objects were painted on transparent foils called cels so they could be laid up over the background.

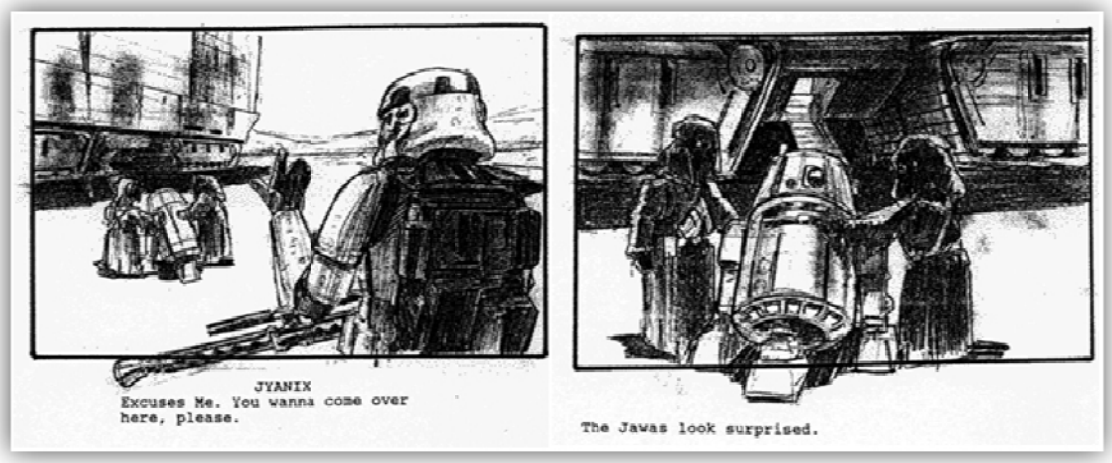


Figure 2-1: Sample storyboards from 'Troops' – a short film set in the Star Wars universe by Kevin Rubio & Co., Art Director - Eric Hilleary.

Most of the traditional animations were created by using method called keyframing. In this method, the lead animator creates the most important frames, also called as the key frames, and other animators create the frames in between. The most basic computer animation tools help animators to create some of these frames of the animation. Other animation tools were developed to composite together multiple layers of the frames of the animation, similar way that layers of cels are laid up in hand animation. More information on traditional animation can be found in (1).

In addition to tools, which helped animators with traditional animation process, the computer also introduces new areas for the use of animation. Computer animations are generated in real-time for purposes of video games and other interactive media. Though realistic rendering and animation techniques an interactive character can be seamlessly blended with real world footage. There are several techniques that are used in the process of creating complex computer animations. According to (2) they can be divided into two classes:

- Two dimensional (2D) – focuses mainly on image manipulation
- Three dimensional (3D) – creation of virtual words in which characters and objects move and interact

As described in (3) there are two ways of considering computer animation and its evolution. The first approach corresponds to an extension of traditional animation by the use of the computer. The second approach corresponds to simulation methods based on laws of mechanics and other laws of physics. Each method is more suited for a different application, neither is strictly better. For example character animation is easier created using a keyframe method than using mechanical laws.

Figure 2-2: Still image from Final Fantasy: Spirit Within

Nowadays, computer animation is also used to create animated movies such as Toy Story, Shrek or Finding Nemo. These are rendered mainly in non-photorealistic way. Toy Story was the first feature-length completely computer-animated movie. It was released in the United States on November 22, 1995 by Disney, and it was created by Pixar Animation Studios. Although there were attempts to create feature-length photorealistic CGI movies, they did not end up with much success. Worth mentioning is the movie Final Fantasy: The Spirits Within released on July 13, 2001 which was also the fist serious attempt to create photorealistic CGI humans. Despite the aggressive promotion, the movie ended up with serious losses, effectively bankrupting its creator, the Square Pictures.

2.2 Two-dimensional animation

Two-dimensional (2D) animation techniques contribute a great deal to computer animation by providing the tools used for sprite-based animation, blending or morphing between images, embedding graphical objects in video footage, or creating abstract patterns from mathematical equations.

Sprite animation is most common form of two-dimensional animation. Sprite is a set of images that are composited over a background to produce the illusion of motion. The animation is created as a sequence of images; each displays the moving object in different position throughout the sequence. With current graphic hardware, sprite animation can be done very fast. Sprite animation is most commonly used in interactive media, where speed is more important than realism. This form of animation is also commonly used in mobile phones to create animated backgrounds.

Morphing is a form of animation where an image is seamlessly changed (or morphed) into another. It is often used in movies to change one person into another through some magical or technological means. In early movies the morph effect was achieved simply by cross-fading from the motion picture of one actor to another. The major drawback of this technique was that the actors had to stay practically motionless in front of the background that not changes or moves. Since the early 1990s, this has been replaced by computer software to create more realistic transitions. The most notable uses of this technique are Michael Jackson's music video Black or White and the movie Terminator 2: Judgment Day. Morphing can be used in 3D as well, which is known as 3D volume morphing. In this case one three-dimensional model is directly morphed into another. 3D morphing overcomes several shortcomings of the 2D morphing, such as more accurate lightning or invariability to camera position. Morphing software continues to advance today and many programs can automatically morph images that correspond closely enough with relatively little instruction from the user.

Figure 2-3: 3D volume morphing of human head into orangutan head. Source and target volume were obtained by CT scans. Images are from (4)

Embedding graphical objects into an existing image allows new elements to be added to a scene. One of the most notable uses of this technique is the Jurassic Park movie, where many of the dinosaurs were generated by CGI and then added to the real footage. Object can be also removed from the scene.

Mathematical equations are often used to create abstract motion patterns. Fractals are well known example of functions that create motion patterns, which can be rather attractive.

2.3 Three-dimensional animation

Three-dimensional animation deals with constructing of a virtual world in which object and characters move and interact. The 3D scene has to be modeled, animated and finally rendered. Briefly stated, modeling deals with creating and describing individual characters and objects in the scene. Animation specifies the temporal behavior of each character and object in the 3D scene. This includes changes in position and orientation (motion), changes in object's description and others. Rendering then converts the animation into images. The process of modeling and rendering are practically independent from the animation process. However, there are some modifications that are required.

In order to generate motion of an object, not only the static description of object is needed, but also information about how object moves is necessary. There is one commonly used method to provide the user with this additional information and that is by so called articulated model. Articulated model is a tree-like hierarchical structure, which consist of collection of objects joint together.

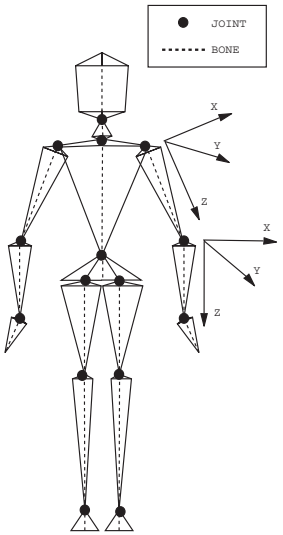


Figure 2-4: Articulated model of a human male.

Each object affects location of all objects that are below in the hierarchy. For example, the movement of the elbow in the Figure 2-4 will also affect the objects in lower arm and hand and fingers. The object at the top of the structure is called root, and it affects the position and orientation of entire model. The skeleton model is used for the animation only. The model to which is the skeleton attached moves according to the movement of the skeleton. When the model is rendered, its skeleton is not visible.

Particle systems are another type of model that is used in computer animation. These systems are used to simulate certain fuzzy phenomena, which are otherwise hard to reproduce using other

rendering techniques. Each system is usually formed by an emitter, which generates particles according do defined rules. Particles move through the simulation system according to specified laws, which often correspond to physical laws, such as gravity, particles can collide with each other or others object in the scene. Particles are used to simulate various gaseous phenomena (smoke, clouds and fog), liquids (flow of water), explosions, even falling leaves and other natural phenomena or special effects.

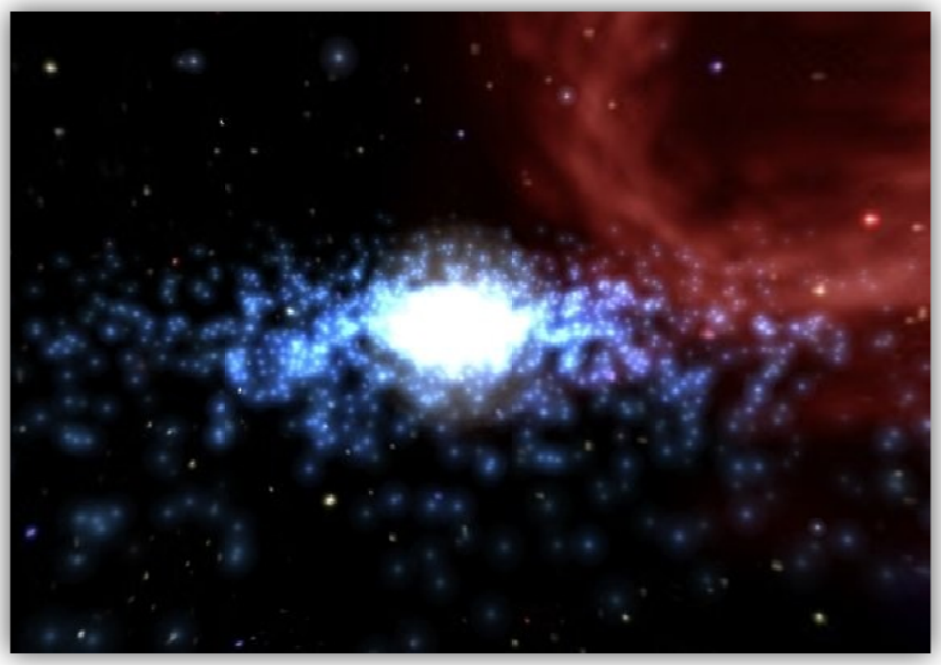


Figure 2-5: A particle system used to render a galaxy

Soft body dynamics is used to animate deformable object. This means that the shape of object can be changed by the influence of the external forces. There are many dynamic forces that have direct influence on an object's behavior (friction, gravity, collisions, springs, wind, etc.), and using soft body dynamics can create an almost authentic and believable illusion of deformable object. Clothes, hair, sand, and water are the same examples of objects that are animated using soft body dynamics.

Animation is often displayed in specific rate of frames per second, usually 24 or 30. The resulting animation is therefore sampled. In real-time computer animation each frame is usually perfect representation of scene in specific time. If the animation is showing rapidly moving objects, it can create unnatural “jumpy” effects, because high frequencies are sampled to low frequencies. This problem is called aliasing and it is well known in signal or image processing.

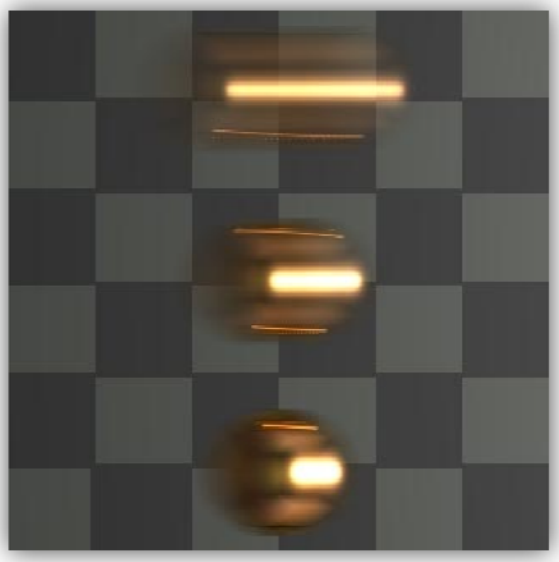


Figure 2-6: Rendered frame with motion blur effect

Motion blur method is used to solve this problem. Moving object is rendered in several positions it had during the time period represented by single frame. Resulting image of the object looks blurred. Although it may seem odd, this technique improves the quality of resulting animation and thus using motion blur creates more realistic animations. Motion blur is also applied, when camera itself is moving. From the camera point of view, the whole scene is moving so the motion blur is applied to the whole scene, not just a single object, blurring the whole resulting frame in that process.

2.4 The art of motion

Although it may seem that specifying motion is rather easy, the opposite is true. It is in human nature to skillfully observe motion and to easily detect those that are unnatural or improbable. Even a simple task to animate bouncing tennis ball can prove to be rather difficult.

Various kinds of techniques have been developed to specify animation. Popular animation method is keyframing. It provides great control over result, but this method require some skills, since it provide little to ensure the naturalness of the resulting motion. On the other hand, procedural methods or motion capture provide tools to ensure the naturalness of the result, but offer less control.

2.4.1 Keyframe animation

Borrowing the name from the art of traditional animation, the workflow of the keyframing animation method is basically the same. The animator creates the key frames, which are the most important phases of the specified motion and the computer computes the in-between frames and determines position of animated object in each frame. In the example of bouncing tennis ball, the animator would position the ball in several key positions e.g. the starting frame, the frame where the ball hits the floor, the frame of the highest position that the ball reaches when bounces of the floor, again the

frame where the ball hits the floor and so on. The remaining frames are calculated by computer, using specified interpolation method. This process is called “in-betweening” or simply “tweening” Linear interpolation is the simplest interpolation method, but often not efficient enough, since the resulting motion is inconsistent, with abrupt changes in velocity of the object. In reality different methods of interpolation are used, such as spline or cubic interpolation.

In reality, a parametric keyframe animation is often used. Each entity (object, camera, light) is described by set of parameters. These include position, orientation, scale, color and many others. The animator creates the key frame and by specifying specific appropriate set of parameter values at a given time. Parameters are then interpolated and each frame is individually constructed using these calculated parameters.

2.4.2 Forward kinematics

The essential concept of forward kinematics animation is that the positions of particular parts of the model at a specified time are calculated from the position and orientation of the object, together with any information on the joints of an articulated model. So for example if the object to be animated is an arm with the shoulder remaining at a fixed location, the location of the tip of the thumb would be calculated from the angles of the shoulder, elbow, wrist, thumb and knuckle joints. Three of these joints (the shoulder, wrist and the base of the thumb) have more than one degree of freedom, all of which must be taken into account. If the model were an entire human figure, then the location of the shoulder would also have to be calculated from other properties of the model.

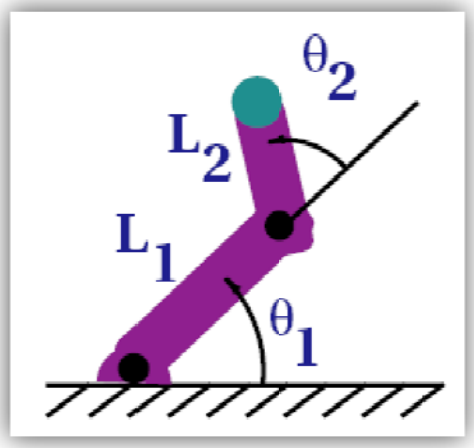


Figure 2-7: Forward kinematics

When using inverse kinematics the angles are specified by animator. The computer calculates the position of the end joint:

$$x = L_1 \sin\theta_1 + L_2 \cos(\theta_1 + \theta_2) \quad (1)$$

$$y = L_1 \sin\theta_1 + L_2 \sin(\theta_1 + \theta_2) \quad (2)$$

Forward kinematics animation can be distinguished from inverse kinematics animation by this means of calculation – in inverse kinematics the orientation of articulated parts is calculated from the desired position of certain points on the model. It is also distinguished from other animation systems by the fact that the motion of the model is defined directly by the animator – no account is taken of any physical laws that might be in effect on the model, such as gravity or collision with other models.

2.4.3 Inverse kinematics

Inverse kinematics is another technique that can make the specification of individual keyframes easier. This technique aids in the placement skeleton models by allowing the animator to specify the position of one object. The position of each object above it in the hierarchy is then computed automatically. In contrast to forward kinematics animation, where each movement for each component must be planned, only the starting and ending locations of the limb are necessary.

For example, when one wants to reach for a door handle, their brain must make the necessary calculations to position his limbs and torso such that the hand locates near the door. The main objective is to move the hand but the many complex articulations of several joints must occur to get the hand to the desired location. Similarly with many technological applications, inverse kinematics mathematical calculations must be performed to articulate limbs in the correct ways to meet desired goals.

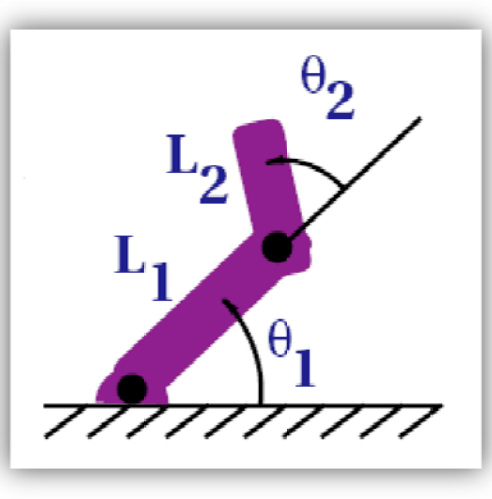


Figure 2-8: Inverse kinematics

When using inverse kinematics, the position of end joint is specified by animator. The computer calculates the angles:

$$\theta_2 = \frac{\cos(x^2 + y^2 - {L_1}^2 - {L_2}^2)}{2L_1L_2} \quad (3)$$

$$\theta_1 = \frac{-(L_2 \sin\theta_2)x + (L_1 + L_2 \cos\theta_2)y}{(L_2 \sin\theta_2)y + (L_1 + L_2 \cos\theta_2)x} \quad (4)$$

Commercial animation packages include inverse kinematics and interpolation routines designed specially for animating human characters. These tools take into consideration such factors as maintaining balance, joint angle limitations, and collisions between the limbs and the body. Although these techniques make animation easier, keyframed animation nevertheless requires that the animator intimately understand how the animated object should behave and have the talent to express that behavior in keyframes.

2.4.4 Procedural methods

Procedural method uses algorithms for specific desired type of motion, in order to create animation. Computer follows the steps of the algorithm to generate motion. There are two great advantages of the procedural methods over the keyframing techniques:

- They provide tools to easy to generate family of similar motions
- They can used for systems that would be too complex to be animated by hand

Group of procedural methods that are based on laws of physics, or an approximation to those laws are called physically based simulations. The realism of the simulated motion depends on the realism of the simulation model. If the model captures the relevant physical law, the simulated motion will feel realistic. Although the realism is often considered to be an advantage, there are cases when simulation model is too complex to create, or when it is more useful to use other techniques. This is especially the case, when greater flexibility is required.

Simulation systems can be divided into two categories: active and passive. Passive systems have no internal energy, and changes in their behavior are a direct result of external forces that influence the system. Passive simulations are usually used when the behavior of the system is determined by physical laws and its initial state. Such systems include for example water, cloth, hair and leaves.

Active systems have an internal source of energy and can change their behavior at their own discretion. Such systems include for example people, animals and robots. These systems are obviously more complex to model, as their behavior is not only determined by physical laws, but also by their internal control systems. In addition to that, behavior of muscles or motors has to be specified. Control system must be designed to allow the model to walk, run, or perform other actions.

Procedural methods are also use to simulate motion of group of objects. Various group behavior algorithms are used in this case for example to simulate crowds of people, flocks of birds, etc. Although procedural methods are currently computationally too expensive to generate motion in real-time for complicated scenes, advances in computer technology may render this possible.

2.4.5 Motion capture

Another technique of creating motion is through motion capture. This technique is based on sensors, or markers, which are attached near each joint of a performer. The markers are used to capture motion of the performer. The recorded data can be then used to create motion in a computer animation. The motion capture computer software records the positions, angles, velocities, accelerations and impulses, providing an accurate digital representation of the motion.

Figure 2-9: A dancer wearing a suit used in an optical motion capture system

Complex animation of human character can created in quite sort time, especially when compared to other animation method such as keyframing. Also motion capture can capture “secondary animation”, which is both complex and time consuming for the traditional animators to create. For example, a quick movement of the head by the actor might cause his hip to twist slightly. This nuance might be understood by a traditional animator but be too time consuming and difficult to accurately represent, but it is captured accurately motion capture.

Although motion capture may seem an ideal way to create motion in computer animation, in has it disadvantages. While it is quite easy and common to use motion capture on a human, using this technique on animal such as horse or lion is difficult. Also manipulating the recorded data is rather complicated; it is often easier to re-shoot the whole scene. To record some motions may be difficult, physically demanding or even dangerous for the performer. Although motion capture produces "realistic" movement, hand animation often allows for stronger applications of traditional techniques like squash and stretch, secondary motion, and anticipation, creating characters with greater impact and personality. Finally the motion capture equipment is very expensive to be used by wider public.

Despite its disadvantages motion capture is often used technique in movies to capture motion for CGI effects, and in video games for football, basketball or ice hockey.

2.5 The art of interpolation

In this chapter, the most common interpolation methods used in computer animation are described. The whole animation is represented by a series of values k(t) where t is integer and it denotes number of the frame. There are a number of well-known techniques for interpolating between two values k(t) and k(t+1). The value *u* represents the fractional part of t in the equations below and it acquires values that 0 ≤ u ≤ 1. Also k(0) and k(1) represents k(t) and k(t+1) respectively.

2.5.1 Linear interpolation:

Linear interpolation is the most common method of interpolation. Certainly its speed of calculation is an advantage, but sharp changes of gradient at each keyframe can be visually disturbing.

The equation for the linear interpolation between two values is as follows:

$$k(u) = k(0) + u\big(k(1) - k(0)\big) \qquad (5)$$

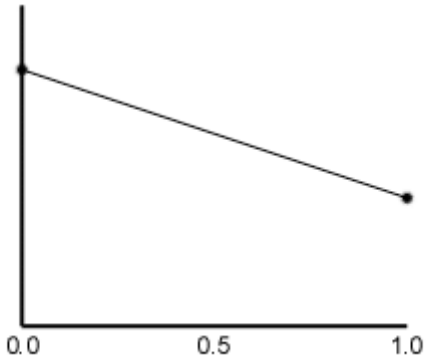


Figure 2-10: Linear interpolation

2.5.2 Cubic interpolation

This interpolation overcomes the gradient change problem, as it has C(1) continuity (continuity of both position and gradient). However, it achieves this by reducing the gradient to zero at each keyframe, so the interpolation of a series of co-linear values of *k(t)* would produce a curve with a peculiar oscillating gradient.

$$k(u) = k(0).(2u^3 - 3u^2 + 1) + k(1).(3u^2 - 2u^3) \qquad (6)$$

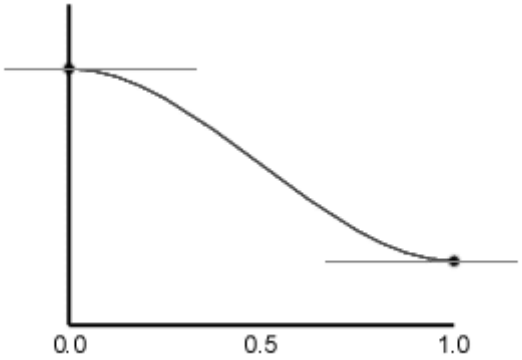


Figure 2-11: Cubic interpolation

2.5.3 Catmull-Rom spline interpolation

Catmull-Rom spline interpolation requires the gradients of the tangents at *k(0)* and *k(1)*, denoted *m(0)* and *m(1)*:

$$k(u) = k(0).(2u^3 - 3u^2 + 1) + m(0).(u^3 - 2u^2 + u) + k(1).(3u^2 - 2u^3) + m(1).(u^3 - u^2) \quad (7)$$

The equation presented above is actually Cubic Hermite Spline. For a Catmull-Rom spline with *n+1* points, each curve have a starting point *k(i)* and an ending point *k(i+1)* with starting tangent *m(i)* and ending tangent *m(i)*, the tangents are defined by:

$$m(i) = \frac{1}{2}(k(i+1) - k.(i-1)) \quad (8)$$

By modifying the cubic interpolation in this way, the curve can tend to a tangent other than 0. When an appropriate tangent is chosen, the resulting curve fits the keyframes very smoothly.

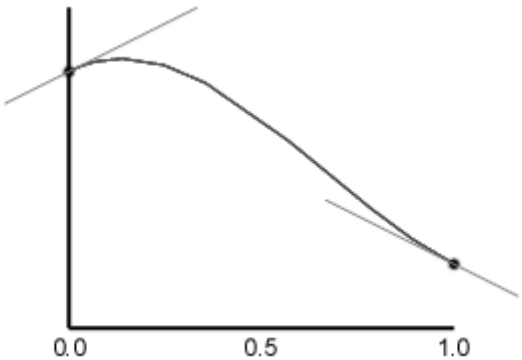


Figure 2-12: Catmull-Rom spline interpolation

2.5.4 Kochanek-Bartels spline interpolation

Kochanek-Bartels spline is (similarly to Catmull-Rom spline interpolation) based cubic Hermite spline, but it adds tension, bias and continuity parameters defined to change the behavior of the gradients. For a Kochanek-Bartels spline with *n+1* points, to be interpolated with *n* cubic Hermite curve segments, each curve have a starting point *k(i)* and an ending point *k(i+1)* with starting tangent *d(i)* and ending tangent *s(i+1)* defined by:

$$s(i) = \frac{(1-t).(1+b).(1-c)}{2}\big(k(i) - k(i-1)\big) + \frac{(1-t).(1-b).(1+c)}{2}\big(k(i+1) - k(i)\big) \quad (9)$$

$$d(i) = \frac{(1-t).(1+b).(1+c)}{2}\big(k(i) - k(i-1)\big) + \frac{(1-t).(1-b).(1-c)}{2}\big(k(i+1) - k(i)\big) \quad (10)$$

where *t* is the tension, *b* is the bias and *c* is the continuity parameter. The tension parameter, *t*, changes the length of the tangent vector. The bias parameter, *b*, changes the direction of the tangent vector. Finally the continuity parameter, *c*, changes the sharpness in change between tangents. Setting these parameters to zero will give a Catmull-Rom spline.

3 Current animation software

In this chapter, the most commonly used animation software is described. First two products are commercial solutions developed by Autodesk Media and Entertainment, an entertainment division of Autodesk, Inc. Third product is an free / open source solution developed by Blender Foundation.

3.1 Maya

Academy Award winning Autodesk Maya software is one of the world's most versatile 3D modeling and animation solutions. Maya, used in many feature movies today, is named from the Sanskrit word meaning illusion and is a popular, proprietary integrated 3D software suite, evolved from Alias PowerAnimator. One of Maya's most appealing factors to large studios is its openness to third-party software, enabling rapid development of proprietary software manipulations and extensions.

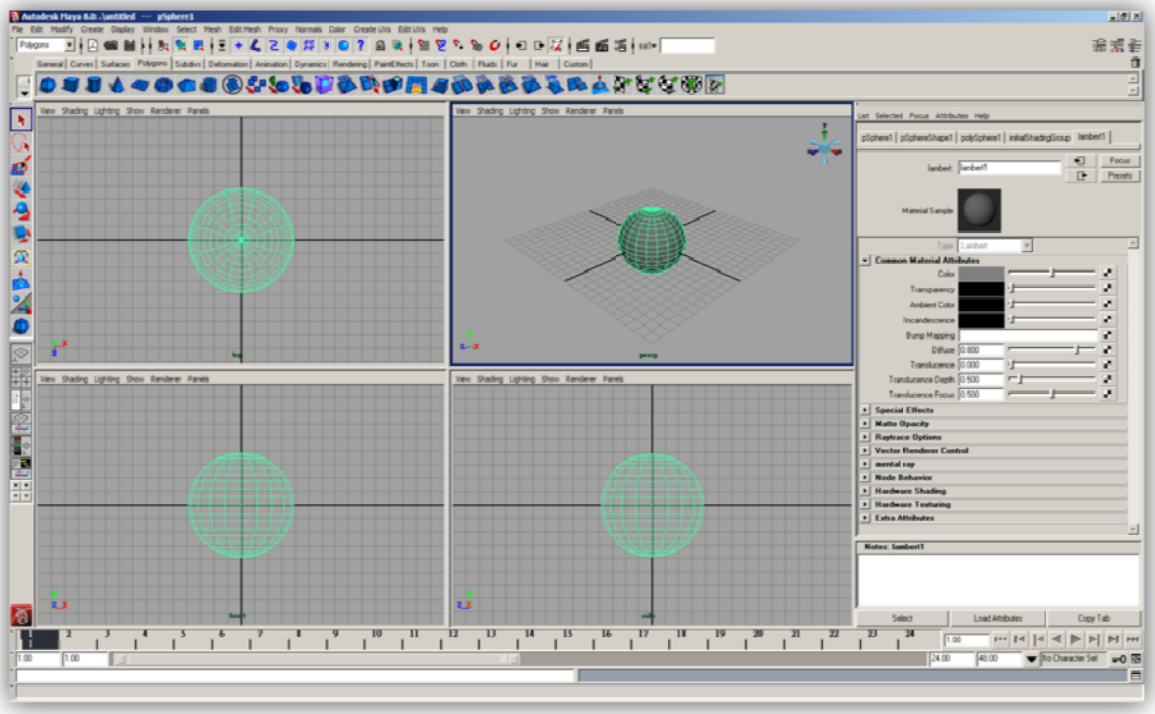


Figure 3-1: Screenshot from Autodesk Maya 8.0

Maya was originally developed by the company Alias, which was acquired by Autodesk in 2006. Maya offers various modeling, animation and rendering techniques. Animation capabilities of Maya include keyframe animation, nonlinear animation, path based animation, motion capture, skeletons (FK, IK, Full Body, IK Solver), Skinning, Deformers, particles dynamics, soft and rigid body simulations.

3.2 3ds Max

3ds Max is a professional 3D modeling, animation and rendering software suite which provides advanced tools for character animation, game development, design visualization and visual effects production. 3ds Max supports animators, designers and game developers with a unified object-oriented platform, customizable real-time interface, multiple-processor support and 3D graphics

acceleration capabilities, including extension via a wide range of plug-ins and specialized products such as Ball's character studio.

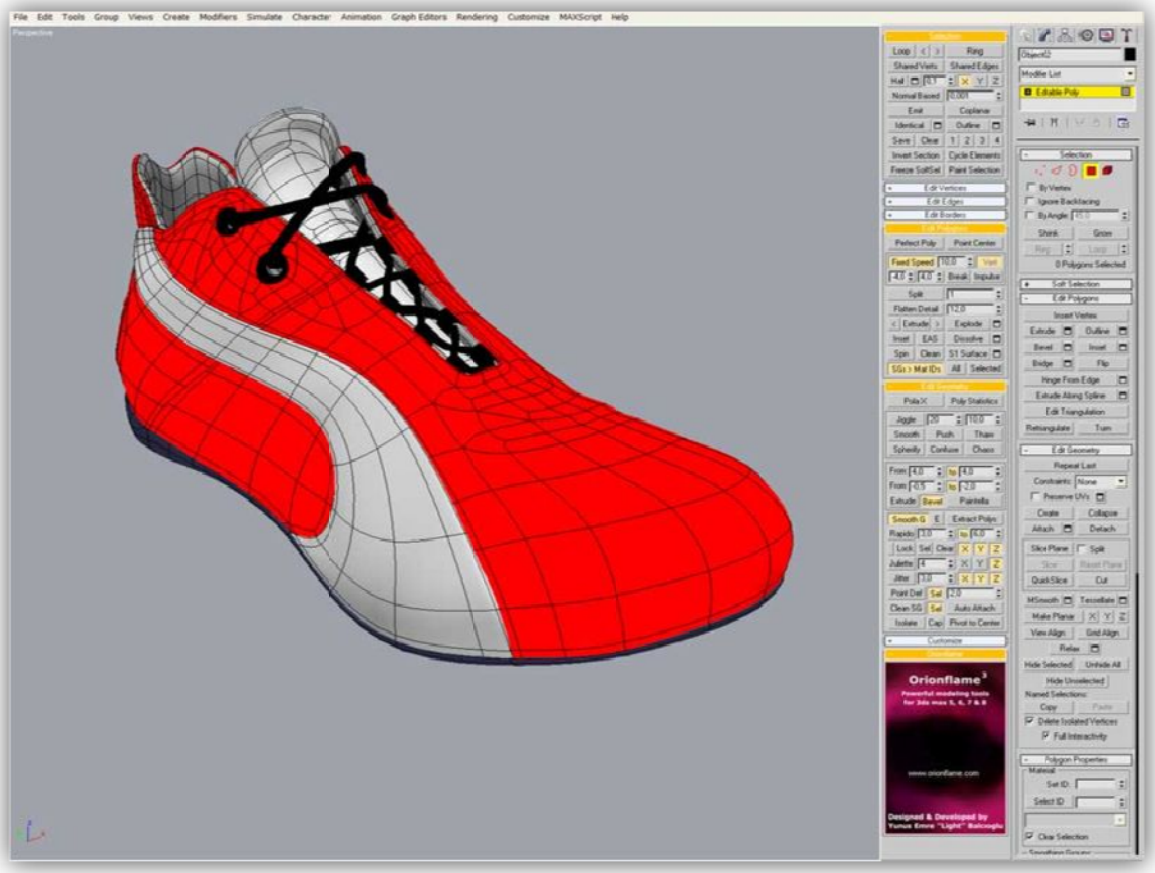


Figure 3-2: Screenshot from Autodesk 3ds Max version 8

3.3 Blender

Blender is a free program originally developed the company Not a Number Technologies (NaN) as shareware program. After NaN went bankrupt it was released as free software. Blender is now a free/open source program and it is being actively developed by the Blender Foundation.

Keyframed Animation tools including inverse kinematics, armature (skeletal), hook, curve and lattice-based deformation, shape keys (morphing), non-linear animation, constraints, vertex weighting, soft body dynamics including mesh collision detection, fluid dynamics, Bullet rigid body dynamics, particle based hair, and a particle system with collision detection.

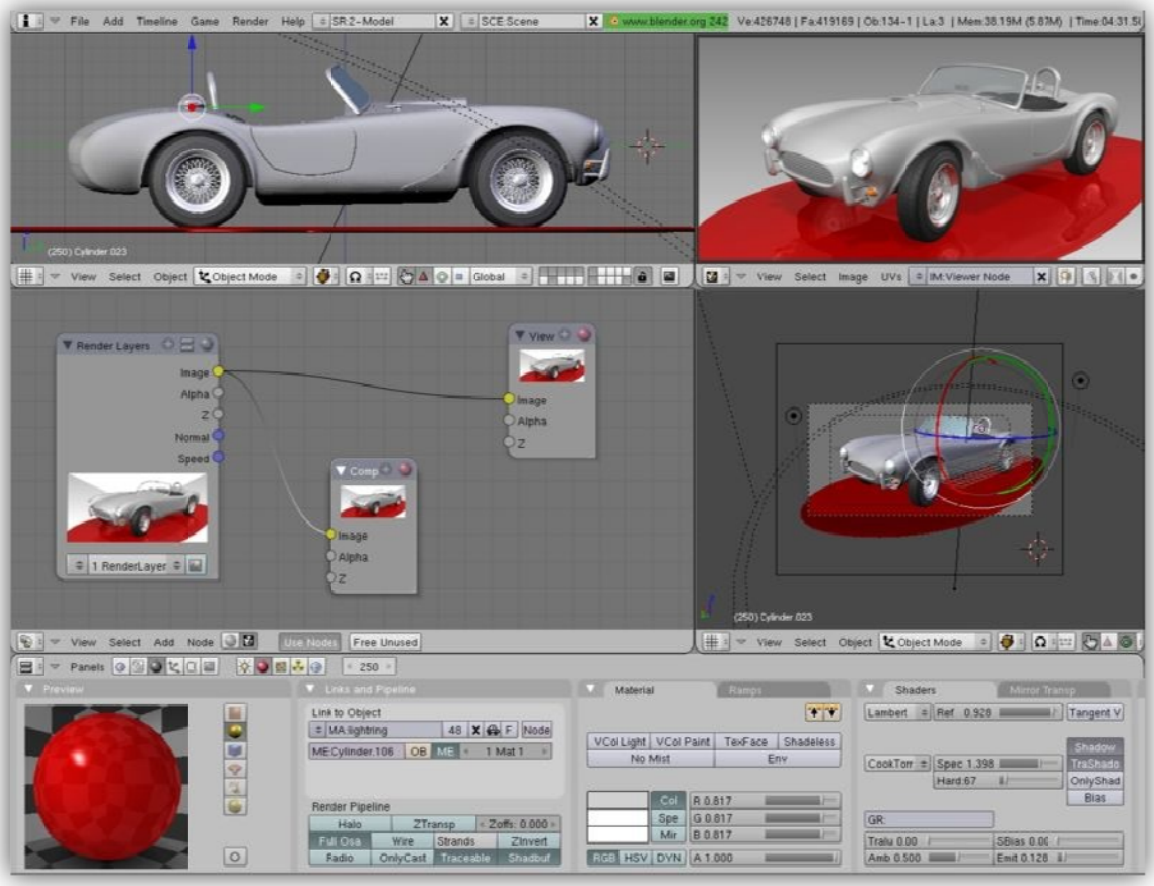


Figure 3-3: Screenshot from Blender 2.42a

4 Open Inventor and VRML

This chapter describes Open Inventor and VRML formats and its animation capabilities. The key characteristics of both formats that were used in the implementation of the animation editor are described more in detail.

4.1 Open Inventor

Open Inventor is a C++ object oriented retained mode 3D graphic API designed by SGI to provide a higher layer of programming for OpenGL. This project uses Coin3D implementation of Open Inventor, released under GNU GPL by the company Systems in Motion. This implementation is fully compatible with original Open Inventor API. Coin library is still thriving under active development, and has added numerous improvements to the original Inventor API like extensive support for the VRML standard. Despite its age, the Open Inventor API is still widely used for a wide range of scientific and engineering visualization systems around the world, having proven itself well designed for effective development of complex 3D application software.

Coin3D implementation of Open Inventor also provides an interconnection with several windows toolkits such Win32, Qt and Xt. This project uses Qt toolkit to create graphical user interface of animation editor. Qt is a cross-platform application development framework, widely used for the development of GUI programs. It is produced by the company Trolltech, formerly Quasar Technologies. Qt uses standard C++, but extends the language by providing an additional pre-processor that generates the C++ code which is necessary to implement Qt's extensions. More information on Qt can be found in **Error! Reference source not found.**. More information on Coin3D can be found in **Error! Reference source not found.**.

The 3D scene in Open Inventor is represented by a tree, which can consist of several types of nodes. The basic node types are nodes that describe object geometry, its characteristics (such as material). Other groups of nodes include spatial transformation nodes (translation, rotation and scale), manipulator nodes, camera, light and others. Next to nodes open inventor defines special groups of object e.g. sensors and engines. Sensors can be attached to other objects and respond to their changes by invoking events. An engine allows encapsulating both motion and geometry into single scene graph. Engine objects include arithmetic (interpolation), animation (time counter), and triggered (counter, on-off) engines. Both basic and advanced techniques of programming in Open Inventor are very well described in (5) and (6).

4.2 VRML

VRML is language to describe virtual words. It is standard for representing three-dimensional interactive vector graphic, designed especially for the use on World Wide Web. VRML is a text file format where vertices and edges for a 3D polygon can be specified along with the surface color, image-mapped textures, shininess, transparency, and so on. URLs can be associated with graphical components so that a web browser might fetch a web-page or a new VRML file from the Internet when the user clicks on the specific graphical component. Animations, sounds, lighting, and other aspects of the virtual world can interact with the user or may be triggered by external events such as timers. A special Script Node allows the addition of program code to a VRML file. VRML files are

commonly called worlds and have the .wrl extension. Most 3D modeling programs including Maya, 3ds Max and Blender can save objects and scenes in VRML format.

Like Open Inventor, VRML is node based, it has tree like structure. It has various node types for object description, world description, sound, fog, background. Another group of nodes are dynamic node. These include manipulators, sensors, interpolators and timers. Formal specification of VRML standard is described in (7).

4.3 Describing scene in Open Inventor

At the programming level, the open inventor three basic sets of tools to create scene:

- **A 3D scene database** – this includes shapes, property, group, engine, and sensor objects. It is used to create hierarchical 3D scene
- **Node kits** – these provide tools to create prebuilt groupings of inventor nodes
- **Manipulators** – are used to manipulate objects in the scene and the user can interact with them directly. There are several types of manipulators including trackball, handle box, etc.

The basic building block of the three dimensional scene databases in Open Inventor is the *node*. Each node holds some specific information about shape, transformation, light, camera, material etc. All scene objects such as shape, light, camera, or attributes are represented by a node.

Collection of nodes is called *scene graph*. Sample scene graph is shown in Figure 4-1. Scene graph is stored in the Open Inventor's database, and Open Inventor takes care about storing and managing the graph.

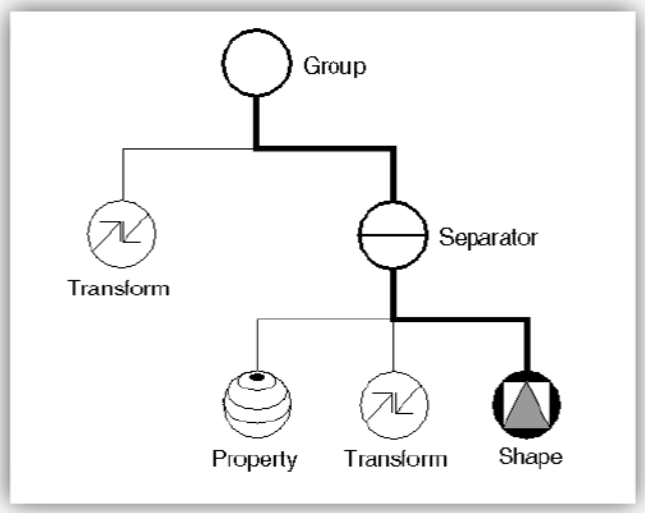


Figure 4-1: Sample scene graph. From (5).

Node kits are used to create structured and consistent databases. Each node kit contains specific nodes that are arranged in specified manner.

Finally, manipulator is a special type of node that enables the user to directly interact with the objects in the scene. Manipulators typically consist of the rendered geometry that provides means for translating events into changes in the database.

Several Open Inventor entities that were important in the process of creating the animation editor are described in detail in the following chapters.

4.3.1 Separators

SoSeparator node is used to isolate effects of nodes in a group. Before the separator node processes its children, it saves it state. After it finishes, it restores that state. Therefore, nodes in the SoSeparator do not affect anything above and to the right in the scene graph.

SoSelection is a special type of separator that enables selection of the nodes in the scene graph below it. Inserting an SoSelection node in the scene graph enables the user to conveniently "pick" with left mouse button to select or deselect objects in the scene.

4.3.2 Paths

Paths are used to locate objects in the scene graph. Paths contain reference to a chain of nodes; each of them is a parent to the next node in the chain, starting with the root node of the search graph (which can be the whole graph scene). Paths are usually used by searching or picking actions.

4.3.3 Camera

A camera node generates a picture of everything after it in the scene graph. Camera must precede all the nodes that are required to be viewed.

All camera nodes in Open Inventor are derived from the abstract class **SoCamera.** Properties of camera node in the Open Inventor:

- ***Position* –** position of the camera in the scene.
- ***Orientation* –** orientation of the camera – it is the rotation of the camera in respect to its default rotation.
- ***Aspect Ratio*** – it is the ratio of the camera viewing width to height. The most common ratios are *Square* (1:1), *Video* (4:3), and HDTV (16:9 – also referred to as widescreen). The cinematic movies have even higher ratio (2.35:1).
- ***Near Distance*** – the distance from the camera viewpoint to the near clipping plane. No geometry is viewed if it is closer than the new clipping plane.
- ***Far Distance*** – the distance from the camera viewpoint to the far clipping plane. No geometry is viewed if it is farther than the far clipping plane.
- ***Focal Distance*** – the distance from the camera viewpoint to the focal point.

Open inventor defines two default camera classes:

- **SoPerspectiveCamera –** produces perspective projection. It tries to emulate the human eye – it is the "natural" projection of an image (as if it is viewed by human observer). Perspective camera introduces two more attributes to the camera **Width Angle** and **Height Angle**.
- **SoOrthographicCamera** – produces parallel projection, with no distortion caused by distance. Orthographic cameras are used for precise design work.

4.3.4 Lights

Scene graph in the Open Inventor uses phong lighting model by default. In order to be able to view the objects, at least one light has to be defined in the scene. The position of the light node in the scene graph determines two things:

- What it illuminates. Since the light is "turned on" after it is processed in the scene graph, the light illuminates only nodes that follows it in the scene. Light is also part of the state that is saved by the SoSeparator node. Therefore, SoSeparator node can be used to isolate light effects within specific sub graph in the scene graph.
- Where the light is located. Some specific types of light define the location of the light, or point of origin of the emitted light, while others define direction in which the light is being emitted. Both location and direction are affected by the current geometric transformation.

All lights in the Open Inventor are derived from the abstract class **SoLight**. It defines three basic attributes of light

- ***On*** – if the light is switched on or off.
- ***Intensity*** – the intensity (brightness) of the light.
- ***Color*** – color of the light.

There are three basic types implemented in the Open Inventor:

- **SoDirectionalLight** – defines only direction in which the light is being emitted. This light is used to simulate the Sun light. The light is illuminated uniformly along specified direction.
- **SoPointLight** – defines only location of the light. The light is radiated equally in all directions from given location in the 3D space.
- **SoSpotLight** – defines both direction of the illuminated light and the location of the light. Spot light is also defined by **Drop off Rate** and **Cut off Angle**. The light is illuminated in the cone from the light's location.

4.3.5 Shapes

The animation editor that was implemented for the purpose of this master thesis does not support any kind of three-dimensional shape creation. The shape can only be imported from separate file (in Inventor or VRML format).

4.4 Animating in Open Inventor

This chapter describes tool provided in Open Inventor that can be used to animate objects in the scene. Among the most import ones are engines and interpolators.

4.4.1 Engines

Engines are special class of objects that enables to encapsulate both motion and geometry into a single scene graph. Objects in the scene can be connected to engines to animate motion or other complex behavior. As it was already mentioned, there are several types of engines. Among the most important ones for animation are the interpolators.

4.4.2 Interpolators

Interpolation nodes are used to linearly interpolate between two values. All interpolation classes in the Open Inventor are derived from **SoInterpolate**. There are two main disadvantages of the interpolation nodes in Open Inventor. Firstly, they can only linearly interpolate between values, and secondly, they can interpolate only between two values. The interpolation engine contains two important attributes:

- **Alpha –** determines the amount of interpolation. For 0 it is the start value and for 1 it is the end value
- **Output** – holds the output of the interpolation engine. Its type depends on the type of interpolator.

Open Inventor defines several types of interpolation engines. These are:

- **SoInterpolateFloat** – interpolation of two float values
- **SoInterpolateRotation** – interpolation of two rotation values
- **SoInterpolateVec2f** – interpolation of two 2d vectors
- **SoInterpolateVec3f** – interpolation of two 3d vectors
- **SoInterpolateVec4f** – interpolation of two 4d vectors

4.4.3 Sensors

Sensors are special type of objects that can be attached to the Open Inventor's database. It can respond to the database changes or to certain timer events by invoking a callback function. In their basic principle, sensors are much like engines, with a few differences. Sensors are part of application rather than a part of scene graph (as it is with engines); they have user defined callback functions, while engines have only a build in functionality; they can be attached to any field, and they can affect objects outside the scene graph. Because of these differences, sensors are very suitable for the use in animation editor for they can "inform" the user interface about the changes in the scene graph. The parent class of all sensors is **SoSensor** class. Basically, there are two types of sensors:

- **Data sensors** – monitor the database and inform the application when it changes. Data sensors can be attached to field (**SoFieldSensor**), node (**SoNodeSensor**), or path (**SoPathSensor**).
- **Timer sensors** – notify the application, when certain types of timer events occur.

4.5 Describing and animation of scene in VRML

The architecture of the scene in VRML is very similar to the Open Inventor's one. However, there are minor differences. The main differences that are significant to the implementation of animation editor are described in following chapters.

4.5.1 VRML compared to Open Inventor

The basic engine class used in VRML to crate animation is **SoVRMLTimerSensor**. The name can be a bit misleading for the node is more an engine, than a sensor. SoVRMLTimerSensor class is a multi-purpose time generator and it generate events over time (when it is switched on – enabled). Properties of SoVRMLTimerSensor:

- **Cycle interval** – duration of each timer cycle.
- **Enable** – enables/disables the timer.
- **Loop** – if true, the sensor will loop when it finishes its cycle.
- **Start time** – defines the time when the sensor is started.
- **Stop time** – defines the time when the sensor is stopped.
- **Cycle time** – this field sends an out event when new cycle is started.
- **Fraction changed** – this field sends an out event, which is sent for each timer tick.

4.5.2 VRML Interpolators

The most significant difference between Open Inventor and VRML, as far as animation editor is concerned, are the interpolators. Unlike Open Inventor's interpolators, interpolators in VRML can interpolate between more than two values. Still, they can only interpolate linearly.

All interpolators in VRML are derived from class **SoVRMLInterpolator**. Important properties of VRML interpolator are as follow:

- **Set Fraction** – is the input of the interpolator. It can be set to any value ranging from 0 to 1.
- **Key** – is a set of keys. Each key is a number between 0 and 1 and it defines the position of the key along the whole animation.
- **Key Values** – is a set o values the interpolator is the value of the key at its time. There are exactly the same number of keys and key values.

5 Designing the animation editor

The animation editor and viewer implemented for the purpose of this master thesis is called **IAnimEd**. The name itself is an abbreviation of words Interactive Animation Editor. Based on the information presented in chapter 2, it has been decided that the animation editor would be key-frame based.

5.1 Functional requirements

The functional requirements on the IAnimEd animation editor and viewer are as follows:

- Importing models/files from both Open Inventor and VRML formats (.iv, .wrl).
- Key frame based animation of translation and rotation of objects.
- Key frame based animation of location and direction of lights.
- Animation of multiple objects and lights in single scene.
- Linear and spline interpolation of the animated values between key frames. Spline interpolation based on Kochanek-Bartels cubic splines.
- Importing and exporting of the animation from/to both Open Inventor and VRML formats (.iv, .wrl).
- Intuitive and easy to use user interface.

5.2 Architecture

The general architecture of the IAnimEd animation editor consists of two parts:

- The User Interface (UI)
- The scene manager (SM)

The UI is implemented using Qt framework, developed by Trolltech(8) and the SM is implemented using the Coin3D framework (9) described in the chapter 4. As the names of the both parts tell, the User Interface manages the user interaction with the editor and the Scene Manager manages the scene (in this case the animation). The Scene Manager keeps track of all objects and lights in the scene, and each object or light keeps track of its own key frames. The more detailed architecture is shown in Figure 5-1.

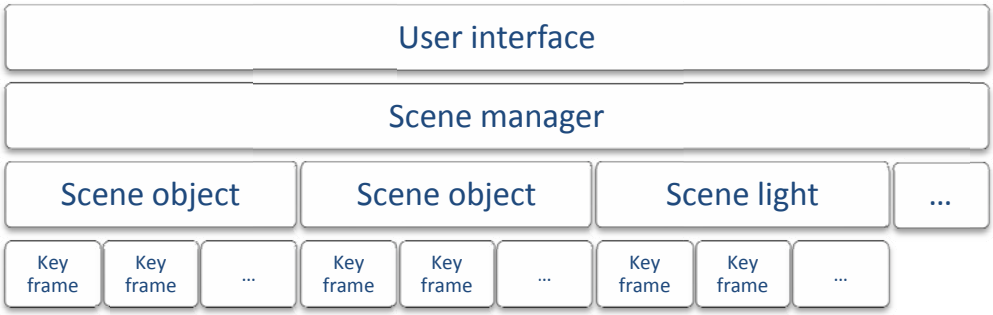


Figure 5-1: The architecture of IAnimEd animation editor

5.3 Scene graph

The scene manager also manages the Open Inventor's scene graph of the editor. The manager has a property called *Editor Root*, which is of type ***SoSeparator***, and it points to the root of the scene graph. Figure 5-2 shows the scene graph of the scene manager. Root node *Editor Root* has a child *Scene Root*. The type of this node is ***SoSelection***, making all nodes below *Scene Root* selectable. This is very important for the convenient management of the objects in the scene for this enables each object or light to be easily picked by a click of a mouse button. The *Scene Root* as well as each *Scene Object* and *Scene Light* have a direct child of type ***SoInfo***. This node holds the attributes for whole scene, object or light. When the scene is saved, the information is passed to the output file and it is used later, when loading the scene. Node *Rendering Style* is used only by editor, and it denotes the whether the object's shape is rendered fully, just its lines, points, or not at all.

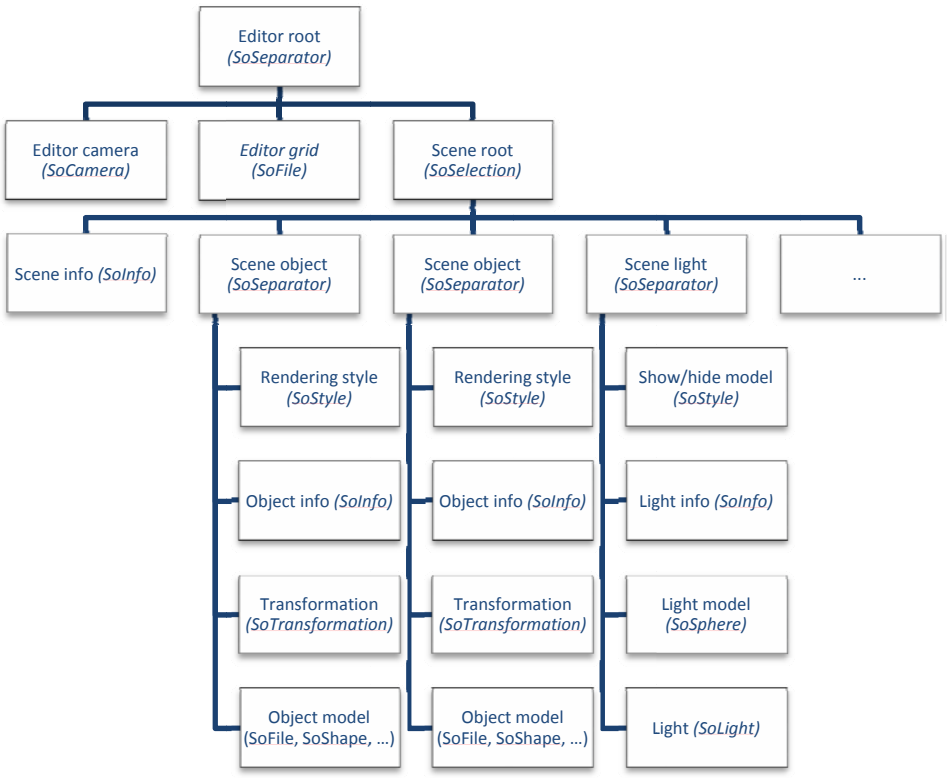


Figure 5-2: Scene graph of the editor's scene. The boxes represent nodes of the graph, with their type shown in the brackets.

5.4 Classes

The IAnimEd animation editor consists of six main classes, which are shown in Figure 5-3. The main application class **IAnimEd** manages the whole applications as well as its user interface. The **SceneManager** class implements the Scene manager, which was mentioned earlier. It contains two lists – list of **SceneObject** classes and list **SceneLight** classes, which represent all objects and all lights in the scene. The **SceneManager** also manages the properties of the animation such as FPS (frames

per second) and duration of the animation. Both **SceneObject** and **SceneLight** classes contain list of **KeyFrame** classes, which represents recorded key frames of each object or light in the scene.

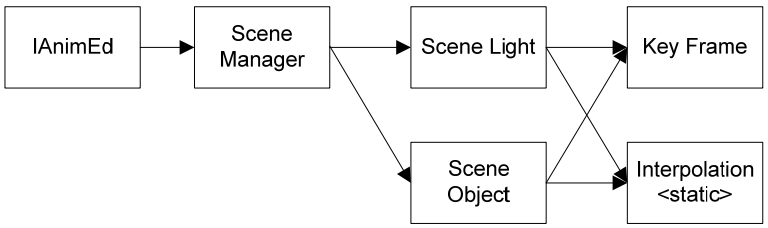


Figure 5-3: Editor's class diagram

Static class **Interpolation** contains method for the linear and spline interpolation of translation or rotation. This class is used only by **SceneObject** and **SceneLight** classes to calculate their position in specific animation frame. More detailed description of each class is presented in chapter 0.

6 Kochanek-Bartels Cubic Splines

The IAnimEd animation editor uses linear and spline interpolation of translation and rotation of objects between key frames. This chapter describes in more detail the implementation of Kochanek-Bartels spline interpolation of 3D vectors. The interpolation of 3D vectors is used in the animation editor to interpolate translation of scene objects, location and/or direction of scene lights. The interpolation of rotation of objects is more complex, the use of quaternions is necessary. Kochanek-Bartels spline interpolation of quaternions is described in chapter 7.

The problem of Kochanek-Bartels cubic splines (also referred to as TCB splines) is described well in detail in (10). The implementation of Kochanek-Bartels splines used in the IAnimEd animation editor uses Hermite interpolation basis described in 2.5. The spline is formed by a sequence of curves $k_i(u),\ 0 \leq i \leq n-2$, and the spline is continuous over all curves and it passes through all control points.

The spline has a continuity of G^1 if both incoming and outgoing tangent vectors at the control point are in the same direction. The spline has a continuity of C^1 if both vectors at the control point are equal (are in the same direction and of equal length).

The Kochanek-Bartels spline interpolation of the IAnimEd animation editor is implemented in class ***Interpolation***. The algorithm for the spline interpolation is shown in Figure 6-1. The function *SplineInterpolate* returns the interpolated 3d vector value at time ***f*** between starting vector ***sp*** and ending vector ***ep***. The algorithm is a simple interpretation of Hermite spline shown earlier in equation (7).

```
SbVec3f SplineInterpolate (SbVec3f sp /* start point */,
      ep, /* end point */
      st, /* outgoing tangent at sp */
      et; /* incomming tangent at ep */
      float f /* fraction of time between sp and ep */ )
{
     float h00 = 2*(f*f*f) – 3*(f*f) + 1;
     float h10 = (f*f*f) – 2*(f*f) + f;
     float h01 = -2*(f*f*f) + 3*(f*f);
     float h11 = (f*f*f) - (f*f);
     return (h00*sp) + (h10*st) + (h01*ep) + (h11*et);
}
```

Figure 6-1: Kochanek-Bartels spline interpolation algorithm pseudo-code

6.1 Tangents

The IAnimEd **Intepolation** class contain methods to calculate spline tangents. Incoming and outgoing tangents evaluated using equations (9) and (10) presented earlier. Figure 6-2 and Figure 6-3 present the algorithms to calculate both tangents of the specified spline segment based on tension, bias and continuity variables.

```
void CalculateStartTangent(SbVec3f sp, /* start point (i) */
      SbVec3f ep, /* end point (i+1) */
      SbVec3f pp, /* previous point (i-1) */
      SbVec3f &s, /* outgoing tangent */
      float t, /* tension */
      float b, /* bias */
      float c  /* continuity */ )
{
s = ( ((1 - t)*(1 + b)*(1 + c)/2)*(sp - pp) )
      + ( ((1 - t)*(1 - b)*(1 - c)/ 2)*(ep - sp) );
}
```

Figure 6-2: Calculation of outgoing tangent algorithm pseudo-code

```
void CalculateEndTangent(SbVec3f sp, /* start point (i) */
      SbVec3f ep, /* end point (i+1) */
      SbVec3f pp, /* previous point (i-1 */
      SbVec3f &d, /* incoming tangent */
      float t, /* tension */
      float b, /* bias */
      float c  /* continuity */ )
{
d = ( ((1 - t)*(1 + b)*(1 - c)/2)*(sp - pp) )
      + ( ((1 - t)*(1 - b)*(1+ c)/ 2)*(ep - sp) );
}
```

Figure 6-3: Calculation of incoming tangent algorithm pseudo-code

The orientation and length of tangent vectors depend on these three variables. The default value of each one of them is zero. In that case the equation for Kochantek-Bartels spline's tangent vector (both incoming and outgoing) would be (after setting tension, continuity and bias variables in equations (9) and (10) to zero) as follows:

$$s(i) = \frac{1}{2}\big(k(i) - k(i-1)\big) + \frac{1}{2}\big(k(i+1) - k(i)\big) \qquad (11)$$

which is, in fact, the equation (8) for Catmull-Rom spline's tangent. Let us have a closer look on each variable and its impact on the shape of the spline.

6.2 Tension

Tension $t \in [-1,1]$ is one of three variables introduced in the equations (9) and (10). The tension variable does not have a dimension, and the default value of the variable is zero. The variable controls the tightness of the curve. For values near 1 the curve is "tightened" at the control point, while values near -1 produce "slack" at control point. The tension variable controls the length of the tangent vector. Shorter vector leads to tightening of the curve and longer vector leads to slackening.

The Figure 6-4 shows two curves with different values of tension at the marked control point. The value of tension of in the left image is -1, while in the right image it is 1.

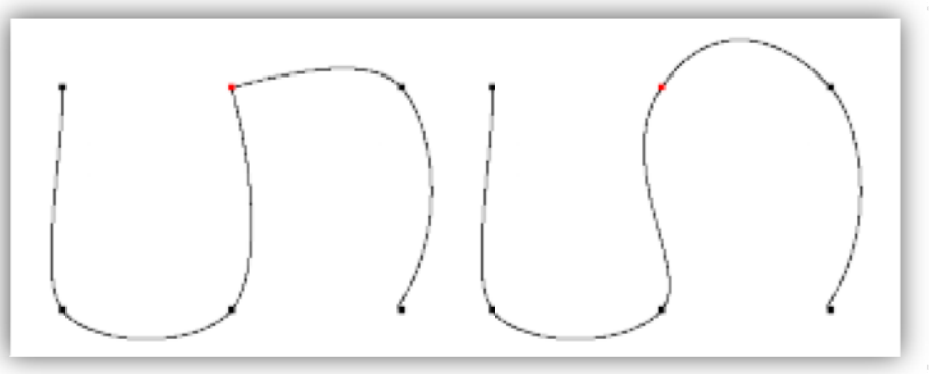


Figure 6-4: Impact of the tension variable on the shape of the curve. Two curves show various values of tension at the red control point. Image adopted from (10).

6.3 Continuity

Continuity $c \in [-1,1]$ is second of three variables introduced in the equations (9) and (10). The non-zero value of this variable forms a "corner" at the control point; the direction of the corner depends on the sign of the continuity variable. The default value of the variable is zero, which leads to continuous tangent vector at the control point. Figure 6-5 show the curves with the continuity value of 1 (on the left) and -1 (on the right).

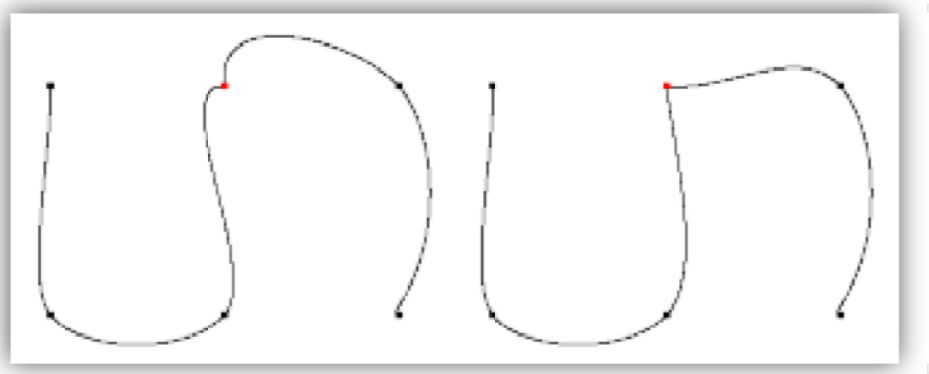


Figure 6-5: Impact of the continuity variable on the shape of the curve. Two curves show various values of continuity at the red control point. Image adopted from (10).

It is obvious that neither of the two curves in the Figure 6-5 have neither C^1 nor G^1 continuity at the red control point.

6.4 Bias

The last of the three control variables is bias $b \in [-1,1]$. When value of bias is equal zero, both incoming and outgoing vectors at the control point are equally weighted producing the Catmull-Rom spline. For values near -1, the outgoing tangent has a greater impact on the direction of the tangent vector at the control point (undershooting). For values near 1, the incoming has a dominating

influence of the direction of the tangent vector (overshooting). Figure 6-6 shows both the curves with both values of 1 (left curve) and -1 (right curve) of the bias variable.

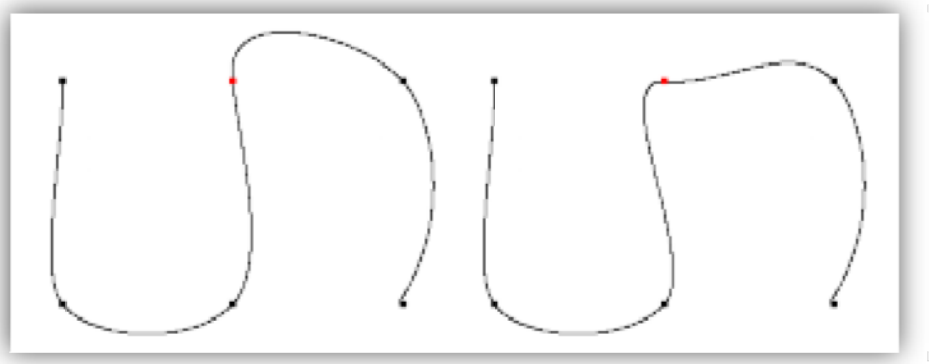


Figure 6-6: Impact of the bias variable on the shape of the curve. Two curves show various values of bias at the red control point. Image adopted from (10).

7 Quaternion

Spline interpolation of the rotation of an object in the scene is rather more difficult than an interpolation of its translation (3D vector). The first problem that has to be solved is how to define rotation of an object in 3D space. After we are able to define two different rotations of an object, the next problem is how to describe smooth (or spline) interpolation between these two values of rotation. And finally, the last problem is how to adapt the Kochanek-Bartels equations for the interpolation of 3D vectors, so they can be used to interpolate rotation. The single answer for all of these questions is quaternion.

7.1 What is a quaternion?

Quaternions are often used in computer graphics to represent rotation. Quaternions were first discovered by Irish mathematician Sir William Rowan Hamilton in 1983. Quaternions form four-dimensional normed division algebra over real numbers. More detailed information about quaternions and quaternion algebra and calculus can be found in (11) and (12). Quaternions can be conveniently used to mathematically describe the orientations and rotations of objects in 3D scene.

A quaternion is defined by the following equation:

$$q = w + xi + yj + zk \qquad (12)$$

where **w**, **x**, **y**, and **z** are real numbers. Rotation in 3D space can be defined by quaternion in following form:

$$q = \left(cos\frac{\alpha}{2}, v.\sin\frac{\alpha}{2}\right) \qquad (13)$$

where **v** specifies the rotation axis and α is the angle by which to rotate around this axis. Quaternion $\boldsymbol{q}$ is an unit quaternion.

Open inventor provides convenient way to specify quaternions through **SbRotation** class. This class is used inventor to represent rotation in 3D space. Internally, the **SbRoration** class store the rotation as a quaternion and it contain methods to set or retrieve the value of rotation directly via quaternion.

7.2 Quaternion algebra

In order to be able to successfully use the quaternion to interpolate rotation, it is necessary to provide some functions and algebraic relations between quaternions. Equations are based on (12), and (13).

Addition and *subtraction* of two quaternions is defined by:

$$q_0 \pm q_1 = (w_o \pm w_1) + (x_o \pm x_1)i + (y_o \pm y_1)j + (z_o \pm z_1)k \qquad (14)$$

Addition and subtraction is implemented in class **SbRoration** in open inventor, so it is not necessary to discuss it any more in detail.

For the *multiplication* of two quaternions, we can divide the quaternion into two parts - the *scalar* part denoted by s, where $s = w$, and the *vector* part denoted by v, where v is a vector and holds that $v = (x, y, z)$. Then, the multiplication of two quaternions $q_0 = [s_0, v_0]$ and $q_1 = [s_1, v_1]$ is given by following equation:

$$q = q_0 q_1 = [s_0 s_1 - v_0 \cdot v_1, v_0 \times v_1 + s_0 v_1 + s_1 v_0] \quad (15)$$

where $\cdot$ denotes scalar product and $\times$ denotes vector product of vectors in R^3. The multiplication of quaternion is implemented in the **Interpolation** class of IAnimEd.

Following definitions of *conjugate, norm, inverse, exponential, and logarithm,* are using the same notation as it was defined earlier, and they are all implemented in the **Interpolation** class of IAnimEd. If not stated otherwise the quaternion is defined by $q = [s, v]$.

The *conjugate* of a quaternion is defined by equation:

$$q^* = [s, -v] \quad (16)$$

Norm function of quaternion is defined by:

$$N(q) = s^2 + v \cdot v \quad (17)$$

Quaternion's *inverse* is defined by:

$$q^{-1} = \frac{q^*}{N(q)} \quad (18)$$

The *logarithm* of a unit quaternion $\boldsymbol{q} = [\mathbf{cos}(\boldsymbol{\alpha}), \boldsymbol{v}.\mathbf{sin}(\boldsymbol{\alpha})]$ is defined by:

$$\log(q) = [0, v.\alpha] \quad (19)$$

Let a quaternion be of a form $q = [0, \alpha.v]$. Then, its *exponent* is defined by:

$$q = [\cos(\alpha), v.\sin(\alpha)] \quad (20)$$

7.3 Spherical Linear Interpolation

Spherical Linear Interpolation or *Slerp* was introduced in concept of quaternion interpolation for the purpose of animating 3D rotation. The function of *Slerp* is best illustrated by Figure 7-1. Slerp interpolates between two rotations given by quaternions q_0 and q_1. The interpolation is given by:

$$Slerp(q_0, q_1, t) = \frac{q_0\big((1-t)\alpha\big) + q_1 \sin(t\alpha)}{\sin(\alpha)} \quad (21)$$

where $0 \leq t \leq 1$ is the time value. As t varies from 0 to 1, the values $q(t)$ varies uniformly along the circular arc (as it is shown in Figure 7-1).

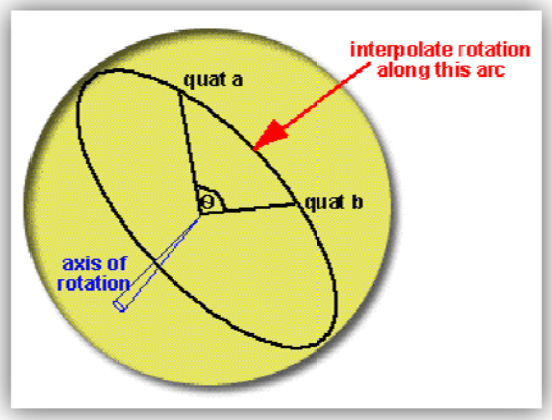


Figure 7-1: Spherical Linear Interpolation between quaternion a and quaternion b. Source: (14).

If quaternions q_0 and q_1 are both unit quaternions then it is possible to rewrite equation (21) to the following format:

$$Slerp(q_0, q_1, t) = q_0({q_0}^{-1}q_1)^t \tag{22}$$

And finally, for $t = 0$ and $t = 1$:

$$q_0 = Slerp(q_0, q_1, 0) \tag{23}$$

$$q_1 = Slerp(q_0, q_1, 1) \tag{24}$$

The Open Inventor's class **SbRotation** contains the implementation of *Slerp* function and thus it is not necessary to describe the background around *Slerp* in any further detail. More information can be found in (14) and (15).

7.4 Spherical Cubic Interpolation using Kochanek-Bartels Splines

Spherical Cubic Interpolation or *Squad* is used to interpolate quaternions using cubic interpolation. As it was stated in (12), the *Squad* can be achieved by use of bilinear interpolation on a quadrilateral. The evaluation uses application of tree *Slerp* functions, and it is denoted by:

$$Squad(q_0, s_0, s_1, q_1, t) = Slerp\big(Slerp(q_0, q_1, t), Slerp(s_0, s_1, t), 2t(1-t)\big) \tag{25}$$

where $\boldsymbol{q_0}, s_0, s_1, \boldsymbol{q_1}$ are ordered vertices of a quadrilateral. The resulting quaternion is interpolated between q_0, q_1, and the s_0, s_1 are so called intermediate terms.

According to (16), *Squad* construction for quaternions can be modified so it supports Kochanek-Bartels spline ideas. Equations (9) and (10) can be easily extended to quaternions, and for s_0 and s_1 they are defined as:

$$T_0 = \frac{(1-t).(1+b).(1+c)}{2} log\ ({q_{-1}}^{-1}q_0) + \frac{(1-t).(1-b).(1-c)}{2} log\ ({q_0}^{-1}q_1) \tag{26}$$

$$T_1 = \frac{(1-t).(1+b).(1-c)}{2} log\ ({q_0}^{-1}q_1) + \frac{(1-t).(1-b).(1+c)}{2} log\ ({q_1}^{-1}q_2) \quad (27)$$

$$s_0 = q_0\, exp\left(\frac{T_0 - log({q_0}^{-1}q_1)}{2}\right) \quad (28)$$

$$s_1 = q_0\, exp\left(\frac{log({q_{-1}}^{-1}q_0) - T_1}{2}\right) \quad (29)$$

Squad function is implemented in **Interpolation** class of IAnimEd. Pseudo-codes of *Squad* algorithms are presented in Figure 7-2, Figure 7-3 and Figure 7-4.

```
SbRotation SplineInterpolate(
      SbRotation sp, /* quaternion q0 */
      SbRotation ep, /* quaternion q1 */
      SbRotation st, /* intermediate term s0 */
      SbRotation et, /* intermediate term s1 */
      float f)
{
      return slerp(slerp(sp, ep, f),slerp(st, et, f), 2*f*(1-f) );
}
```

Figure 7-2: Squad algorithm pseudo-code

```
void CalculateStartTangent(
      SbRotation sp, /* q0 */
      SbRotation ep, /* q1 */
      SbRotation pp, /* q-1 */
      SbRotation &s, /* s0 */
      float t, /* tension */
      float b, /* bias */
      float c) /* continuity */
{
      //get quaternions
      SbVec4f sq, eq, pq, t0, an;
      //set quaternions
      sp.getValue(sq[0], sq[1], sq[2], sq[3]);
      ep.getValue(eq[0], eq[1], eq[2], eq[3]);
      pp.getValue(pq[0], pq[1], pq[2], pq[3]);
      //create tangent
      t0 = ( ((1-t)*(1-c)*(1-b)/2) * Log(Mul(Inv(sq),eq)) )
            + ( ((1-t)*(1+c)*(1+b)/2) * Log(Mul(Inv(pq),sq)) );
      //create intermediate term
      an = Mul(sq, Exp( (t0 - Log(Mul(Inv(sq),eq))) / 2 ));
      //create rotation
      s.setValue(an[0], an[1], an[2], an[3]);
}
```

Figure 7-3: Pseudo-code of algorithm to calculate intermediate term s_0.

```
void CalculateEndTangent(
      SbRotation sp, /* q0 */
      SbRotation ep, /* q1 */
      SbRotation pp, /* q-1 */
      SbRotation &s, /* s0 */
      float t, /* tension */
      float b, /* bias */
      float c) /* continuity */
{
      //get quaternions
      SbVec4f sq, eq, pq, t1, bn;
      //set quaternions
      sp.getValue(sq[0], sq[1], sq[2], sq[3]);
      ep.getValue(eq[0], eq[1], eq[2], eq[3]);
      pp.getValue(pq[0], pq[1], pq[2], pq[3]);
      //create tangent
      t1 = ( ((1-t)*(1+c)*(1-b)/2) * Log(Mul(Inv(sq),eq)) )
            + ( ((1-t)*(1-c)*(1+b)/2) * Log(Mul(Inv(pq),sq)) );
      //create intermediate point
      bn = Mul(sq, Exp( (Log(Mul(Inv(pq),sq)) - t1) / 2 ));
      //create rotation
      d.setValue(bn[0], bn[1], bn[2], bn[3]);
}
```

Figure 7-4: Pseudo-code of algorithm to calculate intermediate term s_1.

8 Implementing the Editor

This chapter contains detailed description of the implementation of animation editor IAnimEd. It discusses the basic structure of the application as well as the detailed description of each key part of the application.

8.1 Basic Layout

As it was already stated earlier in chapter 4, the animation editor consists of two main parts – the user interface and the scene manager.

The user interface is implemented in class **IAnimEd** and the scene manager is implemented in class **SceneManager**. Scene manager manages all lights and objects in the scene. The scene object is implemented in class **SceneObject** and the light is implemented in class **SceneLight**. Each object and light in the editor can be animated. The animation itself, as it was mentioned earlier, is implemented using key-framing animation technique. The objects and lights in the scene manage their own key-frames. Single key-frame is represented by an object, which is implemented in class **KeyFrame**. Finally various key values of translation or rotation of lights and objects throughout the time line needs to be interpolated to create an animation. As noted earlier, all necessary methods, used to interpolate position, location, direction or rotation, are implemented in class **Interpolation**.

Following chapters contain a detailed description of each class in the implementation of IAnimEd editor.

8.2 IAnimEd

The **IAnimEd** class, as it names may denote, is the main class of the whole application. This class manages mainly the user interface of the application. It enables to import objects from file to the scene, import and export whole scene to supported file formats, as well as the to animate objects and lights.

User interface of the application is created by Qt toolkit. The main window is made of several dock widgets that can be moved around freely or can be hidden. Dock widgets are implemented via the **QDockWidget** class. Main window contain these dock widgets:

- ***dproperties*** – is a dock widget to provide UI to set animation attributes (duration and frames per second of the animation)
- ***dseeker*** – is used to move to specific frame of the animation
- ***dobjects*** – contain list of all objects in the scene. Objects can be modified or deleted.
- ***dlights*** - contain list of all lights in the scene. Lighs can be modified, deletedor switched off.
- ***dkeyframes*** – manages key frames of currently selected object or light.

Figure 8-1 shows the collaboration diagram of **IAnimEd** class. It shows that **IAnimEd** class collaborates only with classes **SceneManager** and **FrameView**.

IAnimEd class collaborates with **SceneManager** class through the ***manager*** property. Upon the creation of **IAnimEd** class, the scene manager is created and this property is set to point to the

instance of **SceneManager** class. Every user input, such as selecting and deselecting of objects, change in objects or light attributes is passed down to the ***manager***.

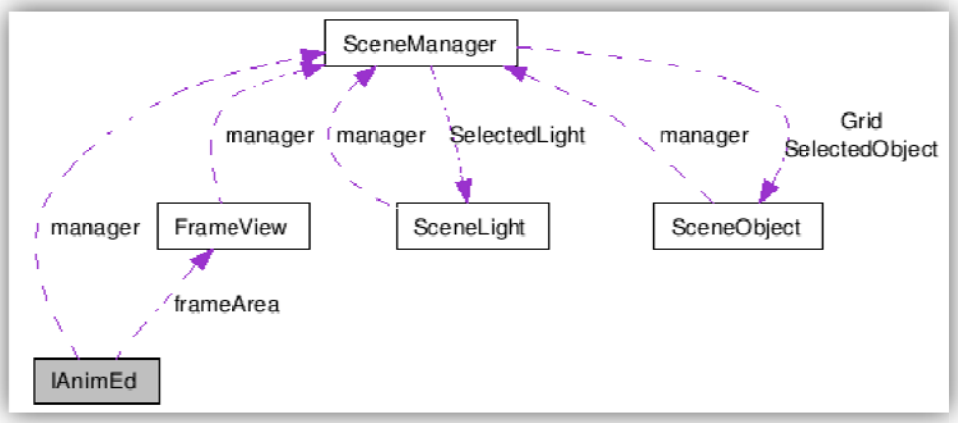


Figure 8-1: Collaboration diagram of IAnimEd class.

Static methods ***select*** and ***deselect*** of **IAnimEd** class are used to manage selection in the manager's scene tree. They are used as callback methods by ***manager*** to select lights and objects.

Property ***frameArea*** points to the instance of **FrameView** class, which implements a user interface component. It is inherited form **QWidget** class and it overrides the parrent's ***paintEvent*** method. In this method, the widget draws itself as an information panel that shows key frames of currently selected object or light and also it shows current frame of the animation that is being edited. The example of **FrameView** component is shown in Figure 8-2.

Figure 8-2: FrameView component example. The slider and frame counter is not part of the FrameView component.

IAnimEd class also implements the main menu trough **QMenu** class and set of toolbars trough **QToolbar** class. Menu and toolbars contain set of actions implemented by **QAction** class, which correspond to every action that can be performed by the user interface such as importing model to scene, loading and saving, creating lights, etc.

8.3 Scene Manager

The **SceneManager** class is the hearth of the while animation editor as its manager. It contains methods to manage objects and lights in the scene, for recording and playing the animation, importing and exporting of the scene, etc.

The property ***EditorRoot*** of type ***SoSeparator*** is the root of the whole scene graph of the editor. Its children are editor's camera ***EditorCamera***, root of the model of the ***Grid*** and ***SceneRoot***. The ***SceneRoot*** is of type **SoSelection**, making all of its children selectable. It is the root of the scene graph representing the edited scene. The two methods ***select*** and ***deselect*** that were mentioned

earlier are set as the selection callback functions of the ***SceneRoot***. All objects in the scene that can be animated are children of the ***SceneRoot***.

As it is obvious from the Figure 8-3, the **SceneManager** communicate only with the classes **SceneLight** and **SceneObject**. Manager contain to lists – the list of all objects in the scene and the list of all lights in the scene. The ***SceneList*** property is of type **SbList<SceneObject *>**. It contains objects of **SceneObject** type, so it holds all objects in the scene. The ***LightList*** on the other hand, is of type **SbList<SceneLight *>** and it contains all lights in the scene. Properties ***SelectedLight*** and ***SelectedObject*** contain pointer to currently selected object or light. On one of them can be set at the same time. If nothing is selected, both properties are set to *NULL*.

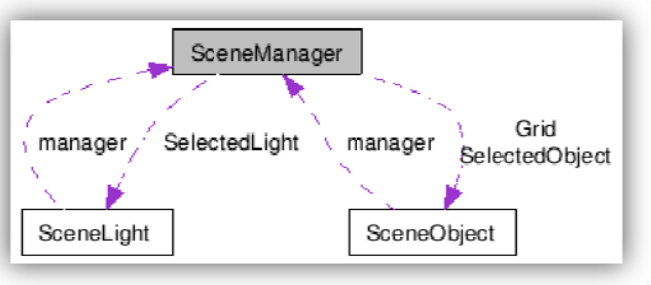


Figure 8-3: Collaboration diagram of SceneManager class.

Properties ***AnimFPS***, ***animDuration*** and ***currentFrame*** holds the information about the animation fps, duration of the animation and currently selected frame in the editor. ***AnimEngine*** of type **SoOneShot** is timer (or engine) used for the animation playback.

The ***SelectedPath*** property, which is of type **SoVRMLIndexedLineSet**, points to a line that shows the animation path of currently selected object.

8.4 Scene Object

SceneObject class represents a single object in the editor's scene. It contains method to manage position and rotation of the object, to set various attributes, and to manage key frames of the object.

The ***manager*** property of **SceneObject** class points to the scene manager that manages this object. It is used mainly for the internal purposes. Class also collaborates only with the class **SceneManager** and it is through the ***manager*** property.

Each scene object has its ***root*** of type **SoSeparator** that points to the root of the object's scene graph. This sub-graph of contains the property ***transform*** that holds current transformation properties of the object in the scene. Property ***lastTransform*** is used when recording the animation to determine of the object's translation or rotation has been changed. If so, key frame is recorded. The ***root*** also contains the nodes that determine the shape of the scene objects.

Methods ***setManip*** and ***unsetManip*** are used to replace the ***transform*** node in the scene graph of the object with the manipulator when selected or vice versa if deselected. Manipulator is of type **SoTransformerManip** and it provides a user interface tool through with the user can easily manipulate with the object. The example of such manipulator in editor is shown in Figure 8-4.

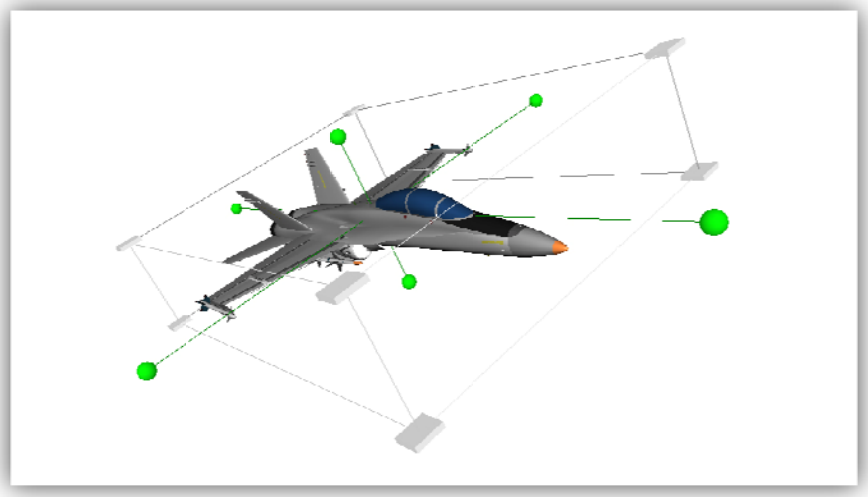


Figure 8-4: SoTransformerManip manipulator attached to an object in the scene.

The **SceneObject** also contains the list ***keyFrameList*** of type **SbList<KeyFrame *>**. This list contains all key-frames that have been recorded for this object.

Method ***updateObject*** is used to update position and rotation of the object in the *currentFrame* set in object's ***manager***. It uses method **Interpolate** to interpolate the object's position and rotation at *currentFrame*.

Finally, the **SceneObject** contains two interpolator properties - the ***orientationInterpolator***, which is of type **SoVRMLOrientationInterpolator** and the ***coordinateInterpolator***, which is of type **SoVRMLCoordinateInterpolator**. These interpolators are used by for animation playback to create the animation of the object according to its key-frames. Prior the playback, the animation has to be set up by method ***setAnimation***. This method sets both interpolators and creates a key value in each interpolator for each frame in the animation. Since the Open Inventor's interpolators can interpolate only linearly, the key values has to be set for each frame in order to achieve the look of smooth animation.

8.5 Scene Light

SceneLight at its basics is completely same as the **SceneObject** class, but it manages lights instead. Apart from each property that has been mention for the **SceneObject** class, the **SceneLight** contains the property **light**, which can be of one of following types:

- **SoDirectionalLight**
- **SoPointLight**
- **SoSpotLight**

Property ***ltype*** then denotes the type of the light. Once created the type of the light cannot be changed. The **SceneLight** also contain the ***model*** property of type **SoSphere**. Since the light is invisible, a simple sphere is used to show the light in the editor. This sphere model is not exported, so it is not visible in the resulting animation file.

Rest of the **SceneLight** class is the same as **SceneObject** class.

8.6 Key-frame

KeyFrame class implements the single key frame structure that holds information about the frame and time of the key, recorded position and rotation, and the interpolation parameters.

Frame property holds the frame number, in which the key frame was recorded. ***Transform*** property of type **SoTransform** then denotes the position and rotation value of the key-frame. Properties ***bias***, ***tension*** and **continuity** hold the float attributes of key-frame for bias, tension and continuity, respectively. These attributes are used in Kochanek-Bartels spline interpolation.

The interpolation type of **KeyFrame** is determined by ***type*** property. It can hold values *Linear* and *Spline* defined in the enumeration type **InterpolationType**.

8.7 Interpolation

The **Interpoation** class is a static class that is used for interpolation purposes. It contain methods for linear and spline interpolation and also some quaternion functions. Pseudo-codes of interpolation algorithms were presented earlier in this thesis.

The interpolation methods implemented in this class are ***LinearInterpolate***, ***SplineInterpolate***, ***CalculateStartTangent*** and ***CalculateEndTangent***. Each function is overloaded with two kinds of parameters - one for interpolation of 3D vectors of type **SbVec3f** and one for interpolation of rotation of type **SbRotation**.

The quaternion functions implemented in this static class are ***Mul*** for multiplication of quaternions, ***Norm*** for quaternion norm, ***Conjugate*** for the conjugate of a quaternion, ***Inv*** for quaternion's inverse, ***Log*** for quaternion's logarithm and ***Exp*** for exponential of quaternion.

8.8 Importing and exporting

The importing and exporting functions are handled in the **SceneManager** class.

Single object can be imported to scene from file in either open inventor (.iv) or VRML (.wrl) format. This is achieved via method ***ImportModelFromFile***. The model is imported "as is", which means that the whole scene graph saved in the file is imported with all shapes, lights or animation. The model is represented in the scene as a whole and separate parts of the object cannot be animated.

The whole scene can be also exported to open inventor or VRML file formats. In order to be able to later import the scene from file, **SoInfo** nodes are inserted into the scene graph. First info node is inserted right after the scene root node. It identifies the file as a file created by IAnimEd animation editor and stores animation properties of a scene. Also every scene object or scene light contains the info node that contains information about the object's key frames.

9 Using the Animation Editor

This chapter describes, in detail, the basic usage of the IAnimEd animation editor.

9.1 Using the interface

The basic user interface of the application is shown in Figure 9-1. In the default layout, there is a set of dock widgets on the right, which are used to set various attributes or to manage lights or objects in the scene. Various actions can be also performed by actions in the main menu or in the tool bars.

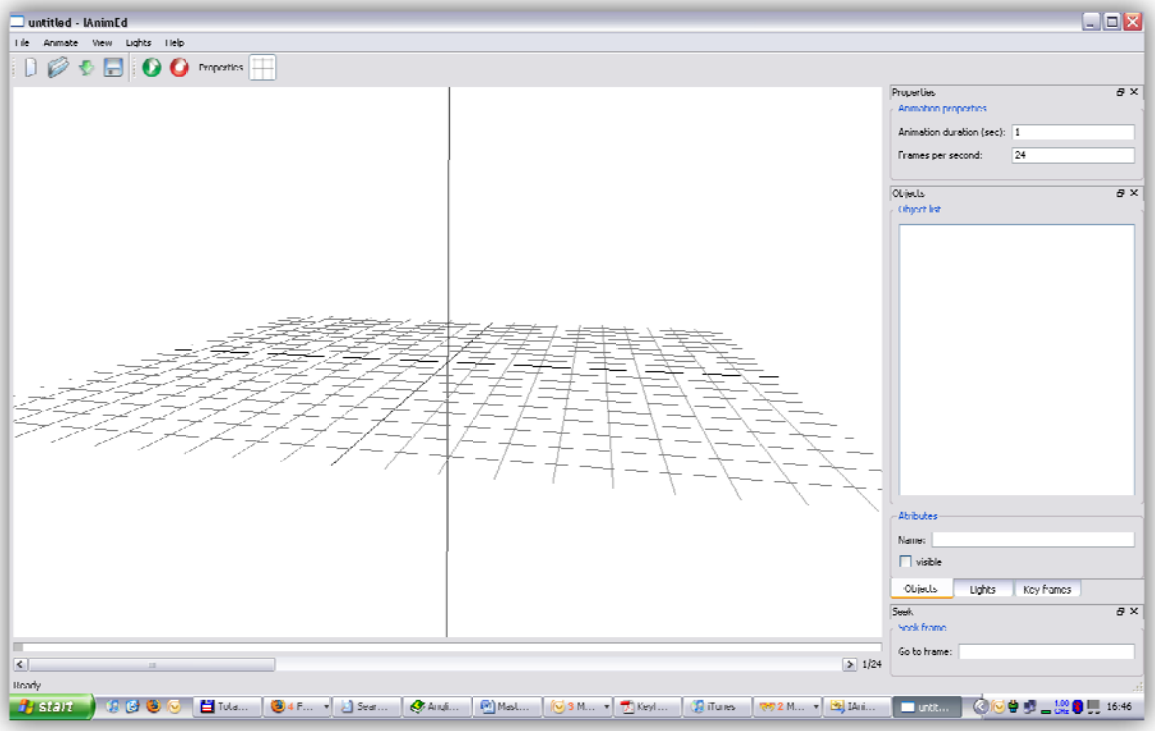


Figure 9-1: User interface of the IAnimEd animation editor.

9.2 Recording animation

The **SceneManager** class and thus also the animation editor can be set to the record mode. If the editor is in the record mode, all changes to the object's position or rotation are recorded as a key-frame at current frame. If the object is selected, its key frames are shown in the **FrameView** component area.

Objects and lights can be manipulated via manipulators. After object or light is selected, its corresponding manipulator is show and can be used to manipulate the object. Typical editor's scene in the record mode is shown in Figure 9-2.

When selected object has recorded two or more key-frames, the object's path along the time line is shown.

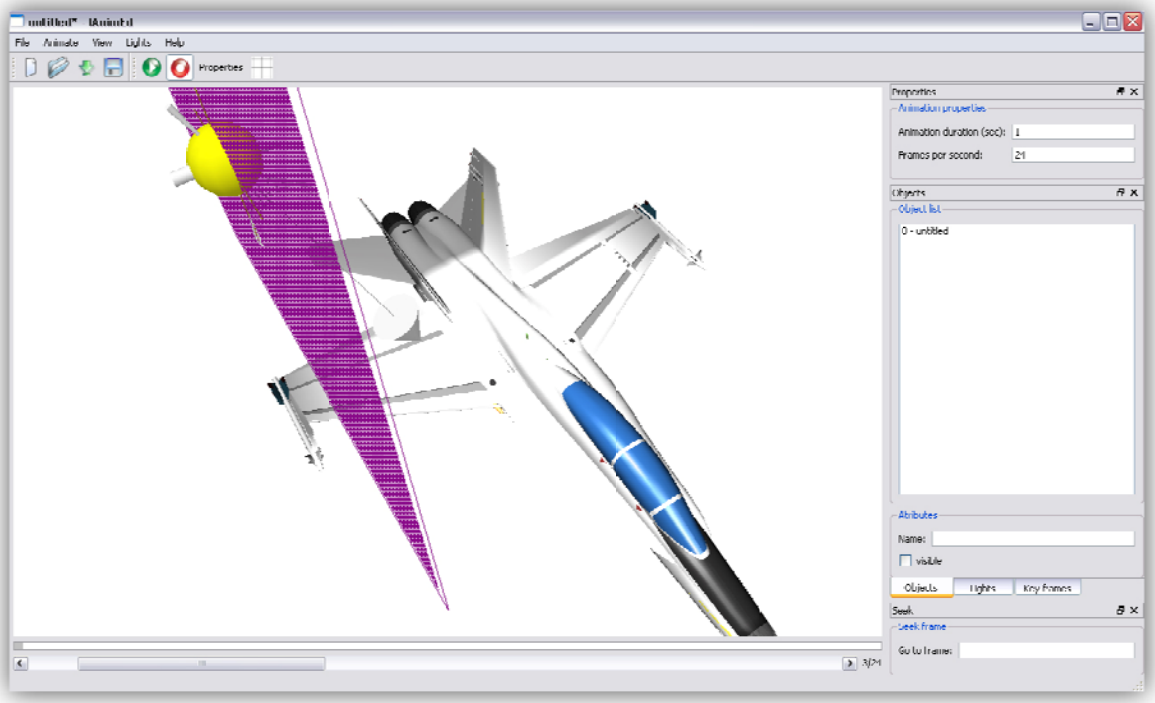


Figure 9-2: Animation editor in the record mode, with spot light manipulator.

9.3 Playing animation

The animation can be played by using the play action in the menu or in the toolbar. When animation is started, all objects are deselected, and the recording mode is turned off. **SceneManager** will then set up the animation by calling method ***setAnimation*** of each of its objects and lights in ***SceneList*** and ***LightList***. Interpolators are connected to the ***AnimEngine***, which is set to duration of animation. Afterwards, the engine is triggered and the animation is played.

9.4 Advanced usage

To take the full advantage of the IAnimEd animation editor user interface, the extensive use of keyboard shortcuts is essential. The full description of shortcuts can be found in Appendix B.

10 Conclusion

This master thesis discussed the problem of computer animation. It contains description of the most common approaches used in computer animation today, basic principles of animation, and mathematical background necessary for the computer animation.

This paper presents the description of the widely spread computer animation technique – animation based on key-frames. This technique is based on traditional animation. Its main idea, is that animator defines several key frames of animation and the computer interpolates all other frames.

The main contribution of this master thesis is the implementation of key-frame based animation editor called IAnimEd. This editor enables to animate position and rotation of multiple objects, and location and direction of multiple lights in single scene. Values of these variables are interpolated between each key-frame in order to create smooth animation.

Editor uses both linear and spline interpolation of values between key-frames. Spline interpolation is implemented using Kochanek-Bartels spline. This approach allows the user to define three variables – tension, bias and continuity – at each control point in order to even further control the object's animation. Ideas introduced in Kochanek-Bartels spline are also used to interpolate rotation. Editor uses the quaternion to define rotation and the rotation interpolation is implemented using Slepr (Spherical Linear Interpolation) and Squad (Spherical Cubic Interpolation) functions to interpolate between to quaternions.

The resulting editor allows is easy to use, it allows its user to import objects from two different file formats (Open Inventor and VRML) and animate their position and rotation in the scene. User can also add and animate multiple lights of three different kinds. Final animation can then exported to VRML or Open Inventor file format and can be played using different player. Exported scene can be imported back later, preserving its previous settings.

10.1 Master thesis goals accomplishment

1. The study of the various animation principles and methods is essential in order to fully apprehend the problem of computer animation and animation editing. The acquired knowledge of these methods has been later used while implementing the animation editor. The animation capabilities of VRML and Open Inventor are described in chapter 4.
2. Simple animation editor was implemented for the purpose of the term project, which preceded this master thesis. Implemented animation editor set the foundation for the IAnimEd animation editor implemented for the purpose of this master thesis. The animation editor is based on Open Inventor and VRML. The design of the animation editor is presented in chapter 5.
3. Proposed animation editor has been implemented using open source toolkits Coin3d and Qt. Both toolkits are platform independent. However, the application has been primarily developed for Windows platform; other platforms have not been tested.

4. Experience acquired during the implementation of animation editor for the term project proved useful. Following advanced animation techniques has been proposed and implemented:
    a. Animation of multiple objects.
    b. Interpolation between multiple key-frames (more than two).
    c. Spline interpolation of position and rotation of objects using Kochanek-Bartels splines.
    d. Animation of multiple lights.
5. The IAnimEd animation editor has been released on the internet with GPL license.

10.2 Future work

Implemented animation editor allows animation of only few objects attributes. Only position and rotation of an object can be animated. In order to be able to use the editor more sophisticatedly, it is necessary that more or all possible attributes such as scale, visibility, deformation etc. can be animated.

The possibility to animate camera is also interesting and useful way to improve the animation editor.

Introduction of advanced animation techniques such as forward and inverse kinematics is another area that can be pursued in the future.

As it can be seen in the commercial animation software, there are plenty of areas, in which the animation editor could be improved. Possibilities for future work on this project are there fore almost endless.

11 Bibliography

1. *Principles of traditional animation applied to 3D computer animation.* **Lasseter, J.** 4, Pixar, San Rafael, CA : s.n., July 1987, ACM SIGGRAPH Computer Graphics, Vol. 21. Available on URL: http://portal.acm.org/citation.cfm?doid=37401.37407 [Cited: December 5, 2006.].

2. *Computer Animation.* **Hodgins, J., O'Brien, J., Bodenheimer, R.** [ed.] John G. Webster. 1999, Wiley Encyclopedia of Electrical and Electronics Engineering, Vol. 3, pp. 686-690. Avialable on URL: http://www.cc.gatech.edu/gvu/animation/papers/ency.pdf [Cited: January 3, 2007.].

3. **Thalmann, N., Thalmann, D.** Computer Animation in Future Technologies. [Online] 1998. [Cited: December 5, 2006.] http://vrlab.epfl.ch/Publications/pdf/Magnenat_Thalmann_Thalmann_InteractiveCA_96.pdf.

4. 3D volume morphing. [Online] [Cited: December 5, 2006.] http://graphics.stanford.edu/~tolis/toli/research/morph.html.

5. **Wernecke, J.** *The Inventor Mentor:Programming Object Oriented 3D Graphics with Open Inventor.* Release 2. 1994.

6. *The Inventor Toolmaker, Extending Open Inventor.* Release 2. 1994.

7. ISO/IEC 14772-1:1997, the Virtual Reality Modeling Language (VRML) specification. [Online] [Cited: December 5, 2006.] http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97/.

8. Qt Toolkit. *Trolltech web site.* [Online] 2007. [Cited: May 10, 2007.] http://www.trolltech.com/products/qt.

9. Coin3D. *Coin3D web site.* [Online] 2007. [Cited: May 10, 2007.] http://www.coin3d.org/.

10. **Eberly, David.** *Kochanek-Bartels Cubic Splines (TCB Splines).* s.l. : Geometric Tools, Inc., 2003.

11. Quaternion. *Wikipedia, the free encyclopedia.* [Online] 05 2, 2007. [Cited: May 10, 2007.] http://en.wikipedia.org/wiki/Quaternion.

12. **Eberly, David.** *Quaternion Algebra and Calculus.* s.l. : Geometric Tools, Inc., 2002. Available on URL: http://www.geometrictools.com/Documentation/Quaternions.pdf [Cited: May 10, 2007.].

13. **Gourdeau, Richard.** Documentation. *ROBOOP, A Robotic Object Oriented Package.* [Online] Département de génie électrique, École Polytechnique de Montréal, 2007. [Cited: May 10, 2007.] http://www.cours.polymtl.ca/roboop/docs/robot.html.

14. **Baker, Martin John.** Maths - Quaternion Interpolation (SLERP). *EuclideanSpace .* [Online] 2007. [Cited: May 21, 2007.]

15. Slerp. *Wikipedia, the free encyclopedia.* [Online] 2007. [Cited: May 21, 2007.] http://en.wikipedia.org/wiki/Slerp.

16. **Eberly, David.** *Key Frame Interpolation via Splines and Quaternions.* s.l. : Geometric Tools, Inc., 2002. Available on URL: http://www.geometrictools.com/Documentation/KeyframeAnimation.pdf [Cited: May 10, 2007.].

17. **Palguta, M.** *Modelovanie 3D scény a jej vizualizácia.* Brno University of Technology. Brno : s.n., 2005. Bachelor Thesis.

18. **Langridge, J.** Smooth interpolation of irregularly spaced keyframes. *http://www.gamedev.net/reference/articles/article1497.asp.* [Online] [Cited: December 5, 2006.]

19. **Pečiva, J.** Open Inventor: Knihovna pro realtimovou 3D grafiku. *Root.cz.* [Online] [Cited: December 5, 2006.] http://www.root.cz/clanky/open-inventor/.

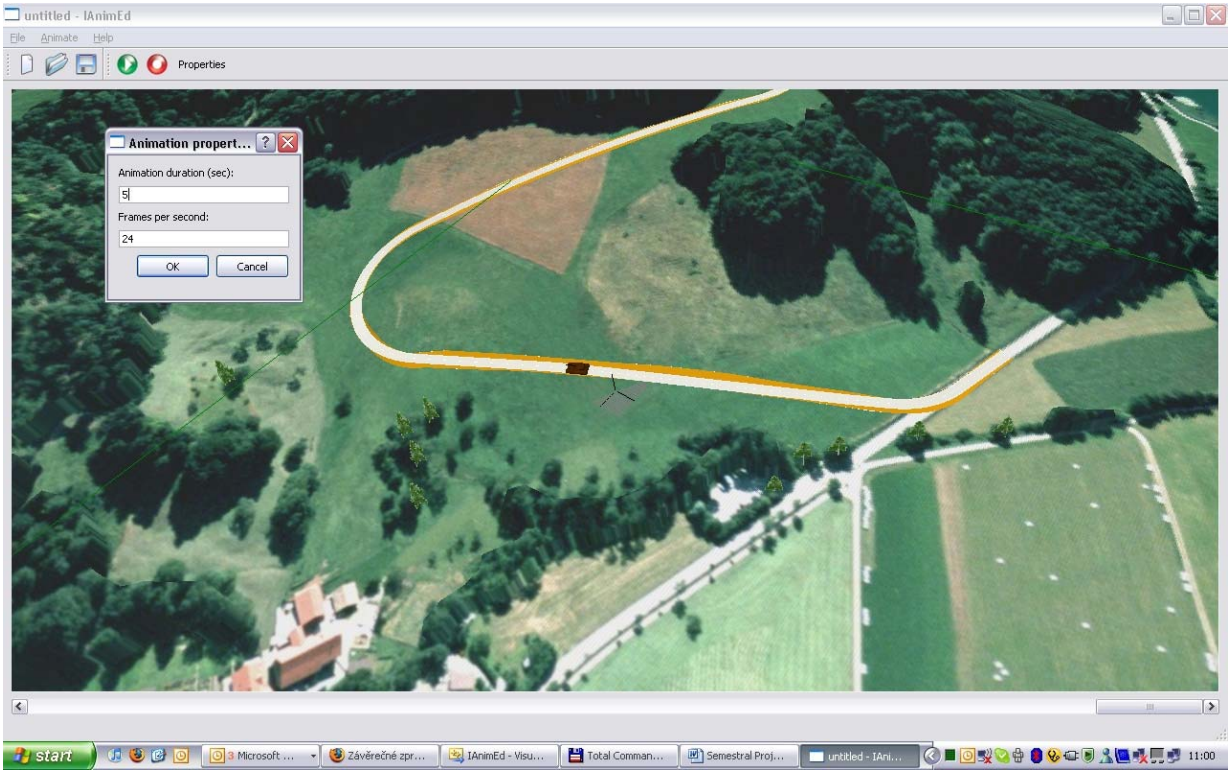
20. **Šimánek, O.** Principy virtuální reality, VRML. [Online] [Cited: January 3, 2007.] http://netra.felk.cvut.cz/~apg/apg-tutorials02/ch09s220.html.

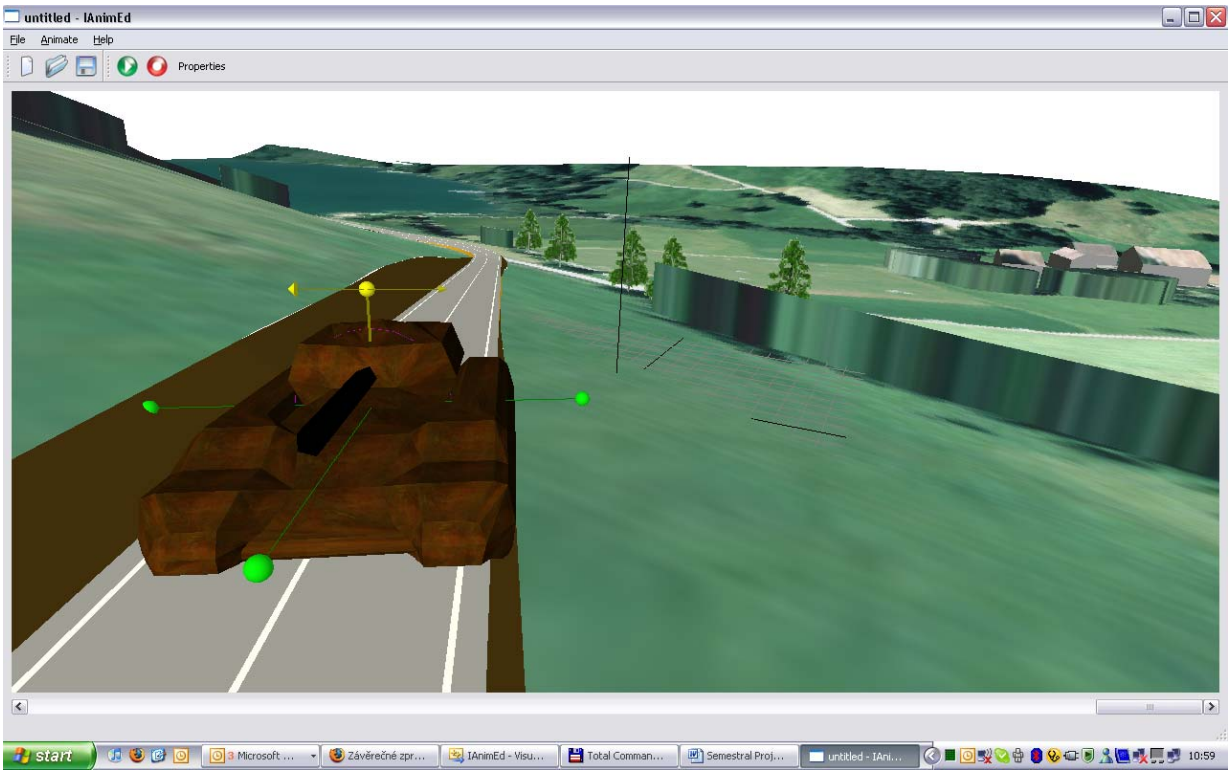
List of figures

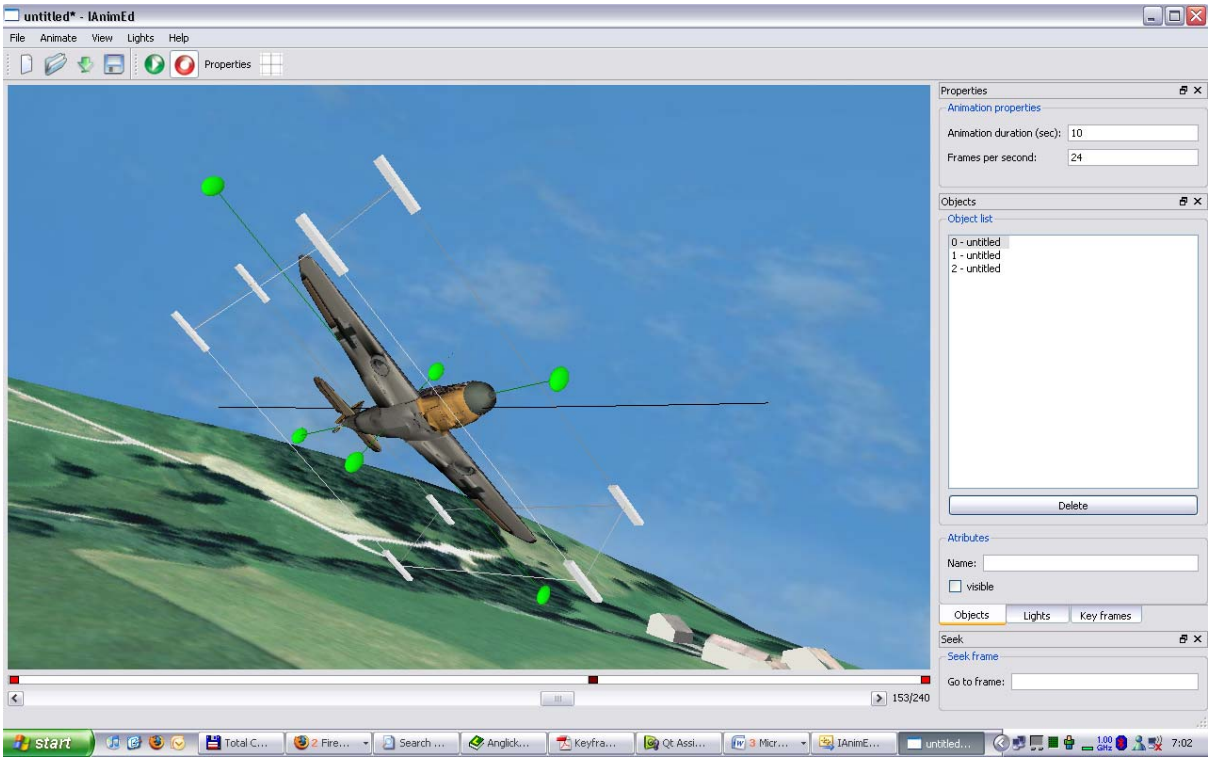
List of appendixes

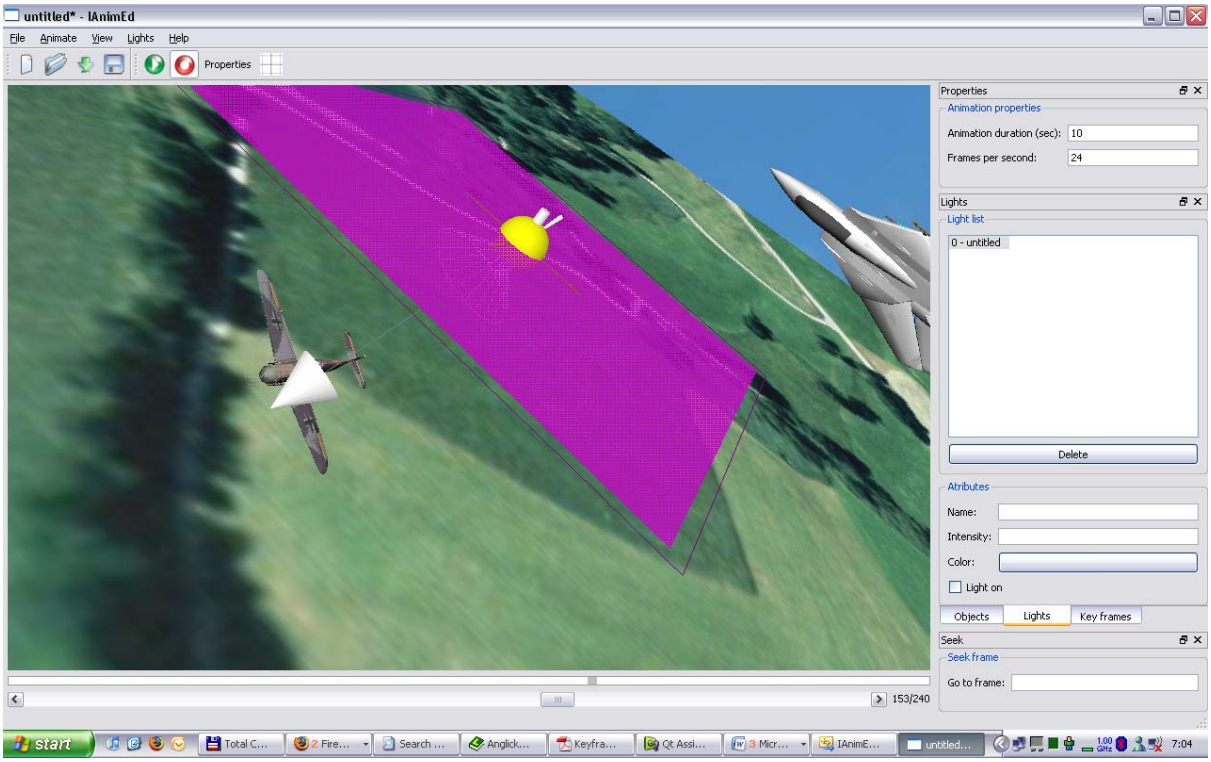
Appendix A Application screenshots

This appendix presents several screenshots of IAnimEd animation editor.






untitled* - IAnimEd
File Animate View Lights Help
Properties
Animation properties
Animation duration (sec): 10
Frames per second: 24
Objects
Object list
0 - untitled
1 - untitled
2 - untitled
Delete
Atributes
Name:
visible
Objects Lights Key frames
Seek
Seek frame
Go to frame:
153/240
start
7:02


untitled* - IAnimEd
File Animate View Lights Help
Properties
Animation properties
Animation duration (sec): 10
Frames per second: 24
Lights
Light list
0 - untitled
Delete
Atributes
Name:
Intensity:
Color:
Light on
Objects Lights Key frames
Seek
Seek frame
Go to frame:
153/240
start
7:04

Appendix B Keyboard shortcuts

This appendix presents list of keyboard shortcuts used in the application.

- R Record toggle
- G Grid toggle
- Alt Camera rotation
- Esc Toggle viewing in the editor
- Ctrl + P Play the animation
- Ctrl + I Import model
- Ctrl + N New scene / reset scene
- Ctrl + O Import saved scene
- Ctrl + S Save / export current scene
- Ctrl + Q Close the application