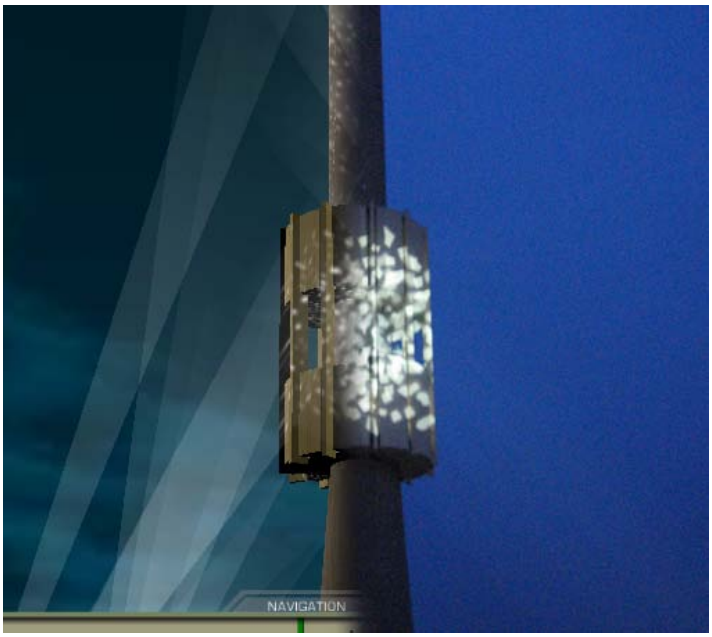
Aalborg University Copenhagen

Medialogy Cand. Sc. – 10th semester project

Project Report

"DMX Director" - Architecture of a 3D light-programming application, in a multi-user Internet environment



29/05/2006

Group number: K1005

Student:
Smilen Dimitrov

Project Supervisor:
Stefania Serafin

AALBORG UNIVERSITET

Abstract / synopsis

The system architecture described in this report, is an integral part of an event, where a two-week light-show is envisioned, with multiple light projectors being placed around the chimney of a power plant. The central idea in this event is that the programming of this light show, is made by pupils of participating secondary-education schools in the neighboring area, in the months preceding this two-week event. For the purpose of programming the lights, the students will use a simple web interface, which will display a 3D model of the location, as well as allow for light programming facilities. The programming made and recorded through this web interface, shall eventually be used as the driving data of the actual light show itself. This document is a discussion of the major problems and issues in providing a general solution for a prototype of this kind of a web interface, internally called "DMX Director" during development.

Acknowledgments

I would like to thank my parents, Vladimir and Todorka Dimitrovi, for their support during the course of my studies; Thomas Brockmann, Jan Bryld and the rest of Brother, Brother & Sons ApS, Copenhagen, Jens Lind of Seelite, Aps, Copenhagen, Vincent Agerbach, Allan Johansen and Brian Johansen for the great experiences and cooperation during this project; project supervisor Stefania Serafin, for the kind assistance with this report; and all those that helped me through the paper-and-pencil bachelor days at University “Sts. Cyril & Methodius”, Skopje, Macedonia.

Table of Contents

1 Introduction

The system described in this report, is an integral part of an event, entitled "*Fra Damp til Digital*" [40], where a two-week light-show is envisioned, with 60 light projectors being placed around the chimney of a power plant, located at the harbor in Randers. The central idea in this event is that the programming of this light show is made by pupils of participating secondary education schools in Randers Kommune, in the months preceding this two-week event, which is scheduled for October 2006. For the purpose of programming the lights, the students will use a simple web interface, which will display a 3D model of the location, as well as allow for light programming facilities. The programming made and recorded through this web interface, shall eventually be used as the driving data of the actual light show itself. The name used for this web interface system during prototype development was “DMX Director”, which is used as the title of this document as well.

Figure 1. Images of the tower in Randers during light projector tests

Figure 2. Images of the area surrounding the tower in Randers

Each class from participating schools will be allocated 15 minutes for which to provide a light programming. For each day of the event, a playlist is made and a definite timeslot is allocated to one of the participating classes; the playback of light programs each day is limited to 1 hour (along with a fixed light scene being displayed for a few additional minutes at the start and end of each session), and is scheduled for the evening – to allow family and friends of participating pupils to follow the unfolding of their light-programming live. Video recording is also scheduled from three camera locations around the tower, whose output is planned for two purposes: a live Internet video webcast of each day's show, as well as recording for the Danish regional station TV Danmark, where these recordings are planned as a video background in the hours when there is no scheduled programming, during the days of the event.

The two-week light show event, "*Fra Damp til Digital*", is a part of a greater celebration event set to celebrate the 100 years of electrification of Randers Kommune in Denmark. This celebration event, entitled "*Strøm i 100 år*" [41] is organized by Energi Randers, the local energy and electric power company (which manages the lit power plant as well), with Peter Westphael as event coordinator – and it is envisioned to contain other happenings besides the light show in the autumn months of 2006, such as concerts, child theater festival and exhibitions.

The company standing behind the planning and execution of the light show event is Seelite A/S, from Copenhagen-Århus, Denmark – the biggest light production and rental company in Denmark. Already since the early stages of planning of the event in late 2004/early 2005, Seelite contracted Brother, Brother & Sons ApS - a development company working in the area of lighting and projector technology from Copenhagen, Denmark, for the purpose of developing the web interface. As the project developed, in 2006, a group of M.Sc. students from Medialogy studies at Aalborg University, consisting of Vincent Agerbach, Allan Johansen, Brian Johansen and Smilen Dimitrov, as software developers, joined BB&S and Seelite in providing a solution for the web interface part; in the particular the 3D interface developed in Virtools. At the time of writing of the report, when the actual light-show event has not yet happened, this web interface – DMX Director – has reached a level of a working prototype, where with minor performance issues, almost all of the envisioned functions can be performed. This report is a discussion of the major problems and issues in providing a general solution for this prototype of the web interface.

2 Initial considerations and analysis

Within the initial discussions while approaching the problem of the web interface, several issues immediately proved themselves important. As first, management of the light show, and other administrative tasks related to the setup of classes and their light programs on the event days, should have already been accounted for in the system, before the pupils start logging into the system and making their light programs. That consideration, coupled with the fact that a multitude of online users are expect to save their programs in a centrally accessible location, immediately poses the problem of designing an online database solution; which in turn demands a design of an administration system. Hence, a strict definition of levels of usage of the entire system is also needed as a part of an overall database administration system.

The other problem is to find a platform that can render real-time 3D over the web – and to practically solve the real-time rendering of a virtual world that is dictated by a user-created light program; while at the same time keeping in mind that the end target group are school pupils. More precisely, the exact target group are pupils from the 7th to the 10th class within the Danish basic education system (which comprises both primary and lower secondary education) – typically teenagers within 13 to 16 years of age. In the end, it is them who should find the user interface interesting, motivating, and easy to use and understand. The problem of organising how a class (composed of maybe 10-30 pupils), should divide the 15 minutes (allocated to the entire class for a light show) between themselves, naturally arises as well, and it too demands to be considered in the database design. Therefore, in addition to finding a platform that can deliver 3D in a browser over the web, it must also communicate to a database system. Naturally, the database system should discriminate between users, and allow personalised storage of a light program for the event. As such, the problem of user authorisation arises as well; both in the end user interface that the students will use, and the administration interface.

The administration of users and database design is however not the only problem of such a specific web interface. When talking about a physical light show, executed by a number of projectors, and a web (or any) kind of interface through

which it might be programmed, the specifics of the possibilities of the projectors cannot be ignored, and must be accounted for – as eventually, what is demanded from such a 3D interface is a realistic representation of the designed light show, as it would unfold in reality. This demands that the modelling of the environment is taken into account, and the realisation that in essence there are two parameters of the modelling:

1. Realistic 3D modelling of the environment
2. Realistic modelling of the light reflections from this environment

Each of these problems poses a difficulty in development in its own right: modelling the environment is a difficult process, which requires precision and care; as well as considerations in the precision of the model, in relation to the rendering capabilities of the target machine. However, most 3D engines can be expected to be optimised for the display of world model data, once the model is inside.

On the other hand, realistic modelling of real-world light reflection, as in the particular case of standard industry light projectors, cannot be expected to be a standard function in 3D engines. Most light projectors today are programmable through an industry-standard serial digital protocol known as USITT DMX-512/1990 [39] (hereafter referred to as DMX), and will provide a light cone with a known angular spread. The colour of this cone can be controlled real-time via DMX, usually by a light-programming console, as the source of the signal. In addition, a lot of light projectors feature small metallic or coated glass discs with engraved patterns known as gobos, whish are inserted into projectors on rotating wheels; thereby patterns are obtained when light from the projector is cast on a surface. Gobos are pre-selected on the projector wheels before a show; they can however appear one at a time, and they can be programmatically changed via DMX – so at a particular instance in time, a particular projector can be changed to a different displayed pattern.

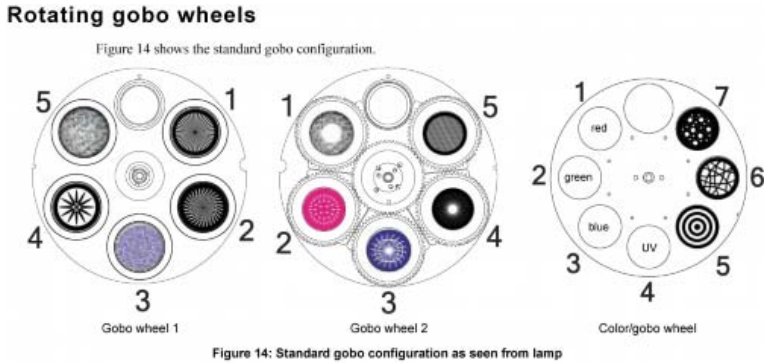


Figure 3. Image of rotating gobo wheels (Martin Mac 2000 spot projector user manual, from Ref.[1])

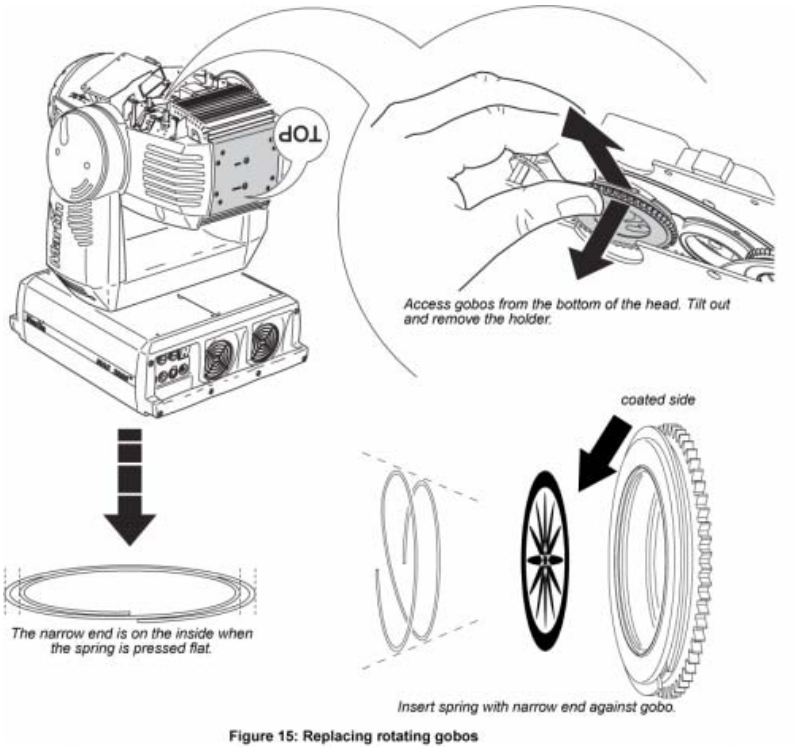


Figure 4. Image illustrating replacement of gobo wheels in a projector, illustrating their placement in it (Martin Mac 2000 spot projector user manual, from Ref.[1])

Certain projectors also allow for rotation of individual gobos, and mixing of patterns from gobos placed on several wheels - and the possibility of controlling all this via DMX is there as well. The importance of choice of gobos in light design is just as big as the importance of choice of colour of a given projector, so it has to be taken into account into the web interface. Besides, it can be easily assumed that pupils will find an interface allowing possibility of programming patterns of light, much more interesting than just the option of changing colour – and every feature that might increase their motivation is of importance, as it may be expected that in that age, some teenagers may find the task of programming lights overwhelming. However, realistic modelling of light cones and rotating gobos reflecting from 3D model surface is definitely not something that can be expected to be a part of the standard programming interfaces of 3D engines: although a “spot light”, a model of light with a spread cone which does model projector spotlights, is a common lighting model in many a 3D graphic cards and programming environments, its use is usually limited, and it can be hardly expected that it provides for something similar to gobo patterns. Hence a custom solution needs to be implemented, which could model gobo pattern light reflections from 3D model surfaces, for an amount of at least 20 projectors, in

real-time, with information about individual rotating gobo patterns – with a decent level of realistic approximation, yet without choking the 3D engine, which still has to be able to run as a part (meaning with the limitations) of a web interface. The situation is made more problematic by the fact that the exact range of PC machines intended to run the web interface is unknown; however, it can be expected that most of the use will take place in school classrooms, on computer equipment that may not be exactly up to 3D rendering requirements – and this is yet another important point to consider.

Finally, when talking about the 3D interface, one needs to consider the question of what functionality is needed, so that a light program sequence (for a given environment and light projector set-up) can be entered and stored by pupils aged 13-16 in an efficient manner. As an easing circumstance, most of these teenagers can be considered computer literate to a certain point, especially in the context of video games. Hence, there was a natural determination from early on to make the user interface as simple and close to video games as possible, to motivate the students to understand the task of light programming as a game. In that sense, certain things can be inherited from contemporary 3D video games, like camera navigation styles and general design and usage of an accompanying 2D interface, however there is one specific point as well. That is, as mentioned before, facilitation of entering and playback of a light program.

The light program can be considered a sequence of *cues* - a light design terminology for data structures, which carry a timestamp (when to execute) as well as corresponding information for the state of a given projector or range of projectors, such as projected colour, gobo, inclination etc. at the given time. For most contemporary projectors, such properties are addressable by protocols such as DMX-512, and cues can be programmed into light consoles or dedicated software; their real-time playback through these hardware engines generates a stream of DMX data which changes the corresponding properties of the projectors at the needed time. This technique of operation naturally has to be reflected in the 3D interface as well. The pupils, while using the 3D interface must have an awareness about a cue, what it does, and what is more difficult, how it relates to the time frame of 15 minutes allocated to the class. Obviously, this demands a graphical solution, such that it does allow realistic creation of a light program, yet it is easy enough to visually understand and use by an average pupil. Henceforth, one needs to resort for inspiration to solutions already

implemented in audio and video sequencing software, where visualisation of cues in relation to time is a standard problem; however, it cannot be expected that pupils of the target group age would, on a general basis, have experience with sequencing software, whether audio or video (although some of them might). Eventually, one must remember that, the data which is rendered as cue information must be storable and retrievable in the database located at a remote server – and technically implementing a connection between the cue, visually rendered as time related information in 2D, and effectuation of its content in the 3D world in realtime is yet another problem, which may (and indeed, does) put additional demands on the rendering engine.

Finally, it must be stressed the light program that ends up being stored should be also used as master source for the actual event and the real light show – that means it has to interface in the world of light consoles, projectors and DMX; and eventually one mustn't forget that the entire web project must be eventually be integrated with the rest of the web content related to the centennial event, possibly meaning transport of the system to the web server hosting the event related web site. Here it can be added that the event related web site, will also host the web pages carrying the webcam streams, as well as general information about the event, however that is out of the problem scope of this project.

Although it may be difficult to summarise the results of the initial analysis in brief statements, several classes of problems naturally emerge:

1. Database design, allowance for user distinction and storage of individual programs, and a corresponding administration interface
2. User authorisation, administration, and data security and integrity
3. Targeting a wide variety of users, and thus wide possible spectrum of hardware and viewing platforms (both in the sense of monitor displays and of web browser software), as well as a variety of available bandwidth via internet connections
4. Delivery of interactive, 3D content through the web – hardware and software compatibility issues
5. Design of realistic 3D models of the environment for the web interface
6. Design of a realistic preview of the light show within this environment, dictated by a light program

7. Design of a cue programming engine and interface, integrated with both the rendering preview and the database storage – and eventually producing data that can serve as a master for the actual light show.
8. Design of the previous points (5-7) considering that the end target group is teenagers, which will use the interface as a part of a school activity, for a limited and brief amount of time.
9. Integration of the development product with the rest of the web site structure, related to the event.

3 Related work

This project is in many respects unique – hence finding related work to use as a reference was difficult. The primary idea of the system – to allow light programming through an online interface, to a multitude of users, with special focus to inherit the look and feel of a 3D game (so as to cater to the age of the primary target group), and end purpose to reproduce the saved programs in the real world, with an actual light projector set-up (which had been simulated in the interface) – does not seem to have been explored previously. Internal research at BB&S couldn't identify similar projects. That is why a lot of research presented in this document is quite basic – looking at similar 3D interfaces for light programming on one hand, and looking for GUI solutions for sequencing and playback tools on the other, for instance; so as to provide a basis for development.

A simple way to describe the system could be as a 3D display system, within an online multi-user collaborative environment. This setting has been a topic of academic research: Naemura et al discuss a multi-user immersive stereo system, which is a proposal for "*a multi-user system, in which several users can feel the adequate depth sensation simultaneously* [33]", where the users are supposed to share the same multi-screen system and view panoramic stereoscopic images. Broll et al work in an augmented reality environment and propose "The Virtual Round Table", where "*augmentation is realized using semi-transparent stereo projection glasses* [34]" and the users can collaborate within an environment composed of both actual visual data and computer generated visual cues. Common for these two projects is that they work with more or less custom display hardware (like multi-screen environments or semi-transparent projection glasses), and most likely custom communication solutions within the system – in that sense, they are experimental, as deployment of technology to a user base is nowhere an issue. On the contrary, this is one of the first problems in this project – how to distribute the program to the school classes, and ensure hassle free experience; which also includes some considerations about what machines might be available to the schools.

Leung et al work in an environment technologically much closer to the level in this project – they develop a "*multi-user 3-D virtual environment called NetICE,*

Networked Intelligent Collaborative System [35]" aiming towards "*transparent communication between participants to make them feel that they are meeting in the same place despite their physical locations* [35]". Apart from innovations like using a light pen in a context of an interactive whiteboard, the display hardware here seems to remain the desktop monitor in a PC environment, as authors discuss problems of avatar representation and directional sound, in addition to the collaborative aspects of their system. As such, this project exemplifies similar intentions that exist in the public domain, which is to provide avatar based 3D environments for a multitude of users, which can interact and collaborate real-time – which in most obvious form can be seen in the emergence of MMORPGs (massively multiplayer online role-playing game, [36]) which are based in 3D (note some MMORPGs are text based), like for instance Anarchy Online [38].

Whereas the academic examples do not necessarily take into account target groups, available hardware and deployment of applications, in the case of MMORPGs that is certainly important – just looking at an overview of such games, like [37], it becomes apparent that an important parameter is whether the game is browser based or client based. Hence, it could be said that this project is more closely related to the problems in this context, although with its own specific properties. The main difference is that MMORPGs are developed so they "last" – that is, the intent is that they are being played continually. As such, it pays off, both for the user and for the developer, to go toward development of a custom, binary client engine. For some 3D applications, the process of installing such a client can be quite tedious – for instance, Anarchy Online has a client download of over 500 MB, and the entire installation process, along with updates can take up to several hours on a client machine with low internet speed. Once that is done, however, the user can play the game (provided it is compatible with the hadware) "indefinitely" – only updating the client periodically.

The important difference in this project is that the time frame in which the application should be used is strictly limited in a window of a month or so. The application purpose "expires" so to speak, as soon as the event (the actual light show) is performed. As such, continual updates are not applicable as an approach, and in that sense, the effort put into developing a custom binary client is not reasonable, provided the great troubles that regularly occur on such development – especially in the sense of compatibility, once the client has to be installed on the target system. This aspect of the application having a limited time of usage and an "expiration date" raises

the issues about technological deployment, which becomes almost critical in the context of multi-user consumption – and deployment is usually not a topic of discussion in similar research.

In addition, one needs to consider that once a class gets allocated a 15 minute timeslot, a problem arises of how to distribute that slot between student groups within the class, yet at the same time allow student groups to visualise how their particular part relates to the entire programmed show. Because of such consideration, the current version of the prototype does facilitate collaboration – however, that is a non-real-time, asynchronous collaboration, with a slow frequency of information exchange (a comparison can be made to interaction through an online forum, or e-mail communication). For that matter, most games and academic research focuses on multi-user interaction and collaboration in, ideally, a real-time setting; instead, here the focus is on the individual user interaction with the world, and the collaborative aspect is expressed through the principle of dividing a class allocated timeslot in group slots, and giving the end users per-group permission.

Let's just mention that on the matter of technology deployment, the development on this project focused upon web browser plugins as a delivery technology for 3D; although this does not seem to be a hot topic of discussion at this moment, online documents like [8] or [12] can serve as a starting point. Finally, we should note that the specific focus on usage of light projectors is also not much present in the projects of this kind, that mostly deal with software development and networking – most of the research in that area seems to be conducted internally within the industry (possibly because of the prohibitive prices of such equipment). Some of these issues are elaborated in more detail further on in this report.

4 Development work phases and report structure

It is obvious that answering the demands that arise from the initial considerations is a complex task; and it can be hardly expected that all functions can be performed using a single interface. Hence, it was determined early on, that the solution should include two separate interfaces, an **administration interface** and a **user interface**, catering to the needs of four general groups of users of the system: ***administrators***, ***teachers***, ***student groups*** and ***guests***. This, in short, also represents the general model of the current web interface prototype, which is discussed in more detail further on in this report.

As a starting point, initial inquiry into the feasibility of several platforms was undertaken, and in that relation, BB&S provided the domain name dmxdirector.com and the corresponding web space, as a development server for prototyping work. An initial prototype was developed, which although non-functional, served as a basis for analysis of further developments in the project – this prototype is briefly presented further on in this report.

After evaluation of this initial prototype, the choice fell on Virtools as the main development platform for the web 3D interface, and development begun on the prototype version described in this document. During the development, two field trips to Randers related to this project were undertaken as well (once together with the software developers), where projector set-ups were tested live, and photo documentation was taken – part of which is presented in this report.

Eventually, it is difficult to note down all the details of the considerations during the several months of intensive development work in this project, in a design-and-implementation model traditionally employed for web applications. The project work was pretty even, although several phases can be identified, which had several iterations of prototypes within them, and each version would address different aspects of the issues introduced so far. Thus, it seems plausible to introduce only the general phases of project work and the most important version changes. Though, as the details become simply too numerous to investigate in full, two topics emerge as technically crucial in discussing the proposed system:

1. Database and user set up and administration
2. Development of a cue programming and playback (sequencing) engine, integrated both with a real-time display of a 3D world and the above-mentioned database.

In addition, the focus on targeting pupils acts as a constant constraint, slightly more special than the other technical details.

Thus, this report will introduce the general model of the system and the initial prototype. Thereupon, database and user set up will be discussed in a separate section, through a brief discussion of the current version of the administration interface, noting important version changes and relations to other important design details as appropriate. Similarly, the second topic – development of a cue-programming engine, integrated with a real-time 3D display and a database, will be discussed in a separate section, through a brief discussion of the current version of the 3D user interface.

It is important to note that at the time of writing of the report, the most current version of these interfaces do not perform the entire spectrum of planned functions, although they perform most; and the report will correspondingly discuss this (most current) version. Also, at the time of writing of this report, the real light-show event is still a few months ahead (although time does indeed fly fast); yet, the current version of the prototype, as it has most of the planned features, is already being presented to teachers of schools in Randers Kommune participating in this project.

5 General application model

As mentioned before, it was determined early on, that the solution should include two separate interfaces, an administration interface and a user interface, catering to the needs of four general groups of users of the system: administrators, teachers, student groups and guests. Both interfaces would call on the same central database, located on the remote server, for user authorisation and personalised data retrieval, as rendered on the figure below:

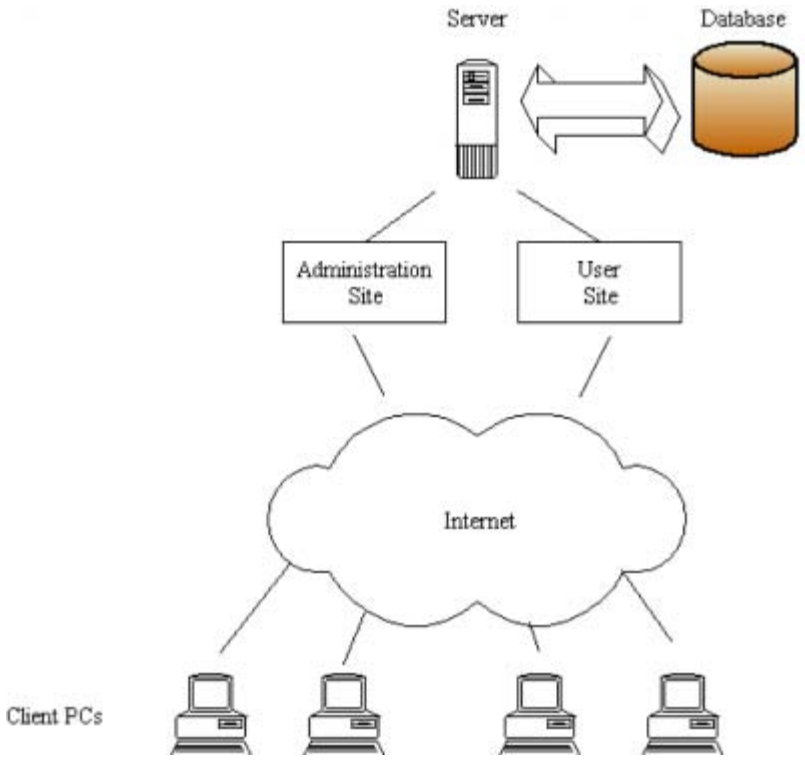


Figure 5. A general model of the proposed web interface

Obviously, the different uses of these web interfaces will determine both the classes of users allowed access to them, and the technical solution of formatting of the content. The administration site is, of course, intended to administer the database related data, such as participating school names, classes, reservation of time slots in the eventual real light-program, as well as administration of logins and passwords. As such, in general, it doesn't require more then standard display of HTML formatted textual information; administrating database information through HTML tables

rendered by a server side script is a standard practise. The user site is the 3D web interface, intended primarily for use by the pupils, and it is here where delivery of 3D content on the web, and the technical details of implementing a cue-programming engine, are of essence.

The implementation of this general model will be discussed in more detail in the corresponding section, "Database and user set up and administration" – as it is mostly concerned with the back-end perspective. For now, let's just mention that because of the specific uses and users accessing each part, the user access was included in this model already from its inception, as rendered on the following diagram:

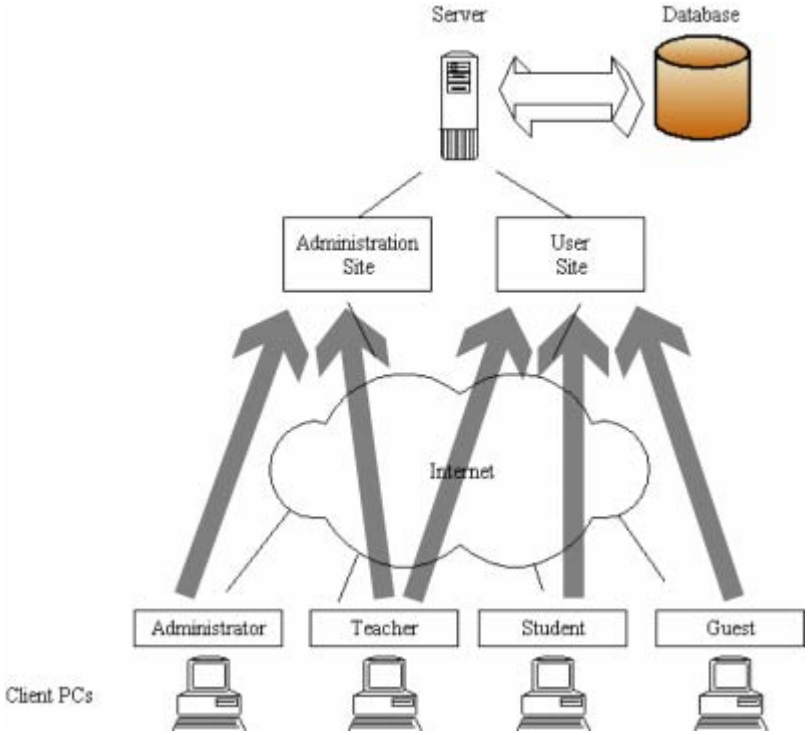


Figure 6. The general model of the web application, with illustration of user access rights

That is, only a teacher identity is meant to have access to both interfaces – however, with specific limitations. The details of such user organisation are also discussed further in this document.

6 Initial prototype

Due to the specific differences between the functionality of the administration and user web interface, different technological implementation, and the need to eventually move the application between servers, early in the development it became of importance that both applications are written in that way, so that all corresponding files can be put into a folder, and all links would be relative to the root of that folder. Ideally, a copy of such a folder on the server would again represent a working clone version of the respective site, without additional effort put into server set up.

A folder on the dmxdirector.com server was allocated for the development of initial prototype – at that time simply called SkorstensBelysning, which hosted the separate folder for the administration site prototype, as well as a simple demo of a 3D interface, accessible through a single web page, which represented the user site. These applications were accessible through the web through the following links:

http://dmxdirector.com/SkorstensBelysning/Admin/

http://dmxdirector.com/SkorstensBelysning/SBLysProto2.0.htm

A screenshot of this early prototype of the administration site is given below:

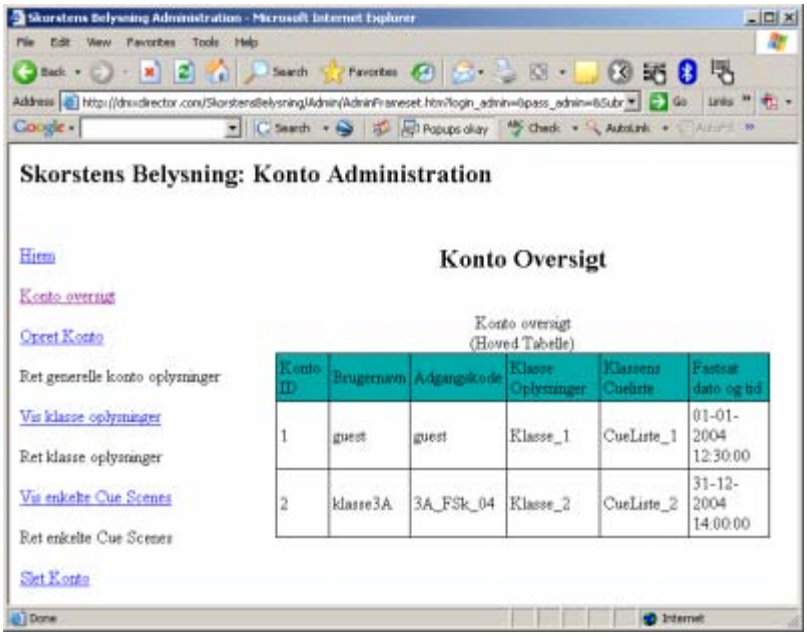


Figure 7. Screenshot of the early administration site prototype

This initial prototype featured a non-functional login page, after which, as can be seen, it is a simple HTML based website for table administration. It was written in ASP .NET C#, connecting to a Microsoft Access database. It was the first attempt to provide some sort of a way to administer users, and correspondingly, information in the database. Mostly, it built on ASP .NET components like DataGrid to provide display of database tables as HTML tables. As such, it was the first attempt at designing the database as well.

This initial database design revolved around a main "account table" – which stored information about administrators, teachers, and the guest account. Each of these entries would be linked through one of their columns to a separate table, describing the class (if applicable) and containing the eventual light program. Such a design of the database proved to be most problematic to code, as often one had to calculate a given table's name before writing the SQL statements, which just complicates the logic. Hence, the design of the prototype is extremely simple (there is no styling whatsoever) and it simply serves to demonstrate that a connection to a database is achieved, and one can create, modify and delete accounts using this simple interface. Furthermore, the application worked by passing query strings through the URL back to server, which are of course visible in the web browser address bar each time a command is performed, and in that sense can pose a security risk, if not properly handled.

Screenshots of the early prototype of the user site is given below:

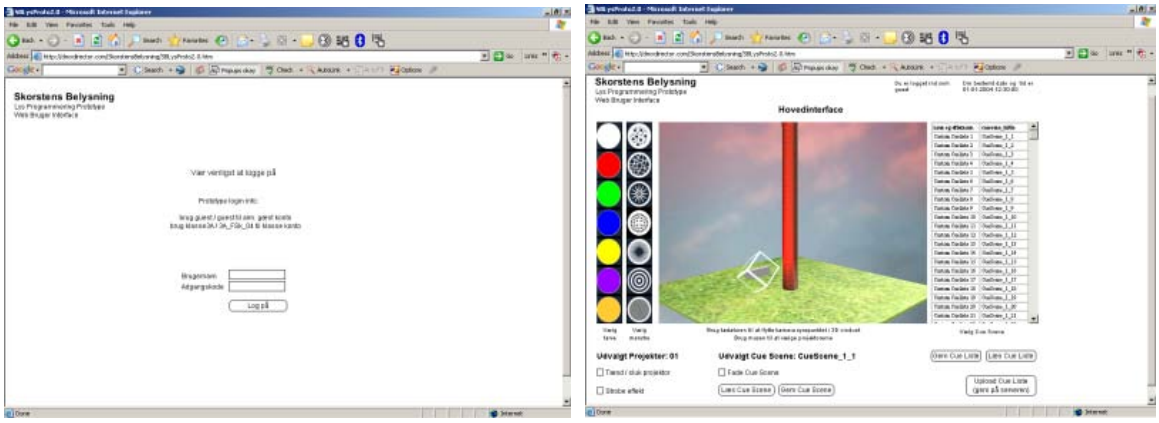


Figure 8. The login screen (left) and the main screen (right) of the initial prototype user site

The user site prototype was accessible through a single HTML page, as it was programmed as a Macromedia Director movie file – all additional branching in the application is achieved internally. In essence, it consisted of three parts, two of which are presented in Figure 8: Login screen, walkthrough (or help screen – not on the figure) and main screen, where the 3D is rendered.

This demo featured a simulated user authorisation (that is, there were legal login/password combinations hard-coded in the application, to simulate responses for wrong login – however, no connection to the database, or checking against the data there was ever made), whereupon the user was presented with a choice of going through the walkthrough or going to the main screen.

The main screen featured a simple 2D interface, with text simulating representation of personalised data (such as school name, class name, or scheduled event date for the logged identity). A text table with a copy of database contents was also included, more or less as a reminder that other class data might be necessary to display in the versions further on. This version of the prototype also featured an external textual dictionary file, which contained all of the texts related to the interface, so as to allow easy translation. In addition, the main screen featured a simple 2D interface, which illustrated graphically the main changeable parameters of the light projectors – colours and gobos; and of course the 3D rendering window.

The 3D rendering window displayed an extremely simple world model – the insides of a sphere, with cloud texture mapped on the inside, a simple plane with a grass texture as the world surface, a simple cylinder modelling the tower, and a box modelling the projector. There was a simple 3D selection and navigation engine implemented as well. The user could navigate the camera using the arrow keyboard keys and/or the right mouse button; hovering the mouse over the projector would result with the standard Windows “hand” mouse icon, clicking on the projector would select it – which was indicated by a rotating transparent wireframe box. Deselecting the projector was also possible, by clicking elsewhere in the 3D window.

The actual projector consisted not only of the box mesh, but also of an actual spot light model fit as a child inside the box. As such, Director’s 3D engine could automatically calculate the light reflection from the tower surface, and as such, this was a demonstration of one possible solution to accurately model the situation – by

using already provided spot light models, and using the built in rendering capability of, in this case, Director. In that sense, while a projector was selected, the user could click on the colour icons of the graphic interface, and the child light colour of the projector would be changed, which was indicated on the reflection as well. Hence, this was also a demonstration of the connection between the 2D interface and 3D output, which is necessary as part of the interaction design.

7 Initial prototype evaluation and platform choice

The initial prototype, in spite of its primitive functionality, is certainly a necessary step, first and foremost as an initial physical illustration of the entire system as a concept, which is useful both for communication about the project concept with other interested parties at that time of development, and as a base for critique in further development. It was also important as an initial test of technologies to be applied for the production versions of the product.

The administration site, besides being in need of a major redesign (especially since the user may need to administer time related data through it), also revealed that the database design at that point – with a separate table storing the data for the class – may be an approach a bit too complex for a start, which could complicate the database side of coding. This demanded a simpler database solution, which would allow more time for other coding problems. By that time, the roles of the administrator and teacher were more clearly differentiated as well, so it was obvious that the administration site must reflect this differentiation as well. Finally, working with Microsoft Access databases at that time, revealed that its largest string entry field, MEMO, is limited (generally) in size at 64 Kbytes (although that is not necessarily the case, see [2]). As eventually the database was expected to accept strings of text that would represent the light program, depending on the eventual solution, they could be represented by long strings of text – hence, a choice was made to implement a MySQL database instead, which can be utilised through an ASP .NET C# page by using its own ODBC driver [3], and does not appear to have such limitations: “In most respects, you can regard a BLOB column as a VARBINARY column that can be as large as you like. Similarly, you can regard a TEXT column as a VARCHAR column[4]”. In other respects, ASP .NET was left as a server-side platform for the administration site, mostly because of its inclusion of components that automatically deal with table

display, paging and indexing of database data, as well as components that deal with time formatting and display, which is of essence in the administration interface.

The user site presented a different set of problems. In essence, it demanded an evaluation of the 3D engine. Macromedia Director (now owned by Adobe [5]) is a plugin[1] technology that does not target 3D specifically, but rather offers it as an addition; and although it offers rather extensive software manipulation capabilities for 3D, even with only the simplistic model illustrated above, there seemed to be performance problems – for instance, the rendering did not seem smooth enough when interacting with the camera. As the eventual application should be able to support up to 60 projectors and display of their state according to a user-defined program, Director begun to seem unfit to sustain such demands. In addition, 3D modelling of the environment immediately stuck out as a critical point in the development, and previous experiences with Director up to that point were not glittering: Director accepted only the .OBJ file format, which for some modelling applications, demanded intermediate conversion into .OBJ before importing into Director - which then occasionally resulted with loss of hierarchical data. This is maybe even more of a critical problem, especially in a project like this, where the environment modelling, and alignment of code and model hierarchy, is of crucial importance.

At that time, Virtools [6] also became visible as an option. Other options existed as well – for instance using 3D Java [7], however, performance problems in the context of the project were envisioned here as well; other options, like Wild Tangent [9], 3D Groove[10] or Blender3D (web plugin not supported anymore [11]) were simply too obscure at the time, and getting acquainted with their potential quirks was quite risky. On the other hand, delivering a binary executable was also out of the question, due to the large expected variety of target machines, and possible OS incompatibility problems. In all, web delivery seems the best approach, yet it also seems that the web-based interactive 3D plugins at the current time are not as common as say, Macromedia Flash, so it is not easy to determine the “best” platform. In all, it seems that only serious contenders that can be trusted for 3D web delivery (for more, see [8]) in the context of this project are Director/Shockwave or Virtools; the only problem being that Virtools costs nearly 6 times as much as Director.

[1] the playback in a web page is achieved through a so-called Shockwave player plugin

However, the benefits of Virtools far outweigh Director – Virtools is targeted towards 3D (even game development for Xbox), can be trusted with the web plugin at least for Internet Explorer on Windows, and plugin installation initiates automatically. Furthermore, a scripting engine that allows both graphical program (through Behaviour Block objects) and traditional scripting through the C-like Virtools Scripting Language (VSL), with classes and methods oriented towards 3D world manipulation, is certainly a great ease in development, especially when details, usually expected from a 3D game, come into play. It has native web connectivity, and allows for programmable pixel and vertex shaders. It does sport an SDK as well, and can be extended with custom made DLLs; - however, using this functionality unfortunately increases the licensing fees. Eventually, increased assurance that the envisioned product can be delivered on time in all its complexity to a broad target group, and previous positive experience of the developers with this package, was what made the final choice fall on Virtools as the delivery platform for the 3D light programming user site, in spite of the staggering price tag. In retrospect, this seems the right choice to make, as it is doubtful if the prototype could achieve all the functionality existing in its current version, in the existing quality, in the same amount of time.

8 Version development overview

In any case, the development work continued in a similar spirit – a folder was allocated on the dmxdirector.com server, this time called /udvikling, which hosts the individual folders of the development versions of both administration and user site. In this manner, each new version can be independently tested online as it comes out, which would act as a preventive filter to detect problems in the published files (which proved fruitful on at least one occasion, when Virtools changed the web player version, and suddenly the user site versions stopped working). The administrative site folders are named AdminSiteXX, whereas the user site folders BrugerSiteXX, where XX is a version number. At the time of writing of the report, the latest versions of the administration and user site are, respectively:

http://dmxdirector.com/udvikling/AdminSite02/
http://dmxdirector.com/udvikling/BrugerSite20A/

The development initially started on the administration site, which was developed in two major versions (an additional, minor redesign was performed later on). Afterwards, work begun on the user site versions using models of a world provided in Virtools, in parallel with work on the environment modelling. Version 11 supported saving and loading of a light program sequence to hard disk, and version 12 did the same with the remote database. The first merge of the environment models with a user site engine was in 13th version (BrugerSite13), at which time there was already a basic cue and interaction engine set up built in. Work proceeded more specifically on coming up with and algorithm for the projector light simulation, as well as improvement of the environment models and rest of the engine.

Up to version 13, actual spot light models were used, and version 14 introduced a different algorithm, which could simulate a light cone and a gobo. In version 16, a complete makeover of the 2D interface started, which concluded with the current last version, BrugerSite20 (and a version with identical functionality, but with projectors set up according to the actual light design for the event, the aforementioned BrugerSite20A).

The rest of the document will describe some of the more important specifics of the administration system, and at the same time introduce the administration site in its current version, in chapter 9: Database and user set-up and administration; similarly, some specific issues with the user site will be discussed through a simultaneous introduction of its current state in chapter 10: 3D user interface development.

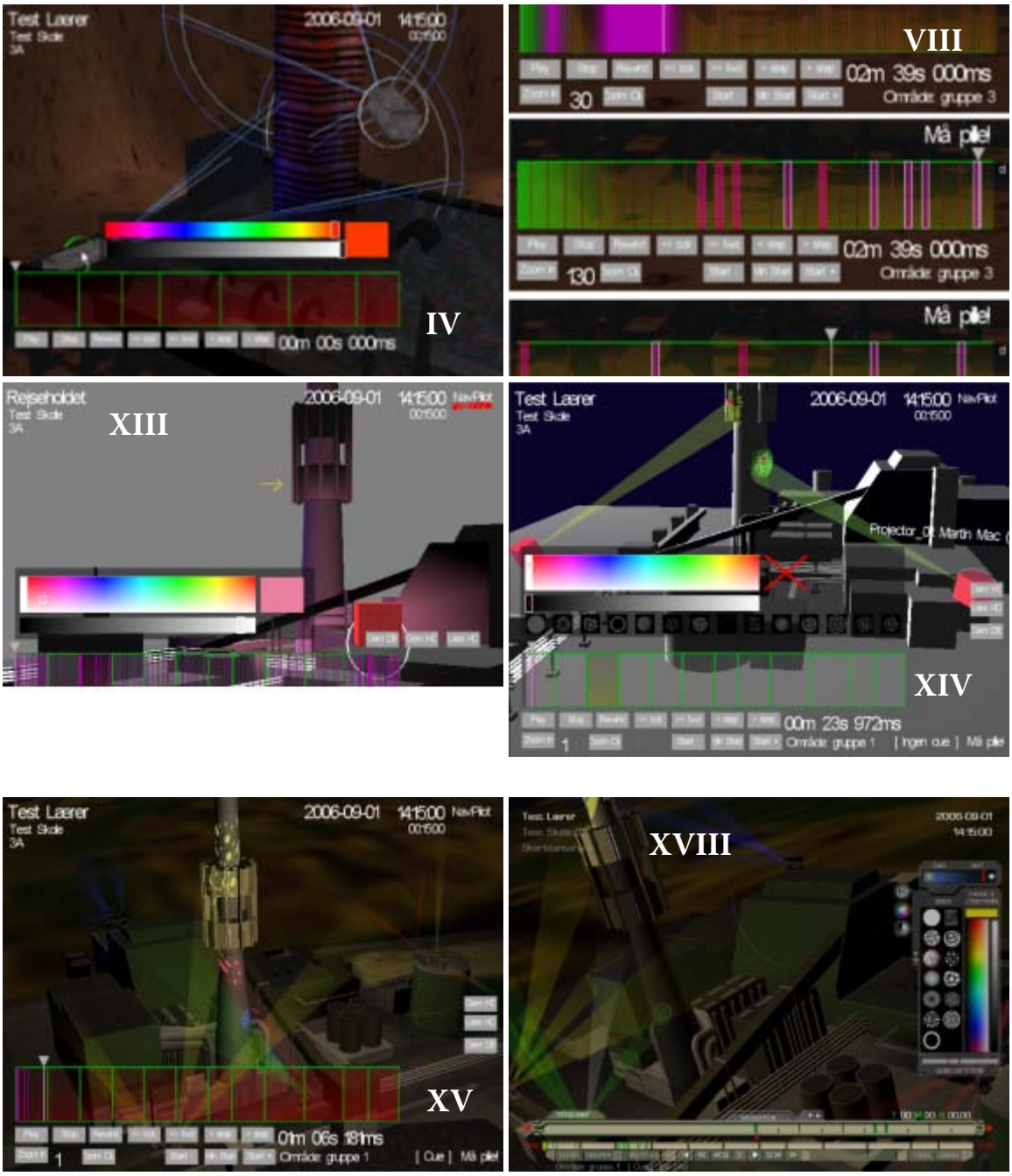


Figure 9. Screenshots of several different versions of the user site prototype. The roman numerals indicate version numbers.

9 Database and user set-up and administration

9.1 Usage phases and user set-up

As illustrated on Figure 6. The general model of the web application, with illustration of user access rights", there are four groups of users that the system recognizes. This is due to the highly specific roles that these users perform, at distinct times of usage of the system. The following time chart might help us visualise the main phases of usage of the system:

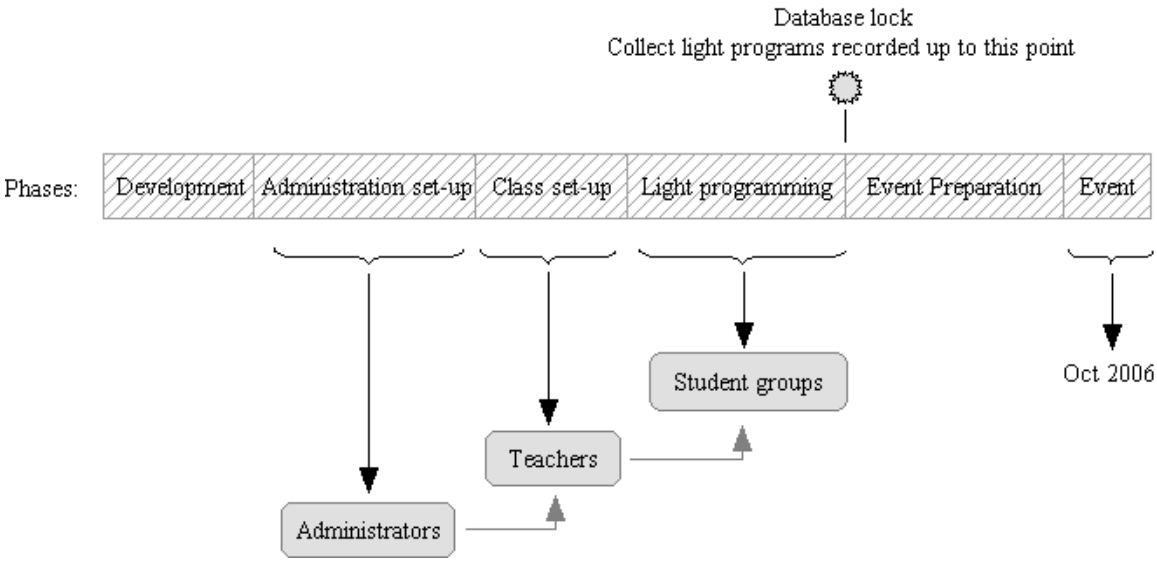


Figure 10. General timeline chart of the phases of usage of the web interface

The development phase is of course, the time period when the prototype is developed and no users have access to it yet. As functions become available, certain groups of users can progressively begin performing tasks in each phase. Here is an overview of the user classes, and their specific tasks in each phase:

- **Administrators** – have access to the administration site, and as first, can add, modify or delete other administrative accounts. Their main task is to collect the data about participating schools and classes, and allocating accounts for them in the database – which means reserving usernames, passwords and a 15-minute timeslot within the final event light program schedule, as well as entering related data. In

addition, they keep in contact with the teachers, as official representatives of participating schools, passing on information about the system if necessary. This is the *administration set-up phase*, mentioned above in the time chart, which obviously must be conducted before any actual users can log into the system. As the administrators have nothing to do with actual light programs scheduled for the event, and the user site needs to reflect the program for an allocated timeslot for a given class, the administrator logins/passwords are not recognised by the user site – however, as the administrator indeed has access to all teacher logins, an administrator can always log into the user site by choosing any teacher identity.

- **Teacher [class]** – One teacher account at the same time represents one class (hence the reference to both terms); if the same teacher happens to lead two different classes from the same school, this is differentiated by first and foremost a different login/password combination. From an organisational aspect, it is difficult to predict what are the expected numbers of students per class for participating classes, or how would a given teacher like to distribute the work on the light programs between the pupils of the class. The most immediate possibility seems to be that a teacher would divide the class in groups, and each group would receive a part of the class-allocated time slot. With such a general organisation, the system could cover the needs of both a class with 30 students divided into say 10 groups, as well as a class of 14 students divided in two groups. However, dividing the class-allocated 15 minutes to student groups is a responsibility that could be hardly delegated to an administrator, as it basically depends on the situation in each individual class. Thus, the teacher is delegated this responsibility: a teacher would login with a login/password combination, obtained from an administrator, determining a given class and its position in the final light program. Thereupon, the teacher could divide the obtained 15 minute timeslot into a desired number of groups, and set up logins, passwords and enter group-related data (such as student group names), thereby allowing the students of the class access to the system. This is the *teacher set-up phase*, mentioned above in the time chart, which must be conducted before any students can log into the system. Obviously, the teachers must have access to the administration system, however they need a separate interface as well. In addition, as a teacher login does point to a specific time within the event schedule, the teacher can have access to the user site as well; in this case, the teacher is differentiated in the user site by having control over cues in the entire 15 minutes allocated to the class.

- **Student [groups]** – Each student account refers actually to a student group which has a definite group slot within the allocated 15 minutes for that class; in the extreme case, a teacher may choose to give an account to each individual student (hence the reference to both terms). The student account, as it should, is only recognized by the user site - the students should not even be aware of the administration site. When a student group identity logs on to the user site, it should be presented with personalized information (such as group name), the actual event start for that class, as well as the entire light program for the class recorded up to that point. The students should be then able to modify the light program and add or delete cues, but only in the timeslot allocated to that group; when done, the students should have the possibility to upload the program to the server and save it. In that way, each student group can work independently of others, yet still have an overview over how does their particular design fit into the light design of other groups in the class, and follow the progress of the entire class light program as it develops. This is the *light-programming phase*, mentioned above in the time chart, which is when the light programs for the actual event are created. To ensure the integrity of the data, this phase is planned to be limited to a period of about one month; whereupon all the accounts are locked, and the light programs entered up to that point are saved as the blueprint for the actual event show; and this marks the beginning of the *event preparation phase*.

- **Guest** – this account serves a two fold purpose: to allow demonstration of the system to visitors outside of the project (meaning, not teachers or students of participating schools) and its functionality, without allowing interference with the database where the light programs for the events are stored. On the other hand, individual students from participating classes may want to try the system on their own time at home. Hence they could login as guests, and experiment without compromising the database saved programs. In this case, as programming a sequence can be pain-staking work, at least the option of saving a light program on hard disk should be present. The guest account is too, recognized only by the user site – and it doesn't even need to be included in the database; a singular login/password combination can be hard-coded into the system for this purpose.

These are the main user groups and the description of their tasks within the online interface, which is to be used, in principle, before the event. Let's just mention that the *event preparation phase*, among other things, deals with establishing the hardware and software solution that will convert the light programs saved in the

database, into DMX compatible data, so that they can be used as actual sources of the event show.

9.2 Database set-up

From the discussion so far, it can be concluded that although there are four main user groups, since the guest account can be handled with a single hard-coded login/password combination, the database has to consider only the first three classes of users: administrators, teachers and student groups. This serves as a basis for reorganisation of the database design provided in the initial prototype, into a simpler one, which indeed simplified the process of writing SQL statements and coding the data exchange with the database.

The current version of the database hosts only three tables, named UL1, UL2 and UL3 ('UL" standing for User List), set-up so that each row represents an account, and holding information about users in each user group, respectively:

- UL1 for Administrators
- UL2 for Teachers [classes]
- UL3 for Students [groups]

As each row represents an account, all tables have login and password column fields in their format. Determining the user level then is simply a matter of finding out which table does the specific login/password combination belong to.

The current format of the administrator table UL1 is:

	Field	Type	Attributes	Null	Default	Extra	Action
☐	id	int(3)	UNSIGNED	No		auto_increment	
☐	login	varchar(20)		No			
☐	password	varchar(20)		No			
☐	name	varchar(64)		No			
☐	email	varchar(64)		Yes	*NULL*		
☐	is_online	int(1)		No	0		
☐	is_active	int(1)		No	0		

Figure 11. Format of the UL1 (administrator) table (screenshot from phpMyAdmin)

The administrator table has id column, which simply serves as a primary key, and login and password fields as all other tables. Actually, the fields of the administrator table can be seen as the bare minimum that is also included in the other two tables in the database. The name field is only used to personalise the display in the administration site, and the email field is used in the email facilities of the administration site. The last two fields are meant as binary switches, indicating whether an account is active (an inactive account would not have access to the respective site), and whether the user is online (which is a special problem in itself, discussed later as well).

The current format of the teacher [class] table UL2 is:

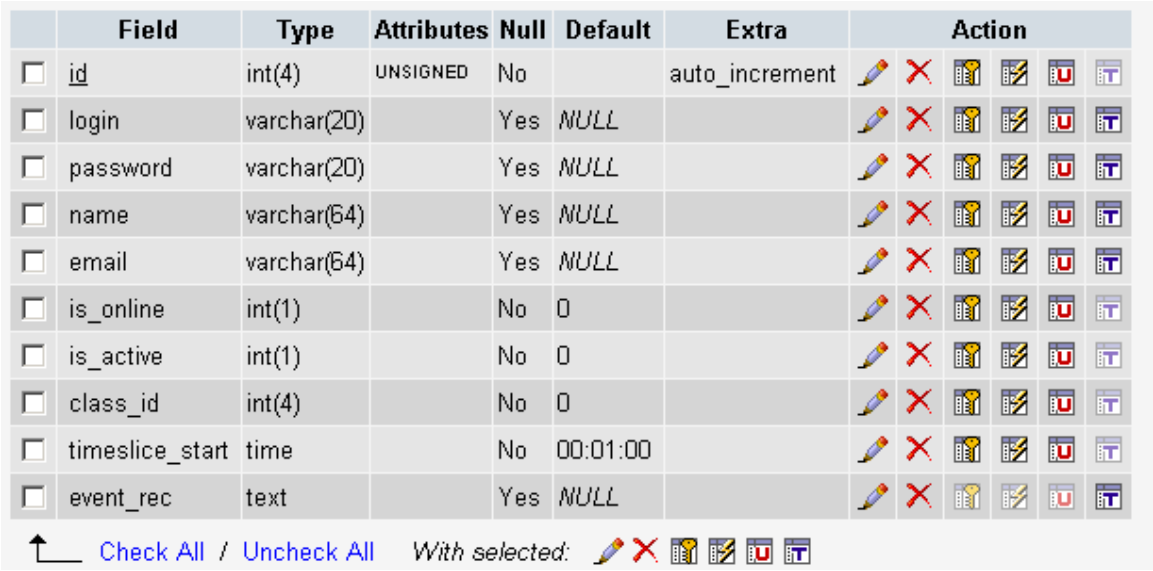
	Field	Type	Attributes	Null	Default	Extra	Action
☐	id	int(4)	UNSIGNED	No		auto_increment	
☐	login	varchar(20)		No			
☐	password	varchar(20)		Yes	*NULL*		
☐	name	varchar(64)		No			
☐	email	varchar(64)		Yes	*NULL*		
☐	is_online	int(1)		No	0		
☐	is_active	int(1)		No	0		
☐	school_name	varchar(64)		No			
☐	class_name	varchar(64)		No			
☐	timeline_length	time		No	00:15:00		
☐	event_date	date		No	2006-09-01		
☐	event_time	time		No	14:30:00		

Figure 12. Format of the UL2 (teacher/class) table (screenshot from phpMyAdmin)

All the fields from the UL1 table are here as well, as well as some class specific ones, such as school name, class name, and event date and time – referring to the actual schedule of the real event. There is also a timeline length field, so as to allow time slots of different lengths to be allocated to different classes. Although the system currently supports this option, there is a firm determination to use 15 minutes as the default timeslot length for all classes, so as to minimise problems in setting up the schedules for the actual event. From the database perspective, one could say that the

task of the administrator identity is to manage information stored in this table (as well as its own, administrator table).

The current format of the student [group] table UL3 is:



	Field	Type	Attributes	Null	Default	Extra	Action
☐	id	int(4)	UNSIGNED	No		auto_increment	
☐	login	varchar(20)		Yes	*NULL*		
☐	password	varchar(20)		Yes	*NULL*		
☐	name	varchar(64)		Yes	*NULL*		
☐	email	varchar(64)		Yes	*NULL*		
☐	is_online	int(1)		No	0		
☐	is_active	int(1)		No	0		
☐	class_id	int(4)		No	0		
☐	timeslice_start	time		No	00:01:00		
☐	event_rec	text		Yes	*NULL*		

Check All / Uncheck All *With selected:*

Figure 13. Format of the UL3 (student/group) table (screenshot from phpMyAdmin)

Again, all the fields from the UL1 table are here too. Added are class id, which can be seen as a foreign key, as it is a link to the teacher id, and only way to differentiate which class (and school) do the students belong to – as the design of the tables is one row – one account, this table's rows would represent a collection of all student groups from all schools. The field timeslice_start contains information about the start of the group allocated part within the 15-minute class timeslot, relative to the start of the 15-minute timeslot (the length of the group allocated part is derived from this start data of all groups in a class). Finally, event_rec is a text field, where individual light programs from each group are saved as strings. This also means that all programs from all students are saved in this table alone. From the database perspective, one could say that the task of the teacher identity is to manage information stored in this table.

It is unclear whether such a design is an optimal design in sense of database consumption – especially since the most critical data, which is also the largest, is all tucked into a single table. However, it is simple, and it allowed quick construction of

SQL sentences and troubleshooting while coding the database interaction. Finally, it works, and so far no database related problems were identified in the prototype testing process.

9.3 Administrator site walkthrough

The administrator site, in its current version is programmed as an ASP .NET application in C#, with a code-behind DLL (usually placed in a /bin folder at the server). The corresponding active server pages and other files are grouped into a folder, which for the current version is AdminSite02. The current version includes 25 files in the folder, among which three active aspx pages and thirteen so-called User Controls, or ascx files – a lot of them building on pre-programmed .NET components such as Calendar or DataGrid. As mentioned, the database is MySQL, and also, a small amount of Flash is used, which is why upon entry in this site, Flash auto detection is performed. Once that is done, and the correct Flash version is found, the user is presented with a login screen. The site also features a slight amount of styling via external CSS style sheets, and an external dictionary file containing all the texts displayed, to facilitate easy translation. JavaScript is also used on several occasions.

Upon finding the correct version of Flash, the user is presented with a login screen:

Figure 14. Login screen of the administration site

From this point on, if the user is authorised, either the administrator or the teacher perspective of the administration site will be presented – otherwise the login screen will be presented again. For both perspectives, personal information such as name and login status is presented on top.

For the administrator version, this is the initial screen displayed:

Figure 15. Initial administrator screen (help tab)

In both versions, after login, the user is displayed with a horizontal tab strip menu, containing clickable buttons that link to each section of the site for the given type of user, and the first choice on the menu, as well as the first displayed screen is the help screen, containing basic instructions. The administrator user has 5 choices in the administration site:

1. Help
2. Edit Administrators
3. Event Calendar
4. Edit Teachers
5. Send e-mail

The second and fourth choices allow the administrator user to edit the contents of tables UL1 (administrators) and UL2 (teachers/classes). This is done through a unified interface for editing database table data, which is built upon the DataGrid component in ASP.NET. The table is pageable, and five rows per page are displayed; the user can add rows and delete them, as well as modify their contents.

The regular view at a table through this interface looks like this:

Figure 16. A regular view at the administrators table

An entry can be edited by clicking the corresponding button on the left – the view changes, and the user can enter new data and update, cancel, or delete the entry:

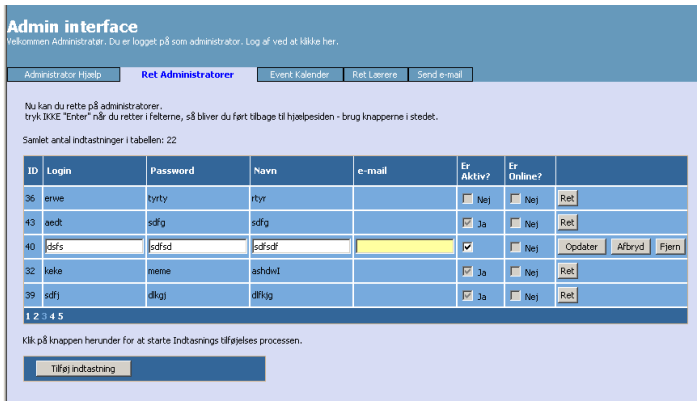


Figure 17. An edit expanded view of a row

Finally, a row can be added by expanding the menu by clicking the button on the bottom – a view is shown where data can be entered, and added to the table:

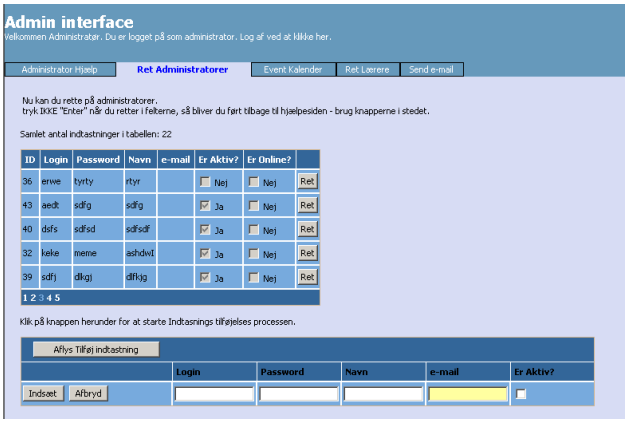


Figure 18. An expanded row addition menu

The third choice, Event Calendar, presents a slightly different view at the entries in the teacher [class] table – using the Calendar ASP .NET user control , the administrator can choose a date, and obtain an overview of entries scheduled for that date, ordered by starting times – so in that sense, it can facilitate easier administration of scheduling light programmes for the event:

Figure 19. A calendar view at the entries in he teacher/class table

The same type of table, with the associated functionality, is included here as well, although not all fields from the table are included. Finally, the Send Email option also appears in both perspectives:

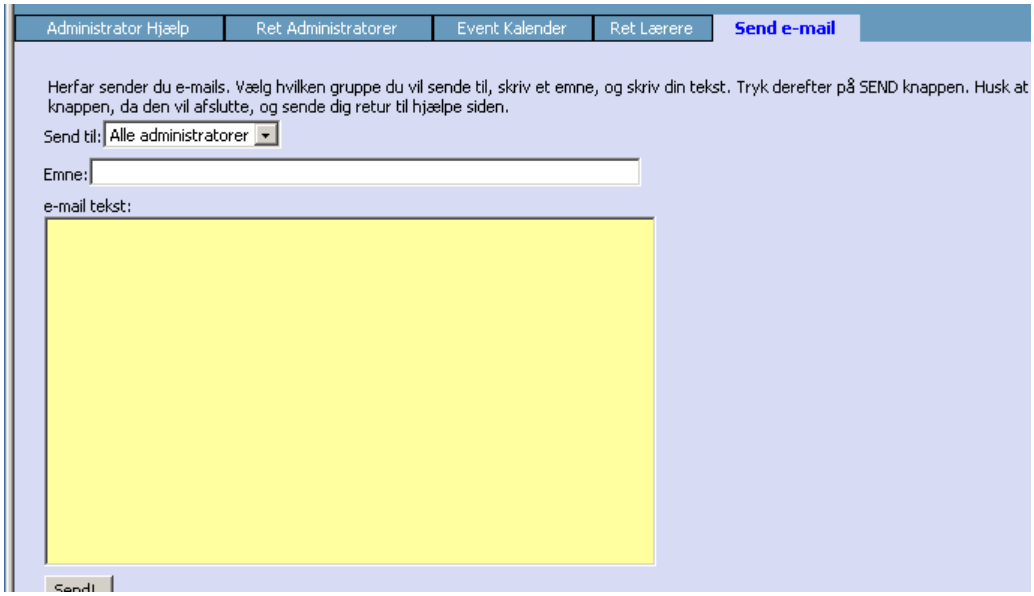


Figure 20. The Send Email screen

The Send Email screen is meant to facilitate easier communication between the user groups – especially between teachers and administrators, in case of needs for problem troubleshooting. It is a simple field that allows, for administrators, sending of a message either to all administrators, or to all teachers, by using the e-mail information stored in the e-mail column of the respective table.

For the teacher version, this is the initial screen displayed:

Figure 21. Initial teacher/class screen (help tab)

The teacher user identity has only 4 choices in the administration site:

1. Help
2. Class set-up
3. Edit class
4. Send e-mail

Help and Send e-mail are exactly the same layouts as for the administrator identity, with the corresponding information displayed. Edit class is a tabular representation of the entries in the UL3 table, belonging to that teacher/class, similar to the corresponding pages in the administrator perspective. Class set-up is slightly different, as it allows for an easy visual setup of groups by using a Flash plugin.

Let's remember that the teacher is in this case delegated to set up the groups in her own class – hence, at first login, there would be no groups whatsoever. This is indicated in both Class-set up and Edit class views:

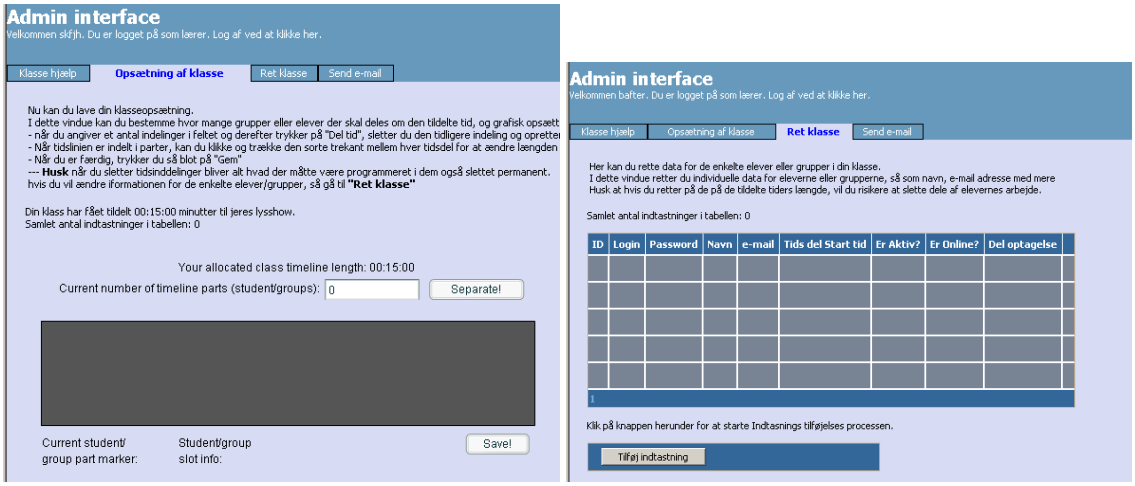


Figure 22. Class set-up (left) and Edit class (right) views, when the teacher does not have any groups set up

In this case, the teacher user is expected to enter the desired number of groups in the Class set-up screen, and click on the separate button, upon which equal separations are obtained visually on the timeline. The class set up features a graphical timeline, that corresponds to the default 15 minutes allocated to a class for a light show. The teacher user can thereupon modify the group part positions, and hence the lengths of the group parts as well. When the desired time setup is obtained, this can be saved in the database – after the process is finished, this will be indicated on the Edit class view as well, with entries with empty login, password and other information.

Figure 23.Class set-up (left) and Edit class (right) views, when the teacher after the initial group set-up

From this point on, the teacher can enter login and password information for his student groups, and once that is set up, the student groups are ready to log into the user system. As such, as the login password information comes from different sources, it is critical to implement a check for unique login/password combination for each student group involved. The Edit class screen also allows the teacher to control whether a given group has entered any light program cues in their allocated time slot.

This concludes the overview of the functionality of the current version of the administration site. As it is, it seems to fulfil its purpose, and since it uses POST form parameters instead of query strings, the security is improved as well. There is however one issue, which is critical for the user site as well, and that is determining the online status of the user. As the administration site attempts to address this issue, this is briefly discussed in the next section.

9.4 The user's online status and browser closing problem

User authorisation, as a simple check of a login and password in the database, may not be enough as a user identification measure – especially since eventually, it may be expected that accounts (logins and passwords) may be shared, especially between student groups (although they in particular are not users of the administration site). This is the main reason to implement an online check as well – that is, upon successful login, a marker is written in the database for that user – specifically, the is_online field is a part of the tables for that purpose, and setting this flag to 1 should indicate that the user is online. Ideally, that would prevent more than one user to login on the same account (since after a successful login and password, the program could check against this field as well, and deny login if that same user is already logged in), so the data integrity would be better maintained.

Thus, programming of a solution was attempted for the administration site. In general, this is not a difficult procedure – the only thing needed is simply additional if/then constructs denying login in case of successful authorisation of a user that is already online. If that is not the case, at successful login, the system would raise the is_online flag in the database for the respective user to 1 at login. When the user logs off, the system sets the flag back to 0, and the same user (or a different one with the same credentials) can login back again into the system. This is all fine in theory, up to

the point where the user log off is defined. This is actually handled quite easily by placing a log-off button. The problem is however, that most computer users, after they are done with a login based application, almost never click on “log-off” buttons for that purpose; instead, almost always they choose to close the web browser altogether, (by clicking, in most browsers, the X button in the corner of the title bar). Occasionally, a different address may be entered into the address bar as well.

The problem is, Internet communication on the web is implemented through the client-server model. That is, the user’s PC is the client, and it always initiates contact by sending a text request to the server (like, for instance when an Internet address is typed and enter pressed, or anytime a link or button is pressed). The servers read these requests, and prepare a text response (mostly, the HTML output for the requested web page) and send it back to the client. Beyond this exchange of information, there is no shared data between the client and the server; and the event of closing down the web browser window is considered a client side action. Hence, the server does not by default “know” when the user has closed the window, and hence cannot perform the action of lowering the is_online flag at logoff, which prevents further logins afterwards; with that, the entire online check for a unique user becomes problematic. Some sort of a client request must be initiated upon the closing of the browser window, and that event is unfortunately not handled by the currently adopted web technologies.

This seems to be a common problem in Internet application design; Internet searched revealed this to be a debated topic. Finally, the following comments were taken as guidelines in the current state of affairs:

- “*That said, the closing of the browser window does not raise any trappable events. The window_onclose event does not fire when the browser is closed… There is no solution. You simply can not do it. That is why a stateless "web application" is not the proper solution for every situation.* [13]”
- “*I want to detect, if user has close the browser, (after logging to my site), that user has closed the browser ...*
  Point is that there is no 100% way to guarantee that. You can try having js scripts to send something to the server when browser gets closed or something else happens, but if user has disabled js, you are out of luck…
  You should only respond on the session timing out, or the button being clicked - other than that assume that your client is still connected. [14]”

The current version of the administration site raises the is_online flag for the respective user upon successful login. When the user is done, she is expected to click on the log-off button, which loads a page, that reads through the session variables, finds the user ID, and resets it's is_online flag back to zero. For the case when this doesn't happen, and the user chooses to simply close, another attempt was undertaken. When the administration site is loaded at start, a child JavaScript pop-up window is spawned, whose task is to listen (periodically check) whether the parent opener window (the site) still exists:

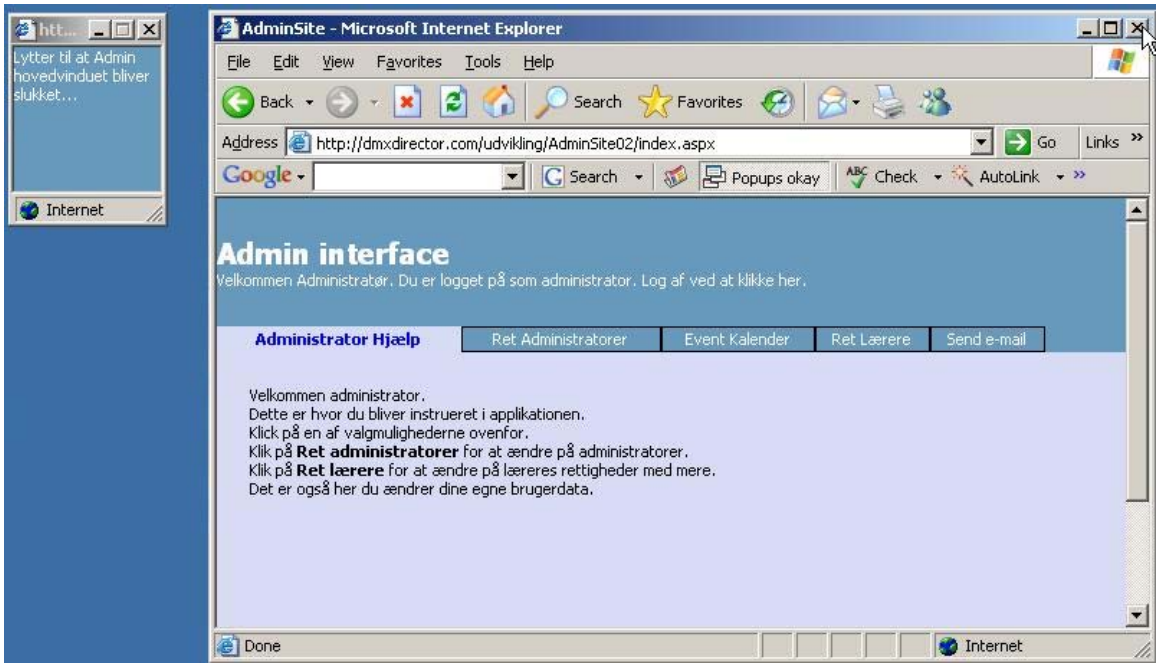


Figure 24. Screenshot of the administration site and the listener pop-up window

However, this is yet another problem – both JavaScript and pop-ups have for a long time been considered evil on the Net, and most safety-conscious users choose to block them. This trend has been reflected in the latest service pack for the still most widely used browser, IE, which when encountering the administration site for the first time, tends to respond with the, now classic, line:

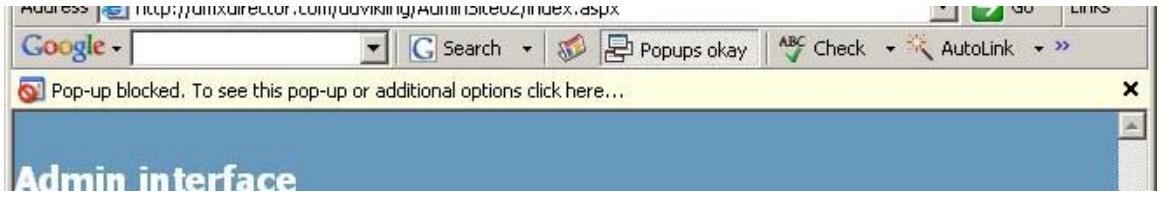


Figure 25. The standard answer of a service-packed IE to pop-ups

Certainly, instructed users will give permission to the popup window, but this is not yet the end of the story. Once the user closes the main browser window, the child listener must initiate a request to the server, meaning it must load a page(in particular, the page resetting the is_online flag) somewhere – and in the current version, it tries to open *yet another* pop-up, and as soon as it's done, a dialog box is displayed, which when clicked closes both pop-ups. Thus, unless pop-ups are allowed to the entire domain (in this case, dmxdirector.com) from the browser, the following situation occurs upon window close:

Figure 26. A pop-up blocked from the pop-up upon administration site window close, preventing proper log-off

In this case, in spite of the notification given to the user that she is logged off the system, that is in fact untrue – since the inactive page hasn't been loaded in the second pop-up, the reset of the is_online flag is never undertaken. With site-wide permission, the correct situation "upon close" looks like this:

Figure 27. Situation upon proper log-off upon main window close, with two pop-up windows

At the present time, this problematic behaviour with the pop-ups is the only serious issue with the administration site – as often accounts are left with "hanging" is_online flags set to 1, preventing future logins, which demands a reset of these switches directly from the database. That is why a similar check for unique logged user is not yet attempted for the user site. This issue is however, not as problematic as it seems, at least in the case of the administration site – it is standard practise that 20 minutes is taken as the default session timeout period. This being said, ASP .NET allows custom coding of a session end event – whether it is raised by the user or by a timeout; and code for closing the is_online flag can be inserted in this global method. This is in fact, already done, and tested fine locally on a development machine – however, the current hosting provider of dmxdirector.com does not allow direct access to the IIS server, which is needed to activate the programmed session end event. Hence, this cannot be demonstrated on the online version of the prototype, and thus this complicated solution with JavaScript.

That being said, smarter possibilities to solve this problem are possible, and are high on the list of priorities. Again, this is not so much of a problem for the user site, because session timeout will do – when the user operates the administration site, each button press is a call to the server, which resets the session timeout timer for the given user. However, for the user site, there is a different story – there are, in general, only two calls to the server – once to authorise and read the class light program, and once, chosen by the user, to save a program in the online database. All other user actions in the user site are handled locally by the Virtools application; calls to the server are never made. As sequence programming, especially in the current context of simulated light design, is not a trivial job, it can be expected that times in the range of hours can easily be expected when working on the user site, from initial login, to the point when the user decides to save a light program up to the database. So, care needs to be taken with a possible timeout solution, and eventually, a JavaScript child listener as the current one may prove being the last viable option.

10 3D user interface development

With the user and database set-up in place, it is easier to commence the work on the 3D user interface. Considering the previous discussion, there are three main points to consider about the 3D interface:

- Distribution and deployment of the software over the Internet, as hassle-free as possible, since the primary end-users are pupils.
- Modelling in 3D, realistic to a certain degree, of a given environment.
- Modelling the specifics of light show programming – both in 3D (as the expected effects of the projectors), and as an interaction and software engine that allows light programming to the primary end-users.

The most important aspects of deployment were discussed during the process of platform choice, which resulted with Virtools being chosen as the development platform. As for the other two points, the development team looked at several software packages for inspiration.

The most important reference comes again from the world of professional light design. In modern light design, most of the design work is performed on a so-called light console, which acts as a central computer where the light show program is stored and reproduced from; all the projectors involved in a show are then connected to the light console, and controlled by it via DMX. One of the most advanced such consoles is GrandMA, produced by the German company MA Lighting. Besides the console itself, this company on its site also offers supporting software, among which the visualizer software package GrandMA 3D [15].

Figure 28. Image of a GrandMA console (left) and a stage design on GrandMA 3D (right) (From Ref.[16])

This software package is actually free for download, however, keep in mind that this is an application aimed at users of the GrandMA console, as closer inspection at the complete user interface reveals:

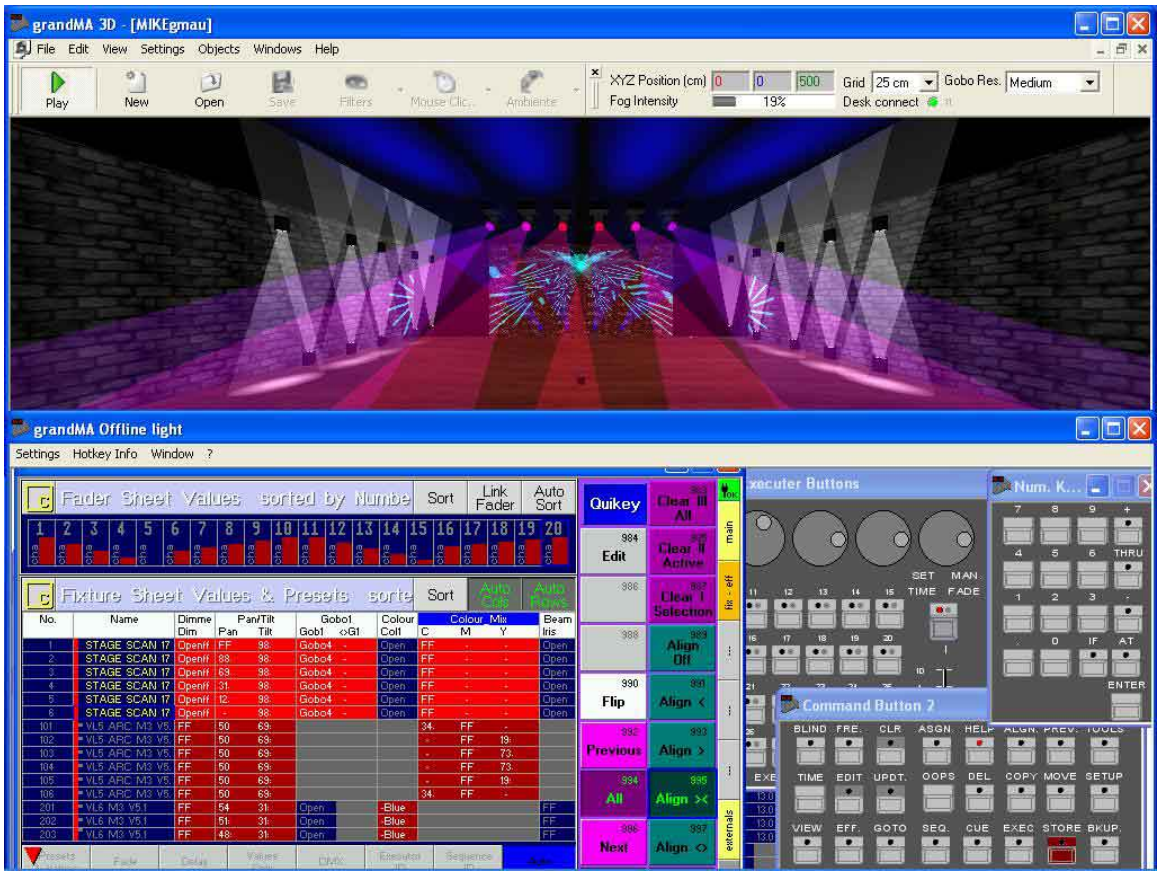


Figure 29. A closer look at GrandMA 3D and GrandMA Offline (Ref.[17])

The primary intent of GrandMA 3D is in fact to allow testing of light programs stored on a console – a console can be connected to a PC, GrandMA 3D can be fed with a corresponding 3D model, and the console can then reproduce the light program with GrandMA 3D as recipient, instead of actual projectors. To facilitate light programming on the PC without the console, MA Lighting provides two other software packages, that with a different degree simulate the functions and operation of a real console on a PC – GrandMA offline and GrandMA onPC: "*One version, called grandMA offline, mimics in software all the hardware of the grandMA or grandMA light consoles. The other is not designed to mimic the grandMA hardware, but rather, it is designed to be a highly flexible and easy to use version of the grandMA console on a PC* [18]". In any case, as Figure 29 shows, these applications emulate the interface of the console – which is definitely too complex for our primary target group.

Thus, in spite of the impressive hardware interfacing capabilities, GrandMA 3D serves only as an inspiration in relation to the degree of accuracy that is expected of the 3D part of modelling of a light show. The possibility of rendering gobos is there, and the light cones a projector creates are visible as well, maybe even exaggerated (though this might be seen as a tool to assist the light designer while working). Apart from that, even without rendering impressive environment textures, the overall effect and impression of a light show within a given environment is easily perceivable through the GrandMA 3D interface – and a similar degree of quality and immersion was set as a goal for the user web interface as well. In regards to camera interaction, it is more similar to CAD applications, than for instance a game.

In any case, solving an interaction interface that will facilitate easy entering of a light program needs to refer back to sequencing / editing applications, nowadays simply known as media production tools, that exist for nearly any kind of media that can be processed on a PC. Classic examples of such editing applications include Steinberg Cubase [19] for MIDI/Audio processing, Adobe Premiere for video processing, or Macromedia Flash [21] for 2D vector animation on the web. Regardless of what type of application we talk about, the user creates a composition (or enters a program) by entering time-based data (such data could be generically referred to as events), different depending on the type of media. Therefore, the user is always presented with a related GUI toolset, which usually includes a timeline – a visual (usually horizontal) frame representing the time, a time marker, indicating the current instant of time in relation to the timeline, tracks for different organisation of program data, and some sort of visual indication of entered events – as well as a corresponding visual toolset to allow inspection of the contents of a given event.

Figure 30. The sequencer related GUI of Steinberg Cubase

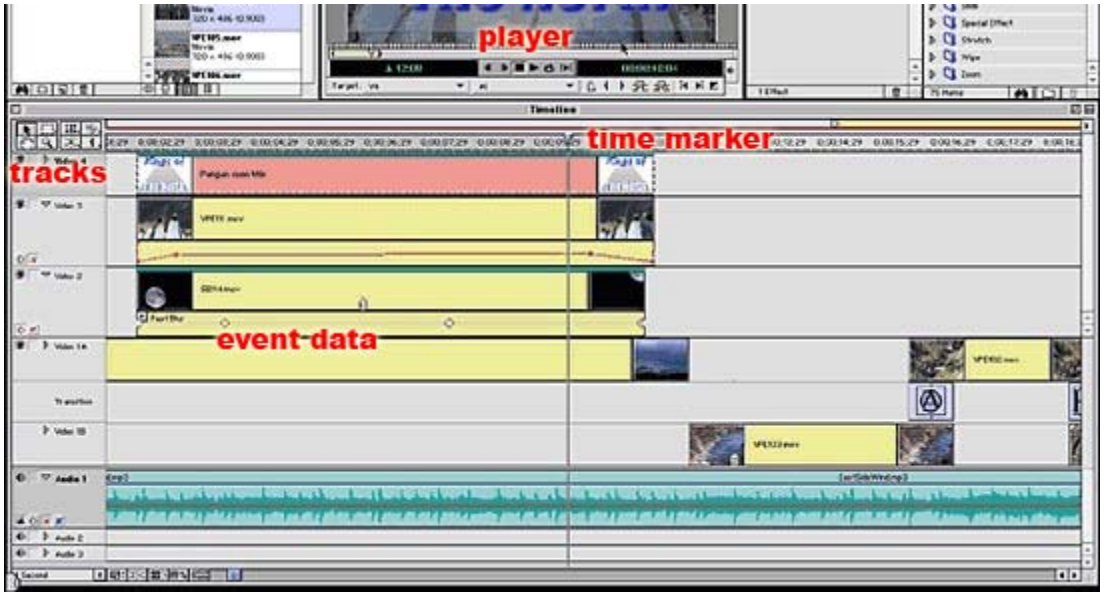


Figure 31. The sequencer related GUI of Adobe Premiere

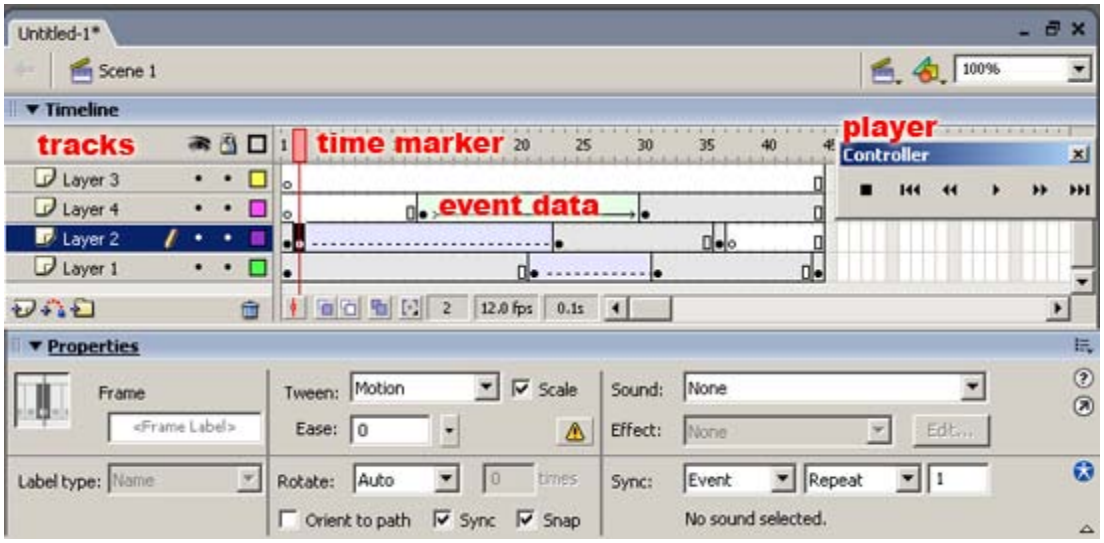


Figure 32. The sequencer related GUI of Macromedia Flash

One general conclusion for these interfaces is, besides having common points like organisation in tracks, player controls, graphic event indication and a current player time marker, they also contain vast amounts of visual information; as such this can certainly work overwhelming on the target group, and a simpler implementation of a similar GUI is needed. One example of such a simplified sequencer GUI comes from a real-time video titling and effects application called BluffTitler[22].

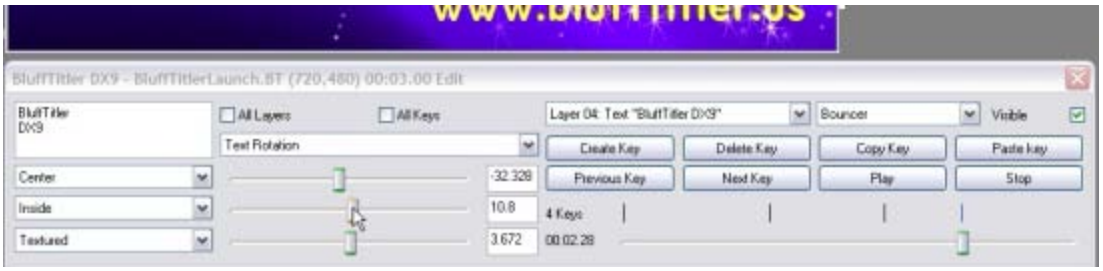


Figure 33. The sequencer related GUI of BluffTitler

The GUI of BluffTitler is much simpler – rendering each event (here called key) simply as a vertical line; no track information is displayed. In that sense it is much easier to use, as there is essentially not much of a visual language to learn.

Eventually, these findings formulated the initial goals of the user application – to offer a level of 3D modelling that approximates that of light design software such as GrandMA 3D, and at the same time, implement an event programming/sequencer engine with a simplified accompanying GUI. In addition, the two step action built into the initial prototype – click on a projector to select it, and click on a 2D interface to change a parameter, was taken as the base action for inserting a cue. As the task of light programming might still end up being overwhelming for the age of the expected primary users, it was agreed that in all other aspects, the user site application should adopt, as much as possible, the look and feel of PC video games – as it was expected that most in the target group would have had previous experience with it. The term "cue" as it is used in the GrandMA software was internally adopted instead of "event" as used so far (referring to a data structure describing the state of a given projector at a given time), and hence the event sequencer engine is referred to as cue programming engine in this document.

In the meantime, specifications about the amount (in total 64) and type of projectors to be used on the final event, were given as well:

1–20: Martin MAC 2000 Profile II (colormix, gobos, spread of 10° - 26°)
21– 28: Citybeam (1800 and 1200) IP54 (colormix, spread of 8° - 11° • 15° - 20°)
29–40: Martin Exterior Mac 600 (colormix, spread of 12° • 18-25° • 22-38° • 65° • 100°)
41–64: Alkalite Tracpod/TP-81 LED (colormix)

Some of these projectors have rotating gobos and some do not; some actually are intended for placement where no light cone is visible either. This must be taken account for in the application. This projector specification also helped to determine the number of parameters that should be taken into account in the cue programming engine. In its current state, the cue engine in the user site prototype supports the following parameters of a projector: colour, intensity (dimmer), gobo and gobo rotation speed. Static parameters of a projector, such as the model, or the light spread, can be manipulated through an external text file, but not as part of the cue sequencer engine. Although the DMX standard and the projectors used, support other controllable parameters as well – such as movement of the light beam, this are not used in the application; for one reason, they would increase the complexity and stability of the entire application, and for another, additional set of parameters might make it even more overwhelming for use by teenagers.

The cue sequencing aspect raises an important issue: the frame-rendering loop (the one calculating the 3D output at each frame of the application) must be kept independent from the player time calculations. As cue programming is the intent of the application, playback of the entered composition is a necessity – and manipulating playback as well. This implies, that when the player is paused, so should the 3D indication of the state at that moment in time freeze as well; this means, that although the player timing loop is stopped, the frame rendering loop isn't – it simply renders the same state over again. Where in the case of editing 2D audio and video this may be immediately implied, here it is of slightly greater importance, since an independent player timing loop means also that other calculations can be made independent as well – so, for instance, camera navigation can work independently, regardless of whether the internal cue player is playing or not.

As the distinction between the frame rendering loop and the player time loop becomes a necessity, one can expect that eventually the execution of the surrounding code will take up the majority of resources of the application – and this motivated the non-real time design of the interaction of the user site with the database. In fact, there are only two calls to the database made from the user site – once to read, and once to store a light program; all other aspects of the user site are basically the same as a traditional, self-contained monolithic program, rendering the state of the 3D environment based on the initial program received from the server, and subsequent user input. A networking protocol that would allow some sort of a real-time multi-

user collaborative environment, similar to the massive multiplayer online games, would demand a network loop to be threaded with the needed frame rendering and player timing loop; and this could be a rather difficult task that might eventually take its toll on the rest of the application performance. Thus, the user site application can be generally seen as a non real-time collaborative environment, although at each login, the users should receive the latest uploaded light program in the database; however, the frequency of updates is more similar to an online posting forum (hours).

All of these considerations were eventually integrated in the engine, with varying amounts of detail. The most important aspects are discussed in more detail in the following three chapters.

10.1 Interaction and graphic user interface

The key features of the interaction and graphic user interface will be presented through a short walkthrough, illustrated by screenshots based on the latest version, BrugerSite20A, which contains projector models, positioned accordingly to the light design. Upon loading of the page, the Virtools plugin checks for player version, and auto-updates accordingly. With the proper Virtools plugin in place, the compiled Virtools player file (.vmo) begins being downloaded into the browser; during this, the standard Virtools download message is shown. The .vmo file is approximately 3.5 MB in size, which adds up to some five minutes of download time on a 64 Kbps Internet line. However, once, and as long as the vmo file is in the browser cache, the download process does not repeat; instead, the cached file is used. The application contains only two scenes, a login scene and a main scene.

When the application is first loaded, the user is presented with a login screen:

Figure 34. Screenshot of the login screen

Figure 34 shows the login screen, with login and password fields (1), a login button (2) and a server response text field (3). When presented with this screen, the user is expected to enter a login and password combination, and press on the login button. This initiates a request for login authorisation to the server, and the stages of this process (as it is asynchronous, and the exact time of server response is unknown), are followed through the server response field. Eventual problems with the server connectivity would also be displayed in this field, so it has a troubleshooting purpose as well.

If the user gets authorised, a user class is automatically determined as well. This interface recognizes three classes of users: teachers [class], students [groups] and guests, which are distinguished by minor differences in functionality and the user interface. Upon authorisation, if the user is a teacher or a student, the next step is yet another trip to the database, this time loading the entire light program, saved up to that point, for that particular class – the guest login is not connected to this part of the system, so in that case, a loading process from the database is skipped.

Once eventual light programs are loaded from the database, the login scene ends, the main scene is loaded, and the initialisation (hereafter called init) phase begins.

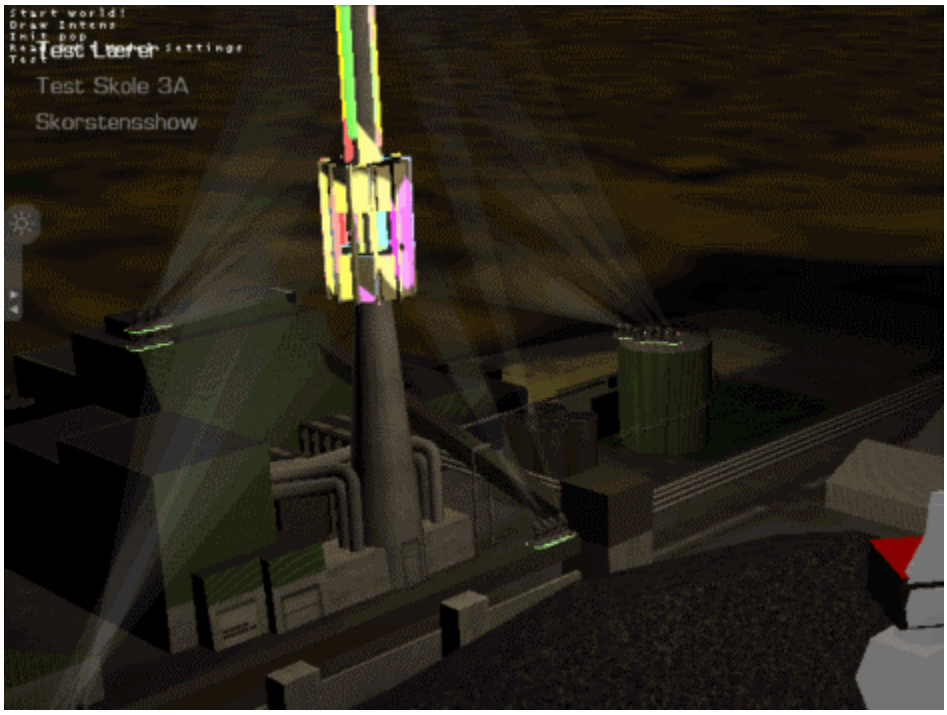


Figure 35. The rendered world during the init phase

The init phase spawns and sets up the projectors within the world, spawns their light cones, calculates which surfaces are to render a projector light and parses the loaded light program if any. At the same time, the rest of the user interface is initialised as well. That is why this init phase can last up to several minutes, even on newer Pentiums. During the init phase, the interface engine does not work in full, and one has to wait for this phase to finish before using the application. The init phase is at the moment indicated by all projectors being surrounded by static selector objects, white light cones, and reflected gobos appearing on the surfaces as they are calculated. In the future versions, a screen might be implemented to cover the display during the init phase, giving a clearer message to the user.

Once the init phase is finished, the selectors disappear from the projectors, the application's interface is ready to use, and the initial cue from the loaded light program is displayed in the world, if the user is a teacher or a student (a guest gets all projector set to white and no gobos as initial cue, as there is no light program):

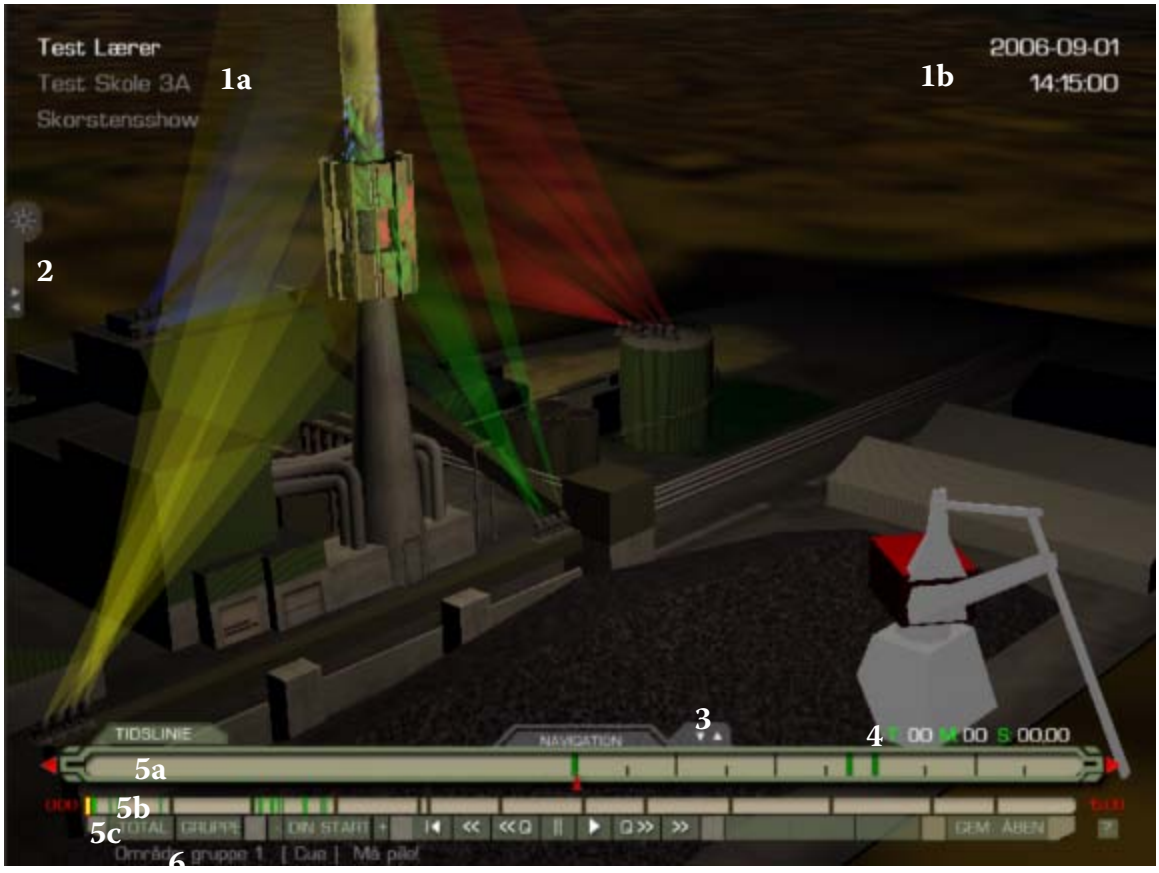


Figure 36. The initialized user interface - right after the init phase, displaying the initial cue of a light program

The state presented on Figure 36 renders almost all main elements of the graphic user interface (but one, discussed later), which are:

1) Personalized title texts: information about
    i. Name, school and class, and the title “Skorstenshow” (1a)
    ii. Date and time of the planned reproduction of the light programming within the event (1b)
2) Menu tab for the projector selection (hereafter referred to as “left”) panel
3) Menu tab for the navigation panel
4) Current player time indicator
5) The graphic timeline, composed of a
    i. Detail timeline, which always renders a fixed time span of 10 seconds, centered around the current player time, which is fixed in the middle – and a detail resolution view of the cues (which are clickable). (5a)
    ii. Group timeline, which shows the entire timespan of the class allocated slot, or a timespan of a given group, depending on the zoom level – and a rough resolution view of cues (not clickable) (5b)
    iii. Player controls – buttons for manipulation of current player time, controlling the zoom level of the group timeline and cue and file operations (help button slightly on the right). (5c)
6) Supporting texts: textual indication of current group area, whether the current time instant has a cue or not, and whether it is allowed to change it, and number of selected cues.

The user interface provides a first-person 3D view that fills out the entire viewing window; most of the corresponding user interface is 2D graphics, organised in menus and compacted in tabs. This represents a direct intent to make the interface look like a contemporary PC game, and to inherit a look and feel that should be familiar to our primary target group. The viewing window, within the starting HTML page, is a fixed 800 x 600 pixels rectangle; however, the application is meant to be viewed in full screen mode, which is performed by right clicking on the Virtools plugin in the webpage, an making the appropriate choice. The right menu also allows

different resolutions to be chosen for the full screen display – which presents a special problem for the chosen graphic interface.

The different user classes are mostly distinguished by the personalized title texts, and features mostly related to the timeline. For instance, the guest account is not connected to a database program at all, and that is reflected by a non-separated, blue group timeline:

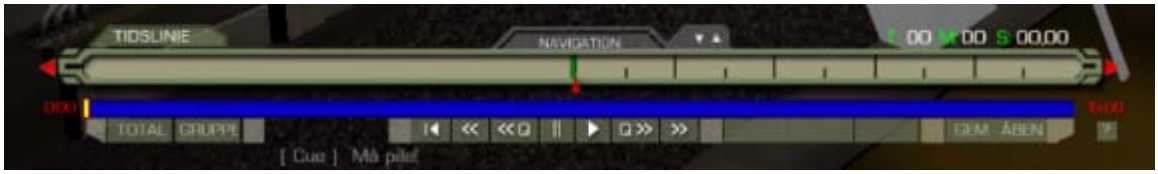


Figure 37. The timeline for a guest account

In addition, the buttons for group start navigation are missing, as there are no groups defined for a guest account (however, this visually breaks the design, so it should render disabled buttons instead). Correspondingly, there is no option for saving to the database for a guest account – and the guest starts only with the initial cue, which sets the starting values for all projectors in the world. As the guest account is not accounted for in the database, it is authorized within the Virtools plugin, and the login/password combination is hard-coded as gaest / gaest.

As the teacher and student logins are connected to groups and programs on the database, the display for the group timeline is slightly different. First, for a teacher that has no groups set up yet, a message is displayed:

Figure 38. Message that no groups are set up for a teacher account

The application then gets locked in this case, until the teacher logs back in the administration site and sets up groups. Once groups are set up, both the teacher and the students can log into the interface. In this case, after init, the teacher gets a timeline bright throughout, with the group delimitation and visual indication of cues saved up to that point:

Figure 39. The timeline of a teacher account

This is to indicate that the teacher has the permission to edit cues throughout the entire class allocated timeslot. In contrast, the student group timeline, after init looks slightly different:

Figure 40. The timeline of a student account

That is, only the time slot belonging to the group is shown as bright, indicating that the user has the permission to edit cues only there – the slots belonging to other slots are rendered with a dark shade. However, the student user does have permission to view the entire class light show.

These are the main indications of differences between user classes in the user interface. Apart from that, the user interface is the same for all users, and serves, in general, to allow the performing of three main tasks:

- Camera navigation process
- Projector selection and edit process
- Player time / cue selection and edit process

The following subsections briefly illustrate each of these tasks, and the corresponding functions within the user interface.

10.1.1 Camera navigation

Interaction with the first person camera can happen either through the navigation panel (clicking) or by using the keyboard keys in two navigation modes, internally called “pilot” and “orbit”.

Figure 41. The navigation panel in collapsed (left) and expanded (right) state

When the navigation panel is expanded, it indicates the current navigation mode, which first and foremost relates to the usage of keyboard keys. The navigation mode can be changed either by clicking the orbit or pilot button on the panel, or by clicking the N button on the keyboards (switches from one to the other). The zoom in/out buttons on the left of the panel, and the orbit buttons to the right, are linked to the corresponding key functions of the orbit navigation mode.

The main difference between the two navigation modes is in how the camera is handled. In pilot mode, the camera is free and emulates a free-floating vehicle – one can elevate or rotate the camera freely – which sometimes can result with a rather messy view. On the other hand, the orbit mode always focuses the view on the center of the model, and the keys result with the camera orbiting around the tower. In both cases, the keyboard keys chosen are based around the common WASD setup of keys for camera movement, where "W/S control forward and backward and A/D control strafing left and right. These mimic the arrow keys, which are also commonly used for movement[23]". This is another intention to simulate the interface of a contemporary game as much as possible, and inherit the familiarity the users might have. The complete list of keyboard commands is given in Appendix A.

The camera interaction is mostly coded using internal Virtools blocks, like Switch on Key BB that waits for certain keys to be activated; the presses on the panel button, like most buttons in the interface, are captured by Push Button BB; and then basic 3D transformation blocks like Rotate BB, Dolly BB, Translate BB with the main camera as the target entity compose the most of the pilot mode. The orbit mode initially used the Virtools object Keyboard Camera Orbit BB, but for certain angles the camera "flipped" – a custm solution was attempted, built on the pilot mode and the Look At BB and Keep At Constant Distance BB, simulating orbiting movement. Since it also suffered from "flipping" a limiting condition was attempted, however unsuccessfully - so now there is a slight bug in the application: for certain angles, when using the orbit up or down commands, the engine can lock and stop responding to these camera commands – in which case the user can reset the camera with space, or switch to the pilot mode.

In addition to the basic functions, there are a couple of other camera related functions that are situated on the left panel.

The left panel can be accessed by clicking the left panel tab:

Figure 42. Collapsed (left) and expanded (right) state of the left panel

The bottom part of this panel has a camera section. By clicking one of the position buttons, the camera will be instantly placed at one of the three locations, where the cameras/webcams recording the event will be placed. In addition to this, three fly-by tours are available, built upon three different 3D Curves in Virtools, and usage of the Curve Follow BB in Virtools. One more aspect of camera interaction is that all of the objects composing the world are placed in a Virtools Group, and as a child script of the camera, the Object Slider BB is used to check for collision of the camera (within a given radius) with these objects. As this amounts to quite a few objects (608 to be exact), this could be potential performance handicap for the application – still, it allows the sensation that the world is physical, and that the user "bumps" into objects, which is another thing expected of 3D games (although it can sometimes result with a geometry problem for the camera, in which the space key is unavoidable).

These represent the most important aspects of camera interaction and 3D navigation within the world. The top of the left panel, introduced previously, is related to projector selection (not yet implemented in this version of the application), which as a user process is discussed in the next subsection.

10.1.2 Projector selection and edit

As discussed previously, the base action for inserting a cue was taken to be: click on a projector to select it, and click on a 2D interface to change a parameter. Thus, selection of a projector by clicking is the primary means of projector interaction accounted for in this version of the user site. It chiefly relies on the 2D Picking BB in Virtools to determine whether the mouse pointer of the user is hovered over a projector. If that is the case, a left click on the mouse selects the projector. Multiple projector selections are handled by holding shift on the keyboard, and clicking on a different projector. An "empty" click cancels a projector selection (deselects) – and an empty click in this context, is defined as a click anywhere in the 3D view, which is not on an area covered by a 2D interface element, and is not a projector model.

To follow the different stages in this process, the following indication is implemented:

- Mouse hover over a projector – 3D model, representing a green rotating circular curve appears, surrounding the projector (hereafter called "selector") – lasts as long as the hover
- Projector selected – a selector representing a white curve appears, along with text in 3D, aligned towards the user. At the same time, the projector edit panel (hereafter called "right") appears, either in collapsed or expanded state.

Figure 43. A projector, selected in the horizon, and the corresponding indication

The right panel has not been introduced so far, since after the init phase, there is no projector selection by default (the appearance of the selectors around the projectors during the init phase is due to an initial reset loop; each projector must be selected and deselected once by the application for the engine to work properly. Note that sometimes the application "forgets" to deselect the projectors after the init phase – nothing an empty click won't solve). As Figure 43 shows, the indication through the appearance of the right panel is quite visible – however, for distant projectors, the 3D text and the rotating selector are barely visible. Another problem with the 3D indication can be detected for closer ranges to projectors, as shown on the following image:

Figure 44. Two selected projectors, and a third hovered in the middle, and the corresponding indication

So, as the fonts on the 3D text scale with distance (as an actual text panel in a 3D world would), for multiple selections that leads to a graphic conflict and low readability – hence a redesign of the 3D part of projector selection indication may be necessary.

As soon as a selection is made, the user can read out the current value of the parameters of the selection (if they are equal) on the right panel. Afterwards, by making any change in the right panel, the user either inserts a cue, if none is present at the current time location; or modifies an existing cue – the display on the timeline after such an action is updated accordingly. Let's also mention that the right panel also functions as a tab, and can be permanently collapsed:

Figure 45. Collapsed (left) and expanded (right) state of the right panel

The right panel functions only if it is expanded, and in that state, it features the following functional areas:

1. Day – night slider
2. Gobo selection area
3. Colormix window (non clickable, indication only)
4. Color slider (2D)
5. Dimmer (light intensity) slider
6. Gobo rotation speed slider

The four parameters that can be edited for a projector are regulated through their corresponding areas. The gobo selection area (2) features all gobos available for the show as clickable buttons. The first button (in the upper left corner) is used when no gobo is applied to the projector. A red outline shows the gobo of a selected projector (or multiple projectors, if they all have the same gobo) at a given moment in player time. Clicking on a button sets that gobo pattern for the selected projector(s) for that instance of player time in a cue. As not all projectors planned for the show feature gobos, this section should be disabled (visually as well) when such a projector is selected. That is not yet implemented, as all 20 projectors in the current version emulate the Martin MAC2000 projectors, which do have gobos. The story is similar for the gobo rotation speed slider (6) – it indicates, and allows setting, of the gobo rotation speed parameter, for a given projector selection, at a given moment in time. A slider cursor located in the middle means no rotation, to the right – clockwise, and to the left – counterclockwise gobo rotation, with the speed increasing as the ends of the slider are approached.

In the world of professional light design, color of a projector is usually specified through two parameters – the color of the projector light (as RGB), and its intensity (as an 8 bit integer) that is usually known as a "dimmer" parameter of a projector. This is reflected in most interfaces dealing with light programming through having two separate interface objects for these parameters – this application is no different, and it offers a 2D color slider (4) and dimmer slider (5) – the colormix window (6) simply shows the final color output as combination of the values of the color and dimmer slider. Cursors on both sliders show the current color and dimmer state of a selected projector (or multiple projectors, if they all have the same gobo) at a given moment in player time.

It's important to note that all these projector parameter sliders show the current values for a selection – as such, if the user selects a projector, keeps the right panel expanded, and plays the light program, the user can follow the interpolation of the projector data "real-time" – that is, the panel will indicate the selection parameters state according to a given player time, and a given light program, so this also works for fast forwarding and rewinding as well. Since the same panel is used for both single and multiple projector selections, there can be problems related to display of multiple selection data. As an initial solution, there is additional indication built into the right panel – specifically in the slider cursors.

For a single projector selection, if the player time is on a cue that also includes the selected projector, the 1D cursors are bright, and a cross is used as the color 2D slider cursor. If the player time is not on a cue, then the values of the projector shown in the world and in the right panel are interpolated; hence X is used as a 2D slider indicator, and the 1D slider indicators become transparent. The same language is used for multiple projector selection, however, only when the corresponding parameters of all projectors are equal – otherwise, the other state is shown (in particular, if two projectors do not have the same color at the same time, X is shown in the colormix window).

Figure 46. Indicators for: single selection, not on cue/interpolated (transparent - left); single selection, on own cue (full - middle); multiple selection, different projector colors (none - right)

This planned behavior is still buggy at the moment - for instance, for a single projector selection, bright cursors are shown even on a cue that does not include the selected projector, contrary to the design - so a better solution needs to be implemented. Let's also note that using the panel to show interpolated values while the player is playing, is another possible performance consumer in the application.

Apart from that, the editing process is quite easy to conduct – a user makes a selection, clicks on the desired value for a given parameter, and a cue is inserted or modified (for multiple selected projectors, all projectors inherit the inserted value). Preliminary tests have been done on this part of the engine; some further testing may be needed to identify potential failures in the implemented algorithm.

There is one more aspect to be considered - the projectors are placed at different "geographic" locations in the world, and sometimes a certain amount of "driving" is needed until a view that allows selection of the needed projectors appears. As this is a potential threat to usability (since the user might lose the original view and the idea, while she selects all projectors to apply a value to), the top section of the left panel shown on Figure 42 is planned – although, as mentioned, it is not yet implemented. The idea is that all projectors are organized into groups, according to location – and each group features two subgroups, A and B. By clicking these buttons, the user would be able to select all projectors within a group/subgroup, and be in a position to quickly apply values to groups of projectors, while at the same time having a wide view, which would make the projectors impossible to select with the mouse.

Finally, there is one more feature residing in the right panel, which actually is not related to a projector selection, but instead to a global feature – and that is the day/night slider – whose purpose is to change the ambient light conditions in the world (currently, it only changes the background sky texture filtering – the background sky effect is obtained by using Sky Around BB in Virtools, and the slider simply changes the gray amount of the Filtering Color parameter of this BB). The feature is useful; as it would allow the students to visualize different ambiance conditions when their show is being performed live at the event.

Figure 47. Virtual day (left) and virtual night (right) in a virtual world

This concludes the discussion about projector selection and editing, and the accompanying right panel.

10.1.3 Player time / cue selection and edit

The part of the interaction interface that deals with cue sequencing, selection, and editing, as well as the control of player time, is the aforementioned timeline. Let's take a more detailed look at the timeline:

Figure 48. A view of the timeline with all buttons shown

In the above image, the "t" symbol refers to a current player time marker. There are three different visual interpretations of the current player time. On the top right, there is a textual indication of the current player time. For the detail timeline, the current time marker is the middle of the timeline and it is static – its position is fixed; the detail timeline always displays 10 seconds window around the current player time – from 5 seconds into the past to 5 seconds into the future, located at the ends of the timeline. Hence, the background of the detail timeline changes, and animates as the player time flows (if that is the case) – the static red marker in the middle indicates the current player time moment. For the group timeline, the current time marker is a draggable button, which indicates the current player time in relation to the bounds of the group timeline (the entire 15 minutes for a total zoom, or the group boundaries for a group zoom). In the case of the group timeline, the background is static, and it is the marker that moves/animates as player time flows. The three current time displays are always synchronised.

A careful reader may notice that this is not exactly the kind of a timeline found on common sequencing software, as briefly discussed at the start of the chapter. This solution was partly stimulated by a very limited user research: as some of the employees at BB&S have children with the approximate age as the primary target group, earlier versions of the prototype were demonstrated to them, and their response recorded. The early version of the interface featured a very simplified timeline, with zoom in/zoom out buttons found on all similar applications. Only with zoom levels greater than 160 times, the application would start rendering cues as draggable buttons – otherwise, only a texture was drawn, with no immediate

interaction allowed with cues. It was found that such a solution with “free” zoom may confuse the users of that age – hence this solution (implemented after the major redesign) includes both an overview of the entire class-allocated timeslot (through the group timeline) and a fixed zoom view, large enough so the quantization steps are visible, and implementing button interaction on the events makes sense. In that way, the user should always have an overview both of “the big picture” as well as the fine detail where cues are manipulated.

Talking about quantization steps, the system is limited to a quantization step of 250 ms; the detail timeline renders whole seconds as a full height black line, and 500 ms steps as half height black line, as a visual assistance to determining the quantization steps. As mentioned before, the detail timeline allows interaction with the cues – selection by clicking. Apart from that, the rest of the timeline also features clickable functions, most notably the player buttons. In all, the functions performed by the timeline can be summarized as (the indexes refer to Figure 48):

- Group zoom level functions (total [1] and group [2] zoom)
- Player time control functions (quantization step backward [s1], forward [s2], previous group start [3], your group start [4], next group start [5], rewind [6], fast backward [7], previous cue [8], play [9], next cue [10], fast forward [11]) + the draggable current time marker on the group timeline [t]
- Cue editing functions (delete [12], copy [13], paste [14] - in conjunction with cue selection/interaction on the detail timeline)
- File (program archivation) functions (save [16] and load [17])

Additionally, the help button [18] can be seen as a category for itself. Let’s briefly take a look at each of these groups.

- **Group zoom level functions**

These functions are related to time slot displayed on the group timeline. The total zoom button sets the display to the entire class 15-minute timeslot:

Figure 49. Total zoom displayed on the group timaline

The numbers in red indicate the time positions of the bounds of the group timeline, relative to the start of the class timeslot. Hence, for a total zoom, they always display "0:00" for the start and "15:00" for the end. When total zoom is displayed, the group timeline graphically shows the boundaries between group slots as dark lines, and a rough-resolution preview of the light program cues as green lines; none of these are interactive – only the current time marker on the group timeline is draggable.

When the group zoom is clicked, the display on the group timeline zooms to the according scale, to display the group slot where the current player time currently is:

Figure 50. Group zoom displayed on the group timeline

The light program cues preview is updated accordingly, and as expected, the boundaries of that group are stretched to the ends of the group timeline – the position of the current player time marker on the group timeline is updated as well, as are the time bounds texts in red. It's important to note that when the current player time on goes outside of the group bounds, the group zoom level is automatically reset to total. Also, in the case of students, the background colour is preserved as well, indicating ownership of a group slot - Figure 50 shows how a student account would see a group slot not of its own; for the own slot, the background colour is bright:

Figure 51. Total (top) and group (bottom) view of a slot on the group timeline, that belongs to the logged account

- **Player time control functions**

These represent one of the key functions in the application. It should be mentioned that all of these functions actually trigger the main player function – which simply calculates a new time moment (relative to the beginning of the class time slot) to be the current player time. This calculation of a new time moment, however triggers the entire projector update engine – that is, for each projector, previous and next cues relative to the new current player time are found, the values of the parameters are read and interpolated for the new current time, and these values are applied to the projector light, which results with an updated state the next time a frame is rendered – so the unfolding of a light program in time can be seen as an animation, as it would unfold in reality. However, the entire algorithm is quite complex, and a decrease in performance when the player is "playing" – that is, when "new" time moments, and the interpolated values for all projectors based on them, are continuously recalculated each frame – is clearly visible. A calculation of a new time moment, by necessity also updates the displays at the detail and group timeline as well, resulting with their animation in sync with the state displayed in the 3D world (and the right panel as well, if needed).

Having that in mind, here is a bit more precise specification of the player time control buttons (the indexes refer to Figure 48):

- Quantization step backward [s1], forward [s2] – these arrows situated at the ends of the detail timeline, move the current player time one quantization step (250 ms) forward or backward. If the new player time happens to not fall on a quantized time instance (which in practise happens only when the player is stopped after hitting play), it is quantized by rounding to nearest position (auto-quantizes).
- Previous group start [3], your group start [4], next group start [5] – move the current player time at the respective group start time, easing the temporal navigation within the player. 'Your group start' moves the player time to the beginning of the timeline for a teacher account – and the respective group start for a student account.
- Rewind [6] – sets the current player time to 0 – the relative start of the class allocated time slot

- Fast backward [7], fast forward [11] – adds or removes a second from the current player time as long as these buttons are held – simulating a fast forward/backward (auto-quantizes)
- Previous cue [8], next cue [10] – sets the new current time to the position of the previous or next cue, relative to the 'old' current player time (auto-quantizes)
- Play [9] – initiates a timer function, and a continuous looping re-execution of the time calculation and the update functions that result with a new frame render. As soon as the frame is rendered, the timer function (which is independent) is checked, to derive the proper time, taking into account that because of complexity the frame render process may not complete in equal steps – hence, the current time can end up on non-quantized positions.
- The draggable current time marker on the group timeline [t] (or the group timeline slider) – as it is dragged along the timeline, a corresponding player time is calculated (relative to the groop zoom setting) from its x position along it, and this time is used as the "new" current time (auto-quantizes)

This kind of a solution, allows that: the player time; the player time indication; and the corresponding result in the 3D world (based on the light program) are always in sync – and in that sense, this version of the application behaves in principle as a video editing application, from the perspective of player controls. That is, play or fast forward will result with a playback of the light program "forward" in time, and fast backward will result with a playback of the program "backward" in time – when the player is stopped, the corresponding visual feedback in the 3D world is "frozen". This completely corresponds to a video file player, for a static camera – the difference here is that the camera operation is independent of the light program player timing operation, and hence a visual output that interlaces both dynamic camera positions and dynamic playback of the light program is possible (however, at expense of performance).

However, this is not the desired end behaviour in only one aspect, critical to light designers. That is, when light designers program their show in a real, physical set with a console, although the light program player of the console may be "paused", the effect on the projectors is not completely "frozen" in the sense that gobos keep

rotating, based on the cue data for that particular player time. This is a great visual aid to light designers, and as such is reproduced in PC software dealing with light programming as well – and so it should be implemented in this user interface too. That simply means, that the control of the gobo rotation should be removed from the main time based update loop, and should be made independent, referring only to the rotation speed cue data for that instance in time for a given project. As it is now, the gobos rotate only when the player time is continuously updated – which includes the play, and fast forward/backward functions – any continuous clicks on the right panel refresh the interface as well, so such an event will render rotating gobos as well although no new player time is calculated.

- **Cue editing functions**

The cue editing functions refer to functions that allow selection of multiple cues in the detail timeline, deleting them, copying them and pasting them in a new location. This is achieved by the cue selection/interaction on the detail timeline, in conjunction with the player buttons (the indexes refer to Figure 48): delete [12], copy [13], and paste [14].

The cue selection on the detail timeline can be described as follows. A cue is rendered as a green rectangle in the detail timeline. A mouse pointer hovering over a cue in the detail timeline is rendered with yellow:

Figure 52. A mouse hover over a cue indicated with yellow

With no cue selection, no delete [12], copy [13], or paste [14] are active. Left clicking on a hovered cue selects it, and the colour for a selected cue is changed to blue. Holding shift while selecting cues adds them to the selection.

Figure 53. Ilustration of a multiple selection of cues in the detail timeline

When a selection of cues is made, delete [12] and copy [13] buttons appear, depending on the user rights: a student group has rights to edit (delete, copy, paste) cues only

within the limits of their group slot – the teacher has access to the entire class timeline.

When a multiple selection of cues is made, the user can choose one as reference, and while still holding shift, click on the reference cue and drag it along the detail timeline – this will initiate movement (displacement) of all the cues in the selection upon release:

Figure 54. Illustration of the process of dragging a multiple selection of cues

While dragging, several "ghost" cue images appear – bright green indicates the starting cue within the selection taken as reference (from where the drag started), white outline indicates the current position of the mouse cursor along the timeline, the bright yellow indicates the closest quantized time position to the current mouse cursor position – that is the position where the reference cue will end up if the dragging press is released; all the other cues within the selection would be moved accordingly. It is important to note that moving in this version is a non-destructible operation – that it, if any cues within the selection, happen to overlap an existing cue on their new positions, the moving process is cancelled, and the yellow quantized indicator turns red. The same happens if any of the cues in selection would end up outside of the group or class timeslot bounds. If these tests pass, and the drag press is released while the indicator is yellow, then the selected cues assume their new position upon release.

Let's note that in this version, each cue edit action resets any selections previously made, as the easiest possibility to handle programmatically. Independent clearing of a cue selection (deselecting) is also possible, again through an "empty" click – here the empty click is defined as a click on the detail timeline only, where no cue is present. A selection within the group bounds of the account, gives access to copying and deleting cues. There is no undo for deleted cues. Copying cues puts them in a buffer; the amount of cues in the buffer is textually indicated on the supporting texts (below player buttons). If there are contents in the buffer, and if the current player time is within the group bounds of the account, the paste button appears. The paste action inserts the first cue in the copy buffer at the current time player locations, and all others "in the future", correspondingly. Pasting in this version is a destructible

operation, however within own group bounds only – meaning that, if a position of a pasted cue happens to overlap with already existing cue, the existing cue is deleted and replaced by the pasted one. Again, this can happen only within the group bounds of the account – the pasting operation stops as soon as the group bounds are overrun, that is, the pasted cues get truncated at the bounds of the group.

Let's also note that because of the cue interpolation in the engine, the engine always needs to refer to "the last cue" – hence, the very first, initial cue (cue 0) has special meaning, and it always contains definitions for all programmable parameters for all projectors, and it must stand on the very first time instant. Hence, it is given special status, and it cannot be copied, deleted or moved, regardless of the user rights – however, its contents can be changed, if the user rights allow it (for a teacher and the very first group of a class).

- **File (program archivation) functions and help**

For all accounts, the file functions deal with saving and loading a light program to and from the local hard disk (on the clients PC), in the form of a custom text file, having the extension .lys. This is achieved by the two player buttons save [16] and load [17] (the indexes refer to Figure 48). There are however, specific account based limitations. Guest and teacher account always load and save the entire class allocated timeline (the 15 minutes) – the student group accounts always save only their own respective group part of the light program. When a guest or a teacher load a .lys program from hard disk, they always replace the entire previous contents of the timeline – when a student group account loads a program from hard disk they replace only the cues in their own group slot – if there are additional cues in the file, they are truncated.

The hard disk operations are there to allow the students to log as a guest account, and try doing a light program on their own if they feel like it, without endangering the program stored at the database. For that matter, some student groups may organise their work in this manner – for instance, each member tries on their own, and then the winner's program, or elements from individual programs, gets loaded to the group slot, and saved to the database. In any case, it is a matter of work organisation of every different class.

The load operation triggered by the load button [17] always refers to loading a program from hard disk. In that sense, it triggers the familiar OS-based file dialog:

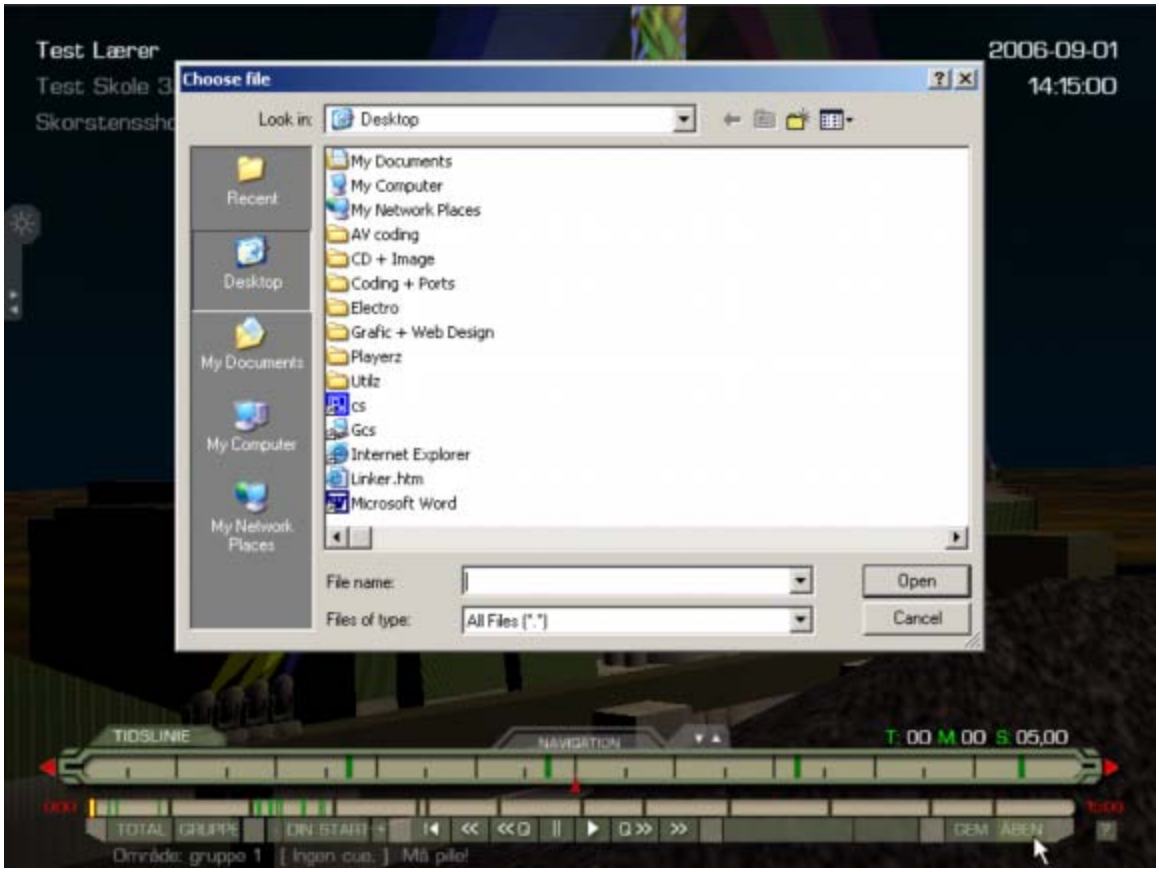


Figure 55. File open dialog upon execution of load command

This command always refers to loading from hard disk, since loading of a light program from database is only performed upon login, and only for teacher and student group accounts – for a single user, all the changes made in the light program after this point are local, until they get uploaded to the server. While this strategy is fine as long as sequential work of the group users is guaranteed, it may get problematic if say, two users log in at approximately the same time. This makes them receive the same initial version last saved on the server, and if they end up working on their designs for different amounts of time, and then saving their results at different times – then the former save is in danger of being overwritten by the latter. This is the case, however, only for simultaneous logins belonging to the same group or a teacher being logged in simultaneously with a student group – which is why a unique check is necessary.

The save button command opens a dialog box, which offers the options of saving to a database or saving to hard disk for teacher or student group accounts – the guest account has only access to save to hard disk.

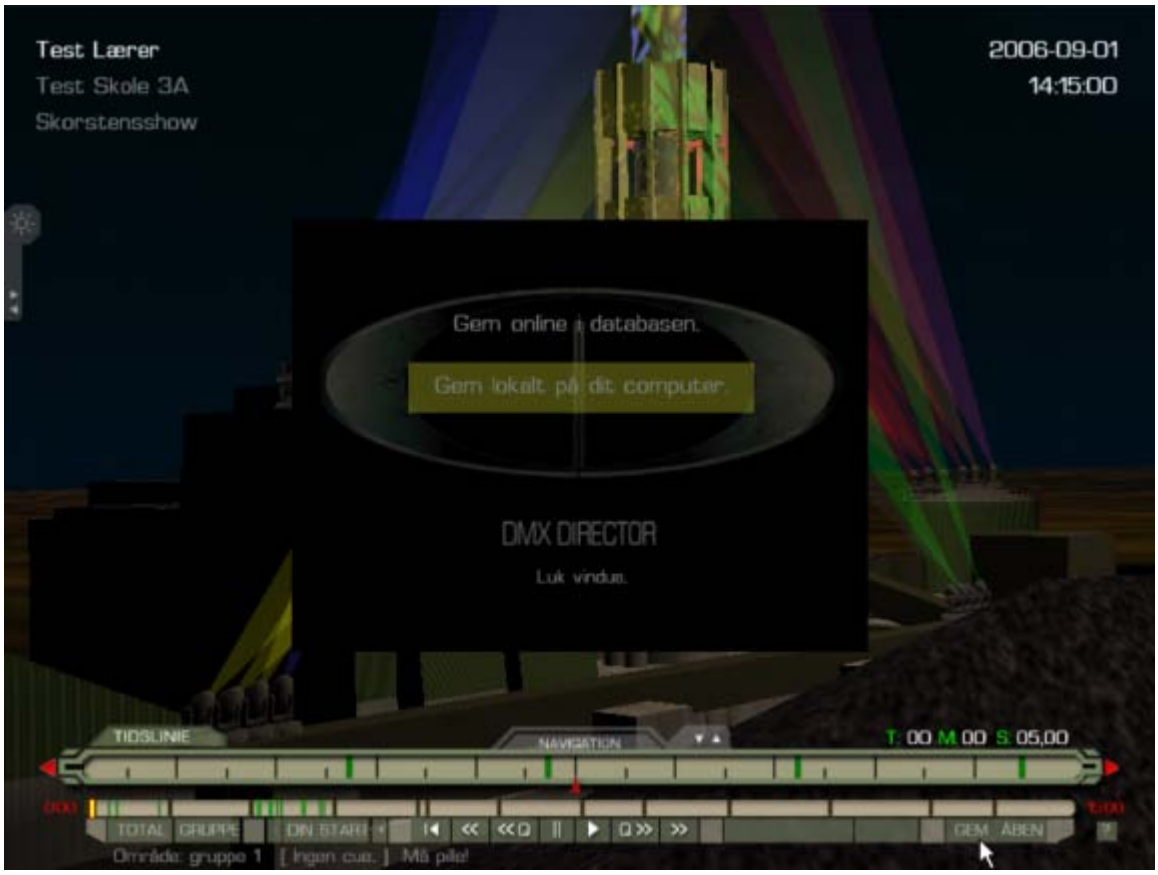


Figure 56. The save dialog with online and local saving options

The online saving process is a bit interesting here, as it should cause a file dialog again – however, in this context, the light program lays locally in the Virtools applet memory; as such, it actually does not have access to the local file system, and the only way to raise a file save dialog box is to force a download. Hence, when the save locally button is pressed, the light program gets converted into a single string, and sent to the remote server, where a specially programmed aspx page does nothing else, but reflecting this string of text back to the browser, forcing a download. In short, it is a trip to the server (and the delays associated with it), with no other purpose than raising a dialog box. This must be done as there are difficult to find other user friendly options to choosing a file location for our target group - having pupils type, by hand, absolute directory paths to the Desktop of their computer is certainly out of the question.

Otherwise, the file saving possibility on hard disk comes from the capability of Virtools to save its own internal Arrays as text file – this can however occur only in temporary directories where browser applets have access on the local computer. A reasonably complex algorithm parses all of the cues on a light program down to a single string, which then is saved as a text file. Let's also mention that the database construction demands that each student group account is a separate row, which then has the event-recording field. Hence, when saving to the database, the light program needs to be parsed as strings separate to each group, and then each such group portion is saved at the corresponding field in the database (that is completely true only for the teacher database save - the student group account replaces only its own program entry at database save). The same per-group parsing algorithm is applied when a student group account saves to hard disk as well – so a student group can "export" to hard disk only its own program – not those that their peers made. Therefore, the students can "cheat" the system only if the originator of a light program agrees to give a hard copy to the "copier" – otherwise, attempts for unauthorised copy of the light programs of the class peers, is discouraged by the application.

Finally, let us briefly mention the help button [18]. This button again rises a dialog box similar to the save dialog – for the current version, it is not fully implemented yet, as it shows only a single page. Eventually, the system should be able to parse in the language dictionary file, find how many help pages there are, and set up tab buttons within the help dialog for multi-page functionality.

10.1.4 Issues with the user interface

The user interface in its presented form is a result of several months of development, both as conceptual and technical solution. From the technical side, this is mostly related to implementing 2D interface elements, and correspondingly - taking into account the viewing window resolution, specifically for Virtools, as the chosen development environment.

The first issue with developing a user interface in Virtools, is that not all functionality desired is available. The Push Button BB exists in Virtools, to turn a 2D frame into a button, having the possibility to change the appearance according to

three separate materials for the released, pressed and hovered state. This behaviour block has been implemented at several spots in the application. However, all of the sliders are custom programmed using Virtools BBs and the scripting language VSL, so as to achieve that the slider marker (head) gets positioned where the user clicks on the slider – instead of having dragging functionality (in which user has to hover the marker first, click, hold, drag and release). The colour slider is also a 2D slider, in contrast to the other, 1D sliders. Similarly, the user interaction functionality of the detail timeline is custom programmed. As it also involves texture redraws as well, here is where using a scripting language maybe gets a little too slow, and starts taking it toll on the overall performance. Ideally, all the functionality of these user elements could be programmed on a machine level (say in C++) using the Virtools SDK, and stored a custom DLL. In this way, significant performance gains could be achieved.

The second issue is related to the fact that Virtools is, in the end, a 3D API engine; hence, development of a 2D interface has its difficulties in this scope. In essence, one shouldn't lose from sight that real-time 3D rendering is enabled by graphics hardware on board the PC (the graphics card, usually referred to by its processing engine, the GPU – graphics processing unit), which then introduces its restrictions: in this case, on size of bitmap images that can be imported as textures.

In that sense, it's important to remember the primary function of a GPU – to convert a 3D scene composed of data for geometric primitives (triangles, etc) into an image, or an array of pixels [24]. Historically, the very first feature on GPUs deals with an important step in this process of converting geometric into image data, and that is to apply a texture (a bitmap image) to the geometric meshes, or perform texture mapping [24]: "*Texture mapping is employed on high-end graphics workstations and rendering systems to increase the visual complexity of a scene without increasing its geometric complexity [...] Texture mapping allows a rendering system to map an image onto simple scene geometry to make objects look much more complex or realistic than the underlying geometry. Recently, texture mapping hardware has become available on lower-end workstations, personal computers, and home game systems... One of the costs of texture mapping is that the texture images often require a large amount of memory.* [25]" In addition to memory issues, texture mapping needs to be optimised as well: "*while texture mapping does add considerable realism to a scene, it also adds a number of new problems. The most obvious visual problems that appear when using textures in a scene are the aliasing artifacts that are visible when texture-mapped polygons are*

some distance from the viewpoint … MIP-mapping helps alleviate this problem. [26]." The technique of mip-mapping often demands that a so-called image pyramid is made, which represents the original texture bitmap on, progressively, half the size each level, down to a single pixel representation of the texture: "*The image pyramid or MIP-map is a commonly used means of reducing the cost of per-pixel texture accesses during filter construction … This is done by progressively averaging groups of four neighboring texels to form each new layer of the image pyramid. This process is continued from the initial, full detail, or base texture level, referred to as level 0, until the final single texel level is reached. The texture minification represented by each layer is based on the side dimensions, not the area. Initial texture dimensions must be a power-of-two magnitude* [27]". Thus, the requirement to reach a single texel (or a single pixel in terms of a bitmap image) by halving the dimensions of the original texture bitmap image in the mip-map process, imposes the restriction that images used in GPUs as textures must have "power-of-two", or 2^n dimensions in pixels: "*The width and height of a texture is restricted in most hardware to being a power of two for each dimension, and some hardware imposes maximums and other rules on these sizes* [28]". It might be that this restriction to texture size is rooted in optimisations down to the machine level execution, as mentioned here: ""*If you go out on the Internet and look for affine texture mappers, you'll undoubtedly run into a lot of very optimised x86 code that only works with power-of-two source texture sizes, and specifically two to the eighth power (or 256-bytes wide for 8bpp textures), because if you keep your textures to a power-of-two width, you can very easily handle the V carry we're discussing using some special x86 instructions that operate on 8-bit portions of the full registers*[29]"

Whatever the reason, the restriction is here, and it needs to be taken into account, especially for the 2D interface, since the artwork for it is eventually provided as bitmap images. Since the GPU hardware is in charge of rendering everything in the world, the interface as well, let's remember that it expects vertex data (geometry) as input, and then renders the pixel image out of it – so the same would be valid for the 2D interface elements – they will simply be rectangles, aligned to the camera, and textured by the GPU with the interface artwork. The first problem that comes out from this is that the original interface artwork might not conform to the 2^n specification. To retain the original scaling, the original artwork image (for instance 100 X 200) can be simply padded with transparent pixels, so it conforms to the power-of-two requirement (128 X 256). This then implies, that if a corresponding rectangle (100 x 200 pixels) is to host the interface element, one then must manually map the

pixel contents from the padded image into the host rectangle, by manipulating the texture u/v coordinates for that rectangle.

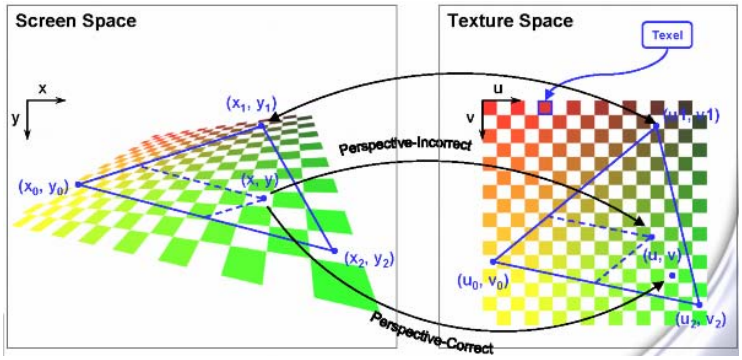


Figure 57. Texture Mapping: Texture Coordinates Interpolation (Ref.[24])

Fortunately, Virtools allows direct manual editing of the UV (texture) mapping for a 2D Frame - which is an internal Virtools data structure, representing a rectangle expressed in the coordinate system of the screen (not the 3D world). Another problem comes when a given user interface is made of several layers and needs to be composited. In that case, good practice is to set a 2D Frame as a holder in Virtools, with the same dimensions as the texture file; then child 2D Frames can be positioned as children, expressed in coordinates relative to the parent – and thus the positions of the child frames will be the same as the pixel positions within the texture image.

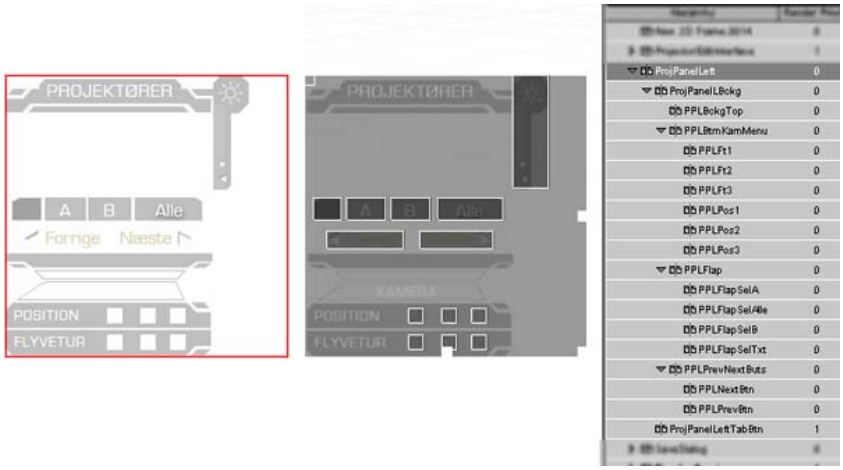


Figure 58. 256x256 bitmap texture file (left), 256x256 2D frame with child frames in Virtools (middle), corresponding hierarchy for the Virtools object (right)

Let's just note that in this case, if any frames have Push Button BB applied, they must not be covered by another frame (the stacking order is controlled by Render Priority parameter in the hierarchy), otherwise the push button behaviour will not trigger.

The final issue related to the 2D interface aspect is a problem, which in one or another way is always present in web design, and that is layout appearance at different screen resolutions. In the 3D context, it is essential to repeat the concept of a view vindow, or a viewport: "*A viewport is the actual 2D window that the 3D world is shown within*[30]"

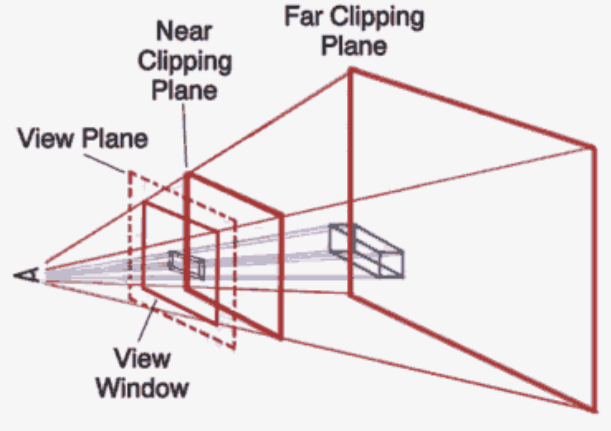


Figure 59. Illustration of a camera frustrum and a viewport (Ref. [30])

The Virtools program is eventually consumed as a plugin in a web page, embedded through the OBJECT or EMBED tags, where its size is specified as well – and that is eventually the size of the viewport. This size can be made fixed in pixels, or in percentages – in which case, the viewport will scale according to the browser window size. The change of viewport size can happen in one more way – if the user chooses a full screen display instead. In this case, by right clicking on the Virtools object in the webpage, one can change the full screen resolution – which is also a change of size of the viewport. Scaling the viewport does not change the camera angle of view into the 3D world, the view is simply extended – however, it does have impact on 2D frames which compose the user interface, as their position is determined relative to the viewport. In that sense, early versions of the prototype showed erroneous behaviour when viewed in full screen mode.

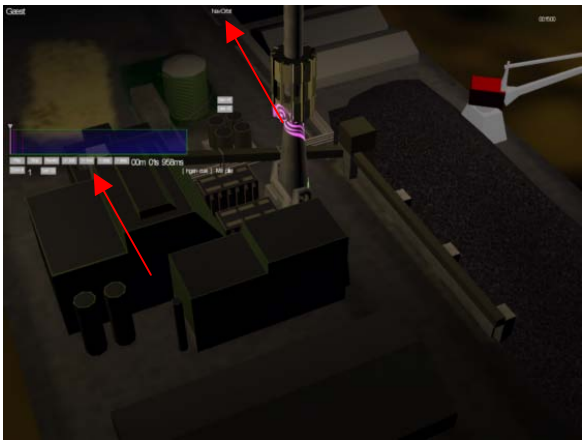


Figure 60. Erroneous repositioning of user interface elements (arrows) upon change of viewport resolution

Virtools offers so-called “stick” parameters, to help alleviate such problems – the erroneous behavior might be alleviated by having the elements stick to all corners of the viewport, in which case they would be scaled. However, stick scaling is relative to the viewport size in the Virtools development environment (which is often changed manually during development work), so it is hard to set up their sizes manually. There is however, one more solution, and that is what Virtools internally calls homogeneous coordinates: “*In 2D, a screen position expressed relative to the current screen resolution along each axis. Homogeneous Coordinates are used to ensure that, no matter the screen resolution, 2D elements can maintain a constant relative size and position* [31]”. As the design for the current version of the prototype was originally made as a 1024 x 768 image, the relative positions and sizes of all UI elements can be measured and the corresponding homogenous coordinates calculated. In that manner, all of the made 2D interfaces were converted into homogeneous coordinates, starting from the outermost children, progressively through each parent level, until the master holder was finally transferred to homogenous coordinates. In particular, the size and position of the original element in the original design was determined, these were divided by the original design dimensions to obtain the corresponding homogenous coordinates, which were finally entered for the master UI elements. In that way, the relative relationships of the original design UI elements with the original design size, is always duplicated in the Virtools application, no matter what chosen resolution. This is however, not the whole solution to the problem – as the actual image contents rendered in a frame are UV interpolated from an image that refers to the original design, at very high resolutions for the viewport, although the UI elements scale correctly, the pixel content inside them gets blurred.

Finally, 2D Frames in Virtools scale automatically with change of the viewport – however, if one uses automatic interface layouts in Virtools, such as 2D Grid Layout BB, to position buttons within a frame with automatic scaling, these effects are not automatically refreshed upon change of viewport size, although their target may be a homogenous coordinate 2D Frame. Thus, a VSL script was coded in the Virtools user interface, that checks (each frame) what the current size of the viewport is; when it changes, it triggers a redraw of all aspects of the UI which do not scale automatically. Let’s mention that text rendering in Virtools, like in other 3D engines, is based again on a 2^n texture, which is again a problem to consider – along with the scaling of fonts when the viewport size changes. In conclusion, one can say that development of a 2D interface in a 3D application can be technically challenging, as the previous discussion

indicates – and was certainly one of the development hotspots that were least expected in this project.

10.2 Projector lights modelling

A key problem in the user site application is modelling the visual appearance of light projectors and their lights, to achieve a realistic 3D preview of the light design. There are two major issues in respect to this. First, the lights should eventually render a light cone, and a reflection of a gobo pattern on the affected environment models; this is the *projector light simulation issue*. As this is not a standard feature on GPU's, a corresponding algorithm must be devised. Another issue is that the overall light design involves positioning of the projector models within the world environment, in their final planned locations, in addition to the light program that controls the parameters accounted for in the cue playback engine (let's remember that the only parameters that can be manipulated through the cue sequencer are color, dimmer, gobo pattern, and gobo speed – apart from that all other parameters, including positioning of the lights, are static and pre-designed). As such, we can identify two phases related to this problem, which we can call *light placement design issue*:

- Environment design phase – when the world environment objects are modelled in 3D, without considering any projectors.
- Light design phase – which involves placement and fine-tuning of projector objects in the finished world environment model

Such organisation is a motivation to program the engine of the user site to be independent of the environment models. Eventually, this would allow the code engine (related to interaction, database connectivity, etc) to be reused in a different context – ideally, the only thing needed would be different environment models and different projector placement, and the project could be repeated in a different location, without coding a new engine; in that sense, the engine can be reprogrammed and improved independently as well.

This has been accounted for in the current version of the user site prototype, by implementing an engine, that reads the current parameters of the light design (the static ones, like position of projectors) from the environment model itself, and from an external text file, acting as a database. Based on this data, the user site engine dynamically spawns (creates) the 3D objects necessary to represent the projectors and

their light cones (hereafter called site spawning process), which directly addresses the light placement design issue – within this code, measures have to be taken to properly initialise the projector light simulation as well. Thus, this algorithm (hereafter called projector setup process), which composes most of the init phase of the user site (mentioned first at the beginning of the chapter), must address both the light design (site spawning) and (to a certain degree) projector simulation issues. Let's briefly look at each of the projector simulation first.

10.2.1 Projector light simulation

As mentioned before, lights should eventually render a light cone, and a reflection of a gobo pattern on the affected environment models. Rendering the light cone is eventually a matter of creating a corresponding 3D object, so it is a part of the site spawning process, which is discussed afterwards. In relation to simulating gobo pattern reflections, we should again look at the GPU, and see what choices do we have at disposal; keeping in mind that earlier generations of GPUs might not facilitate easy implementation of this desired feature.

The place to start is to look at the light facilities in Virtools itself: "*Virtools uses three types of Light … Spot lights offer interesting visual effects but take more processing time than the other types of lights … Spot: A cone of light is emitted and only objects within this cone are lit* [31](Light Setup)".

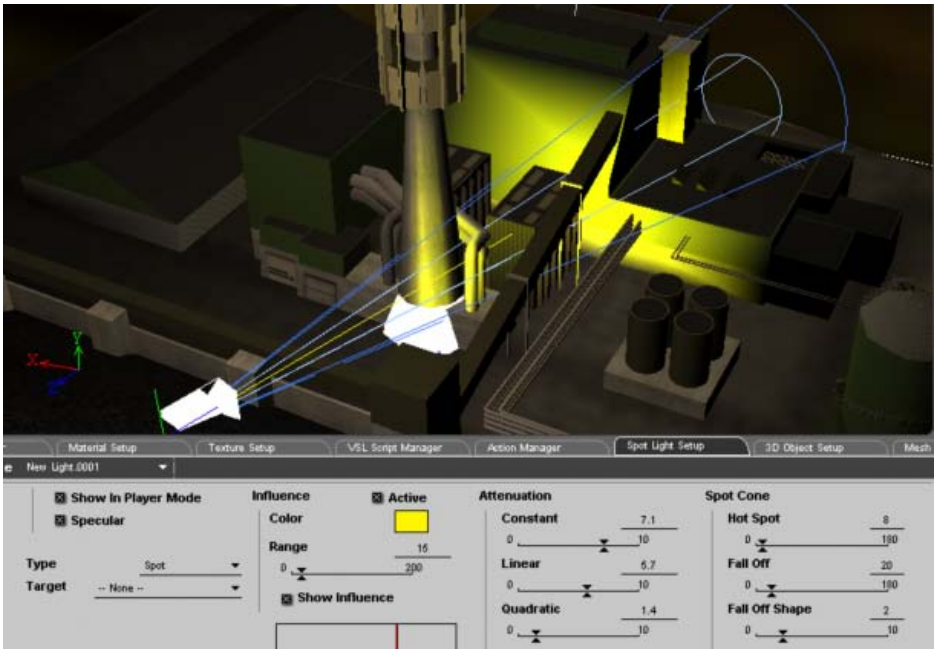


Figure 61. Model and appearance of a native spot light type in Virtools

As visible on Figure 61, the spot light accurately models a projector, in all respects apart from the rendering of a gobo pattern. However, there is one more problem: "*Light Properties are also used in the Light calculation ... from this table, a Spot light requires the most calculations ... Lighting calculation is done for each vertex of the scene and for each light ... Warning: The maximum number of active lights is limited when using a video card that support Transform and Lighting (T&L). In general the maximum number of active lights is 8.* [31] (Lighting, Shadows and Reflection Whitepaper)". Thus, the issue of targeting a particular GPU and its limitations starts occurring, even in this native case: to use native Virtools spot lights, the graphics card must support transform & lighting, and even in that case, apart from lack of support of rendering gobo patterns, there is a maximum limitation of 8 lights.

Looking at NVIDIAs introduction document [24], which also provides an overview of the capabilities of different generations of graphic cards (presented as "Evolution of the PC hardware graphics pipeline"), informs us that *transform & lighting* (see also [32]) is a technology, that as a GPU feature first gets introduced in graphic cards from 1999-2000. The transform and lighting unit found on these cards allows hardware accelerated processing of bump mapping, cube texture mapping and most important, projective texture mapping.

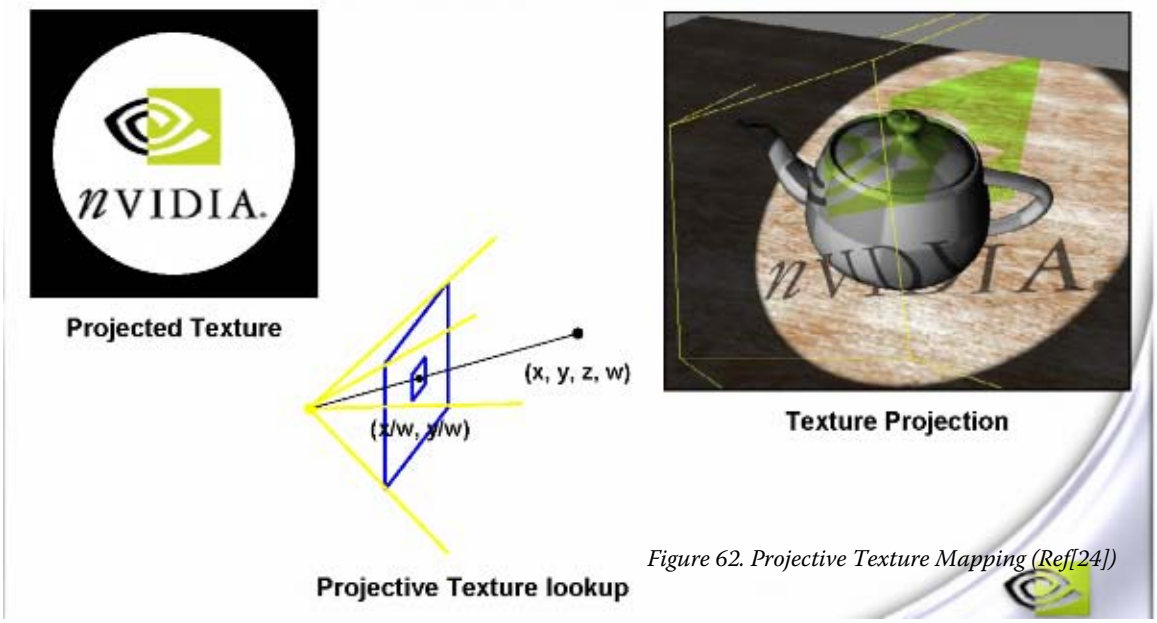


Figure 62. Projective Texture Mapping (Ref[24])

As Figure 62 demonstrates, projective texture mapping is the necessary processing function needed to simulate gobo lights.

A possibility is to target one earlier generation of graphic cards, those that support multitexturing (feature from 1998, [24]), as multitexturing allows for hardware accelerated calculation of light maps. However, for rotating gobos, that would mean continuous redraw of the light map texture, which is an operation impossible to achieve without going down to the Virtools SDK level and coding custom DLLs for texture drawing, which increases the licensing fees. The same argument goes if the goal is to target the earliest generation of GPUs, which support only simple texture mapping.

Virtools allows access to projective texture mapping through so-called effects that can be applied to a material. A particular type of effect is so called TexGen, which can also accept a pointer to a referential object; thus, when that material is applied to any other object, the display of that material will change according to the distance to the referential object.

While at this let's briefly clear the terminology used: the data encapsulation for a 3D object in Virtools is called 3D Entity; a 3D Entity has a Mesh, which "*is the physical structure of a 3D Object, it describes the geometrical representation of the object. A Mesh consists of Faces or Lines. Faces are made up of edges, which are themselves made up of vertices* [31] (Mesh (CKMesh) and Patch Mesh (CKPatchMesh))". A 3D Entity can have a Material applied to it. A Material "*holds the surface description for 3D Sprites, 2D Frames and for the faces of a Mesh. A Material most often uses a Texture* [31] (Material (CKMaterial)) ". Finally, a Texture holds one or more bitmap images, which contain the actual pixel information of a given visual appearance: "*A Texture can contain more than one image - in this case each image is a slot, with slots starting at zero (0) ... A Texture surface information is stored in a buffer in system memory and loaded into video memory when needed. The Virtools internal format is 32 bit, 8:8:8:8 (red, green, blue, alpha).* [31] (Texture (CKTexture))". The memory issues with GPUs and textures, discussed in the beginning of this section, are immediately emphasized in the Virtools documentation: "*You are recommended to use square Textures wherever possible for optimization; note that some old graphics cards only support square Textures. You should also ensure that Texture sizes are to the power of two, i.e. that the width or length of a Texture is 2, 4, 16, 32, 64, 128, or 256 pixels, again for optimization. Note that some graphics cards can only support Textures up to a certain size, e.g. 256x256 pixels. Check the specifications of the graphics card of*

your target platform. [31] (Texture (CKTexture))". In addition, a Mesh itself can have several so-called Material Channels: "*A Mesh can have up to 8 additional Material Channels. A Channel is a Material that is blended with the main Material or other Channels. Each Channel is associated with one Material for the faces, and one set of Texture coordinates. For example, in addition to the standard materials (up to one by face) for a Mesh, the Mesh could have a Channel of environment mapping, and a Channel of shadows, etc.* [31] (Mesh (CKMesh) and Patch Mesh (CKPatchMesh))"

The principle of the solution would then go like this: each object (3D Entity) that is to be lit by a projector (in other words, which should render a gobo pattern reflection), gets a global material applied to its Mesh (on the level of a 3D object) that defines the objects regular appearance (for instance, appearance of concrete, dictated by a texture), as any other object in the world. In addition, it gets additional materials that will render the gobo reflections (lets call these light materials), added as Material Channels to the Mesh. Each light material then uses the TexGen-with-referential effect, where the referential object is set to be a projector; the TexGen algorithm is chosen to be Planar in this version. In this way, an important problem is alleviated – and that is the problem that a given object may be lit by several projectors; henceforth one needs a separate gobo texture and projector reference for each projector. The TexGen-with-referential effect accepts only a single reference, so by having separate materials for each projector, they can be stacked in Material Channels for an object, and the desired effect can be achieved.

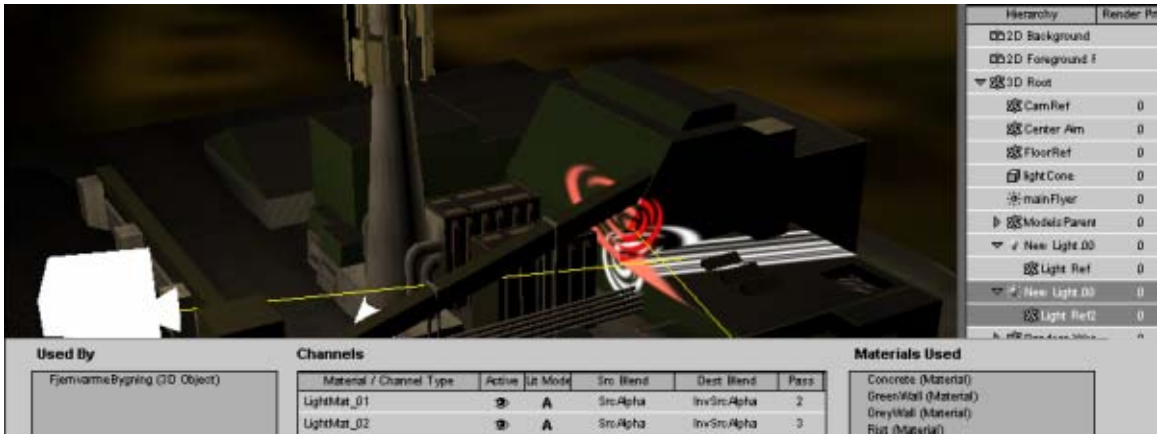


Figure 63. Illustration of compositing of gobo patterns, by adding materials as channels to the mesh

Figure 63 shows a situation of two projector models being placed in the world, and the corresponding pattern reflections. The only object lit is the building, since that

is the only one (in that example) that has light materials applied as channels to its Mesh – one for rendering each gobo pattern respectively. Each light material refers to its own gobo texture (in this case they are both the same) and has individual color information. In addition, each light material has the TexGen-with-referential effect applied, where each material refers to its own projector. To be more precise, each projector model is composed of light model as a parent (not active, just acts as a holder) and a 3D Frame set as a child, which acts as the actual reference for the respective light material. Note that, as the TexGen settings are chosen to be Planar, the eventual reflection output does not depend on the distance between the projector and the building – instead, it only depends on the size of the face of the bounding box of the reference object! In that sense, it is not a totally accurate representation of projector light reflections. This is true in yet another regard – the faces of the lit object do not "know" if they are in a shadowed area, so they render the reflection anyways.

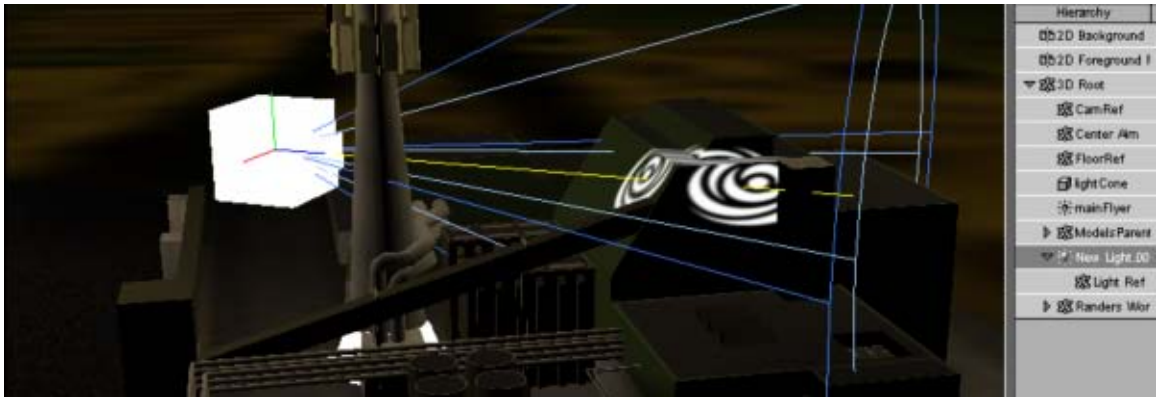


Figure 64. Illustration f wrongly lit faces; influence cones deliberately left to indicate that Planar reflection depends only on the size of the cube which acts as referential object

So, although this is not an ideal simulation of projector gobo pattern reflections, it still solves a lot of problems – how to composite images from several projectors, primarily. In addition, the default behaviour of this effect is such that rotation of the referential object is effectuated as rotation of the reflected pattern – in that sense, it is a great relief to the cue engine, as effectuating the cue rotation value for a given projector at a given time means only setting the corresponding rotation for the projector referential object; this operation is relatively trivial and economical in the context of Virtools scripting language (compared to drawing a texture "manually" from the same VSL scope, which is impossible to conduct in real-time – hence

utilising hardware acceleration of the GPU is a necessity for a custom texture drawing of gobos).

In this case, as well as when using a spot light model, there are no visible light cones "in the air" coming out from the projectors – only the light effects are rendered on the mesh materials. Since these cones can be seen as visual aid during light programming, they are simulated - that is handled by creating actual geometric 3D cone models, with dimensions approximating the expected physical set-up situation. As the light cones end up being actual conical 3D models, the dependence of the applied TexGen algorithm solely on the size of the referential object for rendering the eventual reflection output, actually allows a more-less accurate approximation of the gobo reflections - provided that the dimensions of light cones are calculated to simulate the actual projector set-up, and it is them acting as referential objects for the pattern reflections. The problem of faces in the shadow being wrongly lit, can be handled by separating the lit object (in this case, the chimney tower) into constituent face objects, and running a rough ray-tracing algorithm to determine which of them should reflect a given gobo; those that pass the test get a light material added as a channel, and compositing can occur on the hardware level as described previously.

This is the base of the solution of the projector light simulation in the current version of the user site. In essence, it tries to satisfy the real-time demands of the cue playback engine, by using light materials, added as channels to the mesh of the lit surface, that refer to one projector (and the gobo pattern thereof) each. The reference objects are in fact the projector light cone models, with dimensions calculated to reflect the physical projector setup; the corresponding gobo pattern is a planar reflection of their bounding box, calculated on the hardware level by the GPU (due usage of a TexGen effect in the light materials). Thus, the cue playback engine can settle with applying corresponding rotations to all light cone objects each render frame, which is possible to be performed on a Virtools scripting level in real-time; all the corresponding calculations of the gobo reflections, including colours, rotations and compositing, are passed on to the GPU for that render frame. Let's just mention that the cue playback engine, when applying colour, applies it both to the material of a light cone, and to the light material that renders the reflection pattern on the lit object.

By choosing a solution that incorporates material TexGen effects, we again incorporate a GPU feature: "*TexGen (Texture Coordinates Generators) are Fixed-*

Functions defined through States, that modify object's texture coordinates according to certain State values. Pseudo environment mapping, chrome effect or planar mapping can be achieved with TexGens [31] (Shaders Glossary)". The term fixed-functions stands for "*Hard coded rendering algorithms used by GPU on 3D graphics cards. The real-time 3D card industry started building specialized chipsets (GPU - Graphic Processor Unit) meant to relieve the CPU from computing most common 3D tasks (flat shading, gouraud shading, perspective, texture mapping and some blending operations). Many of those chipsets made the success of SGI workstations, whose technology quickly trickled down to the game industry via 3D card manufacturers like 3dfx, NVidia and ATI. Until recently, the main drawback of these inexpensive 3D cards was that rendering algorithms were limited to hard-coded functions (called Fixed-Functions - used in the Fixed-Pipeline) implemented by card manufacturers. This constraint is one of the reasons many 3D games look and feel alike.* [31] (Shaders Glossary)" The fixed function pipeline is a feature introduced with the 1999-2000 Tranform and Lighting GPUs [24] - so, by using TexGen effects, the target deployment PCs are again limited to those computers that also have a graphics card of that or a newer generation.

As mentioned previously, most of the calculations related to this specific projector light simulation, are conducted as part of the projector setup process, within the init phase of the user site. This process is briefly discussed in the next subsection. Before continuing, let's just mention that as a part of the development for a projector simulation algorithm, an attempt was made to code a pixel shader. It's development was abandoned primarily because during testing, a machine with an older GPU, which could run the TexGen effect, could not run the pixel shader, in spite of it being written in the lowest version of the HLSL language. Apart from that, it was possible to achieve a gobo reflection (although not an accurate one), and to make it respond to the referential object movements by using this experimental shader.

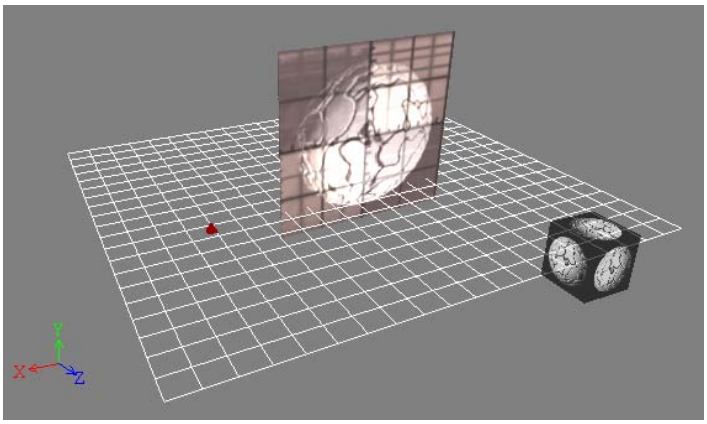


Figure 65. Example of the experimental pixel shader, rendering a gobo pattern

10.2.2 Projector setup process

The role of the projector setup process in the init phase of the user site, can be briefly summarised like this:

- To dynamically create (spawn) the projector models and light cone objects, based on information in the environment model and an external text file.
- Using the light cones generated in the previous step as referential objects, to determine which objects in the environment are lit by which projectors, and to add the corresponding light materials as Material Channels to the Mesh of the lit object.

The first step, the site spawning process, takes into account and anticipates the environment design phase and light design phase mentioned previously. That means, the first step expected is the design of environment models. The second step is when the environment is available, and projectors can be placed – this is the light design phase. The main issue in the light design phase is that when the projectors are placed, there are many aspects to consider – choosing the projector type, placing it at a given location, and focusing the beam – aiming it at a target and determining its angular spread. To be able to adjust to these phases, the user site utilizes pre-determined hierarchical relationships between the 3D models, and an external text file.

When the projector setup process starts, all projectors in the 3D scene are enumerated first, and stored in a Virtools Array. At this point, the engine expects a specific hierarchical relationship: for instance, all of the models composing the environment world must be parented to a single 3D Frame, situated in the coordinate center of the 3D world.

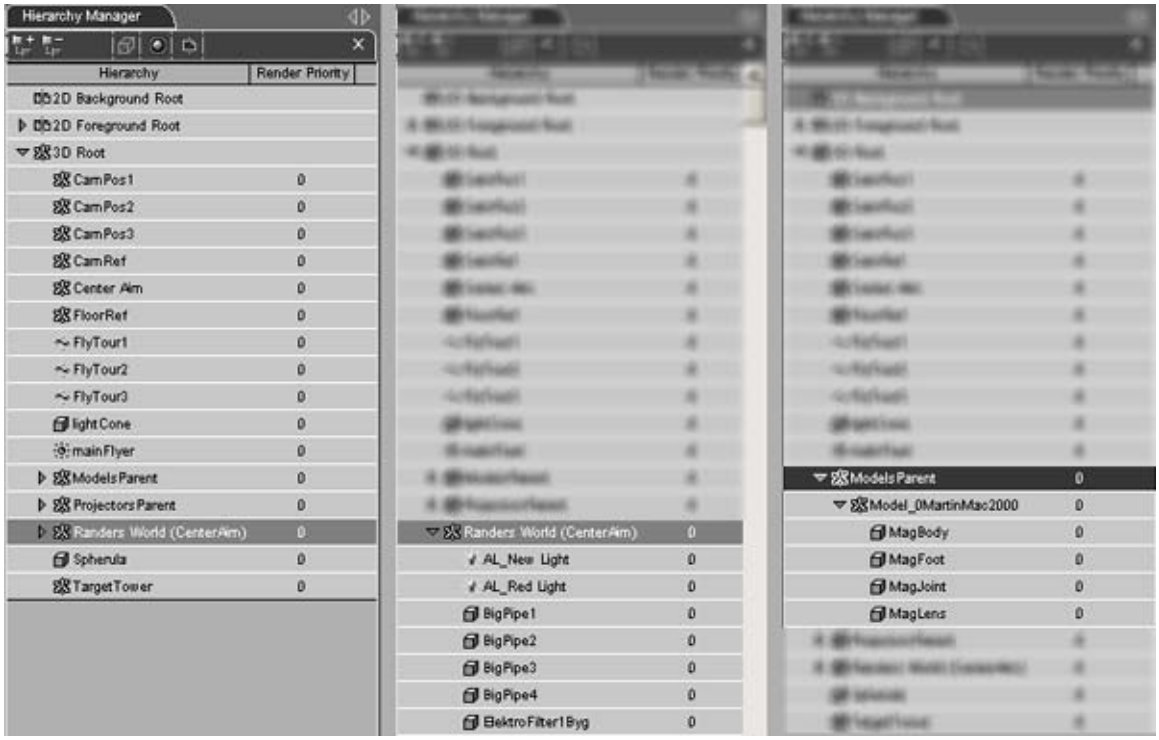


Figure 66.The expected hierarchy of objects: at the root of the scene (left), expanded view at the master node for the world objects (middle), expanded view at the master projector models node

Figure 66 shows several views at the expected hierarchy of objects. There are only few children of the root node of the scene, among them Randers World (Center Aim) which acts as the single parent of all the objects composing the world. Besides this child node, two others are special to the system. The first one is Models Parent, whose children are 3D frames, each representing a different type of projector, and containing the 3D model parts for the given projector type (at the moment, only one projector is modelled, Martin Mac 2000). The second is Projectors Parent, which contains the starting projector container models.

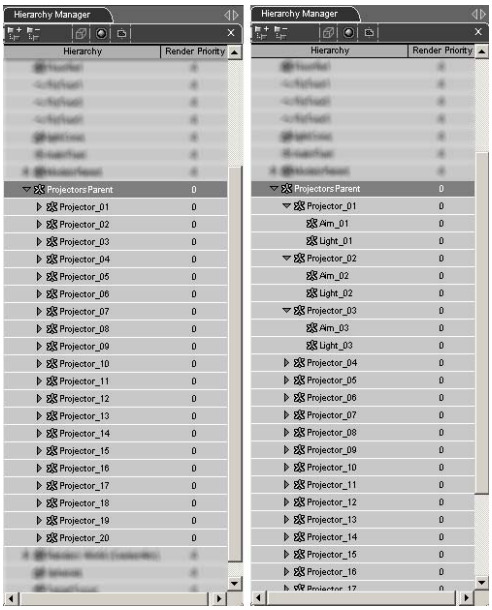


Figure 67. A look at the projectors parent node: first level children (projector containers) only (left), several projector containers expanded (right)

Besides the Projectors Parent, the master light cone object, the master selector object and the main camera reside in the root of the 3D scene, and are important to the projector setup itself; otherwise, 3D frames for the camera positions, the master target of the chimney tower and the curves for the fly-by tours also reside in the root.

The construction of a projector container model is again specific. Each projector has a master container 3D Frame, called Projector_XX where XX is a two-character integer index; it is through this naming policy that the system enumerates the projectors in the current version, although it might be just as well solved hierarchically. Each such master container has two children 3D Frames, Light_XX and Aim_XX.

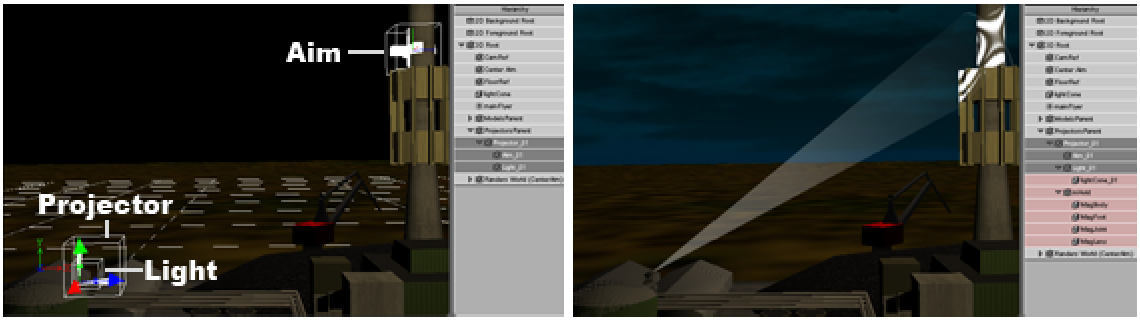


Figure 68. Example of a projector container, with light and aim children (left), the result after the projector setup (right)

The Projector_ frame acts as a general holder and position reference, and to facilitate easier movement of both itself and the Aim_ while placing the projectors within the world. The Light_ child frame ends up being the parent for the light cone model copy, and has the same position and orientation as the parent. The Aim_ child frame, is of course a reference to where the beam should be pointed to. In this case, it also acts as a limit of the cone length. In essence, the algorithm uses the distance between Aim_ and Light_ to calculate the proper size and orientation of the light cone. However, there is one thing unknown in such a setup, and that is the angular spread of the projector spotlight. As there are other such parameters attributed to a projector as well (such as a model), all these parameters (the spread included) are inserted in a spreadsheet text file – a tab delimited text file, called Projector.stp. This file can be edited in a text editor like Notepad or a spreadsheet like Excel, and allows assignment of individual angular spread, in degrees to each projector. These parameters are linked to a given projector through the Projector_ naming policy.

So, in more detail, the projector setup algorithm looks like this. First the applications looks for all Projectors_ and enumerates them in an array (Projectors). Then it reads the Projector.stp file, and concatenates its values to the previously created Projectors array (the number of rows and the projector names in the Projector.stp MUST match the projector containers in the model). Then, a loop is executed for all children of the Projectors array, in which:

- A reference to the corresponding Light_ and Aim_ children is obtained
- The distance d between them in world coordinates is obtained.
- The corresponding angular spread is read from the Projectors array.
- The final dimension of the light cone is calculated.
- A copy of the master light cone is made, with the previously calculated dimensions applied, positioned at position of Light_ and oriented towards Aim_.
- The particular projector model is read from the array. The corresponding original models are found in the Models Parent hierarchy, and copied at the location of Projector_. Parts are oriented towards the Aim_ accordingly (the careful reader may notice on Figure 44, for instance, that this is not always successful)

Once the distance d between the Light_ and the Aim_, and the angular spread in degrees α is known for a given projector, it is a matter of elementary trigonometry to find the dimensions of the bounding box and thus the light cone itself (the cone is taken to have a square base, with sides 2h in length).

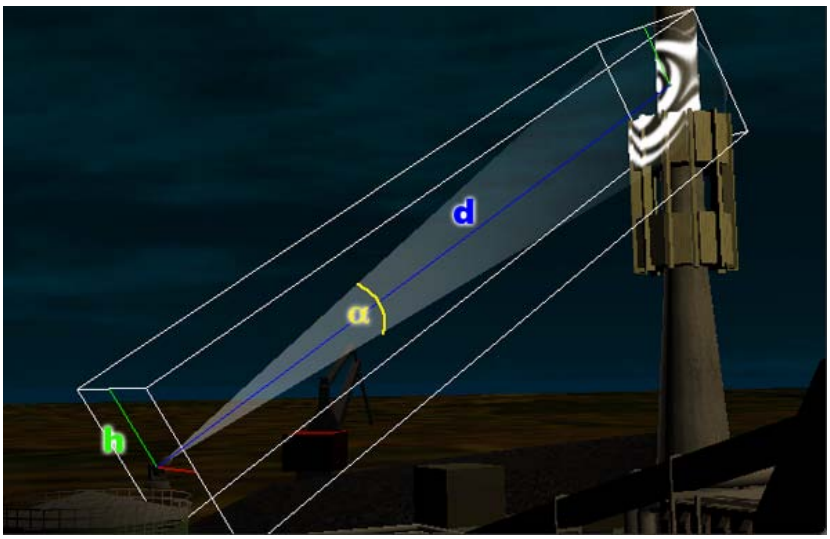


Figure 69. The geometry of a light cone's bounding box

According to Figure 69, the unknown side h is:

$$h = d \tan\left(\alpha/2\right) \qquad \textit{Eq 10-1}$$

It's a little more difficult to apply that data to the scaling of the light cone, as the original dimensions of the master cone need to be taken into account first. In any case, this is the core of the spawning process. The benefits of such organisation are in the fact that minimal number of 3D data is stored, as the projectors and the cones are spawned (although they probably would not contribute too much to file size anyway). This is because Virtools distinguishes between a 3D Entity, which does carry mesh data, and a 3D Frame, which doesn't, although it does have position and size (that is why all Virtools screenshots feature them as a white cross or a cube – few of the default appearances of a 3D Frame. 3D Frames are invisible in the final exported file). Then, the engine is generally independent of the environment models and the light positions – as long as the hierarchy is followed, and the Projector.stp contents matches the Projectors Parent hierarchy.

Once models are available for a given environment, the projector positioning part of the light design can continue in a Virtools file, that has only the environment models, the given projector hierarchy, a setup text file, and a copy of the spawning algorithm. The light designers can then move about the Light_ and Aim_ frames, and check by running the algorithm what the effects of spawning will be like – until a conclusive light design placement is achieved. The file saved as a result of this process, can simply have the simulation script removed, and can then be simply merged with the main engine Virtools file. As the current version expects the models to be included with the engine in a single file – such a step equivalents to replacing the previous hierarchy in the main Virtools file. Better usability might be achieved if the main engine is reprogrammed to read an external model file. Ideally, instead of working in Virtools, an intermediate application would be available, allowing the light designer to read an environment model, place a desired number of projectors with the desired types, and set their placements and spreads interactively – finishing with an export of the designed model files, and a correct corresponding setup file.

One more step remains, however, in the context of projector setup, and that is determining which particular faces of the chimney tower should be lit by a given light

cone once it is available. This was one of the most problematic tasks to approach. In essence, some type of a test is needed to determine which surface is lit (and get an additional light material channel) and which one isn't; this type of test needs to be conducted for each individual projector/light cone.

The first measure to take here is to limit the number of objects that can be lit. Hence, all objects composing the chimney tower, which are candidates for a gobo reflection, are placed in a Virtools Group (the Group in Virtools is a data structure separate from the hierarchy). The current version of the user site contains 464 objects in the group; that means that for a scene with 20 projectors, 9280 such tests must be performed with the limitation.

The test of which surface is lit went through several proposals. It is based around a double loop – one of them loops through all projectors; and for each projector, another loop goes through all objects in the lighting candidate group. The test is performed with a reference to a given projector from the first loop, and a reference to a given group object from the second loop. The very first version involved checking whether there is an intersection of the bounding boxes of these two objects, by using the Virtools BoxBoxIntersection function. However, as the bounding box can extend beyond a given face, this often produced erroneous results.

The next approach was to do some form of a ray intersection test. Initially, the approach was to take an initial ray from the projector direction, and then cast rays that fall within the cone with a given spread, spaced out evenly. One approach is to include a child object to the Aim_, orient the Aim_ and the child to the projector – so that they and the projector have parallel front planes, and have this child object cross equal distances along and across the front plane. Then rays can be cast from the Light_ to this moving child. Another approach is to start from the projector, find the maximum base radius (corresponding for the spread angle), and then cast a ray from the Light_ to the base radius at the other end – which is a ray on the cone itself. Then, the base radius is rotated in equal steps filling 360 degrees, and rays are cast accordingly. When the circle is fully tessellated, the radius is lowered, and the process repeated, until radius 0 is reached. In any case, this represents sampling within the volume of the light cone, and as such is limited by the "sampling resolution" or the number of steps of ray casting. Too many steps, and the application starts getting slow; too few, and errors are immediately visible.

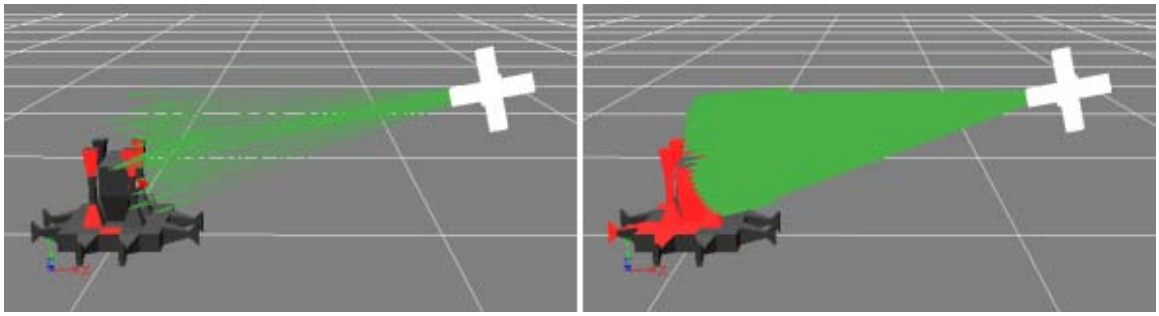


Figure 70. Dependence of the ray-tracing algorithm (based on cone tessellating) on the sampling resolution

An additional difficulty in the ray casting algorithm, is that what we need is to cast a ray, and get the first object intersected by it – this is achievable in Virtools only through the Ray Intersection BB (as a graphical block). There is a RayIntersection function which can be used as a method of a 3D Entity within VSL, however this method tells only if a given object intersects a given ray; there is no information whether it is first in line of sight of the ray or not. That means that the double loop (each projector X each face in the group) must be in some way implemented on a level of a graphic schematic in Virtools, such that the Ray Intersection BB is properly triggered at each step. This, coupled with the fact that Ray Intersection BB is computationally intensive, as well as the big number of objects, accounts for most of the long duration of the init phase (which even on decent Pentiums, can sometimes take up to several minutes to perform).

In the current version of the prototype, the problem of choosing steps for tessellating the cone with rays (and thus possibly losing faces from the test), is skipped altogether, and a slightly different approach is taken. During the double loop (each projector/light cone x each face in the group), first a basic BoxBoxIntersection is taken, as a measure to see whether the object falls within the light cone influence. If that is passed, a ray is cast from the centre of each face, back to the projector, and the first object intersected is obtained. If this first intersected object happens to be belonging to the lit objects group (that is, it is a part of the lit chimney tower) that is taken to mean that another object is actually the first in line of sight of the ray seen from the projector side; hence this is taken as a sign that the particular object should *not* have a light material applied – otherwise all others that pass the BoxBoxIntersection get a light material applied, that corresponds to the tested projector. This algorithm eventually takes up the majority of time of the init phase.

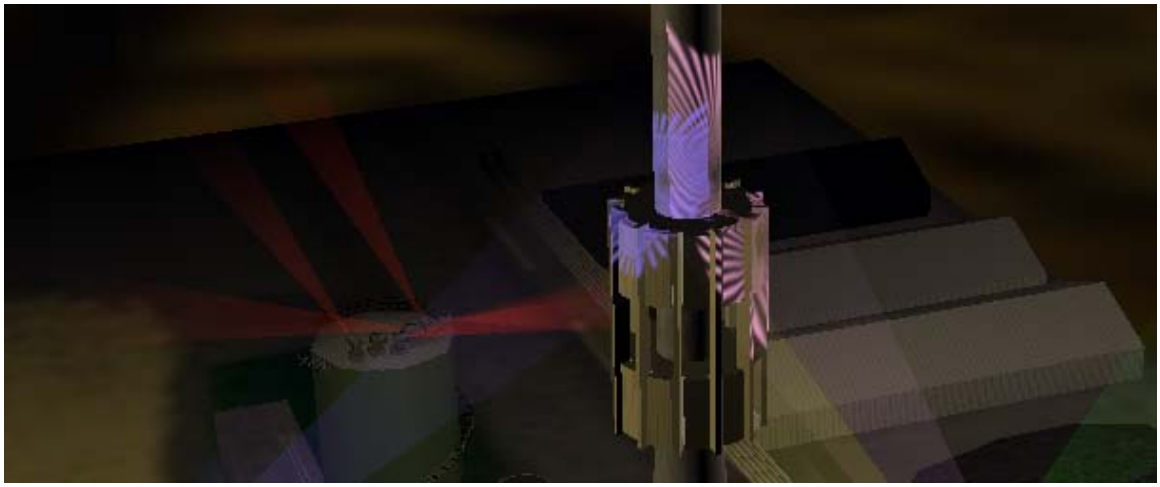


Figure 71. The results of the current version of the ray intersection testing

The effects of the current version of the light testing algorithm can be seen on Figure 71, where it is also visible that there is a quite sharp cut-off between surfaces that passed the test and those that didn't – but apart from that, the approximation is reasonable enough to allow visualisation of the eventual results in the real world. Let's also mention that the assignment of the light materials to their projectors is the only part of the algorithm described so far that is not automatised – since the referential object for a TexGen cannot be set for a material dynamically (through programming) in Virtools. Hence, the light materials have to be manually prepared within the model file, such that they all have TexGen effect applied, and the referential is set to the Light_ object (as during development, the actual light cones are not spawned yet – the good thing is that the Light_ parent will inherit the bounding box of its child cone, when it is spawned).

This concludes the discussion about the general projector set-up algorithm. Apart from this task, the init phase also parses the initial light program, read from the database (if any), and sets up the main application arrays, as well as the user interface. When the init phase ends, the user site engine goes into the main process loop. The main process loop initially only checks the keyboard and mouse listeners, and executes actions accordingly. These details are briefly discussed in the general overview of the engine in the next section.

10.3 General overview of the Virtools user site engine

The Virtools composition is organised in two Virtools Scenes – Login Scene and World Scene. There is only one Level based script, which loads the Login Scene; all other scripts are children of their respective scenes, and work only when the Scenes are active (the same is valid for any 2D/3D objects assigned to a Scene).

Upon start of the Login Scene, the two external text files are read – dansk.dct, which contains the language strings for most texts in the user interface, and projector.stp, which was previously discussed – and loaded as Virtools Arrays. The login screen interface is initialised, and the user is prompted to enter a login and password combination, and click on the login button. Upon click, the entered texts are passed to an external aspx page, to validate the user in the database. If the user is validated, and is not a guest, another request is sent to another aspx page, which retrieves all the information related to the user, and the entire class program saved up to that point in the database, for the class associated with that particular user. As soon as this response is received, all data is parsed apart from the light program (which gets parsed later), and the World Scene is loaded. If the user does not authorise, the World Scene does not load, and the user can enter a login password combination again.

When the World Scene loads, the init phase starts. As described before, it spawns the projector models and light cones, determines which surfaces are to be lit by which projector, and assigns the corresponding materials to them. In addition, the light program is parsed, and stored in an array; the other application level arrays are initialised too. Parts of the 2D interface are initialised as well, initial sensing of the viewport size is performed. When all these task are performed, a projector reset loop is performed, which performs the operation of selecting and deselecting each projector (for some reason, the 3D selectors around the projectors will not react to mouseovers without this step). Finally, when this is finished, a trigger is sent to the cue playback engine, which retrieves the initial player time as 0. Correspondingly, the cue update engine is triggered, and the contents of the initial cue are effectuated on the environment.

With this, the init phase is finished, and the application enters a state of constant listening to user input via keyboard and mouse listeners; this is the main process loop of the application. Accordingly to user input, the camera is manipulated, projector selection is checked, clicks on right panel (which modify a projector state, and modify/insert a cue) are processed, and clicks on the timeline – both cue manipulation and player buttons – are processed. Accordingly, the states of the internal arrays are changed, most important of which is the LightProgramArray, which as the name suggests, carries all cues in a light program for a given class. If necessary, a player time is obtained and the state for it effectuated in the world; in the case of a Play command being issued, this process is executed repeatedly within the main process loop. All the operations related to light programming are performed in the main process loop, on the clients PC.

The only exceptions to this mode of operation could be seen in the cases where yet another trip to the server is made, and these are the cases when saving a file on hard disk or to the database. In the first case, the light program text is simply reflected back to the client in order to open a dialog box, where in the second case we have actual parsing of the light program data, and inserting it in the proper place(s) in the database. However, the operation of the main process loop, in general does not depend on the server response in these cases. For a visual overview of these stages, please consult Appendix B.

Most of the init phase and the main process loop is coded in a script, child of the World Scene in the Virtools composition, called "Player Listener" – as these scripts are coded graphically in Virtools, here is a rough screenshot.

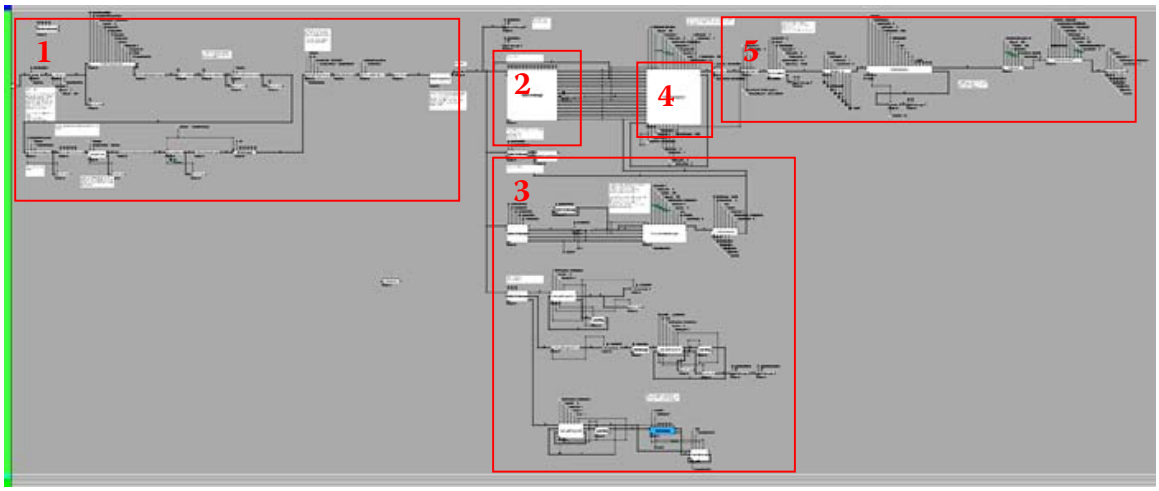


Figure 72. Rough screenshot of the Player Listener script

The numerated blocks on Figure 72 delimit the main function blocks:

1. Init phase scripts
2. Player listener (listener to player controls, like play, rewind, etc)
3. Other listeners (save/load; cue functions – click, copy etc; zoom clicks)
4. TimePlayerSequencer – the script that calculates the next player moment (+ some additional functions)
5. Cue interpolation and application scripts – scripts that interpolate the cue values for a given time moment, and apply them to the corresponding elements in the world.

Lets mention here also the main application arrays. Those are:

- GoboButtonsArray – a list of all gobo buttons (dynamically created, based on number of gobo textures)
- Lang – language array, holding the contents of the dictionary file dansk.dct
- PartsStartNorm – array holding the relative start times for each group in the class
- ProjectorModelSettings – projector setup file array, holding the contents of the projector.stp file
- Projectors – array listing all projectors, references to them, and their parameters, both static and dynamic (ran by the cue engine) – for dynamic, the most current values of the parameters
- SelectedProjectors – reference to any selected projectors
- ProjectorGroups – enumeration of projector groups (for the not yet implemented left panel)
- ShownCueEvents – list of the cues currently shown in the detail timeline
- SelectedCueEvents – list of references to any selected cues
- CopiedCueEvents – list of references to any copied cues
- LightProgramPix – help array used for rendering pixels
- LightProgramArray – the array holding the light program: a list of references to all cues in the program with a time stamp.

In addition, all cues are arrays as well, containing lists of affected projectors, and the set values for their dynamic parameters.

The key in this organisation is the Projectors array – any actions on the user interface that change a projector parameter, change it in the Projector array. Similarly, when a cue is interpolated, the values are applied to the projector entry in Projectors array. Finally, when a redraw is needed, whether in the 3D world or on the 2D interface, the Projector array serves as the source of information about the current projector values.

This is also the ground for the future interfacing with DMX – as a player engine is present, for each new defined moment in time, it obtains the previous and the next cue (relative to the current player time) for each individual projector, interpolates the values, and then writes them in the corresponding location in the Projectors array. As the player engine is demonstrated to perform its function, there is no reason why, instead of writing the interpolated value in an Array, one couldn't format it and send it as a DMX packet. The issue is of course not all that trivial, as timing of signals needs to be taken into consideration, as well as myriad of other measures – however, it is a good enough demonstration that the concept in itself is technically feasible. With this, we conclude the discussion about the solutions implemented in the user site.

11 Conclusion and perspectives

The online system, whose architecture is presented in this document, answers the initial demands, which is to provide an online client that facilitates programming of a light show by groups of students in a 3D environment. The programming of lights by the students should be conducted within a limited time window, and the results of the programming should be stored in a central location. The light programs saved in this way will serve as a blueprint for the actual playback of the programmed light show in the real environment – in this case, as a part of the centennial celebration in Randers. The present version of the prototype is a result of several months of work by the development team of BB&S and Seelite employees and Aalborg University students, and although it features a few bugs and is not yet complete, development work is still under way and the light show event is expected to take place as planned.

As discussed previously, the solution needs to, first, take into account the organisation and administration of the database, to be used for user authorisation and data storage. Hence, the problem outlined above is in fact answered by two websites – an administration site, aimed strictly at administrating the database, and a user site, which hosts the 3D application site. In both cases, there are no novel technologies used per se; however, care needs to be taken in choosing appropriate technologies for the delivery of the application. On one hand, the aspect of hardware compatibility of graphic cards with the software solution immediately springs out when talking about real-time 3D rendering, and this project is no different – especially since a part of it is expected to be conducted at schools, which might not boast the latest gaming hardware. For the same reason, developing a custom binary client is avoided as well – mostly to avoid cross-platform compatibility issues, but also to possibly avoid big download times and problematic installations, which may be beyond the capabilities of the primary user target group. In any case, delivery and deployment of the application should be as short and as trouble free as possible, mostly because of the short intended time (month or more) of the actual use of the application; not much time is available for updates and hardware related troubleshooting if problems occur.

Therefore, the development goes into a solution delivered through a web browser plugin as a more secure approach, and Virtools and Director are identified as

possible delivery technologies. Virtools is chosen, is spite of the huge price tag, as a 3D API that is more specifically related to issues in 3D applications, and as such offers greater assurance in development. This proves to be a good choice, as the testing (ran on a limited number of machines with Windows OS ranging from 2000 to XP) during development did not show serious problems with installation of the plugin and running the application within it; a lot of facilities that are used to emulate standard gameplay comes as pre-programmed functions; coding these in a low level library like OpenSG would certainly not be as trouble free as it was in the case of Virtools. However developing an application in Virtools still has its specific issues; such as building a 2D user interface and taking into account the viewing resolutions, or finding an algorithm for simulation of gobo pattern reflections. The gobo reflection algorithm, as discussed, uses certain hardware acceleration capabilities of the GPU, and as such limits the deployment of the application on PCs that have at least a transform & lighting generation of a graphic card (and of course, the PC must satisfy the requirements for installation of the Virtools plugin first). Another benefit of using Virtools here is that the plugin installation itself acts as a preliminary test, which will indicate whether a given target machine has the abilities to run the application. It is interesting, that from a technical perspective, the development of this project deals with problems present in both traditional web design and those particular to real-time 3D rendering. As mentioned, there is nothing new in the implemented technical solutions as such; although the approach may be novel, as it demands connection of different browser technologies (in this case ASP .NET, JavaScript, Flash and Virtools) to implement the full scope of the prototype.

From a graphical and interaction design perspective, both the administration and the user site are built to be simple interfaces that will facilitate easy execution of the corresponding tasks. In the user site, this problem gets more specific – how to merge the aspects of sequencing and playback of a program (which already has a defined, though complex visual language, in other media editing applications) with the aspect of 3D modelling which is accurate enough for light show design; and furthermore make them more available to the primary target group. The general approach in this case, is to make the application resemble a usual 3D game whenever possible, which resulted with the discussed 2D interface, camera navigation and 3D display (consider that a CAD application would regularly display the 3D view as a window in a 3D interface, and will usually allow interaction with the rendered 3D through mouse-dragging; having a 3D window that fills out the contents of the screen

and keyboard based, first-person camera navigation, as implemented in the prototype, bear a more direct resemblance to a game).

In retrospect, the application outlined in this document bears many specifics, so related projects are difficult to identify – indeed, it may be the first of its kind as an idea. Many of those specifics come from the fact that this is an application based on and around light programming, which in particular is coupled with an actual real light show. In that sense, it is in the overall context of the event, and the eventual interfacing with the real projectors, where the application gains its true importance; and where it can be seen that it does facilitate online collaboration (although limited and not in real-time sense as commonly expected from modern online 3D applications). Within the time frame of both the light programming phase and the entire event duration, the application can be seen to facilitate both collaboration, and in some sense, audience interaction – as the creators of the show are themselves expected to be a key part of the audience as well. Maybe this is here where the biggest quality of the event as a whole lies – it puts existing technologies (seen in a larger scope: from the level of browser technologies, to the level of light projectors as a technology) together in a novel approach; approach which in the end, aims to gather people together at the celebration in the outdoors.

This approach is a fresh view upon celebration events, as here it is the audience (or at least a key part of it) that produces the artwork, the actual show - everything else, from the web application to the set-up of actual projectors, can be seen as simply a tool to unleash the creativity of the participants. In that sense (its highly asynchronous/slow nature taken into account) it could serve as a new metaphor within audience participation and interaction, which as an intention is present ever since “post-modernism” and “performance art” became household terms on the arts scene – and is also an ever more occurring topic in media technology research. In any case, the social aspect of the event may be in itself a quality that could stimulate performing of the same type of events in different contexts and environments. Thus, one could expect further development of similar kind of events, where the application could be put in use – hence the motivation to produce an engine that is independent of environment models and projector setups, which would eventually allow easier porting of the application in a context of a different event.

Appendix

Appendix A. List of keyboard camera commands

Navigation Pilot mode:

W – forward (zoom in)
S – back (zoom out)
D – turn (rotate) right
A - turn (rotate) left
Arrow up – move up (rise)
Arrow down – move down (descend)
Arrow left – move left (strife)
Arrow right – move right (strife)
PageUp – look up (rotate)
PageDown – look down (rotate)
X – Look at tower center
Space - reset camera to initial position

Navigation Orbit mode

W - forward (zoom in) [mapped to panel]
S - back (zoom out) [mapped to panel]
D - Orbit right (keep distance) [mapped to panel]
A - Orbit left (keep distance) [mapped to panel]
PageUp - Orbit up (keep distance) [mapped to panel]
PageDown - Orbit down (keep distance) [mapped to panel]
Arrow up – rise up, continually looking at the tower
Arrow down - descend down, continually looking at the tower
Space - reset camera

Appendix B. Client-server communication chart

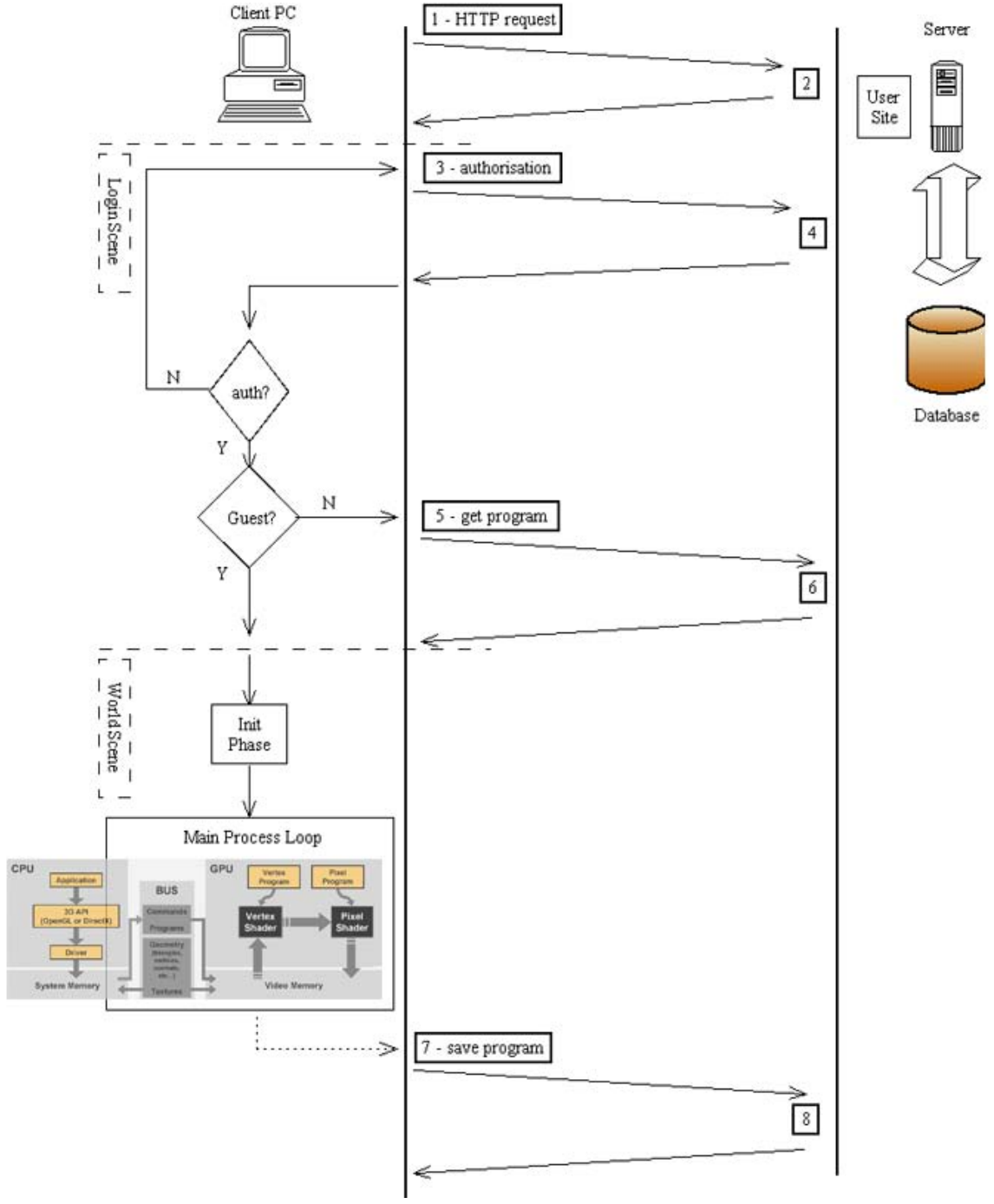


Figure 73. Chart of the communication process, and stages in the user site – the client side (the Virtools program) is on the left, the server side on the right; time axis points downwards. The CPU-GPU communication chart from Ref. [24] is placed at the main process loop, to stress the mportance of the GPU in this stage.

List of references

[1]. Martin, User manual, MAC 2000 Profile II (Rev. M), http://martin.com/service/downloadfile.asp?name=UM_MAC2000ProfileII_EN_M.pdf&cat=65

[2]. DevX.com Forums - Access memo field 64k limit..., http://forums.devx.com/showthread.php?t=51283

[3]. MySQL AB :: MySQL Connector/ODBC 3.51 Downloads, http://dev.mysql.com/downloads/connector/odbc/3.51.html

[4]. MySQL 5.1 Reference Manual :: 11.4.3 The BLOB and TEXT Types, http://dev.mysql.com/doc/refman/5.1/en/blob.html

[5]. Macromedia - Director MX 2004 page, http://www.adobe.com/products/director/

[6]. Virtools, The Behavior Company webpage, http://www.virtools.com/

[7]. java3d: Java 3D Parent Project, https://java3d.dev.java.net/

[8]. Rosenzweig, G., GDC 2004: Web-Based 3D Gaming presentation, http://garyrosenzweig.com/presentations/gdc2004/gdc2004-web3dtutorial-part1.pdf

[9]. Wild Tangent homepage, http://www.wildtangent.com

[10]. 3D Groove, Technology webpage, http://www.3dgroove.com/website/technology.html

[11]. blender3d.org :: 3D Web Plug-in page, http://www.blender3d.org/cms/3D_Web_Plug-in.161.0.html

[12]. Gamedev-GDC web games report (3D browser technologies), http://3d.5341.com/msg/6289.html

[13]. ASP - Tracking users when browser is closing, forum entry, http://www.codecomments.com/archive288-2004-7-226979.html

[14]. ASP .NET - How to detect browser close, forum entry, http://www.mcse.ms/archive109-2004-10-1192907.html

[15]. MA Lighting: Lighting Consoles / grandMA 3D, http://www.malighting.com/43.0.html?&tx_lightpowerpdb_pi1[parent_gruppe]=35&tx_lightpowerpdb_pi1[produkt_id]=1862&tx_lightpowerpdb_pi1[bereich_id]=1&tx_lightpowerpdb_pi1[image_id]=0&cHash=2de70224be

[16]. www.shownet.ch - GrandMA 3D, forum entry, http://www.shownet.ch/?board=MA;action=display;num=1091027825

[17]. gMA PC software, http://www.actlighting.com/MAlighting/gmapcpresnetation.htm
[18]. MA Lighting: Lighting Consoles / grandMA 3D / FAQs, http://www.malighting.com/43.0.html?&tx_lightpowerpdb_pi1[parent_gruppe]=35&tx_lightpowerpdb_pi1[produkt_id]=1862&tx_lightpowerpdb_pi1[bereich_id]=4&cHash=00d4c58273
[19]. Music Production :: Steinberg Media Technologies GmbH, http://www.steinberg.net/27+M52087573ab0.html
[20]. Adobe Premiere homepage, http://www.adobe.com/products/premiere/
[21]. Adobe Macromedia Flash Professional homepage, http://www.adobe.com/products/flash/flashpro/
[22]. BluffTitler DX9: Realtime 3D Video Titling and VJ Effects – Homepage, http://www.blufftitler.us/
[23]. WASD - Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/WASD
[24]. Programming Graphics Hardware, NVIDIA Tutorial, Eirographics 2004, http://download.nvidia.com/developer/presentations/2004/Eurographics/EG_04_IntroductionToGPU.pdf
[25]. Beers, A. C. et al., "Rendering from Compressed Textures", Stanford University, http://graphics.stanford.edu/papers/vqtexture/paper.pdf
[26]. Flavell, A., "Run-Time MIP-Map Filtering", Gamasutra article, http://www.gamasutra.com/features/19981211/flavell_01.htm
[27]. Ewins, J. P. et al., "MIP-Map Level Selection for Texture Mapping", IEEE Transactions on Visualization and Computer Graphics, vol. 4, no. 4, October-December 1998
[28]. Geczy, G, "2D Programming in a 3D World: Developing a 2D Game Engine Using DirectX 8 Direct3D", http://www.gamasutra.com/features/20010629/geczy_02.htm
[29]. Hecker, C., "Perspective Texture Mapping, Part V: It's About Time", Game Developer magazine, April/May 1996
[30]. Cole, D., Director Online Article: A New Dimension: 3D Lingo in Director, http://director-online.com/buildArticle.php?id=355
[31]. Virtools Dev help file
[32]. NVIDIA Corporation, "Second Generation Transform and Lighting (T&L)", http://www.nvidia.com/object/transform_lighting.html
[33]. Naemura, T., et al., "Multi-User Immersive Stereo", Dept. of Inform. & Commun. Eng., The University of Tokyo, Japan

[34]. Broll, W. et al, "The Virtual Round Table - a Collaborative Augmented Multi-User Environment", GMD - German National Research Center for Information Technology, Sankt Augustin, Germany
[35]. Leung, W. H. et al, "A Multi-User 3-D Virtual Environment with Interactive Collaboration and Shared Whiteboard Technologies", Carnegie Mellon University
[36]. MMORPG - Wikipedia, the free encyclopedia , http://en.wikipedia.org/wiki/MMORPG
[37]. List of MMORPGs - Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/List_of_MMORPGs
[38]. Anarchy Online - Massive multiplayer online roleplaying game – MMORPG, http://www.anarchy-online.com/
[39]. USITT DMX512 homepage, http://www.usitt.org/standards/DMX512.html
[40]. Verdenspremiere på digital lyskunst, ”Fra Damp til Digital” information page, http://www.energiranders.dk/el100aar/side7.html
[41]. Strøm i 100 år – start page, http://www.energiranders.dk/el100aar/forside.html