

Helsinki University of Technology

Department of Automation and Systems Technology

Laboratory of Media Technology

**Samu Ruuhonen**

## **Visual Information of 3D Objects in Real-Time Rendering**

Master's Thesis

6<sup>th</sup> of December 2004

Supervisor: Professor Pirkko Oittinen

Instructor: Ville Rousu M.Sc.

HELSINKI UNIVERSITY OF TECHNOLOGY Department of Automation and Systems Technology		ABSTRACT OF MASTER'S THESIS	
Author Samu Ruuhonen		Date 6.12.2004	
		Pages 62 + appendices	
Title of thesis Visual Information of 3D Objects in Real-Time Rendering			
Professorship AS-75 Imaging Technology		Professorship Code 2135	
Supervisor Professor Pirkko Oittinen			
Instructor Ville Rousu M.Sc.			
<p>The purpose of the thesis was to study methods of adding visual information into Tekla Structures, which is structural building information modeling (BIM) system made by Tekla Corporation, and possibly implement some of them. At the time, only some colors and lighting were used to separate different materials of 3D objects from each other, when viewing the structural model in a computer screen.</p> <p>The author of this thesis had some previous knowledge about real-time rendering and based on that some advanced 3D graphics techniques were studied and tested with simple self-made testing software. These techniques included 2D texturing, texture projector functions, bump mapping, procedural texturing, shading languages and anti-aliasing.</p> <p>Based on research and testing, the 2D texturing support was implemented into Tekla Z-kit, which is a 3D graphics software library used by Tekla Structures. Material and lighting editors were also created to make the use of Z-kit easier. In the future, the purpose is to take some shading language in use, and study what other 3D techniques mentioned in this thesis could and should be implemented in Tekla.</p>			
Keywords real-time rendering, texture mapping, bump mapping, procedural texturing, shading languages			

TEKNILLINEN KORKEAKOULU Automaatio- ja systeemitekniikan osasto		DIPLOMITYÖN TIIVISTELMÄ	
Tekijä  Samu Ruuhonen		Päiväys 6.12.2004	
		Sivumäärä 62 + liitteet	
Työn nimi Kolmiulotteisten kappaleiden visuaalinen informaatio reaaliaikaisessa piirroksessa			
Professori AS-75 Kuvatekniikka		Koodi 2135	
Työn valvoja Professori Pirkko Oittinen			
Työn ohjaaja DI Ville Rousu			
<p>Diplomityön aiheena oli tutkia menetelmiä, joilla voitaisiin lisätä visuaalista informaatiota Tekla Structures rakennesuunnittelu- ja mallinnusohjelmistoon ja mahdollisesti toteuttaa osa näistä. Tehtävänannon aikaan käytössä oli vain joitakin väri- ja valaistusmäärittäjiä, joilla voitiin erottaa eri kappaleet mallissa tietokoneen ruudulla katseltaessa.</p> <p>Diplomityön tekijällä oli jonkin verran aikaisempaa tietoa tietokoneiden reaaliaikaisesta piirrosta ja näiden perusteella joitakin kehittyneempiä 3D-tekniikoita tutkittiin ja testattiin itse tehdyllä testausohjelmalla. Testattuja tekniikoita olivat 2D-teksturointi, tekstuurien projisiointifunktiot, kohotekstuurit, prosessuaalinen teksturointi, varjostuskielet ja reunan pehmennys.</p> <p>Tutkimuksen ja testauksen perusteella 2D-teksturointituki toteutettiin Teklan Z-kit 3D-grafiikkaohjelmistokirjastoon, jota Tekla Structures käyttää. Myös materiaali- ja valaistuseditorit luotiin helpottamaan Z-kit kirjaston käyttöä. Jatkossa on tarkoitus ottaa jokin varjostuskielistä käyttöön ja tutkia mitä muita tässä työssä mainittuja 3D-tekniikoita voitaisiin ja pitäisi toteuttaa Teklassa.</p>			
Avainsanat reaaliaikainen piirto, teksturointi, kohotekstuurit, prosessuaalinen teksturointi, varjostuskielet			

## Acknowledgements

This thesis concludes my studies at the Helsinki University of Technology that have lasted ten years. At some point it even seemed I will never reach this point, but nevertheless here I am and now its time to turn the next page on my life and move on.

It has been pleasure to work with this thesis and I have learned quite a lot while making it, and not just from the subject. Some extra difficulties have risen from the facts that English is not my mother tongue and I have been living in Estonia for the last three years.

Fortunately, I have received lots of support from co-workers and family. I send my grateful compliments to my girlfriend Sini, who gave me the strength to finish my studies and also helped on grammatical issues on this thesis. I also thank my co-workers Juha Laukala for the help on implementations and technical support and Kari Sydänmaanlakka for the help on project coordination, and of course my instructor Ville Rousu and supervisor Pirkko Oittinen for the guidance.

7<sup>th</sup> of November 2004, in Tartu, Estonia

Samu Ruuhonen

## Table of Abbreviations

<i>API</i>	<i>Application Program Interface</i> – the specific method prescribed by a computer operating system or by an application program by which a programmer writing an application program can make requests of the operating system or another application.
<i>ARB</i>	<i>Architectural Review Board</i> – group of hardware and software vendors, that approves and maintains the list of supported OpenGL extensions.
<i>CPU</i>	<i>Central Processing Unit</i> – a term for the processor of the computer.
<i>GPU</i>	<i>Graphics Processing Unit</i> – a term for processor located in the graphics hardware board and responsible for the rendering work.
<i>OS</i>	<i>Operating System</i> – such as Windows, Linux and UNIX.
<i>RGB</i>	<i>Red, Green, Blue</i> – three main color components in computer graphics, which are used to represent all other colors.
<i>T-n-L</i>	<i>Transform and Lighting</i> – one part of the <i>GPU</i> responsible for transforming the vertices from model coordinates to view coordinates, and calculating the color of each vertex depending on lighting conditions.

## Table of Contents

<b>1</b>	<b>Introduction.....</b>	<b>1</b>
1.1	Structure of the chapters.....	2
<b>2</b>	<b>Background Information of 3D Graphics.....</b>	<b>3</b>
2.1	Graphics APIs .....	4
2.1.1	<i>OpenGL</i> .....	4
2.1.2	<i>Direct3D</i> .....	5
2.2	Fundamental 3D graphics concepts .....	5
2.2.1	<i>Vertex, line, triangle</i> .....	5
2.2.2	<i>Material</i> .....	6
2.2.3	<i>Transparency and fog</i> .....	7
2.2.4	<i>Lighting and shading</i> .....	8
2.2.5	<i>Texture mapping</i> .....	10
2.2.6	<i>Screen aliasing / anti-aliasing</i> .....	16
2.3	3D pipeline .....	17
2.3.1	<i>Vertex operations</i> .....	18
2.3.2	<i>Pixel operations</i> .....	19
2.3.3	<i>Frame buffer operations</i> .....	20
<b>3</b>	<b>Research Problem .....</b>	<b>21</b>
3.1	Adding visual information to the model .....	21
3.2	Modeling materials efficiently.....	22
3.3	Editing materials easily.....	23
<b>4</b>	<b>Review of the Most Used and New 3D Graphics Technologies.....</b>	<b>24</b>
4.1	Bump mapping .....	24
4.1.1	<i>Tangent space</i> .....	26
4.1.2	<i>Emboss bump mapping</i> .....	26
4.1.3	<i>Dot product bump mapping</i> .....	27
4.2	Displacement mapping.....	28
4.3	Procedural texturing .....	30
4.3.1	<i>Perlin noise</i> .....	31
4.4	Programmable pipeline .....	33
4.4.1	<i>Vertex shading</i> .....	33
4.4.2	<i>Pixel shading</i> .....	34
4.4.3	<i>Shading languages</i> .....	35
<b>5</b>	<b>Research Work.....</b>	<b>38</b>

5.1	Testing environment.....	38
5.2	Testing various techniques.....	38
5.2.1	<i>Testing texture mapping projector functions .....</i>	39
5.2.2	<i>Anti-aliasing testing with Steelmark .....</i>	42
5.2.3	<i>Testing dot product bump mapping .....</i>	43
5.2.4	<i>Testing pixel shader materials using Cg.....</i>	44
5.3	Comparing various techniques.....	49
5.3.1	<i>Texture mapping .....</i>	49
5.3.2	<i>Anti-aliasing.....</i>	49
5.3.3	<i>Bump mapping .....</i>	50
5.3.4	<i>Procedural texturing .....</i>	50
5.3.5	<i>Shading languages .....</i>	51
5.4	Summary of research work .....	51
<b>6</b>	<b>Implementations.....</b>	<b>54</b>
6.1	Fog and texture mapping support.....	54
6.2	Support editors for Z-kit.....	55
<b>7</b>	<b>Conclusions and Future Work.....</b>	<b>58</b>
7.1	Conclusions .....	58
7.1.1	<i>Comparing results to problem statements .....</i>	59
7.2	Future Work .....	60
	<b>References .....</b>	<b>61</b>
	<b>Appendix A – Steelmark run .....</b>	<b>63</b>
	<b>Appendix B – Code for pixel shaders .....</b>	<b>64</b>
	<b>Appendix C – Code for planar and cylindrical mapping.....</b>	<b>66</b>
	<b>Appendix D – Documentation and code for Z-kit editors .....</b>	<b>69</b>

## Table of Figures

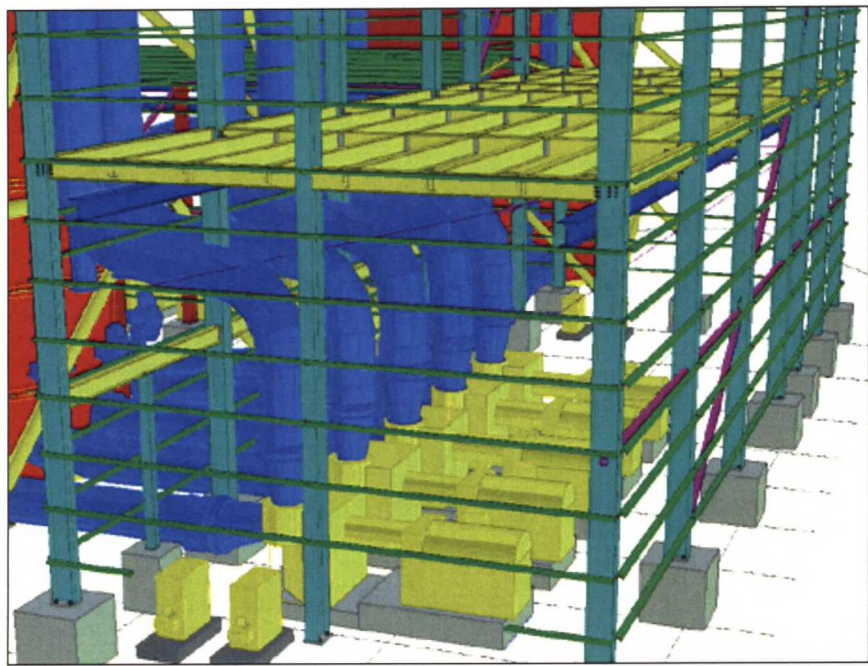
<b>Fig 1.</b>	A model created and viewed with Tekla Structures.....	1
<b>Fig 2.</b>	Real-time 3D graphics flowchart .....	3
<b>Fig 3.</b>	Specular reflection .....	7
<b>Fig 4.</b>	A scene rendered without and with fog.....	8
<b>Fig 5.</b>	A cylinder rendered with wire-frame, with Gouraud shading and with vertex normals shown .....	9
<b>Fig 6.</b>	Flat, Gouraud and Phong shading .....	10
<b>Fig 7.</b>	Texture mapping .....	11
<b>Fig 8.</b>	Simple texture .....	12
<b>Fig 9.</b>	Planar mapping applied to a cube, a sphere and a cylinder .....	12
<b>Fig 10.</b>	Cylindrical mapping applied to a cube, a sphere and a cylinder.....	13
<b>Fig 11.</b>	Spherical mapping applied to a cube, a sphere and a cylinder.....	13
<b>Fig 12.</b>	Aliasing effect on the left, mipmapping used on the right.....	14
<b>Fig 13.</b>	Mipmapping applied to original texture.....	15
<b>Fig 14.</b>	Trilinear filtering compared to anisotropic filtering .....	16
<b>Fig 15.</b>	Comparing aliased graphics with anti-aliased graphics.....	16
<b>Fig 16.</b>	Direct3D 8.0 pipeline flowchart.....	17
<b>Fig 17.</b>	View frustum when using perspective projection.....	19
<b>Fig 18.</b>	Clipping a triangle.....	19
<b>Fig 19.</b>	Disney Concert Hall modeled with Tekla Structures.....	21
<b>Fig 20.</b>	A simple texture mapped to a sphere with planar and cubic mappings.....	23
<b>Fig 21.</b>	A sphere rendered with world texture and bump-mapped world texture...	24
<b>Fig 22.</b>	An elevation map of the world.....	25
<b>Fig 23.</b>	A normal map created from the elevation map .....	25
<b>Fig 24.</b>	Tangent space in three vertices (© Imagination Technologies Ltd, 2000)	26
<b>Fig 25.</b>	An emboss map created from the elevation map .....	27
<b>Fig 26.</b>	Dot product bump map created from the normal map and applied to a sphere .....	28
<b>Fig 27.</b>	Bump mapping on the left, displacement mapping on the right (© Microsoft Research Asia, 2003).....	29
<b>Fig 28.</b>	Displaced vertices created from the vertices of the surface along the normal 29	
<b>Fig 29.</b>	Control mesh, subdivided mesh and displaced mesh (© Lee, Moreton and Hoppe, 2000).....	30
<b>Fig 30.</b>	One dimensional noise .....	31
<b>Fig 31.</b>	Salt and pepper noise blurred and up-sampled .....	32
<b>Fig 32.</b>	Taking average of noises at different octaves yields usable results.....	32
<b>Fig 33.</b>	Creation of procedural marble texture .....	33
<b>Fig 34.</b>	Waving flag created from flat surface using vertex shader.....	34
<b>Fig 35.</b>	Flat surface rendered with per-pixel lighting using vertex and pixel shaders (© ATI, 2003) .....	35
<b>Fig 36.</b>	Three test objects; an I-beam with holes, an Y-pipe and a half-pipe.....	39
<b>Fig 37.</b>	A simple texture planar mapped to an I-beam from two directions.....	40
<b>Fig 38.</b>	Hatch texture planar mapped onto an I-beam from two different directions 40	
<b>Fig 39.</b>	A simple texture planar mapped onto an Y-pipe from two directions.....	41
<b>Fig 40.</b>	Two different textures cylindrical mapped to a half-pipe.....	41
<b>Fig 41.</b>	Zoomed image from the model without and with anti-aliasing .....	43

<b>Fig 42.</b>	Half-pipe textured with tiles, without and with bump mapping .....	44
<b>Fig 43.</b>	Y-pipe covered with 3D perlin noise with two different frequencies .....	45
<b>Fig 44.</b>	Y-pipe object covered with concrete material .....	46
<b>Fig 45.</b>	Y-pipe object covered with stone material .....	47
<b>Fig 46.</b>	Wood rings created with pixel shader .....	47
<b>Fig 47.</b>	Y-pipe covered with wood material .....	48
<b>Fig 48.</b>	Fog effect on material helps the viewer to see the depth in the scene .....	54
<b>Fig 49.</b>	Brick wall created with the texture support of the Z-kit .....	55
<b>Fig 50.</b>	Material editor for editing Z-kit materials used in Tekla Structures .....	55
<b>Fig 51.</b>	Presentation editor for editing Z-kit presentations used in Tekla Structures	56
<b>Fig 52.</b>	Lighting editor for editing lights in the window .....	56

# 1 Introduction

Tekla Corporation is the leading supplier of model-based operational software products for infrastructure management in the world. The Building & Construction business unit develops 3D model-based software for the construction industry. Tekla Structures is the first structural building information modeling (BIM) system covering the entire structural design process from conceptual design to detailing, fabrication and construction. Tekla Structures allows for real-time collaboration between users across industries and project phases. The 3D model contains all the information required for design, manufacturing and construction; all drawings and reports are fully integrated within the model – generating consistent output. [TEK04]

At the beginning of this project one could create different materials and apply some surface properties to them (color, shininess, transparency) and attach the material to an object of a model. It was also possible to create multiple light sources and position them into the scene. When viewing the model on a two dimensional computer screen, the rendering device calculates how the material properties and light sources interact with each other, and renders the model so that different objects can be distinguished from each other and makes the model look three dimensional. (Fig 1)



**Fig 1.** A model created and viewed with Tekla Structures

The number of needed materials is increasing and soon it might be impossible to create materials that are possible to distinguish on the computer screen when using only color and shininess, since the human visual system can easily separate only a few dozens of different colors. This is important for the designer who is viewing the 3D model, since (s)he must be able to distinguish the different objects from each other quickly. And for the architect it would be much better, if e.g. wood and concrete would look somewhat realistic.

The purpose of this work was to research various ways to add visual information to 3D objects in real-time rendered 3D model and to develop the current 3D software library (Tekla Z-kit) used by Tekla Structures to support these materials.

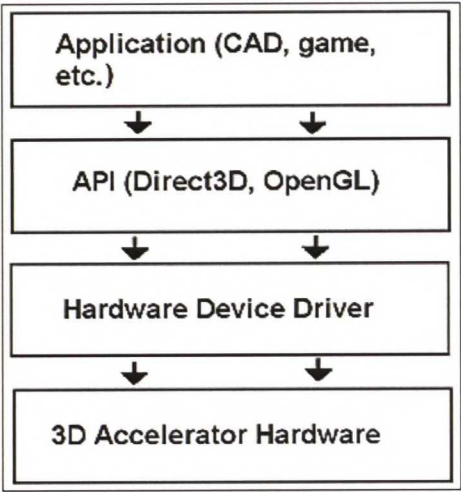
## **1.1 Structure of the chapters**

In the next chapter (chapter 2) an introduction is given to the field of 3D graphics and some of its aspects related to this work are explained. After that the research problem is studied more closely and defined more precisely (chapter 3). After getting familiar with the basic 3D graphics concepts and the research problem, some carefully selected advanced 3D techniques are studied and explained (chapter 4). Some of the techniques that could be used in the future are tested and reviewed (chapter 5) and as a result some implementations are made (chapter 6). And finally, a summary of the work and future thoughts can be found in the last chapter (chapter 7).

## 2 Background Information of 3D Graphics

In this chapter some basic background information of computer graphics focusing on real-time 3D graphics are presented. This chapter is intended for readers who are not familiar with the subject, others can jump directly to the next chapter. More introductions to computer graphics can be found from Hearn and Baker's book [HEA97] and a very thorough introduction to the real-time 3D graphics rendering can be found from Akenine-Möller and Haines' book [AKE02]. From the web one can read the article titled "3D Pipeline Tutorial" in ExtremeTech.com written by Dave Salvator [SAL01].

Real-time 3D graphics is the common naming convention for interactive computer graphics processed in three dimensions and usually projected to a flat two dimensional computer screen for viewing. Interactivity means that the user can change the appearance of the graphics by using the mouse or the keyboard (or other



**Fig 2.** Real-time 3D graphics flowchart

input device). The basic flowchart for a common 3D application can be seen in the figure on the left (**Fig 2**). An application such as CAD software or a 3D computer game is linked with a graphics *API* (usually Direct3D or OpenGL). In application run-time the graphics *API* calls are routed to the hardware device driver. The driver is usually made by the hardware vendor and it is 3D accelerator specific.

The main computer components affecting real-time rendering include the *CPU*, the memory and the bus. The bus is the data path on the computer's motherboard that interconnects the *CPU* and the memory with attachments in expansion slots, such as the graphics accelerator. The *CPU* retrieves the data from the memory and sends it to the *GPU* through the bus. This procedure is called the push method. The *GPU* can also directly read the memory through the bus, and this procedure is called the pull method or direct memory access (DMA). The rendering speed is limited by the

slowest component of the computer, and to take full advantage of all the components the system must be balanced. [AKE02]

## 2.1 Graphics APIs

Currently there are two major real-time 3D graphics *APIs* in use; OpenGL and Direct3D. OpenGL is mostly used in the UNIX world and Direct3D in Microsoft's Windows. Currently, the Z-kit library is using OpenGL as its graphics *API*, but in the near future the support for Direct3D is planned to be implemented. In this chapter a brief introduction to both *APIs* is given.

### 2.1.1 OpenGL

The OpenGL *API* [SEG97] began as an initiative by SGI (Silicon Graphics, Inc.) to create a single, vendor-independent *API* for the development of 2D and 3D graphics applications. Prior to the introduction of OpenGL in the early 90s, many hardware vendors had different graphics libraries. This situation made it expensive for software developers to support versions of their applications on multiple hardware platforms, and it made porting of applications from one hardware platform to another very time-consuming and difficult. The result of SGI's work was the OpenGL *API*, which was largely based on earlier work on the SGI IRIS GL library. The OpenGL *API* began as a specification, and then SGI produced a sample implementation that hardware vendors could use to develop OpenGL drivers for their hardware. [SGI04]

Modifications to the OpenGL *API* are made through the OpenGL Architecture Review Board, an industry group containing founding, permanent, and auxiliary members. The current version of the OpenGL *API* is 1.5, but Microsoft has implemented only the version 1.1 in the Windows OS [MIC04a]. Software developers do not need to license OpenGL to use it in their applications. They can simply link to a library provided by a hardware vendor. [SGI04]

OpenGL is a very dynamic *API* in a sense that every new technique can be implemented quickly with the extension mechanism. This also allows the new techniques to be used with Windows OS, even though only the 1.1 version is implemented by Microsoft. A big disadvantage that comes with the extensions is the fact that they are mostly made by hardware vendors and you can never be sure how

an extension will behave in some specific hardware configuration without testing it. A great resource for programming with OpenGL can be found from Woo's book [WOO97] and for advanced use one should check Blythe's web site [BLY99].

### 2.1.2 *Direct3D*

In 1995 and 1996 Microsoft established a new program to support games on PCs running its Windows 95 operating system. Microsoft chose not to use the OpenGL technology it already provided in Windows NT to handle 3D graphics for games. Instead, Microsoft purchased Rendermorphics, Ltd. and acquired its 3D graphics *API* known as RealityLab. Microsoft reworked the device driver design for RealityLab and announced the result as a new 3D graphics *API* called Direct3D Immediate-Mode. [AKI98]

For now the Direct3D has come to a version number 9.0 [MIC04d] and almost every Windows software is using it as their graphics *API*. The advantage that comes with the Direct3D is the fact that it is very exactly defined and specified, which makes it less vulnerable to drawing errors and malfunctions. A good introductory book about Direct3D graphics programming is written by Frank Luna [LUN03].

## 2.2 Fundamental 3D graphics concepts

In this chapter the fundamental 3D graphics concepts mostly affecting the visual appearance of 3D objects and which are common to most software and hardware 3D graphics systems are presented.

### 2.2.1 *Vertex, line, triangle*

Vertex is the fundamental building block in 3D graphics. It is a single point in a 3D world and is usually represented with three coordinates;  $x$ ,  $y$  and  $z$  (although other representations are also possible). Mathematically the size of a vertex is zero, but usually it is visualized with one pixel on a computer screen.

Line is a straight connection between two vertices. Mathematically the area and volume of a line are zero and it has only one attribute, which is length. The line is normally visualized with a one pixel wide line on a computer screen, but the line width can also have some other values.

Triangle is formed with three consecutive vertices mathematically defining an area, but the thickness of the triangle is zero. Triangles have always a front face and a back face. If a right-handed coordinate system (OpenGL) is in use, the front face is towards the viewer when the vertices of the triangle are processed counter-clockwise. And if a left-handed coordinate system (Direct3D) is in use, the front face is towards the viewer when the vertices are processed clockwise.

Almost all the *GPUs* are optimized for drawing triangles and therefore all the 3D objects build from polygons are tessellated to triangles before sending them to the *GPU*. The reason for this is the mathematical fact that only three vertices can describe a plane unambiguously. An example object can be seen in the figure below (Fig 5 on page 9).

### 2.2.2 **Material**

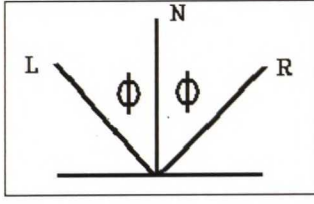
In 3D graphics a material consists of a number of material parameters, namely ambient, diffuse, specular, shininess and emissive. The color of a surface is determined by these parameters, the parameters of the light sources illuminating the surface and a lighting model.

Ambient is a naming for light reflecting from other surfaces. In offline (not real-time) rendering this is normally calculated with ray-tracing and radiosity methods, but in real-time rendering it is not yet possible because of the calculation time needed. Due to this ambient is only modeling the reflection and usually some constant value is given to it.

Diffuse material is totally matte lambertian reflector, which means it reflects light equally to all directions without any highlights. The intensity of the reflecting light is calculated simply by taking the dot product of normalized surface normal and normalized light direction:

$$i_{diff} = N \cdot L = \cos \phi \quad (1)$$

where both  $N$  and  $L$  are normalized. This causes the dot product to be one when the surface is viewed perpendicularly ( $\Phi = 0$ ) and zero if the angle  $\Phi > \pi/2$ , meaning that the surface is facing away from the light.



**Fig 3.** Specular reflection

Specular reflection is used when the reflection is stronger in one viewing direction, i.e., there is a bright spot, called specular highlight. This is readily apparent on shiny surfaces. For an ideal reflector, such as a mirror, the angle of incidence equals the angle of specular reflection (**Fig 3**). In the figure  $L$  is the

incoming light,  $N$  is the normal of the surface and  $R$  is the reflected light. In practice, this means that the specular contribution gets stronger the more closely aligned the reflection vector  $R$  is with the view direction. The reflection vector is calculated by:

$$R = 2(N \cdot L)N - L \quad (2)$$

where  $L$  and  $N$  are assumed to be normalized, and therefore  $R$  is normalized too. If  $N \cdot L < 0$ , then the surface faces away from the light and a highlight is not normally computed. The specular intensity is calculated using the reflection vector  $R$  by:

$$i_{spec} = (R \cdot V)^s \quad (3)$$

where  $V$  is the view vector from the surface to the viewer and  $s$  is the shininess value. Shininess represents the sharpness of specular reflection. With higher values the surface is shinier and the specular highlight area is narrower, and with lower values the specular highlight area is wider.

Emissive material, such as a lighting bulb, is giving away lighting. The emissive color is added to the material after all the other lighting calculations.

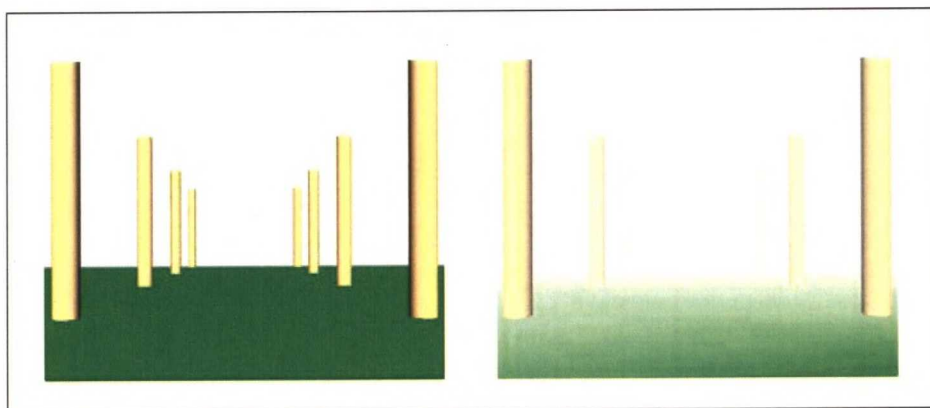
### 2.2.3 Transparency and fog

Transparency effects in real-time rendering systems are relatively simplistic and limited. Effects normally unavailable include the bending of light, attenuation of light due to the thickness of the transparent object, and reflectivity and transmission changes due to the viewing angle.

Real-time systems provide the ability to render a surface semi-transparent, blending its color with the object behind it. For this, the concept of alpha blending is needed. When an object is rendered on the screen, an  $RGB$  color and a  $Z$ -buffer depth are associated with each pixel. Another component, called alpha, can also be generated

and stored. Alpha is a value describing the degree of opacity of an object for a given pixel. An alpha value 1.0 means the object is opaque and a value 0.0 means the object is not showing at all.

Fog is a simple atmospheric effect that can be added to the final image. Fog can be used for several purposes. Since the fog effect increases along the distance from the viewer, it helps the viewer to determine how far away objects are located, and thus it increases the level of realism for outdoor scenes. Fog is often implemented in hardware, so it can be used with little or none additional performance cost. An example of fog can be seen in the next figure (**Fig 4**).



**Fig 4.** A scene rendered without and with fog

#### 2.2.4 Lighting and shading

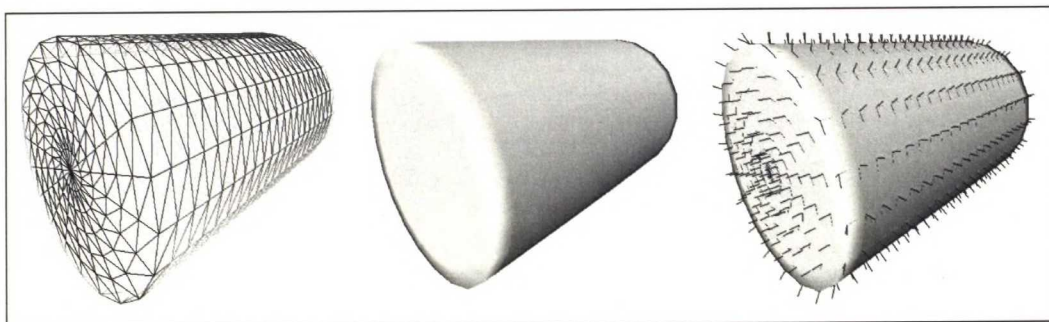
Lighting is a term used to designate the interaction between material and light sources, as well as their interaction with the geometry of the object to be rendered. Lighting equation determines how light sources interact with the material parameters of an object, and it also partly determines the colors of the pixels that a particular object occupies on the screen. This lighting model is a local lighting model, which means that the lighting depends only on light from light sources, not light from other surfaces. The total lighting intensity is the sum of ambient, diffuse and specular components (**Eq 1, Eq 3**):

$$i_{tot} = i_{amb} + i_{diff} + i_{spec} \quad (4)$$

Shading is the process of performing lighting computations and determining the colors of the pixels. There are three main types of shading, namely flat, Gouraud and Phong. These are explained below.

### Vertex normal

In order to calculate the lighting of the surface, a normal vector must be supplied with each vertex. Because every surface is composed from triangles, a surface can be totally smooth only if the triangles are small enough for one triangle to occupy only one pixel in computer screen. This however is neither possible nor practical and thus the normal of the vertex defines the direction of the heading of the surface in that vertex location. This can be visualized with the following figure (**Fig 5**), where a cylinder is rendered with wire-frame (only the edges of the triangles are visible), with smooth Gouraud shading and with vertex normals shown (short lines).



**Fig 5.** A cylinder rendered with wire-frame, with Gouraud shading and with vertex normals shown

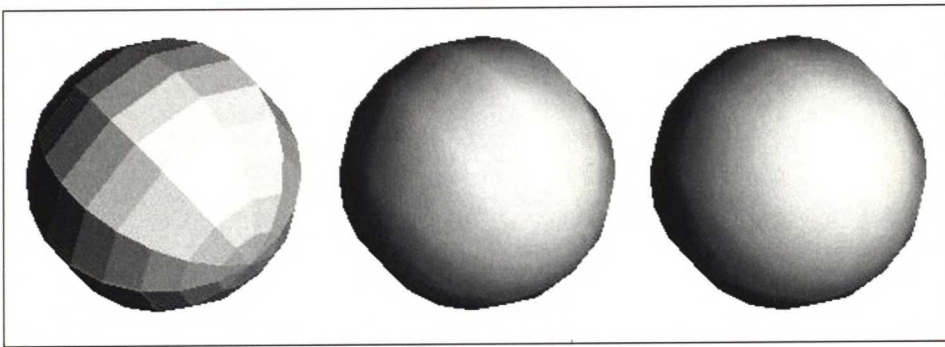
### Flat, Gouraud and Phong shading

Flat shading is the same as shading triangles having only one normal, which is perpendicular with the surface of the triangle. One color is computed for a triangle and the triangle is filled with that color. Consequently all the triangles of the object not facing onto the same direction are easily spotted from the surface (**Fig 6 on the left**). Flat shading runs fast and is simple to implement.

In Gouraud shading [GOU71] the lighting at each vertex of a triangle is determined, and these lighting samples, i.e. computed colors, are interpolated over the surface of the triangle. This produces a much smoother surface and distribution of light on the surface (**Fig 6 on the center**). Gouraud shading runs almost as fast as flat shading, since it is quite simple to implement, but still gives much better visual quality. A

problem with this technique is that the shading is highly dependent on the level of detail of the rendered objects. If the level of detail is too small, there are clearly visible artifacts, especially on the edges of the triangles as can be seen in the figure.

In Phong shading [PHO75] the shading normals stored at the vertices are used to interpolate the shading normal at each pixel in the triangle. This normal is then used to compute the lighting effect on that pixel. Since the lighting is calculated in every pixel, Phong shading is visually very accurate and close to photo-realistic looks (**Fig 6 on the right**). It is however a much slower technique to render than the other two, because per-pixel operations are more complex and much more costly than per-vertex operations.



**Fig 6.** Flat, Gouraud and Phong shading

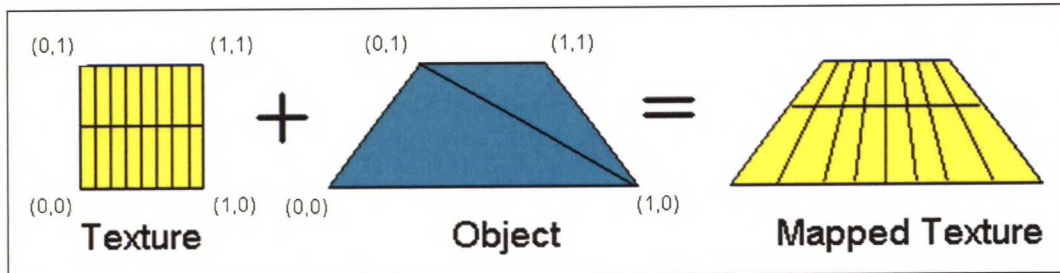
Both OpenGL and Direct3D can do flat and Gouraud shading by just enabling them, but using the Phong shading needs some extra effort. The Phong shading can be done on hardware with texture mapping techniques.

### 2.2.5 *Texture mapping*

In computer graphics, texturing is a process that takes a surface and modifies its appearance at each location using some image, function or other data source. As an example, instead of precisely representing the geometry of a brick wall, a color image of a brick wall is applied to a single polygon. When the polygon is viewed, the color image appears where the polygon is located.

To add texture mapping to the rendering process, texture coordinates must be supplied with each vertex. Texture coordinates define the location on the texture that is mapped to the given vertex. All the other locations between vertices are interpolated from surrounding vertices. In the case of a line, the interpolation is done

between the two vertices of the line, and in the case of a triangle between the three vertices of the triangle. This can be seen in the next figure (**Fig 7**), where an object has been given texture coordinates (u,v) which are mapped to same coordinates of the texture, and a mapped texture object is created.



**Fig 7.** Texture mapping

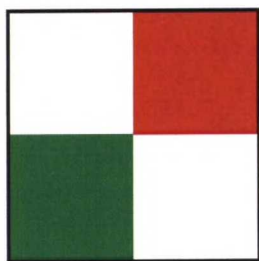
There are four different types of texture mapping, namely 1D, 2D, 3D and a special cube mapping. 1D mapping is usually used only with lines, the texture is one dimensional color table and only one texture coordinate is given with each vertex. 2D mapping is the most used one and is used with surfaces. The texture is a two dimensional color table and two texture coordinates are given with each vertex.

With 3D texture mapping, three texture coordinates must be supplied with each vertex. 3D texture mapping is used quite rarely, because it consumes so much more memory than 2D texturing. For example, with 2D texturing a normal texture bitmap with eight bits per texel (pixel of texture) and 256 texels width and height, is 256 x 256 bytes totaling 65 kB. With 3D texturing this is 256 x 256 x 256 bytes totaling 17 MB. For now, graphics accelerators in most cases have 256 MB memory at to be used by vertex data and textures. Because of this the 3D texturing is mostly used with procedural methods (more on this on chapter 4.3 on page 30).

It is also possible to apply many textures to same object, each with different texture coordinates. The procedure is called multi-texturing and the techniques used vary a lot, but usually the various textures are blended together with various weighting coefficients to produce the final texture to be displayed.

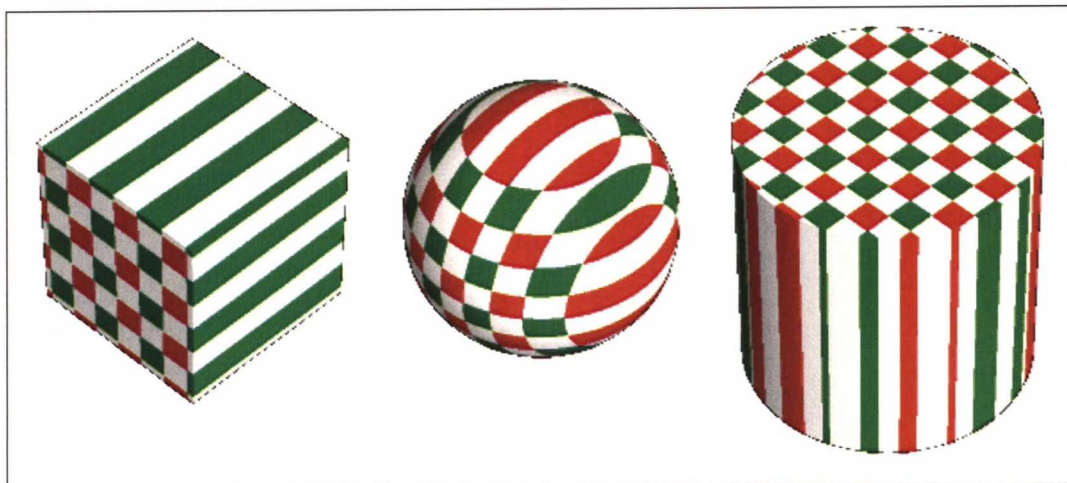
## 2D projector functions

Texture coordinates of objects can be defined manually or by some function. With big models of thousands of triangles the manual definition is very hard and time consuming. A better method is to use some projector function, which calculates the texture coordinates for each vertex according to some predefined function. In the next three figures, three different projector functions, mapping a simple texture (**Fig 8**) to three different objects, are introduced.



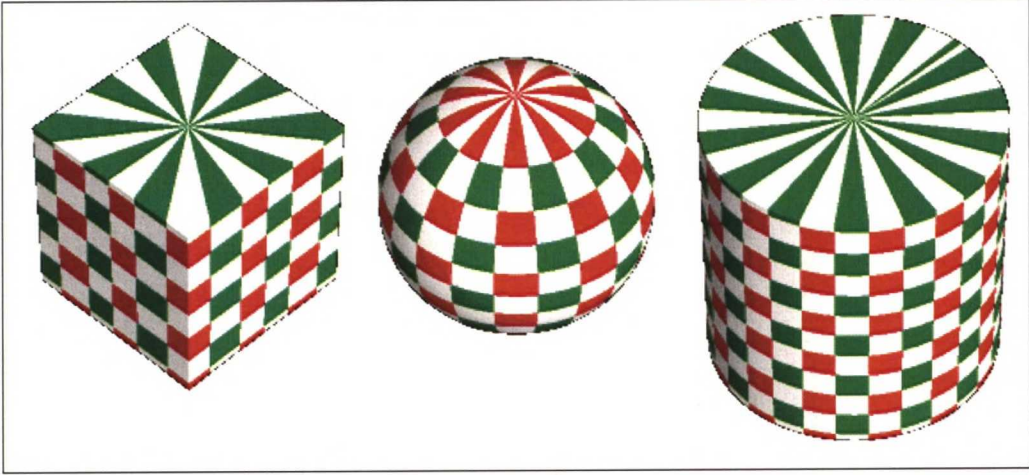
**Fig 8.** Simple texture

Planar mapping is the same as orthographic projection, meaning that the 3D object is projected on to 2D plane from some direction and the 2D texture is mapped directly to this plane. Three samples of planar mappings can be seen in the next figure (**Fig 9**). This produces one-to-one mapping on the flat surfaces, but very deformed mappings on other surfaces.



**Fig 9.** Planar mapping applied to a cube, a sphere and a cylinder

To use the cylindrical mapping, one has to decide the axis of the cylinder and radius. After this the texture is mapped onto the surface of this cylinder and it has no deformations there, but on every other shaped object it has plenty of deformations (**Fig 10**).



**Fig 10.** Cylindrical mapping applied to a cube, a sphere and a cylinder

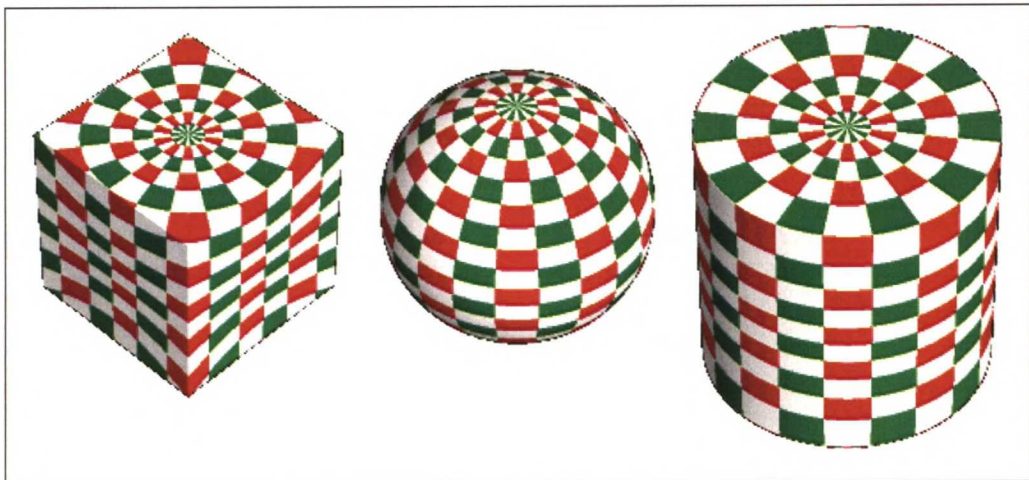
A simple version of the cylindrical mapping can be calculated by:

$$u = \frac{\arctan(z / x) + \pi}{2\pi} \quad (5)$$

$$v = y \quad (6)$$

where  $x$ ,  $y$  and  $z$  are the coordinates of the vertex, and  $u$  and  $v$  are the cylindrical texture coordinates.

The spherical mapping applies the texture on a shape of a sphere. To use it, the radius of the sphere and rotation axis must be defined. The effect of applying the spherical mapping on sphere, cube and cylinder can be seen in the next figure (**Fig 11**).



**Fig 11.** Spherical mapping applied to a cube, a sphere and a cylinder

A simple version of the spherical mapping can be constructed by calculating texture coordinate  $u$  as with cylindrical mapping (**Eq 5**) and texture coordinate  $v$  by:

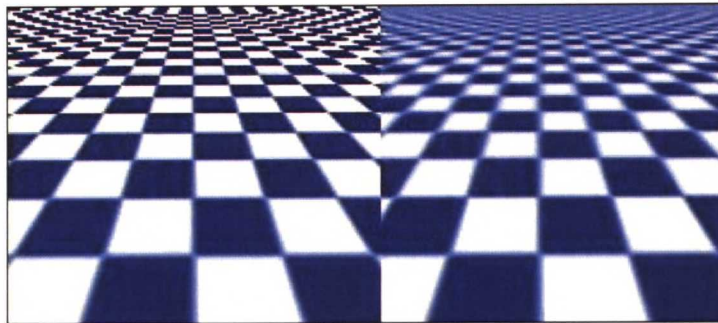
$$v = \frac{\arctan(y / \sqrt{x^2 + z^2}) + \pi}{2\pi} \quad (7)$$

where  $x$ ,  $y$  and  $z$  are the coordinates of the vertex.

## Mipmapping

When the texture image is mapped to a surface, the texture is drawn correctly on the computer screen only if the surface has the same dimensions in pixels as the texture has. This never happens, if not explicitly specified to do so, and therefore the texture image is always somewhat minified or magnified and usually viewed from a certain angle.

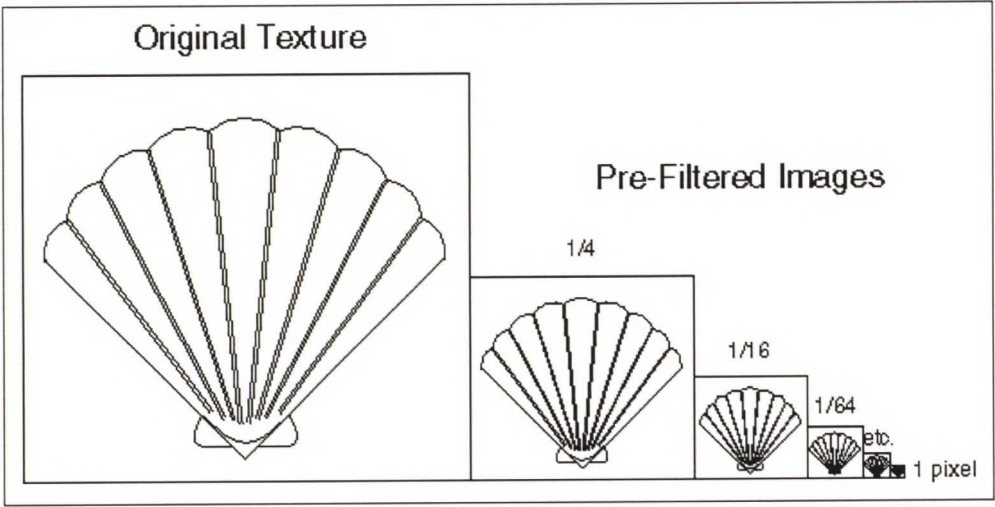
The magnification (zooming into the surface) is not a very big problem, since there is no problem of interpolating the value of a pixel from a few surrounding texels from the texture. The rendered image on the screen just looks somewhat blurred. When minimizing (zooming out of the surface) there is more than one texel per pixel and the interpolation is not possible, so some sort of filtering needs to be done. This is because if the rendering process just takes the nearest texel when coloring a pixel, an aliasing effect appears (**Fig 12 on the left**), and it can be fixed by using mipmapping (**Fig 12 on the right**).



**Fig 12.** Aliasing effect on the left, mipmapping used on the right

Mipmapping is accomplished by pre-filtering the original texture to smaller textures down to a size of one pixel, and then using these smaller copies when the surface is minimized by averaging two of them at a time. An example of a texture and its mipmaps can be seen in the next figure (**Fig 13**). The word “mip” stands for multum

in parvo, Latin for “many things in a small place” – a good name for a process in which the original texture is filtered down repeatedly into smaller images [AKE02].

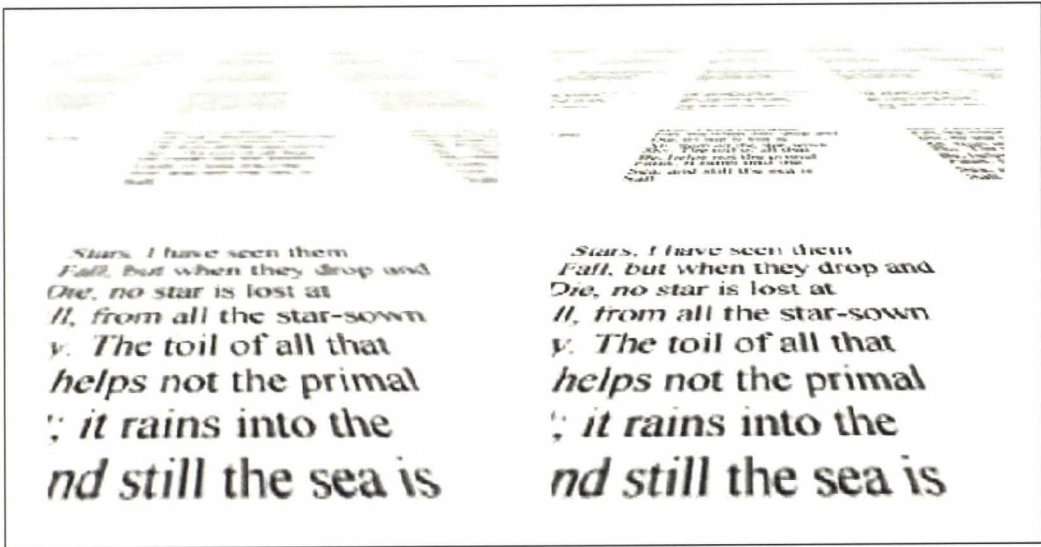


**Fig 13.** Mipmapping applied to original texture

### Anisotropic filtering

Anisotropic filtering (AF) is used to address a specific kind of texture artifact that occurs when a 3D surface is sloped relative to the view camera. A single screen pixel could encompass information from multiple texture elements (texels) in one direction, such as the y-axis, and fewer in the x-axis, or vice-versa. This requires a non-square texture filtering pattern in order to maintain proper perspective and clarity in the screen image. If more texture samples are not obtained in the direction or axis where an image or texture surface is sloped into the distance, the applied texture can appear fuzzy or out of proportion.

To correct the problem, AF uses a rectangular, trapezoidal, or parallelogram-shaped texture-sampling pattern whose length varies in proportion to the orientation of the stretch effect. With AF, textures applied to the sloped surfaces will not look as fuzzy to the viewer. A classic example is a texture with text, as the text scrolls off into the distance, its resolution and legibility both tail off. The effect of applying AF to this kind of texture image can be seen in the next figure (**Fig 14**).

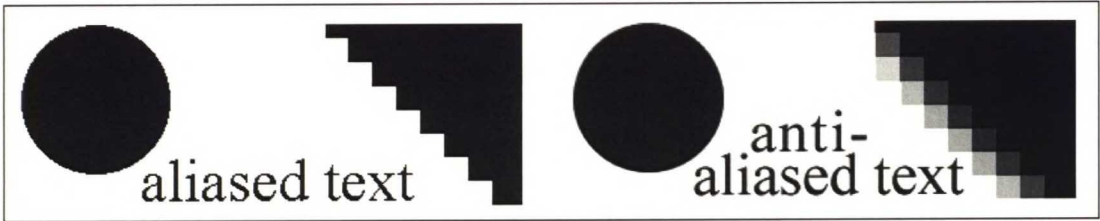


**Fig 14.** Trilinear filtering compared to anisotropic filtering

The text texture on the left is sampled with trilinear mipmap filtering and on the right with anisotropic filtering. It can be clearly seen that anisotropic filtering allows the text to be more readable in far distant. Anisotropic filtering clearly makes the visual quality of textured objects better, but this comes with the increased calculation cost, which decreases the speed of the rendering. The amount of performance hit depends on the used hardware and the number of texture samples used. [AKE02]

**2.2.6 Screen aliasing / anti-aliasing**

When a line or an edge of a triangle in a computer screen is drawn at an angle, it will often appear with jaggedness. This effect is caused by the regular pixel grid in the screen, and is called aliasing. To avoid this effect, the process of anti-aliasing paints some nearby pixels in an intermediate color or brightness. In this way the visual appearance of the line (or the edge) is smoothed out. The effect of using anti-aliasing can be seen in the next figure (Fig 15), where some graphics is drawn without and with anti-aliasing.



**Fig 15.** Comparing aliased graphics with anti-aliased graphics

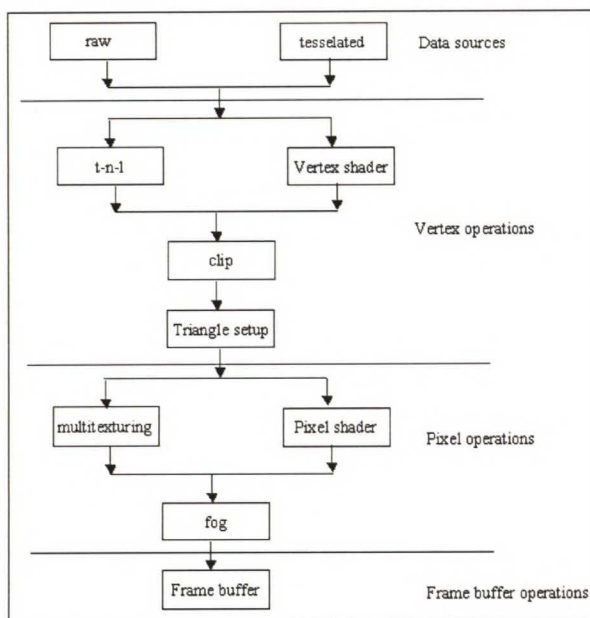
Edges of polygons produce noticeable artifacts if not anti-aliased well. Shadow boundaries, specular highlights and other phenomena where the color is changing rapidly can cause similar problems. Almost any graphics adapter can perform some sort of anti-aliasing. Mostly they use a full-screen anti-aliasing, which means the anti-aliasing is performed at the end of the rendering process to the pixels of the frame buffer. Anti-aliasing always slows down the rendering process, and the quality and speed depend on the algorithm used. The general strategy of screen-based anti-aliasing is to use a sampling pattern for the screen and then weight and sum the samples to produce a pixel color,  $p$ :

$$p(x, y) = \sum_{i=1}^n w_i c(i, x, y) \quad (8)$$

where  $n$  is the number of samples taken for a pixel. The function  $c(i, x, y)$  is a sample color and  $w_i$  is a weight, in the range  $[0,1]$ , that the sample will contribute to the overall pixel color.

## 2.3 3D pipeline

*CPUs* normally have only one programmable processor. In contrast, *GPUs* have at least two programmable processors, the vertex processor and the pixel processor,



**Fig 16.** Direct3D 8.0 pipeline flowchart

plus other non-programmable hardware units. The processors, the non-programmable parts of the graphics hardware, and the application are all linked through data flows, which are called the pipeline. In the previous figure (Fig 16) is a simplified pipeline of the Direct3D version 8.0. The pipeline is divided into four parts, namely data sources, vertex operations, pixel operations and frame buffer operations.

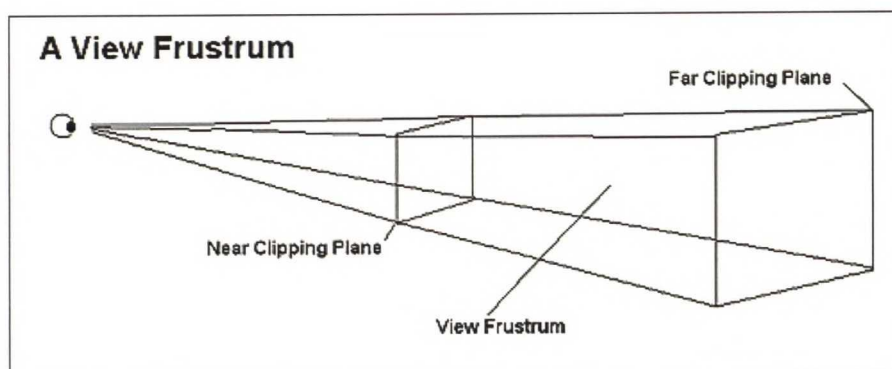
Data source is the application, which can offer the geometric data in a form of polygons, in which case they are tessellated into triangles inside the driver, or in a form of already tessellated triangles. Most graphics adapters transform even the lines into two distinct triangles [MIC04d]. After this the triangles are sent to vertex processor one vertex at a time.

### 2.3.1 *Vertex operations*

Vertex operations can be done with two distinct paths. A few years ago there used to be only one path, the fixed transform and lighting (*T-n-L*) pipeline, but today if the *GPU* is programmable, the fixed transform and lighting can be replaced with the vertex shader (more about this in chapter 4.4.1 on page 33). This part of the pipeline is responsible for transforming the vertices from model space to view space, and calculating the lighting effect on each vertex.

Model space is a three dimensional Cartesian coordinate system, where an object or world has an origin, to which all the vertices are related to. View space is also three dimensional Cartesian coordinate system, where the camera is located at the origin looking along the *z*-axis and usually *y*-axis is pointing upwards. Lighting calculations are done per vertex from the material properties of the vertex and from the lights enabled at the scene. If the programmable vertex shader is used, both these operations must be programmed inside it.

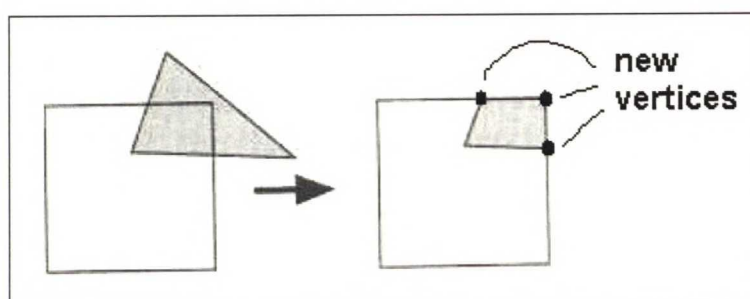
After the transform and lighting are done, rendering system performs projection, which transforms the view space into a unit cube with its extreme points at  $(-1,-1,-1)$  and  $(1,1,1)$ . There are essentially two projection methods, namely orthographic (also called parallel) and perspective projection. The view volume of orthographic viewing is normally a rectangular box and the orthographic projection transforms this view volume into the unit cube. In the perspective projection, the farther away an object lies from the camera, the smaller it appears after projection. The view volume of the perspective projection is called view frustum (**Fig 17**).



**Fig 17.** View frustum when using perspective projection

Although these transformations transform one volume into another, they are called projections because after display the z-coordinate is not stored in the image generated (instead it is stored in a z-buffer). In this way, the models are projected from three to two dimensions.

Only the primitives wholly or partially inside the view volume need to be passed on to the rasterizer stage, which then draws them onto the screen. A primitive lying totally inside the view volume will be passed on to the next stage as it is. Primitives totally outside the view volume are not passed on further, since they are not rendered. The primitives partially inside the view volume require clipping. Due to the projection transformation, the primitives are clipped against the unit cube. In clipping all the parts of the triangles and lines outside the unit cube are discarded and a new vertex is inserted at every clipping point (**Fig 18**). After clipping the vertices are sent to rasterizing stage for pixel operations.



**Fig 18.** Clipping a triangle

### 2.3.2 Pixel operations

Given the transformed and projected vertices, colors and texture coordinates (from the geometry stage), the goal of the rasterizer stage is to assign correct color to the

pixels to render an image correctly. This is the conversion from two dimensional vertices in screen space, each with a z-value, one or two colors and possibly one or more sets of texture coordinates, into the pixels on the screen.

This stage is also responsible for resolving visibility. This means that when the whole scene has been rendered, the color buffer should contain the colors of the primitives in the scene which are visible from the point of view of the camera. For most graphics hardware, this is done with the z-buffer (also called depth buffer) algorithm. For each pixel the z-buffer stores the z-value from the camera to the currently closest primitive. If a new z-value is smaller than the z-value in the z-buffer, the primitive being rendered is closer to the camera than the previous primitive at that pixel. Therefore the z-value and the color of that pixel are updated with the z-value and color from the primitive being drawn.

The z-buffer algorithm allows the primitives to be rendered in any order, which is why it is so fast. However, partially transparent primitives cannot be rendered in just any order. They must be rendered after all opaque primitives and in back-to-front order, which makes it much slower operation.

### **2.3.3 *Frame buffer operations***

After rasterization is completed, a set of frame buffer operations can be performed with the final image. Another name for this is the accumulation buffer, which was first introduced to real-time graphics by Haeberli and Akeley [HAE90]. In this buffer, images can be accumulated using a set of operators. For example, a set of images showing an object in motion can be accumulated and averaged in order to generate motion blur. Other effects that can be generated include depth of field, anti-aliasing and soft shadows.

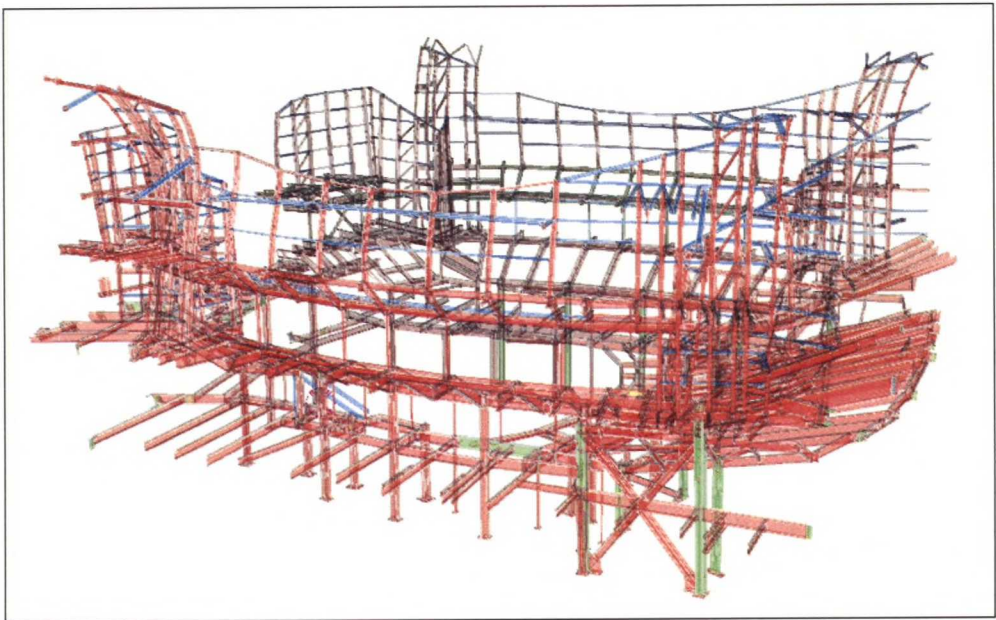
To avoid the human viewer from seeing the primitives as they are being rasterized and sent to the screen, a method called double buffering is used. This means the rendering of a scene takes place off-screen in the back buffer. Once the back buffer is rendered, the contents of it are swapped with the contents of the front buffer which was previously displayed on the screen.

### 3 Research Problem

In this chapter a deeper and more precise meaning is given for the problems introduced in the first chapter.

#### 3.1 Adding visual information to the model

The models designed and constructed with Tekla Structures are coming larger all the time. The consequence of this is the massive amount of information stored in each snapshot of the model. A good example of this can be seen in the figure below (**Fig 19**), where one can find an image of the Disney Concert Hall modeled with Tekla Structures and located at Los Angeles, California.



**Fig 19.** Disney Concert Hall modeled with Tekla Structures

In the image one can notice that there are hundreds of steel members connected to each other. Members are connected together with other structural objects such as bolts, welds and steel plates. Each member contains physical information (geometry, material and finishing) and also a lot of other information about the structure, for example the type of the connection. None of this is shown in the image. When models are coming more and more complex, the needed information on the computer screen rises. This means the modeling is more precise, there are different kinds of information and there are more members on the model. From here we come to the

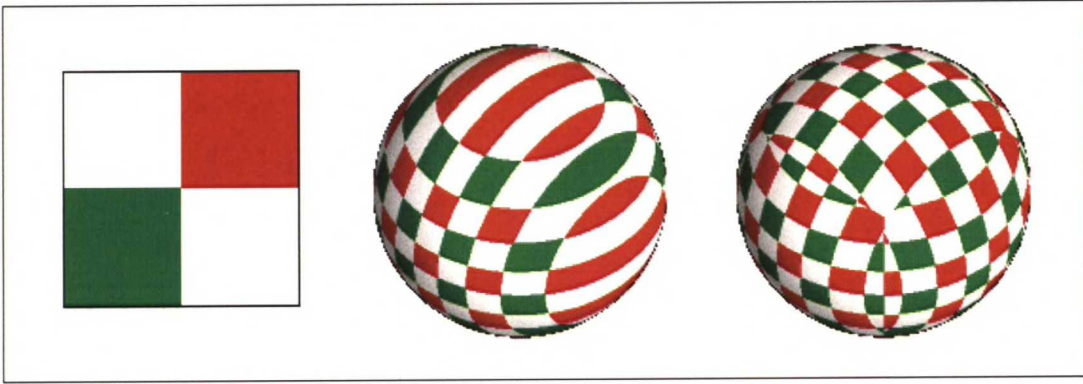
first research question of this thesis, which means that we are finding solutions to add as much information as possible to the model. It must still be viewable by the user in a way that the information is comprehensible and the graphics of the model looks good.

One aspect of this problem is to find out how to represent materials that are needed in structural engineering. These materials include the natural building materials, for example steel, concrete, wood and stone. And another group of materials is consisted of different hatches, used in CAD software, representing some defined material and visualized with line drawings.

## **3.2 Modeling materials efficiently**

The second problem is connected to technical elements. Since there are only some different colors and transparency used with the current materials in Tekla Structures, more information can be added by increasing the number of possible materials in the model. This could be accomplished with the techniques discussed in the next chapters, but since there are so many different techniques and each of them has some pros and cons, the different techniques must be studied and one must decide which ones are suitable for different material definitions. Because the models are large and they must be viewable in real-time, one must continuously bear in mind that the techniques used must use as little memory and be as fast to render as possible.

Another technical problem concerning the new material definitions is the binding problem. This can be stated with the following question. If there is a two dimensional picture of some material, how can it be wrapped around a three dimensional object smoothly with as little deformations and discontinuities as possible? A simple example of these problems can be seen in the next figure (**Fig 20**).



**Fig 20.** A simple texture mapped to a sphere with planar and cubic mappings

In the figure one can see a simple texture (on the left) mapped to a sphere with simple mapping functions; a planar mapping (on the center) and a cubic mapping (on the right). When using the planar mapping to the sphere, the texture looks good on one side but extremely stretched on the other side. When using the cubic mapping the stretching problem is much smaller, but now there are discontinuities where the cubic mapping changes face.

### 3.3 Editing materials easily

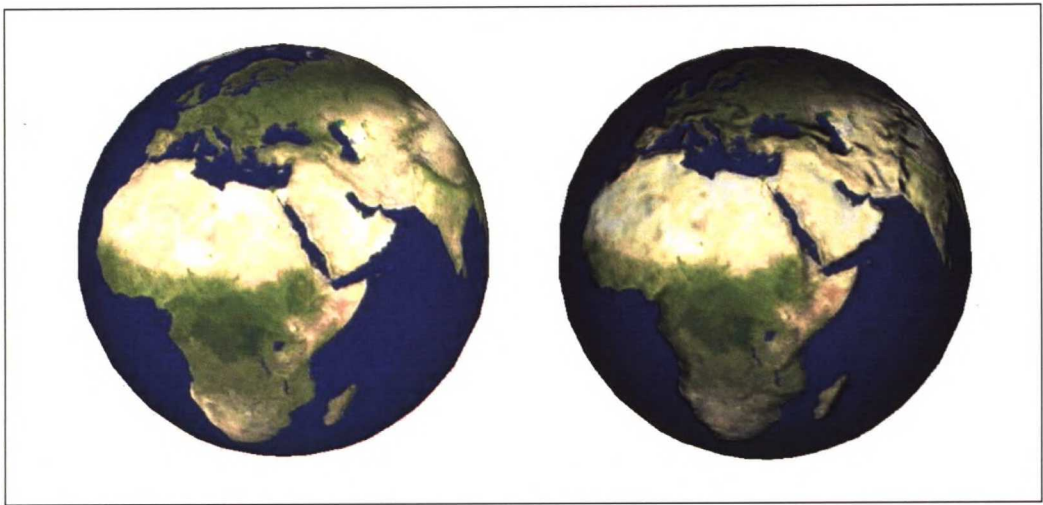
There is one last thing that must be kept in mind when finding solutions to the problems described above. No matter how great some technique would be to solve the problems above, there is no point of using it if it's totally too complicated. This means the solution must be simple enough, so that the user, who designs the new material, can do it without being an expert in the field of 3D technology. The solution should also be reproducible so that as many materials as possible could be constructed with the same technology.

# 4 Review of the Most Used and New 3D Graphics Technologies

In this chapter the advanced 3D rendering techniques that are widely used or are new and might be widely used in the future, are covered. The amount of these techniques is huge, and thus the ones that mostly interact with the problem statements in the previous chapter are selected.

## 4.1 Bump mapping

Bump Mapping is a technique used to give an object more surface detail without increasing the triangle count. It makes the surface of a smooth polygon appear irregular or “bumpy” (**Fig 21**). These irregularities are constantly re-calculated when the object is rotated in front of the camera. The technique was first invented by James Blinn in 1978 when he introduced the concept of wrinkled surfaces [BLI78]. After this pioneering work many other bump mapping techniques have been developed and today they are widely used in many different applications. In this chapter the concepts of bump mapping are introduced and an introduction to some of the most common techniques is given.



**Fig 21.** A sphere rendered with world texture and bump-mapped world texture

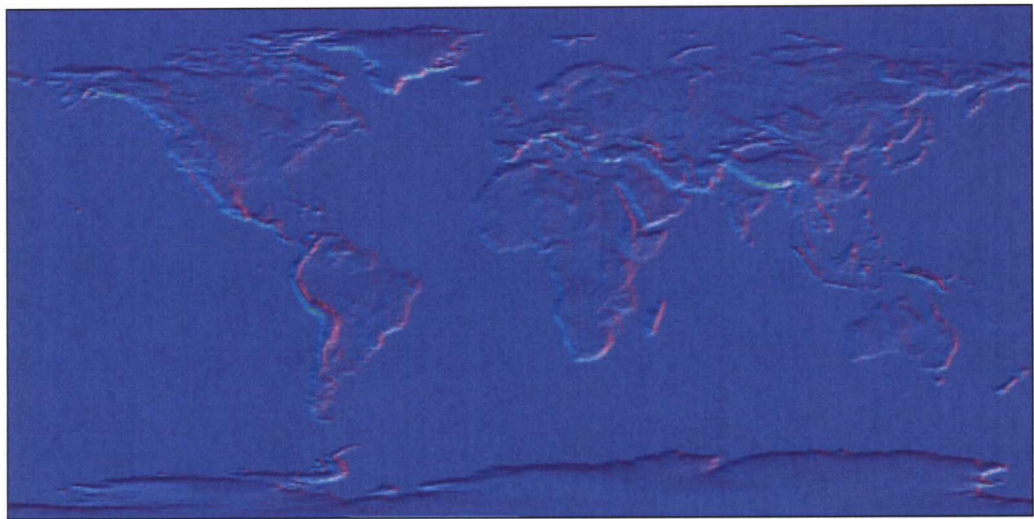
A bump map is a texture map containing the surface information that will be applied into a 3D model. A bump-mapped rendered surface is obtained by combining a base texture with this bump map texture. The surface information can be given in several

forms. One example format is elevation or height map, which stores a height delta value per texel. This texture map is generally stored in a grayscale format, since only one byte is required for this information (**Fig 22**).



**Fig 22.** An elevation map of the world

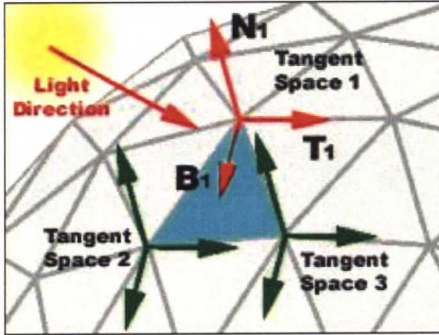
Another example is a normal map which contains the normal vector for each texel in the texture map. The *RGB* data is in this case used to encode the 3D coordinates of the normal vectors (**Fig 23**). Usually these both surface maps are created offline before rendering and used in real-time rendering to produce the bump mapped rendered surface with the selected bump mapping technique. The two most common ones are explained below. [IMA00]



**Fig 23.** A normal map created from the elevation map

### 4.1.1 Tangent space

In order to create bump mapped surface from the elevation or normal map, a light vector ( $L$ ) must be calculated for each vertex of the object and for each light in the rendering scene. Light vector is a vector pointing from the vertex to the light, so it must be calculated only once if directional light is used, and separately for every vertex if point light is used.



**Fig 24.** Tangent space in three vertices (© Imagination Technologies Ltd, 2000)

In order to obtain the light vector affecting each vertex in the object, surface orientation has to be known in every vertex. This is accomplished by adding two other vectors to the vertex normal to define a cartesian space. The normal ( $N$ ), a tangent vector ( $T$ ) and a second tangent, also called binormal vector ( $B$ ), form what is called a “tangent space” (**Fig 24**).

Tangent space is basically a set of vectors used to define a local coordinate system. It is convenient to choose the tangent axis to be parallel to the texture  $u$  direction, and compute the binormal axis to be perpendicular to both tangent axis and the normal vector. Next the light vector must be rotated into tangent space at each vertex. This is done by multiplying the light vector with the matrix constructed from the three vectors of the tangent space:

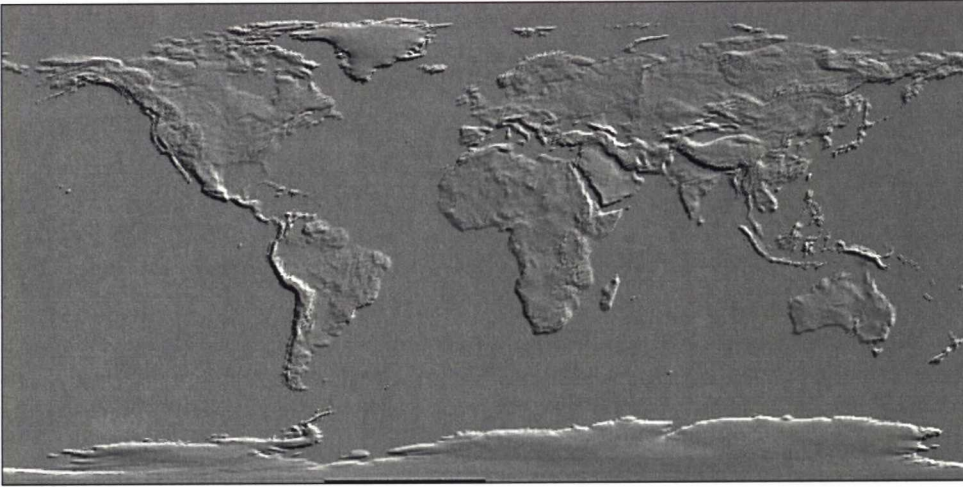
$$L_{rotated} = \begin{bmatrix} L_x & L_y & L_z \end{bmatrix} \times \begin{bmatrix} T_x & N_x & B_x \\ T_y & N_y & B_y \\ T_z & N_z & B_z \end{bmatrix} \quad (9)$$

Finally the direction of the light in each vertex in its own tangent space is known, and the information can be used in bump mapping calculations.

### 4.1.2 Emboss bump mapping

Emboss bump mapping is the simplest existing form of real-time bump mapping. It is quite simple to implement and works in the following way. The elevation map buffer (Fig 22 on page 25) is copied to another buffer and shifted in some amount to

the direction of light. Next the new buffer is subtracted from the original buffer and the result is stored. In the figure below, one can see an example result buffer where the light is coming from top-right corner (**Fig 25**). Finally the emboss map is blended with the original texture and the result is applied to the 3D object.

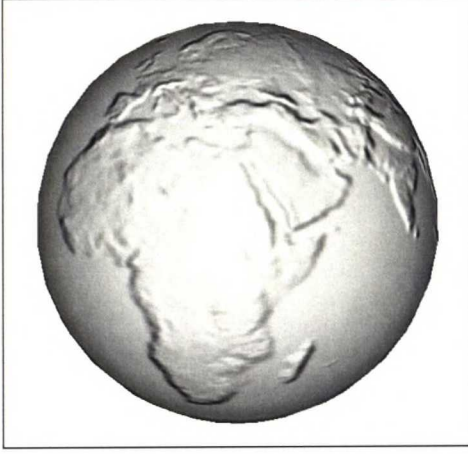


**Fig 25.** An emboss map created from the elevation map

Emboss bump mapping can be used in real-time with hardware that supports multi-pass rendering, but it requires support for multi texturing and three texturing units if it's required to be done in one-pass. Multi-texturing is supported by almost every modern *GPU* (under five years of age), but three or more texturing units have been available for only a few years. Also the bumpy effect created with emboss bump mapping is not visually very convincing, especially when the light hits the surface at a low angle. More history and mathematics about emboss bump mapping can be found from Schlag's article [SCH94].

#### **4.1.3 Dot product bump mapping**

Because the other bump mapping techniques were visually not very good looking, it was probably Mark Kilgard from NVidia who first invented the concept of dot product bump mapping [KIL00]. This makes sense, since NVidia was the leading *GPU* designer and manufacturer at that time. It's much easier to develop some new techniques when you can implement them in hardware at the same time.



**Fig 26.** Dot product bump map created from the normal map and applied to a sphere

The idea of dot product bump mapping is very simple and is based on the dot product of two vectors. If the vectors are normalized (length is exactly 1), the dot product of these two vectors is between -1 and 1. Most interestingly it's 1 if the two vectors are parallel and 0 if they are perpendicular. This result is used by taking the dot product of the light vector  $L$  (chapter 4.1.1 on page 26) and the value  $N$  of the normal map (Fig 23 on page 25) at each pixel,  $p_d(x,y)$ :

$$p_d(x,y) = N \cdot L \quad |N| = 1, |L| = 1 \quad (10)$$

Since both vectors are in the same tangent space, the result indicates the amount of diffuse light intensity in that pixel (**Fig 26**). Finally the dot product bump map is blended with the original texture and the bumpy surface is achieved (Fig 21 right, on page 24).

Dot product bump mapping produces quite convincing results and is today the most used bump mapping technology. Like emboss bump mapping it requires at least three texturing units to be done in one-pass and additionally a special hardware unit which can compute the dot product quick enough. Almost any new *GPU* can do this.

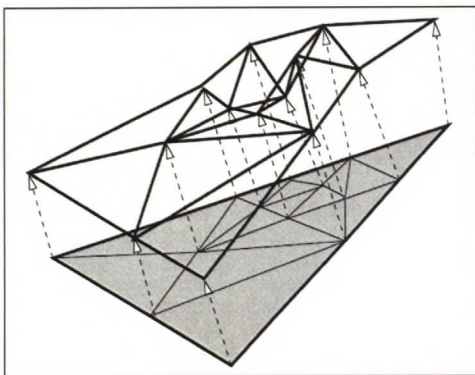
## 4.2 Displacement mapping

Displacement mapping is a powerful technique for adding detail to three dimensional objects. It was first mentioned by Cook [COO84] as a technique for adding surface detail to objects in a similar manner to texture mapping. While bump mapping gives the appearance of increased surface complexity, displacement mapping actually adds surface complexity resulting in correct silhouettes (**Fig 27**). A major benefit of displacement mapping is the ability to use it for both adding surface detail to a model and for creating the model itself. For example all the detail required to model a piece of terrain can be stored in a displacement map and a flat plane can be used for the base surface.



**Fig 27.** Bump mapping on the left, displacement mapping on the right (© Microsoft Research Asia, 2003)

Another way to process displacement mapping is to think it as a method of geometry compression. A low-polygon base mesh is tessellated in some way. The vertices created by this tessellation are then displaced along a vector which is usually the

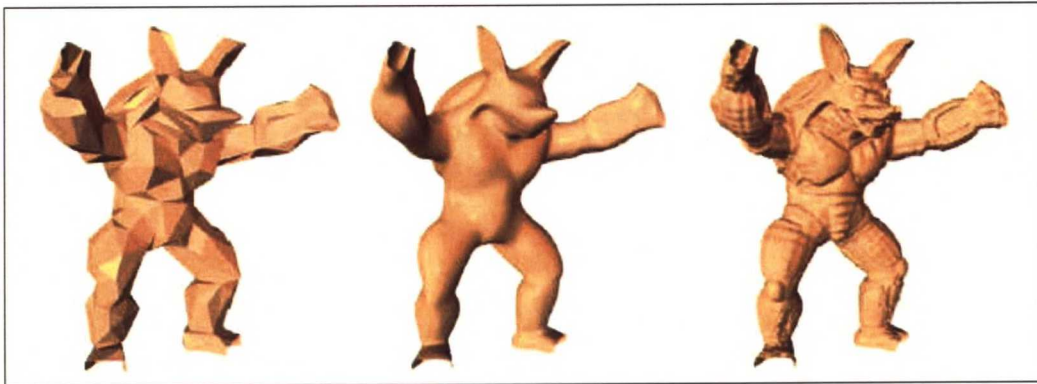


**Fig 28.** Displaced vertices created from the vertices of the surface along the normal

normal of the vertex. The distance they are displaced is looked up in a 2D map called the displacement map. This is visualized in the figure on the left (**Fig 28**). The vertices of the flat tessellated surface are displaced with different amounts along their normals, which produces the effect of displacement. A more advanced view-dependent method can be found from Wang's article [WAN03].

Displacement mapping can be done on the *CPU* since it just involves moving vertices along normals, but highly tessellated meshes can be expensive to send through the bus, and a major point of displacement mapping is to avoid that bus traffic. Hardware can accelerate this process by providing a tessellation unit, which generates triangles on the simpler surface. For each of the vertices on these triangles, the height field is sampled and an interpolated height is provided to the vertex shader. The vertex position is then shifted by this height along the vertex normal. Such functionality is supported in DirectX 9. [AKE02]

Even more compression can be archived by using a technique called displaced subdivision surfaces, which was introduced by Lee, Moreton and Hoppe [LEE00]. Subdivision surfaces is a method for compressing the geometry by storing only a control mesh (**Fig 29, left**), which is then subdivided in real-time by tessellating the polygons with some predefined algorithm to produce a smooth version (**Fig 29, center**). After this, more details are added by using displacement mapping technique (**Fig 29, right**).



**Fig 29.** Control mesh, subdivided mesh and displaced mesh (© Lee, Moreton and Hoppe, 2000)

### 4.3 Procedural texturing

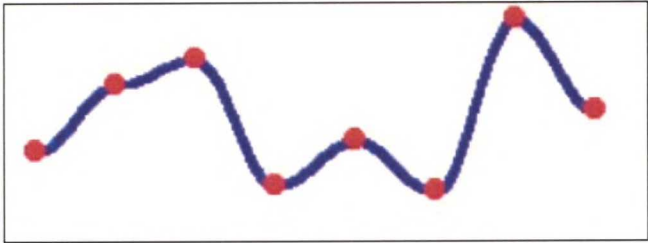
Procedural texturing is quite a new method for producing textures with minimal memory and bandwidth consumption. The ideas were first introduced to 3D graphics by F. Kenton Musgrave and Ken Perlin in the mid 80s, and today the procedural texturing is widely used in real-time graphics. It's a common naming convention for the techniques creating the texture procedurally, which means that the texture is not a stored bitmap but instead it's created in runtime with some mathematical function. Another advantage for using procedural texturing is that it helps the content creator who designs the materials, since (s)he doesn't have to do so much work applying the material to an object. [EBE02]

Procedural texturing is mostly used for materials having some repeating but still somewhat random pattern, like marble, stone or wood. They all look the same when viewing them from a distance but have some random pattern when viewing at a close range. The pattern still varies depending on the viewing location. It can't be used for creating textures with small details or varying patterns, like synthetic photographs or exact detailing.

### 4.3.1 Perlin noise

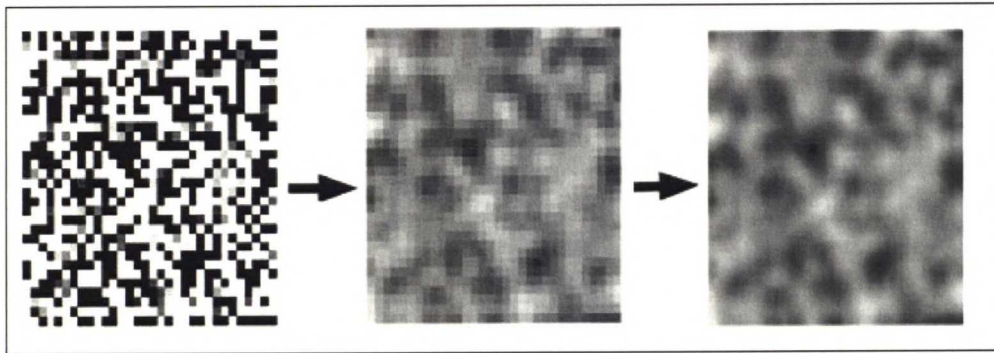
One of the most used procedural techniques is the concept of noise. It was invented by Ken Perlin in the mid 80's [PER85] and today the technique is widely used in almost every game and motion film using computer generated animation or materials. Procedural noise provides a controlled method of adding randomness to various graphical concepts, including textures, bump maps, animation and just about everything. [SPI03]

Noise is not just random values which are non-linear, but instead the noise consists of random values varying smoothly. Another important feature is that the noise values are actually pseudorandom, so with the same seed values it produces the same output results. It can be constructed without obvious repeating pattern continuing endlessly to all directions. In the figure below (Fig 30) a one dimensional noise is drawn with red points marking noise values and blue interpolated smooth curve connecting the values.



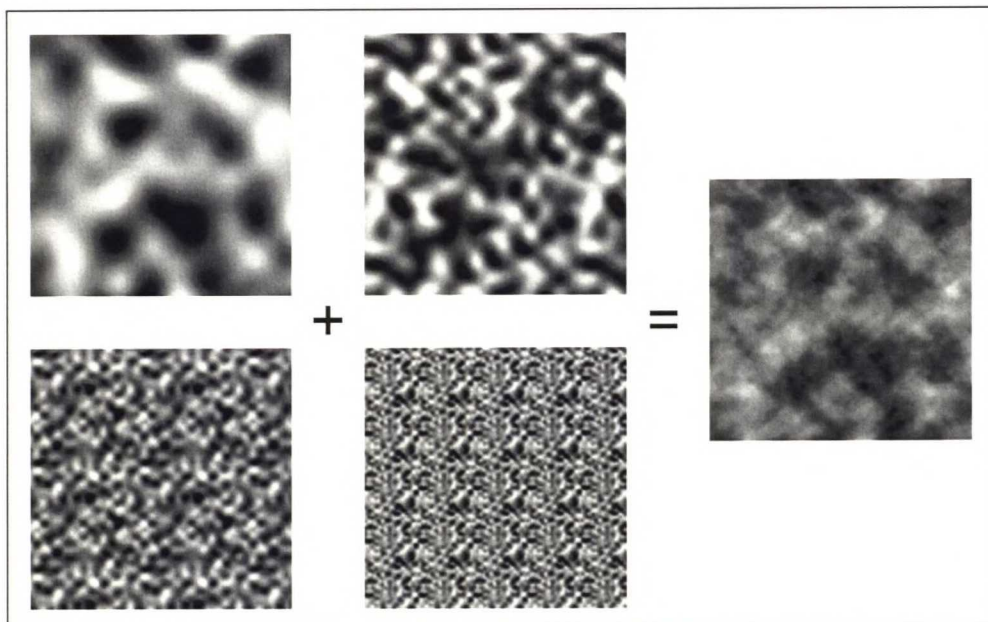
**Fig 30.** One dimensional noise

The noise can be constructed with various methods. The simplest way is that at first one makes an array of salt and pepper noise and blurs it with averaging the values as many times as needed and finally up-sampling to receive a smooth result (Fig 31). This however is a very time consuming task and therefore suitable only for offline content creation.



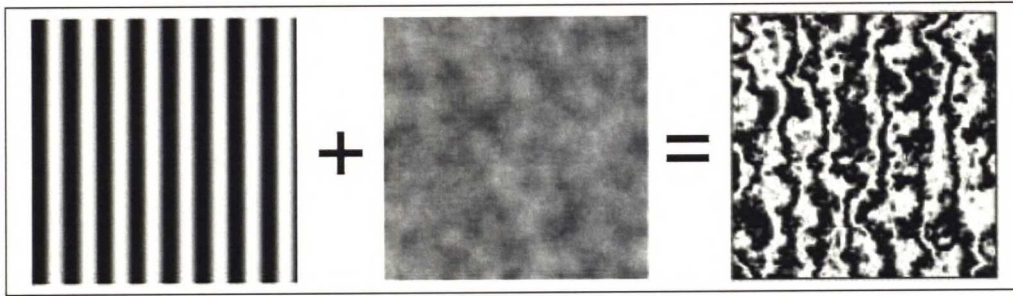
**Fig 31.** Salt and pepper noise blurred and up-sampled

The noise itself is not very interesting or great looking, but it can be used as a building block to produce many natural effects that can be constructed mathematically. It can be made more usable, when many noise maps are summed and averaged. Usually the same noise function can be used to produce noise maps with different frequencies (called octaves since usually the frequency doubles between them) which are then averaged (**Fig 32**).



**Fig 32.** Taking average of noises at different octaves yields usable results

This averaging can be done in various ways, thus constructing very different noise maps with small input variable changes. The actual noise map can be also used with various methods depending on purpose. For example, a simple marble texture is constructed by taking an average of dark and light bar texture and the noise map (**Fig 33**).



**Fig 33.** Creation of procedural marble texture

The creation of the noise can be done in real-time on the *CPU* or *GPU*, or offline on the *CPU* and the choice between these depends on the purpose of using the noise. If it's done in real-time, there is no memory or bus overhead but it consumes more *CPU* or *GPU* processing time. And on the contrary, if the memory and bus traffic do not create a bottleneck, pre-calculated noise can be used and *GPU* power saved for some other tasks.

## 4.4 Programmable pipeline

The fixed-function pipeline can not be complete enough for advanced and more realistic lighting calculations. Because of this the hardware and software developers created the programmable pipeline that could be used as an alternative to the fixed-function pipeline. With it, one can use an assembler type of language to program the *GPU* to do tasks with the data sent from the *CPU* to the *GPU* pipeline. Separate programs must be created for both vertex and pixel processing and the programs can be changed between the objects. It is also possible to replace only one of these and let the *GPU* handle the other.

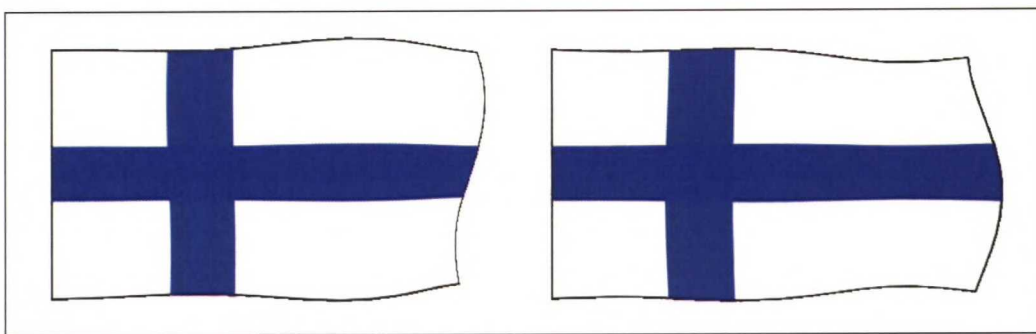
### 4.4.1 Vertex shading

Vertex shaders provide a way to modify values associated with each vertex of a triangle, such as its color, normal, texture coordinates and position. This functionality was first introduced with DirectX 8, and is also available as OpenGL extensions. The capability to perform transform and lighting of vertices on the *GPU* became available on consumer-level hardware in 1999. Up to this point it was normally done on the *CPU*. As a drawback it forces to use the basic Gouraud/Phong model for lighting, and any other variations can not be handled.

The vertex shader offers a big improvement to this situation. When the vertex shader is enabled, the hard-wired T-n-L model (Fig 16 on page 17) is no longer available. In its place the transform and lighting engine is replaced by a vertex shader unit, which executes a series of commands written by the user. Vertex shader and fixed function pipeline cannot be used simultaneously and a choice must be made which one to use. The vertex shader program is stored in a form of assembly language, though macros or higher level languages can be used to help in programming (more on this in chapter 4.4.3 on page 35).

Every vertex passed in is processed by the vertex shader program. The vertex shader can neither create nor destroy vertices and results generated with one vertex cannot be passed on to be used by another vertex. Also there is no conditional branching in the first and second generation versions of vertex shader, which means that no if, for or while statements can be used, and every instruction of the shader is executed with every vertex. Conditional branching is a part of the third generation shader version in the near future. [AKE02]

Some of the using samples of vertex shader include effects such as shadow volume creation, motion blur and silhouette rendering. Also different lens effects, such as the fish-eye lens and object definitions by defining a polygon only once and having it be deformed by the vertex shader (**Fig 34**), are possible.



**Fig 34.** Waving flag created from flat surface using vertex shader

#### **4.4.2 Pixel shading**

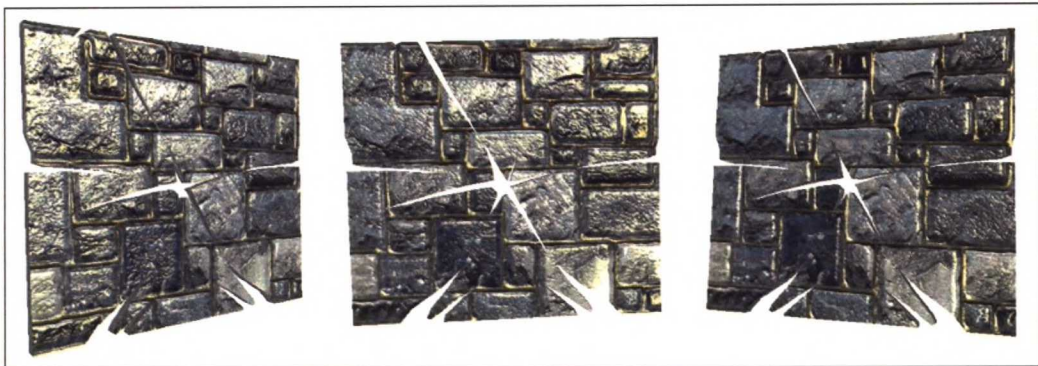
Pixel shading (also called fragment shading) takes place on a per-object, per-pixel basis during the rendering. The idea is the same as with fixed-function multi-texture pipeline, which means that a series of instructions operate on a set of constants, interpolated values and retrieved texture values to produce a pixel color and

optionally an alpha value. Pixel shader can also compute texture coordinates and then use them directly, modify the z-depth value and perform other operations not fitting into the fixed texture stage concept.

The pixel shader is an alternate part of the pipeline that can replace the multi-texturing (Fig 16 on page 17). There are three sets of inputs for pixel shaders; the interpolated diffuse and specular colors and alphas, eight constants, and four or more texture coordinates. Each of these is a vector of up to four values. A pixel shader consists of definitions of the constants, a number of texture address instructions and a number of arithmetic instructions.

The term “texture coordinates” is a little bit misleading, since the data stored can represent anything. Each texture coordinate is accessed by a texture address instruction, which treats the coordinate as a traditional lookup and filtering of a texture, as a vector or as part of a matrix. The texture coordinates themselves can also be passed through and directly accessed by arithmetic instructions. [AKE02]

The possible usage area of pixel shader is wide, but usually they are related to per-pixel effects, such as per-pixel lighting and fragment post-processing. In the figure below is an example where vertex and pixel shaders are used with textures to visualize per-pixel lighting with bump mapping (**Fig 35**).



**Fig 35.** Flat surface rendered with per-pixel lighting using vertex and pixel shaders (© ATI, 2003)

#### **4.4.3 Shading languages**

Creating assembly language programs for vertex and pixel shaders, rather than defining complex multi-texturing pipeline setups, means that editing and reading the code is easier. Even so, individual shaders for particular hardware are hard to write,

have often portability problems and can quickly become obsolete or inefficient without active maintenance to move them to newer architecture.

Although the vertex and pixel shaders are new to real-time graphics, the idea of higher level languages for shading came from Cook's ideas [COO84]. They have been used in an offline rendering software such as Pixar's RenderMan interface [PIX00]. When the real-time shading assembler language came with the new hardware, the hardware and software vendors started to develop a C-style higher level language and compiler for both Direct3D and OpenGL. The first attempt was to develop a language that would be platform independent, but it turned out to be impossible, and due to this now there are three shading languages that are similar but have some individual specialties.

### **OpenGL Shading Language (GLSL)**

The OpenGL Shading Language is based on ANSI C and many of the features have been retained, except when they conflict with performance or ease of implementation. C has been extended with vector and matrix types (with hardware based qualifiers) to make it more concise for the typical operations carried out in 3D graphics. Some mechanisms from C++ have also been borrowed, such as overloading functions based on argument types, and ability to declare variables where they are first needed, instead of at the beginning of blocks. [KES04]

The OpenGL shading language is constructed on top of OpenGL version 1.4 and since the Windows operating system is supporting only version 1.1 of OpenGL, this language is not very important in this thesis.

### **DirectX High-Level Shader Language (HLSL)**

Microsoft DirectX 9.0 contains the first release of a high-level shader language for developing and debugging shaders in a C-like language. This capability is an addition to the assembly language shader capability used to generate vertex shaders, pixel shaders and effects, which began with DirectX 8.0. The language supports many standard language features such as functions, expressions, statements, standard data types, user-designed data types, and preprocessor directives. [MIC04d]

Shader programs are compiled to assembly language of vertex and pixel shaders in real-time, which is usually done once per application runtime, and then the compiled code is used in the vertex or pixel shader. It is possible to make separate shader programs with each object, but usually this is not recommended since loading and offloading the program takes time. It is advised to render the objects in the order of materials, e.g. to render all the objects of the same material in a group, and then move on to the next material.

### **C for graphics (Cg)**

Cg is a similar language to HLSL and was created and is maintained by NVidia Corporation. Actually at the beginning they were the same language, but at some point Microsoft and NVidia decided to start developing their own languages.

Cg is a multiplatform language in a sense it can be compiled to be used for both DirectX and OpenGL. The compilation can be done in real-time or as an offline process in software compilation time. The user must choose the version of the shader which will be used. The different versions are called profiles and a new profile must be created for every new shader version. [NVI02]

## 5 Research Work

In this chapter the various techniques that have been tested during the work on this thesis are explained. The findings made during the testing are presented and a summary of the research is given.

### 5.1 Testing environment

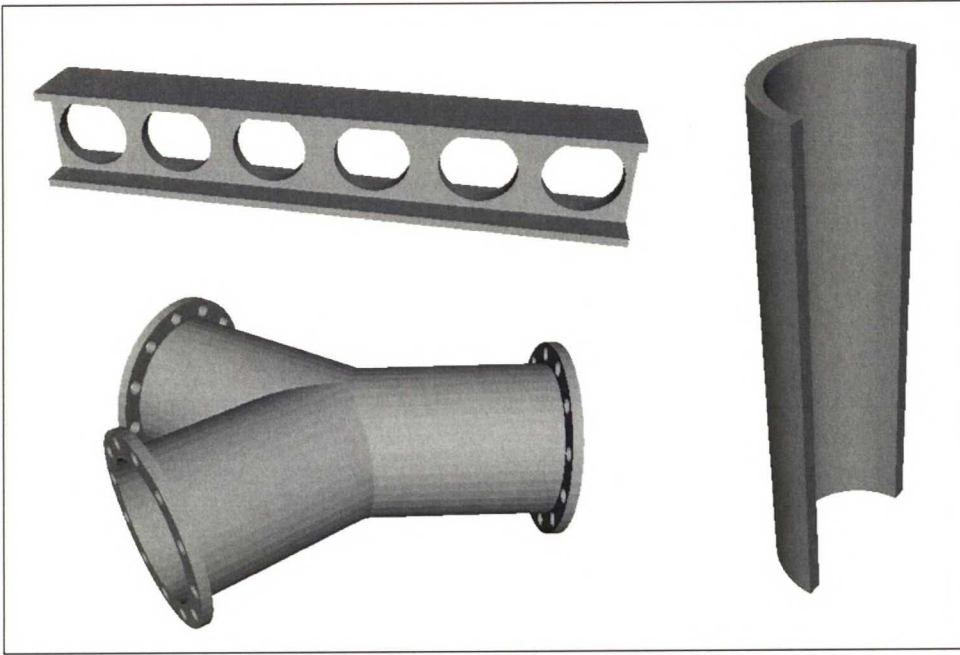
For testing different techniques a graphical application was created that was easy to use and simple to expand. It consists of a single window, in the center of which a single object is presented, and a mouse as an input device, which can be used to rotate the object and zoom in and out. The following hardware and software were used:

- Mainboard: Intel Corporation D815EEA
- Memory: 512MB SDRAM
- CPU: Intel Pentium III 864MHz
- GPU: ATI Radeon 9700 Pro 128MB
- API: OpenGL 1.1
- OS: Windows XP pro (sp1)

The hardware is not balanced, in a sense that the *GPU* is newer and much more powerful compared to the *CPU*. The consequence from this is that the *GPU* has to wait for the *CPU* in certain tasks and because of that is running idle occasionally. This however affects only on speed testing and there is a separate note on text where needed.

### 5.2 Testing various techniques

In this chapter the results of testing various surface rendering techniques are presented. Three different objects were selected for the tests to represent some of the most used object shapes in structural engineering; I-beam with holes, Y-pipe and half-pipe. These can be seen smooth shaded without color in the next figure (**Fig 36**).



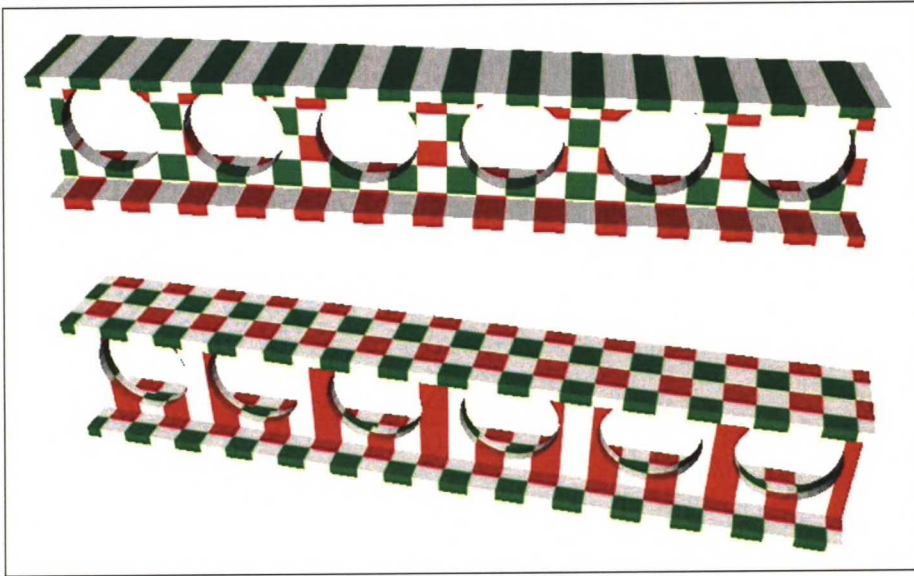
**Fig 36.** Three test objects; an I-beam with holes, an Y-pipe and a half-pipe

### 5.2.1 *Testing texture mapping projector functions*

The first test was to implement different 2D texture mapping projector functions and visualize how they would act when applied to different 3D objects. The purpose of this test was the idea it would be nice and easy to apply a texture to an object with one function that would do it quite automatically. This way there would not be any need for calculating the texture coordinates. The only tasks would be the decision of which function to use and potentially the parameters of the function (starting point, direction). Two different functions were tested, planar and cylindrical, onto three different test objects (Fig 36).

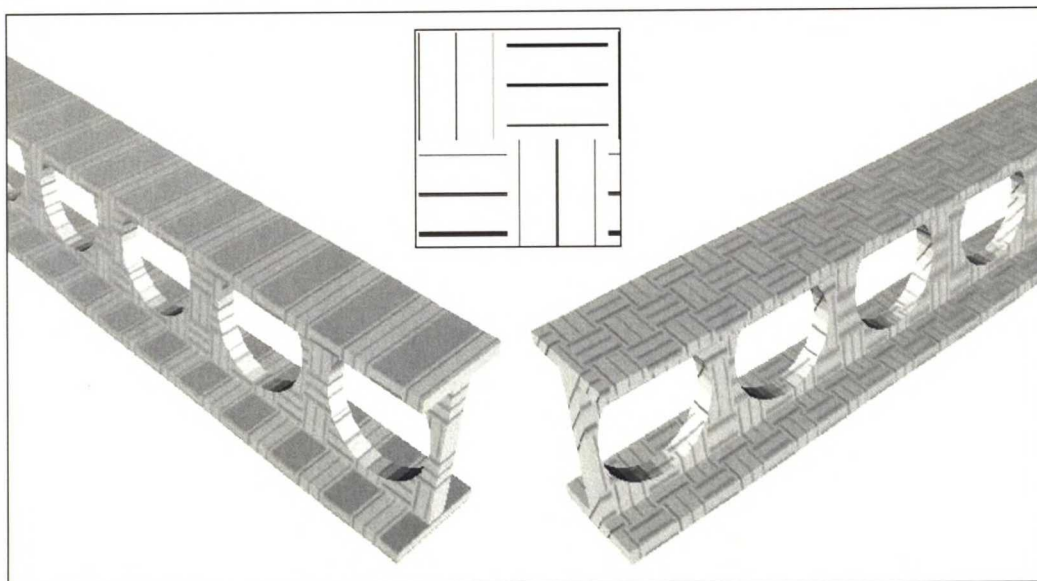
#### **Planar mapping**

As noted earlier (chapter 2.2.5 on page 12), the planar projector function is the simplest one to implement and it produces no discontinuities regardless of the shape of the 3D object. However, it can only be applied to a flat 2D surface without any deformations. If this is not a problem, it can be applied to any object as can be seen in the following figures. In a figure below (**Fig 37**) a simple texture is mapped onto the I-beam with planar mapping from aside and from top.



**Fig 37.** A simple texture planar mapped to an I-beam from two directions

As can be seen, in this case the deformations are not so annoying, since the object contains mainly flat surfaces and right angles. However, the inconvenience of the deformation depends on the texture used and the angle from which the planar mapping is applied. In the figure below, one can see a hatch texture (**Fig 38 on top**) mapped to an I-beam with planar mapping directly from aside (**Fig 38 on left**) and from 45 degree angle (**Fig 38 on right**). On the former method the hatch is not recognizable on top of the I-beam, and on the latter method it is recognizable but it has stretched into one direction.



**Fig 38.** Hatch texture planar mapped onto an I-beam from two different directions

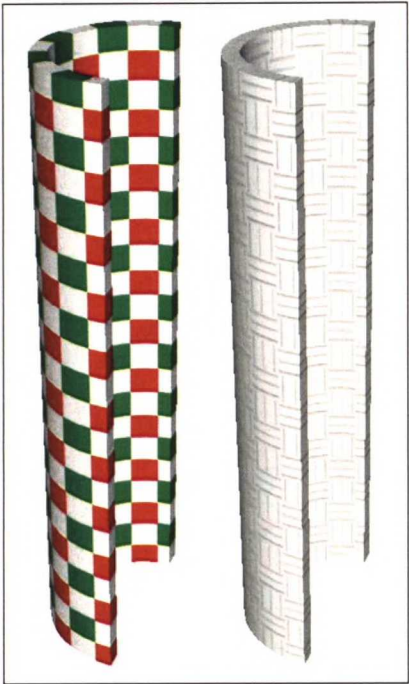
With Y-pipe this is no longer the case, since there are no flat surfaces, and because of this the flat texture becomes deformed almost in every point of the surface of the Y-pipe. This can be seen in the figure below (**Fig 39**), where the simple texture is mapped with planar function onto the Y-pipe from two different angles. The code that was created for the planar mapping can be found from **Appendix C**.



**Fig 39.** A simple texture planar mapped onto an Y-pipe from two directions

### Cylindrical mapping

With cylindrical mapping (chapter 2.2.5 on page 12) one has to decide the axis direction and location of the cylinder. This was tested with a naturally cylindrical half-pipe object. The length axis of the half-pipe



**Fig 40.** Two different textures cylindrical mapped to a half-pipe

was chosen, which guarantees that the texture image is not deformed on the surface of the object. It is however a little smaller on inside surface of the pipe compared to the outside. Two different textures were mapped to the half-pipe, as can be seen in the figure (**Fig 40**).

Cylindrical mapping can be mainly used to cylinder shaped objects, since mapping to objects shaped differently produces both discontinuities and deformations. The calculation of texture coordinates is still quite a simple procedure. The code that was created for the cylindrical mapping can be found from **Appendix C**.

### 5.2.2 *Anti-aliasing testing with Steelmark*

Full-screen anti-aliasing (chapter 2.2.6 on page 16) was chosen for testing, because it makes the rendered image smoother and lowers the amount of jaggedness at the edges of the objects. This is normally perceived as increased visual quality of the rendered image by the viewer. The down-side of this increased visual quality is that the anti-aliasing techniques are very time consuming, and due to this decrease the speed of the rendering. The purpose of this test was to measure the impact on rendering speed. Since the main testing environment (chapter 5.1 on page 38) is not balanced, the following testing hardware and software were used when testing anti-aliasing:

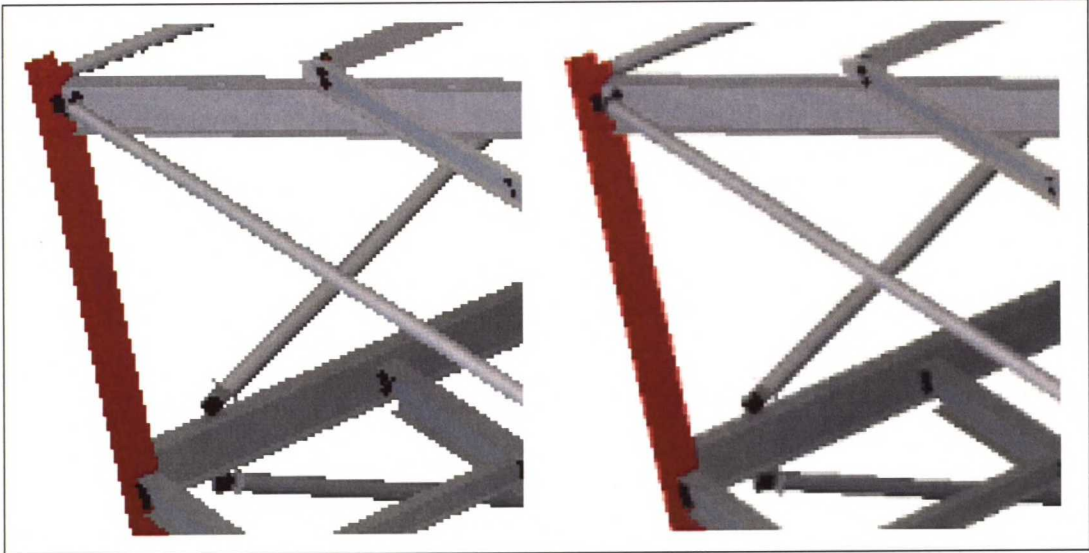
- Mainboard: Dell Latitude C810
- Memory: 512MB SDRAM
- CPU: Intel Pentium III Mobile 1133MHz
- GPU: NVidia GeForce2 Go 32MB
- API: OpenGL 1.1
- OS: Windows XP pro (sp1)

To test the impact of anti-aliasing on rendering speed, a small Tekla utility called Steelmark was used. It is a small program displaying a copy of a real 3D model on the screen with resolution of 800x600x32 (width, height, color bits) and shows a short animation by flying through the model. The rendering speed is measured by calculating the average rendering time per frame and taking an inverse from that time. This tells the average frames per second (fps) speed. The model used was chosen to be of an average size, so that the test result would be usable. The Steelmark was run without anti-aliasing and with 2x anti-aliasing and the result was that without anti-aliasing the rendering speed was 19.8 fps, and with anti-aliasing 11.6 fps. So without anti-aliasing the rendering was 71% faster. (**Appendix A**)

However, this speed difference varies a lot depending on the balance of the *GPU* and the *CPU*. For example, with the main testing environment (chapter 5.1 on page 38), enabling the anti-aliasing has no effect on rendering speed, since the *GPU* outperforms the *CPU* so clearly. And the more powerful the *CPU* is compared to the *GPU*, the more the speed difference will be between anti-aliased and aliased rendering.

Another speed test with almost the same GPU as was with the original testing environment (ATI Radeon 9700, chapter 5.1), but with balanced hardware, was also compared [SAL02]. In this test the rendering speed was measured with the Futuremark's 3DMark2001SE with resolution of 1024x768x32, but the results were similar. The speed score without anti-aliasing was reported to be 14539 and with anti-aliasing 10748. This gives 35% speed increase without anti-aliasing.

The impact of anti-aliasing on the quality of the rendered image was checked by inspecting the image visually. In the figure below (**Fig 41**) there is a zoomed area from the model where one can clearly see the difference between aliased and anti-aliased images. The difference between these two is visually more evitable when the screen resolution is smaller and the screen pixel size is larger.

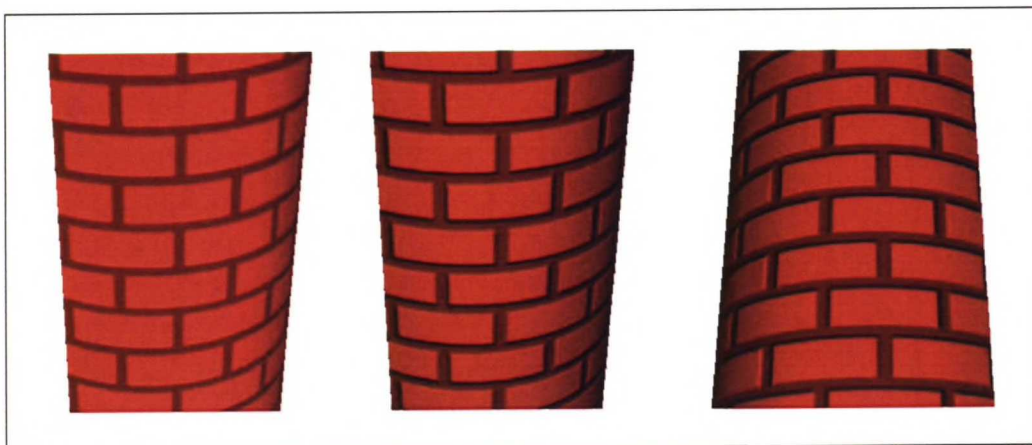


**Fig 41.** Zoomed image from the model without and with anti-aliasing

**5.2.3    *Testing dot product bump mapping***

From different bump mapping techniques (chapter 4.1 on page 24), the dot product bump mapping was chosen for testing, since it is the most used, supported and most likely the best looking bump mapping technique (as can be read from Imagination's bump mapping comparisons [IMA00]). When using OpenGL, there are ARB extensions for direct support of this kind of bump mapping, and this can be used if the underlying hardware is capable of multi-texturing and dot product between two textures.

The bump mapping was implemented with two passes (meaning the object is rendered twice and the results are combined). In the first pass the light vector from the light source to each vertex is calculated in tangent space and the dot product is taken with this light vector and the normal vector from the normal map. This produces the lighting value for each pixel, which is then modulated in the second pass with the base texture color. The results can be seen in the figure below (**Fig 42**), where the half-pipe is textured with tile-texture on the left, with bump mapping enabled on the center and from another angle on the right.



**Fig 42.** Half-pipe textured with tiles, without and with bump mapping

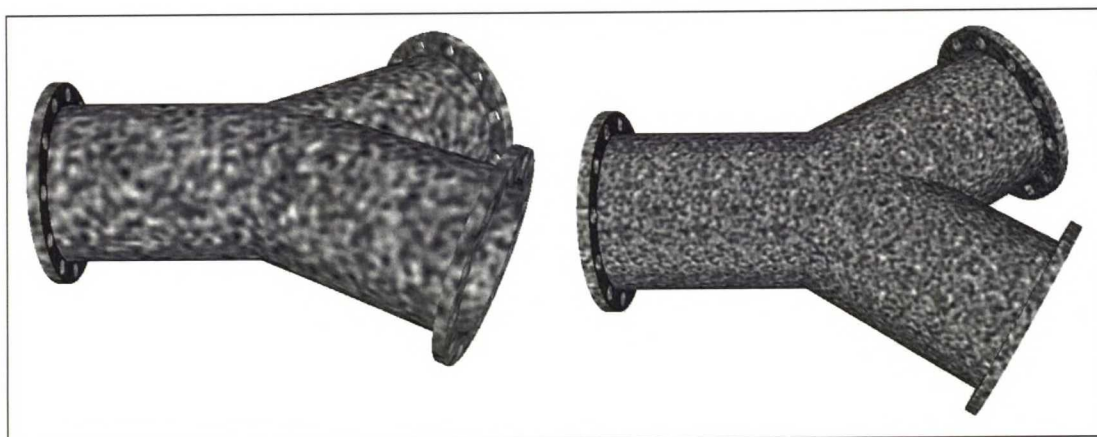
As can be seen, the bump mapped version gives a feeling of individual tiles, but this comes with the cost of incremented drawing time. Even when the dot product is supported by hardware, the bump mapped version of an object will always take quite a lot more time than the smooth object to draw and this should be taken into consideration when using it. Similar results were achieved in Imagination Technologies' bump mapping comparison tests [IMA00].

#### **5.2.4 Testing pixel shader materials using Cg**

For testing various material creations with pixel shader, the Cg language (chapter 4.4.3 on page 37) was chosen as the development environment, because it seemed it could be the future standard of shading languages. The same testing software mentioned before was modified to use Cg, and in this chapter the materials created with Cg are presented.

## Perlin noise

When creating natural materials, the perlin noise (chapter 4.3.1 on page 31) is very essential since it gives the material the feeling and look of some randomness and irregularity common to natural materials. The perlin noise was used in the testing by creating a 3D texture map with the pattern being repeatable to every direction, which assured that no discontinuities were visible. In the next figure (**Fig 43**) one can see the 3D perlin noise texture mapped onto the y-pipe with two different frequencies.



**Fig 43.** Y-pipe covered with 3D perlin noise with two different frequencies

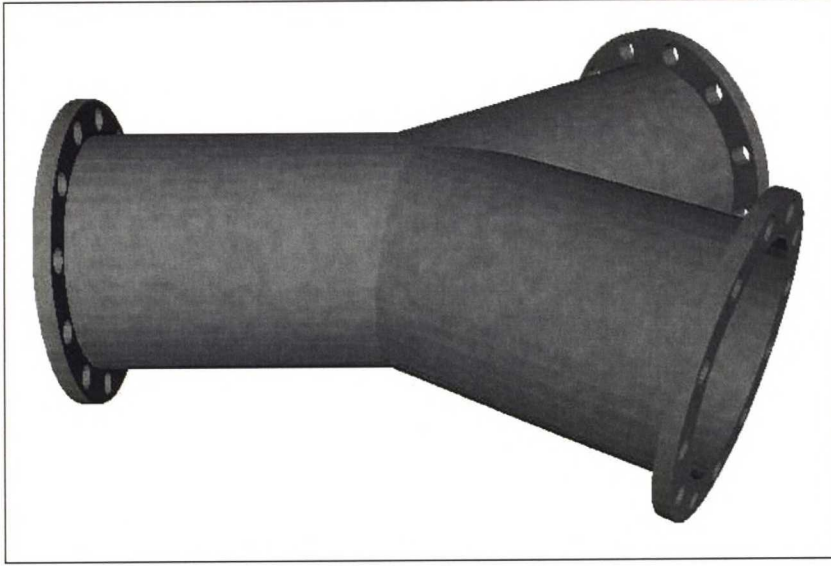
On the right the frequency is doubled compared to the left version. The perlin noise alone is not very useful, but when adding the different frequencies together with different weighting coefficient and some color, interesting results can be seen. This method is used with all the materials described below.

## Concrete

Because concrete is the basic building material in structural engineering, it was chosen to be the first material tried to simulate. It was constructed very simply by just adding three different frequencies of perlin noise with different coefficients, and blending them with grey color. The largest weight was given to the basic frequency, half the weight to the higher frequency and one quarter weight to the highest frequency:

$$p(x, y) = C_v \left( 1 - C \frac{N_{f1} + 0.5N_{f2} + 0.25N_{f4}}{1.75} \right) \quad (11)$$

where  $p(x, y)$  is the resulted pixel color,  $C_v$  is the color value from vertex shader,  $N_f$  is a noise value from three different frequencies and  $C$  is a constant for selecting the contrast between dark and light areas within the concrete. The results of this can be seen in the figure below (**Fig 44**) and the pixel shader code for creating concrete can be found from **Appendix B**.



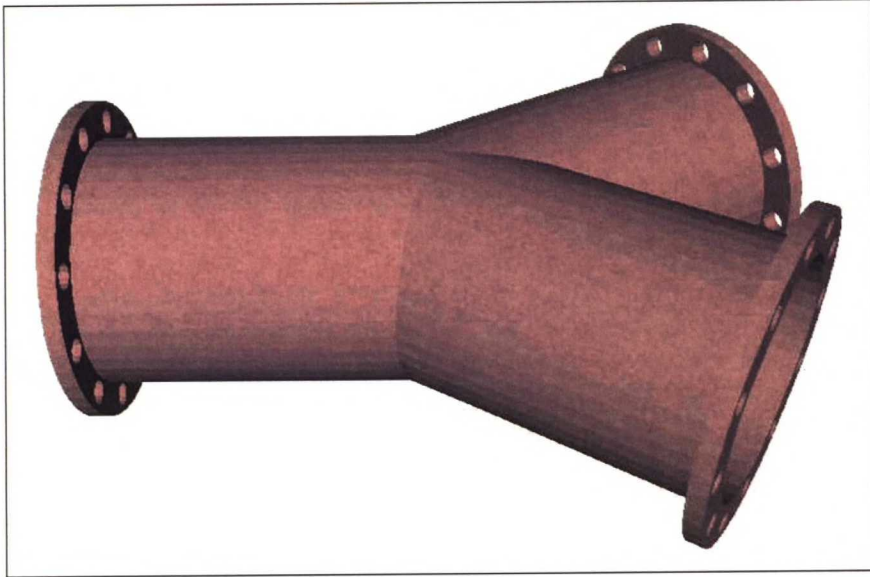
**Fig 44.** Y-pipe object covered with concrete material

### Stone

Stone is another building material in structural engineering and thus it was chosen for testing. Stone was constructed by adding two different frequencies of the perlin noise with different weights and blending them with reddish color. This time the largest weight was given to the highest frequency and half the weight was given to the basic frequency (for explanation, see **Eq 11**):

$$p(x, y) = C_v \left( 1 - C \frac{N_{f4} + 0.5N_{f1}}{1.5} \right) \quad (12)$$

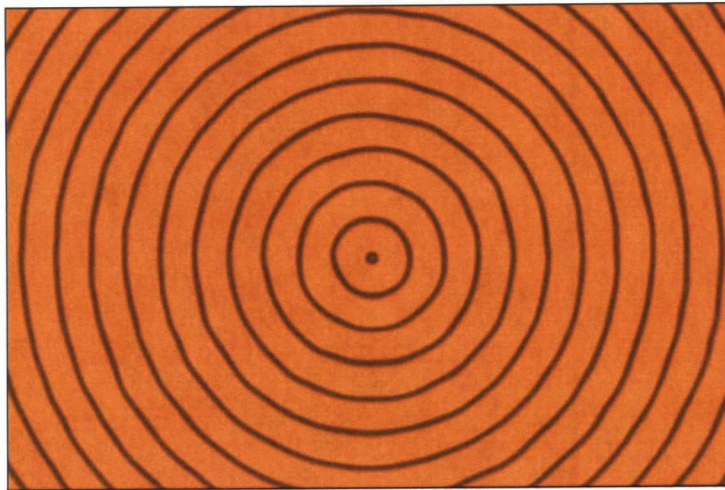
This simulates the small graininess of the surface of a stone. The result can be seen in the next figure (**Fig 45**) and the pixel shader code for creating stone can be found from **Appendix B**.



**Fig 45.** Y-pipe object covered with stone material

## Wood

Wood was chosen to be the third material to be simulated, since it is also important building material and can be modeled quite simply. Wood was created by modeling the structure of a tree in three dimensions. This was done by first creating the cross-section of the tree, which models the annual rings of the tree (**Fig 46**).



**Fig 46.** Wood rings created with pixel shader

The annual rings were created at the pixel shader by calculating the distance to the center of the given object axis, and giving the pixel dark or light color values depending on the distance. The transition from light to dark color was smoothed with slope function. The irregularities at the annual rings were created by taking a random

number from the 3D perlin noise texture map, and varying the distance between the annual rings and their center with this number:

$$d = a(\sqrt{x^2 + y^2} + N / b) \quad (13)$$

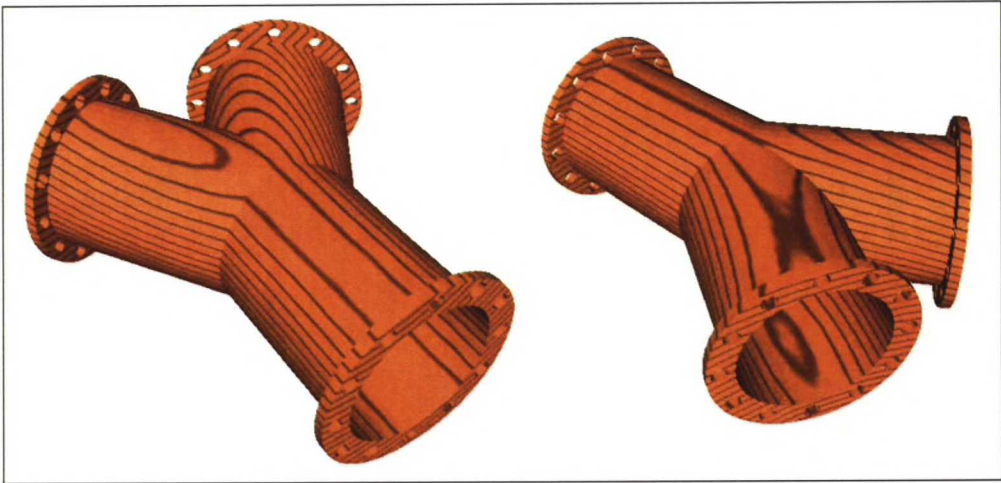
$$f = 3d^2 - 2d^3 \quad (14)$$

where  $d$  is the distance from the center to the texture coordinates  $(x,y)$  shifted with noise value  $N$ .  $a$  is a factor for scaling the rings,  $b$  is a factor for scaling the noise and  $f$  is the used slope function for smooth transition from light to dark color. The final pixel color  $p(x,y)$  was determined by:

$$p(x,y) = C_v f(1 - N / c) \quad (15)$$

where  $C_v$  is the color value from the vertex shader,  $f$  is the slope function (Eq 14),  $N$  is the noise value and  $c$  is a constant for scaling the noise value.

The third dimension, which is parallel to the rings, was also randomized by adding a value from the 3D perlin noise map to the location of the dark rings in the third dimension. When the 3D wood is applied to an object, the vertex coordinates of the object can be used as texture coordinates, and only some shifting and the direction of the rings must be decided. In the next figure (Fig 47) one can see the wood material applied to a Y-pipe and viewed from two different directions. The pixel shader code for creating wood can be found from **Appendix B**.



**Fig 47.** Y-pipe covered with wood material

## 5.3 Comparing various techniques

In this chapter the different techniques discussed previously and tested in the previous chapter are summed up. The advantages and disadvantages for using each of these techniques are given and some thoughts about how they correlate with the problem statements are made (chapter 3).

### 5.3.1 *Texture mapping*

The clear advantage of using texture images and texture mapping (chapter 2.2.5 on page 10) is that complex surface patterns can be represented with one bitmap image, instead of creating a triangle for each separate color of the surface. One can also take a photograph of a real material and use the image as a texture.

The big problem is how the texture coordinates should be generated. There is no common solution how to cover or wrap the three dimensional object with the two dimensional texture without deformations. This can be done only in specific circumstances (chapter 5.2.1 on page 39). With 3D textures the problem would not exist, but the memory consumption with static 3D textures is so large that there can only be a few of them in use.

To sum up texture mapping with the problem statements, one can say that they add visual information to the model. 2D textures are fast to render, don't consume much memory, but the texture coordinates are hard to edit. Static 3D textures are also fast to render, their texture coordinates are easy to edit, but they consume lots of memory.

### 5.3.2 *Anti-aliasing*

Full-screen anti-aliasing (chapter 2.2.6 on page 16) is a very simple technique to use. Today almost every *GPU* has the ability to perform full-screen anti-aliasing, and it can easily be switched on and off by the user from the settings of the *GPU*. It is quite clear the technique makes the rendered image more pleasant to look at, but in some cases this comes with the cost of highly increased rendering time (chapter 5.2.2 on page 42).

### **5.3.3 Bump mapping**

With bump mapping (chapter 4.1 on page 24) one can make the surface of an object look rough or grainy instead of flat. This highly increases the natural look of the surface, especially when it is used with the texture mapping. The most used and supported technique, dot product bump mapping (chapter 4.1.3 on page 27), was tested (chapter 5.2.3 on page 43). As a result, it was noted this technique adds visual information to the object by helping the viewer to see the irregularities of the surface.

The memory consumption is the same as with texture mapping, since the normal maps are stored as texture maps, but the impact on rendering speed is noticeable. However, as the graphics hardware evolves, the difference of the rendering speed between bump mapping and flat surface rendering gets smaller and the technique can be used without a large performance decrease. The implementation of dot product bump mapping requires quite a lot of work, and the material editing problems are almost the same as with texture mapping.

### **5.3.4 Procedural texturing**

The biggest advantage of using procedural texturing (chapter 4.3 on page 30) is its low memory consumption. Since all the texture graphics are calculated on-the-fly, there is no need to store any texture bitmaps (chapter 2.2.5 on page 10), which normally consumes large amounts of memory. Also there is no fixed bitmap resolution, since the level of detail needed is decided on runtime. Additional advantage from this is that there is no repeating pattern like with the normal texture map that is repeated, so the texture can cover arbitrary large areas without repeating.

The biggest disadvantage is the computation time needed to produce the procedural texture. The time needed can vary vastly, but usually it's longer than the time to get the values from a bitmap texture. Another disadvantage is the aliasing effect when using the same procedural function with all distances between a camera and an object. This has the same effect as using texture mapping without mipmapping (Fig 12 on page 14), and it requires lots of work to overcome this problem.

### **5.3.5 Shading languages**

When testing the shading languages, the Cg was chosen as it seemed to be the future standard of shading languages at the time. It is the only language that can be compiled to both OpenGL and Direct3D, and since both of these must be supported (chapter 2.1 on page 4), it was the natural choice. Only the pixel shader was chosen for testing, because the vertex shader is also responsible for lighting calculations, and the basic Gouraud shading (chapter 2.2.4 on page 8) implemented in OpenGL was adequate enough in this work.

Three different materials were created using pixel shader to procedurally create concrete, stone and wood. The advantage of this, compared to bitmaps in texture mapping, is that there is no need to store the bitmap anywhere and hence no memory consumption. This is especially a big advantage when creating true 3D materials, as was the case here. Since the materials are three dimensional they can be applied to any object, regardless of their shape, without any discontinuities or deformations (chapter 3.2 on page 22) and the texture coordinate definitions can be done automatically.

There are also downsides when using pixel shader. With the test configuration, the impact on rendering speed was clearly noticeable when compared to conventional texture mapping, but it will get better in the future with new hardware, since lots of research is put on the development. The bigger problem is the anti-aliasing effect and the problem of solving it. Since the materials are created in pixel level on computer screen, it is hard to construct a mechanism for calculating the anti-aliased material as it is done with the basic texture mapping (chapter 2.2.5 on page 10).

## **5.4 Summary of research work**

The ground for research work was laid out in chapter 3, where the research problem was studied and presented. Keeping these in mind, carefully selected 3D graphics techniques were selected for deeper investigations and these were presented in chapter 4. In this chapter all the material in the first four chapters was studied, and on that basis the research testing was carried out.

At first a testing environment, powerful enough for testing even the newest techniques, was constructed (chapter 5.1). However it turned out unsuitable for speed critical testing and another testing environment was used when needed.

Secondly, testing software was built for testing various 3D techniques introduced previously. It was not possible to test every 3D graphics technique, so only those techniques that seemed possible and meaningful to implement were included in the testing software. The techniques selected for testing and the results of these tests were presented in chapter 5.2.

In the next chapter (5.3) the tested techniques were put on deeper analysis; their advantages and disadvantages were discussed and their relation to the research problem statements (chapter 3) was investigated. As a result it was noted they all add some visual quality to an object and also some visual information to the model.

For efficiency, the rendering speed and memory consumption were studied, and it was noted that rendering speed was decreased with anti-aliasing, bump mapping and pixel shader, but the amount of decrease is dependable on the hardware used. Also as the hardware evolves, the amount of decrease of the rendering speed will get smaller. With memory consumption it was noted that static 2D texturing is possible to implement, but static 3D textures in general consume too much memory to be usable. Bump mapping requires twice as much memory as 2D texturing. With pixel shader materials the memory consumption is very marginal, since they use procedural rendering methods.

The third point of research problem was the editing problem, meaning that no matter how great some technique might be otherwise, it can't be used if it is too hard to be implemented or edited by the user. 2D texturing is quite easy to be implemented, but the texture coordinate definition is quite a challenge. This however can be overcome with the projector functions. Full-screen anti-aliasing is easily enabled from the settings of the *GPU*. Bump mapping requires quite a lot of work to implement, and the texture coordinate definition problem remains the same as with 2D texturing. Also pixel shader material support requires lots of work to implement, but after the support is done, it should be quite easy for the user to create new materials with the Cg language.

In the table below (**Table I**), one can find the five techniques tested and their relation to four indicators, which are visual quality, memory consumption, rendering speed decrease and implementation difficulty.

Technique	Visual quality	Memory consumption	Rendering speed decrease	Implementation difficulty
Texture mapping	Better than plain colors	2D: moderate, 3D: high	Low	Low
Anti-aliasing	Smoother edges	None	High on average	Low
Bump mapping	Adds roughness to surfaces	2 x texture mapping	Moderate	Moderate
Procedural texturing	Varies	Low	Moderate	High
Shading languages	Almost anything is possible	Low	Moderate	High

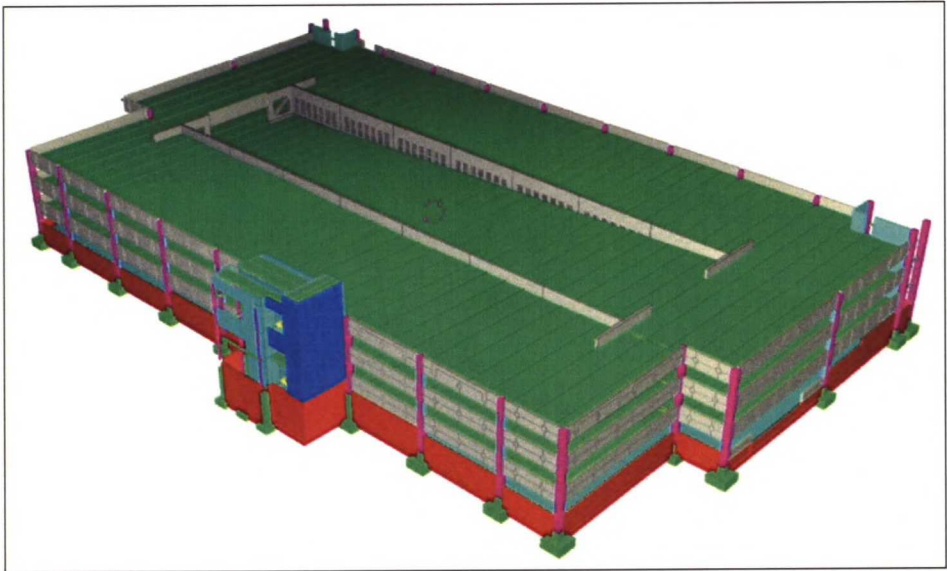
**Table I.** Comparing five tested techniques

# 6 Implementations

During the work on this thesis, some of the techniques described were implemented to the Tekla Z-kit library (chapter 1) and to its support libraries.

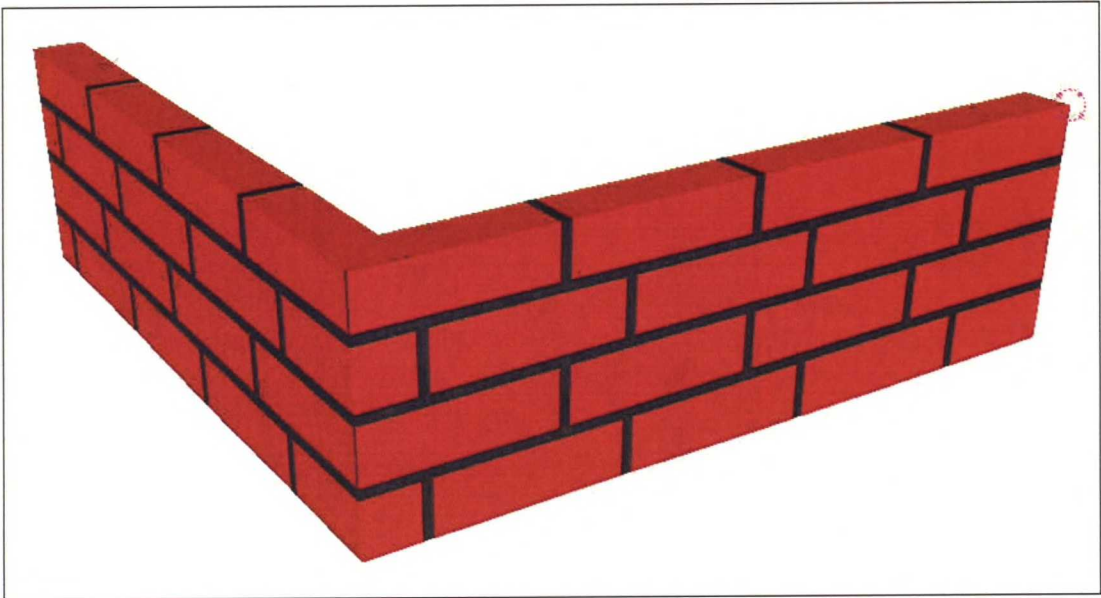
## 6.1 Fog and texture mapping support

First the fog support was implemented (chapter 2.2.3 on page 7) to material definitions, to help the viewer of the model to see more precisely the depth of an object (**Fig 48**). The fog is dynamically calculated so the closest object to the viewer is not affected by the fog effect, and the farthest object has the same amount of fog regardless of the view distance.



**Fig 48.** Fog effect on material helps the viewer to see the depth in the scene

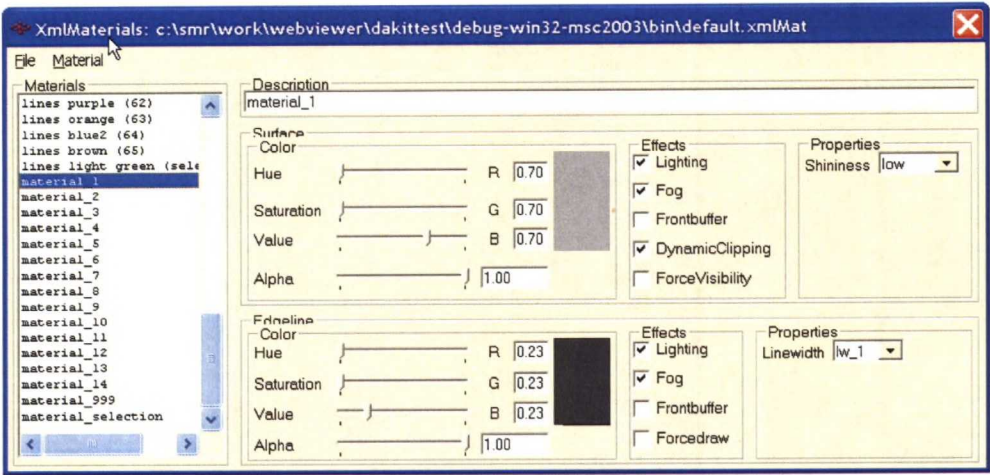
Another technique implemented was the basic texture mapping support (chapter 2.2.5 on page 10). This means the users of Tekla Z-kit have the ability to assign a texture bitmap to a material, and when doing so, the object must be supplied with the texture coordinates. An example of this can be seen in the next figure (**Fig 49**), where a simple object is covered with texture bitmap emulating a simple brick, which makes the object look like a brick wall.



**Fig 49.** Brick wall created with the texture support of the Z-kit

### 6.2 Support editors for Z-kit

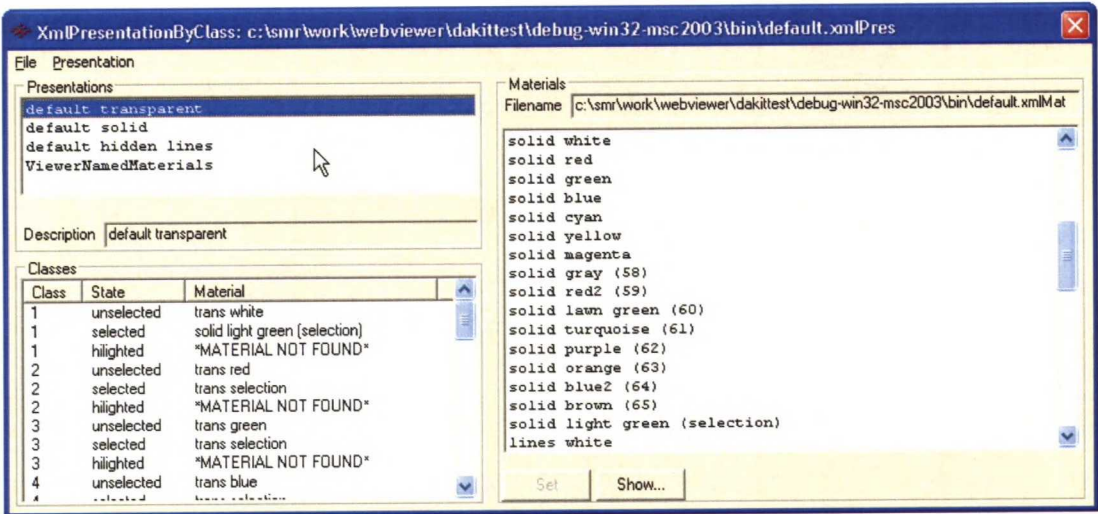
For editing the different material definitions and the lighting of the scene, three simple graphical editors were created. A material editor can be used to edit the properties of a single material definition, such as color, transparency (alpha) and shininess (**Fig 50**). Separate colors must be defined for both surfaces and edge lines of surfaces. Also the visualization effects are separate for both. The material editor is constructed in a way that every change made in the editor can be seen real-time in the model.



**Fig 50.** Material editor for editing Z-kit materials used in Tekla Structures

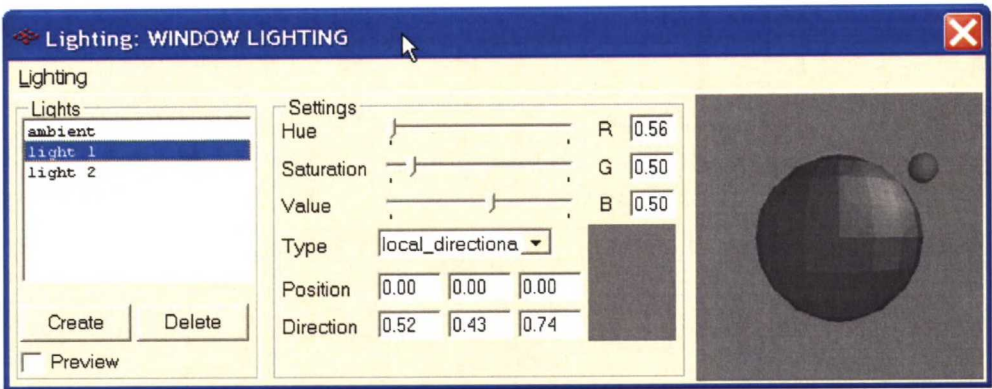
A separate presentation editor was created for creating different presentations. A presentation consists of selected materials (**Fig 51**). The idea of having a presentation

is that it is easy to change the look of the whole model by changing the presentation. Also the presentation editor was created in a way that the modifications made could be previewed in the model at the same time. A class in the editor is a definition of Tekla Structures for materials with different states.



**Fig 51.** Presentation editor for editing Z-kit presentations used in Tekla Structures

The visual look of the materials on the surface of an object depends on the lighting of the scene and how they interact with each other (chapter 2.2.4 on page 8). This interaction is controlled by the lighting processor or the vertex shader of the GPU (Fig 16 on page 17). To help the Z-kit user to see how the different lighting settings affect with the materials, a lighting editor was created that could help to edit the lighting in the window (**Fig 52**).



**Fig 52.** Lighting editor for editing lights in the window

Every window must set the global ambient color and zero or more light sources. Light source can be directional or point light, and for now the lights are always

relative to the camera (local lights). The directional light emulates the behavior of the sun, and the point light is like a lighting bulb, which have a position in space. The direction and position can be altered with the mouse by rotating the small ball around the big ball, as can be seen in the figure above (Fig 52). Also the lighting editor was created in a way that the modifications made could be previewed in the model window at the same time.

The documentation and code for these three support editors can be found from **Appendix D**.

## 7 Conclusions and Future Work

In this final chapter the thesis is reviewed in short, conclusions are made and some thoughts about future work are given.

### 7.1 Conclusions

The problem statement given by the Tekla Corporation was not defined very exactly, which gave the freedom to choose the way it would be approached. It also forced to think deeply what should be included in the thesis and what shouldn't, and how to include all the essential texts that would be required to make this thesis reasonable. The work was started by thinking about the overall problem (chapter 1) and defining the research problem (chapter 3) more clearly. Shortly, the problem was that previously only colors and lighting were used to distinguish different materials from each other in Tekla's structural engineering software Tekla Structures. The work was about finding new techniques to define new materials and possibly implement some of those in Tekla.

The writer had some basic knowledge about 3D graphics techniques and based on those, books were read and articles concerning new 3D techniques searched. From these some of the most used and new 3D techniques in use, were selected, and an introduction to the techniques was given (chapter 4). Bump mapping is widely used, but requires some special hardware support and is somewhat complex to implement. It also has some shortcomings, which are tried to overcome with displacement mapping technique. Procedural texturing techniques are also widely used and they should be used when the memory consumption is the key factor of performance. The newest heading in the field of 3D graphics are the vertex and pixel shaders, which make the *GPU* act very much like the *CPU*, meaning it can be programmed by the user with high level languages. It is however quite difficult to take shaders in use and to build an implementation that supports them.

On the next phase, a testing environment was constructed for testing how difficult the various techniques would be to implement and how difficult it would be to evaluate their performance hit on rendering speed and memory consumption (chapter 5). The testing environment consisted of decent computer and a high-end graphics

accelerator with self-made software that could load and render basic 3D objects with different material definitions. Various 3D techniques from previous chapters were selected for testing, and they were also compared to each other, as well as their interaction with the research problem was studied.

As a result it was noted that the basic static 2D texture mapping is quite easy to implement, but texture coordinates definition is difficult to handle. Projector functions can provide some help, but on general level they can be used only to specific objects. Full-screen anti-aliasing can be easily enabled, but with most hardware configurations it causes considerable decrease of rendering speed. Bump mapping however does not cause any remarkable performance decrease with high-end graphics hardware, but it is quite difficult to implement and use. The newest field in 3D real-time rendering is the vertex and pixel shaders, which bring the programmability of the *CPUs* also to *GPUs*. This opens up huge possibilities since now the graphics accelerator programmer can do almost anything with the *GPU*. In this thesis the pixel shader was used to create procedural materials simulating real-life materials, and it was noted that the result was visually pleasing, but the implementation of the shader support requires some more work.

On the final phase some implementations were carried out (chapter 6). The basic fog and 2D texture mapping support were added to the Z-kit library and material and lighting editors were created as stand-alone support libraries that could be added to any software using the Z-kit library.

### **7.1.1 Comparing results to problem statements**

The main purpose of this work was to find methods how to add visual information to large models created with Tekla Structures (chapter 3.1). Here the visual information was meaning a more pleasant look and feel of the model, and also how one could add some visual object related information on the model. The various techniques tested do give the objects a more pleasant look, and they increase the number of materials possible to view on the screen. They do not, however, increase the methods to add visual object related information very much, and thus more work needs to be done with this problem.

The minor purpose of this work was to possibly implement some of the techniques, keeping in mind that they must be efficient (chapter 3.2) and easy to edit (chapter 3.3). The implemented techniques to Z-kit (fog and basic 2D texture support) are very efficient, since neither of them slows down the rendering process noticeably. Also the fog support consumes no memory, but the texture support needs some extra memory handling routines.

For editing materials easily, three graphical support editors were created. These help the user of the Z-kit to edit the materials and lighting conditions with graphical user interface. Compared to editing them manually with a text editor, the graphical editors give a big improvement.

## 7.2 Future Work

From now on the development of the Z-kit library continues, and in the next phase the intention is to use the vertex shader to do some of the vertex related calculations currently done in the *CPU*. This will increase the rendering speed, since now lots of the rendering work is done on the *CPU*, and the Z-kit library is *CPU* bounded with most hardware configurations. Other implementations to be done are the support of texture coordinate generation into the material editor and the possibility to use projector functions. This would help the user of the Z-kit library to use the implemented texture mapping support. Other 3D techniques introduced in this thesis are also taken into consideration, and they might be in use in the future.

## References

- [AKE02] Akenine-Möller, Tomas & Haines, Eric – *Real-Time Rendering*, 2<sup>nd</sup> ed., A. K. Peters Ltd., 2002
- [AKI98] Akin, Alan – *Microsoft and 3D Graphics: A Case Study in Suppressing Innovation and Competition*,  
<http://www.vcnet.com/bms/features/3d.html>, 1998
- [BLI78] Blinn, James – *Simulation of Wrinkled Surfaces*, SIGGRAPH Proceedings of the 5th annual conference on Computer graphics and interactive techniques, pp. 286-292, 1978
- [BLY99] Blythe, David et al. – *Advanced Graphics Programming Techniques Using OpenGL*, SIGGRAPH '99 Course,  
<http://www.opengl.org/resources/tutorials/sig99/advanced99/notes/notes.html>, 1999
- [COO84] Cook, Robert L. – *Shade Trees*, Computer Graphics, Vol. 18, Nr. 3, pp. 223-231, July 1984
- [DUN02] Dunn, Fletcher & Parberry, Ian – *3D Math Primer for Graphics and Game Development*, Wordware Publishing Inc., 2002
- [EBE02] Ebert, David et al. – *Texturing & Modeling: A Procedural Approach*, 3<sup>rd</sup> ed., Morgan Kaufmann, 2002
- [GOU71] Gouraud, H. – *Computer Display of Curved Surfaces*, Transactions on Computers, Vol. 20, pp. 623-629, 1971
- [HAE90] Haeberli, Paul & Akeley, Kurt – *The accumulation buffer: hardware support for high-quality rendering*, Computer Graphics, Volume 24, Number 4, August 1990
- [HEA97] Hearn, Donald & Baker, M. Pauline – *Computer Graphics, C Version*, 2<sup>nd</sup> ed., Prentice Hall Inc., 1997
- [IMA00] Imagination Technologies Ltd – *A comparison of Bump Mapping techniques*,  
<http://www.pvrdev.com/pub/PC/doc/f/Bump%20Mapping%20Comparison.htm>, 2000
- [KER00] Kerlow, Isaac – *The Art of 3-D Computer Animation and Imaging*, 2<sup>nd</sup> ed., John Wiley & Sons Inc., 2000
- [KES04] Kessenich, John et al. – *The OpenGL Shading Language (Version 1.10)*,  
<http://www.opengl.org/documentation/ogsl.html>, 2004
- [KIL00] Kilgard, Mark – *A Practical and Robust Bump-mapping Technique for Today's GPUs*, Game Developers Conference (GDC) Proceedings: Advanced OpenGL Game Development, 2000
- [LEE00] Lee, Aaron et al. – *Displaced Subdivision Surfaces*, ACM Proceedings of the 27th annual conference on Computer graphics and interactive techniques, pp. 85-94, 2000

- [LUN03] Luna, Frank D. – *Introduction to 3D Game Programming with DirectX 9.0*, Wordware Publishing Inc., 2003
- [MIC04d] Microsoft Corporation – *DirectX Graphics*, MSDN Library, [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9\\_c/directx/graphics/dxgraphics.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/directx/graphics/dxgraphics.asp), 2004
- [MIC04o] Microsoft Corporation – *OpenGL Start Page*, MSDN Library, [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/opengl/openglstart\\_9uw5.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/opengl/openglstart_9uw5.asp), 2004
- [NVI02] NVidia – *Technical Brief: The NVidia Cg Compiler*, <http://www.nvidia.com>, 2002
- [PER85] Perlin, Ken – *An Image Synthesizer*, ACM Proceedings of the 12th annual conference on Computer graphics and interactive techniques, pp. 287-296, 1985
- [PHO75] Phong, Bui Tuong – *Illumination for Computer Generated Pictures*, Communications of the ACM, Vol. 18, Nr. 6, pp. 311-317, June 1975
- [PIX00] Pixar – *The RenderMan Interface, Version 3.2*, <https://renderman.pixar.com/products/rispec/index.htm>, 2000
- [SAL01] Salvator, Dave – *ExtremeTech 3D Pipeline Tutorial*, ExtremeTech, [http://www.extremetech.com/print\\_article/0,1583,a=2674,00.asp](http://www.extremetech.com/print_article/0,1583,a=2674,00.asp), 2001
- [SAL02] Salvator, Dave – *ATI's Radeon 9700 Scores Big*, ExtremeTech, [http://www.extremetech.com/print\\_article2/0,2533,a=30125,00.asp](http://www.extremetech.com/print_article2/0,2533,a=30125,00.asp), 2002
- [SCH94] Schlag, John – *Fast Embossing Effects on Raster Image Data*, Graphics Gems IV, Academic Press Inc., pp. 433-437, 1994
- [SEG97] Segal, Mark & Akeley, Kurt – *The OpenGL Graphics System: A Specification (Version 1.1)*, Silicon Graphics Inc., <http://www.opengl.org/documentation/specs/version1.1/glspec1.1/index.html>, 1997
- [SGI04] SGI – *OpenGL*, <http://www.sgi.com/software/opengl/>, 2004
- [SPI03] Spitzer, John – *Real-Time Procedural Effects*, Game Developers Conference Europe (GDCE), <http://www.gdconf.com/archives/2003E/index.htm>, 2003
- [TEK04] Tekla Corporation – *Tekla Structures*, <http://www.tekla.com>, 2004
- [WAN03] Wang, Lifeng et al. – *View-Dependent Displacement Mapping*, ACM Transactions on Graphics, Vol. 22, Nr. 3, pp. 334-339, July 2003
- [WOO97] Woo, Mason et al. – *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.1*, 2<sup>nd</sup> ed., Addison-Wesley Publishing Co., 1997

## Appendix A – Steelmark run

### Steelmark run without antialiasing:

```
steelmark v0.2
Computer name:      ZSMR
Operating system:  Microsoft Windows XP Professional Service Pack 1 (Build 2600)
Processor:
  count 1 (586, 6, 11, 1) (type, level, model, stepping)
  speed 1129 MHz
  memory 511 Mb
Display driver:
  Dell C810 (nv4_disp.dll)
  6.14.10.4482 (product, version, subversion, build)

digitally (WHQL) signed without date
loadTime 4.68 s
maxFrameTime (ms): 135.32 102.21 101.83 101.78 100.92 100.71 100.61 100.46 100.4
0 100.17 99.89 99.84 99.61 99.59 99.46 99.34
minFrameTime (ms): 19.55 20.20 20.22 20.27 20.29 20.29 20.34 20.35 20.37 20.46 2
0.47 20.49 20.50 20.52 20.71 20.81
total frames 591
total frames time used 29.892641 s (99.642%)
avgFrameTime 50.58 ms (19.77 fps)
```

### Steelmark run with antialiasing:

```
steelmark v0.2
Computer name:      ZSMR
Operating system:  Microsoft Windows XP Professional Service Pack 1 (Build 2600)
Processor:
  count 1 (586, 6, 11, 1) (type, level, model, stepping)
  speed 1129 MHz
  memory 511 Mb
Display driver:
  Dell C810 (nv4_disp.dll)
  6.14.10.4482 (product, version, subversion, build)

digitally (WHQL) signed without date
loadTime 4.79 s
maxFrameTime (ms): 273.87 147.85 146.97 146.02 145.57 145.07 144.73 144.63 143.6
5 142.83 142.72 142.52 142.19 142.15 142.15 140.01
minFrameTime (ms): 37.26 37.52 38.44 39.28 41.11 41.16 41.16 41.18 41.36 41.36 4
1.86 42.49 42.53 42.55 42.69 42.74
total frames 348
total frames time used 29.964377 s (99.881%)
avgFrameTime 86.10 ms (11.61 fps)
```

## Appendix B – Code for pixel shaders

The following is the code for the pixel shader creating concrete:

```
/////////////////////////////////////////////////////////////////
// pixel shader code for creating concrete

struct vertout                                // output from vertex shader
{
    float4 position : POSITION; // position in view coordinates
    float4 color0   : COLOR0;  // pixel color
    float4 texcoord0 : TEXCOORD0; // pixel texture coordinates value
};

//-----
// IN          - incoming fragment to be processed
// noiseTexture3d - a texture for the pixel shader to use, 3d perlin noise map
//-----

struct pixout                                // output from pixel shader
{
    float4 color : COLOR; // pixel color
};

pixout main( vertout IN,
             uniform sampler3D noiseTexture3d)
{
    pixout OUT;
    float noisevalue =
        (tex3D(noiseTexture3d, IN.texcoord0.xyz) +           // normal frequency
         tex3D(noiseTexture3d, IN.texcoord0.xyz * 2) * 0.5 + // double frequency
         tex3D(noiseTexture3d, IN.texcoord0.xyz * 4) * 0.25) / 1.75; //quad frequency

    OUT.color = IN.color0 * (1 - noisevalue / 4);

    return OUT;
}
/////////////////////////////////////////////////////////////////
```

The following is the code for the pixel shader creating stone:

```
/////////////////////////////////////////////////////////////////
// pixel shader code for creating stone

struct vertout                                // output from vertex shader
{
    float4 position : POSITION; // position in view coordinates
    float4 color0   : COLOR0;  // pixel color
    float4 texcoord0 : TEXCOORD0; // pixel texture coordinates value
};

//-----
// IN          - incoming fragment to be processed
// noiseTexture3d - a texture for the pixel shader to use, 3d perlin noise map
//-----

struct pixout                                // output from pixel shader
{
    float4 color : COLOR; // pixel color
};

pixout main( vertout IN,
             uniform sampler3D noiseTexture3d)
{
    pixout OUT;
    float noisevalue =
```

```

        (tex3D(noiseTexture3d, IN.texcoord0.xyz * 4) +           // quad frequency
        tex3D(noiseTexture3d, IN.texcoord0.xyz) * 0.5) / 1.5; // normal frequency

    OUT.color = IN.color0 * (1 - noisevalue / 4);

    return OUT;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

```

**The following is the code for the pixel shader creating wood:**

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// pixel shader code for creating wood

struct vertout          // output from vertex shader
{
    float4 position : POSITION; // position in view coordinates
    float4 color0 : COLOR0; // pixel color
    float4 texcoord0 : TEXCOORD0; // pixel texture coordinates value
};

//-----
// IN          - incoming fragment to be processed
// noiseTexture3d - a texture for the pixel shader to use, 3d perlin noise map
//-----

struct pixout
{
    float4 color : COLOR; // pixel color
};

pixout main( vertout IN,
             uniform sampler3D noiseTexture3d )
{
    pixout OUT;

    // switch x and z, and shift for now, fix texcoords for permanent solution
    float3 coord = IN.texcoord0.zyx + float3(-2,-2.5,0);

    // lower frequency in z, noise3d E[0,1]
    float noise3d = tex3D(noiseTexture3d, float3(coord.x, coord.y, coord.z / 3)).r;

    // distance from center
    float distcent = length(coord.xy);

    // noisy wood ring, noisysdist E[0,1]
    float noisysdist = frac((distcent + noise3d/100) * 15);

    float factor = 1; // light wood
    float p;
    if (noisysdist < 0.15)
    {
        p = noisysdist / 0.15; // smooth transition from light to dark wood
        factor = 1 - 0.5 * (3*p*p - 2*p*p*p); // E[1,0.5]
    }
    else if (noisysdist < 0.2)
    {
        factor = 0.5; // dark wood
    }
    else if (noisysdist < 0.35)
    {
        p = (noisysdist - 0.2) / 0.15; // smooth transition from dark to light wood
        factor = 0.5 + 0.5 * (3*p*p - 2*p*p*p); // E[0.5,1]
    }

    OUT.color = IN.color0 * factor * (1 - noise3d/8);

    return OUT;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

```

## Appendix C – Code for planar and cylindrical mapping

The following is the code for calculating the planar mapping for texture coordinates:

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#include <math.h>

typedef struct
{
    double x, y, z;
} vector3d_t;

typedef struct
{
    double x, y;
} vector2d_t;

/*-----*/
static int
linearMapping3d2d(vector3d_t Vertex,
                  vector3d_t Plane,
                  vector2d_t *pResult)
/*-----*/
{
    double rotangleX, rotangleY, tmpY, tmpZ;
    double piitwo = asin(1); // pi/2

    if (Plane.z != 0.0)
        // rotate angle so that plane.y will be 0
        rotangleX = atan(Plane.y / Plane.z);
    else
        rotangleX = Plane.y < 0.0 ? -piitwo : Plane.y > 0.0 ? piitwo : 0.0;
    Plane.z = Plane.y * sin(rotangleX) + Plane.z * cos(rotangleX);

    if (Plane.z != 0.0)
        // rotate angle so that plane.x will be 0
        rotangleY = atan(- Plane.x / Plane.z);
    else
        rotangleY = Plane.x < 0.0 ? piitwo : Plane.x > 0.0 ? -piitwo : 0.0;

    // rotate vertex around x-axis
    tmpY = Vertex.y;
    Vertex.y = tmpY * cos(rotangleX) - Vertex.z * sin(rotangleX);
    Vertex.z = tmpY * sin(rotangleX) + Vertex.z * cos(rotangleX);

    // rotate vertex around y-axis
    tmpZ = Vertex.z;
    Vertex.z = tmpZ * cos(rotangleY) - Vertex.x * sin(rotangleY);
    Vertex.x = tmpZ * sin(rotangleY) + Vertex.x * cos(rotangleY);

    (*pResult).x = Vertex.x;
    (*pResult).y = Vertex.y;

    return 0;
}

/*-----*/
static int
calcTextureCoordinatesPlanar(vertex_withnormals_t *pVertices, int nVertices,
vector3d_t projectionPlane)
/*-----*/
{
    vector3d_t vertex;
    vector2d_t result;
    double minu, minv, maxu, maxv;

    for (int i = 0; i < nVertices; i++)
    {
        vertex.x = pVertices[i].x;
        vertex.y = pVertices[i].y;
```

```

        vertex.z = pVertices[i].z;
        linearMapping3d2d(vertex, projectionPlane, &result);
        pVertices[i].tu = (float) result.x;
        pVertices[i].tv = (float) result.y;

        if (i==0 || result.x < minu) minu = result.x;
        if (i==0 || result.x > maxu) maxu = result.x;
        if (i==0 || result.y < minv) minv = result.y;
        if (i==0 || result.y > maxv) maxv = result.y;
    }

    for (i = 0; i < nVertices; i++)
    {
        if (g_fScaleTexture)
        {
            pVertices[i].tu *= 3;
            pVertices[i].tv *= 3;
        }
        else
        {
            pVertices[i].tu -= (float) minu;
            pVertices[i].tv -= (float) minv;
            pVertices[i].tu /= (float) (maxu - minu);
            pVertices[i].tv /= (float) (maxv - minv);
        }
    }

    return 0;
}
/////////////////////////////////////////////////////////////////

```

**The following is the code for calculating the cylindrical mapping for texture coordinates:**

```

/////////////////////////////////////////////////////////////////

/*-----*/
static void
CylinderMapSmr(double x, double y, double z,
               double *u, double *v)
/*-----*/
{
    double longitude = atan2(x, z); // E (-PI,PI]

    *u = (longitude + PI) / TWOPI; // E (0,1]
    *v = y;
}

/*-----*/
static int
calcTextureCoordinatesCylinder(vertex_withnormals_t *pVertices, int nVertices)
/*-----*/
{
    vector3d_t vertex;
    vector2d_t result;
    float minu, minv, maxu, maxv;

    for (int i = 0; i < nVertices; i++)
    {
        vertex.x = pVertices[i].x;
        vertex.y = pVertices[i].y;
        vertex.z = pVertices[i].z;
        CylinderMapSmr(vertex.x, vertex.y, vertex.z, &result.x, &result.y);
        pVertices[i].tu = (float) result.x;
        pVertices[i].tv = (float) result.y;

        if (i==0 || result.x < minu) minu = pVertices[i].tu;
        if (i==0 || result.x > maxu) maxu = pVertices[i].tu;
        if (i==0 || result.y < minv) minv = pVertices[i].tv;
        if (i==0 || result.y > maxv) maxv = pVertices[i].tv;
    }
}

```

```

for (i = 0; i < nVertices; i++)
{
    // x determines the sign of angle, when y-axel is used in atan2
    vector3d_t normalx = { 1.0, 0.0, 0.0 };
    vector3d_t normalz = { 0.0, 0.0, 1.0 }; // z determines the side of circle
    vector3d_t trianglevertex[3];
    int i0 = i;
    int i1 = i+1;
    int i2 = i+2;
    if (i0 % 3 == 1)
        i2 -= 3;
    else if (i0 % 3 == 2)
    {
        i1 -= 3;
        i2 -= 3;
    }
    trianglevertex[0].x = pVertices[i0].x;
    trianglevertex[0].y = pVertices[i0].y;
    trianglevertex[0].z = pVertices[i0].z;
    trianglevertex[1].x = pVertices[i1].x;
    trianglevertex[1].y = pVertices[i1].y;
    trianglevertex[1].z = pVertices[i1].z;
    trianglevertex[2].x = pVertices[i2].x;
    trianglevertex[2].y = pVertices[i2].y;
    trianglevertex[2].z = pVertices[i2].z;
    if (classifyVertex(normalx, trianglevertex[0]) < 0 &&
        classifyVertex(normalz, trianglevertex[0]) < 0) // vertex in "min" side
    {
        if ((classifyVertex(normalx, trianglevertex[1]) >= 0 &&
            classifyVertex(normalz, trianglevertex[1]) < 0) ||
            (classifyVertex(normalx, trianglevertex[2]) >= 0 &&
            classifyVertex(normalz, trianglevertex[2]) < 0)) // "max" side
            pVertices[i0].tu += 1.0; // triangle belongs to "max" side
    }
}
for (i = 0; i < nVertices; i++)
{
    int i0 = i;
    int i1 = i+1;
    int i2 = i+2;
    if (i0 % 3 == 1)
        i2 -= 3;
    else if (i0 % 3 == 2)
    {
        i1 -= 3;
        i2 -= 3;
    }
    // fix texture u if vertex lies in y-axel
    if (pVertices[i0].x == 0.0 && pVertices[i0].z == 0.0)
    {
        pVertices[i0].tu = (pVertices[i1].tu + pVertices[i2].tu) / 2;
    }
}
if (g_fScaleTexture)
{
    for (i = 0; i < nVertices; i++)
    {
        pVertices[i].tu *= 9;
        pVertices[i].tv *= 3;
    }
}
return 0;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

```

## **Appendix D – Documentation and code for Z-kit editors**

From the next page starts the documentation and code for the support editors of Z-kit that were created during this thesis.

**Z-kit editors**

Samu Ruuhonen  
Version Master's Thesis  
11/30/2004 12:52:00 PM

# Table of Contents

Main Page.....	iv
Editors and classes.....	iv
Typedefs defined elsewhere.....	iv
Class Index.....	iv
File Index.....	v
Class Documentation.....	vi
DakitLightingEditor.....	1
DakitMaterialEditor.....	1
DakitPresentationByClassEditor.....	21
dakitPresentationeditor_t.....	35
lightingEditor_t.....	47
materialEditor_t.....	48
presentationEditor_t.....	49
File Documentation.....	50
DakitLightingEditor.cpp.....	51
DakitLightingEditor.h.....	51
DakitLightingEditorCB.cpp.....	55
DakitMaterialEditor.cpp.....	56
DakitMaterialEditor.h.....	58
DakitMaterialEditorCB.cpp.....	61
DakitPresentationByClassEditor.cpp.....	62
DakitPresentationByClassEditor.h.....	64
DakitPresentationByClassEditorCB.cpp.....	67
Index.....	68
.....	70

# Z-kit editors Main Page

All the code and documentation in this paper are copyrighted © by Tekla® Corporation. All rights reserved.

This is the documentation and code for the Z-kit editors. Z-kit is a 3D graphics software library used by Tekla Structures. Other libraries used in editors but documented elsewhere include:

- **dakit** user interface library used in Tekla
- **dbfast** fast database library for simple databases

## Editors and classes

Three editors have been documented, namely lighting, material and presentation. The editors are built as stand-alone editors and can be linked with any program using the Z-kit library. Each one is defined in their own classes, named **DakitLightingEditor**, **DakitMaterialEditor** and **DakitPresentationByClassEditor**.

## Typedefs defined elsewhere

The editors are using various typedefs and classes that are defined somewhere else and are not documented here. Here is the list of these variables and a brief description of each of them.

- **dz\_world\_t** Z-kit world for all other Z-kit objects
- **dz\_window\_t** Z-kit window
- **dz\_interactor\_t** Z-kit interactor for handling mouse events
- **dz\_scene\_t** Z-kit scene for storing presentation and lighting data
- **dz\_layer\_t** Z-kit layer for storing 3D objects
- **dz\_renderingdevice\_t** Z-kit rendering device for drawing
- **WebViewerID** Unique identifier class
- **XmlMaterials** Database class for storing materials in xml format
- **XmlPresentationByClass** Database class for storing presentations in xml format

Z-kit editors Class Index

Z-kit editors Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

DakitLightingEditor	1
DakitMaterialEditor	21
DakitPresentationByClassEditor	35
dakitPresentationeditor_t	47
lightingEditor_t	48
materialEditor_t	49
presentationEditor_t	50

Z-kit editors File Index

Z-kit editors File List

Here is a list of all files with brief descriptions:

DakitLightingEditor.cpp	51
DakitLightingEditor.h	55
DakitLightingEditorCB.cpp	56
DakitMaterialEditor.cpp	58
DakitMaterialEditor.h	61
DakitMaterialEditorCB.cpp	62
DakitPresentationByClassEditor.cpp	64
DakitPresentationByClassEditor.h	67
DakitPresentationByClassEditorCB.cpp	68

# Z-kit editors Class Documentation

## DakitLightingEditor Class Reference

#include <DakitLightingEditor.h>

### Public Member Functions

- DakitLightingEditor ()
- ~DakitLightingEditor ()
- int displayEditor ()
- int closeEditor (bool fDeleteDialog)
- int setLighting (WebViewerHeader\_t webViewerHeader, WebViewerLighting\_t webViewerLighting)
- int getLighting (WebViewerHeader\_t &webViewerHeader, WebViewerLighting\_t &webViewerLighting)
- int setWindow (dz\_world\_t worldID, dz\_window\_t windowID)
- int lightSelect ()
- int colorChange (bool fHsv)
- int typeChange ()
- int previewChange ()
- int createLight ()
- int deleteLight ()
- int newLighting ()
- int newLightingFromWindow ()
- int importLighting ()
- int exportLighting ()

### Static Public Member Functions

- DakitLightingEditor \* findEditor (char \*pEditorName)
- DakitLightingEditor \* findById (dz\_world\_t worldID, dz\_interactor\_t interactorID)
- int mouseButtonDownCB (dz\_world\_t worldID, dz\_interactor\_t interactorID, dz\_window\_t windowID, dz\_mousebuttonType\_e buttonType)
- int mouseButtonUpCB (dz\_world\_t worldID, dz\_interactor\_t interactorID, dz\_window\_t windowID, dz\_mousebuttonType\_e buttonType)
- int mouseMoveCB (dz\_world\_t worldID, dz\_interactor\_t interactorID, dz\_window\_t windowID, int x, int y)
- int mouseWheelCB (dz\_world\_t worldID, dz\_interactor\_t interactorID, dz\_window\_t windowID, int rotation)

### Private Member Functions

- int initEditor ()
- int setTitle ()
- int createObjects ()
- int deleteObjects ()
- int getLightingFromWindow (WebViewerLighting\_t &webViewerLighting)
- int setLightingToWindow (WebViewerLighting\_t webViewerLighting)
- int ballPositionChanged ()
- int initView ()
- void rotateBallHorizontal (float angle)
- void rotateBallVertical (float angle)
- void scaleBall (int rotation)
- void setBallDirection (double x, double y, double z)
- void setBallPosition (double x, double y, double z)
- int onMouseButtonDownCB (dz\_world\_t worldID, dz\_interactor\_t interactorID, dz\_window\_t windowID, dz\_mousebuttonType\_e buttonType)
- int onMouseButtonUpCB (dz\_world\_t worldID, dz\_interactor\_t interactorID, dz\_window\_t windowID, dz\_mousebuttonType\_e buttonType)
- int onMouseMoveCB (dz\_world\_t worldID, dz\_interactor\_t interactorID, dz\_window\_t windowID, int x, int y)
- int onMouseWheelCB (dz\_world\_t worldID, dz\_interactor\_t interactorID, dz\_window\_t windowID, int rotation)

### Private Attributes

- dz\_world\_t m\_lightingWorld
- dz\_window\_t m\_lightingWindow
- WebViewerHeader\_t m\_webViewerHeader
- WebViewerLighting\_t m\_webViewerLighting
- WebViewerLighting\_t m\_webViewerLightingOriginal
- WebViewerDatabase m\_database
- DakitDatabaseEditor \* m\_pDatabaseEditor
- char m\_editorName [256]
- int m\_numChanges
- dz\_world\_t m\_worldID
- dz\_renderingdevice\_t m\_renderingDeviceID
- bool m\_fButtonLeft
- dz\_view\_t m\_viewColor
- dz\_window\_t m\_windowColor
- dz\_renderingdevicecontext\_t m\_renderingDeviceContextColor
- dz\_scene\_t m\_sceneColor
- dz\_scenecontext\_t m\_sceneContextColor
- dz\_view\_t m\_viewBall
- dz\_window\_t m\_windowBall
- dz\_interactor\_t m\_interactorBall
- dz\_scene\_t m\_sceneBall
- dz\_scenecontext\_t m\_sceneContextBall
- dz\_renderingdevicecontext\_t m\_renderingDeviceContextBall
- dz\_light\_t m\_lightBall
- dz\_layer\_t m\_layerBall
- int m\_templateBall
- int m\_presentationBall
- int m\_objectBall
- dz\_layer\_t m\_layerBallSmall
- int m\_templateBallSmall
- int m\_presentationBallSmall
- int m\_objectBallSmall
- double m\_ballX
- double m\_ballY
- double m\_ballZ
- double m\_scaleBall

### Detailed Description

Class for lighting editor using dakit. The purpose of this editor is to give the user of the z-kit a GUI for editing the lighting of a window. Usage: create one instance for every window

Definition at line 14 of file DakitLightingEditor.h.

### Constructor & Destructor Documentation

#### DakitLightingEditor::DakitLightingEditor ()

Definition at line 101 of file DakitLightingEditor.cpp:101

```
102     m_lightingWorld(DZ_INVALID_WORLD),
103     m_lightingWindow(DZ_INVALID_WINDOW),
104     m_numChanges (0)
105     /*****
```

```

106 (
107     lightingEditor_t lightingEditor;
108     static int counter = 0;
109
110     m_pDatabaseEditor = new DakitDatabaseEditor(m_database);
111     m_pDatabaseEditor->setEditCallback(this, selectionCb);
112     m_pDatabaseEditor->setItemType(WEBVIEWER_ITEMTYPE_LIGHTING);
113     memset(&m_webviewerLightingOriginal, 0, sizeof (m_webviewerLightingOriginal));
114
115     dzg_librarycreateworld(&m_worldID);
116     dzg_librarycreaterenderingdevice(m_worldID, DZ_RENDERINGDEVICE_WGL,
&m_renderingDeviceID);
117     dzg_librarycreateinteractor(m_worldID, DZ_INTERACTORTYPE_GISBASE, &m_interactorBall);
118     createObjects();
119
120     if (tblLightingEditors)
121         tblLightingEditors = dbf_createtable(&tblLightingEditorsSpec,
&fldLightingEditorSpec);
122
123     if (!idxLightingEditors)
124         idxLightingEditors = dbf_createindex(&tblLightingEditors,
&idxLightingEditorsSpec,
&idxLightingEditorsFields);
125
126     if (!idxLightingEditorsById)
127         idxLightingEditorsById = dbf_createindex(&tblLightingEditors,
&idxLightingEditorsSpec,
&idxLightingEditorsFieldsById);
128
129     countLightingEditors++;
130     sprintf(m_editorName, "LightingEditor %d", ++counter);
131     strcpy(lightingEditor.name, m_editorName);
132     lightingEditor.worldID = m_worldID;
133     lightingEditor.interactorID = m_interactorBall;
134     lightingEditor.pEditor = this;
135     dbf_insert(&tblLightingEditors, &lightingEditor);
136
137     newLighting();
138
139     newLighting();
140 )

```

## DakitLightingEditor::~DakitLightingEditor ()

```

Definition at line 143 of file DakitLightingEditor.cpp:145 (
146     lightingEditor_t lightingEditor;
147
148     strcpy(lightingEditor.name, m_editorName);
149     dbf_delete(&idxLightingEditors, &lightingEditor);
150
151     if (--countLightingEditors == 0)
152     {
153         dbf_dropindex(&idxLightingEditorsById);
154         idxLightingEditorsById = 0;
155         dbf_dropindex(&idxLightingEditors);
156         idxLightingEditors = 0;
157         dbf_droptable(&tblLightingEditors);
158         tblLightingEditors = 0;
159     }
160     deleteObjects();
161     dzg_librarydeleteinteractor(m_worldID, m_interactorBall);
162     dzg_librarydeleterenderingdevice(m_worldID, m_renderingDeviceID);
163     dzg_librarydeleteworld(m_worldID);
164
165     delete m_pDatabaseEditor;
166 )

```

## Member Function Documentation

### int DakitLightingEditor::displayEditor ()

Display this editor on screen

**Return values:**

0 OK  
-1 failure

Definition at line 174 of file DakitLightingEditor.cpp:176 (

```

177     if (dad_getdialogstate(m_editorName) != DAK_STATE_DISPLAYED)
178     {
179         dad_displaydialog(m_editorName, "LightingEditor", NULL);
180
181         dzg_viewdakitsubframebind(m_worldID, m_viewColor, m_editorName, "colorframe");
182         dzg_windowattachview(m_worldID, m_windowColor, m_viewColor);
183         dzg_windowattachrenderingdevicecontext(m_worldID, m_windowColor,
m_renderingDeviceId, m_renderingDeviceContextColor);
184         dzg_windowattachscene(m_worldID, m_windowColor, m_sceneColor);
185         dzg_windowattachscenecontext(m_worldID, m_windowColor, m_sceneContextColor);
186
187         dzg_viewdakitsubframebind(m_worldID, m_viewBall, m_editorName, "ballframe");
188         dzg_windowattachview(m_worldID, m_windowBall, m_viewBall);
189         dzg_windowattachrenderingdevicecontext(m_worldID, m_windowBall, m_renderingDeviceId,
m_renderingDeviceContextBall);
190         dzg_windowattachinteractor(m_worldID, m_windowBall, m_interactorBall);
191         dzg_windowattachscene(m_worldID, m_windowBall, m_sceneBall);
192         dzg_windowattachscenecontext(m_worldID, m_windowBall, m_sceneContextBall);
193
194         initView();
195     }
196     initEditor();
197     return 0;
198 }

```

### int DakitLightingEditor::closeEditor (bool fDeleteDialog)

Close this editor

**Return values:**

0 OK  
-1 failure

**Parameters:**

*fDeleteDialog* true if dakit dialog needs to be closed

```

Definition at line 206 of file DakitLightingEditor.cpp:208 (
209     if (dad_getdialogstate(m_editorName) != DAK_STATE_DISPLAYED)
210         return -1;
211
212     dzg_windowdetachscenecontext(m_worldID, m_windowColor, m_sceneContextColor);
213     dzg_windowdetachrenderingdevicecontext(m_worldID, m_windowColor, m_renderingDeviceId,
m_renderingDeviceContextColor);
214     dzg_windowdetachview(m_worldID, m_windowColor, m_viewColor);
215
216     dzg_windowdetachscenecontext(m_worldID, m_windowBall, m_sceneContextBall);
217     dzg_windowdetachinteractor(m_worldID, m_windowBall, m_interactorBall);
218     dzg_windowdetachrenderingdevicecontext(m_worldID, m_windowBall, m_renderingDeviceId,
m_renderingDeviceContextBall);
219     dzg_windowdetachview(m_worldID, m_windowBall, m_viewBall);
220
221     if (fDeleteDialog)
222         dad_deletediialog(m_editorName);
223     return 0;
224 )

```

### int DakitLightingEditor::setLighting (WebViewerHeader\_t webViewerHeader, WebViewerLighting\_t webViewerLighting)

Set lighting to be edited

**Return values:**

0 OK  
-1 failure

```

Definition at line 232 of file DakitLightingEditor.cpp:235 (
236     memcpy(&m_webviewerHeader, webViewerHeader, sizeof (m_webviewerHeader));
237     memcpy(&m_webviewerLighting, webViewerLighting, sizeof (m_webviewerLighting));
238     m_numchanges = 0;
239     initEditor();
240     return 0;
241 )

```

int DakitLightingEditor::getLighting (WebViewerHeader\_t & webViewerHeader, WebViewerLighting\_t & webViewerLighting)

Get lighting being edited

Return values:

0 OK

-1 failure

```
Definition at line 249 of file DakitLightingEditor.cpp:252 {
253     memcpy(webViewerHeader, &m_webViewerHeader, sizeof (m_webViewerHeader));
254     memcpy(webViewerLighting, &m_webViewerLighting, sizeof (m_webViewerLighting));
255     return 0;
256 }
```

int DakitLightingEditor::setWindow (dz\_world\_t worldID, dz\_window\_t windowID)

Set window for previewing edited lighting

Return values:

0 OK

-1 failure

```
Definition at line 264 of file DakitLightingEditor.cpp:266 {
267     m_lightingWorld = worldID;
268     m_lightingWindow = windowID;
269     getLightingFromWindow(m_webViewerLightingOriginal);
270     return 0;
271 }
```

DakitLightingEditor \* DakitLightingEditor::findEditor (char \* pEditorName) [static]

Find editor from name

Returns:

pointer to DakitLightingEditor class if found, 0 otherwise

```
Definition at line 50 of file DakitLightingEditor.cpp:52 {
53     lightingEditor_t lightingEditor;
54
55     strncpy(lightingEditor.name, pEditorName, sizeof (lightingEditor.name) - 1);
56     if (!lightingEditors ||
57         dbf_select(idxLightingEditors, &lightingEditor) != DBF_ERROR_OK)
58         return 0;
59     return lightingEditor.pEditor;
60 }
```

DakitLightingEditor \* DakitLightingEditor::findByld (dz\_world\_t worldID, dz\_interactor\_t interactorID) [static]

Find editor from world and interactor

Returns:

pointer to DakitLightingEditor class if found, 0 otherwise

```
Definition at line 67 of file DakitLightingEditor.cpp:69 {
70     lightingEditor_t lightingEditor;
71
72     lightingEditor.worldID = worldID;
73     lightingEditor.interactorID = interactorID;
74     if (dbf_select(idxLightingEditorsByld, &lightingEditor) != DBF_ERROR_OK)
75         return 0;
76     return lightingEditor.pEditor;
77 }
```

int DakitLightingEditor::lightSelect ()

Fill editor with the values from selected light

Return values:

0 OK

-1 failure  
-2 no light selected

```
Definition at line 351 of file DakitLightingEditor.cpp:353 {
354     WebViewerColor3f_t color;
355     dz_lightType_t lightType;
356     dz_vector_t position, direction;
357     int id, pos;
358     char *pName;
359     float h, s, v;
360
361     if (dad_getDialogState(m_editorName) != DAK_STATE_DISPLAYED)
362         return -1;
363     dad_setCurrentDialog(m_editorName);
364     pos = DAK_POSITION_FIRST;
365     if (!dad_tableGetRowSelected("lights", &id, &pos, &pName))
366         return -2;
367
368     if (strcmp(pName, "ambient") == 0)
369     {
370         color = m_webViewerLighting.ambient;
371         lightType = DZ_LIGHTTYPE_INVALID;
372         memset(&position, 0, sizeof (position));
373         memset(&direction, 0, sizeof (direction));
374     }
375     else
376     {
377         FunctionsLighting::getLight(m_webViewerLighting, pos-2,
378                                     lightType, color, position, direction);
379
380         dzg_sceneContextSetTextLightColor(m_worldID, m_sceneBall, m_sceneContextBall,
381                                           m_lightBall,
382                                           color.red, color.green, color.blue);
383         if (lightType == DZ_LIGHTTYPE_LOCAL_DIRECTIONAL)
384             setBallDirection(direction.east, direction.north, direction.height);
385         else if (lightType == DZ_LIGHTTYPE_LOCAL_POINT)
386             setBallPosition(position.east, position.north, position.height);
387
388         rgbToHsv(color.red, color.green, color.blue, &h, &s, &v);
389         dad_setFieldvalue("hue", &h);
390         dad_setFieldvalue("saturation", &s);
391         dad_setFieldvalue("value", &v);
392         dad_setFieldvalue("red", &color.red);
393         dad_setFieldvalue("green", &color.green);
394         dad_setFieldvalue("blue", &color.blue);
395         dad_setFieldvalue("type", &lightType);
396         dad_setFieldvalue("positionE", &position.east);
397         dad_setFieldvalue("positionN", &position.north);
398         dad_setFieldvalue("positionH", &position.height);
399         dad_setFieldvalue("directionE", &direction.east);
400         dad_setFieldvalue("directionN", &direction.north);
401         dad_setFieldvalue("directionH", &direction.height);
402
403         dzg_renderingDeviceOpenGLSetBackgroundColor(m_worldID, m_renderingDeviceID,
404                                                     m_renderingDeviceContextColor,
405                                                     color.red, color.green, color.blue);
406         dzg_windowInvalidate(m_worldID, m_windowColor);
407         return 0;
408     }
```

int DakitLightingEditor::colorChange (bool fHsv)

Update changed light or ambient color

Return values:

0 OK

-1 failure

-2 no light or ambient selected

Parameters:

fHsv true if hsv value changed, false if rgb

```
Definition at line 415 of file DakitLightingEditor.cpp:417 {
418     WebViewerColor3f_t color;
```

```

419 int id, pos;
420 float r, g, b;
421 float h, s, v;
422 char *pName;
423
424 if (dad_getdialogstate(m_editorName) != DAK_STATE_DISPLAYED)
425     return -1;
426 dad_setcurrentdialog(m_editorName);
427 if (!fHsv)
428 {
429     dad_getfieldvalue("hue", &h);
430     dad_getfieldvalue("saturation", &s);
431     dad_getfieldvalue("value", &v);
432     hsvToRgb(h, s, v, &r, &g, &b);
433     dad_setfieldvalue("red", &r);
434     dad_setfieldvalue("green", &g);
435     dad_setfieldvalue("blue", &b);
436 }
437 else
438 {
439     dad_getfieldvalue("red", &r);
440     dad_getfieldvalue("green", &g);
441     dad_getfieldvalue("blue", &b);
442     rgbToHsv(r, g, b, &h, &s, &v);
443     dad_setfieldvalue("hue", &h);
444     dad_setfieldvalue("saturation", &s);
445     dad_setfieldvalue("value", &v);
446 }
447 dzg_renderingdeviceopenglsetbackgroundcolor(m_worldID, m_renderingDeviceID,
448 m_renderingDeviceContextColor, r, g, b);
449 dzg_windowinvalidate(m_worldID, m_windowColor);
450 dzg_scenecontextsetlightcolor(m_worldID, m_sceneBall, m_sceneContextBall, m_lightBall,
451 r, g, b);
452
453 pos = DAK_POSITION_FIRST;
454 if (!dad_tablegetrowselected("lights", &id, &pos, &pName))
455     return -2;
456
457 color.red = r;
458 color.green = g;
459 color.blue = b;
460 if (strcmp(pName, "ambient") == 0)
461     m_webViewerLighting.ambient = color;
462 else
463     m_webViewerLighting.aLights[pos-2].color = color;
464
465 m_numChanges++;
466 setTitle();
467 return 0;
468 }

```

## int DakitLightingEditor::typeChange ()

Update changed light type

**Return values:**

0 OK  
-1 failure  
-2 no light selected  
-3 ambient selected

```

479 dz_lighttype_t lightType;
480 int id, pos;
481 char *pName;
482
483 if (dad_getdialogstate(m_editorName) != DAK_STATE_DISPLAYED)
484     return -1;
485 dad_setcurrentdialog(m_editorName);
486 pos = DAK_POSITION_FIRST;
487 if (!dad_tablegetrowselected("lights", &id, &pos, &pName))
488     return -2;
489 if (strcmp(pName, "ambient") == 0)
490     return -3;
491
492 dad_getfieldvalue("type", &lightType);
493 m_webViewerLighting.aLights[pos-2].lightType = lightType;

```

```

494 lightSelect();
495
496 m_numChanges++;
497 setTitle();
498 return 0;
499 }

```

## int DakitLightingEditor::previewChange ()

Preview checkbox changed, set preview on or off

**Return values:**

0 OK  
-1 failure

```

510 static int preview = 0;
511 int previewWas = preview;
512
513 if (dad_getdialogstate(m_editorName) != DAK_STATE_DISPLAYED)
514     return -1;
515 dad_setcurrentdialog(m_editorName);
516 dad_getfieldvalue("preview", &preview);
517 if (preview)
518     setLightingWindow(m_webViewerLighting); // always change to new
519 else if (preview != previewWas)
520     setLightingWindow(m_webViewerLightingOriginal); // change to original only if it
521     wasn't already set
522     return 0;
523 }

```

## int DakitLightingEditor::createLight ()

Create new light

**Return values:**

0 OK  
-1 failure

```

533 if (m_webViewerLighting.nLights == LIGHTING_MAXLIGHTS)
534     return -1;
535 m_webViewerLighting.nLights++;
536
537 m_numChanges++;
538 initEditor();
539 return 0;
540 }

```

## int DakitLightingEditor::deleteLight ()

Delete selected light

**Return values:**

0 OK  
-1 failure  
-2 no light selected  
-3 ambient selected

```

553 int id, pos;
554 char *pName;
555
556 if (dad_getdialogstate(m_editorName) != DAK_STATE_DISPLAYED)
557     return -1;
558 dad_setcurrentdialog(m_editorName);
559 pos = DAK_POSITION_FIRST;
560 if (!dad_tablegetrowselected("lights", &id, &pos, &pName))
561     return -2;
562 if (strcmp(pName, "ambient") == 0)
563     return -3;
564
565 FunctionsLighting::deleteLight(m_webViewerLighting, pos-2);

```

```

566     m_numChanges++;
567     initEditor();
568     return 0;
569 }
570

```

int DakitLightingEditor::newLighting ()

Create new lighting to be edited

Return values:

```

0 OK
-/ failure

```

```

Definition at line 578 of file DakitLightingEditor.cpp:580 (
581     static int counter = 0;
582
583     memset(&m_webViewerHeader, 0, sizeof (m_webViewerHeader));
584     m_webViewerHeader.webViewerID = WebViewerID::createNew();
585     m_webViewerHeader.itemType = WEBVIEWER_ITEMTYPE_LIGHTING;
586     sprintf(m_webViewerHeader.description, "NEW LIGHTING %d", ++counter);
587     sprintf(m_webViewerHeader.application, "DakitTest");
588     sprintf(m_webViewerHeader.applicationversion, "none");
589     sprintf(m_webViewerHeader.applicationhref, "http://www.tekla.com");
590
591     memset(&m_webViewerLighting, 0, sizeof (m_webViewerLighting));
592     m_webViewerLighting.webViewerID = m_webViewerHeader.webViewerID;
593     m_numChanges = 0;
594     initEditor();
595     return 0;
596 )

```

int DakitLightingEditor::newLightingFromWindow ()

Create new lighting using lighting values from user defined window

Return values:

```

0 OK
-/ failure

```

```

Definition at line 604 of file DakitLightingEditor.cpp:606 (
607     if (getLightingFromWindow(m_webViewerLighting) != 0)
608         return -1;
609
610     sprintf(m_webViewerHeader.description, "WINDOW LIGHTING");
611     m_webViewerHeader.webViewerID = m_webViewerLighting.webViewerID =
WebViewerID::createNew();
612     m_numChanges = 0;
613     initEditor();
614     return 0;
615 )

```

int DakitLightingEditor::importLighting ()

Import lighting from database

Return values:

```

0 OK
-/ failure

```

```

Definition at line 623 of file DakitLightingEditor.cpp:625 (
626     m_pDatabaseEditor->displayEditor(m_editorName);
627     return 0;
628 )

```

int DakitLightingEditor::exportLighting ()

Export lighting to database

Return values:

```

0 OK
-/ failure

```

```

Definition at line 636 of file DakitLightingEditor.cpp:638 (
639     WebViewerLighting_t lighting;
640     bool
641         updated;
642
643     if (m_database.selectLighting(m_webViewerHeader, lighting,
m_webViewerLighting.webViewerID))
644         updated = m_database.updateLighting(m_webViewerHeader, m_webViewerLighting);
645     else
646         updated = m_database.insertLighting(m_webViewerHeader, m_webViewerLighting);
647
648     if (updated)
649         m_numChanges = 0;
650     setTitle();
651     m_pDatabaseEditor->displayEditor(m_editorName);
652     return 0;
653 )

```

int DakitLightingEditor::initEditor () [private]

Init editor by filling lights list

Return values:

```

0 OK
-/ failure

```

```

Definition at line 725 of file DakitLightingEditor.cpp:727 (
728     unsigned int i;
729     char pName[256];
730
731     if (dad_getdialogstate(m_editorName) != DAK_STATE_DISPLAYED)
732         return -1;
733     dad_setcurrentdialog(m_editorName);
734     dad_tabledeletearrow("lights", DAK_POSITION_USEPOS, DAK_POSITION_ALL, NULL);
735     dad_tableinsertrow("lights", -1, DAK_POSITION_FIRST, "ambient");
736     for (i = 0; i < m_webViewerLighting.nlights; i++)
737     {
738         sprintf(pName, "light %d", i);
739         dad_tableinsertrow("lights", i, DAK_POSITION_LAST, pName);
740     }
741     setTitle();
742     return 0;
743 }
744

```

int DakitLightingEditor::setTitle () [private]

Set title of lighting editor

Return values:

```

0 OK
-/ failure

```

```

Definition at line 752 of file DakitLightingEditor.cpp:754 (
755     char editorTitle[512], changed[8] = "";
756
757     if (m_numChanges)
758         strcpy(changed, "");
759     sprintf(editorTitle, "Lighting: %s", changed, m_webViewerHeader.description);
760     dad_setframetitle(m_editorName, editorTitle);
761     previewChange();
762     return 0;
763 )

```

int DakitLightingEditor::createObjects () [private]

Create z-kit objects used in lighting editor

Return values:

```

0 OK
-/ failure

```

```

Definition at line 906 of file DakitLightingEditor.cpp:908 (

```

```

909 int materialBall, materialBallSmall;
910
911 // color view
912 dzg_librarycreateView(m_worldID, DZ_VIEWTYPE_EXTERNAL, &m_viewColor);
913 dzg_librarycreatewindow(m_worldID, DZ_WINDOWTYPE_GIBASE, &m_windowColor);
914 dzg_librarycreaterenderingdevicecontext(m_worldID, m_renderingDeviceID,
915     &m_renderingDeviceContextColor);
916 dzg_librarycreatescene(m_worldID, DZ_SCENE_TYPE_GIBASE, &m_sceneColor);
917 dzg_librarycreatescenecontext(m_worldID, m_sceneColor, &m_sceneContextColor);
918
919 // ball view
920 dzg_librarycreateView(m_worldID, DZ_VIEWTYPE_EXTERNAL, &m_viewBall);
921 dzg_librarycreatewindow(m_worldID, DZ_WINDOWTYPE_GIBASE, &m_windowBall);
922 dzg_librarycreaterenderingdevicecontext(m_worldID, m_renderingDeviceID,
923     &m_renderingDeviceContextBall);
924 dzg_librarycreatescene(m_worldID, DZ_SCENE_TYPE_GIBASE, &m_sceneBall);
925 dzg_librarycreatescenecontext(m_worldID, m_sceneBall, &m_sceneContextBall);
926
927 // ball interactor
928 dzg_interactorsetmousebuttoncb(m_worldID, m_interactorBall, mouseButtonDownCB);
929 dzg_interactorsetmousebuttonupcb(m_worldID, m_interactorBall, mouseButtonUpCB);
930 dzg_interactorsetmousemovecb(m_worldID, m_interactorBall, mouseMoveCB);
931 dzg_interactorsetmousewheelcb(m_worldID, m_interactorBall, mouseWheelCB);
932
933 // center ball
934 dzg_libraryobjectcreate(m_worldID, &m_layerBall);
935 dzg_libraryobjectcreatetemplate(m_worldID, m_layerBall, &m_templateBall);
936 dzg_libraryobjectcreaterepresentation(m_worldID, m_layerBall, &m_presentationBall);
937 #if ZKIT_VERSION >= 131
938 dzg_libraryobjectcreatetemplate(m_worldID, m_layerBall, &m_templateBall);
939 #else
940 dzg_libraryobjectcreatetemplate(m_worldID, m_layerBall, m_presentationBall,
941     &materialBall);
942 #endif
943 dzg_libraryobjectcreatetemplate(m_worldID, m_layerBall, m_presentationBall,
944     &materialBall, 0, 0.7f, 0.7f, 0.7f);
945
946 createBall(m_worldID, m_layerBall, m_templateBall, materialBall);
947 dzg_libraryobjectcreateobject(m_worldID, m_layerBall, m_templateBall, &m_objectBall);
948 dzg_sceneattachlayer(m_worldID, m_sceneBall, m_layerBall);
949 dzg_scenecontextsetlayerrepresentation(m_worldID, m_sceneBall, m_sceneContextBall,
950     m_layerBall, m_presentationBall);
951 #if ZKIT_VERSION < 131
952 dzg_scenecontextsetlayertransparency(m_worldID, m_sceneBall, m_sceneContextBall,
953     m_layerBall, true);
954 #endif
955
956 // small ball
957 dzg_libraryobjectcreate(m_worldID, &m_layerBallSmall);
958 dzg_libraryobjectcreatetemplate(m_worldID, m_layerBallSmall, &m_templateBallSmall);
959 dzg_libraryobjectcreaterepresentation(m_worldID, m_layerBallSmall,
960     &m_presentationBallSmall);
961 #if ZKIT_VERSION >= 131
962 dzg_libraryobjectcreatetemplate(m_worldID, m_layerBallSmall, &m_templateBallSmall);
963 #else
964 dzg_libraryobjectcreatetemplate(m_worldID, m_layerBallSmall, m_presentationBallSmall,
965     &materialBallSmall);
966 #endif
967 dzg_libraryobjectcreatetemplate(m_worldID, m_layerBallSmall, m_presentationBallSmall,
968     &materialBallSmall, 0, 1.0, 1.0, 1.0);
969 dzg_libraryobjectcreatetemplate(m_worldID, m_layerBallSmall, m_presentationBallSmall,
970     &materialBallSmall, 0, 1.0, 1.0, 1.0, 1.0);
971
972 createBall(m_worldID, m_layerBallSmall, m_templateBallSmall, materialBallSmall);
973 dzg_libraryobjectcreateobject(m_worldID, m_layerBallSmall, m_templateBallSmall,
974     &m_objectBallSmall);
975 dzg_sceneattachlayer(m_worldID, m_sceneBall, m_layerBallSmall);
976 dzg_scenecontextsetlayerrepresentation(m_worldID, m_sceneBall, m_sceneContextBall,
977     m_layerBallSmall, m_presentationBallSmall);
978
979 return 0;
980 }

```

11

## int DakItLightingEditor::deleteObjects () [private]

Delete z-kit objects used in lighting editor

### Return values:

0 OK

-1 failure

Definition at line 981 of file DakItLightingEditor.cpp:983 (

```

984 // small ball
985 dzg_scenedetachlayer(m_worldID, m_sceneBall, m_layerBallSmall);
986 dzg_libraryobjectdeleteobject(m_worldID, m_layerBallSmall, m_objectBallSmall);
987 dzg_libraryobjectdeleterepresentation(m_worldID, m_layerBallSmall, m_presentationBallSmall);
988 dzg_libraryobjectdeleterepresentation(m_worldID, m_layerBallSmall, m_presentationBallSmall);
989 dzg_libraryobjectdeleterepresentation(m_worldID, m_layerBallSmall, m_presentationBallSmall);
990
991 // center ball
992 dzg_scenedetachlayer(m_worldID, m_sceneBall, m_layerBall);
993 dzg_libraryobjectdeleteobject(m_worldID, m_layerBall, m_objectBall);
994 dzg_libraryobjectdeleterepresentation(m_worldID, m_layerBall, m_presentationBall);
995 dzg_libraryobjectdeleterepresentation(m_worldID, m_layerBall, m_presentationBall);
996 dzg_libraryobjectdeleterepresentation(m_worldID, m_layerBall, m_presentationBall);
997
998 // ball view
999 dzg_librarydeletescenecontext(m_worldID, m_sceneBall, m_sceneContextBall);
1000 dzg_librarydeletescene(m_worldID, m_sceneBall);
1001 dzg_librarydeleterenderingdevicecontext(m_worldID, m_renderingDeviceID,
1002     m_renderingDeviceContextBall);
1003 dzg_librarydeletewindow(m_worldID, m_windowBall);
1004 dzg_librarydeletetemplate(m_worldID, m_viewBall);
1005
1006 // color view
1007 dzg_librarydeletescenecontext(m_worldID, m_sceneColor, m_sceneContextColor);
1008 dzg_librarydeletescene(m_worldID, m_sceneColor);
1009 dzg_librarydeleterenderingdevicecontext(m_worldID, m_renderingDeviceID,
1010     m_renderingDeviceContextColor);
1011 dzg_librarydeletewindow(m_worldID, m_windowColor);
1012 dzg_librarydeletetemplate(m_worldID, m_viewColor);
1013
1014 return 0;
1015 }

```

## int DakItLightingEditor::getLightingFromWindow (WebViewerLighting\_t & webViewerLighting) [private]

Fill lighting struct from values from the user defined window

### Return values:

0 OK

-1 failure

```

774 WebViewerColor3f_t color;
775 dz_scene_t scene;
776 dz_scenecontext_t sceneContext;
777 dz_light_t light;
778 dz_lighttype_t lightType;
779 dz_vector_t position, direction;
780 int nLights;
781
782 if (dzg_windowgetscenecontext(m_lightingWorld, m_lightingWindow,
783     &scene, &sceneContext) != DZ_ERROR_OK)
784     return -1;
785
786 FunctionsLighting::delights(webViewerLighting);
787 dzg_scenecontextgetlightcount(m_lightingWorld, scene, sceneContext, &nLights);
788 for (int i = 0; i < nLights; i++)
789 {
790     dzg_scenecontextgetlight(m_lightingWorld, scene, sceneContext, i, &light);
791     dzg_scenecontextgetlighttype(m_lightingWorld, scene, sceneContext, light,
792     &lightType);
793     dzg_scenecontextgetlightcolor(m_lightingWorld, scene, sceneContext, light,
794     &color.red, &color.green, &color.blue);
795     dzg_scenecontextgetlightposition(m_lightingWorld, scene, sceneContext, light,
796     &position.east, &position.north, &position.height);

```

12

```

794 d3d_scenecontextgetlightdirection(m_lightingWorld, scene, sceneContext, light-
795     direction.east, *direction.north, *direction.height);
796 FunctionsLighting::addLight(webViewerLighting, lightType, color, position,
797     direction);
798 }
799 d3d_scenecontextgetambientlight(m_lightingWorld, scene, sceneContext, *color.red,
800     *color.green, *color.blue);
801 webViewerLighting.ambient = color;
802 return 0;
803 }

```

```
int DakItLightingEditor::setLightingToWindow (WebViewerLighting_t webViewerLighting)  
[private]
```

Set values of lighting to user defined window

Return values:

0 OK

**-/ failure**

Definition at line 808 of file DakitLightingEditor.cpp:810 (

```

Definition at line 808 of file DakitiLightingEditor.cpp:810 {
811     WebViewColor3f_t color;
812     dz_scene t_scene;
813     dz_sceneContext_t sceneContext;
814     dz_light t_light;
815     dz_lightType_t lightType;
816     dz_vector_t position, direction;
817     unsigned int i;
818
819     if (drg_windowGetSceneContext(m_lightingWorld, m_lightingWindow,
820         &scene, &sceneContext) != DZ_ERROR_OK)
821         return -1;
822
823     drg_sceneContextDeleteLights(m_lightingWorld, scene, sceneContext);
824     for (i = 0; i < webViewLighting.nLights; i++)
825     {
826         FunctionsLighting::getLight(webViewLighting, i, lightType, color, position,
827             direction);
828         drg_sceneContextCreateLight(m_lightingWorld, scene, sceneContext, &light);
829         drg_sceneContextSetLightType(m_lightingWorld, scene, sceneContext, light,
830             lightType);
831         drg_sceneContextSetLightColor(m_lightingWorld, scene, sceneContext, light,
832             color.red, color.green, color.blue);
833         drg_sceneContextSetLightPosition(m_lightingWorld, scene, sceneContext, light,
834             position.east, position.north, position.height);
835         drg_sceneContextSetLightDirection(m_lightingWorld, scene, sceneContext, light,
836             direction.east, direction.north, direction.height);
837     }
838     color = webViewLighting.ambient;
839     drg_sceneContextSetAmbientLight(m_lightingWorld, scene, sceneContext, color.red,
840         color.green, color.blue);
841     return 0;
842 }
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

```
int DakitLightingEditor::ballPositionChanged() [private]
```

Small ball position changed, update light direction or position

**Return values:**

0 OK

- / failure

-2 no light selected

-3 ambient selected

```

Definition at line 674 of file DaxiLightingEditor.cpp:676 {
677     dz_lighttype_t lighttype;
678     dz_vector_t position; direction;
679     int id, pos;
680     char *pName;
681
682     if (dad_getdialogstate(m_editorName) != DAK_STATE_DISPLAYED)
683         return -1;
684     dad_setcurrentdialog(m_editorName);
685     pos = DAK_POSITION_FIRST;
686     if (!dad_tablegetrowselected("lights", &id, &pos, &pName))

```

```

587 return -2;
588 if (strcmp(pName, "ambient") == 0)
589 return -3;
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617

```

```
int DakitLightingEditor::initView() [private]
```

### Init z-kit objects in lighting editor

**Return values:**

00K

**- / failure**

Definition at line 1021 of file DakitLightingEditor.cpp:1023 {

```

1024 dz_light t_lightID;
1025
1026 m_ButtonLeft = false;
1027 dz_renderingdeviceopsetbackgroundcolor(m_worldID, m_renderingDeviceID,
1028 m_renderingDeviceContextBall, 0.5, 0.5, 0.5);
1029
1030 // center ball
1031 dzg_layerobjectsetobjectlocation(m_worldID, m_layerBall, m_objectBall, 0.0, 0.0, 0.0);
1032
1033 // small ball
1034 m_ballX = m_ballY = 0.0;
1035 m_ballZ = 1.1;
1036 dzg_layerobjectsetobjectlocation(m_worldID, m_layerBallSmall, m_objectBallSmall,
1037 m_ballX, m_ballY, m_ballZ);
1038 dzg_layerobjectsetobject(m_worldID, m_layerBallSmall, m_objectBallSmall, 0.1, 0.1,
1039 0.1);
1040
1041 // lights
1042 dzg_scenecontextdeletelights(m_worldID, m_sceneBall, m_sceneContextBall);
1043 dzg_scenecontextcreatelight(m_worldID, m_sceneBall, m_sceneContextBall, 6m_lightBall);
1044 dzg_scenecontextsetlighttype(m_worldID, m_sceneBall, m_sceneContextBall, m_lightBall,
1045 DZ_LIGHTTYPE_POINT);
1046 dzg_scenecontextsetlightcolor(m_worldID, m_sceneBall, m_sceneContextBall, m_lightBall,
1047 0.9f, 0.9f, 0.9f);
1048 dzg_scenecontextcreatelight(m_worldID, m_sceneBall, m_sceneContextBall, 6lightID);
1049 dzg_scenecontextsetlighttype(m_worldID, m_sceneBall, m_sceneContextBall, lightID,
1050 DZ_LIGHTTYPE_LOCAL_DIRECTIONAL);
1051 dzg_scenecontextsetlightcolor(m_worldID, m_sceneBall, m_sceneContextBall, lightID,
1052 0.3f, 0.3f, 0.3f);
1053 dzg_scenecontextsetlightdirection(m_worldID, m_sceneBall, m_sceneContextBall, lightID,
1054 0.0, 0.0, 1.0f);
1055
1056 // window
1057 dzg_windowsetlocation(m_worldID, m_windowBall, 0.0, 0.0, 2.5f);
1058 dzg_windowsetdirection(m_worldID, m_windowBall, 0.0, 0.0, -1.0f);
1059 dzg_windowsetupvector(m_worldID, m_windowBall, 0.0, 1.0, 0.0);
1060 dzg_windowsetprojectontype(m_worldID, m_windowBall, DZ_PROJECTIONTYPE_PERSPECTIVE);

```

```

1053 dzg_windowsetviewangle(m_worldID, m_windowBall, 60.0);
1054
1055 setBallDirection(0.0, 0.0, 1.0);
1056 return 0;
1057 }

```

**void DakitLightingEditor::rotateBallHorizontal (float angle/e) [private]**

Rotate small ball around the big ball horizontally

```

Definition at line 1063 of file DakitLightingEditor.cpp.1065 (
1066 double oldx = m_ballx;
1067 double oldz = m_ballz;
1068
1069 m_ballx = (double) (oldx * sin(angle) + oldz * cos(angle));
1070 m_ballz = (double) (oldx * cos(angle) - oldz * sin(angle));
1071 dzg_layerobjectsetobjectlocation(m_worldID, m_layerBallSmall,
1072 m_ballx, m_ballx, m_ballz);
1073
1074 double locx, locy, locz;
1075 dzg_windowgetlocation(m_worldID, m_windowBall, &locx, &locy, &locz);
1076 dzg_scenecontextsetlightposition(m_worldID, m_sceneBall, m_sceneContextBall,
1077 m_lightBall,
1078 m_ballx - locx,
1079 m_ballx - locy,
1080 m_ballz - locz);
1081 return;
1082 }

```

**void DakitLightingEditor::rotateBallVertical (float angle/e) [private]**

Rotate small ball around the big ball vertically

```

Definition at line 1087 of file DakitLightingEditor.cpp.1089 (
1090 double oldy = m_bally;
1091 double oldz = m_ballz;
1092
1093 m_bally = (double) (oldy * cos(angle) - oldz * sin(angle));
1094 m_ballz = (double) (oldy * sin(angle) + oldz * cos(angle));
1095 dzg_layerobjectsetobjectlocation(m_worldID, m_layerBallSmall, m_objectBallSmall,
1096 m_ballx, m_bally, m_ballz);
1097
1098 double locx, locy, locz;
1099 dzg_windowgetlocation(m_worldID, m_windowBall, &locx, &locy, &locz);
1100 dzg_scenecontextsetlightposition(m_worldID, m_sceneBall, m_sceneContextBall,
1101 m_lightBall,
1102 m_ballx - locx,
1103 m_bally - locy,
1104 m_ballz - locz);
1105 return;
1106 }

```

**void DakitLightingEditor::scaleBall (int rotation) [private]**

Scale the big ball

**Parameters:**

*rotation* amount of mouse wheel rotation

```

Definition at line 1111 of file DakitLightingEditor.cpp.1113 (
1114 m_scaleBall *= (1.0 + rotation * 0.1);
1115 if (m_scaleBall > 0.99)
1116 m_scaleBall = 0.99;
1117 dzg_layerobjectsetobject(m_worldID, m_layerBall, m_objectBall, m_scaleBall,
1118 m_scaleBall, m_scaleBall);
1119 return;
1120 }

```

**void DakitLightingEditor::setBallDirection (double x, double y, double z) [private]**

Set small ball direction relative to the big ball

```

Definition at line 1125 of file DakitLightingEditor.cpp.1127 (
1128 double ballradius = sqrt(pow(m_ballx, 2.0) +
1129 pow(m_bally, 2.0) +
1130 pow(m_ballz, 2.0));
1131 double radius = sqrt(x*x + y*y + z*z);
1132 if (radius == 0.0)
1133 z = radius = 0.01;
1134
1135 m_ballx = x * ballradius / radius;
1136 m_bally = y * ballradius / radius;
1137 m_ballz = z * ballradius / radius;
1138 m_scaleBall = 0.6;
1139 dzg_layerobjectsetobject(m_worldID, m_layerBall, m_objectBall, m_scaleBall,
1140 m_scaleBall, m_scaleBall);
1141 rotateBallHorizontal(0);
1142 return;
1143 }

```

**void DakitLightingEditor::setBallPosition (double x, double y, double z) [private]**

Set small ball position relative to the big ball

```

Definition at line 1148 of file DakitLightingEditor.cpp.1150 (
1151 double ballradius = sqrt(pow(m_ballx, 2.0) +
1152 pow(m_bally, 2.0) +
1153 pow(m_ballz, 2.0));
1154 double radius = sqrt(x*x + y*y + z*z);
1155 if (radius == 0.0)
1156 z = radius = 0.01;
1157
1158 m_ballx = x * ballradius / radius;
1159 m_bally = y * ballradius / radius;
1160 m_ballz = z * ballradius / radius;
1161 m_scaleBall = 1.0 / (radius + 1.0);
1162 dzg_layerobjectsetobject(m_worldID, m_layerBall, m_objectBall, m_scaleBall,
1163 m_scaleBall, m_scaleBall);
1164 rotateBallHorizontal(0);
1165 return;
1166 }

```

**int DakitLightingEditor::onMouseButtonDownCB (dz\_world\_t worldID, dz\_interactor\_t interactorID, dz\_window\_t windowID, dz\_mousebutton\_type\_e buttonType) [private]**

Called by the editor when mouse button is pressed down

```

Definition at line 1171 of file DakitLightingEditor.cpp.1176 (
1177 if (buttonType == DZ_MOUSEBUTTONTYPE_LEFT)
1178 m_buttonLeft = true;
1179 return 0;
1180 }

```

**int DakitLightingEditor::onMouseButtonUpCB (dz\_world\_t worldID, dz\_interactor\_t interactorID, dz\_window\_t windowID, dz\_mousebutton\_type\_e buttonType) [private]**

Called by the editor when mouse button is released

```

Definition at line 1186 of file DakitLightingEditor.cpp.1191 (
1192 if (buttonType == DZ_MOUSEBUTTONTYPE_LEFT)
1193 m_buttonLeft = false;
1194 return 0;
1195 }

```

**int DakitLightingEditor::onMouseMoveCB (dz\_world\_t worldID, dz\_interactor\_t interactorID, dz\_window\_t windowID, int x, int y) [private]**

Called by the editor when mouse is moved

```

Definition at line 1201 of file DakitLightingEditor.cpp.1206 (
1207 static int prevx, prevy;
1208 float angle;
1209
1210 if (m_buttonLeft)

```

```

1211 {
1212     angle = (float) (x - prevx) / 50;
1213     rotateBallHorizontal(angle);
1214     angle = (float) (y - prevy) / 50;
1215     rotateBallVertical(angle);
1216     ballPositionChanged();
1217 }
1218 prevx = x;
1219 prevy = y;
1220 return 0;
1221 }
1222 )

```

```

int DakitLightingEditor::onMouseWheelCB (dz_world_t worldID, dz_interactor_t interactorID,
dz_window_t windowID, int rotation) [private]

```

Called by the editor when mouse wheel is rotated

```

Definition at line 1227 of file DakitLightingEditor.cpp.1232 {
1233     scaleBall(rotation);
1234     ballPositionChanged();
1235     return 0;
1236 }

```

```

int DakitLightingEditor::mouseButtonDownCB (dz_world_t worldID, dz_interactor_t interactorID,
dz_window_t windowID, dz_mousebutton_type_e buttonType) [static]

```

Called by the interactor when mouse button is pressed down

```

Definition at line 1242 of file DakitLightingEditor.cpp.1247 {
1248     DakitLightingEditor *pEditor = DakitLightingEditor::findByID(worldID, interactorID);
1249     if (pEditor)
1250         pEditor->onMouseDownCB(worldID, interactorID, windowID, buttonType);
1251     return 0;
1252 }

```

```

int DakitLightingEditor::mouseButtonUpCB (dz_world_t worldID, dz_interactor_t interactorID,
dz_window_t windowID, dz_mousebutton_type_e buttonType) [static]

```

Called by the interactor when mouse button is released

```

Definition at line 1258 of file DakitLightingEditor.cpp.1263 {
1264     DakitLightingEditor *pEditor = DakitLightingEditor::findByID(worldID, interactorID);
1265     if (pEditor)
1266         pEditor->onMouseButtonDownUpCB(worldID, interactorID, windowID, buttonType);
1267     return 0;
1268 }

```

```

int DakitLightingEditor::mouseMoveCB (dz_world_t worldID, dz_interactor_t interactorID,
dz_window_t windowID, int x, int y) [static]

```

Called by the interactor when mouse is moved

```

Definition at line 1274 of file DakitLightingEditor.cpp.1279 {
1280     DakitLightingEditor *pEditor = DakitLightingEditor::findByID(worldID, interactorID);
1281     if (pEditor)
1282         pEditor->onMouseMoveCB(worldID, interactorID, windowID, x, y);
1283     return 0;
1284 }

```

```

int DakitLightingEditor::mouseWheelCB (dz_world_t worldID, dz_interactor_t interactorID,
dz_window_t windowID, int rotation) [static]

```

Called by the interactor when mouse wheel is rotated

```

Definition at line 1290 of file DakitLightingEditor.cpp.1295 {
1296     DakitLightingEditor *pEditor = DakitLightingEditor::findByID(worldID, interactorID);
1297     if (pEditor)
1298         pEditor->onMouseWheelCB(worldID, interactorID, windowID, rotation);
1299     return 0;
1300 }

```

## Member Data Documentation

**dz\_world\_t DakitLightingEditor::m\_lightingWorld** [private]

user defined world

Definition at line 71 of file DakitLightingEditor.h.dz\_world\_t DakitLightingEditor::m\_lightingWindow [private]

user defined window, for previewing edited lighting

Definition at line 72 of file DakitLightingEditor.h.WebViewerHeader\_t DakitLightingEditor::m\_webViewerHeader [private]

header structure for lighting

Definition at line 73 of file DakitLightingEditor.h.WebViewerLighting\_t DakitLightingEditor::m\_webViewerLighting [private]

edited lighting is stored here

Definition at line 74 of file DakitLightingEditor.h.WebViewerLighting\_t DakitLightingEditor::m\_webViewerLightingOriginal [private]

original lighting from window, stored for backup

Definition at line 77 of file DakitLightingEditor.h.WebViewerDatabase DakitLightingEditor::m\_database [private]

database to store lighting

Definition at line 78 of file DakitLightingEditor.h.DakitDatabaseEditor DakitLightingEditor::m\_pDatabaseEditor [private]

editor for the database

Definition at line 79 of file DakitLightingEditor.h.char DakitLightingEditor::m\_editorName[256] [private]

the name of this editor

Definition at line 80 of file DakitLightingEditor.h.int DakitLightingEditor::m\_numChanges [private]

number of changes made to lighting

Definition at line 81 of file DakitLightingEditor.h.dz\_world\_t DakitLightingEditor::m\_worldID [private]

world for color and balls

Definition at line 82 of file DakitLightingEditor.h.dz\_renderingdevice\_t DakitLightingEditor::m\_renderingDeviceID [private]

rendering device for color and balls

Definition at line 83 of file DakitLightingEditor.h.bool DakitLightingEditor::m\_fButtonLeft [private]

true when left mouse button is down

Definition at line 84 of file DakitLightingEditor.h.dz\_view\_t DakitLightingEditor::m\_viewColor [private]

view for color

<p>Definition at line 87 of file DakitLightingEditor.h.dz_window_t DakitLightingEditor::m_windowColor [private]</p> <p>window for color</p>	<p>Definition at line 102 of file DakitLightingEditor.h.int DakitLightingEditor::m_presentationBall [private]</p>
<p>Definition at line 88 of file DakitLightingEditor.h.dz_renderingdevicecontext_t DakitLightingEditor::m_renderingDeviceContextColor [private]</p> <p>rendering device context for color</p>	<p>Definition at line 102 of file DakitLightingEditor.h.int DakitLightingEditor::m_objectBall [private]</p>
<p>Definition at line 89 of file DakitLightingEditor.h.dz_scene_t DakitLightingEditor::m_sceneColor [private]</p> <p>scene for color</p>	<p>Definition at line 102 of file DakitLightingEditor.h.dz_layer_t DakitLightingEditor::m_layerBallSmall [private]</p> <p>layer for small ball</p>
<p>Definition at line 90 of file DakitLightingEditor.h.dz_scenecontext_t_t DakitLightingEditor::m_sceneContextColor [private]</p> <p>scene context for color</p>	<p>Definition at line 103 of file DakitLightingEditor.h.int DakitLightingEditor::m_templateBallSmall [private]</p>
<p>Definition at line 91 of file DakitLightingEditor.h.dz_view_t DakitLightingEditor::m_viewBall [private]</p> <p>view for balls</p>	<p>Definition at line 104 of file DakitLightingEditor.h.int DakitLightingEditor::m_presentationBallSmall [private]</p>
<p>Definition at line 94 of file DakitLightingEditor.h.dz_window_t DakitLightingEditor::m_windowBall [private]</p> <p>window for balls</p>	<p>Definition at line 104 of file DakitLightingEditor.h.int DakitLightingEditor::m_objectBallSmall [private]</p>
<p>Definition at line 95 of file DakitLightingEditor.h.dz_interactor_t DakitLightingEditor::m_interactorBall [private]</p> <p>interactor for balls</p>	<p>Definition at line 104 of file DakitLightingEditor.h.double DakitLightingEditor::m_ballX [private]</p> <p>small ball position</p>
<p>Definition at line 96 of file DakitLightingEditor.h.dz_scene_t DakitLightingEditor::m_sceneBall [private]</p> <p>scene for balls</p>	<p>Definition at line 105 of file DakitLightingEditor.h.double DakitLightingEditor::m_ballY [private]</p> <p>small ball position</p>
<p>Definition at line 97 of file DakitLightingEditor.h.dz_scenecontext_t_t DakitLightingEditor::m_sceneContextBall [private]</p> <p>scene context for balls</p>	<p>Definition at line 106 of file DakitLightingEditor.h.double DakitLightingEditor::m_ballZ [private]</p> <p>small ball position</p>
<p>Definition at line 98 of file DakitLightingEditor.h.dz_renderingdevicecontext_t_t DakitLightingEditor::m_renderingDeviceContextBall [private]</p> <p>rendering device context for balls</p>	<p>Definition at line 107 of file DakitLightingEditor.h.double DakitLightingEditor::m_scaleBall [private]</p> <p>scale for big ball</p>
<p>Definition at line 99 of file DakitLightingEditor.h.dz_light_t DakitLightingEditor::m_lightBall [private]</p> <p>light in scene context of balls</p>	<p>Definition at line 108 of file DakitLightingEditor.h</p> <p>The documentation for this class was generated from the following files:</p> <ul style="list-style-type: none"> <li>DakitLightingEditor.h</li> <li>DakitLightingEditor.cpp</li> </ul>
<p>Definition at line 100 of file DakitLightingEditor.h.dz_layer_t DakitLightingEditor::m_layerBall [private]</p> <p>layer for balls</p>	
<p>Definition at line 101 of file DakitLightingEditor.h.int DakitLightingEditor::m_templateBall [private]</p>	

## Detailed Description

Class for material editor using dakit. The purpose of this editor is to give the user of the z-kit a GUI for editing the materials used in Tekla Structures. Usage: create one instance for each XmlMaterials object.

Definition at line 12 of file DakitMaterialEditor.h.

## Constructor & Destructor Documentation

### DakitMaterialEditor::DakitMaterialEditor (XmlMaterials & xmlMaterials)

Definition at line 62 of file DakitMaterialEditor.cpp:62

```
62 :
63     m_xmlMaterials(xmlMaterials),
64     m_fupdateEditor(true)
65     /*-----*/
66 {
67     materialEditor_t materialEditor;
68     static int counter = 0;
69
70     dzg_librarycreateWorld(&m_worldID);
71     dzg_librarycreateView(m_worldID, DZ_VIEWTYPE_EXTERNAL, &m_viewID);
72     dzg_librarycreateWindow(m_worldID, DZ_WINDOWTYPE_GISBASE, &m_windowID);
73     dzg_librarycreaterenderingdevice(m_worldID, DZ_RENDERINGDEVICE_WGL,
74                                     &m_renderingDeviceID);
75     dzg_librarycreaterenderingdevicecontext(m_worldID, m_renderingDeviceID,
76                                             &m_renderingDeviceContextID);
77     dzg_librarycreatescene(m_worldID, DZ_SCENETYPE_GISBASE, &m_sceneID);
78     dzg_librarycreatescenecontext(m_worldID, m_sceneID, &m_sceneContextID);
79
80     // edgeline view
81     dzg_librarycreateView(m_worldID, DZ_VIEWTYPE_EXTERNAL, &m_viewEdge);
82     dzg_librarycreateWindow(m_worldID, DZ_WINDOWTYPE_GISBASE, &m_windowEdge);
83     dzg_librarycreaterenderingdevicecontext(m_worldID, m_renderingDeviceID,
84                                             &m_renderingDeviceContextEdge);
85
86     if (tblMaterialEditors = dbf_createtable(&tblMaterialEditorsSpec,
87                                             fIdMaterialEditorsSpec);
88         idxMaterialEditors = dbf_createindex(tblMaterialEditors,
89                                             fIdxMaterialEditorsFields);
90         countMaterialEditors++;
91         sprintf(m_editorName, "MaterialEditor_%d", ++counter);
92         strcpy(materialEditor.name, m_editorName);
93         materialEditor.pEditor = this;
94         dbf_insert(tblMaterialEditors, &materialEditor);
95
96         m_xmlMaterials.attachObserver(this, false);
97 }
```

### DakitMaterialEditor::~DakitMaterialEditor ()

Definition at line 100 of file DakitMaterialEditor.cpp:102

```
103 materialEditor_t materialEditor;
104
105 m_xmlMaterials.detachObserver(this);
106
107 strcpy(materialEditor.name, m_editorName);
108 dbf_delete(idxMaterialEditors, &materialEditor);
109
110 if (--countMaterialEditors == 0)
111 {
112     dbf_dropindex(idxMaterialEditors);
113     idxMaterialEditors = 0;
114     dbf_droptable(tblMaterialEditors);
115     tblMaterialEditors = 0;
116 }
```

## DakitMaterialEditor Class Reference

#include <DakitMaterialEditor.h>

### Public Member Functions

- DakitMaterialEditor (XmlMaterials &xmlMaterials)
- ~DakitMaterialEditor ()
- bool displayEditor ()
- bool closeEditor (bool fDeleteDialog)
- bool showMaterial (const WebViewerID &materialID)
- bool materialSelected ()
- bool descriptionChange ()
- bool colorChange (bool fHsv)
- bool edgeColorChange (bool fHsv)
- bool shininessChange ()
- bool linewidthChange ()
- bool effectChange ()
- bool edgeEffectChange ()
- bool revert ()
- bool save ()
- bool export (char \*filename)
- bool createMaterial ()
- bool deleteMaterial ()

### Static Public Member Functions

- DakitMaterialEditor \* findEditor (char \*pEditorName)

### Protected Member Functions

- void onInsert (const XmlMaterials &materials, const WebViewerID &materialID, const MaterialDef\_t &material)
- void onUpdate (const XmlMaterials &materials, const WebViewerID &materialID, const MaterialDef\_t &material)
- void onDelete (const XmlMaterials &materials, const WebViewerID &materialID, const MaterialDef\_t &material)

### Private Member Functions

- bool initMaterialsList ()
- bool updateMaterialsList (const WebViewerID &updateMaterialID)
- bool setTitle ()
- bool materialChanged (const WebViewerID &materialID, const MaterialDef\_t &materialDef)
- bool getSelectedMaterial (WebViewerID &materialID, MaterialDef\_t &materialDef)
- DakitMaterialEditor operator= (const DakitMaterialEditor &other)

### Private Attributes

- XmlMaterials & m\_xmlMaterials
- char m\_editorName [256]
- bool m\_fupdateEditor
- dz\_world\_t m\_worldID
- dz\_renderingdevice\_t m\_renderingDeviceID
- dz\_view\_t m\_viewID
- dz\_window\_t m\_windowID
- dz\_renderingdevicecontext\_t m\_renderingDeviceContextID
- dz\_view\_t m\_viewEdge
- dz\_window\_t m\_windowEdge
- dz\_renderingdevicecontext\_t m\_renderingDeviceContextEdge
- dz\_scene\_t m\_sceneID
- dz\_scenecontext\_t m\_sceneContextID

```

116 )
117 // edgeline view
118 dzg_librarydeleterenderingdevicecontext(m_worldID, m_renderingDeviceID,
119 renderingDeviceContextID);
120 dzg_librarydeletewindow(m_worldID, m_windowEdge);
121 dzg_librarydeleteview(m_worldID, m_viewEdge);
122
123 dzg_librarydeletescenecontext(m_worldID, m_sceneID, m_sceneContextID);
124 dzg_librarydeletescene(m_worldID, m_sceneID);
125 dzg_librarydeleterenderingdevicecontext(m_worldID, m_renderingDeviceID,
126 renderingDeviceContextID);
127 dzg_librarydeleterenderingdevice(m_worldID, m_renderingDeviceID);
128 dzg_librarydeletewindow(m_worldID, m_windowID);
129 dzg_librarydeleteworld(m_worldID);
130 }

```

## Member Function Documentation

### bool DakitMaterialEditor::displayEditor ()

Display this editor on screen

**Returns:**  
true if successful, false if not

```

Definition at line 139 of file DakitMaterialEditor.cpp:141 {
142 if (dad_getdialogstate(m_editorName) != DAK_STATE_DISPLAYED)
143 {
144 if (dao_findobject(DAK_ROOT, DAO_DIALOG, "MaterialEditor", DAK_FIRST) == DAK_NULL)
145 if (dad_readmadedetail("dakitmaterialeditor", NULL) != DAK_ERROR_OK)
146 return false;
147
148 dad_displaydialog(m_editorName, "MaterialEditor", NULL);
149
150 dzg_viewsdakitsubframebind(m_worldID, m_viewID, m_editorName, "colorframe");
151 dzg_windowattachview(m_worldID, m_windowID, m_viewID);
152 dzg_windowattachrenderingdevicecontext(m_worldID, m_windowID, m_renderingDeviceID,
153 renderingDeviceContextID);
154 dzg_windowattachscene(m_worldID, m_windowID, m_sceneID);
155 dzg_windowattachscenecontext(m_worldID, m_windowID, m_sceneContextID);
156
157 // edgeline view
158 dzg_viewsdakitsubframebind(m_worldID, m_viewEdge, m_editorName, "colorframe_edge");
159 dzg_windowattachview(m_worldID, m_windowEdge, m_viewEdge);
160 dzg_windowattachrenderingdevicecontext(m_worldID, m_windowEdge, m_renderingDeviceID,
161 renderingDeviceContextEdge);
162 dzg_windowattachscene(m_worldID, m_windowEdge, m_sceneID);
163 dzg_windowattachscenecontext(m_worldID, m_windowEdge, m_sceneContextID);
164
165 initMaterialsList();
166 setTitle();
167 dzg_raiseframe(m_editorName);
168 dzg_raiseframe(m_editorName);
169 dzg_deiconifyframe(m_editorName);
170 return true;
171 }
172 }

```

### bool DakitMaterialEditor::closeEditor (bool fDeleteDialog)

Close this editor

**Returns:**  
true if successful, false if not

**Parameters:**

*fDeleteDialog* true if dakit dialog needs to be closed

```

Definition at line 175 of file DakitMaterialEditor.cpp:177 {
178 if (dad_getdialogstate(m_editorName) != DAK_STATE_DISPLAYED)
179 return false;
180
181 dzg_windowdetachscenecontext(m_worldID, m_windowID, m_sceneContextID);

```

```

182 dzg_windowdetachrenderingdevicecontext(m_worldID, m_windowID, m_renderingDeviceID,
183 renderingDeviceContextID);
184 dzg_windowdetachview(m_worldID, m_windowID, m_viewID);
185
186 // edgeline view
187 dzg_windowdetachscenecontext(m_worldID, m_windowEdge, m_sceneContextID);
188 dzg_windowdetachrenderingdevicecontext(m_worldID, m_windowEdge, m_renderingDeviceID,
189 renderingDeviceContextEdge);
190 dzg_windowdetachview(m_worldID, m_windowEdge, m_viewEdge);
191 if (fDeleteDialog)
192 dad_deletediolog(m_editorName);
193 return true;
194 }

```

### bool DakitMaterialEditor::showWebViewerID (const WebViewerID & materialID)

Scroll material list and select given material

**Returns:**  
true if successful, false if not

```

Definition at line 200 of file DakitMaterialEditor.cpp:202 {
203 int id, position;
204
205 if (dad_getdialogstate(m_editorName) != DAK_STATE_DISPLAYED)
206 return false;
207
208 WebViewerID webViewerID;
209 XmlMaterialsItemIterator itemIterator = m_xmlMaterials.createItemIterator();
210 for (int iMaterial = 1; itemIterator.getNext(webViewerID); iMaterial++)
211 if (webViewerID == materialID)
212 {
213 id = iMaterial; // simple solution, id set in initMaterialsList()
214 position = DAK_POSITION_USED;
215 dad_setcurrentdialog(m_editorName);
216 if (dad_tablesetrowselected("materials", id, position, NULL, 1) != DAK_ERROR_OK)
217 return false;
218 dad_tableshowrow("materials", id, position, NULL);
219 materialSelected();
220 return true;
221 }
222 return false;
223 }

```

### DakitMaterialEditor \* DakitMaterialEditor::findEditor (char \* pEditorName) [static]

Find editor from name

**Returns:**  
pointer to DakitMaterialEditor class if found, 0 otherwise

```

Definition at line 45 of file DakitMaterialEditor.cpp:47 {
48 materialEditor_t materialEditor;
49
50 materialEditor.name[sizeof (materialEditor.name) - 1] = 0;
51 strcpy(materialEditor.name, pEditorName, sizeof (materialEditor.name) - 1);
52 if (!iDkMaterialsEditors)
53 ddf_select(iDkMaterialsEditors, &materialEditor) != DDF_ERROR_OK)
54 return 0;
55 return materialEditor.pEditor;
56 }

```

### bool DakitMaterialEditor::materialSelected ()

Fill editor with values from selected material

**Returns:**  
true if successful, false if not

```

Definition at line 302 of file DakitMaterialEditor.cpp:304 {
305 int intshininess, intlinewidth;
306 int intlighting, intfog, intfrontbuffer, intdynamicclipping, intforcevisibility;
307 int intlightingedge, intfogedge, intfrontbufferedge, intforcedrawedge;

```

```

308 float r, g, b, a;
309 float h, s, v;
310 float er, eg, eb, ea;
311 float eh, es, ev;
312 WebViewerID materialID;
313 MaterialDef_t materialDef;
314 if (!getSelectedMaterial(materialID, materialDef))
315     memset(&materialDef, 0, sizeof materialDef);
316
317 if (dad_getdialogstate(m_editorName) != DAK_STATE_DISPLAYED)
318     return false;
319
320 dad_setcurrentdialog(m_editorName);
321 r = materialDef.materialColor[0];
322 g = materialDef.materialColor[1];
323 b = materialDef.materialColor[2];
324 a = materialDef.materialColor[3];
325 er = materialDef.edgelineColor[0];
326 eg = materialDef.edgelineColor[1];
327 eb = materialDef.edgelineColor[2];
328 ea = materialDef.edgelineColor[3];
329
330 int shininess = (int) materialDef.shininess;
331 int linewidth = (int) materialDef.linewidth;
332 int lighting = (int) materialDef.lightingEffect;
333 int fog = (int) materialDef.fogEffect;
334 int frontbuffer = (int) materialDef.frontbuffer;
335 int dynamicclipping = (int) materialDef.dynamicClipping;
336 int forcevisibility = (int) materialDef.forceVisibility;
337 int lightingedge = (int) materialDef.edgelineLightingEffect;
338 int fogedge = (int) materialDef.edgelineFogEffect;
339 int frontbufferedge = (int) materialDef.edgelineFrontbuffer;
340 int forcedrawedge = (int) materialDef.edgelineForcedraw;
341 rgbToHsv(r, g, b, sh, es, ev);
342 rgbToHsv(er, eg, eb, eh, es, ev);
343 dad_setfieldvalue("hue", h);
344 dad_setfieldvalue("saturation", s);
345 dad_setfieldvalue("value", v);
346 dad_setfieldvalue("red", r);
347 dad_setfieldvalue("green", g);
348 dad_setfieldvalue("blue", b);
349 dad_setfieldvalue("alpha", a);
350 dad_setfieldvalue("alpha digit", sh);
351 dad_setfieldvalue("lighting", sIntLighting);
352 dad_setfieldvalue("fog", sIntFog);
353 dad_setfieldvalue("frontbuffer", sIntFrontbuffer);
354 dad_setfieldvalue("dynamicclipping", sIntDynamicclipping);
355 dad_setfieldvalue("forcevisibility", sIntForcevisibility);
356 dad_setfieldvalue("hue edge", eh);
357 dad_setfieldvalue("saturation edge", es);
358 dad_setfieldvalue("value edge", ev);
359 dad_setfieldvalue("red edge", er);
360 dad_setfieldvalue("green edge", eg);
361 dad_setfieldvalue("blue edge", eb);
362 dad_setfieldvalue("alpha edge", ea);
363 dad_setfieldvalue("alpha digit edge", sh);
364 dad_setfieldvalue("lighting edge", sIntLightingedge);
365 dad_setfieldvalue("fog edge", sIntFogedge);
366 dad_setfieldvalue("frontbuffer edge", sIntFrontbufferedge);
367 dad_setfieldvalue("forcedraw edge", sIntForcedrawedge);
368 dad_setfieldvalue("shininess", sIntShininess);
369 dad_setfieldvalue("linewidth", sIntLinewidth);
370 dad_setfieldvalue("description", &materialDef.description);
371
372 int sensitivity = 1;
373 if (materialDef.fBuiltIn)
374     sensitivity = 0;
375
376 dad_setfieldattribute("description", DAK_FIELD_SENSITIVITY, &sensitivity);
377 dad_setfieldattribute("hue", DAK_FIELD_SENSITIVITY, &sensitivity);
378 dad_setfieldattribute("saturation", DAK_FIELD_SENSITIVITY, &sensitivity);
379 dad_setfieldattribute("value", DAK_FIELD_SENSITIVITY, &sensitivity);
380 dad_setfieldattribute("red", DAK_FIELD_SENSITIVITY, &sensitivity);
381 dad_setfieldattribute("green", DAK_FIELD_SENSITIVITY, &sensitivity);
382 dad_setfieldattribute("blue", DAK_FIELD_SENSITIVITY, &sensitivity);
383 dad_setfieldattribute("alpha digit", DAK_FIELD_SENSITIVITY, &sensitivity);
384 dad_setfieldattribute("lighting", DAK_FIELD_SENSITIVITY, &sensitivity);
385 dad_setfieldattribute("fog", DAK_FIELD_SENSITIVITY, &sensitivity);
386 dad_setfieldattribute("frontbuffer", DAK_FIELD_SENSITIVITY, &sensitivity);
387 dad_setfieldattribute("dynamicclipping", DAK_FIELD_SENSITIVITY, &sensitivity);

```

```

388 dad_setfieldattribute("forcevisibility", DAK_FIELD_SENSITIVITY, &sensitivity);
389 dad_setfieldattribute("hue edge", DAK_FIELD_SENSITIVITY, &sensitivity);
390 dad_setfieldattribute("saturation edge", DAK_FIELD_SENSITIVITY, &sensitivity);
391 dad_setfieldattribute("value edge", DAK_FIELD_SENSITIVITY, &sensitivity);
392 dad_setfieldattribute("red edge", DAK_FIELD_SENSITIVITY, &sensitivity);
393 dad_setfieldattribute("green edge", DAK_FIELD_SENSITIVITY, &sensitivity);
394 dad_setfieldattribute("blue edge", DAK_FIELD_SENSITIVITY, &sensitivity);
395 dad_setfieldattribute("alpha edge", DAK_FIELD_SENSITIVITY, &sensitivity);
396 dad_setfieldattribute("alpha digit edge", DAK_FIELD_SENSITIVITY, &sensitivity);
397 dad_setfieldattribute("lighting edge", DAK_FIELD_SENSITIVITY, &sensitivity);
398 dad_setfieldattribute("fog edge", DAK_FIELD_SENSITIVITY, &sensitivity);
399 dad_setfieldattribute("frontbuffer edge", DAK_FIELD_SENSITIVITY, &sensitivity);
400 dad_setfieldattribute("forcedraw edge", DAK_FIELD_SENSITIVITY, &sensitivity);
401 dad_setfieldattribute("shininess", DAK_FIELD_SENSITIVITY, &sensitivity);
402 dad_setfieldattribute("linewidth", DAK_FIELD_SENSITIVITY, &sensitivity);
403
404 dzg_renderingdeviceopenglsetbackgroundcolor(m_worldID, m_renderingDeviceID,
m_renderingDeviceContextID, r, g, b);
405 dzg_windowinvalidate(m_worldID, m_windowID);
406
407 dzg_renderingdeviceopenglsetbackgroundcolor(m_worldID, m_renderingDeviceID,
m_renderingDeviceContextEdge, er, eg, eb);
408 dzg_windowinvalidate(m_worldID, m_windowEdge);
409 return true;
410 }

```

## bool DakitMaterialEditor::descriptionChange ()

Update changed material description

### Returns:

true if successful, false if not

```

Definition at line 417 of file DakitMaterialEditor.cpp:419 (
420 char      description[256];
421 WebViewerID materialID;
422 MaterialDef_t materialDef;
423
424 if (!getSelectedMaterial(materialID, materialDef))
425     return false;
426
427 dad_setcurrentdialog(m_editorName);
428 dad_getfieldvalue("description", &description);
429 memset(materialDef.description, 0, sizeof (materialDef.description));
430 strncpy(materialDef.description, description, sizeof (materialDef.description));
431
432 materialChanged(materialID, materialDef);
433 return true;
434 )

```

## bool DakitMaterialEditor::colorChange (bool fHsv)

Update changed material surface color

### Returns:

true if successful, false if not

### Parameters:

fHsv true if hsv value changed, false if rgb

```

Definition at line 441 of file DakitMaterialEditor.cpp:443 (
444 float r, g, b, a;
445 float h, s, v;
446 WebViewerID materialID;
447 MaterialDef_t materialDef;
448 bool selected;
449
450 selected = getSelectedMaterial(materialID, materialDef);
451 if (!selected)
452     memset(&materialDef, 0, sizeof materialDef);
453
454 if (dad_getdialogstate(m_editorName) != DAK_STATE_DISPLAYED)
455     return false;
456 dad_setcurrentdialog(m_editorName);

```

```

457 if (fHsv)
458 {
459     dad_getfieldvalue("hue", &h);
460     dad_getfieldvalue("saturation", &s);
461     dad_getfieldvalue("value", &v);
462     dad_getfieldvalue("alpha", &a);
463     hsvToRgb(h, s, v, &r, &g, &b);
464     dad_setfieldvalue("red", &r);
465     dad_setfieldvalue("green", &g);
466     dad_setfieldvalue("blue", &b);
467     dad_setfieldvalue("alpha_digit", &a);
468 }
469 else
470 {
471     dad_getfieldvalue("red", &r);
472     dad_getfieldvalue("green", &g);
473     dad_getfieldvalue("blue", &b);
474     dad_getfieldvalue("alpha_digit", &a);
475     rgbToHsv(r, g, b, &h, &s, &v);
476     dad_setfieldvalue("hue", &h);
477     dad_setfieldvalue("saturation", &s);
478     dad_setfieldvalue("value", &v);
479     dad_setfieldvalue("alpha", &a);
480 }
481 materialDef.materialColor[0] = r;
482 materialDef.materialColor[1] = g;
483 materialDef.materialColor[2] = b;
484 materialDef.materialColor[3] = a;
485
486 dzg_renderingDeviceOpenGLsetBackgroundColor(m_worldID, m_renderingDeviceID,
487 dzg_renderingDeviceContextID, r, g, b);
488 dzg_windowinvalidate(m_worldID, m_windowID);
489
490 if (selected)
491     materialChanged(materialID, materialDef);
492 return selected;
493 }

```

## bool DakitMaterialEditor::edgeColorChange (bool fHsv)

Update changed material edge color

### Returns:

true if successful, false if not

```

Definition at line 499 of file DakitMaterialEditor.cpp:501 {
502 float r, g, b, a;
503 float h, s, v;
504 WebViewerID materialID;
505 MaterialDef_t materialDef;
506 bool selected;
507
508 selected = getSelectedMaterial(materialID, materialDef);
509 if (!selected)
510     memset(&materialDef, 0, sizeof materialDef);
511
512 if (dad_getDialogState(m_editorName) != DAK_STATE_DISPLAYED)
513     return false;
514 dad_setcurrentDialog(m_editorName);
515 if (fHsv)
516 {
517     dad_getfieldvalue("hue", &h);
518     dad_getfieldvalue("saturation", &s);
519     dad_getfieldvalue("value", &v);
520     dad_getfieldvalue("alpha", &a);
521     hsvToRgb(h, s, v, &r, &g, &b);
522     dad_setfieldvalue("red", &r);
523     dad_setfieldvalue("green", &g);
524     dad_setfieldvalue("blue", &b);
525     dad_setfieldvalue("alpha_digit", &a);
526 }
527 else
528 {
529     dad_getfieldvalue("red", &r);
530     dad_getfieldvalue("green", &g);
531     dad_getfieldvalue("blue", &b);
532     dad_getfieldvalue("alpha_digit", &a);
533     rgbToHsv(r, g, b, &h, &s, &v);

```

```

534     dad_setfieldvalue("hue", &h);
535     dad_setfieldvalue("saturation", &s);
536     dad_setfieldvalue("value", &v);
537     dad_setfieldvalue("alpha", &a);
538 }
539 materialDef.edgelineColor[0] = r;
540 materialDef.edgelineColor[1] = g;
541 materialDef.edgelineColor[2] = b;
542 materialDef.edgelineColor[3] = a;
543
544 dzg_renderingDeviceOpenGLsetBackgroundColor(m_worldID, m_renderingDeviceID,
545 dzg_renderingDeviceContextEdge, r, g, b);
546 dzg_windowinvalidate(m_worldID, m_windowEdge);
547
548 if (selected)
549     materialChanged(materialID, materialDef);
550 return selected;
551 }

```

## bool DakitMaterialEditor::shininessChange ()

Update changed material shininess

### Returns:

true if successful, false if not

```

Definition at line 557 of file DakitMaterialEditor.cpp:559 {
560 int shininess;
561 WebViewerID materialID;
562 MaterialDef_t materialDef;
563
564 if (!getSelectedMaterial(materialID, materialDef))
565     return false;
566
567 dad_setcurrentDialog(m_editorName);
568 dad_getfieldvalue("shininess", &shininess);
569 materialDef.shininess = (dz_materialshininess_t) shininess;
570
571 materialChanged(materialID, materialDef);
572 return true;
573 }

```

## bool DakitMaterialEditor::linewidthChange ()

Update changed material linewidth

### Returns:

true if successful, false if not

```

Definition at line 580 of file DakitMaterialEditor.cpp:582 {
583 int linewidth;
584 WebViewerID materialID;
585 MaterialDef_t materialDef;
586
587 if (!getSelectedMaterial(materialID, materialDef))
588     return false;
589
590 dad_setcurrentDialog(m_editorName);
591 dad_getfieldvalue("linewidth", &linewidth);
592 materialDef.linewidth = (dz_materiallinewidth_t) linewidth;
593
594 materialChanged(materialID, materialDef);
595 return true;
596 }

```

## bool DakitMaterialEditor::effectChange ()

Update changed material surface effect

### Returns:

true if successful, false if not

```

Definition at line 603 of file DakitMaterialEditor.cpp:605 {
606 int lighting, fog, frontbuffer, dynamicclipping, forcevisibility;
607 WebViewerID materialID;

```

```

608 MaterialDef_t materialDef;
609
610 if (!getSelectedMaterial(materialID, materialDef))
611     return false;
612
613 dad_setcurrentDialog(m_editorName);
614 dad_getfieldvalue("lighting", &lighting);
615 dad_getfieldvalue("fog", &fog);
616 dad_getfieldvalue("frontbuffer", &frontbuffer);
617 dad_getfieldvalue("dynamicclipping", &dynamicclipping);
618 dad_getfieldvalue("forcevisibility", &forcevisibility);
619
620 materialDef.lightingEffect = lighting ? true : false;
621 materialDef.fogEffect = fog ? true : false;
622 materialDef.frontbuffer = frontbuffer ? true : false;
623 materialDef.dynamicclipping = dynamicclipping ? true : false;
624 materialDef.forcevisibility = forcevisibility ? true : false;
625
626 materialChanged(materialID, materialDef);
627 return true;
628 }

```

## bool DakitMaterialEditor::edgeEffectChange ()

Update changed material edge effect

**Returns:**  
true if successful, false if not

```

Definition at line 635 of file DakitMaterialEditor.cpp:637 (
638 int lighting, fog, frontbuffer, forcedraw;
639 WebViewerID materialID;
640 MaterialDef_t materialDef;
641
642 if (!getSelectedMaterial(materialID, materialDef))
643     return false;
644
645 dad_setcurrentDialog(m_editorName);
646 dad_getfieldvalue("lighting_edge", &lighting);
647 dad_getfieldvalue("fog_edge", &fog);
648 dad_getfieldvalue("frontbuffer_edge", &frontbuffer);
649 dad_getfieldvalue("forcedraw_edge", &forcedraw);
650
651 materialDef.edgeLineLightingEffect = lighting ? true : false;
652 materialDef.edgeLineFogEffect = fog ? true : false;
653 materialDef.edgeLineFrontbuffer = frontbuffer ? true : false;
654 materialDef.edgeLineForcedraw = forcedraw ? true : false;
655
656 materialChanged(materialID, materialDef);
657 return true;
658 }

```

## bool DakitMaterialEditor::revert ()

Revert file

**Returns:**  
true if successful, false if not

```

Definition at line 665 of file DakitMaterialEditor.cpp:667 (
668 bool revert;
669
670 m_updateEditor = false;
671 revert = m_xmlMaterials.revert();
672 m_updateEditor = true;
673 initMaterialsList();
674 setTitle();
675 return revert;
676 }

```

## bool DakitMaterialEditor::save ()

Save file

**Returns:**  
true if successful, false if not

```

Definition at line 683 of file DakitMaterialEditor.cpp:685 (
686 bool save;
687
688 save = m_xmlMaterials.save();
689 setTitle();
690 return save;
691 )

```

## bool DakitMaterialEditor::export (char\* filename)

Export materials to file

**Returns:**  
true if successful, false if not

```

Definition at line 698 of file DakitMaterialEditor.cpp:700 (
701 bool export;
702
703 export = m_xmlMaterials.exportFile(filename);
704 return export;
705 )

```

## bool DakitMaterialEditor::createMaterial ()

Create new material into editor

**Returns:**  
true if successful, false if not

```

Definition at line 712 of file DakitMaterialEditor.cpp:714 (
715 WebViewerID materialID = WebViewerID::createNew();
716 MaterialDef_t materialDef;
717 static int_ idCounter = 0;
718
719 memset(&materialDef, 0, sizeof (materialDef));
720 sprintf(materialDef.description, "NEW MATERIAL id", idCounter++);
721
722 if (!m_xmlMaterials.insertMaterial(materialID, materialDef))
723     return false;
724
725 showMaterial(materialID);
726 return true;
727 )

```

## bool DakitMaterialEditor::deleteMaterial ()

Delete material from editor

**Returns:**  
true if successful, false if not

```

Definition at line 734 of file DakitMaterialEditor.cpp:736 (
737 WebViewerID materialID;
738 MaterialDef_t materialDef;
739
740 if (!getSelectedMaterial(materialID, materialDef))
741     return false;
742
743 return m_xmlMaterials.deleteMaterial(materialID);
744 )

```

**void DakitMaterialEditor::onInsert (const XmlMaterials & materials, const WebViewerID & materialID, const MaterialDef\_t & material) [protected]**

Called when new material is inserted from outside of the editor

```

Definition at line 750 of file DakitMaterialEditor.cpp:754 (
755 initMaterialsList();

```

```

756 setTitle();
757 return;
758 }

```

```

void DakitMaterialEditor::onUpdate (const XmlMaterials & materials, const WebViewID &
materialID, const MaterialDef_t & material) [protected]

```

Called when material is updated from outside of the editor

```

Definition at line 764 of file DakitMaterialEditor.cpp:768 (
769 updateMaterialsList(materialID);
770 materialSelected();
771 setTitle();
772 return;
773 )

```

```

void DakitMaterialEditor::onDelete (const XmlMaterials & materials, const WebViewID & materialID,
const MaterialDef_t & material) [protected]

```

Called when material is deleted from outside of the editor

```

Definition at line 779 of file DakitMaterialEditor.cpp:783 (
784 initMaterialsList();
785 setTitle();
786 return;
787 )

```

```

bool DakitMaterialEditor::initMaterialsList () [private]

```

Init materials list in editor

**Returns:**  
true if successful, false if not

```

Definition at line 794 of file DakitMaterialEditor.cpp:796 (
797 WebViewID materialID;
798 MaterialDef_t materialDef;
799
800 if (!m_fUpdateEditor)
801     return false;
802
803 if (dad_getDialogState(m_editorName) != DAK_STATE_DISPLAYED)
804     return false;
805
806 dad_setCurrentDialog(m_editorName);
807 dad_tableDeleteRow("materials", DAK_POSITION_USEPOS, DAK_POSITION_ALL, NULL);
808
809 XmlMaterialsItemIterator itemIterator = m_xmlMaterials.createItemIterator();
810 for (int iMaterial = 1; itemIterator.getNext(materialID); iMaterial++)
811 {
812     m_xmlMaterials.fetchMaterial(materialID, materialDef);
813     dad_tableInsertRow("materials", iMaterial, DAK_POSITION_LAST,
materialDef.description);
814 }
815 materialSelected();
816 return true;
817 )

```

```

bool DakitMaterialEditor::updateMaterialsList (const WebViewID & updateMaterialID) [private]

```

Update one material in materials list

**Returns:**  
true if successful, false if not

```

Definition at line 823 of file DakitMaterialEditor.cpp:825 (
826 WebViewID materialID;
827 MaterialDef_t materialDef;
828
829 if (dad_getDialogState(m_editorName) != DAK_STATE_DISPLAYED)
830     return false;
831
832 XmlMaterialsItemIterator materialIterator = m_xmlMaterials.createItemIterator();

```

```

833 for (int iMaterial = 1; materialIterator.getNext(materialID); iMaterial++)
834 {
835     if (updateMaterialID == materialID)
836     {
837         m_xmlMaterials.fetchMaterial(materialID, materialDef);
838         int id = iMaterial; // id set in initMaterialsList()
839         int position = DAK_POSITION_USEID;
840         char *description = "";
841         dad_setCurrentDialog(m_editorName);
842         dad_tableGetRow("materials", id, iMaterial, iMaterialDef.description);
843         if (strcmp(description, materialDef.description) != 0)
844         {
845             dad_tableSetRow("materials", id, position, materialDef.description);
846
847             // setrow seems to remove selection, must select again
848             id = DAK_POSITION_USEPOS;
849             position = DAK_POSITION_FIRST;
850             if (dad_tableGetRowSelected("materials", id, iMaterial, iMaterialDef.description))
851             {
852                 dad_tableSetRowSelected("materials", id, position, NULL);
853                 dad_tableSetRowSelected("materials", id, position, NULL, 1);
854             }
855             return true; // description changed
856         }
857         return false;
858     }
859     return false;
860 }
861 )

```

```

bool DakitMaterialEditor::setTitle () [private]

```

Set title of the material editor

**Returns:**  
true if successful, false if not

```

Definition at line 899 of file DakitMaterialEditor.cpp:901 (
902 char editorTitle[512], changed[16] = "";
903 char *filename = "";
904 int size = 0;
905
906 if (!m_fUpdateEditor)
907     return false;
908
909 if (m_xmlMaterials.getFilename(0, size))
910 {
911     filename = (char *) malloc(size);
912     m_xmlMaterials.getFilename(filename, size);
913
914     if (m_xmlMaterials.isChanged())
915         sprintf(changed, "%s",
sprintf(editorTitle, "XmlMaterials: %s", changed, filename);
916         if (*filename)
917             free(filename);
918         dad_setFrameTitle(m_editorName, editorTitle);
919         return true;
920     }
921 )
922 )

```

```

bool DakitMaterialEditor::materialChanged (const WebViewID & materialID, const MaterialDef_t &
materialDef) [private]

```

Update material from new definitions

**Returns:**  
true if successful, false if not

**Parameters:**

*materialID* material to update

*materialDef* new definitions

```
Definition at line 929 of file DakitMaterialEditor.cpp:932 (
933   MaterialDef_t oldMaterialDef;
934
935   if (!m_xmlMaterials.fetchMaterial(materialID, oldMaterialDef))
936       return false;
937
938   m_xmlMaterials.updateMaterial(materialID, materialDef);
939   setTitle();
940   return true;
941 )
```

**bool DakitMaterialEditor::getSelectedMaterial (WebViewID & materialID, MaterialDef\_t & materialDef) [private]**

Returns selected material in materials list

**Returns:**  
true if successful, false if not

```
Definition at line 868 of file DakitMaterialEditor.cpp:871 (
872   int id, position;
873   char *description;
874
875   if (dad_getdialogstate(m_editorName) != DAK_STATE_DISPLAYED)
876       return false;
877   dad_setcurrentdialog(m_editorName);
878   position = DAK_POSITION_FIRST;
879   if (!dad_tablegetrowselected("materials", &id, &position, &description))
880       return false;
881
882   WebViewID webViewID;
883   XmlMaterialsItemIterator itemIterator = m_xmlMaterials.createItemIterator();
884   for (int iMaterial = 1; itemIterator.getNext(webViewID); iMaterial++)
885       if (iMaterial == id) // simple solution, id set in initMaterialsList()
886           {
887               materialID = webViewID;
888               m_xmlMaterials.fetchMaterial(materialID, materialDef);
889               return true;
890           }
891   return false;
892 )
```

**DakitMaterialEditor DakitMaterialEditor::operator= (const DakitMaterialEditor & other) [private]**

### Member Data Documentation

**XmlMaterials & DakitMaterialEditor::m\_xmlMaterials [private]**

the materials in this editor are stored here

**Definition at line 52 of file DakitMaterialEditor.h:char DakitMaterialEditor::m\_editorName[256] [private]**

the name of this editor

**Definition at line 55 of file DakitMaterialEditor.h:bool DakitMaterialEditor::m\_fUpdateEditor [private]**

true if the values in the GUI should be updated when they are changed

**Definition at line 56 of file DakitMaterialEditor.h:dz\_world\_t DakitMaterialEditor::m\_worldID [private]**

world for colors

**Definition at line 59 of file DakitMaterialEditor.h:dz\_renderingdevice\_t DakitMaterialEditor::m\_renderingDeviceID [private]**

rendering device for colors

**Definition at line 60 of file DakitMaterialEditor.h:dz\_view\_t DakitMaterialEditor::m\_viewID [private]**  
view for surface color

**Definition at line 61 of file DakitMaterialEditor.h:dz\_window\_t DakitMaterialEditor::m\_windowID [private]**  
window for surface color

**Definition at line 62 of file DakitMaterialEditor.h:dz\_renderingdevicecontext\_t DakitMaterialEditor::m\_renderingDeviceContextID [private]**  
rendering device context for surface color

**Definition at line 63 of file DakitMaterialEditor.h:dz\_view\_t DakitMaterialEditor::m\_viewEdge [private]**  
view for edge color

**Definition at line 64 of file DakitMaterialEditor.h:dz\_window\_t DakitMaterialEditor::m\_windowEdge [private]**  
window for edge color

**Definition at line 65 of file DakitMaterialEditor.h:dz\_renderingdevicecontext\_t DakitMaterialEditor::m\_renderingDeviceContextEdge [private]**  
rendering device context for edge color

**Definition at line 66 of file DakitMaterialEditor.h:dz\_scene\_t DakitMaterialEditor::m\_sceneID [private]**  
scene for colors

**Definition at line 67 of file DakitMaterialEditor.h:dz\_scenecontext\_t DakitMaterialEditor::m\_sceneContextID [private]**  
scene context for colors

Definition at line 68 of file DakitMaterialEditor.h.

The documentation for this class was generated from the following files:

- **DakitMaterialEditor.h**
- **DakitMaterialEditor.cpp**

## DakitPresentationByClassEditor Class Reference

#include <DakitPresentationByClassEditor.h>

### Public Member Functions

- DakitPresentationByClassEditor (XmlPresentationByClass &xmlPresentationByClass, XmlMaterials &xmlMaterials)
- ~DakitPresentationByClassEditor ()
- bool displayEditor ()
- bool closeEditor (bool fDeleteDialog)
- bool setShowMaterialCB (void \*pContext, bool(\*showMaterialCB)(void \*pContext, const WebViewID &materialID))
- bool descriptionChange ()
- bool presentationSelected ()
- bool showMaterial ()
- bool setMaterial ()
- bool revert ()
- bool save ()
- bool export (char \*filename)
- bool createPresentation ()
- bool deletePresentation ()

### Static Public Member Functions

- DakitPresentationByClassEditor \* findEditor (char \*pEditorName)

### Protected Member Functions

- void onInsert (const XmlPresentationByClass &presentation, const WebViewID &presentationID)
- void onDelete (const XmlPresentationByClass &presentation, const WebViewID &presentationID)
- void onUpdate (const XmlPresentationByClass &presentation, const WebViewID &presentationID, PresentationByClassType\_t classType, PresentationByClassState\_t state, const WebViewID &materialID)
- void onInsert (const XmlMaterials &materials, const WebViewID &materialID, const MaterialDef\_t &material)
- void onUpdate (const XmlMaterials &materials, const WebViewID &materialID, const MaterialDef\_t &material)
- void onDelete (const XmlMaterials &materials, const WebViewID &materialID, const MaterialDef\_t &material)

### Private Member Functions

- bool initPresentationList ()
- bool initClassesTable ()
- bool initMaterialsList ()
- bool updateClassesTable (PresentationByClassType\_t classType, PresentationByClassState\_t classState)
- bool updateMaterialsList (const WebViewID &updateMaterialID)
- bool setTitle ()
- bool showPresentation (const WebViewID &presentationID)
- bool getSelectedPresentation (WebViewID &presentationID)
- bool getSelectedMaterial (WebViewID &materialID)
- bool getSelectedClass (PresentationByClassType\_t &classType, PresentationByClassState\_t &classState)
- DakitPresentationByClassEditor operator= (const DakitPresentationByClassEditor &other)

### Private Attributes

- XmlPresentationByClass & m\_xmlPresentationByClass
- XmlMaterials & m\_xmlMaterials
- void \* m\_pShowMaterialContext
- bool(\* m\_pShowMaterialCB )(void \*pContext, const WebViewID &materialID)
- char m\_editorName [256]
- bool m\_fUpdateEditor

## Detailed Description

Class for presentation editor using dakit. Usage: create one instance for each xmlPresentation and provide the materials to be used.

Definition at line 12 of file DakitPresentationByClassEditor.h.

## Constructor & Destructor Documentation

**DakitPresentationByClassEditor::DakitPresentationByClassEditor (XmlPresentationByClass & xmlPresentationByClass, XmlMaterials & xmlMaterials)**

Definition at line 134 of file DakitPresentationByClassEditor.cpp.135

```
136 m_xmlPresentationByClass(xmlPresentationByClass),
137 m_xmlMaterials(xmlMaterials)
138 /*-----*/
139 {
140     presentationEditor_t presentationEditor;
141     static int counter = 0;
142
143     if (!tblPresentationEditors)
144         tblPresentationEditors = dbf_createTable(tblPresentationEditorSpec,
145                                                 fldPresentationEditorSpec);
146     if (!idxPresentationEditors)
147         idxPresentationEditors = dbf_createIndex(tblPresentationEditors,
148                                                 idxPresentationEditorSpec,
149                                                 idxPresentationEditorsFields);
150     countPresentationEditors++;
151     sprintf(m_editorName, "PresentationEditor_%d", ++counter);
152     strcpy(presentationEditor.name, m_editorName);
153     presentationEditor.peditor = this;
154     dbf_insert(tblPresentationEditors, &presentationEditor);
155
156     m_xmlPresentationByClass.attachObserver(this, false);
157     m_xmlMaterials.attachObserver(this, false);
158 }
```

**DakitPresentationByClassEditor::~DakitPresentationByClassEditor ()**

Definition at line 161 of file DakitPresentationByClassEditor.cpp.163

```
164 presentationEditor_t presentationEditor;
165
166 m_xmlMaterials.detachObserver(this);
167 m_xmlPresentationByClass.detachObserver(this);
168
169 strcpy(presentationEditor.name, m_editorName);
170 dbf_delete(idxPresentationEditors, &presentationEditor);
171
172 if (--countPresentationEditors == 0)
173 {
174     dbf_dropIndex(idxPresentationEditors);
175     idxPresentationEditors = 0;
176     dbf_dropTable(tblPresentationEditors);
177     tblPresentationEditors = 0;
178 }
179 }
```

## Member Function Documentation

**bool DakitPresentationByClassEditor::displayEditor ()**

Display this editor on screen

57 )

bool DakitPresentationByClassEditor::descriptionChange ()

Update changed presentation description

Returns:

true if successful, false if not

```
Definition at line 243 of file DakitPresentationByClassEditor.cpp:245 {
246 WebViewerID presentationID;
247 char description[256];
248 if (!getSelectedPresentation(presentationID))
249 return false;
250
251 dad_setcurrentdialog(m_editorName);
252 dad_setfieldvalue("description", fdescription);
253 if (!m_xmlPresentationByClass.updatePresentation(presentationID, description))
254 return false;
255
256 initPresentationList();
257 showPresentation(presentationID);
258 setTitle();
259 return true;
260 }
261 }
```

bool DakitPresentationByClassEditor::presentationSelected ()

Fill editor with values from selected presentation

Returns:

true if successful, false if not

```
Definition at line 268 of file DakitPresentationByClassEditor.cpp:270 {
271 WebViewerID presentationID;
272 char description[256];
273 bool fBuildIn = true;
274
275 if (!getSelectedPresentation(presentationID))
276 sprintf(description, "");
277 else if (!m_xmlPresentationByClass.fetchPresentation(presentationID, description,
fBuildIn))
278 sprintf(description, "**PRESENTATION NOT FOUND**");
279
280 if (dad_getdialogstate(m_editorName) != DAK_STATE_DISPLAYED)
281 return false;
282 dad_setcurrentdialog(m_editorName);
283 dad_setfieldvalue("description", description);
284
285 int sensitivity = 1;
286 if (fBuildIn)
287 sensitivity = 0;
288 dad_setfieldattribute("description", DAK_FIELD_SENSITIVITY, &sensitivity);
289 dad_setfieldattribute("set", DAK_FIELD_SENSITIVITY, &sensitivity);
290
291 initClassesTable();
292 return true;
293 }
```

bool DakitPresentationByClassEditor::showMaterial ()

Call user defined showMaterial callback

Returns:

true if successful, false if not

```
Definition at line 300 of file DakitPresentationByClassEditor.cpp:302 {
303 WebViewerID materialID;
304 getSelectedMaterial(materialID);
305
306 if (!m_pShowMaterialCB)
307 return false;
308
309 return (*m_pShowMaterialCB)(m_pShowMaterialContext, materialID);
}
```

Returns:

true if successful, false if not

```
Definition at line 189 of file DakitPresentationByClassEditor.cpp:191 {
192 if (dad_getdialogstate(m_editorName) != DAK_STATE_DISPLAYED)
193 {
194 if (dao_findobject(DAK_ROOT, DAO_DIALOG, "PresentationEditor", DAK_FIRST) ==
DAK_NULL)
195 if (dad_readembeddeddetail("dakitpresentationeditor", NULL) != DAK_ERROR_OK)
196 return false;
197
198 dad_displaydialog(m_editorName, "PresentationEditor", NULL);
199
200 initPresentationList();
201 initMaterialsList();
202 setTitle();
203 dad_raiseframe(m_editorName);
204 dad_geiconifyframe(m_editorName);
205 return true;
206 }
```

bool DakitPresentationByClassEditor::closeEditor (bool fDeleteDialog)

Close this editor

Returns:

true if successful, false if not

Parameters:

fDeleteDialog true if dakit dialog needs to be closed

```
Definition at line 213 of file DakitPresentationByClassEditor.cpp:215 {
216 if (dad_getdialogstate(m_editorName) != DAK_STATE_DISPLAYED)
217 return false;
218
219 if (fDeleteDialog)
220 dad_deletediialog(m_editorName);
221 return true;
222 }
```

bool DakitPresentationByClassEditor::setShowMaterialCB (void \* pContext, bool f)(void \*pContext, const WebViewerID &materialID) showMaterialCB)

Set callback that is called when show material button is pressed

Returns:

true if successful, false if not

```
Definition at line 229 of file DakitPresentationByClassEditor.cpp:232 {
233 m_pShowMaterialContext = pContext;
234 m_pShowMaterialCB = showMaterialCB;
235 return true;
236 }
```

DakitPresentationByClassEditor \* DakitPresentationByClassEditor::findEditor (char \* pEditorName) [static]

Find editor from name

Returns:

pointer to DakitPresentationByClassEditor class if found, 0 otherwise

```
Definition at line 46 of file DakitPresentationByClassEditor.cpp:48 {
49 PresentationEditor_t presentationEditor;
50
51 presentationEditor.name[sizeof (presentationEditor.name) - 1] = 0;
52 strcpy(presentationEditor.name, pEditorName, sizeof (presentationEditor.name) - 1);
53 if (!idxPresentationEditors || !
54 dbr_select(idxPresentationEditors, &presentationEditor) != DBF_ERROR_OK)
55 return 0;
56 return presentationEditor.pEditor;
}
```

```

310 }

bool DakitPresentationByClassEditor::setMaterial ()
Set selected material to selected presentation with selected class

Returns:
true if successful, false if not

Definition at line 317 of file DakitPresentationByClassEditor.cpp:319 {
320 WebViewerID presentationID, materialID;
321 PresentationByClassType_t classType;
322 PresentationByClassState_t classState;
323 char description[256];
324 bool fBuildIn = true;
325
326 if (!getSelectedPresentation(presentationID) ||
327 !getSelectedMaterial(materialID) ||
328 !getSelectedClass(classType, classState))
329 return false;
330
331 m_xmlPresentationByClass.fetchPresentation(presentationID, description, fBuildIn);
332 if (fBuildIn)
333 return false;
334
335 if (!m_xmlPresentationByClass.setMaterial(presentationID, classType, classState,
336 materialID))
337 return false;
338 return true;
339 }

bool DakitPresentationByClassEditor::revert ()
Revert file

Returns:
true if successful, false if not

Definition at line 345 of file DakitPresentationByClassEditor.cpp:347 {
348 bool revert;
349
350 m_updateEditor = false;
351 revert = m_xmlPresentationByClass.revert();
352 m_updateEditor = true;
353 initPresentationList();
354 setTitle();
355 return revert;
356 }

bool DakitPresentationByClassEditor::save ()
Save file

Returns:
true if successful, false if not

Definition at line 363 of file DakitPresentationByClassEditor.cpp:365 {
366 bool save;
367
368 save = m_xmlPresentationByClass.save();
369 setTitle();
370 return save;
371 }

bool DakitPresentationByClassEditor::export (char * filename)
Export presentations to file

Returns:
true if successful, false if not

Definition at line 378 of file DakitPresentationByClassEditor.cpp:380 {
381 bool export;

```

```

382 export = m_xmlPresentationByClass.exportFile(filename);
383 return export;
384 }

bool DakitPresentationByClassEditor::createPresentation ()
Create new presentation into editor

Returns:
true if successful, false if not

Definition at line 392 of file DakitPresentationByClassEditor.cpp:394 {
395 WebViewerID presentationID = WebViewerID::createNew(), defaultMaterialID;
396 char description[256];
397 static int idCounter = 0;
398 sprintf(description, "NEW PRESENTATION %d", idCounter++);
399 if (!m_xmlPresentationByClass.createPresentation(presentationID, description, false,
400 defaultMaterialID))
401 return false;
402 showPresentation(presentationID);
403 return true;
404 }

bool DakitPresentationByClassEditor::deletePresentation ()
Delete presentation from editor

Returns:
true if successful, false if not

Definition at line 413 of file DakitPresentationByClassEditor.cpp:415 {
416 WebViewerID presentationID;
417
418 if (!getSelectedPresentation(presentationID))
419 return false;
420
421 return m_xmlPresentationByClass.deletePresentation(presentationID);
422 }

void DakitPresentationByClassEditor::onInsert (const XmlPresentationByClass & presentation, const
WebViewerID & presentationID) [protected]

Called when new presentation is inserted from outside of the editor

Definition at line 428 of file DakitPresentationByClassEditor.cpp:431 {
432 initPresentationList();
433 setTitle();
434 return;
435 }

void DakitPresentationByClassEditor::onDelete (const XmlPresentationByClass & presentation,
const WebViewerID & presentationID) [protected]

Called when material is deleted from outside of the editor

Definition at line 457 of file DakitPresentationByClassEditor.cpp:460 {
461 initPresentationList();
462 setTitle();
463 return;
464 }

void DakitPresentationByClassEditor::onUpdate (const XmlPresentationByClass & presentation,
const WebViewerID & presentationID, PresentationByClassType_t classType,
PresentationByClassState_t state, const WebViewerID & materialID) [protected]

Called when presentation is updated from outside of the editor

```

```

Definition at line 441 of file DakitPresentationByClassEditor.cpp:447 {
448 updateClassesTable(className, state);
449 setTitle();
450 return;
451 }

```

**void DakitPresentationByClassEditor::onInsert (const XmlMaterials & materials, const WebViewID & materialID, const MaterialDef\_t & material) [protected]**

Called when new material is inserted from outside of the editor

```

Definition at line 470 of file DakitPresentationByClassEditor.cpp:474 {
475 initMaterialsList();
476 return;
477 }

```

**void DakitPresentationByClassEditor::onUpdate (const XmlMaterials & materials, const WebViewID & materialID, const MaterialDef\_t & material) [protected]**

Called when material is updated from outside of the editor

```

Definition at line 483 of file DakitPresentationByClassEditor.cpp:487 {
488 if (updateMaterialsList(materialID)) // returns true if description changed
489 initClassesTable();
490 return;
491 }

```

**void DakitPresentationByClassEditor::onDelete (const XmlMaterials & materials, const WebViewID & materialID, const MaterialDef\_t & material) [protected]**

Called when material is deleted from outside of the editor

```

Definition at line 497 of file DakitPresentationByClassEditor.cpp:501 {
502 initMaterialsList();
503 initClassesTable();
504 return;
505 }

```

**bool DakitPresentationByClassEditor::initPresentationList () [private]**

Init presentations list in editor

**Returns:**  
true if successful, false if not

```

Definition at line 512 of file DakitPresentationByClassEditor.cpp:514 {
515 WebViewID presentationID;
516 char description[256];
517 bool fBuildIn;
518 if (!m_fUpdateEditor)
519 return false;
520 if (dad_getDialogState(m_editorName) != DAK_STATE_DISPLAYED)
521 return false;
522 dad_setCurrentDialog(m_editorName);
523 dad_tableDeleteRow("presentations", DAK_POSITION_USEPOS, DAK_POSITION_ALL, NULL);
524 XmlPresentationByClassItemIterator itemIterator =
m_xmlPresentationByClass.createItemIterator();
525 for (int iPresentation = 1; itemIterator.getNext(presentationID); iPresentation++)
526 {
527 m_xmlPresentationByClass.fetchPresentation(presentationID, description, fBuildIn);
528 dad_tableInsertRow("presentations", iPresentation, DAK_POSITION_LAST, description);
529 presentationSelected();
530 return true;
531 }
532 }
533 }
534 }
535 }

```

**bool DakitPresentationByClassEditor::initClassesTable () [private]**

Init classes table in editor

**Returns:**

true if successful, false if not

```

Definition at line 683 of file DakitPresentationByClassEditor.cpp:685 {
686 WebViewID presentationID, materialID;
687 MaterialDef_t materialDef;
688 if (dad_getDialogState(m_editorName) != DAK_STATE_DISPLAYED)
689 return false;
690 dad_setCurrentDialog(m_editorName);
691 dad_tableDeleteRow("classes", DAK_POSITION_USEPOS, DAK_POSITION_ALL, NULL);
692 if (!getSelectedPresentation(presentationID))
693 return false;
694 PresentationByClassType_t classType;
695 PresentationByClassState_t classState;
696 dakitPresentationEditor_t classMaterial;
697 memset(classMaterial, 0, sizeof(classMaterial));
698 for (unsigned int iClass = 0; iClass < XMLPRESENTATIONBYCLASS_NUMCLASSES; iClass++)
699 {
700 if (unsigned int iState = 0; iState < XMLPRESENTATIONBYCLASS_NUMSTATES; iState++)
701 {
702 classType = m_xmlPresentationByClass.indexToClass(iClass);
703 classState = m_xmlPresentationByClass.indexToState(iState);
704 materialID = m_xmlPresentationByClass.getMaterial(presentationID, classType,
classState);
705 if (!m_xmlMaterials.fetchMaterial(materialID, materialDef))
706 sprintf(materialDef.description, "MATERIAL NOT FOUND");
707 classMaterial.classID = intFromClass(classType);
708 strcpy(classMaterial.state, stringFromState(classState));
709 strcpy(classMaterial.material, materialDef.description, sizeof
(classMaterial.material));
710 int rowId = XMLPRESENTATIONBYCLASS_NUMCLASSES * iState + iClass;
711 dad_tableInsertRow("classes", rowId, DAK_POSITION_LAST, &classMaterial);
712 }
713 }
714 return true;
715 }
716 }
717 }

```

**bool DakitPresentationByClassEditor::initMaterialsList () [private]**

Init materials list in editor

**Returns:**

true if successful, false if not

```

Definition at line 571 of file DakitPresentationByClassEditor.cpp:573 {
574 WebViewID materialID;
575 MaterialDef_t materialDef;
576 if (dad_getDialogState(m_editorName) != DAK_STATE_DISPLAYED)
577 return false;
578 dad_setCurrentDialog(m_editorName);
579 dad_tableDeleteRow("materials", DAK_POSITION_USEPOS, DAK_POSITION_ALL, NULL);
580 XmlMaterialItemIterator materialIterator = m_xmlMaterials.createItemIterator();
581 for (int iMaterial = 1; materialIterator.getNext(materialID); iMaterial++)
582 {
583 m_xmlMaterials.fetchMaterial(materialID, materialDef);
584 dad_tableInsertRow("materials", iMaterial, DAK_POSITION_LAST,
materialDef.description);
585 }
586 char *filename = "";
587 int size = 0;
588 if (m_xmlMaterials.getFilename(0, size))
589 {
590 filename = (char *) malloc(size);
591 m_xmlMaterials.getFilename(filename, size);
592 dad_setFieldValue("filename", filename);
593 if (!filename)
594 free(filename);
595 int sensitivity = 0;
596 dad_setFieldAttribute("filename", DAK_FIELD_SENSITIVITY, &sensitivity);
597 return true;
598 }
599 }
600 }
601 }
602 }

```

```
bool DakitPresentationByClassEditor::updateClassesTable (PresentationByClassType_t classType,
PresentationByClassState_t classState) [private]
```

Update classes table in editor

Returns:

true if successful, false if not

```
Definition at line 724 of file DakitPresentationByClassEditor.cpp:727 (
728 dakitPresentationEditor t_classMaterial;
729 WebViewerID presentationID, materialID;
730 MaterialDef_t materialDef;
731
732 if (dad_getdialogstate(m_editorName) != DAK_STATE_DISPLAYED)
733     return false;
734 dad_setcurrentdialog(m_editorName);
735 if (!getSelectedPresentation(presentationID))
736     return false;
737
738 materialID = m_xmlPresentationByClass.getMaterial(presentationID, classType,
classState);
739 if (!m_xmlMaterials.fetchMaterial(materialID, materialDef))
740     sprintf(materialDef.description, "MATERIAL NOT FOUND");
741 classMaterial.classID = intFromClass(classType);
742 strcpy(classMaterial.state, stringFromState(classState));
743 strcpy(classMaterial.material, materialDef.description, sizeof (classMaterial.material)
- 1);
744 unsigned int iClass = m_xmlPresentationByClass.classToIndex(classType);
745 unsigned int iState = m_xmlPresentationByClass.stateToIndex(classState);
746 int rowid = XMLPRESENTATIONBYCLASS_NUMCLASSES * iState + iClass; // rowid set in
initClassesTable()
747 if (dad_tablesetrow("Classes", rowid, DAK_POSITION_USEID, &classMaterial) !=
DAK_ERROR_OK)
748     return false;
749
750 return true;
751 )
```

```
bool DakitPresentationByClassEditor::updateMaterialsList (const WebViewerID & updateMaterialID)
[private]
```

Update one material in materials list

Returns:

true if successful, false if not

```
Definition at line 609 of file DakitPresentationByClassEditor.cpp:611 (
612 WebViewerID materialID;
613 MaterialDef_t materialDef;
614
615 if (dad_getdialogstate(m_editorName) != DAK_STATE_DISPLAYED)
616     return false;
617
618 XmlMaterialsItemIterator materialIterator = m_xmlMaterials.createItemIterator();
619 for (int iMaterial = 1; materialIterator.getNext(materialID); iMaterial++)
620 {
621     if (updateMaterialID == materialID)
622     {
623         m_xmlMaterials.fetchMaterial(materialID, materialDef);
624         int id = iMaterial; // id set in initMaterialsList()
625         int position = DAK_POSITION_USEID;
626         char *description = "";
627         dad_setcurrentdialog(m_editorName);
628         dad_tablegetrow("materials", &id, &position, &description);
629         if (strcmp(description, materialDef.description) != 0)
630         {
631             dad_tablesetrow("materials", id, position, materialDef.description);
632
633             // setrow seems to remove selection, must select again
634             id = DAK_POSITION_USEPOS;
635             position = DAK_POSITION_FIRST;
636             if (dad_tablegetrowselected("materials", &id, &position, &description))
637             {
638                 dad_tablesetrow("materials", id, position, NULL);
639                 dad_tablesetrowselected("materials", id, position, NULL, 1);
640             }
641         }
642     }
643 }
```

```
641         return true; // description changed
642     }
643     return false;
644 }
645 }
646 return false;
647 )
```

```
bool DakitPresentationByClassEditor::setTitle () [private]
```

Set title of the material editor

Returns:

true if successful, false if not

```
Definition at line 782 of file DakitPresentationByClassEditor.cpp:784 (
785 char editorTitle[512], changed[16] = "";
786 char *filename = "";
787 int size = 0;
788
789 if (!m_fUpdatedEditor)
790     return false;
791
792 if (m_xmlPresentationByClass.getFilename(0, size))
793 {
794     filename = (char *) malloc(size);
795     m_xmlPresentationByClass.getFilename(filename, size);
796 }
797 if (m_xmlPresentationByClass.isChanged())
798     sprintf(changed, "**");
799 sprintf(editorTitle, "XmlPresentationByClass: %s", changed, filename);
800 if (*filename)
801     free(filename);
802
803 dad_setframetitle(m_editorName, editorTitle);
804 return true;
805 )
```

```
bool DakitPresentationByClassEditor::showPresentation (const WebViewerID & presentationID)
[private]
```

Scroll presentations list and select given presentation

Returns:

true if successful, false if not

```
Definition at line 812 of file DakitPresentationByClassEditor.cpp:814 (
815 int id, position;
816
817 if (dad_getdialogstate(m_editorName) != DAK_STATE_DISPLAYED)
818     return false;
819
820 WebViewerID webViewerID;
821 XmlPresentationByClassItemIterator itemIterator =
m_xmlPresentationByClass.createItemIterator();
822 for (int iPresentation = 1; itemIterator.getNext(webViewerID); iPresentation++)
823     if (webViewerID == presentationID)
824     {
825         id = iPresentation; // simple solution, id set in initPresentationList()
826         position = DAK_POSITION_USEID;
827         dad_setcurrentdialog(m_editorName);
828         if (dad_tablesetrowselected("presentations", id, position, NULL, 1) !=
DAK_ERROR_OK)
829             return false;
830         dad_tablesetrow("presentations", id, position, NULL);
831         presentationSelected();
832         return true;
833     }
834     return false;
835 }
```

```
bool DakitPresentationByClassEditor::getSelectedPresentation (WebViewerID & presentationID)
[private]
```

Returns selected presentation in presentations list

**Returns:**  
true if successful, false if not

```
Definition at line 542 of file DakitPresentationByClassEditor.cpp.544 {
545 int id, position;
546 char *description;
547
548 if (dad_getdialogstate(m_editorName) != DAK_STATE_DISPLAYED)
549 return false;
550 dad_setcurrentdialog(m_editorName);
551 position = DAK_POSITION_FIRST;
552 if (!dad_tablegetrowselected("presentations", &id, &position, &description))
553 return false;
554
555 WebViewerID webViewerID;
556 XmlPresentationByClassItemIterator itemIterator =
557 m_xmlPresentationByClass.createItemIterator();
558 for (int iPresentation = 1; itemIterator.getNext(webViewerID); iPresentation++)
559 if (iPresentation == id) // simple solution, id set in initPresentationList()
560 {
561 presentationID = webViewerID;
562 return true;
563 }
564 return false;
565 }
```

**bool DakitPresentationByClassEditor::getSelectedMaterial (WebViewerID & materialID) [private]**  
Returns selected material in materials list

**Returns:**  
true if successful, false if not

```
Definition at line 654 of file DakitPresentationByClassEditor.cpp.656 {
657 int id, position;
658 char *description;
659
660 if (dad_getdialogstate(m_editorName) != DAK_STATE_DISPLAYED)
661 return false;
662 dad_setcurrentdialog(m_editorName);
663 position = DAK_POSITION_FIRST;
664 if (!dad_tablegetrowselected("materials", &id, &position, &description))
665 return false;
666
667 WebViewerID webViewerID;
668 XmlMaterialsItemIterator itemIterator = m_xmlMaterials.createItemIterator();
669 for (int iMaterial = 1; itemIterator.getNext(webViewerID); iMaterial++)
670 if (iMaterial == id) // simple solution, id set in initMaterialsList()
671 {
672 materialID = webViewerID;
673 return true;
674 }
675 return false;
676 }
```

**bool DakitPresentationByClassEditor::getSelectedClass (PresentationByClassType\_t & classType, PresentationByClassState\_t & classState) [private]**

Returns selected class in classes table

**Returns:**  
true if successful, false if not

```
Definition at line 758 of file DakitPresentationByClassEditor.cpp.761 {
762 dakitpresentationeditor_t classMaterial;
763 int id, position;
764
765 if (dad_getdialogstate(m_editorName) != DAK_STATE_DISPLAYED)
766 return false;
767 dad_setcurrentdialog(m_editorName);
768 position = DAK_POSITION_FIRST;
769 if (!dad_tablegetrowselected("classes", &id, &position, &classMaterial))
770 return false;
771 }
```

```
772 classType = classFromInt(classMaterial.classID);
773 classState = stateFromString(classMaterial.state);
774 return true;
775 }
```

**DakitPresentationByClassEditor DakitPresentationByClassEditor::operator= (const DakitPresentationByClassEditor & other) [private]**

**Member Data Documentation**

**XmlPresentationByClass & DakitPresentationByClassEditor::m\_xmlPresentationByClass [private]**  
the presentations in this editor are stored here

**Definition at line 62 of file DakitPresentationByClassEditor.h.XmlMaterials & DakitPresentationByClassEditor::m\_xmlMaterials [private]**  
the materials used in presentations are stored here

**Definition at line 63 of file DakitPresentationByClassEditor.h.void\* DakitPresentationByClassEditor::m\_pShowMaterialContext [private]**  
user defined parameter to use with m\_pShowMaterialCB

**Definition at line 64 of file DakitPresentationByClassEditor.h.bool(" DakitPresentationByClassEditor::m\_pShowMaterialCB)(void \*pContext, const WebViewerID &materialID) [private]**  
user defined callback

**char DakitPresentationByClassEditor::m\_editorName[256] [private]**  
the name of this editor

**Definition at line 68 of file DakitPresentationByClassEditor.h.bool DakitPresentationByClassEditor::m\_fUpdateEditor [private]**  
true if the values in the GUI should be updated when they are changed

Definition at line 69 of file DakitPresentationByClassEditor.h.

The documentation for this class was generated from the following files:

- **DakitPresentationByClassEditor.h**
- **DakitPresentationByClassEditor.cpp**

**dakitpresentationeditor\_t Struct Reference**

#include <dakitpresentationeditor\_t.h>

**Public Attributes**

- int classID
- char state [32]
- char material [256]

**Member Data Documentation**

int dakitpresentationeditor\_t::classID

Definition at line 6 of file dakitpresentationeditor\_t.h.char dakitpresentationeditor\_t::state[32]

Definition at line 7 of file dakitpresentationeditor\_t.h.char dakitpresentationeditor\_t::material[256]

Definition at line 8 of file dakitpresentationeditor\_t.h.

The documentation for this struct was generated from the following file:

- dakitpresentationeditor\_t.h

**lightingEditor\_t Struct Reference**

**Public Attributes**

- char name [256]
- dz\_world\_t worldID
- dz\_interactor\_t interactorID
- DakitLightingEditor \* pEditor

**Member Data Documentation**

char lightingEditor\_t::name[256]

Definition at line 21 of file DakitLightingEditor.cpp.dz\_world\_t lightingEditor\_t::worldID

Definition at line 22 of file DakitLightingEditor.cpp.dz\_interactor\_t lightingEditor\_t::interactorID

Definition at line 23 of file DakitLightingEditor.cpp.DakitLightingEditor\* lightingEditor\_t::pEditor

Definition at line 24 of file DakitLightingEditor.cpp.

The documentation for this struct was generated from the following file:

- DakitLightingEditor.cpp

materialEditor\_t Struct Reference

Public Attributes

- char name [256]
- DakitMaterialEditor \* pEditor

Member Data Documentation

char materialEditor\_t::name[256]

Definition at line 22 of file DakitMaterialEditor.cpp.DakitMaterialEditor\* materialEditor\_t::pEditor

Definition at line 23 of file DakitMaterialEditor.cpp.

The documentation for this struct was generated from the following file:

- DakitMaterialEditor.cpp

presentationEditor\_t Struct Reference

Public Attributes

- char name [256]
- DakitPresentationByClassEditor \* pEditor

Member Data Documentation

char presentationEditor\_t::name[256]

Definition at line 23 of file DakitPresentationByClassEditor.cpp.DakitPresentationByClassEditor\* presentationEditor\_t::pEditor

Definition at line 24 of file DakitPresentationByClassEditor.cpp.

The documentation for this struct was generated from the following file:

- DakitPresentationByClassEditor.cpp

DakitLightingEditor.cpp File Reference

```
#include <math.h>
#include "zkit.h"
#include "DakitLightingEditor.h"
#include "FunctionsLighting.h"
#include "dokit.h"
#include "dakkit.h"
#include "dbfast.h"
```

Classes

- struct **lightingEditor\_t**

Defines

- #define **PI** 3.141592653589793
- #define **MIN(a, b)** (a<b?a:b)
- #define **MAX(a, b)** (a>b?a:b)

Functions

- void **selectionCB** (void \*pUserContext, WebViewDatabase \*pWebViewDatabase, const WebViewID &webViewID)
- void **hsvToRgb** (float h, float s, float v, float \*pr, float \*pg, float \*pb)
- void **rgbToHsv** (float r, float g, float b, float \*ph, float \*ps, float \*pv)
- void **normalize** (double \*pX, double \*pY, double \*pZ)
- int **createBall** (dz\_world\_t world, dz\_layer\_t layer, int templateID, int material)

Variables

- dbf\_tablespec\_t **tblLightingEditorSpec** = { 10, 10 }
- dbf\_fieldspec\_t **fldLightingEditorSpec** []
- dbf\_indexespec\_t **idxLightingEditorsSpec** = { DBF\_INDEX\_HASH, 1 }
- int **idxLightingEditorsFields** [] = { 0, -1 }
- int **idxLightingEditorsFieldsById** [] = { 1, 2, -1 }
- dbf\_table\_t \* **tblLightingEditors** = 0
- dbf\_index\_t \* **idxLightingEditors** = 0
- dbf\_index\_t \* **idxLightingEditorsById** = 0
- int **countLightingEditors** = 0

Define Documentation

#define **PI** 3.141592653589793

Definition at line 14 of file **DakitLightingEditor.cpp**.#define **MIN(a, b)** (a<b?a:b)

Definition at line 274 of file **DakitLightingEditor.cpp**.#define **MAX(a, b)** (a>b?a:b)

Definition at line 275 of file **DakitLightingEditor.cpp**.

void **selectionCB** (void \* pUserContext, WebViewDatabase \* pWebViewDatabase, const WebViewID & webViewID) [static]

Callback called from the database editor when select button is pressed

Parameters:

*pUserContext* user defined context

*pWebViewDatabase* pointer to calling database

*webViewID* ID for selected lighting

Definition at line 86 of file **DakitLightingEditor.cpp**.90 ( 91 DakitLightingEditor \*pEditor = (DakitLightingEditor \*) pUserContext; 92 WebViewHeader\_t header; 93 WebViewLighting\_t lighting; 94 95 if (pWebViewDatabase->selectLighting(header, lighting, webViewID)) 96 pEditor->setLighting(header, lighting); 97 return; 98 )

void **hsvToRgb** (float h, float s, float v, float \* pr, float \* pg, float \* pb) [static]

Definition at line 279 of file **DakitLightingEditor.cpp**.282 ( 283 int i; 284 float aa, bb, cc, f; 285 286 if (s == 0) // greyscale 287 \*pr = \*pg = \*pb = v; 288 else 289 { 290 if (h == 1.0) 291 h = 0; 292 h \*= 6.0; 293 i = (int) floor(h); 294 f = h - i; 295 aa = v \* (1 - s); 296 bb = v \* (1 - (s \* f)); 297 cc = v \* (1 - (s \* (1 - f))); 298 switch (i) 299 { 300 case 0: \*pr = v; \*pg = cc; \*pb = aa; break; 301 case 1: \*pr = bb; \*pg = v; \*pb = aa; break; 302 case 2: \*pr = aa; \*pg = v; \*pb = cc; break; 303 case 3: \*pr = aa; \*pg = bb; \*pb = v; break; 304 case 4: \*pr = cc; \*pg = aa; \*pb = v; break; 305 case 5: \*pr = v; \*pg = aa; \*pb = bb; break; 306 } 307 return; 308 } 309 )

void **rgbToHsv** (float r, float g, float b, float \* ph, float \* ps, float \* pv) [static]

Definition at line 313 of file **DakitLightingEditor.cpp**.316 ( 317 float max = MAX(r,MAX(g,b)); 318 float min = MIN(r,MIN(g,b)); 319 float delta = max - min; 320 321 \*pv = max; 322 if (max != 0.0) 323 \*ps = delta / max; 324 else 325 \*ps = 0.0; 326 if (\*ps == 0.0) 327 \*ph = 0.0; // no hue

```

890 dzg_layerobjectsettemplatematerial(world, layer, templateID, material);
891 dzg_layerobjectaddtriangles(world, layer, templateID, c, pTriangles);
892 #else
893 dzg_layerobjectaddtriangles(world, layer, templateID, material, c, pTriangles);
894 #endif
895 free(pTriangles);
896
897 return 0;
898 }

```

## Variable Documentation

**dbf\_tablespec\_t** tblLightingEditorSpec = { 10, 10 } [static]

Definition at line 27 of file DakItLightingEditor.cpp.dbf\_fieldspec\_t fldLightingEditorSpec[] [static]

```

Initial value:
{
    DBF_FIELD_STRING,    256,
    DBF_FIELD_INT,       0,
    DBF_FIELD_INT,       0,
    DBF_FIELD_POINTER,   0,
    DBF_FIELD_END,       0
}

```

Definition at line 28 of file DakItLightingEditor.cpp.dbf\_indexspec\_t idxLightingEditorsSpec = { DBF\_INDEX\_HASH, 1 } [static]

Definition at line 37 of file DakItLightingEditor.cpp.int\_idxLightingEditorsFields[] = { 0, -1 } [static]

Definition at line 38 of file DakItLightingEditor.cpp.int\_idxLightingEditorsFieldsById[] = { 1, 2, -1 } [static]

Definition at line 39 of file DakItLightingEditor.cpp.dbf\_table\_t\* tblLightingEditors = 0 [static]

Definition at line 40 of file DakItLightingEditor.cpp.dbf\_index\_t\* idxLightingEditors = 0 [static]

Definition at line 41 of file DakItLightingEditor.cpp.dbf\_index\_t\* idxLightingEditorsById = 0 [static]

Definition at line 42 of file DakItLightingEditor.cpp.int countLightingEditors = 0 [static]

Definition at line 43 of file DakItLightingEditor.cpp.

```

328 else
329 {
330     if (r == max)
331         *ph = (g - b) / delta;
332     else if (g == max)
333         *ph = 2 + (b - r) / delta;
334     else // b == max
335         *ph = 4 + (r - g) / delta;
336     *ph *= 60.0;
337     if (*ph < 0.0)
338         *ph += 360.0;
339     *ph /= 360.0;
340     return;
341 }
342 }

```

**void normalize (double \*pX, double \*pY, double \*pZ) [static]**

```

Definition at line 656 of file DakItLightingEditor.cpp.658 {
659     double radius = sqrt(*pX * *pX + *pY * *pY + *pZ * *pZ);
660     *pX /= radius;
661     *pY /= radius;
662     *pZ /= radius;
663     return;
664 }

```

**int createBall (dz\_world\_t world, dz\_layer\_t layer, int templateID, int material) [static]**

```

Definition at line 840 of file DakItLightingEditor.cpp.842 {
843     dzg_layerobjecttriangle_t *pTriangles = 0;
844     double step;
845     float x1, x2, x3, x4, y1, y2, y3, y4, z1, z2, z3, z4;
846     int c = 0;
847
848     step = PI / 10;
849     for (double a = 0.0; a < PI; a += step)
850     {
851         y1 = y4 = (float) cos(a);
852         y2 = y3 = (float) cos(a+step);
853         for (double b = 0.0; b < PI; b += step, c++)
854         {
855             x1 = (float) sin(a) * cos(b);
856             z1 = (float) sin(a) * sin(b);
857             x4 = (float) sin(a) * cos(b+step);
858             z4 = (float) sin(a) * sin(b+step);
859             x3 = (float) sin(a+step) * cos(b);
860             z3 = (float) sin(a+step) * cos(b+step);
861             x2 = (float) sin(a+step) * sin(b);
862             z2 = (float) sin(a+step) * sin(b+step);
863             if (c % 20 == 0)
864             {
865                 pTriangles = (dzg_layerobjecttriangle_t *)
866                     realloc(pTriangles, (c+20) * sizeof(*pTriangles));
867             }
868             pTriangles[c].east1 = x1;
869             pTriangles[c].east2 = x2;
870             pTriangles[c].east3 = x3;
871             pTriangles[c].north1 = y1;
872             pTriangles[c].north2 = y2;
873             pTriangles[c].north3 = y3;
874             pTriangles[c].height1 = z1;
875             pTriangles[c].height2 = z2;
876             pTriangles[c].height3 = z3;
877             c++;
878             pTriangles[c].east1 = x1;
879             pTriangles[c].east2 = x4;
880             pTriangles[c].east3 = x2;
881             pTriangles[c].north1 = y1;
882             pTriangles[c].north2 = y4;
883             pTriangles[c].north3 = y2;
884             pTriangles[c].height1 = z1;
885             pTriangles[c].height2 = z4;
886             pTriangles[c].height3 = z2;
887         }
888     }
889     #if ZKIT_VERSION >= 131

```

## DakitLightingEditor.h File Reference

```
#include "WebViewerDatabase.h"
#include "DakitDatabaseEditor.h"
```

### Classes

- class **DakitLightingEditor**

## DakitLightingEditorCB.cpp File Reference

```
#include "dxkit.h"
#include "dakkit.h"
#include "zkit.h"
#include "DakitLightingEditor.h"
```

### Functions

- void **DakCB\_LightSelect** (char \*dialog, char \*field, char \*parameter)
- void **DakCB\_LightingChange** (char \*dialog, char \*field, char \*parameter)
- void **DakCB\_LightingButton** (char \*dialog, char \*field, int button, char \*parameter)
- void **DakCB\_LightingMenu** (char \*parameter)

### Detailed Description

Callback functions for **DakitLightingEditor**. These are called by dakit and defined in .ail

Definition in file **DakitLightingEditorCB.cpp**.

### Function Documentation

**void DakCB\_LightSelect (char \* dialog, char \*, char \*)**

Called when user selects a light in editor

**Parameters:**

*dialog* name of the editor

Definition at line 25 of file DakitLightingEditorCB.cpp:29 (

```
30 DakitLightingEditor *pEditor = DakitLightingEditor::findEditor(dialog);
31 if (pEditor)
32     pEditor->lightSelect();
33     return;
34 )
```

**void DakCB\_LightingChange (char \* dialog, char \*, char \* parameter)**

Called when user changes a field value

**Parameters:**

*dialog* name of the editor

*parameter* which field was changed

Definition at line 40 of file DakitLightingEditorCB.cpp:44 (

```
45 DakitLightingEditor *pEditor = DakitLightingEditor::findEditor(dialog);
46 if (pEditor)
47 {
48     if (strcmp(parameter, "color_hsv") == 0)
49         pEditor->colorChange(true);
50     if (strcmp(parameter, "color_rgb") == 0)
51         pEditor->colorChange(false);
52     if (strcmp(parameter, "type") == 0)
53         pEditor->typeChange();
54     if (strcmp(parameter, "preview") == 0)
55         pEditor->previewChange();
56 }
57 return;
58 )
```

**void DakCB\_LightingButton (char\* dialog, char\*, int, char\* parameter)**  
Called when user presses a button in editor

**Parameters:**  
*dialog* name of the editor

*parameter* which button was pressed

Definition at line 64 of file DakitLightingEditorCB.cpp.69 {  
70 DakitLightingEditor \*pEditor = DakitLightingEditor::findEditor(dialog);  
71 if (pEditor)  
72 {  
73 if (strcmp(parameter, "CREATE") == 0)  
74 pEditor->createLight();  
75 if (strcmp(parameter, "DELETE") == 0)  
76 pEditor->deleteLight();  
77 }  
78 return;  
79 }

**void DakCB\_LightingMenu (char\* parameter)**  
Called when user selects a menu entry

**Parameters:**  
*parameter* which menu entry was selected

Definition at line 85 of file DakitLightingEditorCB.cpp.87 {  
88 char \*dialog = daf\_getcurrentframe();  
89 DakitLightingEditor \*pEditor = DakitLightingEditor::findEditor(dialog);  
90 if (pEditor)  
91 {  
92 if (strcmp(parameter, "NEW") == 0)  
93 pEditor->newLighting();  
94 if (strcmp(parameter, "NEWFROMWINDOW") == 0)  
95 pEditor->newLightingFromWindow();  
96 if (strcmp(parameter, "IMPORT") == 0)  
97 pEditor->importLighting();  
98 if (strcmp(parameter, "EXPORT") == 0)  
99 pEditor->exportLighting();  
100 }  
101 return;  
102 }

## DakitMaterialEditor.cpp File Reference

```
#include <string.h>
#include <math.h>
#include "zkit.h"
#include "DakitMaterialEditor.h"
#include "WebViewerID.h"
#include "dxkit.h"
#include "dakkit.h"
#include "dakdynamic.h"
#include "dbfast.h"
```

## Classes

- struct **materialEditor\_t**

## Defines

- #define **MIN(a, b)** (a<b?a:b)
- #define **MAX(a, b)** (a>b?a:b)

## Functions

- void **hsvToRgb** (float h, float s, float v, float \*pr, float \*pg, float \*pb)
- void **rgbToHsv** (float r, float g, float b, float \*ph, float \*ps, float \*pv)

## Variables

- dbf\_tablespec\_t **tblMaterialEditorSpec** = { 10, 10 }
- dbf\_fieldspec\_t **tblMaterialEditorSpec** []
- dbf\_indexspec\_t **tblMaterialEditorsSpec** = { DBF\_INDEX\_HASH, 1 }
- int **tblMaterialEditorsFields** [] = { 0, -1 }
- dbf\_table\_t \* **tblMaterialEditors** = 0
- dbf\_index\_t \* **tblMaterialEditors** = 0
- int **countMaterialEditors** = 0
- char \* **da\_aildump\_dakitmaterialeditor** []
- char \*\* **tmp** = **da\_aildump\_dakitmaterialeditor**

## Define Documentation

**#define MIN(a, b)** (a<b?a:b)

Definition at line 227 of file DakitMaterialEditor.cpp.**#define MAX(a, b)** (a>b?a:b)

Definition at line 228 of file DakitMaterialEditor.cpp.

## Function Documentation

**void hsvToRgb** (float h, float s, float v, float \* pr, float \* pg, float \* pb) [static]

Definition at line 232 of file DakitMaterialEditor.cpp.235 {  
236 int i;  
237 float aa, bb, cc, f;  
238  
239 if (s == 0) // greyscale  
240 \*pr = \*pg = \*pb = v;  
241 else  
242 {  
243 if (h == 1.0)

```
244     h = 0;
245     h *= 6.0;
246     i = (int) floor(h);
247     f = h - i;
248     aa = v * (1 - s);
249     bb = v * (1 - (s * f));
250     cc = v * (1 - (s * f));
251     switch (i)
252     {
253     case 0: *pr = v; *pg = cc; *pb = aa; break;
254     case 1: *pr = bb; *pg = v; *pb = aa; break;
255     case 2: *pr = aa; *pg = v; *pb = cc; break;
256     case 3: *pr = aa; *pg = bb; *pb = v; break;
257     case 4: *pr = cc; *pg = aa; *pb = v; break;
258     case 5: *pr = v; *pg = aa; *pb = bb; break;
259     }
260     }
261     return;
262 }
```

void rgbToHsv (float r, float g, float b, float \* ph, float \* ps, float \* pv) [static]

```
Definition at line 266 of file DakitMaterialEditor.cpp.269 {
270     float max = MAX(r,MAX(g,b));
271     float min = MIN(r,MIN(g,b));
272     float delta = max - min;
273
274     *pv = max;
275     if (max != 0.0)
276     {
277         *ps = delta / max;
278     }
279     else
280     {
281         *ps = 0.0;
282     }
283     if (*ps == 0.0)
284     {
285         *ph = 0.0; // no hue
286     }
287     else
288     {
289         if (r == max)
290         {
291             *ph = (g - b) / delta;
292         }
293         else if (g == max)
294         {
295             *ph = 2 + (b - r) / delta;
296         }
297         else // b == max
298         {
299             *ph = 4 + (r - g) / delta;
300         }
301         *ph *= 60.0;
302         if (*ph < 0.0)
303             *ph += 360.0;
304         *ph /= 360.0;
305     }
306     return;
307 }
```

Variable Documentation

dbf\_tablespec\_t tblMaterialEditorSpec = { 10, 10 } [static]

```
Definition at line 26 of file DakitMaterialEditor.cpp.dbf_fieldspec_t fldMaterialEditorSpec[] [static]
Initial value:
(
    DBF_FIELD_STRING,      256,
    DBF_FIELD_POINTER,     0,
    DBF_FIELD_END,         0
)
```

Definition at line 27 of file DakitMaterialEditor.cpp.dbf\_indexespec\_t idxMaterialEditorsSpec = { DBF\_INDEX\_HASH, 1 } [static]

Definition at line 34 of file DakitMaterialEditor.cpp.int\_idxMaterialEditorsFields[] = { 0, -1 } [static]

Definition at line 35 of file DakitMaterialEditor.cpp.dbf\_table\_t\* tblMaterialEditors = 0 [static]

Definition at line 36 of file DakitMaterialEditor.cpp.dbf\_index\_t\* idxMaterialEditors = 0 [static]

Definition at line 37 of file DakitMaterialEditor.cpp.int countMaterialEditors = 0 [static]

Definition at line 38 of file DakitMaterialEditor.cpp.char\* da\_alldump\_dakitmaterialeditor[] from dakbind.c

Definition at line 131 of file DakitMaterialEditor.cpp.char\* tmp = da\_alldump\_dakitmaterialeditor [static]

used for das\_readembeddedail to work

Definition at line 132 of file DakitMaterialEditor.cpp.

## DakitMaterialEditor.h File Reference

```
#include "XmlMaterials.h"
```

### Classes

- class **DakitMaterialEditor**

## DakitMaterialEditorCB.cpp File Reference

```
#include <string.h>
#include "dxkit.h"
#include "dakkit.h"
#include "zkit.h"
#include "DakitMaterialEditor.h"
```

### Functions

- void **DakCB\_MaterialField** (char \*dialog, char \*field, char \*parameter)
- void **DakCB\_MaterialMenu** (char \*parameter)
- void **exportFileCB** (char \*pFilename)

### Variables

- **DakitMaterialEditor** \* pEditorStatic = 0

### Detailed Description

Callback functions for **DakitMaterialEditor**. These are called by dakit and defined in all

Definition in file **DakitMaterialEditorCB.cpp**.

### Function Documentation

**void DakCB\_MaterialField** (char \* *dialog*, char \*, char \* *parameter*)

Called when user changes a field value

Parameters:

*dialog* name of the editor

*parameter* which field was changed

Definition at line 25 of file DakitMaterialEditorCB.cpp.29 {  
30 DakitMaterialEditor \*pEditor = DakitMaterialEditor::findEditor(dialog);  
31 if (pEditor)  
32 {  
33 if (strcmp(parameter, "listselect") == 0)  
34 pEditor->materialSelected();  
35 if (strcmp(parameter, "description") == 0)  
36 pEditor->descriptionChange();  
37 if (strcmp(parameter, "color\_hsv") == 0)  
38 pEditor->colorChange(true);  
39 if (strcmp(parameter, "color\_rgb") == 0)  
40 pEditor->colorChange(false);  
41 if (strcmp(parameter, "color\_hsv\_edge") == 0)  
42 pEditor->edgeColorChange(true);  
43 if (strcmp(parameter, "color\_rgb\_edge") == 0)  
44 pEditor->edgeColorChange(false);  
45 if (strcmp(parameter, "shininess") == 0)  
46 pEditor->shininessChange();  
47 if (strcmp(parameter, "linewidth") == 0)  
48 pEditor->linewidthChange();  
49 if (strcmp(parameter, "lighting") == 0)  
50 pEditor->effectChange();  
51 if (strcmp(parameter, "fog") == 0)  
52 pEditor->effectChange();  
53 if (strcmp(parameter, "frontbuffer") == 0)  
54 pEditor->effectChange();  
55 if (strcmp(parameter, "dynamicclipping") == 0)  
56 pEditor->effectChange();  
57 if (strcmp(parameter, "forcevisibility") == 0)  
58 pEditor->effectChange();

```

59 if (strcmp(parameter, "lighting_edge") == 0)
60     pEditor->edgeEffectChange();
61 if (strcmp(parameter, "fog_edge") == 0)
62     pEditor->edgeEffectChange();
63 if (strcmp(parameter, "frontbuffer_edge") == 0)
64     pEditor->edgeEffectChange();
65 if (strcmp(parameter, "forcedraw_edge") == 0)
66     pEditor->edgeEffectChange();
67 }
68 return;
69 }

```

## void DakCB\_MaterialMenu (char \* parameter)

Called when user selects a menu entry

### Parameters:

*parameter* what menu entry was selected

```

Definition at line 98 of file DakitMaterialEditorCB.cpp:100 (
101 char *dialog = daf.GetCurrentFrame();
102 DakitMaterialEditor *pEditor = DakitMaterialEditor::findEditor(dialog);
103 if (pEditor)
104 {
105     if (strcmp(parameter, "REVERT") == 0)
106         pEditor->revert();
107     if (strcmp(parameter, "SAVE") == 0)
108         pEditor->save();
109     if (strcmp(parameter, "EXPORT") == 0)
110     {
111         pEditorStatic = pEditor;
112         dad_setFileselectionParameters("", "XmlMaterials (*.xml)|*.xml||", "No help what
so ever.");
113         dad_displayFileselectionbox("Export", exportFileCB);
114     }
115     if (strcmp(parameter, "CREATE") == 0)
116         pEditor->createMaterial();
117     if (strcmp(parameter, "DELETE") == 0)
118         pEditor->deleteMaterial();
119 }
120 return;
121 )

```

## void exportFileCB (char \* pFilename) [static]

Callback for exporting materials to file

```

Definition at line 77 of file DakitMaterialEditorCB.cpp:79 (
80 char filename[512];
81
82 if (pEditorStatic)
83 {
84     if (istrstr(pFilename, ".xml"))
85         sprintf(filename, "%s.xml", pFilename);
86     else
87         sprintf(filename, "%s", pFilename);
88     pEditorStatic->export(filename);
89 }
90 pEditorStatic = 0;
91 return;
92 )

```

## Variable Documentation

DakitMaterialEditor\* pEditorStatic = 0 [static]

Definition at line 71 of file DakitMaterialEditorCB.cpp.

## DakitPresentationByClassEditor.cpp File Reference

```

#include <string.h>
#include <math.h>
#include "zkit.h"
#include "DakitPresentationByClassesEditor.h"
#include "WebViewerID.h"
#include "dakitpresentationeditor_t.h"
#include "dxkit.h"
#include "dakkit.h"
#include "dakdynamic.h"
#include "dbfast.h"

```

## Classes

- struct presentationEditor\_t

## Functions

- PresentationByClassType\_t classFromInt (int classType)
- int intFromClass (PresentationByClassType\_t classType)
- PresentationByClassState\_t stateFromString (char \*string)
- char \* stringFromState (PresentationByClassState\_t state)

## Variables

- dbf\_tablespec\_t tblPresentationEditorSpec = { 10, 10 }
- dbf\_fieldspec\_t fldPresentationEditorSpec []
- dbf\_indexspec\_t idxPresentationEditorsSpec = { DBF\_INDEX\_HASH, 1 }
- int idxPresentationEditorsFields [] = { 0, -1 }
- dbf\_table\_t \* tblPresentationEditors = 0
- dbf\_index\_t \* idxPresentationEditors = 0
- int countPresentationEditors = 0
- char \* unselected = "unselected"
- char \* selected = "selected"
- char \* highlighted = "highlighted"
- char \* da\_aildump\_dakitpresentationeditor []
- char \*\* tmp = da\_aildump\_dakitpresentationeditor

## Function Documentation

PresentationByClassType\_t classFromInt (int classType) [static]

```

Definition at line 63 of file DakitPresentationByClassEditor.cpp:65 (
66 if (classType == 1) return CLASS_1;
67 if (classType == 2) return CLASS_2;
68 if (classType == 3) return CLASS_3;
69 if (classType == 4) return CLASS_4;
70 if (classType == 5) return CLASS_5;
71 if (classType == 6) return CLASS_6;
72 if (classType == 7) return CLASS_7;
73 if (classType == 8) return CLASS_8;
74 if (classType == 9) return CLASS_9;
75 if (classType == 10) return CLASS_10;
76 if (classType == 11) return CLASS_11;
77 if (classType == 12) return CLASS_12;
78 if (classType == 13) return CLASS_13;
79 if (classType == 14) return CLASS_14;
80 if (classType == 999) return CLASS_999;
81 return CLASS_1;
82 )

```

int intFromClass (PresentationByClassType\_t classType) [static]

```
Definition at line 86 of file DakitPresentationByClassEditor.cpp:88 (
89     if (classType == CLASS_1) return 1;
90     if (classType == CLASS_2) return 2;
91     if (classType == CLASS_3) return 3;
92     if (classType == CLASS_4) return 4;
93     if (classType == CLASS_5) return 5;
94     if (classType == CLASS_6) return 6;
95     if (classType == CLASS_7) return 7;
96     if (classType == CLASS_8) return 8;
97     if (classType == CLASS_9) return 9;
98     if (classType == CLASS_10) return 10;
99     if (classType == CLASS_11) return 11;
100     if (classType == CLASS_12) return 12;
101     if (classType == CLASS_13) return 13;
102     if (classType == CLASS_14) return 14;
103     if (classType == CLASS_999) return 999;
104     return 1;
105 )
```

PresentationByClassState\_t stateFromString (char \* string) [static]

```
Definition at line 113 of file DakitPresentationByClassEditor.cpp:115 (
116     if (strcmp(string, unselected) == 0) return STATE_UNSELECTED;
117     if (strcmp(string, selected) == 0) return STATE_SELECTED;
118     if (strcmp(string, highlighted) == 0) return STATE_HIGHLIGHTED;
119     return STATE_UNSELECTED;
120 )
```

char\* stringFromState (PresentationByClassState\_t state) [static]

```
Definition at line 124 of file DakitPresentationByClassEditor.cpp:126 (
127     if (state == STATE_UNSELECTED) return unselected;
128     if (state == STATE_SELECTED) return selected;
129     if (state == STATE_HIGHLIGHTED) return highlighted;
130     return unselected;
131 )
```

## Variable Documentation

dbf\_tablespec\_t tblPresentationEditorSpec = { 10, 10 } [static]

Definition at line 27 of file DakitPresentationByClassEditor.cpp:dbf\_fieldspec\_t fldPresentationEditorSpec[] [static]

```
Initial value:
(
    DBF_FIELD_STRING,    256,
    DBF_FIELD_POINTER,   0,
    DBF_FIELD_END,       0
)
```

Definition at line 28 of file DakitPresentationByClassEditor.cpp:dbf\_indexspec\_t idxPresentationEditorsSpec = { DBF\_INDEX\_HASH, 1 } [static]

Definition at line 35 of file DakitPresentationByClassEditor.cpp:int\_idxPresentationEditorsFields[] = { 0, -1 } [static]

Definition at line 36 of file DakitPresentationByClassEditor.cpp:dbf\_table\_t\* tblPresentationEditors = 0 [static]

Definition at line 37 of file DakitPresentationByClassEditor.cpp:dbf\_index\_t\* idxPresentationEditors = 0 [static]

Definition at line 38 of file DakitPresentationByClassEditor.cpp:int countPresentationEditors = 0 [static]

Definition at line 39 of file DakitPresentationByClassEditor.cpp:char\* unselected = "unselected" [static]

Definition at line 107 of file DakitPresentationByClassEditor.cpp:char\* selected = "selected" [static]

Definition at line 108 of file DakitPresentationByClassEditor.cpp:char\* highlighted = "highlighted" [static]

Definition at line 109 of file DakitPresentationByClassEditor.cpp:char\* da\_alldump\_dakitpresentationeditor[] from dakbind.c

Definition at line 181 of file DakitPresentationByClassEditor.cpp:char\*\* tmp = da\_alldump\_dakitpresentationeditor [static] used for das\_readembeddedail to work

Definition at line 182 of file DakitPresentationByClassEditor.cpp.

DakitPresentationByClassEditor.h File Reference

```
#include "XmlPresentation.h"
#include "XmlMaterials.h"
```

Classes

- class DakitPresentationByClassEditor

DakitPresentationByClassEditorCB.cpp File Reference

```
#include "dxkit.h"
#include "dakit.h"
#include "zkit.h"
#include "DakitPresentationByClassEditor.h"
```

Functions

- void DakCB\_PresentationByClassField (char \*dialog, char \*field, char \*parameter)
- void DakCB\_PresentationByClassButton (char \*dialog, char \*field, int button, char \*parameter)
- void DakCB\_PresentationByClassMenu (char \*parameter)
- void exportFileCB (char \*pFilename)

Variables

- DakitPresentationByClassEditor \* pEditorStatic = 0

Detailed Description

Callback functions for DakitPresentationByClassEditor. These are called by dakit and defined in .ail

Definition in file DakitPresentationByClassEditorCB.cpp.

Function Documentation

void DakCB\_PresentationByClassField (char \* dialog, char \*, char \* parameter)

Called when user changes a field value

Parameters:

*dialog* name of the editor

*parameter* which field was changed

```
Definition at line 24 of file DakitPresentationByClassEditorCB.cpp:28 (
29  DakitPresentationByClassEditor *pEditor =
DakitPresentationByClassEditor::findEditor(dialog);
30  if (pEditor)
31  {
32      if (strcmp(parameter, "DESCRIPTION") == 0)
33          pEditor->descriptionChange();
34      if (strcmp(parameter, "PRESENTATION") == 0)
35          pEditor->presentationSelected();
36      }
37      return;
38 )
```

void DakCB\_PresentationByClassButton (char \* dialog, char \*, int, char \* parameter)

Called when user presses a button in editor

Parameters:

*dialog* name of the editor

*parameter* which button was pressed

```
Definition at line 44 of file DakitPresentationByClassEditorCB.cpp:49 (
50  DakitPresentationByClassEditor *pEditor =
DakitPresentationByClassEditor::findEditor(dialog);
```

# Index

~DakitLightingEditor  
DakitLightingEditor, 3  
~DakitMaterialEditor  
DakitMaterialEditor, 22  
~DakitPresentationByClassEditor  
DakitPresentationByClassEditor, 36  
ballPositionChanged  
deleteLight, 8  
deleteObjects, 12  
displayEditor, 3  
exportLighting, 9  
findById, 5  
findEditor, 5  
getLighting, 5  
getLightingFromWindow, 12  
importLighting, 9  
initEditor, 10  
initView, 14  
lightSelect, 5  
m\_ballX, 20  
m\_ballY, 20  
m\_ballZ, 20  
m\_database, 18  
m\_editorName, 18  
m\_interactorBall, 19  
m\_layerBall, 19  
m\_layerBallSmall, 20  
m\_lightBall, 19  
m\_lightingWindow, 18  
m\_lightingWorld, 18  
m\_numChanges, 18  
m\_objectBall, 20  
m\_objectBallSmall, 20  
m\_pDatabaseEditor, 18  
m\_presentationBall, 20  
m\_presentationBallSmall, 20  
m\_renderingDeviceContextBall, 19  
m\_renderingDeviceContextColor, 19  
m\_renderingDeviceID, 18  
m\_scaleBall, 20  
m\_sceneBall, 19  
m\_sceneColor, 19  
m\_sceneContextBall, 19  
m\_sceneContextColor, 19  
m\_templateBall, 19  
m\_templateBallSmall, 20  
m\_viewBall, 19  
m\_viewColor, 18  
m\_webViewerHeader, 18  
m\_webViewerLighting, 18  
m\_webViewerLightingOriginal, 18  
m\_windowBall, 19  
m\_windowColor, 19  
m\_worldID, 18  
mouseButtonDownCB, 17  
mouseButtonUpCB, 17  
mouseMoveCB, 17  
mouseWheelCB, 17  
newLighting, 9

~DakitLightingEditor, 3  
~DakitMaterialEditor  
DakitMaterialEditor, 22  
~DakitPresentationByClassEditor  
DakitPresentationByClassEditor, 36  
ballPositionChanged  
deleteLight, 8  
deleteObjects, 12  
displayEditor, 3  
exportLighting, 9  
findById, 5  
findEditor, 5  
getLighting, 5  
getLightingFromWindow, 12  
importLighting, 9  
initEditor, 10  
initView, 14  
lightSelect, 5  
m\_ballX, 20  
m\_ballY, 20  
m\_ballZ, 20  
m\_database, 18  
m\_editorName, 18  
m\_interactorBall, 19  
m\_layerBall, 19  
m\_layerBallSmall, 20  
m\_lightBall, 19  
m\_lightingWindow, 18  
m\_lightingWorld, 18  
m\_numChanges, 18  
m\_objectBall, 20  
m\_objectBallSmall, 20  
m\_pDatabaseEditor, 18  
m\_presentationBall, 20  
m\_presentationBallSmall, 20  
m\_renderingDeviceContextBall, 19  
m\_renderingDeviceContextColor, 19  
m\_renderingDeviceID, 18  
m\_scaleBall, 20  
m\_sceneBall, 19  
m\_sceneColor, 19  
m\_sceneContextBall, 19  
m\_sceneContextColor, 19  
m\_templateBall, 19  
m\_templateBallSmall, 20  
m\_viewBall, 19  
m\_viewColor, 18  
m\_webViewerHeader, 18  
m\_webViewerLighting, 18  
m\_webViewerLightingOriginal, 18  
m\_windowBall, 19  
m\_windowColor, 19  
m\_worldID, 18  
mouseButtonDownCB, 17  
mouseButtonUpCB, 17  
mouseMoveCB, 17  
mouseWheelCB, 17  
newLighting, 9

```
51 if (pEditor)
52 {
53     if (strcmp(parameter, "SHOW") == 0)
54         pEditor->showMaterial();
55     if (strcmp(parameter, "SET") == 0)
56         pEditor->setMaterial();
57 }
58 return;
59 }
```

## void Dakit\_PresentationByClassMenu (char \* parameter)

Called when user selects a menu entry

### Parameters:

*parameter* what menu entry was selected

```
Definition at line 88 of file DakitPresentationByClassEditorCB.cpp:90 {
91 DakitPresentationByClassEditor *pEditor =
DakitPresentationByClassEditor::findEditor(daf_getcurrentframe());
92 if (pEditor)
93 {
94     if (strcmp(parameter, "REVERT") == 0)
95         pEditor->revert();
96     if (strcmp(parameter, "SAVE") == 0)
97         pEditor->save();
98     if (strcmp(parameter, "EXPORT") == 0)
99     {
100         pEditorStatic = pEditor;
dad_setfileselectionparameters("", "XmlPresentationByClasses (*.xml)|*.xml|",
101 "No help what so ever.");
dad_displayfileselectionbox("Export", exportFileCB);
102 }
103 }
104 if (strcmp(parameter, "CREATE") == 0)
105     pEditor->createPresentation();
106 if (strcmp(parameter, "DELETE") == 0)
107     pEditor->deletePresentation();
108 }
109 return;
110 }
```

## void exportFileCB (char \* pFilename) [static]

Callback for exporting presentations to file

```
Definition at line 67 of file DakitPresentationByClassEditorCB.cpp:69 {
70 char filename[512];
71
72 if (pEditorStatic)
73 {
74     if (!strcmp(pFilename, ".xml"))
75         sprintf(filename, "%s.xml", pFilename);
76     else
77         sprintf(filename, "%s", pFilename);
78     pEditorStatic->export(filename);
79 }
80 pEditorStatic = 0;
81 return;
82 }
```

## Variable Documentation

DakitPresentationByClassEditor\* pEditorStatic = 0 [static]

Definition at line 61 of file DakitPresentationByClassEditorCB.cpp.

newLightingFromWindow, 9  
onMouseButtonDownCB, 16  
onMouseButtonUpCB, 16  
onMouseMoveCB, 16  
onMouseWheelCB, 17  
previewChange, 8  
rotateBallHorizontal, 15  
rotateBallVertical, 15  
scaleBall, 15  
setBallDirection, 15  
setBallPosition, 16  
setLighting, 4  
setLightingToWindow, 13  
setTitle, 10  
setWindow, 5  
typeChange, 7  
DakitLightingEditor.cpp, 51  
countLightingEditors, 54  
createBall, 53  
fIdLightingEditorSpec, 54  
hsvToRgb, 52  
idxLightingEditors, 54  
idxLightingEditorsById, 54  
idxLightingEditorsFields, 54  
idxLightingEditorsFieldsById, 54  
idxLightingEditorsFieldsById, 54  
idxLightingEditorsSpec, 54  
MAX, 51  
MIN, 51  
normalize, 53  
PI, 51  
rgbToHsv, 52  
selectionCb, 52  
tblLightingEditors, 54  
tblLightingEditorSpec, 54  
tblLightingEditorSpec, 54  
DakitLightingEditor.h, 55  
DakitLightingEditorCB.cpp, 56  
DakCB\_LightingButton, 57  
DakCB\_LightingChange, 56  
DakCB\_LightingMenu, 57  
DakCB\_LightSelect, 56  
DakitMaterialEditor, 21  
~DakitMaterialEditor, 22  
closeEditor, 23  
colorChange, 26  
createMaterial, 30  
DakitMaterialEditor, 22  
deleteMaterial, 30  
descriptionChange, 26  
displayEditor, 23  
edgeColorChange, 27  
edgeEffectChange, 29  
effectChange, 28  
export, 30  
findEditor, 24  
getSelectedMaterial, 33  
initMaterialsList, 31  
lineWidthChange, 28  
m\_editorName, 33  
m\_renderingDeviceContext, 33  
m\_renderingDeviceContextEdge, 34  
m\_renderingDeviceContextID, 34  
m\_renderingDeviceID, 33  
m\_sceneContextID, 34

m\_sceneID, 34  
m\_viewEdge, 34  
m\_viewID, 34  
m\_windowEdge, 34  
m\_windowID, 34  
m\_worldID, 33  
m\_xmlMaterials, 33  
materialChanged, 32  
materialSelected, 24  
onDelete, 31  
onInsert, 30  
onUpdate, 31  
operator=, 33  
revert, 29  
save, 29  
setTitle, 32  
shininessChange, 28  
showMaterial, 24  
updateMaterialsList, 31  
fIdLightingEditor.cpp, 58  
DakitMaterialEditors, 60  
countMaterialEditors, 60  
da\_aIdump\_dakitmaterialEditor, 60  
fIdMaterialEditorSpec, 59  
hsvToRgb, 58  
idxMaterialEditors, 60  
idxMaterialEditorsById, 59  
idxMaterialEditorsFields, 59  
idxMaterialEditorsSpec, 59  
MAX, 58  
MIN, 58  
rgbToHsv, 59  
tblMaterialEditors, 60  
tblMaterialEditorSpec, 59  
tmp, 60  
DakitMaterialEditor.h, 61  
DakitMaterialEditorCB.cpp, 62  
DakCB\_MaterialField, 62  
DakCB\_MaterialMenu, 63  
exportFileCB, 63  
pEditorStatic, 63  
DakitPresentationByClassEditor, 35  
~DakitPresentationByClassEditor, 36  
closeEditor, 37  
createPresentation, 40  
DakitPresentationByClassEditor, 36  
deletePresentation, 40  
descriptionChange, 38  
displayEditor, 36  
export, 39  
findEditor, 37  
getSelectedClass, 45  
getSelectedMaterial, 45  
getSelectedPresentation, 44  
initClassesTable, 41  
initMaterialsList, 42  
initPresentationList, 41  
m\_editorName, 46  
m\_fUpdateEditor, 46  
m\_pShowMaterialCB, 46  
m\_pShowMaterialContext, 46  
m\_xmlMaterials, 46  
m\_xmlPresentationByClass, 46  
onDelete, 40, 41  
onInsert, 40, 41

onUpdate, 40, 41  
operator=, 46  
presentationSelected, 38  
revert, 39  
save, 39  
setMaterial, 39  
setShowMaterialCB, 37  
setTitle, 44  
showMaterial, 38  
showPresentation, 44  
updateClassesTable, 43  
updateMaterialsList, 43  
DakitPresentationByClassEditor.cpp, 64  
classFromInt, 64  
countPresentationEditors, 66  
da\_aIdump\_dakitpresentationEditor, 66  
fIdPresentationEditorSpec, 65  
highlighted, 66  
idxPresentationEditors, 66  
idxPresentationEditorsFields, 65  
idxPresentationEditorsSpec, 65  
intFromClass, 65  
selected, 66  
stateFromString, 65  
stringFromState, 65  
tblPresentationEditors, 66  
tblPresentationEditorSpec, 65  
tmp, 66  
unselected, 66  
DakitPresentationByClassEditor.h, 67  
DakitPresentationByClassEditorCB.cpp, 68  
DakCB\_PresentationByClassButton, 68  
DakCB\_PresentationByClassField, 68  
DakCB\_PresentationByClassMenu, 69  
exportFileCB, 69  
pEditorStatic, 69  
dakitpresentationeditor\_t, 47  
classID, 47  
material, 47  
state, 47  
deleteLight  
DakitLightingEditor, 8  
deleteMaterial  
DakitMaterialEditor, 30  
deleteObjects  
DakitLightingEditor, 12  
deletePresentation  
DakitPresentationByClassEditor, 40  
descriptionChange  
DakitMaterialEditor, 26  
DakitPresentationByClassEditor, 38  
displayEditor  
DakitLightingEditor, 3  
DakitMaterialEditor, 23  
DakitPresentationByClassEditor, 36  
edgeColorChange  
DakitMaterialEditor, 27  
edgeEffectChange  
DakitMaterialEditor, 29  
effectChange  
DakitMaterialEditor, 28  
export  
DakitMaterialEditor, 30

DakitPresentationByClassEditor, 39  
exportFileCB  
DakitMaterialEditorCB.cpp, 63  
DakitPresentationByClassEditorCB.cpp, 69  
exportLighting  
findById  
DakitLightingEditor, 9  
findEditor  
DakitLightingEditor, 5  
DakitLightingEditor, 5  
DakitMaterialEditor, 24  
DakitPresentationByClassEditor, 37  
fIdLightingEditorSpec  
DakitLightingEditor.cpp, 54  
fIdMaterialEditorSpec  
DakitMaterialEditor.cpp, 59  
fIdPresentationEditorSpec  
DakitPresentationByClassEditor.cpp, 65  
getLighting  
DakitLightingEditor, 5  
getLightingFromWindow  
DakitLightingEditor, 12  
getSelectedClass  
DakitPresentationByClassEditor, 45  
getSelectedMaterial  
DakitMaterialEditor, 33  
DakitPresentationByClassEditor, 45  
getSelectedPresentation  
DakitPresentationByClassEditor, 44  
highlighted  
DakitPresentationByClassEditor.cpp, 66  
hsvToRgb  
DakitLightingEditor.cpp, 52  
DakitMaterialEditor.cpp, 58  
idxLightingEditors  
DakitLightingEditor.cpp, 54  
idxLightingEditorsById  
DakitLightingEditor.cpp, 54  
idxLightingEditorsFields  
DakitLightingEditor.cpp, 54  
idxLightingEditorsFieldsById  
DakitLightingEditor.cpp, 54  
idxLightingEditorsSpec  
DakitLightingEditor.cpp, 54  
idxMaterialEditors  
DakitMaterialEditor.cpp, 60  
idxMaterialEditorsFields  
DakitMaterialEditor.cpp, 59  
idxMaterialEditorsSpec  
DakitMaterialEditor.cpp, 59  
idxPresentationEditors  
DakitPresentationByClassEditor.cpp, 66  
idxPresentationEditorsFields  
DakitPresentationByClassEditor.cpp, 65  
idxPresentationEditorsSpec  
DakitPresentationByClassEditor.cpp, 65  
importLighting  
DakitLightingEditor, 9  
initClassesTable  
DakitPresentationByClassEditor, 41  
initEditor  
DakitLightingEditor, 10  
initMaterialsList

DakitMaterialEditor, 31  
DakitPresentationByClassEditor, 42  
initPresentationList  
DakitPresentationByClassEditor, 41  
initView  
DakitLightingEditor, 14  
interactorID  
lightingEditor\_t, 48  
initFromClass  
DakitPresentationByClassEditor.cpp, 65  
lightingEditor\_t, 48  
interactorID, 48  
name, 48  
pEditor, 48  
worldID, 48  
lightSelect  
DakitLightingEditor, 5  
linewidthChange  
DakitMaterialEditor, 28  
m\_ballX  
DakitLightingEditor, 20  
m\_ballY  
DakitLightingEditor, 20  
m\_ballZ  
DakitLightingEditor, 20  
m\_database  
DakitLightingEditor, 18  
DakitLightingEditor, 18  
m\_editorName  
DakitLightingEditor, 18  
DakitMaterialEditor, 33  
DakitPresentationByClassEditor, 46  
m\_ButtonLeft  
DakitLightingEditor, 18  
m\_UpdateEditor  
DakitMaterialEditor, 33  
DakitPresentationByClassEditor, 46  
m\_interactorBall  
DakitLightingEditor, 19  
m\_layerBall  
DakitLightingEditor, 19  
m\_layerBallSmall  
DakitLightingEditor, 20  
m\_lightBall  
DakitLightingEditor, 19  
m\_lightingWindow  
DakitLightingEditor, 18  
m\_lightingWorld  
DakitLightingEditor, 18  
m\_numChanges  
DakitLightingEditor, 18  
m\_objectBall  
DakitLightingEditor, 20  
m\_objectBallSmall  
DakitLightingEditor, 20  
m\_pDatabaseEditor  
DakitLightingEditor, 18  
m\_presentationBall  
DakitLightingEditor, 20  
m\_presentationBallSmall  
DakitLightingEditor, 20  
m\_pShowMaterialCB  
DakitPresentationByClassEditor, 46  
m\_pShowMaterialContext

DakitMaterialEditor, 32  
materialEditor\_t, 49  
name, 49  
pEditor, 49  
materialSelected  
DakitMaterialEditor, 24  
MAX  
DakitLightingEditor.cpp, 51  
DakitMaterialEditor.cpp, 58  
MIN  
DakitLightingEditor.cpp, 51  
DakitMaterialEditor.cpp, 58  
mouseButtonDownCB  
DakitLightingEditor, 17  
mouseButtonUpCB  
DakitLightingEditor, 17  
mouseMoveCB  
DakitLightingEditor, 17  
mouseWheelCB  
DakitLightingEditor, 17  
name  
lightingEditor\_t, 48  
materialEditor\_t, 49  
presentationEditor\_t, 50  
newLighting  
DakitLightingEditor, 9  
newLightingFromWindow  
DakitLightingEditor, 9  
normalize  
DakitLightingEditor.cpp, 53  
onDelete  
DakitMaterialEditor, 31  
DakitPresentationByClassEditor, 40, 41  
onInsert  
DakitMaterialEditor, 30  
DakitPresentationByClassEditor, 40, 41  
onMouseButtonDownCB  
DakitLightingEditor, 16  
onMouseButtonUpCB  
DakitLightingEditor, 16  
onMouseMoveCB  
DakitLightingEditor, 16  
onMouseWheelCB  
DakitLightingEditor, 17  
onUpdate  
DakitMaterialEditor, 31  
DakitPresentationByClassEditor, 40, 41  
operator=  
DakitMaterialEditor, 33  
DakitPresentationByClassEditor, 46  
pEditor  
lightingEditor\_t, 48  
materialEditor\_t, 49  
presentationEditor\_t, 50  
pEditorStatic  
DakitMaterialEditorCB.cpp, 63  
DakitPresentationByClassEditorCB.cpp, 69  
PI  
DakitLightingEditor.cpp, 51  
presentationEditor\_t, 50  
name, 50  
pEditor, 50  
presentationSelected

DakitPresentationByClassEditor, 38  
previewChange  
DakitLightingEditor, 8  
revert  
DakitMaterialEditor, 29  
DakitPresentationByClassEditor, 39  
rgbToHsv  
DakitLightingEditor.cpp, 52  
DakitMaterialEditor.cpp, 59  
rotateBallHorizontal  
DakitLightingEditor, 15  
rotateBallVertical  
DakitLightingEditor, 15  
save  
DakitMaterialEditor, 29  
DakitPresentationByClassEditor, 39  
scaleBall  
DakitLightingEditor, 15  
selected  
DakitPresentationByClassEditor.cpp, 66  
selectionCB  
DakitLightingEditor.cpp, 52  
setBallDirection  
DakitLightingEditor, 15  
setBallPosition  
DakitLightingEditor, 16  
setLighting  
DakitLightingEditor, 4  
setLightingToWindow  
DakitLightingEditor, 13  
setMaterial  
DakitPresentationByClassEditor, 39  
setShowMaterialCB  
DakitPresentationByClassEditor, 37  
setTitle  
DakitLightingEditor, 10  
DakitMaterialEditor, 32  
DakitPresentationByClassEditor, 44  
setWindow  
DakitLightingEditor, 5  
shininessChange  
DakitMaterialEditor, 28  
showMaterial  
DakitMaterialEditor, 24  
DakitPresentationByClassEditor, 38  
showPresentation  
DakitPresentationByClassEditor, 44  
state  
dakipresentationeditor\_t, 47  
stateFromString  
DakitPresentationByClassEditor.cpp, 65  
stringFromState  
DakitPresentationByClassEditor.cpp, 65  
tblLightingEditors  
DakitLightingEditor.cpp, 54  
tblLightingEditorSpec  
DakitLightingEditor.cpp, 54  
tblMaterialEditors  
DakitMaterialEditor.cpp, 60  
tblMaterialEditorSpec  
DakitMaterialEditor.cpp, 59  
tblPresentationEditors  
DakitPresentationByClassEditor.cpp, 66

tblPresentationEditorSpec  
    DakitPresentationByClassEditor.cpp, 65  
tmp  
    DakitMaterialEditor.cpp, 60  
    DakitPresentationByClassEditor.cpp, 66  
typeChange  
    DakitLightingEditor, 7  
unselected  
    DakitPresentationByClassEditor.cpp, 66  
updateClassesTable  
    DakitPresentationByClassEditor, 43  
updateMaterialsList  
    DakitMaterialEditor, 31  
    DakitPresentationByClassEditor, 43  
worldID  
    lightingEditor\_t, 48