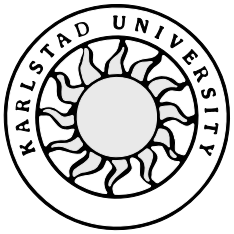

Computer Science

Anders Åslund

Object-Oriented Design of Special Effects in OpenGL

Bachelor's Project

2000:34

Object-Oriented Design of Special Effects in OpenGL

Anders Åslund

This report is submitted in partial fulfilment of the requirements for the Bachelor's degree in Computer Science. All material in this report which is not my own work has been identified and no material is included for which a degree has previously been conferred.

Anders Åslund

Approved, 00-08-21

Advisor: Hua Shu

Examiner: Stefan Lindskog

Abstract

Saab Bofors Dynamics' needs lie in a software system that provides a modular and component-based structure built for simulations and visualization purposes. This thesis is a description of an attempt to design such a system. The market offers several software components that are useful in the process of building visualization and simulation systems. The design of the system presented in this thesis is not dependent on the choice of product and can with a little work be used with all the products. The design is based on patterns in the sense of well-defined solutions and documented structures. The design also provides a component structure where functionality is embedded in Dynamic Linked Libraries. The most important parts of the design are implemented and the design is proven to be effective and meet the requirements for the project.

Contents

List of Figures

List of tables

1 About the thesis

This section is a brief introduction to each of the following chapters.

Introduction

This chapter describes the background of this project and the problems that are supposed to be solved, SAAB Bofors Dynamics' current situation and their need for this type of project. Conditions and requirements are primarily formed according to the demands and needs of SAAB Bofors Dynamics.

Preliminaries

This chapter is a brief resume of things that is necessary to know before continuing to read about the design and solution.

Description of design and solution

This chapter describes the solution and design thoroughly. The understanding of the following parts needs some knowledge in software-design and object oriented methods.

Implementation and testing

This chapter describes the implementation and all the coding solutions that are necessary to make the design work. It also describes some nice coding structures and frameworks.

Experiences and recommendations

This chapter discusses things to learn from this project.

Please note that chapters 3.1 and 3.2 are written together with Henrik Hedlund, in conjunction with his Bachelor's project "Virtual Reality Applications - Evaluation of Development Environments."

2 Introduction

This thesis is the first result of a Master's project and is written as a Bachelor's project at the Department of Computer Science, Karlstad University for SAAB Bofors Dynamics. The first chapter describes the background of this project and the problems that are supposed to be solved, SAAB Bofors Dynamics' current situation and their need for this type of project.

2.1 Background

Today's defence industry is very hi-tech oriented and has a need of impressive and cost efficient visual demonstrations of newly developed systems. Simulators for educational and practice purposes are often demanded by the customers. SAAB Bofors Dynamics has long experience in the area of developed simulations but lacks the expertise in visual simulators. Subcontractors have developed simulators for a couple of the systems and some of them have been implemented at Bofors. To use a visual simulator or a demonstrator for marketing, an impressive and realistic presentation is important. Special effects and dynamics make the presentation more realistic and increase the feeling of quality in the product.

2.2 Problems

The problem is that the defence industry uses special computers (Silicon Graphics) for simulations, but standard computers have in recent years gotten a greater breakthrough on the market. There exist a number of interesting commercial products that will assist with the development of visual simulators. The software and the quality of the products vary greatly. These potential tools that can handle the task, have to be evaluated. This project is intended to be flexible in choice of products and software. But certain requirements have to be fulfilled. The design of the project has to be modular and extendable so that after it is released additional functionalities can be added.

2.3 Purpose

Today's computer games, running on ordinary home computers, are often more advanced and realistic than the old simulators that the defence-industry is using. The purpose of this project is to make visual simulators and demonstrators that run on ordinary home computers with

new graphics hardware. Increased knowledge and a well-defined and flexible structure to build the future simulators on are also important. The project is a base for a Master project that will contain a better-studied structure with a more extensive implementation of special effects and preparations to make the system distributed over a network.

2.4 Conditions

These conditions and requirements are primarily formed according to the demands and needs of SAAB Bofors Dynamics. The time-related conditions are based on the Bachelor's project course time span. During the spring term of 2000, two or three days each week have been dedicated to work with the Bachelor's project. The following stages describe the steps of the development.

2.4.1 Stages

Since the development of software is an iterative process, the boundaries of these stages might overlap each other.

Information gathering

The first step in the project is an information-gathering process, where commercial software is evaluated. The evaluation is built on what the software and programs have to offer and how flexible they are. This information will be used as a base for the next step.

Design and development

The analysis of the gathered data and information helps to draw the conclusion of which software suits our purpose best. The design and development work will lead to the actual assignment and a requirement specification is defined.

Implementation

The design is implemented in a demo version and refinements of the design are made. Testing the system comes naturally during the implementation process.

Documentation

Daily notes, documentation of the design and well-documented code are the base for the documentation of the system.

2.5 Requirements

The following are the requirements for the Bachelor project.

2.5.1 Software

These are the requirements for the software to be purchased.

The general visual requirements are not really important but some kind of realistic impression must be presented. The design must allow additions with extensive real-time computations like dynamics, as in aerodynamics, motion-models, of for example a boat or an aeroplane, and particle-systems, like rain or fire. Pre-calculated data of more advanced computations can be fed to the system from an external simulation over a network or from a file. More important is the development environment that must support object-oriented solutions. An abstract interface that makes the programming easier and other features are of interest. The software must offer a good base for realistic, fast and good-looking presentation on screen. Multi-platform support is not a requirement but it is preferable considering the nature of the project. Figure 2.1 describes the systems of interest and their relation to each other. To reach the widest market of platforms and hardware, OpenGL must be the core of the software because it is the most supported graphics API on the market. Software licenses are often very expensive, not only the development environment, but also run-time licenses are charged by the software companies. The distribution of the developed products resulting from an API depends on the price of the license. An expensive license would restrict the width of the distribution.

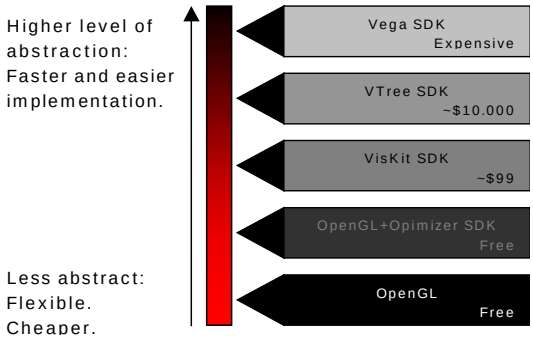

Figure 2.1 - Summary of APIs

The support for various formats of geometries and images is necessary. Outdoor environments are often used in simulations, which means that a large amount of geometric data has to be processed. Some way to make these large data sets more efficiently processed is really useful. One way is to have different stages of Level of Detail (LOD), which means that a simplified model is used when viewed from a distance or viewed at high speed.

2.5.2 3D Graphics

Special effects are the main aspect of this project. However in the Bachelor project only the preparation of a flexible structure is required. The structure and design must allow a flexible use of special effects. There are a lot of effects that have to be taken into account. Dynamic effects have a limited lifetime, while permanent effects are persistent and do not disappear. Connections between objects are therefore necessary to test collisions and other links. Infra red (IR) vision and sensors in the 3D-world are not required.

2.5.3 Distributed interaction

Interaction over a network or over Internet is of interest and has to be included as criterion when designing the system. To make a distributed simulation work, some aspects and questions have to be looked into. The prediction of events is needed to synchronize events in both ends of the communication link. In this project, we do not consider the distribution of geometries of terrain or models.

2.5.4 Architecture

The design must be modular, flexible and have the ability to be reused and altered both before and after compilation. Using plug-ins solves this problem and makes the system scalable and flexible. By dynamically allocating the plug-ins the system becomes completely reusable and alterable after compilation. A simple graphic user interface (GUI) can be implemented for each plug-in, to configure different parameters. Hardware in form of keyboard, joystick, pointer or other is implemented in case of use.

2.5.5 Design patterns

The basic design structure is based on the Booch notation (Appendix D), which describes in an object-oriented way the design of an application. The use of patterns, known solutions to defined problems or designs, are preferable. In this way the modularity can be achieved.

3 Preliminaries

The following chapter is a brief resume of things that is necessary to know before continuing to read about the design. Readers that have experience and knowledge in 3-dimensional visualization, OpenGL and object-oriented methods can skip this chapter.

3.1 Simulation/visualization

Simulation can generally be described in two ways, either as an accurate, numeric calculation of how something behaves under certain conditions, or as a real-time, interactive visualization of a scenario based on some realistic approximations. Basically this means that it is impossible to use accurate calculations and behaviour models in real-time visualizations. These accurate simulations are used for exact predictions of a scenario, and are calculated over a large time span. On the other hand it is possible to pre-calculate data and then feed it to a real-time visualization. This project does not span all the ingredients of accurate simulation; it is mainly focused on visualization.

Visualization in turn is a multi-faceted task, ranging from the illustration of simple graphs and tables to three-dimensional (3D) presentation of a sequence of actions. The particular area of interest in this project is computer-aided 3-dimensional visualizations of real-time calculated data. To achieve the visualization in real-time, simplified models of behaviour are required. This principle applies in many of today's popular computer games, where speed is crucial for viewing pleasure. Other possible areas of application include architecture, design and prototyping, education, conferencing, military training simulators, scientific visualization and surgical practice. SAAB Bofors Dynamics' connection to the defence industry leads to the particular area of military training simulators and demonstrators. These include, but are not limited to, driving, flight, ship and tank simulators.

3.2 Three-dimensional graphics

In this section, a short introduction to 3D graphics provides the reader with a basic understanding of 3D graphics programming.

The *coordinate system* shown in Figure 3.1 is fundamental to understand 3D orientation. Placement and direction of the axes can vary dependent on different systems and environments.

The smallest building block is called a *vertex*, and represents one point in the coordinate system. Lines consist of two vertices, start-point and end-point.

By connecting multiple lines basic two-dimensional primitives such as *triangles* (3 vertices), *quadrants* (4 vertices) and *polygons* (N vertices) can be formed.

The most common 3D *primitives* are boxes, cones, cylinders and spheres. In addition, surfaces and materials are defined and solid objects can be formed. Material consists of different aspects such as light dependent attributes (shininess and emissiveness) and colours.

To add further realism to the surface a *texture* can be applied. The texture is a picture that adds the appearance of a real surface.

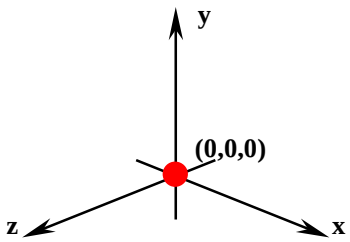

Figure 3.1 – Coordinate system

Transformation is used to translate and rotate an object. To translate an object is to place, or move, it in the coordinate system. Shading and lightning are other features that add realism and atmosphere to the scene.

A *scene* is a virtual environment that contains objects and attributes. A graph is often used to represent the scene, by ordering all attributes and objects in a hierarchic manner (Figure 3.2). This means that a child inherits the attributes of its parent. A very complex engine performs the actual rendering, but this is out of the scope of this thesis.

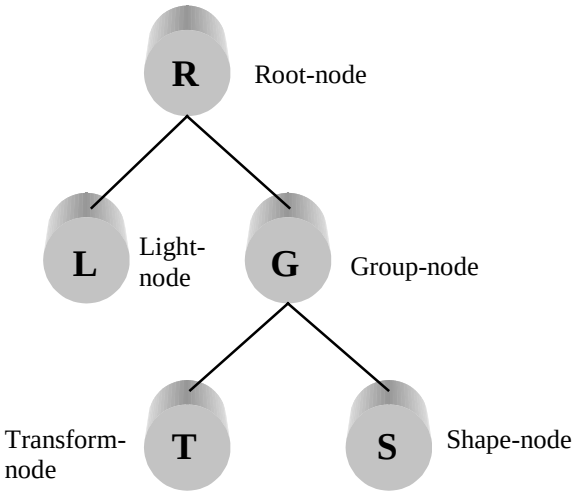

Figure 3.2 – Scene graph

A scene graph is a *directed acyclic graph*. The graph in Figure 3.2 is an example that contains five different nodes. These are the most common nodes, but the different types of nodes can vary from system to system. A group-node is a node, which does not have any attributes in itself, but can have children. The root-node is basically a group-node without any parents, thus being the group-node of the whole scene. Light-nodes contain light-attributes for the scene. A transformation-node determines the placement and orientation of the geometry, which is contained in the shape-node.

3.3 OpenGL

This and the following sections assume that the reader has at least a rudimentary understanding in programming and some familiarity with basic terms used in computer graphics. The OpenGL Graphics System: A Specification (Version 1.2.1) [1] gives the following definition of OpenGL.

> "OpenGL (for "Open Graphics Library") is a software interface to graphics hardware. The interface consists of a set of several hundred procedures and functions that allow a programmer to specify the objects and operations involved in producing high-quality graphical images, specifically colour images of three-dimensional objects. Most of OpenGL requires that the graphics hardware contain a framebuffer. Many OpenGL calls pertain to drawing objects such as points, lines, polygons, and bitmaps, but the way that some of this drawing occurs (such as when antialiasing or texturing is enabled) relies on the existence of a framebuf-

fer. Further, some of OpenGL is specifically concerned with framebuffer manipulation.” page 1

Another way to see it is that OpenGL is a set of commands that allow the specification of two- or three-dimensional geometric objects and commands that specify how these will be rendered into the framebuffer. OpenGL specifies how geometries are lit, coloured and mapped from the two- or three-dimensional model space to the two-dimensional screen. Graphics hardware may consist of various degrees of graphics acceleration. The choice of commands to be executed by the graphics hardware or by the central processing unit (CPU) to get optimum performance is done automatically. The OpenGL Graphics System: A Specification (Version 1.2.1) [1] also describes Silicon Graphics' (SGI’s) view of their product.

> “We view OpenGL as a state machine that controls a set of specific drawing operations. This model should engender a specification that satisfies the needs of both programmers and implementors. It does not, however, necessarily provide a model for implementation. An implementation must produce results conforming to those produced by the specified methods, but there may be ways to carry out a particular computation that are more efficient than one specified.”, page 3

3.4 Plug-ins/Dynamic Link Libraries

Executable files are generally programs that perform various tasks and can receive user input. Dynamic link libraries (DLLs) are usually not executables, and do not receive messages. Both DLLs and executables contain executable code and/or resources for an application or DLL. DLLs are files that contain functions that can be called by programs and other DLLs [2]. Usually the DLLs are loaded by Windows only once, and can thereafter be shared by several applications.

There are two kinds of DLLs, statically and dynamically linked DLLs. Statically linked DLLs are used as libraries with functions during the implementation. When linked to an application, they become a permanent part of the application’s executable file. This type of DLL resides in file with the extension ".LIB" and is basically an object file. This type of DLL is frequently used in this project.

More interesting are the dynamically linked DLLs. Dynamically linked DLLs provide the means to load a library to memory in runtime, resolve references to functions so that they can be called by an application, and then unload when they are no longer needed. Implicit dynamic linking is used to make the operating system load the DLLs and let several applications

use it. Explicit dynamic linking is used to explicitly load the DLLs into memory, use them, and then unload them if they are of no use anymore. The structure of the system resulting from this project is built with dynamically linked DLLs.

3.5 Patterns and framework

The main goal is to get a good architecture that does not need to be redesigned. The object-oriented concept is based on collaboration among objects, and modularity that makes complex system far simpler to overview. It is hard to make reusable object-oriented software and it takes a long time to analyse and design. It takes a really experienced designer to make a good design that is reusable. This needs a solution that is concentrated on the problem and the requirements. Therefore the design of this system is based on design patterns, i.e. designs made by very experienced designers. Patterns are well-described solutions for certain types of problems and requirements. The purpose of patterns is to suggest a standardized solution that is well documented and easily recognized by other programmers and designers. Using patterns is a really easy way to get a good design that is tested and has the properties of being flexible and reusable. Design pattern and framework helps to improve quality of software and shows the possibility to re-use components, code and even designs. Framework is the implementation of patterns using object-oriented programming languages. It is an adaptation of the pattern to make it suitable for the problem at stake and to make it compatible with the language in use. Often a pattern can be used to document and introduce frameworks.

4 Description of design and solution

This chapter describes the solution and design thoroughly. The understanding of the following parts needs some knowledge in software design and object oriented methods.

4.1 Environment

Choosing an environment in form of application programming interfaces (APIs) and platform was the first task. The Windows NT platform was chosen because it is cost-efficient and offers a wide spectrum of hardware and software support. A high performance to a low cost is achieved with the new gaming graphics hardware, which supports all the features needed when making realistic simulations and visualizations. The APIs in question have to support OpenGL because it supports multiple platforms and large amounts of hardware. The APIs must have an object-oriented structure that admits advanced object-oriented solutions.

4.1.1 Vega

Vega is the most expensive API/program. It has a GUI, which makes it more like a program with drag-and-drop functionality. Nevertheless Vega requires some programming to develop software. Vega is a well though through API that makes a good impression. Vega is more expensive than VTree and it requires a runtime license for each workstation it is executed on. It is object-oriented friendly and allows good programming solutions.

4.1.2 VTree

VTree is a complete API that contains all necessary functionality and is built on OpenGL. It is developed to be an effective programming API when making simulator for games or professional use. Vtree supports visualizations of large data sets and geometries. A developer license costs less than Vega and there are 25 runtime licenses included. This makes the distribution easier compared to Vega. The software kit includes good programs, examples and documentation. Vtree is developed using object-oriented methods and it is very flexible.

4.1.3 VisKit

VisKit is mainly built on Direct3D, which is Microsoft’s version of a low-level graphics API. OpenGL is partially supported. VisKit costs only about a hundred dollars and has some good features. It admits object-oriented solutions and is very adapted to Windows.

4.1.4 OpenGL Optimizer

OpenGL does the image processing and additional features like scene handling is done by Cosmo3D, which is also developed by SGI, and the optimising is done by the OpenGL Optimizer. This is developed for professional visualization of large data sets and geometries. It is freely downloaded from SGIs homepage and includes a good documentation and some sample code. Is totally object-oriented and admits advanced solutions.

4.1.5 OpenGL

OpenGL is a professional low-level graphics API that is developed by SGI and is used by most CAD-softwares and some games. It is included in the operating system and it does not require any additional purchases. It is therefore very cheap but it requires a great deal of programming. It does not contain any specific object-oriented methods or solutions; it is therefore a low-level API.

4.1.6 Conclusion

The conclusion is that all APIs allow object-oriented solutions in one way or another (OpenGL needs more work than the others). Pure OpenGL demands a lot of designing and programming to develop software. OpenGL Optimizer was a good contender and is still interesting. VisKit was not sufficiently advanced and Vega is too expensive. Finally VTree was chosen and one license was bought. It was the best solution that did not demand excessive programming work. VTree is well adapted to object-oriented methods and solutions, which makes it very flexible.

4.2 Design

This section will describe the design of the system. The design is based on two main ideas, an abstract interface to the graphics API and the dynamic loading of plug-ins. The designs of all the components as well as the interface to the graphics API are based on design patterns.

4.2.1 Object orientation

The main aspects of the design are the requirements of the reusability of the system architecture and modularity both in design structure and in the separation of the system in DLLs. To match the requirements and problems to different design solutions using patterns is rather easy if you compare the work involved to do the whole design from scratch. One of the big problems to focus on is that even if this system is implemented only using Windows NT and

the VTree API, it should be easy to migrate it to a different platform and API. The reusability of the components is in this case the main concern.

4.3 Bridge pattern

The bridge pattern is used to abstract functionality. The implementation is separated and hidden behind an interface. The bridge pattern is often called interface pattern.

4.3.1 Intent

Decouple an abstraction from its implementation so that the two can vary independently [3].

4.3.2 Applicability

Use the Brige pattern when:

- You want to avoid permanent binding between an abstraction and its implementation. This might be the case, for example, when the implementation must be selected or switched at run-time [3].
- Changes in the implementation of an abstraction should have no impact on clients; that is, their code should not have to be recompiled [3].
- (C++)You want to hide the implementation of an abstraction completely from clients. In C++ the representation of a class is visible in the class interface [3].
- You want to share an implementation among multiple objects (perhaps using reference counting), and this fact should be hidden from the client.

4.3.3 Motivation

Figure 4.1 shows how the application uses the VTree API. The problem is how do we use the application on top of a different API, without rewriting the application.

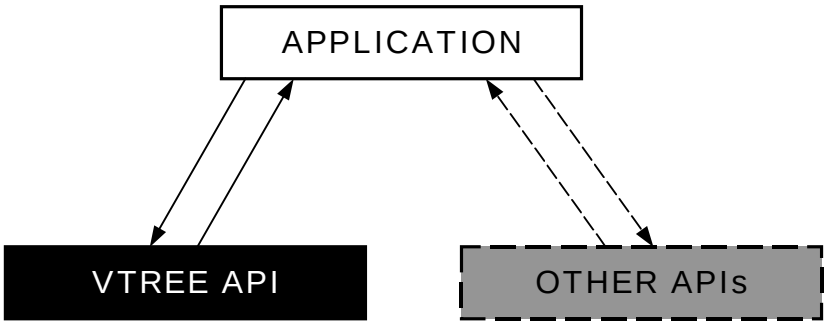

Figure 4.1 - Different APIs

The easiest way to do this is to define an interface with an adaptation of the APIs to make them fit this interface, as shown in Figure 4.2.

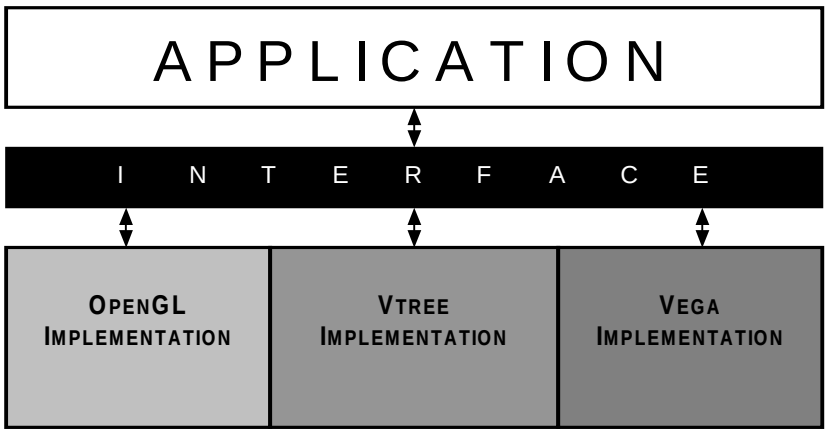

Figure 4.2 - The interface

This structure is now very easy to map to a design pattern. The suitable pattern for this structure is the Bridge pattern, which is illustrated in Figure 4.3. The main idea is to abstract the implementation of specific functionality. The Bridge avoids permanent bindings between an application and its implementation. The only drawback is that there is a lot of work involved to implement the interface with all the APIs. But that is a future problem.

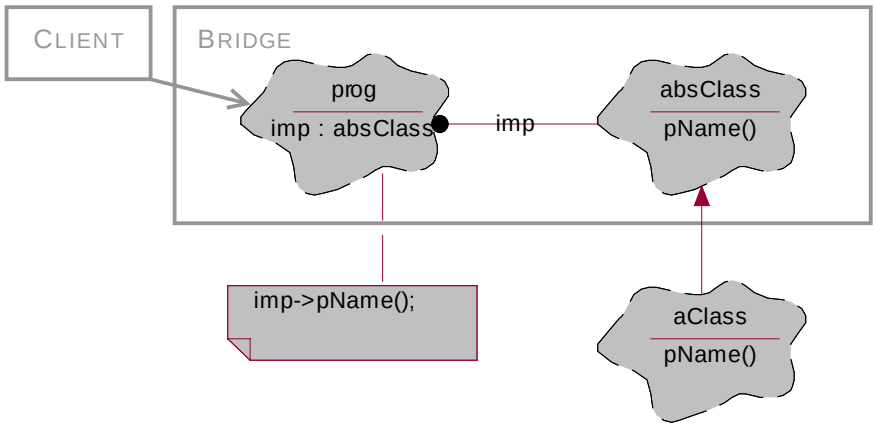

Figure 4.3 - The bridge pattern

4.4 Abstract factory pattern

The abstract factory pattern is a modification of the factory pattern. The factory pattern abstracts the process of creating objects and desired factory can be chosen at runtime. The abstract factory pattern is a factory with an abstract interface. All the products that the abstract factory produces must have an abstract interface. This removes type dependencies from clients.

4.4.1 Intent

Provide an interface for creating families of related or dependent objects without specifying their concrete classes [3].

4.4.2 Applicability

Use the Abstract Factory pattern when:

- A system should be independent of how its products are created, composed, and represented [3].
- A system should be configured with one of multiple families of products [3].
- A family of related product objects is designed to be used together, and you need to enforce this constraint [3].
- You want to provide a class library of products, and you want to reveal just their interfaces, not their implementations [3].

4.4.3 Motivation

The structure in Figure 4.3 is used as a simple example to visualize the solution to the problems that are encountered while designing the system. The client (prog), for example, must know which class is used to implement the abstract interface class (absClass). This is not very good because the application class (prog) has to be modified if we want to replace the implementation class (aClass). Moving the creation of the implementation class outside the client solves this. This eliminates the type dependencies in the client so that it won't need to be rewritten. To make the design of this part just as good as the rest, a design pattern for the problem is required. The abstract factory pattern is used, which is shown in Figure 4.4. The only drawback by doing this is that the factory class, aFactory in this case, needs to be implemented for each implementation class.

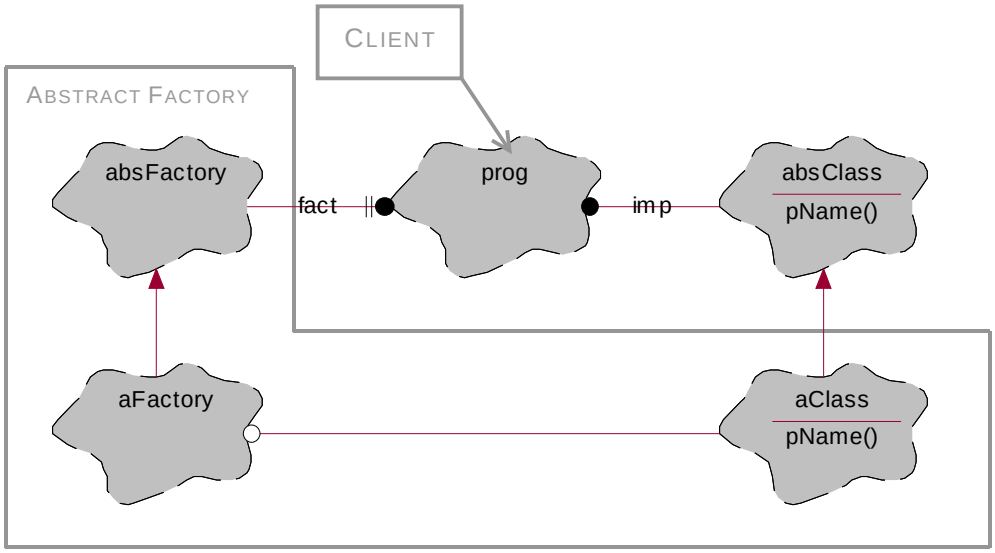

Figure 4.4 - The abstract factory pattern

4.5 Adapter pattern

Now to the tricky part, namely to get the implementation classes collaborating in the right way with each other. One can regard the implementation class as an adaptation of the interface to match it to another interface such as an existing API. This leads to the next pattern to be used called the adapter pattern. The adapter pattern is used to convert one interface into another interface. It adapts the concrete implementor to fit the target interface.

4.5.1 Intent

Convert the interface of a class into another interface client expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces [3].

4.5.2 Applicability

Use the Adapter pattern when:

- You want to use an existing class, and its interface does not match the one you need [3].
- You want to create a reusable class that cooperates with unrelated or unforeseen classes, that is, classes that don't necessarily have compatible interfaces [3].
- (object adapter only)You need to use several existing subclasses, but it's impractical to adapt their interface by subclassing every one. An object adapter can adapt the interface of its parentclass [3].

4.5.3 Motivation

The main use of this pattern in this case is to create an interface that cooperates with unrelated or unforeseen classes, classes that do not necessarily have compatible interfaces. The class adapter uses multiple inheritances to adapt one interface to another, which is shown in Figure 4.5.

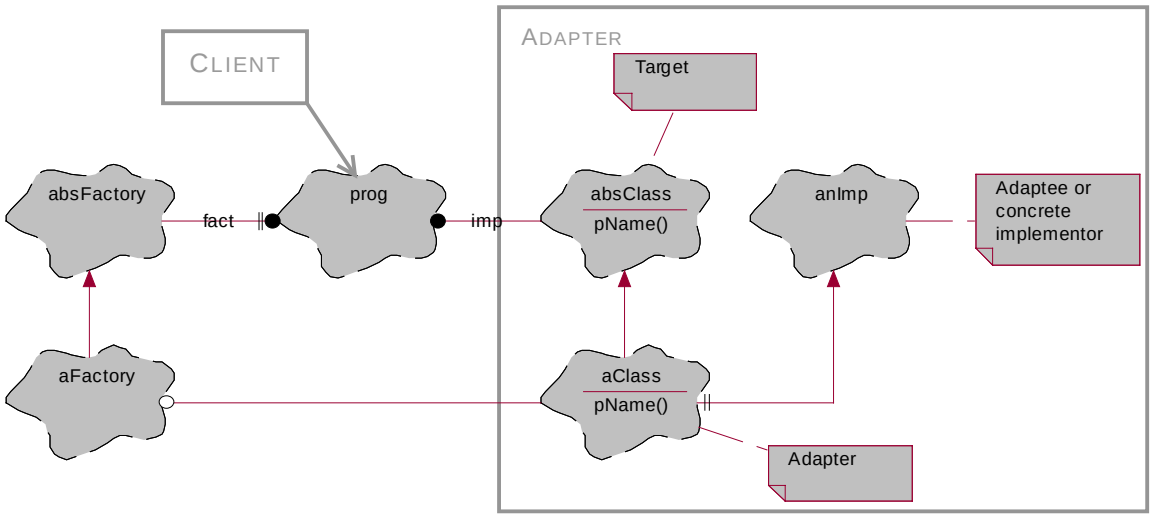

Figure 4.5 - The adapter pattern

This example (shown in Figure 4.6) is implemented as a framework and can be found in appendix B; the structure has been modified to show how the objects collaborate with each other. This is important to the functionality of the real system.

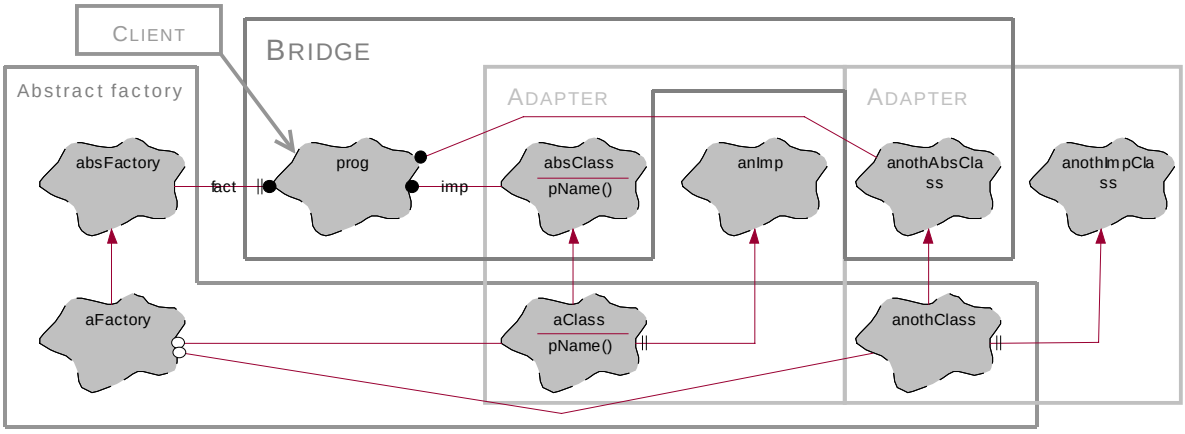

Figure 4.6 - Patterns working together

4.6 Plug-ins

Now to the DLLs and how they are designed. The concept is almost the same as an abstract interface. The big difference is that a DLL cannot be instantiated like a class. The interface is defined in a module definition file. This is a very rough way to describe an interface and great caution must be taken. This is because the heads of the functions in the DLL are not defined; there is only a list of function names. Figure 4.7 shows the similarity between the module definition file and the abstract class interface in Figure 4.2. The DLL-handler is described in section 5.2.

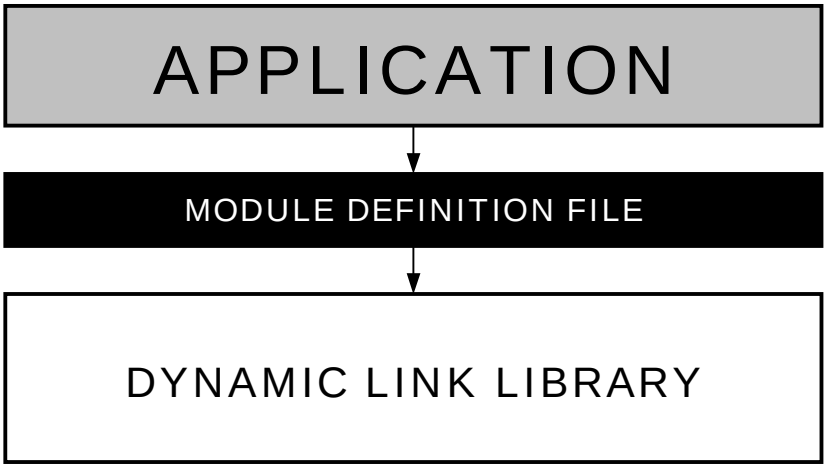

Figure 4.7 - Module definition file

5 Implementation and testing

This chapter describes the implementation and all the coding solutions that are necessary to make the design work. It also describes some nice coding structures and frameworks.

5.1 Framework

The design is a scheme that shows the collaboration between the classes. The framework is the implementation of the design scheme. This will shed some light over the implementation. First the detailed scheme is shown (Figure 5.1*)* to visualize the complexity of this project. Figure 5.1 also introduces the API and the abstract interface. It must be added that the implementation is not complete and this picture is not final in any way, but this is a base for the construction.

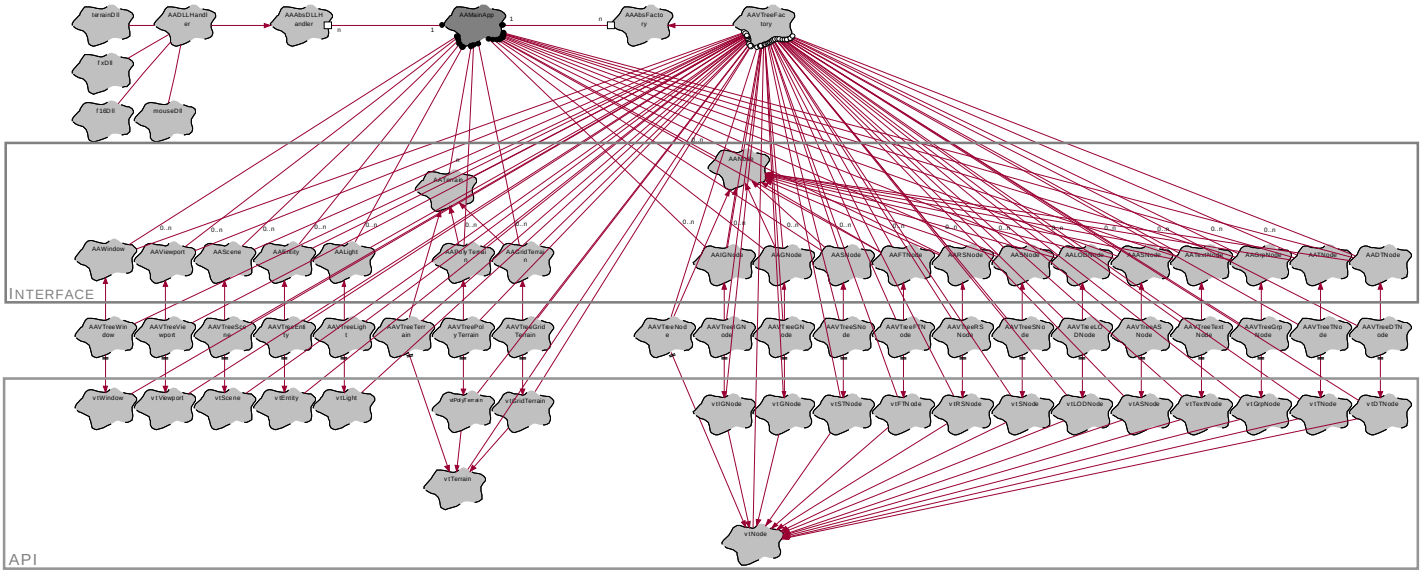

Figure 5.1 - Class diagram, overview

The purpose of the design is to place all the simulated actions and event handling in DLLs. This makes the system totally alterable after compilation. If the DLL-handling admits, one DLL can be replaced by another to alter the behaviour of the application.

The first step is to find the patterns that were defined in the design phase. The next picture (Figure 5.2) is a simpler picture of the same design with the patterns defined. The factory pattern is hard to visualize in this picture but there is a good description of it in section 4.4.

There are a couple of issues that need to be addressed to explain how the patterns are implemented.

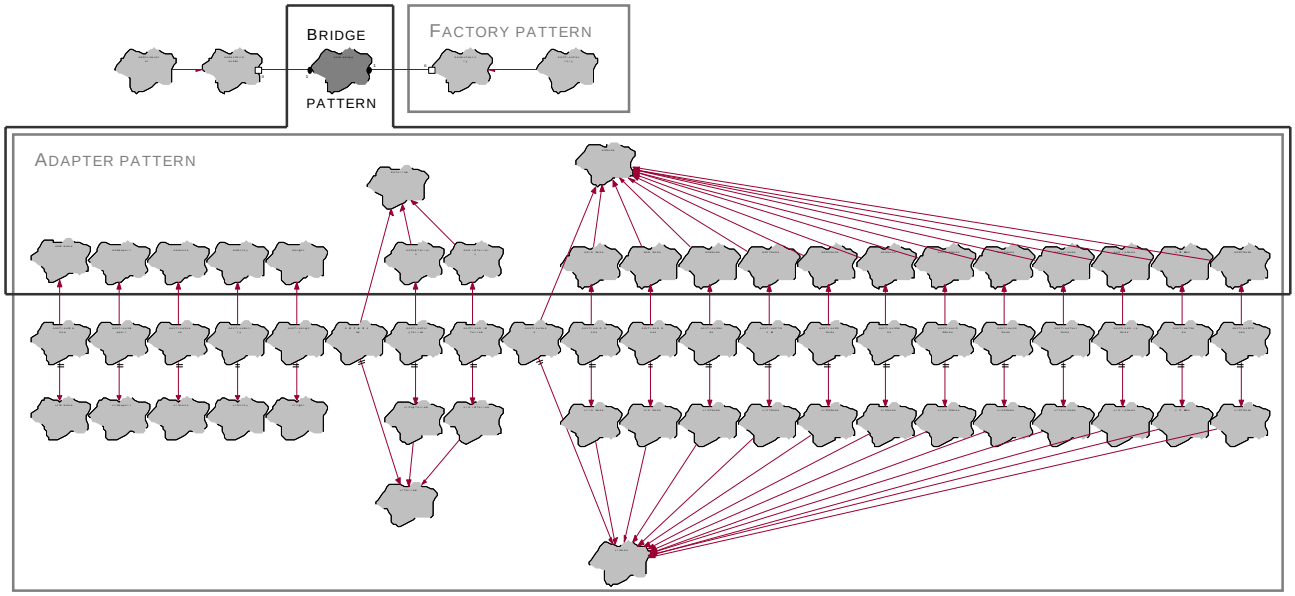

Figure 5.2 - Class diagram, patterns

There are two ways to implement the factory pattern, one concrete factory-class for each adapter class or just one concrete factory-class for the whole abstract interface. In our case there is no need for different factories so the system is implemented with the second alternative. The code for the factory is very simple and does not need much explanation. In appendix B there is a good and easy example of a factory class that shows how it is implemented. A problem that was solved by this structure was that the DLLs do not use the same environment as the executable part of the system. If an object was created in one DLL that was supposed to add something to the scene it would not be reachable from the application. This structure admits that the factory is passed to the DLLs as a pointer or reference and the DLLs create all the objects in the applications environment instead of the DLL's *(*Figure 5.3*)*.

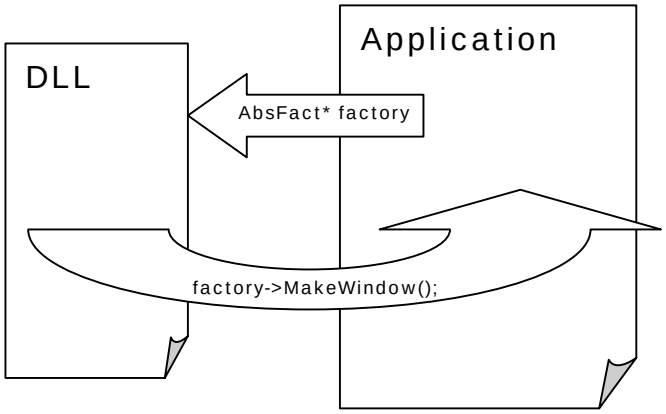

Figure 5.3 - Memory areas

The adapter pattern is not much more complicated than the factory pattern but it took some time to figure out how it could be implemented. The big problems were that the different classes in the API use each other, and take each other as parameters. Using the class adapter pattern, distinguished from the object adapter pattern, solved this. The class adapter pattern uses inheritance to inherit the procedures and functionality. The object adapter pattern instantiates the class that it needs and uses it as a parameter. There are also certain relationships between them such as hierarchical relationships in form of inheritance. These structures were preserved by the abstract interface so that the functionality is not altered. Figure 5.4 shows an example where a hierarchical structure is preserved in the abstract interface from the API.

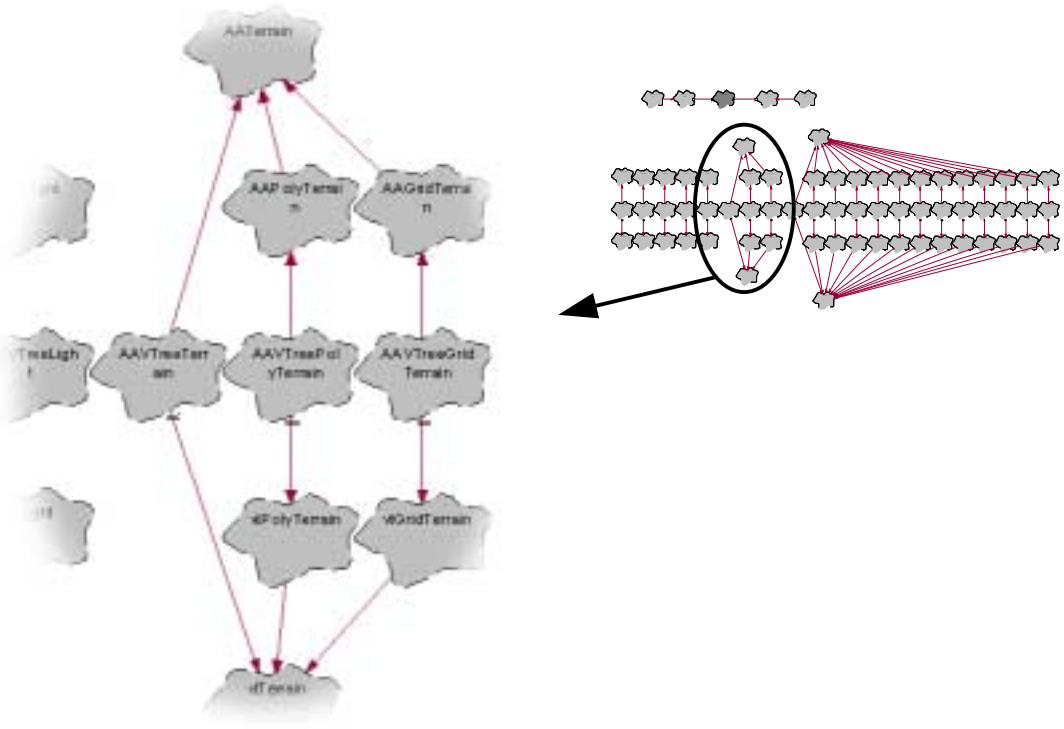

Figure 5.4 - Hierarchy

When multiple inheritance is applied in a system, caution must be taken so that a class does not inherit the same class twice. If such a case appears the keyword virtual is used when the class is inherited the first time. This tells the system to overwrite it the second time when it is inherited.

And finally the bridge pattern requires some designing work but it is nothing crucial. The implementation has nothing unusual but it must follow the prescribed path through the factory pattern as shown in Figure 4.6.

5.2 Plug-in handling

The DLLs have four points of entry. The first is a function that returns the type of the DLL; currently there are six different types. Secondly comes the initiation procedure, where all the instantiations and configuration are placed. The remaining two procedures are used during the execution of the program. The execution is a loop that calls all the DLLs at each turn of the loop *(*Figure 5.5*)*. Simulated events and concurrent actions are placed here. If an event occurs, like a mouse- or keyboard-event, the DLLs is called, so that the DLL/DLLs can take the proper actions.

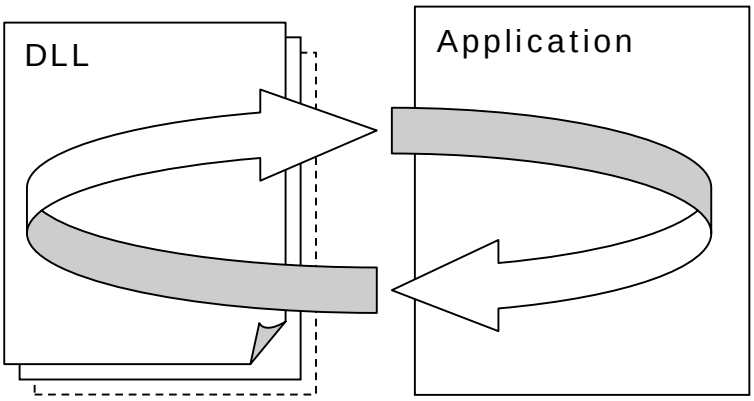

Figure 5.5 - Execution loop

At the moment, a simple handler has been implemented to handle all the DLLs. It searches in a folder called system, and if it finds any DLLs it puts them in a list. This list is traversed every time when the DLLs are used. This handler can be equipped with a graphic interface and let the user choose which components and execution order are to be used for the session. It can also be modified to unload and load DLLs in runtime. The current implementation is very basic and offers no selection or order of execution.

5.3 Demo implementation

This project includes a small demo version with limited functionality and only a fraction of this system is implemented. DLLs are implemented as examples to show how they are made and what functionality could be implemented in them. The visual impression is very good because the frame rate is preserved and visual effects as fog is used (Figure 5.6). The demo application represents an aeroplane that can be controlled with the arrow-buttons. Each part resides in a DLL, which means that the plane has a DLL, the terrain has a DLL etc. etc. The code for this demo implementation is attached in appendix C. It must be added that the pur-

pose of this project is not to develop a fully functional simulation or visualization system, but to do a preliminary design and to show that the design works satisfactorily.

Figure 5.6 - Result

6 Experiences and recommendations

The experiences gained through this project are rather comprehensive and may be somewhat basic. The work involved in design is more time consuming than expected. Translating thoughts and design ideas to code is very delicate. Sometimes the whole design needs to be reviewed and redesigned because there surfaces new and unexpected problems and aspects. An experienced software designer has often better view over problems and knows from the beginning where the problems will arise and can easily avoid redesigning the system. This leads to patterns and framework.

Patterns are useful at the design of a system and should be used both by experienced and inexperienced designers. The solutions that patterns provide are not very complex to understand or to implement as a framework. The patterns are simple, tested and well-documented and solve rather difficult problems. The patterns do not demand any special coding solutions that cannot be done in a standard object-oriented language. Patterns also provide links to other patterns and make perfect reusable components. The only disadvantage is that the design patterns are often not the most effective solution in all situations; they can render in excessive programming and inefficient code.

DLLs have certain pros and cons. The DLLs encapsulate implementation and makes the system more modular, seen from the outside. The DLLs can save space in memory because several applications and processes can use them at the same time and some DLLs can also be loaded and unloaded at any time. On the other hand DLLs, especially the explicit dynamically linked DLLs, creates more overheads because the way they need to be handled. Shared and not shared memory can be somewhat confusing, but there are ways to solve most of the problems.

Choosing suitable APIs and Software kits are not easy. Choosing a packet from a well-known company can prove to be the best solution, even though it is more expensive than one from a smaller company. The comparison of several APIs and software solutions leads to the conclusion that many of the functionalities are almost the same in several of the APIs.

Vega and VTree, for example, are very similar but they have certain characteristics of their own in form of GUIs and functionalities. The OpenGL Optimizer combined with Cosmo 3D is on the other hand for free and is developed by SGI. It lacks some functionality in comparison to Vega and VTree but is guaranteed to deliver maximum performance if used properly.

The demo implementation works satisfactorily. The design is efficient and correct. That leads to the conclusion that the steps of designing and refining the design have been successful and the goals of this project have been met.

References

[1] The OpenGL Graphics System: A Specification (Version 1.2.1). Mark Segal, Kurt Akeley. Silicon Graphics, Inc.

[2] Microsoft Developer Network: MSDN Library Visual Studio 6a.

[3] Design Patterns: Elements of Reusable Object-Oriented Software, Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.

A Glossary

3D, 3-dimensional

API, Application Programming Interface

Bamse, an anti-aircraft defence system that SAAB Bofors Dynamics develops

Booch, A notation used when describing relationships in an object-oriented way, similar to UML.

CAD, Computer Aided Design

Direct3D, a 3D graphics API developed by Microsoft

Directed acyclic graph, a graph, e.g. a scene graph, that is directed, i.e. it's nodes direct to order of the graph, and acyclic, i.e. there cannot be cycles in the node structure

Framebuffer, The buffer that the picture is rendered to before it is flushed to the screen.

Framework, The framework is implementation of pattern using object-oriented language. Using framework means customization of behavior according to framework level of abstraction.

GUI, Graphical User Interface

LOD, Level Of Detail is a way to optimize the viewing of large data sets. By using a simplified model when it is seen from a great distance or at high speed.

Module definition file, An interface description for an explicitly dynamically linked DLL.

OpenGL, *Open Graphics Library*, low-level graphics API developed by SGI. Industry standard used in several CAD and 3D applications

Pattern, Pattern is an abstract description of a problem solution expressed using designed object model. The object model consists of relationships between classes, relationships between objects and relationships between classes and objects.

Render, to generate an image, from a 3D scene, that will be displayed on a 2D screen

Scene graph, a hierarchical representation of a 3D graphics scene

SDK, Software Development Kit

SGI, Silicon Graphics, Inc.

Texturing, Applying a bitmap to a surface to make it more realistic.

Translation, the position of a geometry in a coordinate system; *to translate*, to position a geometry in the coordinate system

Vertex, The smallest component of a 3D scene, consisting of just one point

B Source code for sample program

The following part is the code for a sample program that visualizes a part of the design in a smaller scale.

B.1 Main

```
// classtest.cpp : Defines the entry point for the console application.
//
#include "stdafx.h"
#include "string.h"
#ifndef __factory__
#include "fact.h"
#endif
#ifndef __aclass__
#include "aclass.h"
#endif
#ifndef __anothclass__
#include "anothclass.h"
#endif
/*************************************************************************/
class prog
{
public:
     prog(absFactory* f){factory = f;};
     void exec()
     {
          absClass *someone = factory->makeClass("ola", 3);
          // ABSTRACT INTERFACE CLASS OBJECT IS
          //CREATED AS "ACLASS" OBJECT
          anothAbsClass *another = factory->makeAbsClass();
          // ANOTHER ABSTRACT CLASS OBJECT IS
          //CREATED AS "ANOTCLASS" OBJECT
          another->useme(someone);  // SHOWS HOW THE INTEFACE CLASS OBJECTS
                                    // USE EACHOTHER ALTHOUGHT THE
                                    // IMPLEMENTATION OF THE INTERFACE
                                    // DOESNT WANT A OBJECT OF THAT TYPE.
     };
private:
     absFactory* factory;
};
/*************************************************************************/
int main(int argc, char* argv[])
{
     prog program(new aFactory());
     program.exec();
     return 0;
}
```

B.2 Factory

```
#define __factory__
#ifndef __aclass__
#include "aclass.h"
#endif
#ifndef __anothclass__
#include "anothclass.h"
#endif
/***************************/
class absFactory
{
public:
     virtual absClass* makeClass(char* string, int length) = 0;
     virtual anothAbsClass* makeAbsClass() = 0;
};
/***************************/
class aFactory : public absFactory
{
public:
     absClass* makeClass(char* string, int length)
     {
          return new aClass(string, length);
     };
     anothAbsClass* makeAbsClass()
     {
          return new anothClass();
     };
};
```

B.3 Adapter 1

```
#define __aclass__
/************************/
class absClass                                  //THE ABSTRACT INTEFACE CLASS
{
public:
     virtual void pName(char* string) = 0;
};
/************************/
class anImp                                     // IMPLEMENTOR CLASS
{
public:
     anImp(char* string, int len)
     {
          length = len;
          name = string;
     };
     void printName(char* string)
     {
          printf("Name %s: %s\n", string, name);
     };
private:
     int length;
     char* name;
};
/************************/
class aClass : private anImp, public absClass  // THE IMPLEMENTEE CLASS
{
public:
     aClass(char *s, int l) : anImp(s, l){};
     void pName(char* string)
     {
          printName(string);
     };
private:
};
```

B.4 Adapter 2

```
#define __anothclass__
/*********************************/
class anothAbsClass                                    // ANOTHER ABSTRACT INTERFACE
CLASS
{
public:
     virtual void useme(absClass* a) = 0;
};
/*********************************/
class anothImpClass                                    //ANOTHER IMPLEMENTOR CLASS
{
public:
     void use(anImp* i)
     {    imp = i;
          imp->printName("someone");
     };
private:
     anImp* imp;
};
/*********************************/
class anothClass : private anothImpClass, public anothAbsClass
// ANOTHER IMPLEMENTEE CLASS
{
public:
     void useme(absClass* something)
     {
          use((anImp*)((aClass*) something));
     };
};
```

C Source code for the demo program

This is the code for the demo implementation.

C.1 WinMain

```
// VTClassTest.cpp : Defines the entry point for the application.
//
#include "stdafx.h"
#include "windows.h"

#include <gl\gl.h>                  // Header File For The OpenGL32 Library
#include <gl\glu.h>                 // Header File For The GLu32 Library
#include <gl\glaux.h>               // Header File For The GLaux Library

#ifndef __AAAbsFactory__
#include "AAAbsFactory.h"
#endif
#ifndef __AAVTreeFactory__
#include "AAVTreeFactory.h"
#endif
#ifndef __AADllHandler__
#include "AADllHandler.h"
#endif
#ifndef __AAAbsDllHandler__
#include "AAAbsDllHandler.h"
#endif
#ifndef __AAAbsMainApp__
#include "AAAbsMainApp.h"
#endif
#ifndef __AAMainApp__
#include "AAMainApp.h"
#endif

// Global Variables:
static  HWND hWnd;              // The Window Handle
static    HGLRC hRC;            // Permanent Rendering Context
static    HDC hDC;              // Private GDI Device Context

AAAbsFactory *factory;
AAAbsDllHandler *dllhandler;
AAAbsMainApp *app;


/*****************************************************************************/
/*The Initialization Of Interfaces, Windows And DLLs                         */
/*****************************************************************************/
GLvoid InitGL(GLsizei Width, GLsizei Height) // This Will Be Called Right After The
GL Window Is Created
{
     factory = new AAVTreeFactory();
     dllhandler = new AADllHandler();
     app = new AAMainApp(factory, dllhandler, hWnd);
}

/*****************************************************************************/
/*The Draw Funtion
                */
/*****************************************************************************/
GLvoid DrawGLScene(GLvoid)          //This Will Be Called Each Loop
{
     if (!(app->AADraw())) PostQuitMessage(0);
}

/*****************************************************************************/
/*The Event Handler Funtion                                                  */
/*****************************************************************************/
GLvoid HandleMessage(MSG msg)       // This Will Be Called At Every Event
{
```

```
	switch(msg.message)
	{
	case WM_SIZE:
		break;
	case WM_LBUTTONDOWN:
		break;
	case WM_NCLBUTTONUP:
	case WM_LBUTTONUP:
		break;
	case WM_MBUTTONDOWN:
		break;
	case WM_NCMBUTTONUP:
	case WM_MBUTTONUP:
		break;
	case WM_RBUTTONDOWN:
		break;
	case WM_NCRBUTTONUP:
	case WM_RBUTTONUP:
		break;
	case WM_NCMOUSEMOVE:
	case WM_MOUSEMOVE:
		break;
	case WM_LBUTTONDBLCLK:
		break;
	case WM_MBUTTONDBLCLK:
		break;
	case WM_RBUTTONDBLCLK:
		break;
	case WM_KEYDOWN:
		switch(msg.wParam)
		{
		case VK_ESCAPE:
			PostQuitMessage(0);
			break;
		case VK_RIGHT:
			break;
		case VK_LEFT:
			break;
		case VK_UP:
			break;
		case VK_DOWN:
			break;
		}
		break;
	case WM_KEYUP:
		switch(msg.wParam)
		{
		case VK_RIGHT:
			break;
		case VK_LEFT:
			break;
		case VK_UP:
			break;
		case VK_DOWN:
			break;
		}
		break;
	default:
		break;
	}
	app->AAHandle(msg);
}

/*************************************************************************/
/*The WndProc Callback Routine                                           */
/*************************************************************************/
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
```

```
     GLuint     PixelFormat;
     static     PIXELFORMATDESCRIPTOR pfd=
     {
          sizeof(PIXELFORMATDESCRIPTOR),       // Size Of This Pixel Format De-
scriptor
          1,                                        // Version Number (?)
          PFD_DRAW_TO_WINDOW |                 // Format Must Support Window
          PFD_SUPPORT_OPENGL |                 // Format Must Support OpenGL
          PFD_DOUBLEBUFFER,                         // Must Support Double Buffer-
ing
          PFD_TYPE_RGBA,                            // Request An RGBA Format
          16,                                       // Select A 16Bit Color
Depth
          0, 0, 0, 0, 0, 0,                         // Color Bits Ignored (?)
          0,                                        // No Alpha Buffer
          0,                                        // Shift Bit Ignored (?)
          0,                                        // No Accumulation Buffer
          0, 0, 0, 0,                               // Accumulation Bits Ig-
nored (?)
          16,                                       // 16Bit Z-Buffer (Depth
Buffer)
          0,                                        // No Stencil Buffer
          0,                                        // No Auxiliary Buffer (?)
          PFD_MAIN_PLANE,                           // Main Drawing Layer
          0,                                        // Reserved (?)
          0, 0, 0                                   // Layer Masks Ignored (?)
     };

     switch (message)                               // Tells Windows We Want To
Check The Message
     {
          case WM_CREATE:
               hDC = GetDC(hWnd);         // Gets A Device Context For The Window
               PixelFormat = ChoosePixelFormat(hDC, &pfd);// Finds The Closest
Match To The Pixel Format We Set Above

               if (!PixelFormat)
               {
                    MessageBox(0,"Can't Find A Suitable PixelFor-
mat.","Error",MB_OK|MB_ICONERROR);
                    PostQuitMessage(0);       // This Sends A 'Message' Telling
The Program To Quit
                    break;                    // Prevents The Rest Of The Code
From Running
               }

               if(!SetPixelFormat(hDC,PixelFormat,&pfd))
               {
                    MessageBox(0,"Can't Set The PixelFor-
mat.","Error",MB_OK|MB_ICONERROR);
                    PostQuitMessage(0);       // This Sends A 'Message' Telling
The Program To Quit
                    break;                    // Prevents The Rest Of The Code
From Running
               }

               hRC = wglCreateContext(hDC);
               if(!hRC)
               {
                    MessageBox(0,"Can't Create A GL Rendering Con-
text.","Error",MB_OK|MB_ICONERROR);
                    PostQuitMessage(0);       // This Sends A 'Message' Telling
The Program To Quit
                    break;                    // Prevents The Rest Of The Code
From Running
               }

               if(!wglMakeCurrent(hDC, hRC))
```

```
               {
                    MessageBox(0,"Can't activate
GLRC.","Error",MB_OK|MB_ICONERROR);
                    PostQuitMessage(0);       // This Sends A 'Message' Telling
The Program To Quit
                    break;                    // Prevents The Rest Of The Code
From Running
               }
               break;                         // Prevents The Rest Of The Code
From Running

          case WM_DESTROY:
          case WM_CLOSE:
               ChangeDisplaySettings(NULL, 0); // Nulls The Display Settings
               wglMakeCurrent(hDC,NULL);       // Nulls The Current Hardeware De-
vice Context
               wglDeleteContext(hRC);          // Deletes The hRC
               ReleaseDC(hWnd,hDC);            // Releases The hDC From The Window
               PostQuitMessage(0);             // This Sends A 'Message' Telling
The Program To Quit
               break;

          default:
               return (DefWindowProc(hWnd, message, wParam, lParam));
     }
return (0);
}

/*************************************************************************/
/*WinMain                                                                */
/*************************************************************************/
int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine,
int nCmdShow)
{
     RECT      Screen;         // Used Later On To Get The Size Of The Window
     MSG            msg;       // Windows Message Structure
     WNDCLASS  wc;             // Windows Class Structure Used To Set Up The Type
Of Window

     wc.style             = CS_HREDRAW | CS_VREDRAW | CS_OWNDC;
     wc.lpfnWndProc       = (WNDPROC) WndProc;
     wc.cbClsExtra        = 0;
     wc.cbWndExtra        = 0;
     wc.hInstance         = hInstance;
     wc.hIcon             = NULL;
     wc.hCursor           = LoadCursor(NULL, IDC_ARROW);
     wc.hbrBackground     = NULL;
     wc.lpszMenuName      = NULL;
     wc.lpszClassName     = "OpenGL WinClass";

     if(!RegisterClass(&wc))
     {
          MessageBox(0,"Failed To Register The Window
Class.","Error",MB_OK|MB_ICONERROR);
          return FALSE;
     }

     hWnd = CreateWindow(
     "OpenGL WinClass",
     "The Pure Application.",// Title Appearing At The Top Of The Window

     //WS_OVERLAPPEDWINDOW |   // Creates an overlapped window.
     //WS_OVERLAPPED |         // An overlapped window usually has a caption and a
border.
     WS_POPUP |                // Creates a pop-up window.
     WS_CLIPCHILDREN |         // Excludes the area occupied by child windows when
you draw within the parent window.
     WS_CLIPSIBLINGS |         // Clips child windows relative to each other.
```

```
     WS_VISIBLE,               // Creates a window that is initially visible.

     0, 0,                     // The Position Of The Window On The Screen
     1024, 768,                // The Width And Height Of The WIndow

     NULL,
     NULL,
     hInstance,
     NULL);

     if(!hWnd)
     {
          MessageBox(0,"Window Creation Error.","Error",MB_OK|MB_ICONERROR);
          return FALSE;
     }

     DEVMODE dmScreenSettings;                     // Developer Mode

     memset(&dmScreenSettings, 0, sizeof(DEVMODE)); // Clear Room To Store Settings
     dmScreenSettings.dmSize        = sizeof(DEVMODE);   // Size Of The Devmode
Structure
     dmScreenSettings.dmPelsWidth   = 1024;              // Screen Width
     dmScreenSettings.dmPelsHeight  = 768;               // Screen Height
     dmScreenSettings.dmFields      = DM_PELSWIDTH | DM_PELSHEIGHT; // Pixel Mode

     ShowWindow(hWnd, SW_SHOW);

     GetClientRect(hWnd, &Screen);
     InitGL(Screen.right, Screen.bottom);

     while (1)
     {
          while (PeekMessage(&msg, NULL, 0, 0, PM_NOREMOVE))  // Process All Mes-
sages
          {
               if (GetMessage(&msg, NULL, 0, 0))
               {
                    HandleMessage(msg);
     // Call The Handler Procedure
                    TranslateMessage(&msg);
                    DispatchMessage(&msg);
               }
               else
               {
                    return TRUE;
               }
          }
          DrawGLScene();
     // Call The Draw Procedure
     }
}
```

C.2 Abstract Main Application Class

```
// AAAbsMainApp.h : Defines the interface of the main application class.
//
#define __AAAbsMainApp__

#ifndef __AAAbsFactory__
#include "AAAbsFactory.h"
#endif
#ifndef __AAAbsDllHandler__
#include "AAAbsDllHandler.h"
#endif

class AAAbsMainApp
{
public:
   virtual BOOL AADraw() = 0;
   virtual void AAHandle(MSG) = 0;
private:
};
```

C.3 Main Application Class

```
// AAMainApp.h : Defines the main application class.
//
#define __AAMainApp__

#ifndef __AAAbsMainApp__
#include "AAAbsMainApp.h"
#endif
#ifndef __AAAbsFactory__
#include "AAAbsFactory.h"
#endif
#ifndef __AAAbsDllHandler__
#include "AAAbsDllHandler.h"
#endif

class AAMainApp : public AAAbsMainApp
{
public:
     AAMainApp(AAAbsFactory* fact, AAAbsDllHandler* dlls, HWND hWnd)
     {
          dllhandler = dlls;
          factory = fact;
          dllhandler->AALoadDlls();
          dllhandler->AASetupDlls(factory, hWnd);

     };

     BOOL AADraw()
     {
          dllhandler->AAExecDlls();
          return TRUE;
     };

     void AAHandle(MSG msg)
     {
          dllhandler->AAEventDlls(msg);
     };

private:
     AAAbsDllHandler* dllhandler;
     AAAbsFactory* factory;


     AAWindow* window;
};
```

C.4 Abstract DLL Handler

```
// AAAbsDllHandler.h : Defines the interface of the dll handler class.
//
#define __AAAbsDllHandler__
#include <windows.h>
#ifndef __AAAbsFactory__
#include "AAAbsFactory.h"
#endif

class AAAbsDllHandler
{
public:
     virtual void AALoadDlls() = 0;
     virtual void AASetupDlls(AAAbsFactory*, HWND) = 0;
     virtual void AAExecDlls() = 0;
     virtual void AAEventDlls(MSG msg) = 0;

};
```

C.5 DLL Handler

```
// AADllHandler.h : Defines dll handler class.
//
#define __AADllHandler__
#ifndef __AAAbsDllHandler__
#include "AAAbsDllHandler.h"
#endif
#ifndef __AAAbsFactory__
#include "AAAbsFactory.h"
#endif
#include <stdio.h>
#include <windows.h>

//DEFINITIONS
#define SEARCHSTRING ".\\system\\"
#define SEARCHNAME "*.dll"
#define MAXDLLS 40

class AADllHandler : public AAAbsDllHandler
{
     public:
          //VARIABLES

          //FUNCTIONS
          //Basic fuctions, constructor, destructor.
          AADllHandler();
          ~AADllHandler();

          void AALoadDlls();
          void AASetupDlls(AAAbsFactory* fact, HWND nWnd);
          void AAExecDlls();
          void AAEventDlls(MSG msg);


     private:
          //VARIABLES
          LPWIN32_FIND_DATA dllList[MAXDLLS];
          int nrOfDlls;
          HINSTANCE list[MAXDLLS];
          int dllCount;

          char *buff;
          //FUNCTIONS
          void AASearchForDlls();
};
```

```
// AADllHandler.cpp : Defines implementation of dll handler class.
//
#include "AADllHandler.h"

/******************PUBLIC FUNCTIONS***************/
AADllHandler::AADllHandler()
{
     nrOfDlls = 0;
     buff = new char[MAX_PATH];
     AASearchForDlls();
}

AADllHandler::~AADllHandler()
{
     for (int j = 0 ; j < nrOfDlls ; j++)
     {
          delete dllList[j];
     }
     for (j = 0 ; j < dllCount ; j++)
     {
          FreeLibrary(list[j]);
     }
}

void AADllHandler::AALoadDlls()
{
     dllCount = 0;

     for (int j = 0 ; j < nrOfDlls ; j++)
     {
          sprintf(buff, "%s%s", SEARCHSTRING, dllList[j]->cFileName);
          list[dllCount] = LoadLibrary(buff);
          if (list[dllCount] != NULL)
          {
               FARPROC libfunk = GetProcAddress(list[dllCount], "ddKindOfDll");
               if (libfunk != NULL)
               {
                    dllCount++;
               }
               else
               {
                    FreeLibrary(list[dllCount]);
               }
          }
     }
}

void AADllHandler::AASetupDlls(AAAbsFactory* fact, HWND hWnd)
{
     typedef void (* funk)(AAAbsFactory*, HWND);

     for (int j = 0 ; j < dllCount ; j++)
     {
          if (list[j] != NULL)
          {
               funk libfunk = (funk)GetProcAddress(list[j], "ddSetupDll");
               if (libfunk != NULL)
               {
                    libfunk(fact, hWnd);
               }
               else
               {
                    FreeLibrary(list[j]);
                    MessageBox(0,"Error in executing the
dll.","Error",MB_OK|MB_ICONERROR);

               }
          }
```

```
     }
}

void AADllHandler::AAExecDlls()
{
     for (int j = 0 ; j < dllCount ; j++)
     {
          if (list[j] != NULL)
          {
               FARPROC libfunk = GetProcAddress(list[j], "ddExecDll");
               if (libfunk != NULL)
               {
                    libfunk();
               }
               else
               {
                    FreeLibrary(list[j]);
                    MessageBox(0,"Error in executing the
dll.","Error",MB_OK|MB_ICONERROR);
               }
          }
     }
}

void AADllHandler::AAEventDlls(MSG msg)
{
     typedef void (* funk)(MSG); //STYLE OF TYPEDEFING RETURN FUNCTION.

     for (int j = 0 ; j < dllCount ; j++)
     {
          if (list[j] != NULL)
          {
               funk libfunk = (funk)GetProcAddress(list[j], "ddEventDll");
               if (libfunk != NULL)
               {
                    libfunk(msg);
               }
               else
               {
                    FreeLibrary(list[j]);
                    MessageBox(0,"Error in executing the
dll.","Error",MB_OK|MB_ICONERROR);
               }
          }
     }
}

/***************PRIVATE FUNCTIONS*****************/
void AADllHandler::AASearchForDlls()
{
     dllList[nrOfDlls] = new WIN32_FIND_DATA;
     sprintf(buff, "%s%s", SEARCHSTRING, SEARCHNAME);
     HANDLE foundFile = FindFirstFile(buff, dllList[0]); //The filename is now in -
>cFileName of the list.
     if (foundFile == INVALID_HANDLE_VALUE)
     {
          MessageBox(0,"No *.dll files found.","Error",MB_OK|MB_ICONERROR);
     }
     else
     {
          nrOfDlls = 1;
          dllList[nrOfDlls] = new WIN32_FIND_DATA;
          while (FindNextFile(foundFile, dllList[nrOfDlls]) && (nrOfDlls < 40))
          {
               nrOfDlls++;
               dllList[nrOfDlls] = new WIN32_FIND_DATA;
          }
          delete dllList[nrOfDlls];
```

```
            for (int j = 0 ; j < nrOfDlls ; j++)
            {
                 printf("%s \n", dllList[j]->cFileName);
            }
      }
}
```

C.6 Main DLL

```
;mainDll.def : Defines the interface for the DLL
LIBRARY maniDll
EXPORTS

          ddKindOfDll
          ddSetupDll
          ddExecDll
          ddEventDll



// mainDll.cpp : Defines the entry point for the DLL application.
//
#include "stdafx.h"
#include "dlls.h"

#ifndef __AAAbsFactory__
#include "AAAbsFactory.h"
#endif
#ifndef __AAWindow__
#include "AAWindow.h"
#endif
#ifndef __AAViewport__
#include "AAViewport.h"
#endif
#ifndef __AAScene__
#include "AAScene.h"
#endif
#ifndef __AAEntity__
#include "AAEntity.h"
#endif
#ifndef __AANode__
#include "AANode.h"
#endif

//VARS AND CLASSES
AAAbsFactory* factory;
AAWindow* window;
AAViewport* viewport;
AAScene* scene;
AAEntity* entity;
float deltaHorz, deltaVert, deltaRoll, dist, temp1, temp2, temp3, temp4;
BOOL LEFT_B, FIRST_LEFT;

BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                            )
{
    return TRUE;
}

int ddKindOfDll(void)
{
     return APPDLL;
}

void ddSetupDll(AAAbsFactory* fact, HWND hWnd)
{
     factory = fact;

     window = factory->makeWindow("The Main Window", hWnd, 0, 0, 1024, 768);
```

```
	viewport = factory->makeViewport(window, "The Main Viewport", 0, 0, 1024,
768);
	window->AAAddViewport(viewport);

	scene = factory->makeScene("The Main Scene");
	entity = factory->makeEntity("The Main Entity");
	entity = scene->AALoad(".\\models\\f16.vt");

	viewport->AASetScene(scene);
	viewport->AASetColor(0.87f, 0.87f, 1.0f);
	viewport->AASetPerspective(45.0f, 1.5f,  1.f, 10000.f);
	//viewport->AASetViewLookFromTo(-20.0f, 10.0f, -5.0f, 0.0f, 0.0f, 0.0f);
	viewport->AAFogSetLinearRange(300.0f, 10000.0f);
	viewport->AAFogSetMode(LINEAR);
	viewport->AAFogSetColor(0.9f, 0.9f, 1.0f, 0.3f);
	viewport->AAFogEnable();

	window->AAEnable(TEXTURE);
	//window->AAEnable(LIGHTNING);
	window->AAEnable(BACKFACECULL);
	window->AAEnable(OBJECTCULL);
	window->AAEnable(FOG);

	viewport->AAPrepareForRender();

	dist = 20.0f;
	deltaHorz = 0.f;
	deltaVert = 0.f;
	deltaRoll = 0.f;
	LEFT_B = FALSE;
}

void ddExecDll()
{
	viewport->AASetViewLookAroundAt(factory->makeVector(0.f, 0.f, 0.f), deltaHorz,
deltaVert, deltaRoll, dist);
	window->AAFrame();
	window->AARender();
	deltaVert = deltaHorz = 0.f;

}

void ddEventDll(MSG msg)
{
	switch(msg.message)
	{
	case WM_LBUTTONDOWN:
		LEFT_B = TRUE;
		FIRST_LEFT = TRUE;
		break;
	case WM_NCLBUTTONUP:
	case WM_LBUTTONUP:
		LEFT_B = FALSE;
		break;
	case WM_MBUTTONDOWN:
		break;
	case WM_NCMBUTTONUP:
	case WM_MBUTTONUP:
		break;
	case WM_RBUTTONDOWN:
		break;
	case WM_NCRBUTTONUP:
	case WM_RBUTTONUP:
		break;
	case WM_NCMOUSEMOVE:
	case WM_MOUSEMOVE:
		if (LEFT_B)
		{
```

```
                temp1 = LOWORD(msg.lParam);
                temp3 = HIWORD(msg.lParam);
                if (FIRST_LEFT)
                {
                     temp4 = temp3;
                     temp2 = temp1;
                     FIRST_LEFT = FALSE;
                }
                /*THE HORIZONTAL*/
                temp2 = temp2 - temp1;
                deltaHorz = temp2/500;
                temp2 = temp1;
                /*THE VERTICAL*/
                temp4 = temp4 - temp3;
                deltaVert = temp4/500;
                temp4 = temp3;
           }
           break;
      case WM_LBUTTONDBLCLK:
           break;
      case WM_MBUTTONDBLCLK:
           break;
      case WM_RBUTTONDBLCLK:
           break;
      case WM_KEYDOWN:
           switch(msg.wParam)
           {
           case VK_RIGHT:
                break;
           case VK_LEFT:
                break;
           case VK_UP:
                break;
           case VK_DOWN:
                break;
           }
           break;
      case WM_KEYUP:
           switch(msg.wParam)
           {
           case VK_RIGHT:
                break;
           case VK_LEFT:
                break;
           case VK_UP:
                break;
           case VK_DOWN:
                break;
           }
           break;
      default:
           break;
      }
}
```

C.7 Terrain DLL

```
;mainDll.def : Defines the interface for the DLL
LIBRARY terrainDll
EXPORTS

          ddKindOfDll
          ddSetupDll
          ddExecDll
          ddEventDll



// terrainDLL.cpp : Defines the entry point for the DLL application.
//
#include "stdafx.h"
#include "dlls.h"

#ifndef __AAAbsFactory__
#include "AAAbsFactory.h"
#endif
#ifndef __AAWindow__
#include "AAWindow.h"
#endif
#ifndef __AAViewport__
#include "AAViewport.h"
#endif
#ifndef __AAScene__
#include "AAScene.h"
#endif
#ifndef __AAEntity__
#include "AAEntity.h"
#endif
#ifndef __AANode__
#include "AANode.h"
#endif
#ifndef __AAGMatrix__
#include "AAGMatrix.h"
#endif

//VARS AND CLASSES
AAAbsFactory* factory;
AAWindow* window;

BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                            )
{
    return TRUE;
}

int ddKindOfDll(void)
{
     return TERRAINDLL;
}

void ddSetupDll(AAAbsFactory* fact, HWND hWnd)
{
     factory = fact;

     window = factory->makeWindow("The Main Window", hWnd, 0, 0, 1024, 768);

     AAViewport* viewport = window->AAFindViewport("The Main Viewport");
     AAScene* scene = viewport->AAGetScene();
```

```
	AANode* graph = factory->makeNode("LosAngeles");
	graph = graph->AAReadTree(".\\terrain\\losangeles2.vt");
	AAEntity* entity = factory->makeEntity("Terrain", graph);

	scene->AAAddEntity(entity);

	viewport->AAPrepareForRender();
}

void ddExecDll()
{

}

void ddEventDll(MSG msg)
{
	switch(msg.message)
	{
	case WM_LBUTTONDOWN:
		break;
	case WM_NCLBUTTONUP:
	case WM_LBUTTONUP:
		break;
	case WM_MBUTTONDOWN:
		break;
	case WM_NCMBUTTONUP:
	case WM_MBUTTONUP:
		break;
	case WM_RBUTTONDOWN:
		break;
	case WM_NCRBUTTONUP:
	case WM_RBUTTONUP:
		break;
	case WM_NCMOUSEMOVE:
	case WM_MOUSEMOVE:
		break;
	case WM_LBUTTONDBLCLK:
		break;
	case WM_MBUTTONDBLCLK:
		break;
	case WM_RBUTTONDBLCLK:
		break;
	case WM_KEYDOWN:
		switch(msg.wParam)
		{
		case VK_RIGHT:
			break;
		case VK_LEFT:
			break;
		case VK_UP:
			break;
		case VK_DOWN:
			break;
		}
		break;
	case WM_KEYUP:
		switch(msg.wParam)
		{
		case VK_RIGHT:
			break;
		case VK_LEFT:
			break;
		case VK_UP:
			break;
		case VK_DOWN:
			break;
		}
```

```
            break;
      default:
            break;
      }
}
```

C.8 Abstract Factory

```
// AAAbsFactory.h : Defines the interface of the factory class.
//
#define __AAAbsFactory__
#ifndef __AAWindow__
#include "AAWindow.h"
#endif
#ifndef __AAViewport__
#include "AAViewport.h"
#endif
#ifndef __AAScene__
#include "AAScene.h"
#endif
#ifndef __AAEntity__
#include "AAEntity.h"
#endif
#ifndef __AANode__
#include "AANode.h"
#endif
#ifndef __AATNode__
#include "AATNode.h"
#endif
#ifndef __AAIGNode__
#include "AAIGNode.h"
#endif
#ifndef __AAVector__
#include "AAVector.h"
#endif
#ifndef __AAGMatrix__
#include "AAGMatrix.h"
#endif
#ifndef __AAColor__
#include "AAColor.h"
#endif
#ifndef __AAMaterial__
#include "AAMaterial.h"
#endif
#ifndef __AAFxDebris__
#include "AAFxDebris.h"
#endif
#ifndef __AAFxMissileTrail__
#include "AAFxMissileTrail.h"
#endif
#ifndef __AAFxExplosion__
#include "AAFxExplosion.h"
#endif

class AAAbsFactory
{
public:
     virtual AAWindow* makeWindow(char* string, HWND hWnd, int posx, int posy, int
width, int height) = 0;
     virtual AAViewport* makeViewport(AAWindow* window, char* string, int posx, int
posy, int width, int height) = 0;
     virtual AAScene* makeScene(char* string) = 0;
     virtual AAEntity* makeEntity(char* string, AANode* tree = NULL) = 0;

     virtual AANode* makeNode(char* string, char* path) = 0;
     virtual AATNode* makeTNode(const char *name = NULL) = 0;
     virtual AAIGNode* makeIGNode(const char *name = NULL,
               AANode *next = NULL,
               AANode *child = NULL,
               int ColorType = AAPerNone,
               int NormType = AAPerNone,
```

```
                int TexType = AAPerNone,
                int VertType = AAPerNone,
                unsigned int NodeFlags = 0,
                int Material = 0,
                int Texture = 0,
                BOOL Wireframe = FALSE,
                int NumPrimSets = 0,
                int *PrimTypes = NULL,
                int *NumPrimVerts = NULL,
                int *NumPrims = NULL,
                int NumColors = 0,
                unsigned int *Colors = NULL,
                int NumNorms = 0,
                float *Norms = NULL,
                int NumTexs = 0,
                float *Texs = NULL,
                int NumVerts = 0,
                float *Verts = NULL,
                int NumColorIndex = 0,
                int *ColorIndex = NULL,
                int NumNormIndex = 0,
                int *NormIndex = NULL,
                int NumTexIndex = 0,
                int *TexIndex = NULL,
                int NumVertIndex = 0,
                int *VertIndex = NULL) = 0;

     virtual AAVector* makeVector() = 0;
     virtual AAVector* makeVector(float, float, float) = 0;
     virtual AAGMatrix* makeGMatrix() = 0;

     virtual AAColor* makeColor(float ir = 0.0f, float ig = 0.0f, float ib = 0.0f,
float ia = 1.0f) = 0;
     virtual AAColor* makeColor(AAColor *c) = 0;
     virtual AAColor* makeColor(AAPackColor *p) = 0;
     virtual AAPackColor* makePackColor(const int ir = 0, const int ig = 0, const
int ib = 0, const int ia = 255) = 0;
     virtual AAPackColor* makePackColor(unsigned int c) = 0;
     virtual AAMaterial* makeMaterial(const char *name = NULL,
                                      const float alpha = 1.0,
                                      const AAVector *ambient = NULL,
                                      const AAVector *diffuse = NULL,
                                      const AAVector *specular = NULL,
                                      const AAVector *emission = NULL,
                                      const float shininess = 0.0) = 0;

     //virtual AAGaugeNode* makeGaugeNode(const char *name = NULL, AANode *next =
NULL, AANode *child = NULL, int neededChannels = 0) = 0;
     virtual AA2DGeomElement* make2DGeomElement(AAGeomType2D t, const int numVerts
= 0, float *data = NULL) = 0;
     virtual AA2DGeomGaugeNode* make2DGeomGaugeNode(const char *name = NULL, AANode
*next = NULL, AANode *child = NULL) = 0;
     virtual AAFxDebris* makeFxDebris(const char* name) = 0;
     virtual AAFxMissileTrail* makeFxMissileTrail(const char *name, AAVector *pos,
float dur = 1.0f, float size = 1.0f) = 0;
     virtual AAFxExplosion* makeFxExplosion(char* name = NULL) = 0;
private:

};
```

C.9 VTree Factory

```
// AAVTreeFactory.h : Defines the factory class.
//
#define __AAVTreeFactory__
#ifndef __AAAbsFactory__
#include "AAAbsFactory.h"
#endif
#ifndef __AAVTreeWindow__
#include "AAVTreeWindow.h"
#endif
#ifndef __AAVTreeViewport__
#include "AAVTreeViewport.h"
#endif
#ifndef __AAVTreeScene__
#include "AAVTreeScene.h"
#endif
#ifndef __AAVTreeEntity__
#include "AAVTreeEntity.h"
#endif
#ifndef __AAVTreeNode__
#include "AAVTreeNode.h"
#endif
#ifndef __AAVTreeTNode__
#include "AAVTreeTNode.h"
#endif
#ifndef __AAVTreeIGNode__
#include "AAVTreeIGNode.h"
#endif
#ifndef __AAVTreeVector__
#include "AAVTreeVector.h"
#endif
#ifndef __AAVTreeGMatrix__
#include "AAVTreeGMatrix.h"
#endif
#ifndef __AAVTreeColor__
#include "AAVTreeColor.h"
#endif
#ifndef __AAVTreeMaterial__
#include "AAVTreeMaterial.h"
#endif
#ifndef __AAVTreeFxDebris__
#include "AAVTreeFxDebris.h"
#endif
#ifndef __AAVTreeFxMissileTrail__
#include "AAVTreeFxMissileTrail.h"
#endif
#ifndef __AAVTreeFxExplosion__
#include "AAVTreeFxExplosion.h"
#endif

class AAVTreeFactory : public AAAbsFactory
{
private:
     AAVTreeWindow* window;

public:
     AAVTreeFactory()
     {
          window = NULL;
     };

     AAWindow* makeWindow(char* string, HWND hWnd, int posx, int posy, int width,
int height)//{return NULL;};
     {
```

```
          if (window == NULL)
          {
               vtWinhDC = GetDC(hWnd);                                        // Cap-
tures The Windows Device Context
               vtWinhWnd = hWnd;                                              // Cap-
tures The Window Handle
               vtSetPixelFormat(vtWinhWnd, vtWinhDC);              // Chooses The
pixel Format (Shows A Popup Window The Firs Time)
               vtWinhGLRC = wglCreateContext(vtWinhDC);       // Creates A GL Ren-
dering Context
               wglMakeCurrent(vtWinhDC, vtWinhGLRC);               // Makes The
Rendering Context Current
               vtInit();                                                      // Initi-
ates The Vtree Library
               window = new AAVTreeWindow(string, hWnd, posx, posy, width, height);
          }
          return window;
     };

     AAViewport* makeViewport(AAWindow* wnd, char* string, int posx, int posy, int
width, int height)
     {return new AAVTreeViewport((vtWindow*)((AAVTreeWindow*)wnd), string, posx,
posy, width, height);};

     AAScene* makeScene(char* string)
     {return new AAVTreeScene(string);};

     AAEntity* makeEntity(char* string, AANode* tree)
     {return new AAVTreeEntity(string, (vtNode*)((AAVTreeNode*)tree));};

     AANode* makeNode(char* string, char* path)
     {
          vtNode* node = new vtNode();
          node = vtReadTree(path);
          return new AAVTreeNode(string, node);
     };

     AATNode* makeTNode(const char *name)
     {
          return new AAVTreeTNode(name);
     };

     AAIGNode* makeIGNode(     const char *name = NULL,
          AANode *next = NULL,
          AANode *child = NULL,
          int ColorType = AAPerNone,
          int NormType = AAPerNone,
          int TexType = AAPerNone,
          int VertType = AAPerNone,
          unsigned int NodeFlags = 0,
          int Material = 0,
          int Texture = 0,
          BOOL Wireframe = FALSE,
          int NumPrimSets = 0,
          int *PrimTypes = NULL,
          int *NumPrimVerts = NULL,
          int *NumPrims = NULL,
          int NumColors = 0,
          unsigned int *Colors = NULL,
          int NumNorms = 0,
          float *Norms = NULL,
          int NumTexs = 0,
          float *Texs = NULL,
          int NumVerts = 0,
          float *Verts = NULL,
          int NumColorIndex = 0,
          int *ColorIndex = NULL,
          int NumNormIndex = 0,
```

```
          int *NormIndex = NULL,
          int NumTexIndex = 0,
          int *TexIndex = NULL,
          int NumVertIndex = 0,
          int *VertIndex = NULL)
     {return new AAVTreeIGNode(name, NULL, NULL, ColorType, NormType, TexType, Vert-
Type, NodeFlags,     Material, Texture, Wireframe,
                                        NumPrimSets, PrimTypes,   NumPrimVerts,
NumPrims, NumColors, Colors, NumNorms, Norms, NumTexs, Texs,
                                        NumVerts, Verts, NumColorIndex, ColorIn-
dex, NumNormIndex, NormIndex, NumTexIndex, TexIndex, NumVertIndex,
                                        VertIndex);};

     AAVector* makeVector()
     {return new AAVTreeVector();};

     AAVector* makeVector(float x, float y, float z)
     {return new AAVTreeVector(x, y, z);};

     AAGMatrix* makeGMatrix()
     {return new AAVTreeGMatrix();};

     AAColor* makeColor(float ir = 0.0f, float ig = 0.0f, float ib = 0.0f, float ia
= 1.0f)
     {return new AAVTreeColor(ir, ig, ib, ia);};
     AAColor* makeColor(AAColor *c)
     {return new AAVTreeColor(c);};
     AAColor* makeColor(AAPackColor *p)
     {return new AAVTreeColor(p);};

     AAPackColor* makePackColor(const int ir = 0, const int ig = 0, const int ib =
0, const int ia = 255)
     {return new AAVTreePackColor(ir, ig, ib, ia);};
     AAPackColor* makePackColor(unsigned int c)
     {return new AAVTreePackColor(c);};

     AAMaterial* makeMaterial(const char *name = NULL,
                                  const float alpha = 1.0,
                                  const AAVector *ambient = NULL,
                                  const AAVector *diffuse = NULL,
                                  const AAVector *specular = NULL,
                                  const AAVector *emission = NULL,
                                  const float shininess = 0.0)
     {return new AAVTreeMaterial(name, alpha, ambient, diffuse, specular, emission,
shininess);};

     AA2DGeomElement* make2DGeomElement(AAGeomType2D t, const int numVerts = 0,
float *data = NULL)
     {return new AAVTree2DGeomElement(t, numVerts, data);};
     AA2DGeomGaugeNode* make2DGeomGaugeNode(const char *name = NULL, AANode *next =
NULL, AANode *child = NULL)
     {return new AAVTree2DGeomGaugeNode(name, next, child);};
     //AAGaugeNode* makeGaugeNode(const char *name = NULL, AANode *next = NULL,
AANode *child = NULL, int neededChannels = 0)
     //{return new AAVTreeGaugeNode(name, next, child, neededChannels);};
     AAFxDebris* makeFxDebris(const char* name)
     {return new AAVTreeFxDebris(name);};
     AAFxMissileTrail* makeFxMissileTrail(const char *name, AAVector *pos, float
dur, float size)
     {return new AAVTreeFxMissileTrail(name, pos, dur, size);};
     AAFxExplosion* makeFxExplosion(char* name)
     {return new AAVTreeFxExplosion(name);};
private:

};
```

C.10 The New API (interface)

```
// AAColor.h : Defines the interface of the color class.
//
#define __AAColor__

class AAPackColor;

class AAColor
{
public:
     virtual void AASet(float ir, float ig, float ib, float ia = 1.0f) = 0;
     virtual void AAFromPacked(AAPackColor *pc) = 0;
     virtual float AAr() = 0;
     virtual float AAg() = 0;
     virtual float AAb() = 0;
private:

};


class AAPackColor
{
public:
     virtual void AASet(int ir, int ig, int ib, int ia = 255) = 0;
     virtual void AASet(int c) = 0;
     virtual int AAPacked() = 0;
     virtual int     AAr() = 0;
     virtual int AAg() = 0;
     virtual int AAb() = 0;
private:

};
```

```
// AAEntity.h : Defines the interface of the entity class.
//
#define __AAEntity__

#ifndef __AAFxExplosion__
#include "AAFxExplosion.h"
#endif
#ifndef __AANode__
#include "AANode.h"
#endif
#ifndef __AAVector__
#include "AAVector.h"
#endif
#ifndef __AAGMatrix__
#include "AAGMatrix.h"
#endif
#ifndef __AAFxDebris__
#include "AAFxDebris.h"
#endif
#ifndef __AAFxMissileTrail__
#include "AAFxMissileTrail.h"
#endif


class AAEntity
{
public:
     virtual void AASetDepthCenter(const float, const float, const float) = 0;
     virtual void AASetDepthSorted(const BOOL) = 0;
     virtual void AASetModelToWorld(AAGMatrix*) = 0;
     virtual void AASetPos(const float, const float, const float) = 0;
     virtual void AASetPos(AAVector*) = 0;
     virtual void AASetPosYPR(AAVector*, const float, const float, const float) =
0;
     virtual void AASetPosYPR(const float, const float, const float, const float,
const float, const float) = 0;
     virtual void AASetRot(const float, const float, const float) = 0;
     virtual void AASetTree(AANode* tree) = 0;
     virtual void AASetVisible(BOOL) = 0;
     virtual void AAAddFx(AAFxExplosion*) = 0;
     virtual void AAAddFx(AAFxDebris*) = 0;
     virtual void AAAddFx(AAFxMissileTrail*) = 0;

private:
};
```

```
// AAFx.h : Defines the interface of the fx(special effects) class.
//
#define __AAFx__

#ifndef __AAColor__
#include "AAColor.h"
#endif

class AAFx
{
public:
     virtual void AAStart(float = -1.0f, float = -1.0f) = 0;
     virtual void AAStop(float) = 0;
     virtual void AASetDuration(float) = 0;
     virtual void AASetFadeDuration(float) = 0;
     virtual void AASetColor(AAColor*) = 0;
     virtual BOOL AASetMoveWithParent(const BOOL) = 0;
};
```

```
// AAFxChunk.h : Defines the interface of the fx chunk class.
//
#define __AAFxChunk__

class AAFxChunk
{
public:
     virtual AAColor* AAColor() = 0;
     virtual AAColor* AAColor(const int) = 0;
     virtual AAFxChunk* AADuplicate() = 0;
     virtual float AAGravity() const = 0;
     virtual double AALastTime() = 0;
     virtual AAVector* AAPos() = 0;
     virtual AAVector* AARot() = 0;
     virtual AAVector* AARotRate() = 0;
     virtual BOOL AARunning() const;
     virtual void AASetColor(const unsigned int) = 0;
     virtual void AASetColor(const int, const unsigned int) = 0;
     virtual void AASetGravity(const float) = 0;
     virtual void AASetLastTime(const double) = 0;
     virtual void AASetPos(const AAVector*) = 0;
     virtual void AASetRot(const AAVector*) = 0;
     virtual void AASetRotRate(const AAVector*) = 0;
     virtual void AASetVel(const AAVector*) = 0;
     virtual void AAStart() = 0;
     virtual void AAStop() = 0;
     virtual AAVector* AAVel() = 0;

private:
};
```

```
// AAFxDebris.h : Defines the interface of the fx debris class.
//
#define __AAFxDebris__

#ifndef __AAFx__
#include "AAFx.h"
#endif

class AAFxDebris : public AAFx
{
public:
     virtual float AAChunkSize() = 0;
     virtual int AANumChunks() = 0;
     virtual void AAReset(const BOOL) = 0;
     virtual void AASetChunkSize(const float) = 0;
     virtual void AASetMaxVel(const AAVector*) = 0;
     virtual void AASetMinVel(const AAVector*) = 0;
     virtual void AASetNumChunks(const int, const BOOL) = 0;
     virtual void AASetVel(const AAVector*, const AAVector*) = 0;

private:
};
```

```
// AAFxExplosion.h : Defines the interface of the fx explosion class.
//
#define __AAFxExplosion__

#ifndef __AAFx__
#include "AAFx.h"
#endif

enum AAAnimType {
            AAOneShotType = 1,  // Play the sequence once, or until
                // ...the duration is ended
     AALoopType = 2, // Loop the sequence until the duration ends
};

class AAFxExplosion : public AAFx
{
public:
    // Type methods

    virtual void AASetType(const AAAnimType type) = 0;
     // Sets the animation type (OneShotType, LoopType, etc.)
    virtual AAAnimType AAType() const = 0;
     // Returns the animation type

    // Direction methods

    virtual void AASetDirection(const int dir) = 0;
     // Sets the animation direction (positive and zero means forward,
     // negative means backwards)
    virtual int AADirection() const = 0;
     // Returns the direction

    // HoldLast methods

    virtual void AASetHoldLast(const BOOL hold) = 0;
     // If 'hold' is TRUE, the last frame of the animation will be kept
     // visible (for non looping animations) until the effect duration
     // is exceeded.
    virtual BOOL AAHoldLast() const = 0;
     // Returns TRUE if the last animation frame will be held

    // Frame duration methods

    virtual void AASetFrameDuration(const float dur) = 0;
     // Sets the duration (in seconds) of each frame in the animation
    virtual float AAFrameDuration() const = 0;
     // Returns the frame duration (in seconds) of every frame in
     // the animation

    virtual void AASetDurationAttributes(const BOOL durAttr) = 0;
     // If set to 'TRUE' the attribute trables continue to be
     // interpreted through the end of the effect.  If set to
     // 'FALSE', the attributes stop at the end of the last frame
     // which may not be at the end of the effect.
     // Default is 'FALSE'.
    virtual BOOL AADurationAttributes(void) const = 0;
     // Returns the current value of the m_DurationAttributes flag.

    // Loop methods

    virtual void AASetLoopStart(const int l) = 0;
     // Sets the loop starting frame number (-1 to disable looping)
    virtual int AALoopStart() const = 0;
     // Returns the loop starting frame number
    virtual void AASetLoopEnd(const int l) = 0;
     // Sets the loop ending frame number (-1 to disable looping)
    virtual int AALoopEnd() const = 0;
     // Returns the loop ending frame number
```

```
    virtual void AASetLoop(const int s, const int e) = 0;
     // Sets the loop start and end frames (-1's to disable looping)
    virtual BOOL AAHasLoop() = 0;
     // Returns TRUE if there is an animation loop
    // Table methods
     virtual void AASetSize(float = 50.f, float = 50.f, float = 50.f) = 0;
};
```

```
// AAFxMissileTrail.h : Defines the interface of the fx missile trail class.
//
#define __AAFxMissileTrail__

#ifndef __AAFx__
#include "AAFx.h"
#endif

class AAFxMissileTrail : public AAFx
{
public:
     virtual void AASetAddInterval(const float d) = 0;
     virtual void AASetRotRate(const float dur) = 0;
     virtual void AASetExpandTime(const float dur) = 0;
     virtual void AASetTexScale(const float u, const float v) = 0;
     virtual void AASetGenerate(const BOOL gen) = 0;
     virtual BOOL AAStartTrail(const float t) = 0;
     virtual BOOL AAStopTrail(const float t) = 0;
     virtual BOOL AASetDurationTrail(const float dur) = 0;
     virtual BOOL AASetFadeDurationTrail(const float dur) = 0;
     virtual BOOL AASetNodeDuration(const float dur) = 0;
private:
};
```

```
// AAGMatrix.h : Defines the interface of the G matrix class.
//
#define __AAGMatrix__

#ifndef __AAVector__
#include "AAVector.h"
#endif

class AAGMatrix
{
public:
     virtual void AAArbRotation(float, float, float, float) = 0;
     virtual void AAArbRotation(float, AAVector*) = 0;
     virtual void AABaseRotateArb(float, float, float, float) = 0;
     virtual void AABaseRotateArb(float, AAVector*) = 0;
     virtual void AABaseRotateX(float) = 0;
     virtual void AABaseRotateXYZ(float, float, float) = 0;
     virtual void AABaseRotateXZY(float, float, float) = 0;
     virtual void AABaseRotateY(float) = 0;
     virtual void AABaseRotateYXZ(float, float, float) = 0;
     virtual void AABaseRotateYZX(float, float, float) = 0;
     virtual void AABaseRotateZ(float) = 0;
     virtual void AABaseRotateZXY(float, float, float) = 0;
     virtual void AABaseRotateZYX(float, float, float) = 0;
     virtual void AABaseScale(float, float, float) = 0;
     virtual void AABaseTranslate(float, float, float) = 0;
     virtual void AABaseTranslate(AAVector*) = 0;
     virtual void AAFromPosYPR(AAVector*, float, float, float) = 0;
     virtual void AAFromPosYPR(float, float, float, float, float, float) = 0;
     virtual void AARotateArb(float, float, float, float) = 0;
     virtual void AARotateArb(float, AAVector*) = 0;
     virtual void AARotateX(float) = 0;
     virtual void AARotateXYZ(float, float, float) = 0;
     virtual void AARotateXZY(float, float, float) = 0;
     virtual void AARotateY(float) = 0;
     virtual void AARotateYXZ(float, float, float) = 0;
     virtual void AARotateYZX(float, float, float) = 0;
     virtual void AARotateZ(float) = 0;
     virtual void AARotateZXY(float, float, float) = 0;
     virtual void AARotateZYX(float, float, float) = 0;
     virtual void AAScale(float, float, float) = 0;
     virtual void AASet(AAGMatrix*) = 0;
     virtual void AASub(AAGMatrix*) = 0;
     virtual void AATranslate(AAVector*) = 0;
     virtual void AATranslate(float, float, float) = 0;
     virtual void AATranslation(AAVector*) = 0;
     virtual void AATranslation(float, float, float) = 0;
     virtual void AATranspose() = 0;
protected:
private:
};
```

```
// AAIGNode.h : Defines the interface of the indexed geometry node class.
//
#define __AAIGNode__

#ifndef __AANode__
#include "AANode.h"
#endif

enum AABinding {
         AAPerNone,           // none
         AAPerNode,           // one per node
         AAPerPrim,           // one per prim
         AAPerPoly,           // one per polygon
         AAPerVert};          // one per vertex

enum AAPrimTypes {
          AANullPrim      = 0,  // undefined type
          AAPointPrim     = 1,  // Points
          AALinePrim      = 2,  // Unconnected line segments
          AATriPrim       = 3,  // Unconnected triangles
          AAQuadPrim      = 4,  // Unconnected quadrangles
          AAPolyPrim      = 5,  // Unconnected N-sided
          AALineStripPrim = 6,  // Line strip
          AALineLoopPrim  = 7,  // Line loop
          AATriMeshPrim   = 8,  // Triangle mesh
          AAQuadStripPrim = 9,  // Quadrangle strip
          AATriFanPrim    = 10  // Triangle fan
          };

class AAIGNode : public AANode
{

public:
//    virtual void AACopy(const AAIGNode &r, const BOOL copyBase = TRUE) = 0;
     // Copy method

    // Number of primitives
    virtual int AANumPrimSets() = 0;
     // number of primitive sets
    virtual void AASetNumPrimSets(const int num) = 0;
     // set number of primitive sets

    // Number of polygon vertices methods
    virtual int AANumPrimVerts(const int i) const = 0;
     // number of primitive verts
    virtual void AASetNumPrimVerts(const int i, int n) = 0;
     // set number of primitive verts

    // Primitive type methods
    virtual void AASetPrimTypes(const int num, int *primtypes) = 0;
     // Sets the primitive types using an array
    virtual void AASetPrimType(const int i, int n) = 0;
     // Sets the type of primitive "i" to type "n"
    virtual int AAPrimType(const int i) const = 0;
     // Returns the type of primitive "i"

    // NumPrims methods
    virtual void AASetNumPrims(const int i, int n) = 0;
     // Sets the number of primitives in set "i" to "n"
    virtual void AASetNumPrims(const int num, int *numPrims) = 0;
     // Sets the number of primitives for each primitive set
    virtual int AANumPrims(const int i) const = 0;
     // Returns the number of primitives in set "i"

    // ColorType Methods
    virtual void AASetColorType(const int t) = 0;
     // Sets node color binding type
    virtual int AAColorType() = 0;
```

```
 // Returns the node color binding type

// Colors methods
virtual void AASetNumColors(const int num) = 0;
 // Sets the number of colors
virtual void AASetColors(const int num, unsigned int *Colors) = 0;
 // Sets all colors using an array
virtual int AANumColors() = 0;
 // Returns the number of colors
virtual unsigned int* AAColors() = 0;
 // Returns a pointer to the colors array, or NULL if no colors exist
virtual void AASetColor(const int i, const int c) = 0;
 // Sets color "i" to the given color (cpack version)
virtual unsigned int AAColor(const int i) = 0;
 // Returns color "i" in cpack format

// ColorIndex methods
virtual void AASetNumColorIndices(const int num) = 0;
 // Sets the number of color indices
virtual int AANumColorIndices() const = 0;
 // Returns the number of color indices
virtual void AASetColorIndices(const int num, int *index) = 0;
 // Sets all color indices using an array
virtual int* AAColorIndices() = 0;
 // Returns a pointer to the color indices, or NULL if no color indices exist
virtual void AASetColorIndex(const int i, const int n) = 0;
 // Sets color index "i" to the given index
virtual int AAColorIndex(const int i) = 0;
 // Returns color index "i"

// NormType methods
virtual void AASetNormType(const int t) = 0;
 // Sets the node normal binding type
virtual int AANormType() = 0;
 // Returns the node normal binding type

// Norms methods
virtual void AASetNumNorms(const int num) = 0;
 // Sets the number of normals
virtual int AANumNorms() const = 0;
 // Returns the number of normals
virtual void AASetNorms(const int num, float *norms) = 0;
 // Sets all normals using an array
virtual void AASetNorm(const int i, const AAVector *n) = 0;
 // Sets normal "i" to the given normal
virtual float* AANorms() = 0;
 // Returns a pointer to the normals
virtual AAVector* AANorm(const int i) = 0;
 // Returns a reference to normal "i" (no range checking)

// NormIndex methods
virtual void AASetNumNormIndices(const int num) = 0;
 // Sets the number of norm indices
virtual int AANumNormIndices() const = 0;
 // Returns the number of norm indices
virtual void AASetNormIndices(const int num, int *index) = 0;
 // Sets all normal indices using an array
virtual int* AANormIndices() = 0;
 // Returns a pointer to the norm indices
virtual void AASetNormIndex(const int i, const int num) = 0;
 // Sets normal index "i"
virtual int AANormIndex(const int i) = 0;
 // Returns normal index "i"

// TexType Methods
virtual void AASetTexType(const int t) = 0;
 // Sets the node texture binding type
virtual int AATexType() const = 0;
```

```
 // Returns the node texture binding type

// Texs methods
virtual void AASetNumTexs(const int num) = 0;
 // Sets the number of texture coords
virtual int AANumTexs() const = 0;
 // Returns the number of texture coords
virtual void AASetTexs(const int num, float *Texs) = 0;
 // Sets all texture coords using an array
virtual float* AATexs() = 0;
 // Returns a pointer to the texture coords
virtual void AASetTex(const int i, const float *t) = 0;
 // Sets texture coord "i" to the given coordinate
virtual float* AATex(int i) = 0;
 // Returns a pointer to texture coord "i"

// TexIndex methods
virtual void AASetNumTexIndices(const int num) = 0;
 // Sets the number of texture indices
virtual int AANumTexIndices() const = 0;
 // Returns the number of texture indices
virtual void AASetTexIndices(const int num, int *index) = 0;
 // Sets all texture indices using an array
virtual int* AATexIndices() = 0;
 // Returns a pointer to the texture indices
virtual void AASetTexIndex(const int i, const int num) = 0;
 // Returns texture index "i"
virtual int AATexIndex(const int i) = 0;
 // Returns texture index "i"

// VertType methods
virtual void AASetVertType(const int t) = 0;
 // Sets the node vertex binding (should always be PerVert, really)
virtual int AAVertType() = 0;
 // Returns the node vertex binding

// Verts Methods

virtual void AASetNumVerts(const int num) = 0;
 // Sets the number of vertices
virtual int AANumVerts() = 0;
 // Returns the number of vertices
virtual void AASetVerts(const int num, AAVector* verts) = 0;
 // Sets all vertices using an array
virtual void AASetVert(const int i, AAVector* verts) = 0;
 // Sets vertex "i" to the given coordinate
virtual float* AAVert(int i) = 0;
 // Returns a pointer to vertex "i"

// VertIndex methods
virtual void AASetNumVertIndices(const int num) = 0;
 // Sets the number of vertex indices
virtual int AANumVertIndices() = 0;
 // Returns the number of vertex indices
virtual void AASetVertIndices(const int num, int *index) = 0;
 // Sets all vertex indices using an array
virtual int* AAVertIndices() = 0;
 // Returns a pointer to the vertex indices, or NULL if none exist
virtual void AASetVertIndex(const int i, int num) = 0;
 // Returns vertex index "i"
virtual int AAVertIndex(const int i) = 0;
 // Returns vertex index "i"

// Node flags methods.
virtual void AASetNodeFlags(const unsigned int flags) = 0;
 // Sets the node flags
virtual void AAResetNodeFlags(const unsigned int flags) = 0;
 // Resets node flags (to zero)
```

```
    virtual void AAClearNodeFlags() = 0;
     // Clears all node flags
    virtual unsigned int AANodeFlags() = 0;
     // Returns the node flags

    // Material index methods.
    virtual void AASetMaterial(int m) = 0;
     // Sets the node material for the current depiction level (index version)
    virtual void AASetMaterial(const int depict, const int m) = 0;
     // Sets the node material for depiction level "depict" (index version)
//    virtual int AASetMaterial(AAMaterial *M) = 0;
     // Sets the node material for the current depiction level (pointer version)
//    virtual int AASetMaterial(const int depict, AAMaterial *M) = 0;
     // Sets the node material index for depiction level "depict" (pointer version)
    //virtual int AASetMaterial(const AAMaterial &M) = 0;
     // Sets the node material for the current depiction level (reference version)
    //virtual int AASetMaterial(const int depict, const AAMaterial &M) = 0;
     // Sets the node material for depiction level "depict" (reference version)
    virtual int AAMaterial() const = 0;
     // Returns the node material index for the current depiction level
    virtual int AAMaterial(const int depict) const = 0;
     // Returns the node material index for the given depiction level
    virtual void AAAssignMaterial(const int m) = 0;
     // Assigns material index 'm' to the node for the current depiction (without
setting the flags)
    virtual void AAAssignMaterial(const int depict, const int m) = 0;
     // Assigns material index 'm' to the node for the given depiction (without
setting the flags)

    // Texture index methods.
    virtual void AASetTexture(int t) = 0;
     // Sets the node texture for the current depiction level (index version)
    virtual void AASetTexture(const int depict, const int t) = 0;
     // Sets the node texture for depiction level "depict" (index version)
//    virtual int AASetTexture(AATexture *T) = 0;
     // Sets the node texture for the current depiction level (pointer version)
//    virtual int AASetTexture(const int depict, AATexture *T) = 0;
     // Sets the node texture index for depiction level "depict" (pointer version)
    //virtual int AASetTexture(const AATexture &T) = 0;
     // Sets the node texture for the current depiction level (reference version)
    //virtual int AASetTexture(const int depict, const AATexture &T) = 0;
     // Sets the node texture for depiction level "depict" (reference version)
    virtual int AATexture() = 0;
     // Returns the node texture index for the current depiction level
    virtual int AATexture(const int depict) = 0;
     // Returns the node texture index for the given depiction level
    virtual void AAAssignTexture(const int t) = 0;
     // Assigns texture index "t" to the node for the current depiction (without
setting the flags)
    virtual void AAAssignTexture(const int depict, const int t) = 0;
     // Assigns texture index "t" to the node for the given depiction (without set-
ting the flags)

    // Wireframe methods
    virtual void AASetWireframe(const BOOL wf = TRUE) = 0;
     // Sets the node wireframe flag
    virtual BOOL AAWireframe() = 0;
     // Returns the node wireframe flag

    // Culling volume methods.
    virtual void AASetCull(const AAVector* center, const float radius = 0.5f) = 0;
     // Sets the culling sphere center and radius
    virtual void AASetCull(const float cx = 0.0f,const float cy = 0.0f,const float
cz = 0.0f,const float r = 0.5f) = 0;
     // Sets the culling sphere center and radius (component version)
    virtual AAVector* AACullCenter() const = 0;
     // Returns the culling sphere center
    virtual float AACullRadius() const = 0;
```

```
     // Returns the culling sphere radius
    virtual void AAFindCull(AAVector* rmin=NULL, AAVector* rmax=NULL, AAGMatrix*
mat=NULL, BOOL UpdateCull=TRUE) = 0;
     // Recomputes the geometry limits and, optionally, sets the culling sphere
parameters

    // Transform method
    virtual void AATransform(const AAGMatrix* mat) = 0;
     // Transforms the geometry using the given matrix.  Normals are only rotated
and texture coordinates
        // are unaffected
    virtual void AATexTransform(const AAGMatrix* mat) = 0;
     // Transforms the texture coordinates using the given matrix.

    // Miscellaneous methods
//    virtual void AACountPolys(AAPolyCountP p) = 0;
     // Compute statistics for the node by counting vertices, etc. for the geometry
};
```

```
// AAMaterial.h : Defines the interface of the material class.
//
#define __AAMaterial__

enum {  AAIndexBit =     0x00000001, // Dirty bits
         AAAlphaBit =     0x00000002,
         AAAmbientBit =   0x00000004,
         AADiffuseBit =   0x00000008,
         AASpecularBit =  0x00000010,
         AAEmissionBit =  0x00000020,
         AAShininessBit = 0x00000040,
         AAAllBits =      0x0000007f};


class AAMaterial
{
public:
    // Define and Update operations
    virtual void AADefine() = 0;
    virtual BOOL AAUpdate() = 0;

    // Copy.

    //virtual void Copy(const AAMaterial &M) = 0;
     // Copies the material "M" into this material

    // Index methods

    virtual void AASetIndex(const short index) = 0;
     // Sets the material index
    virtual short AAIndex() const = 0;
     // Returns the material index

    // Alpha methods

    virtual void AASetAlpha(const float alpha) = 0;
     // Sets the material alpha
    virtual float AAAlpha() const = 0;
     // Returns the material alpha

    // Ambient methods

    virtual void AASetAmbient(const float r, const float g, const float b) = 0;
     // Sets the material ambient color
    virtual void AASetAmbient(const AAVector* v) = 0;
     // Sets the material ambient color (vtVector version)
    virtual void AASetAmbient(const AAColor* c) = 0;
     // Sets the material ambient color (vtColor version)
    virtual void AAGetAmbient(AAColor* c) = 0;
     // Gets the material ambient color (vtColor version)
    virtual const AAVector* AAAmbient() const = 0;
     // Returns the material ambient color as a vector

    // Diffuse methods

    virtual void AASetDiffuse(const float r, const float g, const float b) = 0;
     // Sets the material diffuse color
    virtual void AASetDiffuse(const AAVector* v) = 0;
     // Sets the material diffuse color (vtVector version)
    virtual void AASetDiffuse(const AAColor* c) = 0;
     // Sets the material diffuse color (vtColor version)
    virtual void AAGetDiffuse(AAColor* c) = 0;
     // Gets the material diffuse color (vtColor version)
    virtual const AAVector* AADiffuse() const = 0;
     // Returns the material diffuse color as a vector

    // Specular methods
```

```
    virtual void AASetSpecular(const float r, const float g, const float b) = 0;
     // Sets the material specular color
    virtual void AASetSpecular(const AAVector* v) = 0;
     // Sets the material specular color (vtVector version)
    virtual void AASetSpecular(const AAColor* c) = 0;
     // Sets the material specular color (vtColor version)
    virtual void AAGetSpecular(AAColor* c) = 0;
     // Gets the material specular color (vtColor version)
    virtual const AAVector* AASpecular() const = 0;
     // Returns the material specular color as a vector

    // Emission methods

    virtual void AASetEmission(const float r, const float g, const float b) = 0;
     // Sets the material emission color
    virtual void AASetEmission(const AAVector* v) = 0;
     // Sets the material emission color (vtVector version)
    virtual void AASetEmission(const AAColor* c) = 0;
     // Sets the material emission color (vtColor version)
    virtual void AAGetEmission(AAColor* c) = 0;
     // Gets the material emission color (vtColor version)
    virtual const AAVector* AAEmission() const = 0;
     // Returns the material emission color as a vector

    // Shininess methods

    virtual void AASetShininess(const float s) = 0;
     // Sets the material shininess
    virtual float AAShininess() const = 0;
     // Returns the material shininess

    // Dirty bits methods

    virtual void AASetDirty(const unsigned int which) = 0;
     // SetDirty
    virtual void AAResetDirty(const unsigned int which) = 0;
     // ResetDirty
    virtual unsigned int AAIsDirty(const int which=AAAllBits) = 0;
     // IsDirty
    virtual void AASetClean() = 0;
     // SetClean

    // I/O.

//   virtual BOOL Read(istream &in,unsigned int fileVer,BOOL isBinary = TRUE) = 0;
     // Input
//   virtual BOOL Write(ostream &out,BOOL isBinary = TRUE) = 0;
     // Output
//   virtual ostream &Dump(ostream &out = cout) = 0;
//   virtual ostream &Dump(ostream &out,char *title ) = 0;
     // ASCII dump method

    // Operators.

//   virtual friend ostream &operator<<(ostream &out, vtMaterial &M) = 0;
     // ASCII dump function
 //  virtual friend ostream &operator<<(ostream &out, vtMaterial *M) = 0;
     // ASCII dump function (pointer version)
//   virtual vtMaterial &operator=(const vtMaterial &M) = 0;
     // Assignment operator
//   virtual vtBoolean operator==(vtMaterial &M) = 0;
     // Equality operator
 //  virtual vtBoolean operator!=(vtMaterial &M) = 0;
     // Inequality operator
};
```

```
// AANode.h : Defines the interface of the node class.
//
#define __AANode__

class AANode
{
public:
     virtual void AAAddChild(AANode*) = 0;
     virtual void AAAddNext(AANode*) = 0;
     virtual BOOL AAChangeMaterial(int, int) = 0;
     virtual BOOL AAChangeTexture(int, int) = 0;
     virtual AANode* AAChild() = 0;
     virtual BOOL AADelChild(AANode*) = 0;
     virtual BOOL AADelChildren() = 0;
     virtual BOOL AADelNext(AANode*) = 0;
     virtual BOOL AADelNexts() = 0;
     virtual AANode* AANext() = 0;
     virtual int AANumRefs() = 0;
     virtual AANode* AAReadTree(char*) = 0;
     virtual int AARef() = 0;
     virtual BOOL AARemoveChild(AANode*) = 0;
     virtual AANode* AARemoveChildren() = 0;
     virtual BOOL AARemoveNext(AANode*) = 0;
     virtual AANode* AARemoveNexts() = 0;
     virtual void AASetChild(AANode*) = 0;
     virtual void AASetName(char*) = 0;
     virtual void AASetNext(AANode*) = 0;
     virtual int AAUnref() = 0;
private:
};
```

```
// AAScene.h : Defines the interface of the scene class.
//
#define __AAScene__

#ifndef __AAEntity__
#include "AAEntity.h"
#endif

class AAScene
{
public:
     virtual AAEntity* AAAddEntity(AAEntity*) = 0;
     virtual BOOL AARemoveEntity(char*) = 0;
     virtual BOOL AARemoveEntity(AAEntity*) = 0;
     virtual AAEntity* AALoad(char*) = 0;
private:
};
```

```
// AATNode.h : Defines the interface of the transformation node class.
//
#define __AATNode__

#ifndef __AANode__
#include "AANode.h"
#endif
#ifndef __AAVector__
#include "AAVector.h"
#endif
#ifndef __AAGMatrix__
#include "AAGMatrix.h"
#endif

// Transformation node flags.
const int AATN_NoOp = 0x00000001;
const int AATN_TexMat = 0x00000002;
enum AAAxis {
         AAX = 0,         // X AAAxis
         AAY = 1,         // Y AAAxis
         AAZ = 2};        // Z AAAxis

enum AAFlagsType {
     AANoOpFlag = 0x00000001,      // treat this node as the identity matrix
     AATexMatFlag = 0x00000002     // operates on the texture matrix
    };

class AATNode : public AANode
{
public:
    // Matrix methods

    virtual void AASetMat(const AAGMatrix* M) = 0;
     // Sets the matrix to "*M" (pointer version)
    virtual void AASetIdentity(void) = 0;
    // Sets the matrix to the identity matrix
    virtual AAGMatrix* AAMat() = 0;
     // Returns the matrix

    // Flags methods

    virtual void AASetFlags(const int flags) = 0;
     // Sets flags corresponding to bits of "flags"
    virtual void AAResetFlags(const int flags) = 0;
     // Resets (to zero) flags corresponding to bits of "flags"
    virtual void AAClearFlags() = 0;
     // Clears (to zero) all flags
    virtual int AAFlags() = 0;
     // Returns the flags

    // Member copy getting methods.

    virtual AAGMatrix* AAMatCopy() = 0;
     // Returns the copy matrix
    virtual int AAFlagsCopy() = 0;
     // Returns the copy flags

    // Transformation methods wrt this node's coordinates.

    virtual void AAScale(const float sx,const float sy,const float sz) = 0;
     // Scales the matrix, anisotropically (component version)
    virtual void AAScale(const float s) = 0;
     // Scales the matrix, isotropically
    virtual void AAScale(const AAVector* v) = 0;
     // Scales the matrix, anisotropically
    virtual void AARotate(const AAAxis axis,const float rot) = 0;
     // Rotates about "AAAxis" through angle "rot" (radians)
```

```
    virtual void AARotate(const AAAxis AAAxis1,const float rAxis1,const AAAxis AA-
Axis2,const float rAxis2,
     const AAAxis AAAxis3,const float rAxis3) = 0;
     // Rotates about the given axes, in the order specified, by the amounts speci-
fied (radians)
    virtual void AARotateX(const float ang) = 0;
     // Rotates about "X" through the given angle (radians)
    virtual void AARotateY(const float ang) = 0;
     // Rotates about "Y" through the given angle (radians)
    virtual void AARotateZ(const float ang) = 0;
     // Rotates about "Z" through the given angle (radians)
    virtual void AARotateXYZ(const float x,const float y,const float z) = 0;
     // Rotates about "X", "Y", and "Z" through the given angles (radians)
    virtual void AARotateZXY(const float z,const float x,const float y) = 0;
     // Rotates about "Z", "X", and "Y" through the given angles (radians)
    virtual void AARotateYZX(const float y,const float z,const float x) = 0;
     // Rotates about "Y", "Z", and "X" through the given angles (radians)
    virtual void AARotateXZY(const float x,const float z,const float y) = 0;
     // Rotates about "X", "Z", and "Y" through the given angles (radians)
    virtual void AARotateYXZ(const float y,const float x,const float z) = 0;
     // Rotates about "Y", "X", and "Z" through the given angles (radians)
    virtual void AARotateZYX(const float z,const float y,const float x) = 0;
     // Rotates about "Z", "Y", and "X" through the given angles (radians)
    virtual void AARotateXYZ(const AAVector *v) = 0;
     // Rotates about "X", "Y", and "Z" through the given angles (radians)
    virtual void AARotateZXY(const AAVector *v) = 0;
     // Rotates about "Z", "X", and "Y" through the given angles (radians)
    virtual void AARotateYZX(const AAVector *v) = 0;
     // Rotates about "Y", "Z", and "X" through the given angles (radians)
    virtual void AARotateXZY(const AAVector *v) = 0;
     // Rotates about "X", "Z", and "Y" through the given angles (radians)
    virtual void AARotateYXZ(const AAVector *v) = 0;
     // Rotates about "Y", "X", and "Z" through the given angles (radians)
    virtual void AARotateZYX(const AAVector *v) = 0;
     // Rotates about "Z", "Y", and "X" through the given angles (radians)
    virtual void AATranslate(const float x,const float y,const float z) = 0;
     // Translates by the given amounts (component version)
    virtual void AATranslate(const AAVector *t) = 0;
     // Translates by the given amounts (vector version)

    // Transformations methods wrt this node's coordinates with the matrix being
    // cleared first.

    virtual void AAXRotation(const float ang) = 0;
     // Rotates about "X" through the given angle (radians)
    virtual void AAYRotation(const float ang) = 0;
     // Rotates about "Y" through the given angle (radians)
    virtual void AAZRotation(const float ang) = 0;
     // Rotates about "Z" through the given angle (radians)
    virtual void AAXYZRotation(const float x,const float y,const float z) = 0;
     // Rotates about "X", "Y", and "Z" through the given angles (radians)
    virtual void AAZXYRotation(const float z,const float x,const float y) = 0;
     // Rotates about "Z", "X", and "Y" through the given angles (radians)
    virtual void AAYZXRotation(const float y,const float z,const float x) = 0;
     // Rotates about "Y", "Z", and "X" through the given angles (radians)
    virtual void AAXZYRotation(const float x,const float z,const float y) = 0;
     // Rotates about "X", "Z", and "Y" through the given angles (radians)
    virtual void AAYXZRotation(const float y,const float x,const float z) = 0;
     // Rotates about "Y", "X", and "Z" through the given angles (radians)
    virtual void AAZYXRotation(const float z,const float y,const float x) = 0;
     // Rotates about "Z", "Y", and "X" through the given angles (radians)
    virtual void AAXYZRotation(const AAVector *v) = 0;
     // Rotates about "X", "Y", and "Z" through the given angles (radians)
    virtual void AAZXYRotation(const AAVector *v) = 0;
     // Rotates about "Z", "X", and "Y" through the given angles (radians)
    virtual void AAYZXRotation(const AAVector *v) = 0;
     // Rotates about "Y", "Z", and "X" through the given angles (radians)
    virtual void AAXZYRotation(const AAVector *v) = 0;
```

```
     // Rotates about "X", "Z", and "Y" through the given angles (radians)
    virtual void AAYXZRotation(const AAVector *v) = 0;
     // Rotates about "Y", "X", and "Z" through the given angles (radians)
    virtual void AAZYXRotation(const AAVector *v) = 0;
     // Rotates about "Z", "Y", and "X" through the given angles (radians)
    virtual void AATranslation(const float x,const float y,const float z) = 0;
     // Translates by the given amounts (component version)
    virtual void AATranslation(const AAVector *t) = 0;
     // Translates by the given amounts (vector version)


    // Transformation methods wrt this node's base coordinates.

    virtual void AABaseScale(const float sx,const float sy,const float sz) = 0;
     // Base scale the matrix, anisotropically (component version)
    virtual void AABaseScale(const float s) = 0;
     // Base scale the matrix, isotropically
    virtual void AABaseScale(const AAVector *v) = 0;
     // Base scale the matrix, anisotropically
    virtual void AABaseRotate(const AAAxis AAAxis,const float rot) = 0;
     // Base Base rotate about "AAAxis" through angle "rot" (radians)
    virtual void AABaseRotate(const AAAxis AAAxis1,const float rAxis1,const AAAxis
AAAxis2,
     const float rAxis2,const AAAxis AAAxis3,const float rAxis3) = 0;
     // Base rotate about the given axes, in the order specified, by the
     // amounts specified (radians)
    virtual void AABaseRotateX(const float ang) = 0;
     // Base rotate about "X" through the given angle (radians)
    virtual void AABaseRotateY(const float ang) = 0;
     // Base rotate about "Y" through the given angle (radians)
    virtual void AABaseRotateZ(const float ang) = 0;
     // Base rotate about "Z" through the given angle (radians)
    virtual void AABaseRotateXYZ(const float x,const float y,const float z) = 0;
     // Base rotate about "X", "Y", and "Z" through the given angles (radians)
    virtual void AABaseRotateZXY(const float z,const float x,const float y) = 0;
     // Base rotate about "Z", "X", and "Y" through the given angles (radians)
    virtual void AABaseRotateYZX(const float y,const float z,const float x) = 0;
     // Base rotate about "Y", "Z", and "X" through the given angles (radians)
    virtual void AABaseRotateXZY(const float x,const float z,const float y) = 0;
     // Base rotate about "X", "Z", and "Y" through the given angles (radians)
    virtual void AABaseRotateYXZ(const float y,const float x,const float z) = 0;
     // Base rotate about "Y", "X", and "Z" through the given angles (radians)
    virtual void AABaseRotateZYX(const float z,const float y,const float x) = 0;
     // Base rotate about "Z", "Y", and "X" through the given angles (radians)
    virtual void AABaseRotateXYZ(const AAVector *v) = 0;
     // Base rotate about "X", "Y", and "Z" through the given angles (radians)
    virtual void AABaseRotateZXY(const AAVector *v) = 0;
     // Base rotate about "Z", "X", and "Y" through the given angles (radians)
    virtual void AABaseRotateYZX(const AAVector *v) = 0;
     // Base rotate about "Y", "Z", and "X" through the given angles (radians)
    virtual void AABaseRotateXZY(const AAVector *v) = 0;
     // Base rotate about "X", "Z", and "Y" through the given angles (radians)
    virtual void AABaseRotateYXZ(const AAVector *v) = 0;
     // Base rotate about "Y", "X", and "Z" through the given angles (radians)
    virtual void AABaseRotateZYX(const AAVector *v) = 0;
     // Base rotate about "Z", "Y", and "X" through the given angles (radians)
    virtual void AABaseTranslate(const float x,const float y,const float z) = 0;
     // Base translate by the given amounts (component version)
    virtual void AABaseTranslate(const AAVector *t) = 0;
     // Base translate by the given amounts (vector version)

    // Face methods

    virtual void AAFaceX(const AAVector *p) = 0;
     // Computes the matrix such that the +Z AAAxis comes closest to facing
     // the position "p" by rotating about the "X" AAAxis only
    virtual void AAFaceY(const AAVector *p) = 0;
     // Computes the matrix such that the +Z AAAxis comes closest to facing
```

```
     // the position "p" by rotating about the "Y" AAAxis only
    virtual void AAFaceXY(const AAVector *p) = 0;
     // Computes the matrix such that the +Z AAAxis faces the position "p"
     // by rotating about the "X" and "Y" axes
protected:
};
```

```
// AAVector.h : Defines the interface of the vector class.
//
#define __AAVector__

class AAVector
{
public:
     virtual void AAAbs(void) = 0;
     virtual void AAAbs(AAVector*) = 0;
     virtual AAVector* AACross(AAVector*, AAVector*) = 0;
     virtual float AADist(AAVector*) = 0;
     virtual float AADist2(AAVector*) = 0;
     virtual float AAGetX() = 0;
     virtual float AAGetY() = 0;
     virtual float AAGetZ() = 0;
     virtual float AAInnerProd(AAVector*) = 0;
     virtual float AAMax() = 0;
     virtual float AAMagnitude() = 0;
     virtual float AAMagSq() = 0;
     virtual float AAMin() = 0;
     virtual void AANegate() = 0;
     virtual void AANormalize() = 0;
     virtual void AANormalize(AAVector*) = 0;
     virtual void AAScale(float) = 0;
     virtual void AAScale(float, float, float) = 0;
     virtual void AASet(float, float, float) = 0;
     virtual void AASet(AAVector*) = 0;
     virtual void AASetX(const float) = 0;
     virtual void AASetY(const float) = 0;
     virtual void AASetZ(const float) = 0;

     virtual AAVector* operator+(AAVector*) = 0;
     virtual AAVector* operator-(AAVector*) = 0;
     virtual AAVector* operator*(AAVector*) = 0;
     virtual AAVector* operator*(float) = 0;
     virtual AAVector* operator=(AAVector*) = 0;
     virtual BOOL operator==(AAVector*) = 0;
     virtual BOOL operator!=(AAVector*) = 0;
     virtual float operator[](int) = 0;
     virtual const float operator[](int) const = 0;

protected:
private:
};
```

```
// AAViewport.h : Defines the interface of the viewport class.
//
#define __AAViewport__

#ifndef __AAScene__
#include "AAScene.h"
#endif
#ifndef __AAVector__
#include "AAVector.h"
#endif
#ifndef __AAGMatrix__
#include "AAGMatrix.h"
#endif

static enum FOGMODE {LINEAR, EXP, EXPSQR, TABLE};
static enum FLAGS { DRAWENABLE, CLEARPLANES, CLEARDEPTH, CLEARSTENCIL, STENCILE-
NABLE, FLIPVERTIVAL, FLIPHORIZONTAL};

// Basic enable constants
const int  AA_TEXTURE = 1;
const int  AA_LIGHTING = 2;
const int  AA_TRANSPARENCY = 3;
const int  AA_CULL_SPHERES = 4;
const int  AA_DEPTH_TEST = 5;
const int  AA_BACKFACE_CULL = 6;
const int  AA_OBJECT_CULL = 7;
const int  AA_MIPMAP_TEXTURES = 8;
const int  AA_ANTIALIAS = 9;
const int  AA_SMOOTH_SHADE = 10;
const int  AA_SUBFACE = 11;
const int  AA_FOG = 12;

// Light enable constants
const int  AA_LIGHT0 = 1000;
const int  AA_LIGHT1 = 1001;
const int  AA_LIGHT2 = 1002;
const int  AA_LIGHT3 = 1003;
const int  AA_LIGHT4 = 1004;
const int  AA_LIGHT5 = 1005;
const int  AA_LIGHT6 = 1006;
const int  AA_LIGHT7 = 1007;

class AAViewport
{
public:
     virtual BOOL AAAllocStencilBits(const int) = 0;
     virtual void AAClearFlags() = 0;
     virtual void AADisable(const int) = 0;
     virtual void AAEnable(const int) = 0;
     virtual void AAEnableStencil() = 0;
     virtual void AAFogDisable() = 0;
     virtual void AAFogEnable() = 0;
     virtual void AAFogSetColor(const float, const float, const float, const float)
= 0;
     virtual void AAFogSetDensity(const float) = 0;
     virtual void AAFogSetLinearRange(const float, const float) = 0;
     virtual void AAFogSetMode(FOGMODE) = 0;
     virtual int AANumStencilBits() = 0;
     virtual void AAPrepareForRender() = 0;
     virtual AAScene* AAGetScene() = 0;
     virtual void AASetColor(const float, const float, const float, const float a =
1.0f) = 0;
     virtual void AASetFlags(const FLAGS f) = 0;
     virtual void AASetOrtho(const float xmin,const float xmax,
                                  const float ymin,const float ymax,
                                  const float zmin,const float zmax) = 0;
     virtual void AASetPerspective(const float, const float, const float, const
float) = 0;
```

```
     virtual void AASetScene(AAScene*) = 0;
     virtual void AASetViewLookAroundAt(AAVector*, float, float, float, float) = 0;
     virtual void AASetViewLookFromTo(const float, const float, const float, const
float,
     const float, const float, const float ux = 0.0f, const float uy = 1.0f, const
float uz = 0.0f) = 0;
     virtual void AASetViewLookOrbit(AAVector*, float, float, float) = 0;
     virtual void AASetViewMat(AAGMatrix*) = 0;
     virtual void AAResetFlags(const FLAGS f) = 0;
     virtual void AASyncEnables() = 0;

private:
};
```

```
// AAWindow.h : Defines the interface of the window class.
//
#define __AAWindow__
#ifndef __AAVTreeViewport__
#include "AAViewport.h"
#endif

static enum ENABLES {ANTIALIAS, BACKFACECULL, CULLSPHERES, DEPTHTEST, FOG, LIGHT1,
LIGHT2, LIGHT3, LIGHT4, LIGHT5, LIGHT6, LIGHT7, LIGHTNING, MIPMAPTEXTURES, OB-
JECTCULL, SMOOTHSHADE, SUBFACE, TEXTURE, TRANSPARENCY};

class AAWindow
{
public:
     virtual AAViewport* AAAddViewport(AAViewport*) = 0;
     virtual AAViewport* AAFindViewport(char*) = 0;
     virtual BOOL AAAllocStencilBits(const int, unsigned int) = 0;
     virtual float AAAspect() = 0;
     virtual int AAHeight() = 0;
     virtual int AAWidth() = 0;
     virtual int AAX() = 0;
     virtual int AAY() = 0;
     virtual void AADisable(ENABLES) = 0;
     virtual void AAEnable(ENABLES) = 0;
     virtual BOOL AAFrame() = 0;
     virtual float AAFrameRate() = 0;
     virtual BOOL AAPrepareForRender() = 0;
     virtual BOOL AARemoveViewport(AAViewport*) = 0;
     virtual BOOL AARender() = 0;
private:
};
```

C.11 The Adapter Classes

```
// AAVTreeColor.h : Defines the adapter of the color class.
//
#define __AAVTreeColor__

#ifndef __AAColor__
#include "AAColor.h"
#endif

#include "vtcolor.h"

class AAVTreePackColor : public AAPackColor
{
friend class AAVTreeColor;
friend class AAVTree2DGeomElement;
friend class AAVTree2DGeomGaugeNode;

public:
     AAVTreePackColor(const int ir = 0, const int ig = 0, const int ib = 0, const
int ia = 255)
     {
          color.pr = ir;
          color.pg = ig;
          color.pb = ib;
          color.pa = ia;
     };
     AAVTreePackColor(unsigned int c){color.Set(c);};
     virtual void AASet(int ir, int ig, int ib, int ia = 255){color.Set(ir, ig, ib,
ia);};
     virtual void AASet(int c){color.Set(c);};
     virtual int AAPacked(){return color.Packed();};
     virtual int     AAr(){return color.r();};
     virtual int AAg(){return color.g();};
     virtual int AAb(){return color.b();};
     virtual int AAa(){return color.a();};
protected:
     struct vtPackColor color;
};

class AAVTreeColor : public AAColor
{
friend class AAVTreePackColor;
friend class AAVTreeFxDebris;
friend class AAVTreeFxMissileTrail;
friend class AAVTreeMaterial;
friend class AAVTreeFxExplosion;

public:
     AAVTreeColor(float ir = 0.0f, float ig = 0.0f, float ib = 0.0f, float ia =
1.0f)
     {
          color.r = ir;
          color.g = ig;
          color.b = ib;
          color.a = ia;
     };

     AAVTreeColor(AAColor *c)
     {
          color.r = ((AAVTreeColor*)c)->color.r;
          color.g = ((AAVTreeColor*)c)->color.g;
          color.b = ((AAVTreeColor*)c)->color.b;
          color.a = ((AAVTreeColor*)c)->color.a;
     };
```

```
     AAVTreeColor(AAPackColor *pc){color.FromPacked(((AAVTreePackColor*)pc)-
>color);};

     void AASet(float ir, float ig, float ib, float ia = 1.0f){color.Set(ir, ig,
ib, ia);};
     void AAFromPacked(AAPackColor *pc){color.FromPacked(((AAVTreePackColor*)pc)-
>color);};
     float AAr(){return color.r;};
     float AAg(){return color.g;};
     float AAb(){return color.b;};
     float AAa(){return color.a;};

protected:
     struct vtColor color;
};
```

```
// AAVTreeEntity.h : Defines the entity class adapter.
//
#define __AAVTreeEntity__

#include "vtentity.h"

#include "vtFxExplosionBits.h"
#include "vtFxExplosion.h"

#ifndef __AAVTreeFxExplosion__
#include "AAVTreeFxExplosion.h"
#endif
#ifndef __AAVTreeFxDebris__
#include "AAVTreeFxDebris.h"
#endif
#ifndef __AAVTreeFxMissileTrail__
#include "AAVTreeFxMissileTrail.h"
#endif
#ifndef __AAEntity__
#include "AAEntity.h"
#endif
#ifndef __AAVTreeGMatrix__
#include "AAVTreeGMatrix.h"
#endif
#ifndef __AAVTreeVector__
#include "AAVTreeVector.h"
#endif
#ifndef __AAVector__
#include "AAVector.h"
#endif
#ifndef __AAVTreeNode__
#include "AAVTreeNode.h"
#endif

class AAVTreeEntity : public AAEntity, private vtEntity
{
public:
     AAVTreeEntity(char* string, vtNode* tree = NULL) : vtEntity(string, tree){};

     void AASetDepthCenter(const float x, const float y, const float z)
     {this->SetDepthCenter(x, y, z);};

     void AASetDepthSorted(const BOOL b){this->SetDepthSorted(b);};

     void AASetModelToWorld(AAGMatrix *m){this-
>SetModelToWorld(((AAVTreeGMatrix*)m)->matrix);};

     void AASetPos(const float x, const float y, const float z){this->SetPos(x, y,
z);};

     void AASetPos(AAVector* v){this->SetPos(((AAVTreeVector*)v)->vector);};

     void AASetPosYPR(AAVector* v, const float yaw, const float pitch, const float
roll)
     {this->SetPosYPR(((AAVTreeVector*)v)->vector, yaw, pitch, roll);};

     void AASetPosYPR(const float x, const float y, const float z, const float yaw,
const float pitch, const float roll)
     {this->SetPosYPR(x, y, z, yaw, pitch, roll);};

     void AASetRot(const float x, const float y, const float z){this->SetRot(x, y,
z);};

     void AASetTree(AANode* tree){this->SetTree((vtNode*)((AAVTreeNode*)tree));};

     void AASetVisible(BOOL v){this->SetVisible(v);};
```

```
     void AAAddFx(AAFxExplosion* fx){this-
>AddFx((vtFxExplosion*)((AAVTreeFxExplosion*)fx));};
     void AAAddFx(AAFxDebris* fx){this-
>AddFx((vtFxExplosionBits*)((AAVTreeFxDebris*)fx));};
     void AAAddFx(AAFxMissileTrail* fx){this-
>AddFx((vtFxMissileTrail*)((AAVTreeFxMissileTrail*)fx));};

private:
};
```

```
// AAVTreeFxChunk.h : Defines the adapter of the fx chunk class.
//
#define __AAVTreeFxChunk__
#ifndef __AAFxChunk__
#include "AAFxChunk.h"
#endif
#include "vtfxexplosionchunk.h"

class AAVTreeFxChunk : public AAFxChunk, private vtFxExplosionChunks
{
public:
     AAVTreeFxChunk(const AAVector* minPos = DefMinPos, const AAVector* maxPos =
DefMaxPos,
                         const AAVector* minVel = DefMinVel , const AAVector* maxVel
= DefMaxVel,
                         const float gravity = DefGravity , const unsigned int color
= DefColor,
                         const int maxSides = DefMaxSides , const AAVector* minSize
= DefMinSize,
                         const AAVector* maxSize = DefMaxSize , const AAVector* min-
Rate = DefMinRate,
                         const AAVector* maxRate = DefMaxRate) {};
     AAColor* AAColor() {};
     AAColor* AAColor(const int depict) {};
     AAFxChunk* AADuplicate() {};
     float AAGravity() const {};
     double AALastTime() {};
     AAVector* AAPos() {};
     AAVector* AARot() {};
     AAVector* AARotRate() {};
     BOOL AARunning() const {};
     void AASetColor(const unsigned int c) {};
     void AASetColor(const int depict, const unsigned int c) {};
     void AASetGravity(const float g) {};
     void AASetLastTime(const double t) {};
     void AASetPos(const AAVector* pos) {};
     void AASetRot(const AAVector* rot) {};
     void AASetRotRate(const AAVector* rate) {};
     void AASetVel(const AAVector* vel) {};
     void AAStart() {};
     void AAStop() {};
     AAVector* AAVel() {};

private:
};
```

```
// AAVTreeFxDebris.h : Defines the adapter of the fx debris class.
//
#define __AAVTreeFxDebris__
#ifndef __AAFxDebris__
#include "AAFxDebris.h"
#endif
#include "vtfxexplosionbits.h"

#ifndef __AAVector__
#include "AAVector.h"
#endif
#ifndef __AAVTreeVector__
#include "AAVTreeVector.h"
#endif
#ifndef __AAVTreeColor__
#include "AAVTreeColor.h"
#endif


class AAVTreeFxDebris : public AAFxDebris, private vtFxExplosionBits
{
public:
     AAVTreeFxDebris(const char* name) : vtFxExplosionBits(name){};
     //BOOL AAAddChunk(AAFxChunk* chunk) {return this->AddChunk(chunk);};
     //AAFxChunk* AAChunk(const int index) {};
     float AAChunkSize() {return this->ChunkSize();};
     //AAVector* AAMaxVel() {};
     //AAVector* AAMinVel() {};
     int AANumChunks(){return this->NumChunks();};
     //BOOL AARemoveChunk(AAFxChunk* chunk, const BOOL del = TRUE) {};
     void AAReset(const BOOL redoChunks = FALSE) {this->Reset(redoChunks);};
     void AASetChunkSize(const float chunkSize) {this->SetChunkSize(chunkSize);};
     void AASetMaxVel(const AAVector* maxVel) {this-
>SetMaxVel(((AAVTreeVector*)maxVel)->vector);};
     void AASetMinVel(const AAVector* minVel) {this-
>SetMinVel(((AAVTreeVector*)minVel)->vector);};
     void AASetNumChunks(const int num, const BOOL del = TRUE) {this-
>SetNumChunks(num, del);};
     void AASetVel(const AAVector* minVel, const AAVector* maxVel) {this-
>SetVel(((AAVTreeVector*)minVel)->vector, ((AAVTreeVector*)maxVel)->vector);};
     /**********************AAFx METHODS**********************/
     void AAStart(float start, float stop){this->Start(start, stop);};
     void AAStop(float stop){this->Stop(stop);};
     void AASetDuration(float dur){this->SetDuration(dur);};
     void AASetFadeDuration(float fdur){this->SetFadeDuration(fdur);};
     void AASetColor(AAColor* c){this->SetColor(((AAVTreeColor*)c)->color);};
     BOOL AASetMoveWithParent(const BOOL f){return this->SetMoveWithParent(f);};
private:
};
```

```
// AAVTreeFxExplosion.h : Defines the adapter of the fx explosion class.
//
#define __AAVTreeFxExplosion__

#ifndef __AAFxExplosion__
#include "AAFxExplosion.h"
#endif

#ifndef __AAVTreeColor__
#include "AAVTreeColor.h"
#endif


#include "vtfxexplosion.h"

class AAVTreeFxExplosion : public AAFxExplosion, private vtFxExplosion
{
public:
     // Constructors

    AAVTreeFxExplosion(const char *name) : vtFxExplosion(name) {};
     // Constructor

    // Type methods

    void AASetType(const AAAnimType type){this->SetType((AnimType)type);};
     // Sets the animation type (OneShotType, LoopType, etc.)
    AAAnimType AAType() const{return (AAAnimType)this->Type();};
     // Returns the animation type

    // Direction methods

    void AASetDirection(const int dir){this->SetDirection(dir);};
     // Sets the animation direction (positive and zero means forward,
     // negative means backwards)
    int AADirection() const{return this->Direction();};
     // Returns the direction

    // HoldLast methods

    void AASetHoldLast(const BOOL hold){this->SetHoldLast(hold);};
     // If 'hold' is TRUE, the last frame of the animation will be kept
     // visible (for non looping animations) until the effect duration
     // is exceeded.
    BOOL AAHoldLast() const {return this->HoldLast();};
     // Returns TRUE if the last animation frame will be held

    // Frame duration methods

    void AASetFrameDuration(const float dur){this->SetFrameDuration(dur);};
     // Sets the duration (in seconds) of each frame in the animation
    float AAFrameDuration() const{return this->FrameDuration();};
     // Returns the frame duration (in seconds) of every frame in
     // the animation

    void AASetDurationAttributes(const BOOL durAttr){this-
>SetDurationAttributes(durAttr);};
     // If set to 'TRUE' the attribute trables continue to be
     // interpreted through the end of the effect.  If set to
     // 'FALSE', the attributes stop at the end of the last frame
     // which may not be at the end of the effect.
     // Default is 'FALSE'.
    BOOL AADurationAttributes(void) const{return this->DurationAttributes();};
     // Returns the current value of the m_DurationAttributes flag.

    // Loop methods

    void AASetLoopStart(const int l){this->SetLoopStart(l);};
```

```
     // Sets the loop starting frame number (-1 to disable looping)
    int AALoopStart() const{return this->LoopStart();};
     // Returns the loop starting frame number
    void AASetLoopEnd(const int l){this->SetLoopEnd(l);};
     // Sets the loop ending frame number (-1 to disable looping)
    int AALoopEnd() const{return this->LoopEnd();};
     // Returns the loop ending frame number
    void AASetLoop(const int s, const int e){this->SetLoop(s, e);};
     // Sets the loop start and end frames (-1's to disable looping)
    BOOL AAHasLoop(){return this->HasLoop();};
     // Returns TRUE if there is an animation loop
     void AASetSize(float x, float y, float z){this->SetSize(x, y, z);};

     /**********************AAFx METHODS**********************/
     void AAStart(float start, float stop){this->Start(start, stop);};
     void AAStop(float stop){this->Stop(stop);};
     void AASetDuration(float dur){this->SetDuration(dur);};
     void AASetFadeDuration(float fdur){this->SetFadeDuration(fdur);};
     void AASetColor(AAColor* c){this->SetColor(((AAVTreeColor*)c)->color);};
     BOOL AASetMoveWithParent(const BOOL f){return this->SetMoveWithParent(f);};

};
```

```
// AAVTreeFxMissileTrail.h : Defines the adapter of the fx missile trail class.
//
#define __AAVTreeFxMissileTrail__
#ifndef __AAFxMissileTrail__
#include "AAFxMissileTrail.h"
#endif
#include "vtFxMissileTrail.h"

#ifndef __AAVector__
#include "AAVector.h"
#endif
#ifndef __AAVTreeVector__
#include "AAVTreeVector.h"
#endif
#ifndef __AAVTreeColor__
#include "AAVTreeColor.h"
#endif


class AAVTreeFxMissileTrail : public AAFxMissileTrail, private vtFxMissileTrail
{
public:
     AAVTreeFxMissileTrail(const char *name, AAVector *pos, float dur, float size)
: vtFxMissileTrail(name, ((AAVTreeVector*)pos)->vector, dur, size){};
     void AASetAddInterval(const float d){};
     void AASetRotRate(const float dur){};
     void AASetExpandTime(const float dur){};
     void AASetTexScale(const float u, const float v){};
     void AASetGenerate(const BOOL gen){};
     BOOL AAStartTrail(const float t){return this->Start(t);};
     BOOL AAStopTrail(const float t){return this->Stop(t);};
     BOOL AASetDurationTrail(const float dur){return this->SetDuration(dur);};
     BOOL AASetFadeDurationTrail(const float dur){return this-
>SetFadeDuration(dur);};
     BOOL AASetNodeDuration(const float dur){return this->SetNodeDuration(dur);};

     /**********************AAFx METHODS**********************/
     void AAStart(float start, float stop){this->Start(start);};
     void AAStop(float stop){this->Stop(stop);};
     void AASetDuration(float dur){this->SetDuration(dur);};
     void AASetFadeDuration(float fdur){this->SetFadeDuration(fdur);};
     void AASetColor(AAColor* c){this->SetColor(((AAVTreeColor*)c)->color);};
     BOOL AASetMoveWithParent(const BOOL f){return this->SetMoveWithParent(f);};
private:
};
```

```
// AAVTreeGMatrix.h : Defines the adapter of the G matrix class.
//
#define __AAVTreeGMatrix__
#include "vtGMatrix.h"

#ifndef __AAGMatrix__
#include "AAGMatrix.h"
#endif
#ifndef __AAVector__
#include "AAVector.h"
#endif
#ifndef __AAVTreeVector__
#include "AAVTreeVector.h"
#endif

class AAVTreeGMatrix : public AAGMatrix
{
     friend class AAVTreeViewport;
     friend class AAVTreeEntity;
     friend class AAVTree2DGeomElement;
     friend class AAVTree2DGeomGaugeNode;
     friend class AAVTreeTNode;
     friend class AAVTreeIGNode;
public:
     AAVTreeGMatrix(){matrix = vtGIdent;};
     void AAArbRotation(float ang, float x, float y, float z){vtArbRotation(matrix,
ang, x, y, z);};
     void AAArbRotation(float ang, AAVector* axis){vtArbRotation(matrix, ang,
((AAVTreeVector*)axis)->vector);};
     void AABaseRotateArb(float ang, float x, float y, float
z){vtBaseRotateArb(matrix, ang, x, y, z);};
     void AABaseRotateArb(float ang, AAVector* axis){vtBaseRotateArb(matrix, ang,
((AAVTreeVector*)axis)->vector);};
     void AABaseRotateX(float ang){vtBaseRotateX(matrix, ang);};
     void AABaseRotateXYZ(float x, float y, float z){vtBaseRotateXYZ(matrix, x, y,
z);};
     void AABaseRotateXZY(float x, float y, float z){vtBaseRotateXZY(matrix, x, y,
z);};
     void AABaseRotateY(float ang){vtBaseRotateY(matrix, ang);};
     void AABaseRotateYXZ(float x, float y, float z){vtBaseRotateYXZ(matrix, x, y,
z);};
     void AABaseRotateYZX(float x, float y, float z){vtBaseRotateYZX(matrix, x, y,
z);};
     void AABaseRotateZ(float ang){vtBaseRotateZ(matrix, ang);};
     void AABaseRotateZXY(float x, float y, float z){vtBaseRotateZXY(matrix, x, y,
z);};
     void AABaseRotateZYX(float x, float y, float z){vtBaseRotateZYX(matrix, x, y,
z);};
     void AABaseScale(float sx, float sy, float sz){vtBaseScale(matrix, sx, sy,
sz);};
     void AABaseTranslate(AAVector* v){vtBaseTranslate(matrix, ((AAVTreeVector*)v)-
>vector);};
     void AABaseTranslate(float x, float y, float z){vtBaseTranslate(matrix, x, y,
z);};
     void AAFromPosYPR(AAVector* fromPos, float yaw, float pitch, float
roll){vtFromPosYPR(matrix, ((AAVTreeVector*)fromPos)->vector, yaw, pitch, roll);};
     void AAFromPosYPR(float fx, float fy, float fz, float yaw, float pitch, float
roll){vtFromPosYPR(matrix, fx, fy, fz, yaw, pitch, roll);};
     void AARotateArb(float ang, float ax, float ay, float az){vtRotateArb(matrix,
ang, ax, ay, az);};
     void AARotateArb(float ang, AAVector* axis){vtRotateArb(matrix, ang,
((AAVTreeVector*)axis)->vector);};
     void AARotateX(float ang){vtRotateX(matrix, ang);};
     void AARotateXYZ(float x, float y, float z){vtRotateXYZ(matrix, x, y, z);};
     void AARotateXZY(float x, float y, float z){vtRotateXZY(matrix, x, y, z);};
     void AARotateY(float ang){vtRotateY(matrix, ang);};
     void AARotateYXZ(float x, float y, float z){vtRotateYXZ(matrix, x, y, z);};
     void AARotateYZX(float x, float y, float z){vtRotateYZX(matrix, x, y, z);};
```

```
     void AARotateZ(float ang){vtRotateZ(matrix, ang);};
     void AARotateZXY(float x, float y, float z){vtRotateZXY(matrix, x, y, z);};
     void AARotateZYX(float x, float y, float z){vtRotateZYX(matrix, x, y, z);};
     void AAScale(float sx, float sy, float sz){vtScale(matrix, sx, sy, sz);};
     void AASet(AAGMatrix* m){vtSet(matrix, ((AAVTreeGMatrix*)m)->matrix);};
     void AASub(AAGMatrix* m){vtSub(matrix, ((AAVTreeGMatrix*)m)->matrix);};
     void AATranslate(AAVector* v){vtTranslate(matrix, ((AAVTreeVector*)v)-
>vector);};
     void AATranslate(float x, float y, float z){vtTranslate(matrix, x, y, z);};
     void AATranslation(AAVector* v){vtTranslation(matrix, ((AAVTreeVector*)v)-
>vector);};
     void AATranslation(float x, float y, float z){vtTranslation(matrix, x, y,
z);};
     void AATranspose(){vtTranspose(matrix);};
protected:
     vtGMatrix* AAGetMatrix(){return new vtGMatrix(matrix);};
     vtGMatrix matrix;
private:
};
```

```
// AAVTreeIGNode.h : Defines the adapter of the indexed geometry node class.
//
#define __AAVTreeIGNode__

#ifndef __AAIGNode__
#include "AAIGNode.h"
#endif
#include "vtIGNode.h"

class AAVTreeIGNode : public AAIGNode, private vtIGNode
{

public:
    // Constructors
    AAVTreeIGNode(   const char *name = NULL,
          vtNode *next = NULL,
          vtNode *child = NULL,
          int ColorType = AAPerNone,
          int NormType = AAPerNone,
          int TexType = AAPerNone,
          int VertType = AAPerNone,
          unsigned int NodeFlags = 0,
          int Material = 0,
          int Texture = 0,
          BOOL Wireframe = FALSE,
          int NumPrimSets = 0,
          int *PrimTypes = NULL,
          int *NumPrimVerts = NULL,
          int *NumPrims = NULL,
          int NumColors = 0,
          unsigned int *Colors = NULL,
          int NumNorms = 0,
          float *Norms = NULL,
          int NumTexs = 0,
          float *Texs = NULL,
          int NumVerts = 0,
          float *Verts = NULL,
          int NumColorIndex = 0,
          int *ColorIndex = NULL,
          int NumNormIndex = 0,
          int *NormIndex = NULL,
          int NumTexIndex = 0,
          int *TexIndex = NULL,
          int NumVertIndex = 0,
          int *VertIndex = NULL):
     vtIGNode(
          name,
          next,
          child,
          ColorType,
          NormType,
          TexType,
          VertType,
          NodeFlags,
          Material,
          Texture,
          Wireframe,
          NumPrimSets,
          PrimTypes,
          NumPrimVerts,
          NumPrims,
          NumColors,
          Colors,
          NumNorms,
          Norms,
          NumTexs,
          Texs,
          NumVerts,
```

```
          Verts,
          NumColorIndex,
          ColorIndex,
          NumNormIndex,
          NormIndex,
          NumTexIndex,
          TexIndex,
          NumVertIndex,
          VertIndex){};

    // Number of primitives
     int AANumPrimSets(){return this->NumPrimSets();};
     // number of primitive sets
     void AASetNumPrimSets(const int num){this->SetNumPrimSets(num);};
     // set number of primitive sets

    // Number of polygon vertices methods
     int AANumPrimVerts(const int i) const{return this->NumPrimVerts(i);};
     // number of primitive verts
     void AASetNumPrimVerts(const int i, int n){this->SetNumPrimVerts(i, n);};
     // set number of primitive verts

    // Primitive type methods
     void AASetPrimTypes(const int num, int *primtypes){this->SetPrimTypes(num,
primtypes);};
     // Sets the primitive types using an array
     void AASetPrimType(const int i, int n){this->SetPrimType(i, n);};
     // Sets the type of primitive "i" to type "n"
     int AAPrimType(const int i) const{return this->PrimType(i);};
     // Returns the type of primitive "i"

    // NumPrims methods
     void AASetNumPrims(const int i, int n){this->SetNumPrims(i, n);};
     // Sets the number of primitives in set "i" to "n"
     void AASetNumPrims(const int num, int *numPrims){this->SetNumPrims(num, num-
Prims);};
     // Sets the number of primitives for each primitive set
     int AANumPrims(const int i) const{return this->NumPrims(i);};
     // Returns the number of primitives in set "i"

    // ColorType Methods
     void AASetColorType(const int t){this->SetColorType(t);};
     // Sets node color binding type
     int AAColorType(){return this->ColorType();};
     // Returns the node color binding type

    // Colors methods
     void AASetNumColors(const int num){this->SetNumColors(num);};
     // Sets the number of colors
     void AASetColors(const int num, unsigned int *Colors){this->SetColors(num,
Colors);};
     // Sets all colors using an array
     int AANumColors(){return this->NumColors();};
     // Returns the number of colors
     unsigned int* AAColors(){return this->Colors();};
     // Returns a pointer to the colors array, or NULL if no colors exist
     void AASetColor(const int i, const int c){this->SetColor(i, c);};
     // Sets color "i" to the given color (cpack version)
     unsigned int AAColor(const int i){return this->Color(i);};
     // Returns color "i" in cpack format

    // ColorIndex methods
     void AASetNumColorIndices(const int num){this->SetNumColorIndices(num);};
     // Sets the number of color indices
     int AANumColorIndices() const{return this->NumColorIndices();};
     // Returns the number of color indices
     void AASetColorIndices(const int num, int *index){this->SetColorIndices(num,
index);};
```

```
     // Sets all color indices using an array
     int* AAColorIndices(){return this->ColorIndices();};
     // Returns a pointer to the color indices, or NULL if no color indices exist
     void AASetColorIndex(const int i, const int n){this->SetColorIndex(i, n);};
     // Sets color index "i" to the given index
     int AAColorIndex(const int i){return this->ColorIndex(i);};
     // Returns color index "i"

    // NormType methods
     void AASetNormType(const int t){this->SetNormType(t);};
     // Sets the node normal binding type
     int AANormType(){return this->NormType();};
     // Returns the node normal binding type

    // Norms methods
     void AASetNumNorms(const int num){this->SetNumNorms(num);};
     // Sets the number of normals
     int AANumNorms() const{return this->NumNorms();};
     // Returns the number of normals
     void AASetNorms(const int num, float *norms){this->SetNorms(num, norms);};
     // Sets all normals using an array
     void AASetNorm(const int i, const AAVector *n){this->SetNorm(i, (vtVec-
tor)((AAVTreeVector*)n)->vector);};
     // Sets normal "i" to the given normal
     float* AANorms(){return this->Norms();};
     // Returns a pointer to the normals
     AAVector* AANorm(const int i)
      {
            AAVTreeVector* v = new AAVTreeVector();
            v->vector = this->Norm(i);
            return v;
      };
     // Returns a reference to normal "i" (no range checking)

    // NormIndex methods
     void AASetNumNormIndices(const int num){this->SetNumNormIndices(num);};
     // Sets the number of norm indices
     int AANumNormIndices() const{return this->NumNormIndices();};
     // Returns the number of norm indices
     void AASetNormIndices(const int num, int *index){this->SetNormIndices(num, in-
dex);};
     // Sets all normal indices using an array
     int* AANormIndices(){return this->NormIndices();};
     // Returns a pointer to the norm indices
     void AASetNormIndex(const int i, const int num){this->SetNormIndex(i, num);};
     // Sets normal index "i"
     int AANormIndex(const int i){return this->NormIndex(i);};
     // Returns normal index "i"

    // TexType Methods
     void AASetTexType(const int t){this->SetTexType(t);};
     // Sets the node texture binding type
     int AATexType() const{return this->TexType();};
     // Returns the node texture binding type

    // Texs methods
     void AASetNumTexs(const int num){this->SetNumTexs(num);};
     // Sets the number of texture coords
     int AANumTexs() const{return this->NumTexs();};
     // Returns the number of texture coords
     void AASetTexs(const int num, float *Texs){this->SetTexs(num, Texs);};
     // Sets all texture coords using an array
     float* AATexs(){return this->Texs();};
     // Returns a pointer to the texture coords
     void AASetTex(const int i, const float *t){this->SetTex(i, t);};
     // Sets texture coord "i" to the given coordinate
     float* AATex(int i){return this->Tex(i);};
     // Returns a pointer to texture coord "i"
```

```
    // TexIndex methods
     void AASetNumTexIndices(const int num){this->SetNumTexIndices(num);};
     // Sets the number of texture indices
     int AANumTexIndices() const{return this->NumTexIndices();};
     // Returns the number of texture indices
     void AASetTexIndices(const int num, int *index){this->SetTexIndices(num, in-
dex);};
     // Sets all texture indices using an array
     int* AATexIndices(){return this->TexIndices();};
     // Returns a pointer to the texture indices
     void AASetTexIndex(const int i, const int num){this->SetTexIndex(i, num);};
     // Returns texture index "i"
     int AATexIndex(const int i){return this->TexIndex(i);};
     // Returns texture index "i"

    // VertType methods
     void AASetVertType(const int t){this->SetVertType(t);};
     // Sets the node vertex binding (should always be PerVert, really)
     int AAVertType(){return this->VertType();};
     // Returns the node vertex binding

    // Verts Methods

     void AASetNumVerts(const int num){this->SetNumVerts(num);};
     // Sets the number of vertices
     int AANumVerts(){return this->NumVerts();};
     // Returns the number of vertices
     void AASetVerts(const int num, AAVector* verts){this-
>SetVerts(num,(float*)&((vtVector)((AAVTreeVector*)verts)->vector));
      };
     // Returns a pointer to the vertices, or NULL if none exist
     void AASetVert(const int i, AAVector* verts){this->SetVert(i,
(float*)&((vtVector)((AAVTreeVector*)verts)->vector));};
     // Sets vertex "i" to the given coordinate
     float* AAVert(int i){return this->Vert(i);};
     // Returns a pointer to vertex "i"

    // VertIndex methods
     void AASetNumVertIndices(const int num){this->SetNumVertIndices(num);};
     // Sets the number of vertex indices
     int AANumVertIndices(){return this->NumVertIndices();};
     // Returns the number of vertex indices
     void AASetVertIndices(const int num, int *index){this->SetVertIndices(num, in-
dex);};
     // Sets all vertex indices using an array
     int* AAVertIndices(){return this->VertIndices();};
     // Returns a pointer to the vertex indices, or NULL if none exist
     void AASetVertIndex(const int i, int num){this->SetVertIndex(i, num);};
     // Returns vertex index "i"
     int AAVertIndex(const int i){return this->VertIndex(i);};
     // Returns vertex index "i"

    // Node flags methods.
     void AASetNodeFlags(const unsigned int flags){this->SetNodeFlags(flags);};
     // Sets the node flags
     void AAResetNodeFlags(const unsigned int flags){this->ResetNodeFlags(flags);};
      // Resets node flags (to zero)
     void AAClearNodeFlags(){this->ClearNodeFlags();};
     // Clears all node flags
     unsigned int AANodeFlags(){return this->NodeFlags();};
     // Returns the node flags

    // Material index methods.
     void AASetMaterial(int m){this->SetMaterial(m);};
     // Sets the node material for the current depiction level (index version)
     void AASetMaterial(const int depict, const int m){this->SetMaterial(depict,
m);};
```

```
     // Sets the node material for depiction level "depict" (index version)
     int AAMaterial() const{return this->Material();};
     // Returns the node material index for the current depiction level
     int AAMaterial(const int depict) const{return this->Material(depict);};
     // Returns the node material index for the given depiction level
     void AAAssignMaterial(const int m){this->AssignMaterial(m);};
     // Assigns material index 'm' to the node for the current depiction (without
setting the flags)
     void AAAssignMaterial(const int depict, const int m){this-
>AssignMaterial(depict, m);};
     // Assigns material index 'm' to the node for the given depiction (without
setting the flags)

    // Texture index methods.
     void AASetTexture(int t){this->SetTexture(t);};
     // Sets the node texture for the current depiction level (index version)
     void AASetTexture(const int depict, const int t){this->SetTexture(depict,
t);};
     int AATexture(){return this->Texture();};
     // Returns the node texture index for the current depiction level
     int AATexture(const int depict){return this->Texture(depict);};
     // Returns the node texture index for the given depiction level
     void AAAssignTexture(const int t){this->AssignTexture(t);};
     // Assigns texture index "t" to the node for the current depiction (without
setting the flags)
     void AAAssignTexture(const int depict, const int t){this-
>AssignTexture(depict, t);};
     // Assigns texture index "t" to the node for the given depiction (without set-
ting the flags)

    // Wireframe methods
     void AASetWireframe(const BOOL wf = TRUE){this->SetWireframe(wf);};
     // Sets the node wireframe flag
     BOOL AAWireframe(){return this->Wireframe();};
     // Returns the node wireframe flag

    // Culling volume methods.
     void AASetCull(const AAVector* center, const float radius = 0.5f){this-
>SetCull((vtVector)((AAVTreeVector*)center)->vector, radius);};
     // Sets the culling sphere center and radius
     void AASetCull(const float cx = 0.0f,const float cy = 0.0f,const float cz =
0.0f,const float r = 0.5f){this->SetCull(cx, cy, cz, r);};
     // Sets the culling sphere center and radius (component version)
     AAVector* AACullCenter() const
      {
           AAVTreeVector* v = new AAVTreeVector();
           v->vector = CullCenter();
           return v;
      };
     // Returns the culling sphere center
     float AACullRadius() const{return this->CullRadius();};
     // Returns the culling sphere radius
     void AAFindCull(AAVector* rmin=NULL, AAVector* rmax=NULL, AAGMatrix* mat=NULL,
BOOL UpdateCull=TRUE)
      {
          vtVector* vmin = new vtVector((vtVector)((AAVTreeVector*)rmin)->vector);
          vtVector* vmax = new vtVector((vtVector)((AAVTreeVector*)rmax)->vector);
          vtGMatrix* m = new vtGMatrix((vtGMatrix)((AAVTreeGMatrix*)mat)->matrix);
          this->FindCull(vmin, vmax, m, UpdateCull);
      };
     // Recomputes the geometry limits and, optionally, sets the culling sphere
parameters

    // Transform method
     void AATransform(const AAGMatrix* mat){this-
>Transform((vtGMatrix)((AAVTreeGMatrix*)mat)->matrix);};
     // Transforms the geometry using the given matrix.  Normals are only rotated
and texture coordinates
```

```
        // are unaffected
     void AATexTransform(const AAGMatrix* mat){this-
>TexTransform((vtGMatrix)((AAVTreeGMatrix*)mat)->matrix);};
     // Transforms the texture coordinates using the given matrix.

    // Miscellaneous methods
//     void AACountPolys(AAPolyCountP p){};
     // Compute statistics for the node by counting vertices, etc. for the geometry

      /*****************************AAVTreeNode Methods*****************/
     void AAAddChild(AANode* node){this->AddChild((vtNode*)((AAVTreeNode*)node));};
     void AAAddNext(AANode* node){this->AddNext((vtNode*)((AAVTreeNode*)node));};
     BOOL AAChangeMaterial(int oldIdx, int newIdx){return this-
>ChangeMaterial(oldIdx, newIdx);};
     BOOL AAChangeTexture(int oldIdx, int newIdx){return this-
>ChangeMaterial(oldIdx, newIdx);};
     AANode* AAChild(){return (AAVTreeNode*)this->Child();};
     BOOL AADelChild(AANode* node){return this-
>DelChild((vtNode*)((AAVTreeNode*)node));};
     BOOL AADelChildren(){return this->DelChildren();};
     BOOL AADelNext(AANode* node){return this-
>DelNext((vtNode*)((AAVTreeNode*)node));};
     BOOL AADelNexts(){return this->DelNexts();};
     AANode* AANext(){return (AAVTreeNode*)this->Next();};
     int AANumRefs(){return this->NumRefs();};
     AANode* AAReadTree(char* path){return (AAVTreeNode*)vtReadTree(path);};
     int AARef(){return this->Ref();}
     BOOL AARemoveChild(AANode* node){return this-
>RemoveChild((vtNode*)((AAVTreeNode*)node));};
     AANode* AARemoveChildren(){return (AAVTreeNode*)this->RemoveChildren();};
     BOOL AARemoveNext(AANode* node){return this-
>RemoveNext((vtNode*)((AAVTreeNode*)node));};
     AANode* AARemoveNexts(){return (AAVTreeNode*)this->RemoveNexts();};
     void AASetChild(AANode* node){this->SetChild((vtNode*)((AAVTreeNode*)node));};
     void AASetName(char* name){this->SetName(name);};
     void AASetNext(AANode* node){this->SetNext((vtNode*)((AAVTreeNode*)node));};
     int AAUnref(){return this->Unref();};
};
```

```
// AAVTreeMaterial.h : Defines the adapter of the material class.
//
#define __AAVTreeMaterial__

#include "vtMaterial.h"

#ifndef __AAMaterial__
#include "AAMaterial.h"
#endif

#ifndef __AAVTreeVector__
#include "AAVTreeVector.h"
#endif
#ifndef __AAVTreeColor__
#include "AAVTreeColor.h"
#endif

class AAVTreeMaterial : public AAMaterial , private vtMaterial
{
public:
    // Constructors

    AAVTreeMaterial(const char *name = NULL,
          const float alpha = 1.0,
          const AAVector *ambient = NULL,
          const AAVector *diffuse = NULL,
          const AAVector *specular = NULL,
          const AAVector *emission = NULL,
          const float shininess = 0.0){};

    // Define and Update operations
    void AADefine(){this->Define();};
    BOOL AAUpdate(){return this->Update();};

    // Index methods

    void AASetIndex(const short index){this->SetIndex(index);};
     // Sets the material index
    short AAIndex() const {return this->Index();};
     // Returns the material index

    // Alpha methods

    void AASetAlpha(const float alpha){this->SetAlpha(alpha);};
     // Sets the material alpha
    float AAAlpha() const{return this->Alpha();};
     // Returns the material alpha

    // Ambient methods

    void AASetAmbient(const float r, const float g, const float b){this-
>SetAmbient(r, g, b);};
     // Sets the material ambient color
    void AASetAmbient(const AAVector* v){this-
>SetAmbient((vtVector)((AAVTreeVector*)v)->vector);};
     // Sets the material ambient color (vtVector version)
    void AASetAmbient(const AAColor* c){this-
>SetAmbient((vtColor)((AAVTreeColor*)c)->color);};
     // Sets the material ambient color (vtColor version)
    void AAGetAmbient(AAColor* c){this->GetAmbient((vtColor)((AAVTreeColor*)c)-
>color);};
     // Gets the material ambient color (vtColor version)
    const AAVector* AAAmbient() const
     {
          AAVTreeVector* v = new AAVTreeVector();
          v->vector = this->Ambient();
          return v;
     };
```

```
     // Returns the material ambient color as a vector

    // Diffuse methods

    void AASetDiffuse(const float r, const float g, const float b){this-
>SetDiffuse(r, g, b);};
     // Sets the material diffuse color
    void AASetDiffuse(const AAVector* v){this-
>SetDiffuse((vtVector)((AAVTreeVector*)v)->vector);};
     // Sets the material diffuse color (vtVector version)
    void AASetDiffuse(const AAColor* c){this-
>SetDiffuse((vtColor)((AAVTreeColor*)c)->color);};
     // Sets the material diffuse color (vtColor version)
    void AAGetDiffuse(AAColor* c){this->GetDiffuse((vtColor)((AAVTreeColor*)c)-
>color);};
     // Gets the material diffuse color (vtColor version)
    const AAVector* AADiffuse() const
     {
          AAVTreeVector* v = new AAVTreeVector();
          v->vector = this->Diffuse();
          return v;
     };
     // Returns the material diffuse color as a vector

    // Specular methods

    void AASetSpecular(const float r, const float g, const float b){this-
>SetSpecular(r, g, b);};
     // Sets the material specular color
    void AASetSpecular(const AAVector* v){this-
>SetSpecular((vtVector)((AAVTreeVector*)v)->vector);};
     // Sets the material specular color (vtVector version)
    void AASetSpecular(const AAColor* c){this-
>SetSpecular((vtColor)((AAVTreeColor*)c)->color);};
     // Sets the material specular color (vtColor version)
    void AAGetSpecular(AAColor* c){this->GetSpecular((vtColor)((AAVTreeColor*)c)-
>color);};
     // Gets the material specular color (vtColor version)
    const AAVector* AASpecular() const
     {
          AAVTreeVector* v = new AAVTreeVector();
          v->vector = this->Specular();
          return v;
     };
     // Returns the material specular color as a vector

    // Emission methods

    void AASetEmission(const float r, const float g, const float b){this-
>SetEmission(r, g, b);};
     // Sets the material emission color
    void AASetEmission(const AAVector* v){this-
>SetEmission((vtVector)((AAVTreeVector*)v)->vector);};
     // Sets the material emission color (vtVector version)
    void AASetEmission(const AAColor* c){this-
>SetEmission((vtColor)((AAVTreeColor*)c)->color);};
     // Sets the material emission color (vtColor version)
    void AAGetEmission(AAColor* c){this->GetEmission((vtColor)((AAVTreeColor*)c)-
>color);};
     // Gets the material emission color (vtColor version)
    const AAVector* AAEmission() const
     {
          AAVTreeVector* v = new AAVTreeVector();
          v->vector = this->Emission();
          return v;
     };
     // Returns the material emission color as a vector
```

```
    // Shininess methods

    void AASetShininess(const float s){this->SetShininess(s);};
     // Sets the material shininess
    float AAShininess() const{return this->Shininess();};
     // Returns the material shininess

    // Dirty bits methods

    void AASetDirty(const unsigned int which){this->SetDirty(which);};
     // SetDirty
    void AAResetDirty(const unsigned int which){this->ResetDirty(which);};
     // ResetDirty
    unsigned int AAIsDirty(const int which = AAAllBits){return this-
>IsDirty(which);};
     // IsDirty
    void AASetClean(){this->SetClean();};
     // SetClean
};
```

```
// AAVTreeNode.h : Defines the node class adapter.
//
#define __AAVTreeNode__
#ifndef __AANode__
#include "AANode.h"
#endif
#include "vtnode.h"

class AAVTreeNode : public AANode, private vtNode
{
public:
     AAVTreeNode(char* string, vtNode* node) : vtNode(string, NULL, node){};
     void AAAddChild(AANode* node){this->AddChild((vtNode*)((AAVTreeNode*)node));};
     void AAAddNext(AANode* node){this->AddNext((vtNode*)((AAVTreeNode*)node));};
     BOOL AAChangeMaterial(int oldIdx, int newIdx){return this-
>ChangeMaterial(oldIdx, newIdx);};
     BOOL AAChangeTexture(int oldIdx, int newIdx){return this-
>ChangeMaterial(oldIdx, newIdx);};
     AANode* AAChild(){return (AAVTreeNode*)this->Child();};
     BOOL AADelChild(AANode* node){return this-
>DelChild((vtNode*)((AAVTreeNode*)node));};
     BOOL AADelChildren(){return this->DelChildren();};
     BOOL AADelNext(AANode* node){return this-
>DelNext((vtNode*)((AAVTreeNode*)node));};
     BOOL AADelNexts(){return this->DelNexts();};
     AANode* AANext(){return (AAVTreeNode*)this->Next();};
     int AANumRefs(){return this->NumRefs();};
     AANode* AAReadTree(char* path){return (AAVTreeNode*)vtReadTree(path);};
     int AARef(){return this->Ref();}
     BOOL AARemoveChild(AANode* node){return this-
>RemoveChild((vtNode*)((AAVTreeNode*)node));};
     AANode* AARemoveChildren(){return (AAVTreeNode*)this->RemoveChildren();};
     BOOL AARemoveNext(AANode* node){return this-
>RemoveNext((vtNode*)((AAVTreeNode*)node));};
     AANode* AARemoveNexts(){return (AAVTreeNode*)this->RemoveNexts();};
     void AASetChild(AANode* node){this->SetChild((vtNode*)((AAVTreeNode*)node));};
     void AASetName(char* name){this->SetName(name);};
     void AASetNext(AANode* node){this->SetNext((vtNode*)((AAVTreeNode*)node));};
     int AAUnref(){return this->Unref();};
private:
};
```

```
// AAVTreeScene.h : Defines the scene class adapter.
//
#define __AAVTreeScene__
#ifndef __AAScene__
#include "AAScene.h"
#endif
#ifndef __AAVTreeEntity__
#include "AAVTreeEntity.h"
#endif
#ifndef __AAVTreeNode__
#include "AAVTreeNode.h"
#endif

#include "vtscene.h"

class AAVTreeScene : public AAScene, private vtScene
{
public:
     AAVTreeScene(char* string) : vtScene(string){};

     AAEntity* AAAddEntity(AAEntity* entity){return (AAVTreeEntity*)this-
>AddEntity((vtEntity*)((AAVTreeEntity*)entity));};

     BOOL AARemoveEntity(char* name){return this->RemoveEntity(name);};

     BOOL AARemoveEntity(AAEntity* entity){return this-
>RemoveEntity((vtEntity*)((AAVTreeEntity*)entity));};

     AAEntity* AALoad(char* name)
     {
          //vtNode* graph = vtReadTree(name);
          //AAVTreeEntity* entity = new AAVTreeEntity(name, graph);
          //return entity;
          AAVTreeEntity* entity = (AAVTreeEntity*)this->Load(name);
          return entity;
     };

private:
};
```

```
// AAVTreeTexture.h : Defines the adapter of the texture class.
//
#define __AAVTreeTexture__

#ifndef __AATexture__
#include "AATexture.h"
#endif
#ifndef __AAVTreeColor__
#include "AAVTreeColor.h"
#endif
#ifndef __AAVTreeGMatrix__
#include "AAVTreeGMatrix.h"
#endif

class AAVTreeTexture
{
public:

    static unsigned int AAComputeTextureSize(  const int usize,const int vsize,
                          const int ncomp,const BOOL isSVI=FALSE) = 0;
     // Computes the number of bytes required to store the an image with the
     // given attributes
    static char *AAFindTextureFile(const char *name) = 0;
     // Locates a texture file using the VTree texture path

    static AATexture *AALoadTexture(const char *name) = 0;
        // Read a texture file and add it to the global texture list

public:
    // Constructors

    AATexture(  const char *name = NULL,
           const unsigned int flags = 0,
           const unsigned int numComponents = AATEX_Comp_RGB,
           unsigned int *imageData = NULL,
           const int imageSize = 0,
           const int udim = 0,
           const int vdim = 0,
           float *props = NULL,
           const int numProps = 0,
           const char *fileName = NULL,
           const BOOL inPalette = FALSE,
           const AAIVector *position = NULL) = 0;

    // Texture definitions.

    BOOL AADefine() = 0;
     // Defines the texture (loads into OpenGL)
    BOOL AAUpdate() = 0;
     // Updates the texture, redefining it if the attributes have changed

    // Copy.

    void AACopy(const AATexture &M) = 0;
     // Copies the given texture, "M", into this object

    // FileName methods

    void AASetFileName(const char *n) = 0;
     // Sets the file name to "n"
    const char *AAFileName() = 0;
     // Returns the file name

    // Index methods.

    void AASetIndex(const int index) = 0;
     // Sets the texture index to the given index.
    void AAAssignIndex(const int index) = 0;
```

```
 // Assigns the texture index without regard to OpenGL display list management
int AAIndex() const = 0;
 // Returns the texture index

// Flags methods.

void AAAssignFlags(const unsigned int f) = 0;
 // Forces the flags to "f"
void AASetFlags(const unsigned int f) = 0;
 // Sets flags corresponding to bits in "f"
void AAResetFlags(const unsigned int f) = 0;
 // Resets (to zero) flags corresponding to bits in "f"
void AAClearFlags() = 0;
 // Clears all flags
unsigned int AAFlags() const = 0;
 // Returns the flags

// NumComponents methods.

void AASetNumComponents(const unsigned int n) = 0;
 // Sets the number of image components to "n"
unsigned int AANumComponents() const = 0;
 // Returns the number of image components

// Image methods.

void AASetImage(unsigned int *image,const int size) = 0;
 // Sets the image memory and size
unsigned int *AAImage() = 0;
 // Returns the image memory
AAImageBase *AAImageBase() = 0;
 // Return a pointer to the image base

// Texel color methods.

BOOL AAGetTexelColor(vtPackColor &col, const int u,
                const int v) = 0;
 // Retrieves the color of the texel at coord u, v and places
 // it in "col."  Returns TRUE if successful, FALSE if not.
BOOL AAGetTexelColor(vtColor &col, const int u,
                const int v) = 0;
 // Retrieves the color of the texel at coord u, v and places
 // it in "col."  Returns TRUE if successful, FALSE if not.
BOOL AASetTexelColor(const vtPackColor &col, const int u,
                const int v) = 0;
 // Sets the color of the texel at coord u, v to "col".
 // Returns TRUE if successful, FALSE if not.

// Size methods.

void AASetSize(const int u, const int v, const int w=0) = 0;
 // Sets the image size to "u" x "v" x "w" (w is for 3D textures)
int AAUDim() const = 0;
 // Returns the "u" dimension, in texels
int AAVDim() const = 0;
 // Returns the "v" dimension, in texels
void AAGetDim(int &u, int &v) const = 0;
 // Retrieves "u" and "v" texture dimensions

// Props methods.

void AASetProps(float *props) = 0;
 // Sets the properties array
float *AAProps() = 0;
 // Returns the properties array
void AASetNumProps(const int /* numProps */) = 0;
 // Sets the number of properties in the properties array
int AANumProps() const = 0;
```

```
     // Returns the number of properties in the properties array

    // InPalette methods.

    void AASetInPalette(const BOOL inPalette) = 0;
     // Sets the palette flag
    BOOL AAInPalette() = 0;
     // Returns the palette flag

    // Palette position methods.

    void AASetPos(const AAIVector p) = 0;
     // Sets the texture position in the palette
    void AASetPosition(const AAIVector p) = 0;
     // Sets the texture position in the palette
    void AASetPos(const AAIVector *p) = 0;
     // Sets the texture position in the palette (pointer version)
    void ASetPosition(const AAIVector *p) = 0;
     // Sets the texture position in the palette (pointer version)
    void AASetPos(const int x, const int y, const int z) = 0;
     // Sets the texture position in the palette (component version)
    void AASetPosition(const int x, const int y, const int z) = 0;
    const AAIVector &AAPos() = 0;
     // Returns the palette position
    const AAIVector &AAPosition() = 0;
     // Returns the palette position

    // Texture memory usage.

    unsigned int AAMemoryUsage() = 0;
     // Returns the number of bytes required to store this image (not including
mipmapping)

    // Dirty bits methods

    void AASetDirty(const unsigned int which) = 0;
     // Sets the "dirty" bits
    void AAResetDirty(const unsigned int which) = 0;
     // Resets (to zero) the "dirty" bits
    unsigned int AAIsDirty(const unsigned int which = AllBits) = 0;
     // Returns TRUE if all bits of "which" are set in the "dirty" bits flags
    void AASetClean() = 0;
     // Clears (to zero) all dirty bits

    // I/O.

    BOOL AALoadFile(const char *fileName = NULL) = 0;
     // Loads a texture file
    BOOL AARead(istream &in,unsigned int fileVer,BOOL isBinary = TRUE) = 0;
     // Restores a vtTexture from disk
    BOOL AAWrite(ostream &out,BOOL isBinary = TRUE) = 0;
     // Stores a vtTexture to disk
};
```

```
// AAVTreeTNode.h : Defines the adapter of the transformation node class.
//
#define __AAVTreeTNode__

#ifndef __AATNode__
#include "AATNode.h"
#endif
#ifndef __AAVTreeGMatrix__
#include "AAVTreeGMatrix.h"
#endif

#include "vtTNode.h"

class AAVTreeTNode : public AATNode, private vtTNode
{
public:
    AAVTreeTNode(const char *name = NULL)
          : vtTNode(name){};
     // Default constructor
 //   AAVTreeTNode(const AATNode *C);
     // Copy constructor

    // Matrix methods

     void AASetMat(const AAGMatrix* M){this->SetMat(((AAVTreeGMatrix*)M)-
>matrix);};
     // Sets the matrix to "*M" (pointer version)
     void AASetIdentity(void){this->SetIdentity();};
    // Sets the matrix to the identity matrix
     AAGMatrix* AAMat()
      {
           AAVTreeGMatrix* m = new AAVTreeGMatrix();
           m->matrix = this->Mat();
           return m;
      };
     // Returns the matrix

    // Flags methods

     void AASetFlags(const int flags){this->SetFlags(flags);};
     // Sets flags corresponding to bits of "flags"
     void AAResetFlags(const int flags){this->ResetFlags(flags);};
     // Resets (to zero) flags corresponding to bits of "flags"
     void AAClearFlags(){this->ClearFlags();};
     // Clears (to zero) all flags
     int AAFlags(){return this->Flags();};
     // Returns the flags

    // Member copy getting methods.

     AAGMatrix* AAMatCopy()
      {
           AAVTreeGMatrix* m = new AAVTreeGMatrix();
           m->matrix = this->MatCopy();
           return m;
      };
     // Returns the copy matrix
     int AAFlagsCopy(){return this->FlagsCopy();};
     // Returns the copy flags

    // Transformation methods wrt this node's coordinates.

     void AAScale(const float sx,const float sy,const float sz){this->Scale(sx, sy,
sz);};
     // Scales the matrix, anisotropically (component version)
     void AAScale(const float s){this->Scale(s);};
     // Scales the matrix, isotropically
     void AAScale(const AAVector* v){this->Scale(((AAVTreeVector*)v)->vector);};
```

```
     // Scales the matrix, anisotropically
     void AARotate(const AAAxis axis,const float rot){this-
>Rotate((vtTNode::Axis)axis, rot);};
     // Rotates about "AAAxis" through angle "rot" (radians)
     void AARotate(const AAAxis aAxis1,const float rAxis1,const AAAxis aAxis2,const
float rAxis2,const AAAxis aAxis3,const float rAxis3)
      {this->Rotate((vtTNode::Axis)aAxis1, rAxis1, (vtTNode::Axis)aAxis2, rAxis2,
(vtTNode::Axis)aAxis3, rAxis3);};
     // Rotates about the given axes, in the order specified, by the amounts speci-
fied (radians)
     void AARotateX(const float ang){this->RotateX(ang);};
     // Rotates about "X" through the given angle (radians)
     void AARotateY(const float ang){this->RotateY(ang);};
     // Rotates about "Y" through the given angle (radians)
     void AARotateZ(const float ang){this->RotateZ(ang);};
     // Rotates about "Z" through the given angle (radians)
     void AARotateXYZ(const float x,const float y,const float z){this->RotateXYZ(x,
y, z);};
     // Rotates about "X", "Y", and "Z" through the given angles (radians)
     void AARotateZXY(const float z,const float x,const float y){this->RotateZXY(z,
x, y);};
     // Rotates about "Z", "X", and "Y" through the given angles (radians)
     void AARotateYZX(const float y,const float z,const float x){this->RotateYZX(y,
z, x);};
     // Rotates about "Y", "Z", and "X" through the given angles (radians)
     void AARotateXZY(const float x,const float z,const float y){this->RotateXZY(x,
z, y);};
     // Rotates about "X", "Z", and "Y" through the given angles (radians)
     void AARotateYXZ(const float y,const float x,const float z){this->RotateYXZ(y,
x, z);};
     // Rotates about "Y", "X", and "Z" through the given angles (radians)
     void AARotateZYX(const float z,const float y,const float x){this->RotateZYX(z,
y, x);};
     // Rotates about "Z", "Y", and "X" through the given angles (radians)
     void AARotateXYZ(const AAVector *v){this->RotateXYZ(((AAVTreeVector*)v)-
>vector);};
     // Rotates about "X", "Y", and "Z" through the given angles (radians)
     void AARotateZXY(const AAVector *v){this->RotateZXY(((AAVTreeVector*)v)-
>vector);};
     // Rotates about "Z", "X", and "Y" through the given angles (radians)
     void AARotateYZX(const AAVector *v){this->RotateYZX(((AAVTreeVector*)v)-
>vector);};
     // Rotates about "Y", "Z", and "X" through the given angles (radians)
     void AARotateXZY(const AAVector *v){this->RotateXZY(((AAVTreeVector*)v)-
>vector);};
     // Rotates about "X", "Z", and "Y" through the given angles (radians)
     void AARotateYXZ(const AAVector *v){this->RotateYXZ(((AAVTreeVector*)v)-
>vector);};
     // Rotates about "Y", "X", and "Z" through the given angles (radians)
     void AARotateZYX(const AAVector *v){this->RotateZYX(((AAVTreeVector*)v)-
>vector);};
     // Rotates about "Z", "Y", and "X" through the given angles (radians)
     void AATranslate(const float x,const float y,const float z){this->Translate(x,
y, z);};
     // Translates by the given amounts (component version)
     void AATranslate(const AAVector *t){this->Translate(((AAVTreeVector*)t)-
>vector);};
     // Translates by the given amounts (vector version)

    // Transformations methods wrt this node's coordinates with the matrix being
    // cleared first.

     void AAXRotation(const float ang){this->XRotation(ang);};
     // Rotates about "X" through the given angle (radians)
     void AAYRotation(const float ang){this->YRotation(ang);};
     // Rotates about "Y" through the given angle (radians)
     void AAZRotation(const float ang){this->ZRotation(ang);};
     // Rotates about "Z" through the given angle (radians)
```

```
     void AAXYZRotation(const float x,const float y,const float z){this-
>XYZRotation(x, y, z);};
     // Rotates about "X", "Y", and "Z" through the given angles (radians)
     void AAZXYRotation(const float z,const float x,const float y){this-
>ZXYRotation(z, x, y);};
     // Rotates about "Z", "X", and "Y" through the given angles (radians)
     void AAYZXRotation(const float y,const float z,const float x){this-
>YZXRotation(y, z, x);};
     // Rotates about "Y", "Z", and "X" through the given angles (radians)
     void AAXZYRotation(const float x,const float z,const float y){this-
>XZYRotation(x, z, y);};
     // Rotates about "X", "Z", and "Y" through the given angles (radians)
     void AAYXZRotation(const float y,const float x,const float z){this-
>YXZRotation(y, x, z);};
     // Rotates about "Y", "X", and "Z" through the given angles (radians)
     void AAZYXRotation(const float z,const float y,const float x){this-
>ZYXRotation(z, y, x);};
     // Rotates about "Z", "Y", and "X" through the given angles (radians)
     void AAXYZRotation(const AAVector *v){this->XYZRotation(((AAVTreeVector*)v)-
>vector);};
     // Rotates about "X", "Y", and "Z" through the given angles (radians)
     void AAZXYRotation(const AAVector *v){this->ZXYRotation(((AAVTreeVector*)v)-
>vector);};
     // Rotates about "Z", "X", and "Y" through the given angles (radians)
     void AAYZXRotation(const AAVector *v){this->YZXRotation(((AAVTreeVector*)v)-
>vector);};
     // Rotates about "Y", "Z", and "X" through the given angles (radians)
     void AAXZYRotation(const AAVector *v){this->XZYRotation(((AAVTreeVector*)v)-
>vector);};
     // Rotates about "X", "Z", and "Y" through the given angles (radians)
     void AAYXZRotation(const AAVector *v){this->YXZRotation(((AAVTreeVector*)v)-
>vector);};
     // Rotates about "Y", "X", and "Z" through the given angles (radians)
     void AAZYXRotation(const AAVector *v){this->ZYXRotation(((AAVTreeVector*)v)-
>vector);};
     // Rotates about "Z", "Y", and "X" through the given angles (radians)
     void AATranslation(const float x,const float y,const float z){this-
>Translation(x, y, z);};
     // Translates by the given amounts (component version)
     void AATranslation(const AAVector *t){this->Translation(((AAVTreeVector*)t)-
>vector);};
     // Translates by the given amounts (vector version)


    // Transformation methods wrt this node's base coordinates.

     void AABaseScale(const float sx,const float sy,const float sz){this-
>BaseScale(sx, sy, sz);};
     // Base scale the matrix, anisotropically (component version)
     void AABaseScale(const float s){this->BaseScale(s);};
     // Base scale the matrix, isotropically
     void AABaseScale(const AAVector *v){this->BaseScale(((AAVTreeVector*)v)-
>vector);};
     // Base scale the matrix, anisotropically
     void AABaseRotate(const AAAxis axis,const float rot){this-
>BaseRotate((vtTNode::Axis)axis, rot);};
     // Base Base rotate about "AAAxis" through angle "rot" (radians)
      void AABaseRotate(const AAAxis aAxis1,const float rAxis1,const AAAxis
aAxis2,const float rAxis2,const AAAxis aAxis3,const float rAxis3)
      {this->BaseRotate((vtTNode::Axis)aAxis1, rAxis1, (vtTNode::Axis)aAxis2,
rAxis2, (vtTNode::Axis)aAxis3, rAxis3);};
     // Base rotate about the given axes, in the order specified, by the
     // amounts specified (radians)
     void AABaseRotateX(const float ang){this->BaseRotateX(ang);};
     // Base rotate about "X" through the given angle (radians)
     void AABaseRotateY(const float ang){this->BaseRotateY(ang);};
     // Base rotate about "Y" through the given angle (radians)
     void AABaseRotateZ(const float ang){this->BaseRotateZ(ang);};
```

```
     // Base rotate about "Z" through the given angle (radians)
     void AABaseRotateXYZ(const float x,const float y,const float z){this-
>BaseRotateXYZ(x, y, z);};
     // Base rotate about "X", "Y", and "Z" through the given angles (radians)
     void AABaseRotateZXY(const float z,const float x,const float y){this-
>BaseRotateZXY(z, x, y);};
     // Base rotate about "Z", "X", and "Y" through the given angles (radians)
     void AABaseRotateYZX(const float y,const float z,const float x){this-
>BaseRotateYZX(y, z, x);};
     // Base rotate about "Y", "Z", and "X" through the given angles (radians)
     void AABaseRotateXZY(const float x,const float z,const float y){this-
>BaseRotateXZY(x, z, y);};
     // Base rotate about "X", "Z", and "Y" through the given angles (radians)
     void AABaseRotateYXZ(const float y,const float x,const float z){this-
>BaseRotateYXZ(y, x, z);};
     // Base rotate about "Y", "X", and "Z" through the given angles (radians)
     void AABaseRotateZYX(const float z,const float y,const float x){this-
>BaseRotateZYX(z, y, x);};
     // Base rotate about "Z", "Y", and "X" through the given angles (radians)
     void AABaseRotateXYZ(const AAVector *v){this-
>BaseRotateXYZ(((AAVTreeVector*)v)->vector);};
     // Base rotate about "X", "Y", and "Z" through the given angles (radians)
     void AABaseRotateZXY(const AAVector *v){this-
>BaseRotateZXY(((AAVTreeVector*)v)->vector);};
     // Base rotate about "Z", "X", and "Y" through the given angles (radians)
     void AABaseRotateYZX(const AAVector *v){this-
>BaseRotateYZX(((AAVTreeVector*)v)->vector);};
     // Base rotate about "Y", "Z", and "X" through the given angles (radians)
     void AABaseRotateXZY(const AAVector *v){this-
>BaseRotateXZY(((AAVTreeVector*)v)->vector);};
     // Base rotate about "X", "Z", and "Y" through the given angles (radians)
     void AABaseRotateYXZ(const AAVector *v){this-
>BaseRotateYXZ(((AAVTreeVector*)v)->vector);};
     // Base rotate about "Y", "X", and "Z" through the given angles (radians)
     void AABaseRotateZYX(const AAVector *v){this-
>BaseRotateZYX(((AAVTreeVector*)v)->vector);};
     // Base rotate about "Z", "Y", and "X" through the given angles (radians)
     void AABaseTranslate(const float x,const float y,const float z){this-
>BaseTranslate(x, y, z);};
     // Base translate by the given amounts (component version)
     void AABaseTranslate(const AAVector *t){this-
>BaseTranslate(((AAVTreeVector*)t)->vector);};
     // Base translate by the given amounts (vector version)

    // Face methods

     void AAFaceX(const AAVector *p){};
     // Computes the matrix such that the +Z AAAxis comes closest to facing
     // the position "p" by rotating about the "X" AAAxis only
     void AAFaceY(const AAVector *p){};
     // Computes the matrix such that the +Z AAAxis comes closest to facing
     // the position "p" by rotating about the "Y" AAAxis only
     void AAFaceXY(const AAVector *p){};
     // Computes the matrix such that the +Z AAAxis faces the position "p"
     // by rotating about the "X" and "Y" axes

      /*************************AAVTREENODE METHODS*************************/
     void AAAddChild(AANode* node){this->AddChild((vtNode*)((AAVTreeNode*)node));};
     void AAAddNext(AANode* node){this->AddNext((vtNode*)((AAVTreeNode*)node));};
     BOOL AAChangeMaterial(int oldIdx, int newIdx){return this-
>ChangeMaterial(oldIdx, newIdx);};
     BOOL AAChangeTexture(int oldIdx, int newIdx){return this-
>ChangeMaterial(oldIdx, newIdx);};
     AANode* AAChild(){return (AAVTreeNode*)this->Child();};
     BOOL AADelChild(AANode* node){return this-
>DelChild((vtNode*)((AAVTreeNode*)node));};
     BOOL AADelChildren(){return this->DelChildren();};
```

```
     BOOL AADelNext(AANode* node){return this-
>DelNext((vtNode*)((AAVTreeNode*)node));};
     BOOL AADelNexts(){return this->DelNexts();};
     AANode* AANext(){return (AAVTreeNode*)this->Next();};
     int AANumRefs(){return this->NumRefs();};
     AANode* AAReadTree(char* path){return (AAVTreeNode*)vtReadTree(path);};
     int AARef(){return this->Ref();}
     BOOL AARemoveChild(AANode* node){return this-
>RemoveChild((vtNode*)((AAVTreeNode*)node));};
     AANode* AARemoveChildren(){return (AAVTreeNode*)this->RemoveChildren();};
     BOOL AARemoveNext(AANode* node){return this-
>RemoveNext((vtNode*)((AAVTreeNode*)node));};
     AANode* AARemoveNexts(){return (AAVTreeNode*)this->RemoveNexts();};
     void AASetChild(AANode* node){this->SetChild((vtNode*)((AAVTreeNode*)node));};
     void AASetName(char* name){this->SetName(name);};
     void AASetNext(AANode* node){this->SetNext((vtNode*)((AAVTreeNode*)node));};
     int AAUnref(){return this->Unref();};
protected:
};
```

```
// AAVTreeVector.h : Defines the adapter of the vector class.
//
#define __AAVTreeVector__
#include "vtBaseDefs.h"

#ifndef __AAVector__
#include "AAVector.h"
#endif

class AAVTreeVector : public AAVector
{
friend class AAVTreeVector;
friend class AAVTreeGMatrix;
friend class AAVTreeEntity;
friend class AAVTreeViewport;
friend class AAVTree2DGeomGaugeNode;
friend class AAVTreeGaugeNode;
friend class AAVTreeFxDebris;
friend class AAVTreeFxMissileTrail;
friend class AAVTreeTNode;
friend class AAVTreeIGNode;
friend class AAVTreeMaterial;
public:
     AAVTreeVector(){vtSet(vector, 0.0f, 0.0f, 0.0f);};
     AAVTreeVector(float x, float y, float z){vtSet(vector, x, y, z);};

     void AAAbs(void){vtAbs(vector);};
     void AAAbs(AAVector* base){vtAbs(vector, ((AAVTreeVector*)base)->vector);};
     AAVector* AACross(AAVector* vect1, AAVector* vect2)
     {
          vtVector v;
          vtCross(v, ((AAVTreeVector*)vect1)->vector, ((AAVTreeVector*)vect2)-
>vector);
          return new AAVTreeVector(v.x, v.y, v.z);
     };
     float AADist(AAVector* other)
     {return vtDist(vector, ((AAVTreeVector*)other)->vector);};
     float AADist2(AAVector* other)
     {return vtDist2(vector, ((AAVTreeVector*)other)->vector);};
     float AAGetX(){return vector.x;};
     float AAGetY(){return vector.y;};
     float AAGetZ(){return vector.z;};
     float AAInnerProd(AAVector* other)
     {return vtInnerProd(vector, ((AAVTreeVector*)other)->vector);};
     float AAMax()
     {return vtMax(vector);};
     float AAMagnitude()
     {return vtMagnitude(vector);};
     float AAMagSq()
     {return vtMagSq(vector);};
     float AAMin()
     {return vtMin(vector);};
     void AANegate()
     {vtNegate(vector);};
     void AANormalize()
     {vtNormalize(vector);};
     void AANormalize(AAVector* base)
     {vtNormalize(vector, ((AAVTreeVector*)base)->vector);};
     void AAScale(float s)
     {vtScale(vector, s);};
     void AAScale(float x, float y, float z)
     {vtScale(vector, x, y, z);};
     void AASet(float x, float y, float z)
     {vtSet(vector, x, y, z);};
     void AASet(AAVector* other)
     {vtSet(vector, ((AAVTreeVector*)other)->vector);};
     void AASetX(const float x){vector.x = x;};
     void AASetY(const float y){vector.y = y;};
```

```
	void AASetZ(const float z){vector.z = z;};
	AAVector* operator+(AAVector* other)
	{
		vtVector v;
		vtAdd(v, vector, ((AAVTreeVector*)other)->vector);
		return new AAVTreeVector(v.x, v.y, v.z);
	};
	AAVector* operator-(AAVector* other)
	{
		vtVector v;
		vtSub(v, vector, ((AAVTreeVector*)other)->vector);
		return new AAVTreeVector(v.x, v.y, v.z);
	};
	AAVector* operator*(AAVector* other)
	{
		vtVector v;
		vtElemMul(v, vector, ((AAVTreeVector*)other)->vector);
		return new AAVTreeVector(v.x, v.y, v.z);
	};
	AAVector* operator*(float f)
	{
		vtVector v;
		vtSet(v, vector);
		vtScale(v, f);
		return new AAVTreeVector(v.x, v.y, v.z);
	};
	AAVector* operator=(AAVector* other)
	{
		if (this == NULL)
			return new AAVTreeVector(((AAVTreeVector*)other)->vector.x,
((AAVTreeVector*)other)->vector.y, ((AAVTreeVector*)other)->vector.z);
		else
			vtSet(vector, ((AAVTreeVector*)other)->vector);
		return this;
	};
	BOOL operator==(AAVector* other)
	{
		if ((vector.x == ((AAVTreeVector*)other)->vector.x) &&
			(vector.y == ((AAVTreeVector*)other)->vector.y) &&
			(vector.z == ((AAVTreeVector*)other)->vector.z))
			return TRUE;
		return FALSE;
	};
	BOOL operator!=(AAVector* other)
	{
		if (this == other) return FALSE;
		return TRUE;
	};
	float operator[](int n)
	{
		if ((0 <= n) && (n < 3))
			return vector[n];
		return 0.0f;
	};
	const float operator[](int n) const
	{
		if ((0 <= n) && (n < 3))
			return vector[n];
		return 0.0f;
	};

protected:
	vtVector vector;
private:
};
```

```
// AAVTreeViewport.h : Defines the viewport class adapter.
//
#define __AAVTreeViewport__
#include "vtviewport.h"

#ifndef __AAVector__
#include "AAVector.h"
#endif
#ifndef __AAScene__
#include "AAScene.h"
#endif
#ifndef __AAViewport__
#include "AAViewport.h"
#endif
#ifndef __AAVTreeScene__
#include "AAVTreeScene.h"
#endif
#ifndef __AAVTreeVector__
#include "AAVTreeVector.h"
#endif
#ifndef __AAVTreeGMatrix__
#include "AAVTreeGMatrix.h"
#endif

class AAVTreeViewport : public AAViewport, private vtViewport
{
public:
     AAVTreeViewport(vtWindow* wnd, char* string, int posx, int posy, int width,
int height)
     : vtViewport(wnd, string, posx, posy, width, height){};

     BOOL AAAllocStencilBits(const int num){return this->AllocStencilBits(num);};

     void AAClearFlags(){this->ClearFlags();};

     void AADisable(const int what){this->Disable(what);};

     void AAEnable(const int what){this->Enable(what);};

     void AAEnableStencil(){this->EnableStencil();};

     void AAFogDisable(){this->FogDisable();};

     void AAFogEnable(){this->FogEnable();};

     void AAFogSetColor(const float r, const float g, const float b, const float a)
     {this->FogSetColor(r, g, b, a);};

     void AAFogSetDensity(const float d){this->FogSetDensity(d);};

     void AAFogSetLinearRange(const float min, const float max)
     {this->FogSetLinearRange(min, max);};

     void AAFogSetMode(FOGMODE mode)
     {
          switch(mode)
          {
          case LINEAR:
               this->FogSetMode(vtFog::LinearFog);
               break;
          case EXP:
               this->FogSetMode(vtFog::ExpFog);
               break;
          case EXPSQR:
               this->FogSetMode(vtFog::Exp2Fog);
               break;
          case TABLE:
               if (this->FogMaxTableSize() > 0)
```

```
                    this->FogSetMode(vtFog::TableFog);
               break;
          }
     };

     int AANumStencilBits(){return this->NumStencilBits();};

     void AAPrepareForRender(){this->PrepareForRender();};

     AAScene* AAGetScene(){return (AAVTreeScene*)this->Scene();};

     void AASetColor(const float r, const float g, const float b, const float a)
     {this->SetColor(r, g, b, a);};

     void AASetFlags(const FLAGS f)
     {
          switch(f)
          {
          case DRAWENABLE:
               this->SetFlags(DrawEnable);
               break;
          case CLEARPLANES:
               this->SetFlags(ClearPlanes);
               break;
          case CLEARDEPTH:
               this->SetFlags(ClearDepth);
               break;
          case CLEARSTENCIL:
               this->SetFlags(ClearStencil);
               break;
          case STENCILENABLE:
               this->SetFlags(StencilEnable);
               break;
          case FLIPVERTIVAL:
               this->SetFlags(FlipVertical);
               break;
          case FLIPHORIZONTAL:
               this->SetFlags(FlipHorizontal);
               break;
          }
     };
     void AASetOrtho(const float xmin,const float xmax,
                                  const float ymin,const float ymax,
                                  const float zmin,const float zmax)
     {this->SetOrtho(     xmin, xmax, ymin, ymax, zmin, zmax);};

     void AASetPerspective(const float vertfov, const float asprat, const float
nearclip, const float farclip)
     {this->SetPerspective(vtDToR(vertfov), asprat, nearclip, farclip);};

     void AASetScene(AAScene* scene)
     {this->SetScene((vtScene*)((AAVTreeScene*)scene));};

     void AASetViewLookAroundAt(AAVector* at, float deltaHorz, float deltaVert,
float deltaRoll, float dist)
     {this->SetViewLookAroundAt(((AAVTreeVector*)at)->vector, deltaHorz, deltaVert,
deltaRoll, dist);};

     void AASetViewLookFromTo(const float fx, const float fy , const float fz,
const float tx,
     const float ty, const float tz, const float ux, const float uy, const float
uz)
     {this->SetViewLookFromTo(fx, fy, fz, tx, ty, tz, ux, uy, uz);};

     void AASetViewLookOrbit(AAVector* at, float azim, float elev, float dist)
     {this->SetViewLookOrbit(((AAVTreeVector*)at)->vector, azim, elev, dist);};
```

```
     void AASetViewMat(AAGMatrix* matrix){this-
>SetViewMat(((AAVTreeGMatrix*)matrix)->matrix);};

     void AAResetFlags(const FLAGS f)
     {
          switch(f)
          {
          case DRAWENABLE:
               this->ResetFlags(DrawEnable);
               break;
          case CLEARPLANES:
               this->ResetFlags(ClearPlanes);
               break;
          case CLEARDEPTH:
               this->ResetFlags(ClearDepth);
               break;
          case CLEARSTENCIL:
               this->ResetFlags(ClearStencil);
               break;
          case STENCILENABLE:
               this->ResetFlags(StencilEnable);
               break;
          case FLIPVERTIVAL:
               this->ResetFlags(FlipVertical);
               break;
          case FLIPHORIZONTAL:
               this->ResetFlags(FlipHorizontal);
               break;
          }
     };

     void AASyncEnables(){this->SyncEnables();};
private:
};
```

```
// AAVTreeWindow.h : Defines the window class adapter.
//
#define __AAVTreeWindow__
#ifndef __AAVTreeViewport__
#include "AAVTreeViewport.h"
#endif
#ifndef __AAWindow__
#include "AAWindow.h"
#endif
#include "vtwindow.h"
#include "vtwin32.h"
#include "vtinit.h"


class AAVTreeWindow : public AAWindow, private vtWindow
{
public:
     AAVTreeWindow(char* string, HWND hWnd, int posx, int posy, int width, int
height)
     : vtWindow(string, hWnd, posx, posy, width, height){};

     BOOL AAAllocStencilBits(const int num, unsigned int bit0){return this-
>AllocStencilBits(num, bit0);};

     float AAAspect(){return this->Aspect();};
     int AAHeight(){return this->Height();};
     int AAWidth(){return this->Width();};
     int AAX(){return this->X();};
     int AAY(){return this->Y();};

     AAViewport* AAAddViewport(AAViewport* vp){return (AAVTreeViewport*)this-
>AddViewport((vtViewport*)((AAVTreeViewport*)vp));};

     AAViewport* AAFindViewport(char* name){return (AAVTreeViewport*)this-
>FindViewport(name);};

     void AADisable(ENABLES sw)
     {
          switch (sw)
          {
          case ANTIALIAS:
               this->Disable(vtEnableState::Antialias);
               break;
          case BACKFACECULL:
               this->Disable(vtEnableState::BackfaceCull);
               break;
          case CULLSPHERES:
               this->Disable(vtEnableState::CullSpheres);
               break;
          case DEPTHTEST:
               this->Disable(vtEnableState::DepthTest);
               break;
          case FOG:
               this->Disable(vtEnableState::Fog);
               break;
          case LIGHT1:
               this->Disable(vtEnableState::Light1);
               break;
          case LIGHT2:
               this->Disable(vtEnableState::Light2);
               break;
          case LIGHT3:
               this->Disable(vtEnableState::Light3);
               break;
          case LIGHT4:
               this->Disable(vtEnableState::Light4);
               break;
          case LIGHT5:
```

```
            this->Disable(vtEnableState::Light5);
            break;
      case LIGHT6:
            this->Disable(vtEnableState::Light6);
            break;
      case LIGHT7:
            this->Disable(vtEnableState::Light7);
            break;
      case LIGHTNING:
            this->Disable(vtEnableState::Lighting);
            break;
      case MIPMAPTEXTURES:
            this->Disable(vtEnableState::MipmapTextures);
            break;
      case OBJECTCULL:
            this->Disable(vtEnableState::ObjectCull);
            break;
      case SMOOTHSHADE:
            this->Disable(vtEnableState::SmoothShade);
            break;
      case SUBFACE:
            this->Disable(vtEnableState::Subface);
            break;
      case TEXTURE:
            this->Disable(vtEnableState::Texture);
            break;
      case TRANSPARENCY:
            this->Disable(vtEnableState::Transparency);
            break;
      }
};

void AAEnable(ENABLES sw)
{
      switch (sw)
      {
      case ANTIALIAS:
            this->Enable(vtEnableState::Antialias);
            break;
      case BACKFACECULL:
            this->Enable(vtEnableState::BackfaceCull);
            break;
      case CULLSPHERES:
            this->Enable(vtEnableState::CullSpheres);
            break;
      case DEPTHTEST:
            this->Enable(vtEnableState::DepthTest);
            break;
      case FOG:
            this->Enable(vtEnableState::Fog);
            break;
      case LIGHT1:
            this->Enable(vtEnableState::Light1);
            break;
      case LIGHT2:
            this->Enable(vtEnableState::Light2);
            break;
      case LIGHT3:
            this->Enable(vtEnableState::Light3);
            break;
      case LIGHT4:
            this->Enable(vtEnableState::Light4);
            break;
      case LIGHT5:
            this->Enable(vtEnableState::Light5);
            break;
      case LIGHT6:
            this->Enable(vtEnableState::Light6);
```

```
                break;
           case LIGHT7:
                this->Enable(vtEnableState::Light7);
                break;
           case LIGHTNING:
                this->Enable(vtEnableState::Lighting);
                break;
           case MIPMAPTEXTURES:
                this->Enable(vtEnableState::MipmapTextures);
                break;
           case OBJECTCULL:
                this->Enable(vtEnableState::ObjectCull);
                break;
           case SMOOTHSHADE:
                this->Enable(vtEnableState::SmoothShade);
                break;
           case SUBFACE:
                this->Enable(vtEnableState::Subface);
                break;
           case TEXTURE:
                this->Enable(vtEnableState::Texture);
                break;
           case TRANSPARENCY:
                this->Enable(vtEnableState::Transparency);
                break;
           }
      };

      BOOL AAFrame(){return this->Frame();};

      float AAFrameRate(){return this->FrameRate();};

      BOOL AAPrepareForRender()
      {
           this->Enables().ApplyAll();
           return this->PrepareForRender();
      };

      BOOL AARemoveViewport(AAViewport* vp){return this-
>RemoveViewport((vtViewport*)((AAVTreeViewport*)vp));};

      BOOL AARender(){return this->Render();};

protected:
private:
};
```

D Booch Notation

The Booch notation is used to visualize some class diagrams and schemes. This is a short description of the notation.

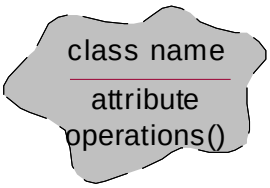

Figure 0.1 - Class

Attributes:

A	Attribute name only
: C	Attribute class only
A : C	Attribute name and class
A : C = E	Attribute name, class and default expression

Table 0.1 - Attributes

Methods:

N()	Operation name only
R N(Arg)	Operation return class, name and formal arguments

Table 0.2 - Methods

Export control (for attributes/operations and relationships)**:**

	Public
|	Protected
||	Private
|||	implementation

Table 0.3 - Export control

Properties (for relationships)**:**

A	abstract class
F	Friend
S	Static
V	Virtual

Table 0.4 - Properties

Class relationships (can be refined with properties)**:**

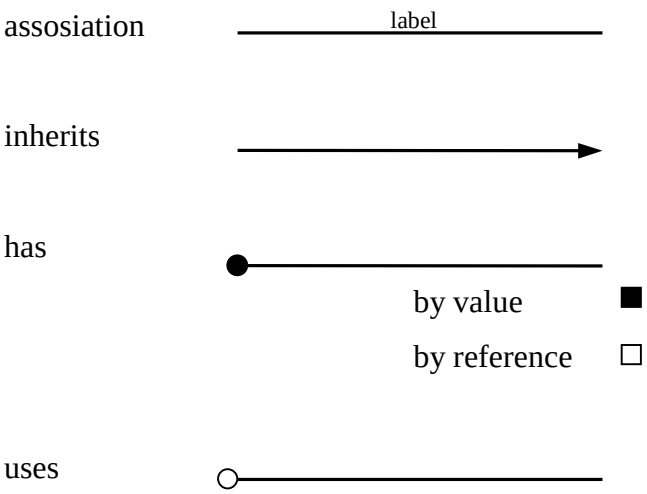

Figure 0.2 - Class relationships

Cardinality:

1	exactly one
n	unlimited number (zero or more)
0 .. n	zero or more
1 .. n	one or more
0 .. 1	zero or one
3 .. 7	specified range
1 .. 3, 7	specified range or exact number

Table 0.5 - Cardinality