ABSTRACT
This paper discusses RAGE – a Java based open source game engine project. The RAGE project aims to abstract the various aspects of game engine functionality into a set of generic interfaces. Platform or API specific implementations of these interfaces then allow applications written for the RAGE engine to run without modification or recompilation on any of the target hardware/operating system configurations.

RAGE: Rage is A Game Engine (http://silica.csu.edu.au/rage/)

1. INTRODUCTION
The Computer Game industry is now a major industry that generates significant income annually. For example, in the United States the 2004 domestic Video and Computer Game Industry was estimated to be around US$10 billion. (Yi, 2004) Indeed, since 2000 the game industry has been recognised as one of the largest growth sectors of the international economy (Vogel, 2001) The worldwide market is estimated to be US$30 billion dollars by 2007 (RocSearch, 2005).

Traditionally computer games are considered relevant only to the entertainment sector. However, there has been a growing acceptance of the idea that the games can play an important role in teaching basic curriculum skills, such as, reading, writing and arithmetic (Crawford, 1984). It has also been recognised that games can assist players develop higher level cognitive functions such as decision making and leadership (Crawford, 1984).

The Australian games industry earned an estimated AUS$100 million dollars in export revenue in 2002 (Howarth, 2004). Despite the growth of the game industry, competition is intense and Australia's survival in the market place is dependent on product innovation and keeping abreast of rapidly changing, opportunities presented by new game technology. Opportunities include the use of multiplatform systems and multi-processor computing architectures to support more life-like simulations in games.

In order to mitigate risk and facilitate rapid application development, game developers are increasingly turning to the methods of software engineering (Rucker, 2003). These methods revolve around the production of modular, reusable code and the application of software development processes. In game development this has resulted in the widespread use of software frameworks referred to as game engines.

Game engines are reusable software frameworks designed to support real-time interactive simulations. Game engines perform all the generic functions of a game, such as loading resources, and executing the game loop. The game loop represents the work the computer must perform to generate a single scene or frame to the player (see Figure 1). The game loop consists of dealing with user input, updating the game world, and rendering the game world to screen. Of these processes, updating the game world, and rendering it to screen are extremely CPU intensive tasks. Updating the game world involves applying artificial intelligence to all the active elements (or game objects) of the game world, evaluating the effects of any interactions such as collisions or conflicts, and then calculating new positions and states for all game objects. Rendering involves selecting game objects visible to the player, applying textures to models, calculating lighting (and sound) effects, and then delivering possibly hundreds of thousands of graphics primitives to the graphics card for display each screen refresh cycle.

Figure 1. The basic tasks of the game engine when performing a scene update or game loop.

2. THE RAGE GAME ENGINE
The RAGE project seeks to decompose the functionality of a game engine into a set of independent modular components. These components are divided into a low level toolkit of ‘core’ and ‘extension’ modules, and a higher level game programming framework.

The low level ‘core’ components consist of abstractions for basic hardware and operating system (OS) services or APIs such as graphics, sound, input, and timers. It also includes utility functions such as maths libraries, and a game ‘kernel’ which provides property management, resource loading and a basic scheduling system for the game loop.

‘Extension’ components also abstract basic OS services or APIs but are considered optional rather than required functionality. They include modules to support platform independent fonts, networking, physics, scripting and artificial intelligence (AI).

Framework components provide high level support for game programming. They include modules such as a scene management system, a menu system, loaders for particular types of mesh file formats, or animation file formats, and a game event system.
2.1 Core Toolkit Components

At the centre of the toolkit is the Kernel. Any RAGE application contains a single kernel which provides hardware timers, a single-threaded task system, resource management (local file and http) and property management. RAGE applications are constructed as a number of tasks. Each task has an update method which is called once per iteration through the game loop. The order of update is controlled through a priority value.

Resource loaders take images, models, and animations created by third party authoring tools and transform them into RAGE’s internal data format. Currently RAGE has loaders for Quake3, HalfLife, and Warcraft3 models which give it access to a huge range of publically accessible 3D content.

The input system abstracts all input device into a generic InputDevice which is considered to consist of a number of buttons, axes, and ‘directionals’. Buttons are simple binary state components, axes can take on real value while directionals can take on a limited number of discrete values. While all input devices can be described using these generic components, for programming convenience the keyboard and mouse are provided with specialized input device classes. The input system runs as a task which polls input device hardware once per game loop. An input event system is layered on top of the basic polling function.

The rendering system is an abstraction of both DirectX and OpenGL fixed function pipelines. It is a low level rendering API which provides a ‘thin’ adapter to platform specific rendering APIs. The rendering system allows the creation of custom ‘materials’ which allow specialized rendering effects such as environment mapping or cell shading. (see Figure 2) Figure 2. Specialized rendering effects using customizable materials. Left: specular and diffuse highlighting. Right: Cell shading

Support for programmable shaders is planned for the near future.

The sound system is based closely on the OpenAL specification. This gives RAGE 3-D multiple sound sources and positional sound capabilities with panning and Doppler effects. Streaming sound support is planned for the near future.

The most important of the framework components is the scenegraph. The scenegraph is critical to game engine performance as it supports rapid processing of the game world to select relevant elements for a variety of processing such as rendering, sound processing, AI processing, collision detection and level of detail processing. The RAGE scenegraph is based on the notion of a ‘View’ which is a specialized subset of the total scenegraph relevant to a specific type of processing.

RAGE also implements a menuing system which is critical in providing a game support infrastructure such as startup screens, a main menu, network configuration and lobby screens, hardware capability configuration screens, high score lists, save/load functionality and other non-game functionality. The menuing system is based on the well understood window paradigm and is derived largely from the Swing and Windows foundation classes.

3. DISCUSSION

The RAGE project has not sought to break new ground in the definition and implementation of these modules. Rather, it has sought to understand the functionality common to a wide range of game engines and produce abstract interface definitions which are easy to implement on top of widely available cross-platform open-source APIs.

The toolkit level provides abstractions for all platform (hardware/OS) or native API dependent aspects of game programming such as graphics, sound, input, timers, networking, and physics. Supporting RAGE games is reduced to providing implementations for these toolkit level components. Since the components, have been designed largely as adapter pattern interfaces to widely implemented stable open source libraries, this implementation task is relatively straightforward. Given an implementation of the toolkit components, any RAGE game will run unmodified on any target platform.

The modular architecture also allows substitution of implementations for any particular component. This allows RAGE to utilize DirectX functionality on Windows platforms while utilizing OpenGL on Mac OSX. Once again RAGE games run unaffected by different module implementations. Another aspect is that experimental implementations of
different modules can be trialed, while still utilizing the rest of the framework. This capability is demonstrated by the different physics implementations.

The most interesting aspect of RAGE is the extent to which it has become a ‘glue’ framework which ties many different open-source API’s or commercial APIs together. RAGE development has become an exercise in identifying and separating game engine functionality into highly independent, separately implemented components. Another interesting aspect is how we have found that non-game functionality such as font rendering, and menuing systems are crucial to the ease of use of the package. RAGE is also highly catholic in regard to its source model formats.

The lack of coupling between toolkit level components contrasts with other open-source projects. For example the Light Weight Java Game Library (LWJGL) is closely coupled with OpenGL. Other Java game related projects such as JOGL, Jinput, and JOAL also have dependencies on other packages. The emphasis on modularity is what distinguishes RAGE from almost all other open-source and commercial game engines.

The Java game engine project most similar to the RAGE engine is the jMonkey Engine or jME. Compared with jME the RAGE engine is relatively raw and unpolished. However, RAGE has taken a higher level approach to game engine functionality with framework level support for task scheduling, menu systems, and AI through scripting. jME is also tightly coupled with the LWJGL while RAGE maintains implementation independence.

RAGE has borrowed widely from many other game engines and game APIs for its architectural and implementation concepts. Auran Jet taught us much about basic object oriented game engine structure, application of common design patterns, game scheduling, and plugin architectures. The DirectX APIs have evolved over many years of intense use, and we have frankly used these as a reference standard when evaluating what functionality to include in an interface. The Unreal engines have always provided an ideal in terms of what scripting and network functionality we should provide.

We have accessed and analysed any and all open-source game engine projects we could find and download. This has allowed us to gain an understanding of what approaches are common, and also identify ‘best practices’ which we then attempt to incorporate in our own design. It has also allowed us to identify those areas where further research and development can take place.

As development has proceeded we have found the emphasis in development has shifted. Initially we were concerned mainly with details and features of rendering, input, and sound. Then scene management became our greatest concern. Now content management is gaining prominence.

4. RESEARCH DIRECTIONS
Processor speed is no longer increasing as rapidly as before, and multiple processor systems are becoming common. New game platforms due for release in the next two years utilize as many as eight processors. Most current game technologies utilize only a single-processor, and it is the goal of this project to develop and test a technology that makes use of these new multiprocessor systems. The next phase of this project will involve the design and implementation of a multi-threaded game engine prototype.

Another project is aimed at solving issues involved in current scene management technologies. Most current scene management systems are tightly coupled with the rest of the engine and provide a limited set of functions. We are researching a fast, modular, flexible framework that allows for the easy integration of multiple specialized traversal and culling algorithms and can be used in non-graphical environments.

5. CONCLUSIONS
RAGE is a learning vehicle for final year Computer Science (Games Technology) students at Charles Sturt University. During its development students have gained valuable insight into the structure and functions of game engines at both a conceptual, and implementation level. We think the work we’ve done in defining our component interfaces will be a useful contribution in evolving an open set of game engine standards, and in providing a useful, flexible, open-source game engine in its own right. As the engine continues to develop the range of supported functions and quality of implementation should continue to improve.

6. ACKNOWLEDGMENTS
Our thanks to the Apple University Consortium Australia for providing us with Macintosh equipment through a seed grant and a developers grant.

7. REFERENCES

Figure 3. A 3D scene running the same game code using different platform implementations. Left: PC Windows using DirectX. Right: Apple Macintosh OSX using OpenGL.