SCIENTIFIC PROGRAMME

GAMES DESIGN AND DEVELOPMENT

A REVIEW OF 3-D ACCELERATOR TECHNOLOGY FOR GAMES

NATHAN CHIA, RICHARD CANT, DAVID AL-DABASS

Department of Computing and Mathematics
The Nottingham Trent University
Nottingham NG1 4BU.
Email: richard.cant/david.al-dabass@ntu.ac.uk

KEYWORDS
OpenGL, DirectX, anti-aliasing, animation graphics.

ABSTRACT

In this paper we attempt to review the current technology of 3 -D accelerators for animating graphics used in games and visual simulation systems, together with associated techniques and software architectures. Topics covered include OpenGl, DirectX, anti-aliasing, motion blur and depth of field. A summary of work in progress is give in the conclusions section.

INTRODUCTION

Ever since 3dfx released their first Voodoo card (the first 3D hardware accelerator for the consumer market), it had always been a 5 horse race between the top video card makers: nVidia, 3dfx, ATI, Matrox and S3. It was a race to see who can hit the highest screen resolution and who can draw the most polygons in a period of time. It was rarely about making the real-time renders closer to real-life.

3dfx came out with a simplified version of the accumulative buffer to gain more speed from the available version. It was named the 'T-buffer' and it was introduced as the solution for drawing effects like motion blur, depth-of-field and multi-sampling anti-aliasing in real-time. Now that 3dfx has faded away and nVidia is bringing out with their own version of multi-sampling anti-aliasing named 'Quincunx AA', it seems everyone had forgotten about the things that the T-buffer had put out to do.

In this paper we will first review the development of 3-D graphics technology and the current state of the art. We will then discuss some effects that remain to be incorporated in 3-D rendering systems. The problems facing 3D accelerators today are mainly spatial aliasing, depthless rendering, temporal aliased animation and images looking 'too perfect'. One might question the motive for this by asking isn't it better to have sharper, perfect renders rather than distorting them? However, including these effects will give a greater feel of realism and improve game play by modifying the difficulty of task.

Hopefully these effects can be included in ways that exploit the features of current hardware such as texture mapping support. This will give software and game makers more freedom to create 3D renders with their very own style in them. The consumers will benefit from this greatly as more real-time graphics is pushed towards reality.

THE HISTORY OF COMPUTER GRAPHICS

Computer Graphics first started in 1945 when one of the earliest electronic computers, the ENIAC (Electronic Numerical Intergrator And Computer), was built at the University of Pennsylvania's Moore School of Engineers. By the 1950s, they were powerful enough to deal with computer graphics. To handle the task of drawing lines, special vector display devices were designed to be interfaced to the computer.

Vector display can produce wire-frame images (Figure 1), which needed only a minimal amount of storage. As computer memory is extremely expensive in those days, it was only capable of drawing a list of segments. As the 60s grew nearer, General Motors and IBM jointly designed the DAC-1 (Design Augmented by Computer). It allowed the user to enter geometric specification of a wire-frame object and view it from different angles (Figure 2).

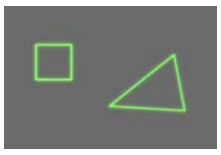


Figure 1 Vector Display

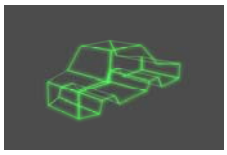


Figure 2 Wire-frame Object

In the 1960s, Digital Equipment Corporation (DEC) produced the PDP-1 computer. MIT bought one of the machines and a group of its students created the first video game – Spacewars. The game was so popular

that it took up more than half the usage time on the PDP-1 and it was used as a benchmark for newer computers. Despite it's popularity, it was never copyrighted because very few people could afford the hardware that runs it, and the concept had been copied numerous times.

In the 1960s, another MIT student, Ivan Sutherland, built the first head mounted display. The device was capable of displaying two separate wire-frame images, one for each eye. The two images were generated from slightly different angles of view that corresponds to the position of the eye. This technique created the stereoscopic effect of depth (Figure 3). In 1967, General Electric (founded by Sutherland and Dave Evans from the University of Utah) developed the first real-time flight simulator for NASA.

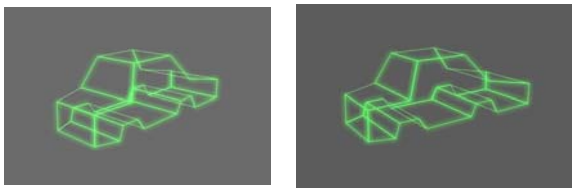


Figure 3 Ivan Sutherland's head mounted display image

There had been a great increase in speed and quantities and decrease in price for computers in the 70s, which rendered the raster display devices practical. Raster display devices represents images as a regular mosaic of dots, known as pixels, each with different colour and shade. Solid object generation is also possible. (In 1977, a 512-by-512 display acquired by the National Institute of Health, in USA, cost US$65000. A similar one today can be bought for less than $50.) In 1971, Henri Gouraud proposed an algorithm to interpolate a polygon's colour to produce a continuous matte object as opposed to flat surfaced ones (Figure 4). Later, in the University of Utah, Phong Bui-Toung proposed a method similar to Gouraud's. It aimed to model shiny surfaces approximated by polygons (Figure 5).

In 1974, Ed Catmull (who went on to lead the graphics division of Lucasfilm) proposed texturing mapping, Z-buffering hidden surface removal, and modelling with curved surfaces. These techniques are now commonly available in all 3D accelerator cards.

Zbuffer is the area of the graphics memory used to store the Z or depth information about rendered objects. The Z-buffer value of a pixel is used to determine if it is behind or in front of another pixel. Z calculations prevent background objects from overwriting foreground objects in the frame buffer.

Two years later, Jim Blinn developed environmental reflection mapping and bump mapping which adds details to a 3D model without adding more polygons (Figure 6). A modern processor, which does the real-time environmental bump mapping very well, is the G400 series from Matrox. Real-time reflection mapping is claimed to be available in the hardware of nVidia's NV15,- it is just the hardware's ability to take snapshots quickly from the scene so that they can be used as reflective textures.

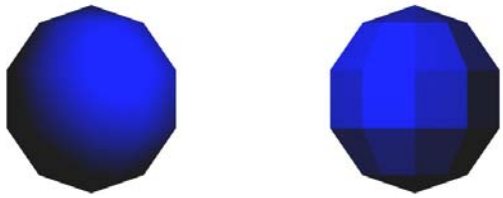


Figure 4 Sphere drawn with and without Gouraud Shading

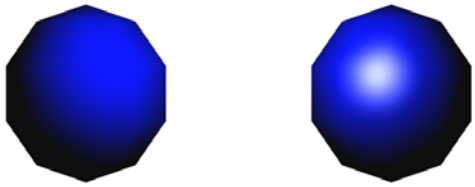


Figure 5 Gouraud Shading and Phong Shading

Bump mapping is a shading technique using multiple textures and lighting effects to simulate wrinkled or bumped surfaces. Bump mapping is useful because it gives a 3D surface the appearance of roughness and other surface detail, such as dimples on a golf ball, without increasing the geometric complexity. Some common types of bump mapping are Emboss Bump Mapping, Dot3 Bump Mapping, Environment Mapped Bump Mapping (EMBM) and True, Reflective Bump Mapping. Dot3 bump mapping is the most effective technique of the three.

In 1982, Silicon Graphics (founded by Jim Clark) produced computers with built-in capabilities for graphics. Since then, computer graphics started to appear in movies. The first was in Disney's Tron, which used approximately 30 minutes of computer graphics. Lucasfilm and its famous Industrial Light and Magic division (later, part of it was branched off as Pixar) progressed from the projection of the Death Star in Return of the Jedi to the sophisticated animation of Jurrasic Park.

3D graphics also started to take off in the gaming industry on the IBM-PC in 1991. In 1991, John Carmack, Adrian Carmack and John Romero founded id Software and released Wolfenstein 3D (Figure 7).

Although the game used a much simplified texture mapping and projection techniques, and "bill-boarding" was used heavily, it considerably raised the expectations for 3D graphics in computer games. This

placed id Software on the map and its now developing the sequel to the classic.

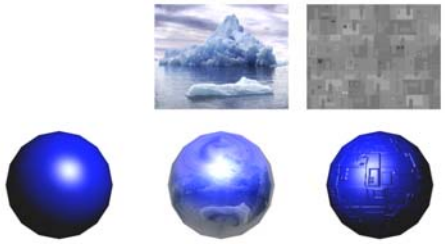


Figure 6 Phong Shading, Environmental Reflection Mapping and Bump Mapping

Bill-boarding is to have the quadrilateral polygon drawn base its orientation on the view direction. As the view changes, the orientation of the polygon changes. Bill-boarding, combined with alpha texturing and animation, can be used to represent many phenomena that do not have solid surfaces. Smoke, fire, explosions, vapor trails, and clouds are just a few of the objects that can be represented by these techniques. But for id's case, bill-boarding is used to pre-draw or pre-render the characters in its game into a texture and is more effective for the computer at that time to draw just one polygon for each character rather than rendering them in real-time.

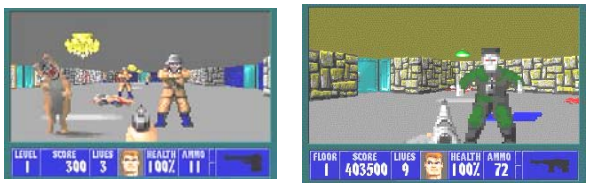


Figure 7 id Software's Wolfenstein 3D

Most recently, graphic cards with 3D algorithms built into the hardware became affordable for the personal computers. Quality real-time 3D graphics that used to be only possible on specialized workstations is now possible on the normal PC (Figure 8).

Figure 8 id Software's sequel to Wolfenstein 3D

The remaining top players are nVidia and ATi. 3dfx was bought over by nVidia and Matrox is now focussing on the 'office' market. Recently, however, 3DPower is formed by the ex-employees of 3dfx and headed by former CTO Scott Sellers. The 3dfx ex-employees secretly gathered without the knowledge of 3dfx or nVidia and their new product will be based specifically on 3dfx's previous Rampage graphics chipset.

With this huge variety of graphics card to program for, developers will soon find that the situation is becoming more and more chaotic. Every PC user owns a different graphics card and new hardware keeps popping day after day. Here's where API comes into the picture. The APIs available now are OpenGL and DirectX.

CURRENT APIS, BENEFITS AND LIMITATIONS

OpenGL

The definition of OpenGL was a software interface to graphics hardware. Better describing it would be a 3D graphics and modelling library that is extremely portable and fast. The greatest strength of OpenGL compared to a ray tracer is speed.

Ray tracing is also known as the screen-to-world method. The idea of ray tracing is quite different to OpenGL world-to-screen method, which is projection of a 3D scene to a two-dimensional buffer. In ray tracing, for every pixel on the screen, a ray is cast into the representation of the virtual world until it intersects with some surface. The colour of that surface in the intersection point is what is supposed to be seen at that pixel on the screen.

It is a new industry standard that had gained enormous support over a few years. It originated from IRIS GL from Silicon Graphics. It is the company's 3D programming API for the IRIS graphics workstations. These machines had specialized hardware optimised for the displaying of sophisticated real-time graphics. The IRIS hardware does hardware matrix transformations, hardware depth buffering, and other features.

OpenGL was born when Silicon Graphics got stuck trying to port IRIS GL to other hardware platforms. OpenGL is in fact an improvement in IRIS GL's portability. It has the same capabilities of IRIS GL but is "OPEN" for adaptability to any hardware platforms and operating systems. OpenGL is more like a C runtime library rather then an application itself and it is intended to be used with specialized hardware that is designed to draw and manipulate 3D graphics.

Because of its efficiency, OpenGL is used in a large number of areas, like CAD, architectural, modelling and animations. And with hardware accelerators and faster processors becoming essentials, 3D graphics will soon be a typical building block for any consumer and business applications. A good example of such is

Quattro Pro, one of the pioneers to utilise 3D to power its 3D charting of the old 2D spreadsheets. OpenGL would add a lot of pleasant appearance to products because appearance does matter.

To ensure that a particular hardware is OpenGL compatible, it must undergo OpenGL conformance tests. These are a set of tests designed to ensure full implementation and production of reasonably acceptable 3D renders.

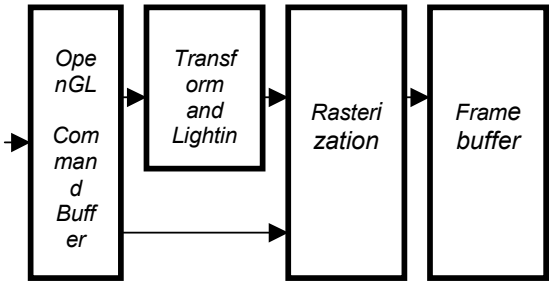


Figure 9 OpenGL API Calls

When an OpenGL API call is made, the commands are placed in a command buffer. Vertex and texture data will be in the same buffer. When the buffer is flushed, the contents in it would be passed to the next stage in the pipeline.

The Transform and Lighting stage with vertices will be recalculated for position and orientation, and lighting calculations are performed to render the shading and colour at each vertex. After the transform and lighting stage, the data is fed to the rasterization segment of the pipeline. The rasterizer actually creates a two-dimensional image from the geometry and texture. This image would finally end up in the frame buffer, which is in the graphic card's display memory (image is visible on the screen at this point).

The initial 3D accelerators were nothing more than fast rasterizers. Only the rasterization portion of the pipeline is accelerated. The CPU had to do the rest. This would mean that the performance of the 3D graphics with such accelerators would be greatly pulled back by the processor.

As computer peripherals get more and more affordable, the T&L stage is implemented in the accelerator, even for low-end consumer hardware. This would mean that higher detailed models and more complex graphics are possible at real-time rendering.

DirectX

DirectX is a multimedia development library, created by Microsoft and provided for royalty-free use in the creation of applications.

The DirectX SDK contains classes that are the foundation of DirectX. They are:

DirectAnimation

DirectDraw -Provides efficient access to the video memory, resulting in smooth animation for game titles.

Direct3D -Provides high-performance rendering of 3D scenes, utilizing the latest 3D accelerators.

DirectSound -Provides audio playback and mixing, including 3D sound effects

DirectMusic -Provides interactive music capabilities, allowing for soundtracks that change with action.

DirectInput -Allows input from keyboard, mouse, and force feedback input devices.

DirectPlay -Provides communications for network applications over the internet and local area network, or through direct connection with modem or serial cable (null-modem cable).

Microsoft intentions were to provide a standard software interface, making interfacing with hardware transparent to the developer. The biggest advantage of developing a multimedia application is the fact that Microsoft Windows is the predominant operating system on home computers and DirectX comes free with the OS.

Direct3D was first developed by a British company called RenderMorphics. It is later acquired by Microsoft and D3D was now a component in the DirectX library. Direct3D covers a set of functionality comparable to OpenGL and is also aimed at providing a unified access to different hardware. Direct3D can performed in 2 modes. HAL(Hardware Abstraction Layer) and HEL(Hardware Emulation Layer).

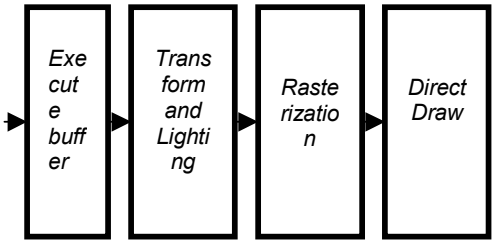


Figure 10 Direct3D API call

The 'Execute Buffer' of Direct3D is quite similar to the buffer of OpenGL's 'Command Buffer'. This buffer would contain the geometric information as

well as commands to be performed. Unlike OpenGL, the execute buffers are quite cumbersome to work with. Multiple data structures must be allocated, locked, filled up in order to construct them.

The final module, 'DirectDraw', actually handles the access of the frame-buffer. Another disadvantage of Direct3D is that it only supports triangles. OpenGL supports polygonal objects as long as all the points are convex and no intersection is occurred.

CURRENT TECHNIQUES

Anti-aliasing

One of the biggest faults of raster graphics to all PC users which needs no introduction is aliasing. Because we are so used to perceive images in an "infinite" resolution, it would be quite impossible to try to achieve that with higher and higher screen resolutions. No matter how high the resolution goes, the eyes will still tell the mind that the staircase edges are still evident when observed closely. It is the same reason why a 320x240 MPEG1 video might looks more pleasing than a 1024x768 real-time 3D render. That is why anti-aliasing plays such an important part in producing a realistic image.

Also due to the impracticality of building an infinite resolution display, another shortcoming of raster graphics is under-sampling. This effect gets more visible when textured objects are drawn smaller or further away. For example, visualise a checkers board in the middle of the screen. It moves further away in the 3D scene. One would expect that as it gets further, the white and black would blur into a shade of grey. It doesn't (Left image of figure 11). In fact, the viewer would witness a shimmering effect at some point. The pixels alternate between black and white.

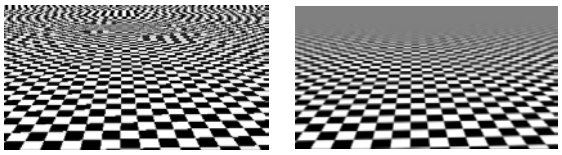


Figure 11 Checkered Texture without filtering (Left) and with filtering (Right)

This aliasing found in Figure 11 can be reduced by the use of bilinear filtering and MIP maps. The essential idea of MIP mapping is to pre-compute the texture at different levels of detail (Figure 12), and then to use smaller textures for polygons further away from the viewer. Bilinear filtering is used for textures that are magnified to smooth out the jaggies. This could be improved further by bilinearly blending neighbouring levels of MIP maps to generate more levels of detail. This is known as trilinear filtering. Although this will not truly eliminate aliasing, the shimmering effect mentioned earlier would be eliminated.

Figure 12 Generating smaller images

Another method of removing aliasing is to make use of the accumulative buffer of the hardware accelerator. This can be quite time consuming and hence cannot be used in real-time animation. It is, however, relatively easy to implement. The technique is to jitter the image one-half a pixel in several directions, to blur the edges of an image but not the solid areas. Only four jitters is necessary to produce a remarkably smooth image but even four jitters will require the whole scene to be rendered 4 times.

Another anti-aliasing technique which most hardware accelerator implements is super-sampling. Super-sampling anti-aliasing draws the scene in a much larger buffer than the screen resolution and scales it down to fit the screen. The filtering procedure will take a group of pixels from the original image and compute the weighted sum of their intensities. The result from this sum will be placed into a filtered image bitmap. This will then be placed onto the frame buffer to show the antialiased image.

This however requires a significant amount of video memory and bandwidth. (Bandwidth is still the main limitations of most hardware accelerators. Most accelerators could be perceived as large buckets with tiny bottlenecks.)

Another technique, which the latest Geforce 3 card from nVidia is using for anti-aliasing, is multisampling. nVidia patented it and calls it HRAA (High Resolution Anti-aliasing) or Quincunx anti-aliasing. nVidia claims that it provides performance that rivals a 4X super-sampling anti-alising render with the performance hit slightly more than a 2X super-sampling anti-aliasing render.

This technique uses a "reconstruction filter" that uses data from the neighbouring pixels to compute the final pixel colour.

nVidia will no longer be "blowing up" the 800x600 screen to 1600x1200 and then shrinking it back down in hardware to do their anti-aliasing in their new Quincunx mode. Quin·cunx (kwin'kungks) - It is an arrangement of five objects, with one at each corner of a rectangle or square and one at the center (like the side '5' of a dice). This is a reference to the way they

sample the pixels that are used to fill in the jaggies. It pulls information from the pixels surrounding a pixel in order to smooth out the lines that are not perfectly vertical or horizontal. This means that a significant amount of bandwidth and memory usage can be greatly reduced.

Motion Blur

The deficiency of motion blur is also a form of aliasing known as temporal aliasing (the previous form aliasing could be called spatial aliasing). The human eyes are fast enough only for 25-30 frames per second, and less in darker scenes. But the human eye is capable for blurring several consecutive frames that are flipped too quickly. That is why real-time 3D animations that moves at 24 frames per second looks less convincing than in the movies which moves at the same rate.

Temporal aliasing can also be seen in some videos of moving carts in cowboy movies where the spokes of turning wheels suddenly appear to be going the opposite direction when it is not. It happens when the sampling rate of the camera gets lower than rate of movement of the spoke moving to the position of the next one.

In 3D graphics, without motion-blurring, the result would be even worse than in the movies. Motion blurring would add a blur trail to hint to us the direction where the spoke is moving towards and gives an illusion of a more fluid animation.

Depth Of Field

The human eye is able to focus on a particular object and the rest of the "extras" in our scene will be defocused. We can often see how this effect is used to allow even a still image to be more dramatic in the movies. How photographers with their award masterpiece capture the essence of an object by using depth of field.

Figure 13 Elsa 3D Revelator

Figure 14 Left and Right Images on Alternate Lines

The effect of 'Depth of Field' can be achieved by the use of some affordable 3D glasses (Figure 13). The glasses has to be in sync with the refresh rate of the monitor. By altering the display drivers so that what the left eye sees will go on the odd lines of the screen and what the right eye sees to the even lines (Figure 14). The left side of the glasses will turn opaque when the even lines are drawn and will turn transparent when the odd lines are drawn. The right side does the exact opposite.

Disadvantages are that the vertical resolution will be reduced by half and it will often cause discomfort to the person viewing when the view of that person is not positioned right in the centre from the centre of the monitor.

The illusion of depth of field can also be achieved by using (once again) the accumulative buffer. The idea is to do multiple renders with the source of the view-port offset slightly around the original position (Figure 15). The target of the view-port will remain at the position of the focus distance.

Once again, the use of the accumulative buffer might be too time-consuming for most hardware accelerators.

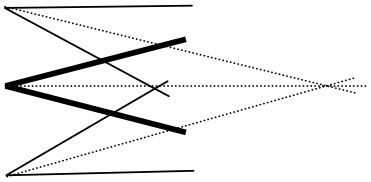


Figure 15 DOF effect with Accumulative Buffer

CONCLUSIONS AND WORK IN PROGRESS

A review of current state of the technology in computer graphics and 3-D accelerators for games was given. A brief history of graphics was outlined which focused on OpenGL and DirectX. A review of current techniques was described which included anti-aliasing, motion blur and depth of field.

Current work addresses the following problems:

Lack of a customizable graphics engine: to demonstrate and implement the algorithms, reasonably believable 3D scenes and models are required to be constructed. There is also a need for better texture management for the graphics engine to be more flexible and easier to use. Textures are also used within the model importer, so, the texture library must be intelligent enough to prevent duplicate loadings of the same texture and yet keep this invisible from the programmer. A model importer that supports multiple textures and can support deformation for character animation is needed too.

Spatial aliasing: due to the nature of raster displays, aliasing artifacts are introduced into a real-time scene. The brute force method of super-sampling antialiasing adopted by current graphics card seems to do the job quite well but has a big performance hit. The hardware filters are also not customizable by the programmer.

Depthless renders: current hardware has the ability of fogging which reduces the depth confusion in an image. But the existence of fogging in a small area, like a room, will make a scene look artificial. There is a need for another mechanism, like depth-of-field, that would reduce that confusion without the use of fog. It should imitate the depth-of-field witnessed in real-life photography.

Temporal aliasing: Because of this effect in real-time computer graphics, graphics cards need to render at very high frame-rates to create the illusion of fluid motion. Multi-pass techniques consumes much processing time and hence power.

REFERENCES

1. Richard S. Wright, Jr and Micheal Sweet, "OpenGL Super Bible", 2nd Edition, By Waite Group Press.
2. Robert Dunlop with Dale Shepherd and Mark Martin, "DirectX7 (Teach Yourself … in 24 hours)",SAMS.
3. Sergei Savchenko, "3D Graphics Programming", SAMS.
4. Rod Stephens, "Visual Basic Graphics Programming", Wiley
5. NVIDIA's Developer Relations Site, (SDKs, technical papers, demos) http://www.nvidia.com/developer.
6. Neon Helium Productions, (Tutorials, base code, AVI loader), http://nehe.gamedev.net.
7. Code Guru, http://www.codeguru.com.
8. Emeran, R., Mongomery, S. & Werfall, J., (2000) Pixel perfect - graphics card review, *Personal Computer World*, Oct. 2000, 195-213.
9. Montgomery, S. (2001) Graphics Cards, *PCW*, Sept. 187-199.
10. Emeran, R. et al (2001) Graphics galore (graphics card group test), PCW, Sept 161-181.
11. Wen et al (2000) Creating Animated Behavioural Game Characters based on Environmental Effects, Proc GAME-ON 2000 Int Conf, pp76-80.

COMPONENT BASED MOTION EDITING ENVIRONMENT FOR 3D GAME CHARACTER DESIGN

Yoshihiro Okada
Graduate School of Information Science and Electrical Engineering,
Kyushu University
6-1 Kasuga-Koen, Kasuga
Fukuoka 816-8580, Japan
E-mail: okada@i.kyushu-u.ac.jp

KEYWORDS
Motion Design, Human Interface, Software Architecture, Game Toolkit, Component ware.

ABSTRACT
For 3D game creation, character design is a very important factor but very hard work. Especially its motion design is very laborious work. We have to spend much time to design character's motions. So this paper proposes new motion editing environment for 3D game character design. The proposed motion editing environment is based on a key-frame animation technology. In conventional tools for key-frame animation, each key-pose is defined and displayed separately on a computer screen so that users can not recognize its complete motion until they see its animation by applying a defined motion to a character. The proposed motion editing environment displays all sequential key-poses at the same time on a computer screen. Then by seeing those key-poses users can recognize a complete motion those key-poses mean and can edit each key-pose interactively and easily by comparison with its adjoining poses. Furthermore, the proposed motion editing environment is realized as composition of several software components so that users can make its copy and transfer the copy to other computers through Internet. As a result, users can create motions only by reusing and modifying the motions already defined by other users.

1. INTRODUCTION

Advances in recent computer hardware technology have made possible 3D rendering images in real-time. However, it is still difficult for end-users to develop 3D graphics software. For this reason, Okada and Tanaka developed a 3D prototype system called *IntelligentBox*(Okada and Tanaka 1995). *IntelligentBox* is a component based construction system. Its application fields include various kinds, e.g., animation creation (Okada and Tanaka, 1999), collaborative virtual environment construction (Okada and Tanaka, 1998), education system development (Okada and Itoh, 2001) and so on. *IntelligentBox* also has aspects as an interactive 3D game development system (Okada et. al., 2000). For 3D game creation as well as 3D animation creation, 3D shape design and motion design are the most laborious works. Traditional motion design is based on key-frame animation. A motion is represented as a sequence of a number of poses those are automatically generated by interpolation of several key-poses, which are poses used as keys of key-frame animation. Each key-pose is defined by specifying the joints angles of a character. In conventional tools for key-frame animation, each key-pose is defined and displayed separately on a computer screen so that users can not recognize its complete motion until they see its animation by applying a defined motion to a character. So this paper proposes new motion editing environment for 3D game character design. The proposed motion editing environment is also based on a key-frame animation technology. However, the proposed motion editing environment displays all sequential key-poses at the same time on a computer screen. Then by seeing those key-poses users can recognize a complete motion those key-poses mean and can edit each key-pose interactively and easily by comparison with its adjoining poses.
We have already developed such motion editing environment using *IntelligentBox*. As mentioned above, *IntelligentBox* is a component based development tool that provides functional components called *boxes*. The proposed motion editing environment is developed as a composite *box*, which includes user-defined motion information itself. Therefore, users can exchange their edited motions each other through *Internet* by copy-and-transfer operations, and create motions only by modifying the motions already defined by other users.

[Related Work]
There are many researches on motion generation for computer animation. (Witkin and Kass, 1988) proposed concept of spacetime constraints. After that, many research papers based on spacetime constraints were published. IK(Inverse Kinematics) is one of the popular methods for efficient motion generation. The motion path functionality is also a popular technique to intuitively define movement of a character's center of mass. Furthermore, the use of motion capture data has been becoming common. Many animation creation software products have been made so far. Most of them provide a traditional key-frame animation function, IK and a motion path function. Those products include 3D Studio Max™, LightWave™ and so on. IK is a powerful tool but it is not always available for arbitrary motion definition. Then key-frame animation is

still used in many cases. Life Forms™ provides a graphical timeline display functionality of key-poses. Users can understand a motion by seeing the graphical timeline of its key-poses. However, it does not include movement information of a character's center of mass. In our motion editing environment, users can edit each key-pose of a character, which includes the movement of its center of mass besides its joints angles. Most products also support the use of motion capture data. However, most of the motion capture data are human motions. For animation creation of animals and other characters except human-like figures, users have to define their motions based on key-frame animation. Therefore, our motion editing environment is significant since there are few researches on efficient and intuitive interface for motion design based on key-frame animation. Gleicher proposed a motion editing based on spacetime constraints (Gleicher, 1997). Lee proposed an interactive motion editing method for human-like figures (Lee and Shin, 1999). Both of their systems provide an interactive and graphical interface for motion editing. Those are very intuitive interfaces. However, their systems are dedicated to generate motions. Our system is an integrated system so that the user can create 3D game characters in the one integrated environment.

The remainder of this paper is organized as follows. Section 2 explains essential mechanisms of *IntelligentBox* and shows its simple composite *box* example. Section 3 explains the motion editing environment and its realization mechanisms. Section 4 shows motion definition examples. Finally section 5 concludes this paper.

2. ESSENTIAL MECHANISMS OF *INTELLIGENTBOX*

IntelligentBox employs the following essential mechanisms inherited from *IntelligentPad* (Tanaka 1996), which is a 2D synthetic media system since *IntelligentBox* is an extension of *IntelligentPad* to 3D graphics applications.

2.1 *Model-View-Controller* (*MVC*) Structure

As shown in Figure 1, each *box* consists of two objects, a *model* and a *display object*. This structure is called an *MD* (*Model-Display object*) structure. Strictly speaking, a *display object* consists of two objects, a *view* and a *controller*. Therefore, this structure is called an *MVC* structure. A *model* holds state values of a *box*. They are stored in variables called *slots*. A *view* defines how the *box* appears on a computer screen. A *controller* defines how the *box* reacts to user operations.

Figure 1 also shows messages between a *display object* and a *model*. This is an example of a *RotationBox*. A *RotationBox* has a *slot* named 'ratio' that holds a double precision number, which means a rotation angle. This value is normalized between zero and one. One means one rotation. Through direct manipulations on a *box*, its associated *slot* value changes. Furthermore, its visual image simultaneously changes according to the *slot* value change. So a *box* reacts to a user's manipulations according to its function.

2.2 Message-Sending Protocol for Slot Connections

Figure 2 illustrates a data linkage concept among *boxes*. As shown in the figure, each *box* has multiple *slots*. Each *slot* can be connected to one of the *slots* of an other *box*. This connection is called a *slot connection*. The *slot connection* is carried out by some messages when there is a parent-child relationship between two *boxes*. There are three standard messages, i.e., a *set* message, a *gimme* message and an *update* message. These messages have the following formats:

(1) Parent box *set* <slotname> <value>.
(2) Parent box *gimme* <slotname>.
(3) Child box *update*.

A <value> in a format (1) represents any value, and a <slotname> in formats (1) and (2) represents a user-selected *slot* of the parent *box* that receives these two messages.

A *set* message writes a child *box slot* value into its parent *box slot*. A *gimme* message reads a parent *box slot* value and sets it into its child *box slot*. *Update* messages are issued from a parent *box* to all of its child *boxes* to tell them that the parent *box slot* value has changed. In this way, these three messages connect a child *box slot* and its parent *box slot*, and combine their two functions.

2.3 A Simple Example of Composite *Boxes*

Figure 3 shows a simple composite *box* example. Figure 4 illustrates its message flow and data flow. In this example, the motor is a counter *box* having a cylindrical shape. Its model has a *slot* with a double precision number value that is increased automatically by a timer process. The toggle button attached to the motor works as a switch that changes the state of the timer process. Pushing the toggle button activates the timer process and hence the motor. The *slot* value of the motor begins to increase automatically, and

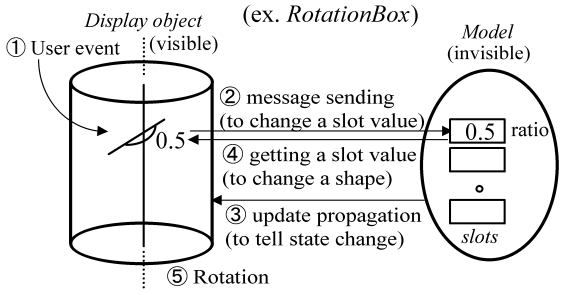


Figure 1. An *MD* structure of a *box* and its internal messages.

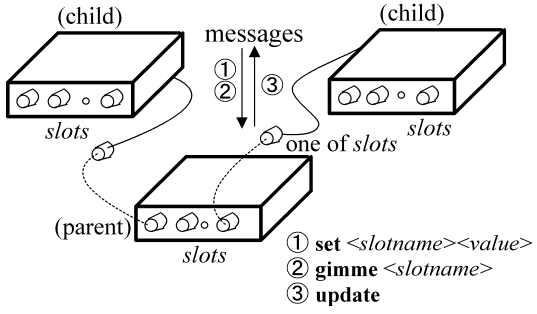


Figure 2. Standard messages between *boxes*.

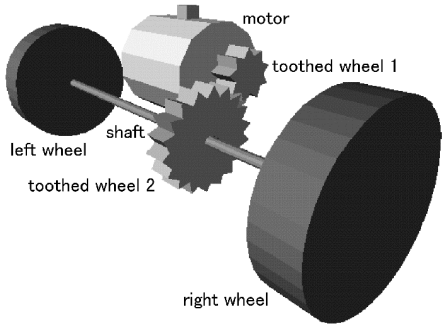


Figure 3. A composite *box* example.

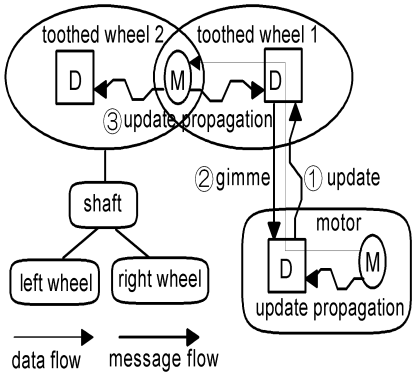


Figure 4. Message and data flow between *boxes*.

hence the motor begins to rotate. Then according to the data flow shown in the figure, the toothed wheel 1 and toothed wheel 2 come to rotate since both are *RotationBoxes*. Consequently, the toothed wheel 2 makes the shaft with the two wheels rotate.

3. COMPONENT BASED MOTION EDITING ENVIRONMENT

3.1 Component structure of a human-like model and its one pose data

Figure 5 shows components of a human-like model. This model is consisted of 17 joints. Each joint is a *3DRotationBox*. The bottom *box* is an *ArrayBox* that stores xyz-angle data of the all joints. Therefore, this *ArrayBox* keeps one pose data. So this composite *box* illustrated in the figure is used as a unit for editing one pose.

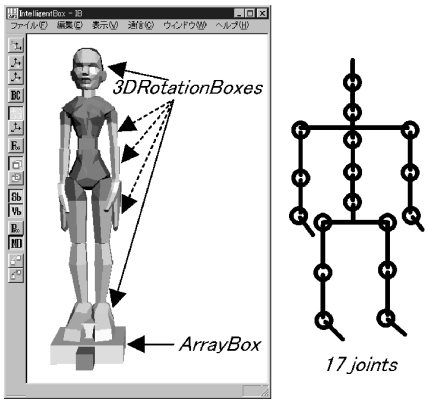


Figure 5. Component structure of a human-like model.

3.2 Editing of multiple key-poses

Figure 6 shows multiple key-poses those mean a walking motion. This motion is consisted of five key-poses. As you see the figure and easily understand that the motion is a walking motion, the figure means an intuitive motion editing environment we propose in this paper. Figure 7(left) shows another motion. This is a jump motion consisting of six different poses. As for a walking motion, the character's center of mass moves gradually in one direction. So it is not difficult to specify each pose by directly dragging the joints on a computer screen. However as for a jump motion, the character's center of mass moves up and then down again. So it is difficult to specify each pose by directly dragging the joints on a computer screen since some poses are occluded. In this case, it is possible to disappear other poses except the one pose that the user is currently modifying. As you see, there is a *ContainerBox* below each pose. A *ContainerBox* controls visibility of its descendant *boxes*. If a user clicks a mouse button on a *ContainerBox*, its descendant *boxes* become invisible, and the user clicks again then its descendant *boxes* become visible. By interactively controlling their visibility, users can edit a pose with showing only one corresponding character as shown in Figure 7(right).

3.3 Mechanism of motion generation

Figure 8 shows data flow and component structure for concatenated motion generation. The upper part of the figure is component structure for one motion generation. There is an *InterpolationBox*. The *InterpolationBox*

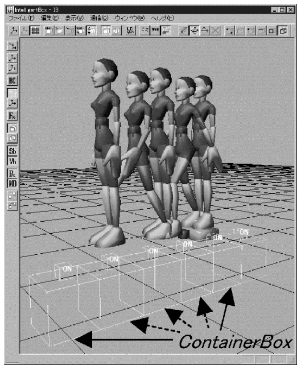


Figure 6. Editing of a walk motion.

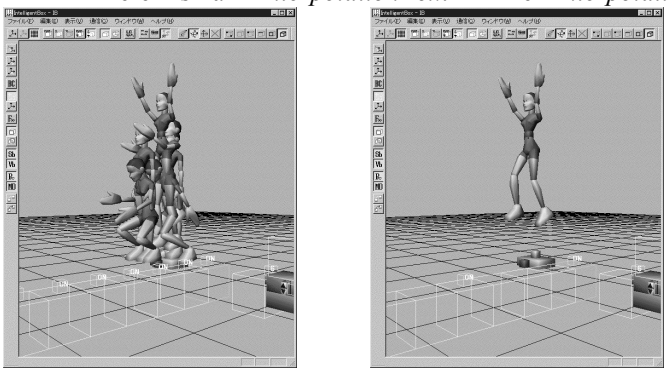


Figure 7. Editing of a jump motion(left) and its one pose(right).

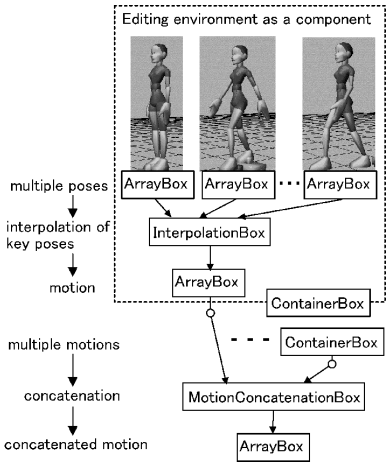


Figure 8. Data flow of concatenated motion generation.

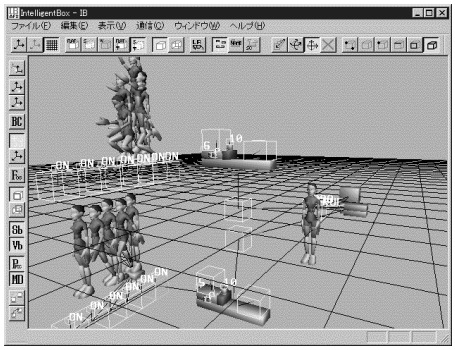
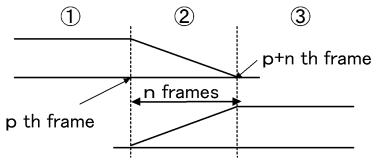


Figure 9. Example model of concatenated motion generation from two different motions.



① if (i <= p) then i th frame of current motion.
② if (p < i <= p + n) then
(p + n − i)/n * i th frame of current motion
+ (i − p)/n * (i − p) th frame of next motion.
③ if (p + n < i) then (i − p) th frame of next motion.

Figure 10. Smooth concatenation of two sequential motions.

generates a motion as a complete sequence of poses generated by interpolation among several key-poses. The motion is stored in a *slot* of an *ArrayBox*. Furthermore the lower part of the figure is component structure for concatenation of several motions. There is a *MotionConcatenationBox*. The *MotionConcatenationBox* generates one motion as a sequence of several motions. Figure 9 shows an example model to generate a concatenated motion from two different motions. One motion is a walking motion and the other is a jump motion. In this figure, the walk motion is assigned a value of zero as its ID number and the jump motion is assigned a value of one. After the user specifies a sequence of ID number values like (0, 1, 1, 0), the *MotionConcatenationBox* generates one concatenated motion. That motion acts in the order of a walk, a jump, a jump and a walk. As shown

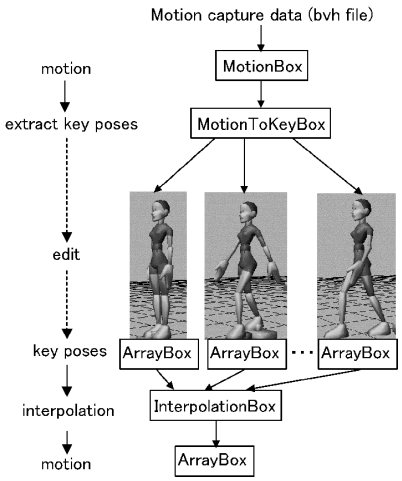


Figure 11. Data flow to generate one new motion from motion capture data.

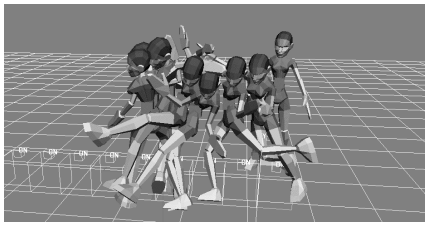


Figure 12. Key-poses extracted from a kick motion file.

in Figure 10, two sequential motions are concatenated smoothly by a linear combination of last n frames of the first motion and first n frames of the second motion. Strictly speaking, concatenation process generates a smooth motion, i.e., the first motion fades out and the second motion simultaneously fades in.

3.4 Motion capture data support

As shown in Figure 11, *IntelligentBox* has provided a particular *box* called a *MotionBox* This *box* reads a motion capture data file and generates a motion as a sequence of several poses. Currently this *box* supports a *BioVision* Inc. BVH file format. In the figure, there is another *box* called a *MotionToKeyBox* under the *MotionBo*x. This *box* automatically extracts multiple poses, which would become key-poses, from one motion data generated by the *MotionBox*.

Figure 12 show a screen image of key-poses extraction. This figure is concerning a kick motion. Its motion capture data file includes 150 frames (150 poses) and the *MotionToKeyBox* extracted eight frames (eight poses) as key-poses from 150 frames. We checked that the key-poses generate almost the same motion as its original motion of the motion capture data file by their interpolation. As shown in Figure 11, after once several poses are extracted as key-poses, the user edits those poses and then creates new motion.

4. OTHER MOTION EXAMPLES

This section describes another motion editing example.

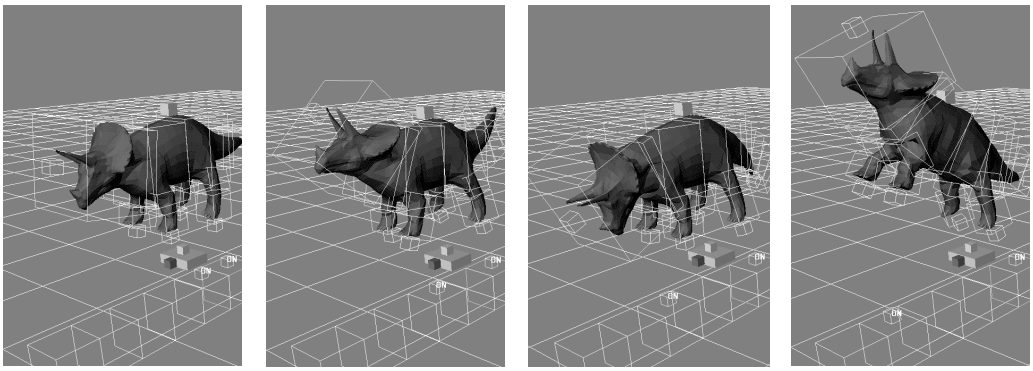


Figure 13. Motion editing example of a triceratops model.

Figure 13 shows four screen images of each different pose of a triceratops. This example uses an *FFDControlBox* to deform a triceratops shape model according to the shape of its skeleton model. Similarly to Figure 5, outside cubic (wire-frame) *boxes* are all *3DRotationBoxes* those work as bones of the skeleton model. The skeleton model is consisted of 17 bones, i.e., 17 joints. Users can define each pose by dragging *3DRotationBoxes* interactively on a computer screen. In this way, using our motion editing environment, users can design any character motions in an intuitive manner as well as design of human-like model motions.

5. CONCLUDING REMARKS

This paper proposed an intuitive motion editing environment in which users can edit a character motion by means of directly defining its multiple key-poses. In the proposed intuitive motion editing environment, all sequential key-poses are displayed as the corresponding CG character's poses on a computer screen simultaneously. Therefore users can see those poses and then rapidly understand the motion that those key-poses mean. We have already developed such a motion editing environment using *IntelligentBox* and described its realization mechanisms in this paper. Furthermore the proposed motion editing environment is realized as a composition of software components. It includes user-defined motion information. So, it is possible to make its copy and transfer the copy to other computer through *Internet*. As a result, users can exchange their defined motions each other through *Internet*.

In the future works, we have to clarify availability of our proposed motion editing environment by evaluation of its efficiency. Currently we have been developing a motion database and a shape model database for easier creation of 3D animation. Our final goal is to build an interactive animation system by which even end-users, especially non-expert users, can create 3D animation rapidly and easily. We will present their new findings as soon as possible.

AKNOWLEDGEMENT

This work is partially supported by research fund of Ministry of Education, Culture, Sports, Science and Technology of Japan, and the Telecommunications Advancement Foundation (TAF) of Japan.

REFERENCES

Gleicher, M., 1997 : Motion editing with spacetime constraints, Proc. of SIGGRAPH'97, pp. 139-148.

Lee, J. and Shin, S.-Y. , 1999 : A hierarchical approach to interactive motion editing for human-like figures, Proc. of SIGGRAPH'99, pp. 39-48.

Okada, Y. and Tanaka, Y., 1995 : IntelligentBox: A Constructive Visual Software Development System for Interactive 3D Graphic Applications, Proc. of Computer Animation ''95, IEEE Computer Society Press, pp. 114-125.

Okada, Y. and Tanaka, Y., 1998 : Collaborative Environments in IntelligentBox for Distributed 3D Graphic Applications, The Visual Computer (CGS special issue), Vol. 14, No. 4, pp. 140-152.

Okada, Y. and Tanaka, Y., 1999 : IntelligentBox: Its Aspect as an Interactive Animation System, Proc. of SCI'99/ISAS'99, Vol.2, pp. 198-201.

Okada, Y. and Itoh, E., 2001 : Aspects of IntelligentBox as an Internet-Supported Tutoring System, Proc. of SAINT2001 Workshops, IEEE Computer Society Press, pp. 27-32.

Okada, Y., Itoh, E. and Hirokawa, S., 2000 : IntelligentBox: Its Aspects as a Rapid Construction System for Interactive 3D Games, Proc. of First International Conference on Intelligent Games and Simulation, SCS Publication, pp. 22-26.

Tanaka, Y., 1996 : Meme Media and a World Wide Meme Pool, Proc. of ACM Multimedia '96, pp. 175-186.

Witkin, A. and Kass, K., 1988 : Spacetime constraints, Proc. of SIGGRAPH'88, pp. 159-168.

Life Forms™ , http://www.credo-interactive.com/products/lifeforms/lf_3-9_studio.html

STRATEGO EXPERT SYSTEM SHELL

Casper Treijtel and **Leon Rothkrantz**
Faculty of Information Technology and Systems
Delft University of Technology
Mekelweg 4 2628 CD Delft University of Technology
E-mail: L.J.M.Rothkrantz@cs.tudelft.nl

KEYWORDS

Games, A.I., Multi-agent, Expert systems, Stratego

ABSTRACT

The field of multi-agent systems is an active area of research. One of the possible applications of a multi-agent system is the use of distributed techniques for problem solving. Instead of approaching the problem from a central point of view, a multi-agent system can impose a new mode of reasoning by breaking the problem down in a totally different way.

In this paper we investigate a distributed approach to playing Stratego. Computational agents that each have their own field of perception, evaluation and behavior represent the individual pieces of the Stratego army.

A first prototype of a framework has been developed that consists of a simulation environment for the agents and an implementation of the agent's evaluation function. The agents have a rule engine that generates behavior that is a resultant of the environment in which they live.

INTRODUCTION

This paper describes a first attempt to play the Stratego game with multiple agents. The Stratego game is a board game where two players battle each other with their armies of pieces. The object of the game is to capture the enemy flag, by moving pieces towards the enemy and trying to capture the enemy pieces. An interesting property of the game is that the information the players have is incomplete, because the identity of the opponent's pieces is concealed until exposed by battles between pieces.

Our motivations for using the multiple agent approach are as follows. When we consider a human society from a central point of view we see that it is a very complex system. A possible attempt to understand the complex behavior of a human society is to consider it as a system that is made up of individuals that have their own characteristics, behavior patterns and interactions with each other. It is the sum of all the local actions and interactions that constitutes the overall behavior of the society. This investigation is an attempt to support this hypothesis by considering the Stratego game. Specifically we want to investigate whether a distributed way of playing this game will provide us with a means to break down the complexity of playing it.

Our work is based on ideas of multiple agents as described by J. Ferber (Ferber 1999) and intelligent agents as developed by P.Maes (Maes 1995) and L.Steels (Steels 1997).

DESIGN

In designing the agents we want to make use of the fact that each piece in the Stratego army has a certain dedicated role. These roles originate from their specific ranks and the rules of the Stratego game. All pieces have secondary goals as well of which possibly the most important one is to stay alive. We propose to define some degrees of freedom in our model of the agent that will allow us to experiment with different types of agents in the Stratego army. Specifically we define for each agent:

- The agent's perception range. Depending on the agent's role in the army the perception will be a diamond of range one to five, or an n x n square of fields. Important pieces will have wider perceptions.
- The agent's 'reactive' behavior. For every agent we define four elementary behaviors that are executed following a reaction in various situations. These behaviors are attack, flee, random walk, and stay and do nothing.
- The agent's 'cognitive' abilities, for example evaluate situation, compute optimal next move, form hypotheses, and make plans.

In our design emotion is modeled as follows. Emotions are related to parameter settings regarding the agent's perception and behavior. For example, if an agent gets upset, afraid or stressed we shrink his field of perception (tunnel view). And if the agent is angry we increase the possibility to attack (McCauley 1998; Scheutz 2000).

We designed two levels of communication among agents. One is communication by means of a blackboard that can be written to and read from by every agent. The blackboard is a container of all information of the board situation that is available. This way all agents can rely on the fact that their field of perception is in accordance with the current board-situation. The blackboard contains strictly information about the board status.

Additionally the agents can use an asynchronous message-passing structure. Agents can send and receive messages to each other containing information about the Stratego battlefield. The communication structure allows sending messages to all other agents, sending messages to agents of a certain rank or sending messages to specific agents. The content of messages can either be known facts, hypotheses or requests.

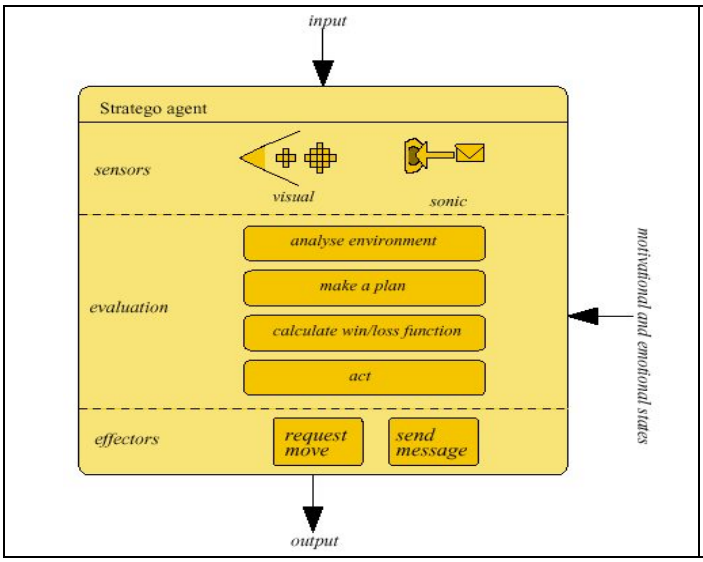


Figure 1: Traditional approach to modeling an Stratego agent based on a functional composition of modules in the evaluation layer.

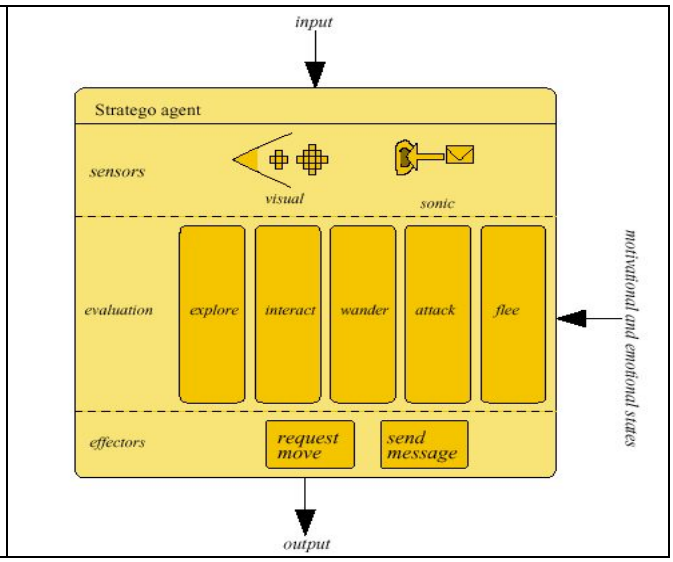


Figure 2: Behavior-based model of the Stratego agent with separate behavior modules in the evaluation layer.

Because only one piece can move at a time, a mechanism was designed that decides which agent is allowed to move. The decision rule was based on scores, where each agent evaluates its current situation and assigns scores to preferences of moving. A higher score will indicate a stronger desire to move and the agent with the highest score will be allowed to move.

ARCHITECTURE OF THE STRATEGO AGENT

For our Stratego agent we defined a three-layered architecture, with a sensor, evaluation and effector layer. These layers relate sensor inputs to actuator outputs. The actual relation between percepts and actions takes place in the evaluation layer. There are various possibilities for filling in the evaluation layer. We discuss the traditional and the behavior-based approach designed by R. Brooks (Brooks 1986).

The traditional approaches to model cognitive systems are based upon a strict functional decomposition of modules. These approaches result in so-called sense-model-plan-act frameworks. The cognitive system contains a number of modules that are built on top of each other, each performing a dedicated function as a part of the system.

One characteristic of these types of frameworks is that every module has a specific function that uses input from the module before it. When applied to the Stratego agent, the traditional framework takes the form as indicated in Figure 1. The three layers, (sensors, evaluation and effectors) are influenced by the motivational and emotional states that the agent undergoes.

The behavior-based approach has the advantage that new modules with new behaviors can be added to the system quite easily. Also, the architecture allows for a combination of modules that may be based on each other or that may be conflicting among each other. It is imaginable that some goals of Stratego agents may very well be conflicting. The architecture of the behavior-based approach seems to be very appropriate for our notion of the Stratego agent, in the sense that for each goal we are able to add a separate behavior module. The three layers are influenced by the motivational and emotional states that the agent undergoes.

As is the case in the subsumption architecture, these modules operate in a considerable autonomous way. The modules shown in Figure 2 are some behaviors that apply to a piece in a Stratego environment. Depending on the situation at hand, one of the behaviors has the overhand and dictates the overall behavior of the agent.

KNOWLEDGE OF THE AGENTS

Since the agents represent pieces of the Stratego army, we want them to express behavior that can be seen as 'rational' from their point of view. In other words, we want them to express behavior that will make the agents successful in achieving their goals. Our approach is based on a rule-set that explicitly defines what to do for a number of situations.

For each of the Stratego agents we have defined a set of rules that specify the behavior, according to the current situation of the agent. We call these rule-sets preference rules, since they indicate preferences to exhibit behavior rather than performing explicit actions. The use of preferences instead of actions in the rules arises from the desire to allow separate behaviors to be activated simultaneously.

We will give some examples of preference rules of the set of 29 preference rules for the "minor"-agent:

Rule 1: This rule will fire the preference "attack" when the following conditions are met:

- Enemy bombs captured
- I have moved
- My rank revealed
- Enemy with unknown rank present at distance 1

Rule 13: This rule will force the preference "flee" when the following conditions are met:

- I have moved
- My rank revealed
- NOT enemy bombs captured
- Enemy with unknown rank present at distance 1

Rule 22: This rule will fire the preference "stay" when the following conditions are met:

- NOT I have moved
- NOT my rank revealed
- NOT enemy bombs captured
- Enemy with higher rank present at distance 1

Rule 27: This rule will fire the preference "attack" if an only if an enemy bomb has been spotted at distance 1.

Upon each move, all agents evaluate their situation and express their desire to act or not. Because of the fact that only one agent can and has to move at a time, one agent has to be selected. This is done according to a weighted function that takes into account all desires of agents. The agent's rule engine has been implemented using the notion of separate behavior patterns that conform to the behavior-based model (Figure 2). The agent's behavior can be explained as being a resultant of all separate behaviors. The agents show emergent behavior that is caused by the sum of all separate behaviors.

A great advantage of this type of emergent behavior is that the agent comes somewhat closer to our notion of an autonomous system. The agent's perception, goals, motivations, etc. all influence the agent's actions. This means that we can define different types of rules for the agent that may harmonize or conflict with each other.

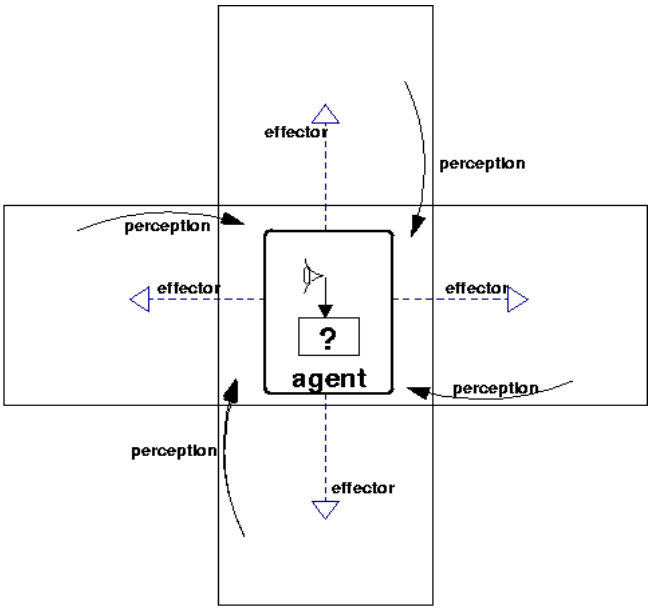


Figure 3: The environment in which the agent playing Stratego lives

IMPLEMENTATION

In this section we will describe an implementation for a prototype called Stratesys, as an acronym of the words Stratego expert system shell. The implementation has been done using the object-oriented programming language Java2. In the current version of the Stratesys we have implemented an agent type that is based on production systems. For the communication among agents and the agent rule-engine we have used the JavaSpaces Technology and an expert system shell called Jess, respectively.

The simulation

According to Russel & Norvig (Russel et all. 1995), an agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors (see Figure 3). All agents have the three layers sensors, evaluation and effectors. See Figure 4 for a schematic view of the agent. Here we can see each layer containing the agent's internals. It also shows the objects it is related to in its environment.

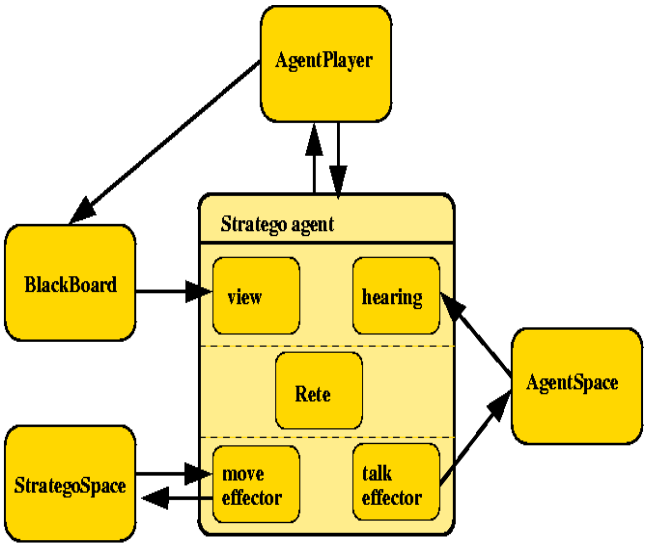


Figure 4: Schematic view of the agent's implementation

The Agent Player functions as a representative of the Stratego army formed by agents. It is responsible for creating all agents upon start-up, initializing them and positioning them on the Stratego board. Also, the Agent Player is responsible for maintaining information on the Blackboard (Cavazza 2000). This is an object that continuously reflects the actual situation on the Stratego board, the way the Agent Player sees it. In other words the Agent Player keeps positions of all pieces and where possible fills in missing information concerning enemy ranks.

The Agent Space is the agent's interface to communicate with its fellow agents (a JavaSpace-service). It is read from by the Hearing object and sent to by the Talk effector. The View object provides the agents with visual perception. It is actually an accurate copy of a small part of the Blackboard. It continuously checks for recent changes on the Blackboard, and updates itself whenever necessary.

The Stratego Space is the communication medium for the Client and the Server. The agent's lifecycle can be viewed as a number of states and transitions. The most important state in the cycle is the Evaluate state. Here, the Rete algorithm is applied using the percepts that have been received. If the Evaluation leads to an action, it will cause a transition to the Sleep state. In the Move state a piece can do an actual move. From the Move state there are two possible transitions to other states. When a move to an empty square was done the agent perceives some changes in

its environment and evaluates them. The other possibility is a battle with an enemy piece. In the Battle state the agent either wins and notifies all fellow agents of the capture, or the agent looses and notifies its death.

The Client-Server model

Since we wanted to be able to play human versus human games, we have created two programs that implement a Client-Server model. The Client is the main Stratesys program. The Server runs in the background, continuously listening for Clients to connect. See Figure 5 for a schematic view of the Client and the Server.

The communication between the Clients en the Server has been implemented by a Java Space-service called 'Stratego Service Space'. Using the space the Clients and the Server can exchange information by reading from and writing messages to the space.

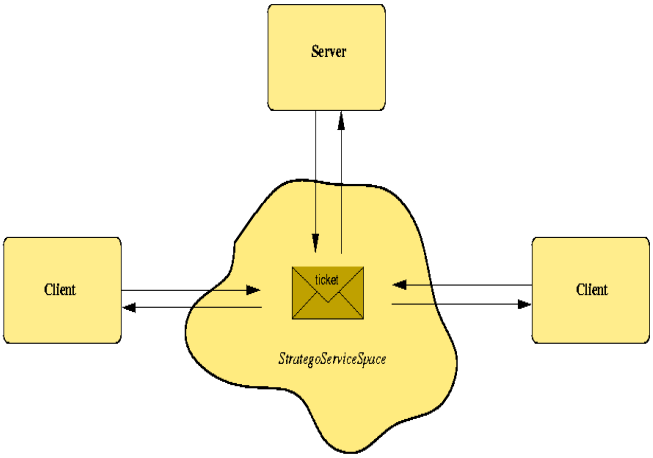


Figure 5: The Client-Server model

The Client is the main Stratesys program. At startup, a window is positioned on the screen with an empty Stratego board. The possible ways of playing the game are a human player playing against an agent army and two human players playing against each other.

At startup, the players will be registered with the Server. The human playing Stratego can position his army by clicking on the squares of the board. A pop-up menu will appear that will allow the player to choose a piece. When all pieces have been positioned, the players can begin to move their pieces. By using mouse-clicks on the squares of the board the human player can select pieces to move. For clarity concerning the situation on board we have chosen to implement the use of animation for each moving piece. When a correct move has been requested the board draws an animation of the moving piece from the initial position to its destination.

Depending on the type of game that is played, one or two Player objects will be created. Only in the human versus human mode will the Client create a one Player object. Naturally this implicates the necessity of another Client in the network. In the other modes of operation, only one Client is used which runs both Player objects. The Player object has both references to its own Pieces and to Enemy Pieces. The Enemy Pieces are actually only 'dummy' Pieces that are a visual representation of the actual enemy pieces. From the Player object to the Server and back are messages to register the player with the board. Messages from Player to Pieces concern position and move messages. The same applies to the messages sent from the Pieces to the Server and back. The Enemy Pieces however only receive messages from the Server and relay them to the Player object. This is because of the fact that these objects are only visual representations, as mentioned before.

Our implementation of the Server can accept two players wanting to play Stratego. These players can reside in one Client program or two. The latter case is only for human versus human games. After the game is over, the Server will wait for new requests for playing. The Client-Server communication consists of four phases. These are registration, positioning pieces, moving pieces and notifying a game over. For each of the phases we have defined specific messages, which we call, tickets.

Tickets are sent as requests and received as answers to that request. The idea behind the concept of a ticket is that a ticket gives a piece the right to position itself somewhere or move to a certain square.

Upon starting the game, the Client creates one or two Player objects, depending on the type of game that is played. The Players send a Registration Ticket to the Server to register. After sending the ticket, they will receive an answer with information about the registration (successful or not).

When two players have registered to the Server, they can position their pieces. For each piece to be positioned a Position Ticket is created and sent to the Server. The Server checks to see if the requested positions are valid, and send the tickets back with this information.

EXAMPLE OF A TEST RUN

In this section we will consider two situations where the sergeant is in the environment as indicated in Figure 6. The sergeant sees an enemy piece with unknown rank (north square) and an enemy scout (northeast square). We will consider the case where the sergeant has already moved and its rank is known. The JESS output gives:

f-51 (enemy-known north east)
f-52 (enemy-unknown north)
f-54 (flee)
f-54 (update scores 0,–200,50,200,50)
f-55 (attack)
f-56 (update-scores 0,50,-50,-50,50)

Let us consider the computation of the scores (see Figure 7) in case that the sergeant has a desire to attack:

Score for staying:	0
Score for moving forward:	-200
Score for moving left:	50
Score for moving backward:	200
Score for moving right:	50

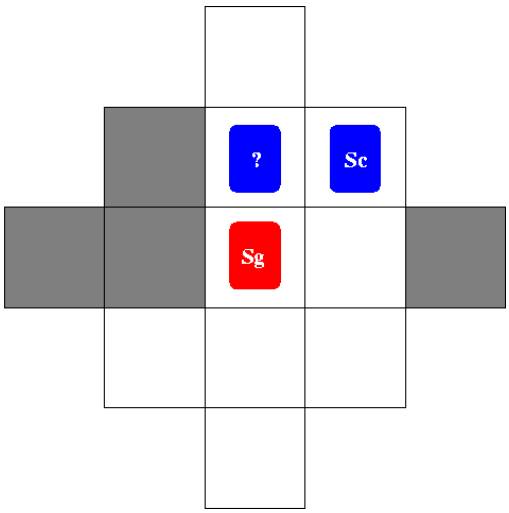


Figure 6: The sergeant's environment

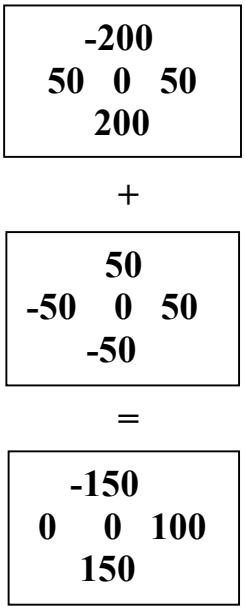


Figure 7: Computation of the scores

In the current implementation of the rule engine, the evaluation consists of a mapping from enemy location to a desire to move (for each direction) or to stay, expressed in scores. In the specific example, the sergeant may want to flee from the unknown enemy. But it also sees an enemy scout that can be beaten. Therefore in this particular case the sergeant's behavior will be a mixture of the desire to flee or to attack.

The scores indicated above express relative desires to go or to attack. Negative scores mean that the agent does not want to go in the corresponding direction. In the example the scores are a resultant of the behaviors to attack or to flee. The fleeing behavior is due to the enemy with unknown rank. Since the sergeant is a piece with a relative low rank, the score to move backward is largest and the sergeant will decide to move backward

CONCLUSION

In this paper we have described a multi-agent approach for playing the game Stratego. This approach involves playing the Stratego game with multiple agents that each represents a piece in the Stratego army. The approach was based on the hypothesis that for some complex problems distributing techniques for solving them can result in more intuitive solutions. We assumed that the Stratego game could serve as an excellent playground for testing the hypothesis. Players have incomplete information on the board status and that results in the high complexity of the game.

We did not make an analysis of the game. We advocate using a corpus-based approach to build up a library of games, which can be used for studies and experiments about Stratego. The Client-Server model that has been implemented provides a framework from which several experiments can be run.

We have tested our prototype program Stratesys by letting the agents play against a human player. The experiments have resulted in some valuable ideas about our multi-agent approach. It proved that playing the game with multiple agents is an excellent approach to break down the complexity of the game.

REFERENCES

Brooks, R.A. (1986). A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation* RA-2:14-23

Cavazza, M, et al (2000), A real-time blackboard system for interpreting agent messages, *Proc. GAME-ON 2000,*49-55

Ferber, J. (1999). Multi-Agent Systems, *An Introduction to Distributed Artificial Intelligence*, Addison Wesley, England

Maes, P. (1995). Artificial life meets entertainment: Life like autonomous agents, *Communications of the ACM* 38, 11:108-114

McCauley, T.L. & Franklin, S. (1998). An architecture for emotion, *AAAI 1998 Fall Symposium "Emotional and Intelligent: The Tangled Knot of Cognition",* AAAI Press.

Russel, S .& Norvig, P. (1995). *Artificial Intelligence-A modern Approach,* Englewood Cliffs, NJ: Prentice Hall

Scheutz, M. et al (2000) Emotional states and realistic agent behavior, *Proc. GAME-ON 2000,* 81-87

Steels, L (1997). A selection mechanism for autonomous behavior acquisition, *Robotics and Autonomous Systems* 20: 117-131

DIRECTIONS FOR FUTURE GAMES DEVELOPMENT

Michael J. Allen, Hussam Suliman, Zhigang Wen, Norman E. Gough and Qasim H. Mehdi

Multimedia and Intelligent Systems Technology Research Group
School of Computing and Information Technology, University of Wolverhampton,
35 – 49 Lichfield Street, Wolverhampton, WV1 1EQ,
United Kingdom
email: ma@scit.wlv.ac.uk

KEYWORDS

Eye Tracking Technology, Artificial Intelligence, Computer Animation.

ABSTRACT

In this paper artificial intelligence (AI), eye tracking technology and computer graphics for character animation are reviewed and current developments examined. In each case the review is followed by a discussion on how these developments may affect the games industry in the future. Game AI and traditional AI methods are investigated as well as the drawbacks of using the traditional approaches in game development. The graphics section examines four of the current methods available for character animation - articulated body animation, single mesh skinning and the N-patch method - and discusses the advantages and disadvantages of each one. The process and evolution of eye movement research is undertaken in the final section which concludes with a list of potential applications for the technology within the Games arena.

INTRODUCTION

In this work, the fields of Artificial Intelligence (AI), computer graphics for computer animation and eye tracking technology are investigated. The aim of the paper is to review how these technologies are currently being used, inside and outside of the games industry, and to highlight where they could be used in the future.

AI is now an established discipline. It enables an electronic system to act intelligently by exhibiting characteristics usually associated with human or animal behaviour. A number of sub-fields have evolved, such as soft-computing, intelligent computational control and intelligent planning, and several other research areas have become synonymous with AI, e.g. machine vision, natural language processing and mobile robotics. Other areas, such as pattern recognition and path planning, may also use AI techniques although alternative non-AI techniques also exist. In Section 2 the use of AI in games is discussed. Game AI is discussed alongside traditional AI techniques and the section concludes with a discussion on computational efficiency and how this affects the use of AI in game production.

With the advance in both computer hardware and software, real time multimedia software applications such as computer games can exhibit much more attractive character animation to the users compared to earlier applications. Furthermore, advances in animation control methods enable the characters to be more intelligent in responding to their environment and the user interaction. Section 3 of this paper starts with the most commonly used character animation method such as rigid-body animation in 3D-computer games, and then describes the single mesh animation method along with the more advanced or recently proposed methods such as mesh skinning and N-patch. The advantages and disadvantages of these methods will also be discussed.

Eye tracking is an emerging technology in which a feature of the eye is tracked to determine where the user is looking. Eye tracking has been used in a number of studies that have analysed where a subject looks when driving, reading, playing fast action sports and interfacing with a computer. Compression techniques, such as the content-based functionalities of MPEG-4 and foreground - background, encode Regions Of Interest (ROI) within an image with more bits to improve the perceptual quality of the encoded image. The ROI are those areas where the user/operator's eye tends to fixate. Section 4 begins with a review of the research undertaken into eye tracking to date and concludes with a list of potential application areas where this research could, and is, being applied in the area of games.

AI IN COMPUTER GAMES

Game AI

There are no generally-accepted guidelines for what methods are included under *games AI*, but the broad definition given by Wright & Marshall (2000) 'Game AI is the high-level control code that determines the behaviour of game agents' is widely accepted. Strictly speaking, many games merely create the illusion of intelligence. Games do not need to process every possible object at every frame in order to create an illusion of

believability. Wright & Marshall (2000) go on to observe that 'virtual worlds built for entertainment purposes need to concentrate on presenting a believable world to the player and need not concentrate on presenting an accurate simulation of the real world'. Furthermore, games AI does not necessarily imply that any traditional soft computing tools such as neural networks are being employed. Hence purists often refer to games AI as "cheating" (Smith, 2000). However, for the purpose of this research, we will adopt a less rigorous definition of AI as one that creates believability in the sense that games characters appear to be interacting intelligently with one another and their environment.

The most common and popular AI technologies (Woodcock 1999, 2000) for Non Player Characters (NPCs) specification are based on *Finite state machines* (FSMs) (Gough *et al*, 2000, Hein 1996, Funge 1999, Hopcroft & Ullman 1979) that are machine-like representations of rule based systems. Hence all reflex or reactive agents that react to a rule of the form *if condition then action* can be implemented by FSMs. They are characterised as having sets of distinct internal states that respond to a sensory input transit to a new state and release an output. FSMs are used in layered or hierarchical structures (Woodcock 1999,2000, Funge 1999) for implementing and customising behaviour. Computer games such as ID software's *Quake* I, II and III use FSM modelled systems. The Valve software team (Valve 2001) used a 'schedule driven' (event driven) FSM for their award winning game *Half-Life* (GameAI 2000). *Unreal* and *Unreal Tournament*, for example, demonstrates the complexity of behaviours available using FSMs. The AI makes extensive use of FSMs to control the behaviour of the player's opponents to an amazingly realistic degree. For example, the monsters appear to exhibit considerable intelligence, as they run away, hide when wounded, call for reinforcements, and even trick the player into ambushes. In addition, group movements exhibited through the use of a flocking algorithm add considerably to the realism, such as the method of Reynolds (1987) used in *The Lion King*. All of these effects were achieved by the developers through the implementation of layering FSMs, which were built on top of an extensible scripting system called *UnrealScript*.

Path finding and *planning* methods for agent navigation are also extensively used in games. The basic problem is to find a route through a network of possible locations in the presence of constraints, such as take the shortest distance, avoid ambush, minimise the number of locations visited, *etc*. There are many well-known search algorithms used in path finding such as b*readth-first*, b*i-directional breadth-first, Dijkstra's algorithm, depth-first* and *A** (Pearl 1984, Russell & Norvig 1995). They share some common features and especially the strategies by which they find paths. One of the most popular algorithms for finding shortest distance paths in games, *A**, is a heuristic search that ranks a node by an estimate of the best route that goes through that node. It combines the tracking of past path lengths with the heuristic of breadth-first search (Hart *et al* 1968,1972, Russell & Norvig 1995).

Path finding in practice also requires obstacle avoidance. However, games involving 2D and 3D virtual environments often avoid this problem by using networks with pre-stored nodes through which NPCs must travel, thus ignoring the existence of obstacles altogether. In a more complicated scheme, the space can be presented as a uniform grid, through which NPCs freely travel in any direction but this requires the development and implementation of complex obstacle avoidance systems for NPCs to 'sense' the obstacles. For example, *influence maps* or *field methods* (Boas 1983) consider objects such as locations or obstacles as field sources that exert influence on distant points.

Game companies are now focusing on extending these techniques for specific games (such as path finding in 3D space). Path finding tools are also beginning to take account of environmental terrain. Terrain analysis is a more difficult than simple path finding in that the game AI must take into consideration geographic features but it has been used successfully, particularly in military games scenarios (Smith 2000).

One of the most interesting topics that has attracted game developers recently is the development of AI software development kits (SDKs). Academics and developers alike have found them to be powerful tools for both game development and AI research. There are many AI SDKs available to the game developers but most of them are intended for use in industrial or business applications. *DirectIA* is an agent based toolkit that uses state machines to specify behaviour. Also using state machine modelling is *GSM Suite,* a set of programs for using FSMs in a graphical fashion. The suite consists of programs that edit, compile, and print state machines (Game AI Page 2001).

Artificial Intelligence Tools

Traditional artificial intelligence has been exemplified by a set of methods commonly referred to as soft computing tools. These include expert systems, fuzzy logic, artificial neural networks (ANNs), evolutionary algorithms, and probabilistic reasoning (McCarthy 1987, Nilsson 2001, Russell & Norvig 1995). Many hybrid methods have also emerged from these basic methods, such as distributed AI, which is logic based AI.

Logical AI involves representing knowledge as sentences in logic (McCarthy 1958, 1987, Poole *et al* 1998, Russell and Norvig 1995). It is used to develop computer programs that represent what they know about the world

primarily by logical formulas and decide what to do primarily by logical reasoning, *i.e.* applying inferences to statements to draw conclusions. The proponents of the symbolic AI approach to intelligence use logic based systems for building intelligence into agents. Many AI systems represent facts by a limited subset of logic, *e.g.* logic programming restricts its representation to *Horn Clauses* (Nilsson 2001, Russell & Norvig 1996), databases often use only *ground formulas* and hardware design usually involves only *Propositional Logic* (Nilsson 2001, Russell & Norvig 1995). Theses restrictions are almost all justified by considerations of computational efficiency.

The goal-based agent architecture used in Quake II games is an example of a rule-based production system. Based on the *SOAR* model of Van Lent & Laird (1999) it uses an arbitration method to decide what action to take when the percept matches more than one rule.

Fuzzy logic (Zadeh 1965, Dubois & Prade 1980) is one of the main components of AI. In a narrow sense it is a branch of multi-valued logic, which provides approximate reasoning in logical operations. It has found applications in control systems, business and computer games. Fuzzy logic developed by Lotfi Zadeh is based on a concept known as *fuzzy sets* (Zadeh 1965), introduced to handle uncertainty and linguistic variables. The development of fuzzy logic has led to many successful implementations of fuzzy systems. A fuzzy inferencing system (FIS) uses the fuzzy sets in a rule-based system to make decisions or draw conclusions. These rules are then combined using *rule composition* and firm conclusions are drawn through *defuzzification*. *Fuzzy State Machines* (FzSMs) (Dubois & Prade 1984) are machine-like representations of rule based systems that have fuzzified states for more modelling power.

Connectionism is a style of modelling based on networks of interconnected simple processing devices. Connectionist systems, also referred to as *Neural Networks* or *Artificial Neural Networks (ANNs)* (Bishop 1995, Hertz et al. 1991, Russell & Norvig 1995). ANNs appeal to many AI researchers due to the analogy to the structure of the human brain and the basic building block, the neuron. The earliest work in neural computing goes back to the 1940s when McCulloch and Pitts introduced the first model. In the 1950s, Rosenblatt's work resulted in a two-layer network, the perceptron (Minski & Papert 1969, Hertz *et al* 1991), which are capable of learning certain classifications by adjusting connection weights. Recent work includes Boltzmann machines, Hopfield nets, Radial Basis Function networks, competitive learning models, and Adaptive Resonance Theory models (Bishop 1995, *Hertz et al.* 1991). ANNs are strongly implicated as robust methods for detecting and classifying patterns in data. Few of the top 100 games claim to use ANNs. *Battle Cruiser:3000AD* developed by Derek Smart (GAMEAI Page 2000) claimed to be the first commercial game to feature ANNs for route finding and goal-oriented design but it used fuzzy logic where the networks were considered to be inadequate. Watson (1996) used a neuro-fuzzy ANN with back propagation to control a space ship on interstellar missions. An example of a computer game using an ANN has been produced and is described in Medhi *et al* (2000).

Evolutionary or *Genetic algorithms (GAs)* (Griffiths *et al,* 1997) are optimisation techniques that are inspired by how animals evolved over time. The idea centres on the principle of survival of the fittest whereby members of a species compete over limited resources. The environment poses a test which can favour particular elements of a species over time. The surviving genes present the codification of the solution to the optimisation problem expressed in a fitness function. These optimisation techniques have received comparatively little attention in games dues to the vast computational costs. However there is potential to evolve agents in artificial life (A-life) communities. One game that claims to have used GAs is *Creatures* from *Cyberlife* (Grand *et al* 1997). GAs have also been used to evolve, controllers for characters and for changing the characters body shapes (Taylor, 2000).

Computational Efficiency

'The fastest computation is the one you don't have to compute'

John Carmack (ID software)

Clearly, any increased complexity of game AI can only be achieved at a price. Firstly, it brings increased computational overheads. Furthermore, the emergent behaviours may be so complex that they interfere with game play and may have to be curtailed. For example, the A-life techniques used to control NPCs in the online game *Ultima Online* were so rich that they had to be compromised in the interests of gameplay (Woodcock, 1999).

Game developers inevitably take short cuts to minimise the computational load. For example, the FPS game *Unreal Tournament* demonstrates realistic bot AI that can compete strongly with the best of players yet is also fun to play against. It intentionally makes mistakes and does not always act optimally. However, it does not incorporate any AI in a strict sense. Although, developers are now looking towards more sophisticated ways of incorporating AI in the traditional sense, they are unlikely to accept them if they involve excessive computational overheads.

In the early days of the gaming industry, tight and often unreadable code was common place. The demand for sophistication in agents and more complex virtual environments, means that recent game designs tend to be well-structured showing an integrated system of

formalised objects and relations for specifying the agent and its environment. This is clearly observed in the scripting languages accompanying games with extensible AIs and Internet released sources of code. Notwithstanding the increasing speed of CPUs today, this clearly serves to emphasise the need for computational efficiency (Rollings & Morris 1999, Saltzmann, 1999).

One approach to saving computational load is to increase the amount of RAM accessible by the games engine. For example, in agent navigation, many paths can be pre-computed and stored in a look-up table, such that a path is chosen by associative recall of locations along the route. Consequently, computationally expensive search algorithms should be applied only when there is no alternative solution. Similarly for obstacle avoidance, memory can aid significantly by explicitly storing the distance to a nearest obstacle rather than incorporating complex strategies.

Formal representations, such as FSMs tend to suffer from the "curse of dimensionality" in that the dimension of the state space increases alarmingly as the number of characters or scenes is increased. A common solution to this problem is to partition FSMs at natural boundaries, such as different scenes, each scene being handled by a separate FSM. This helps to make specification, design and coding manageable by reducing the dimensionality (Wright, 2000). At run time, since scenes are displayed sequentially, the computational load imposed by AI can be distributed on a frame-by frame basis and as in any real-time control system with hard real-time constraints. Details of how to design a games engine to spread the load are given by Wright and Marshal (2000).

Learning architectures such as NNWs or the explanation-based learning used in the SOAR architecture also raise practical difficulties since they normally involve training phases. The need to pre-train before a game is ready for use is not likely to prove popular with game players. However training could be achieved in real-time using (say) reinforcement learning, but this would add a significant computational overheads and research is required to discover under what circumstances it could prove to be feasible.

In summary, the development of new games AI methods should always be accompanied by a detailed analysis of the extra computational load required and limited benefits should never be sought at the expense of excess computational effort.

ADVANCES IN COMPUTER GRAPHICS FOR GAME CHARACTER ANIMATION

This section gives a review of advances in real time computer graphics and animation techniques for 3D computer games. Today's computer game's characters have become very realistic both in terms of their appearance and their intelligence. With the advance in graphics hardware and software, the CPU has been greatly relieved from the traditional over-head 3D graphics processing requirement so that more sophisticated game physics and artificial intelligence algorithms can be applied. For instance, the two most powerful graphics chips for PC are Geforce 3 and Radon 8500; their fill rate now reaches 3.8 billion per second and the memory bandwidth reaches 8.2 GB/s, which promises much more realistic scenes while maintaining high refresh rate (Nvidia, 2001). This review places the emphasis on character animation methods.

Articulated Body Animation

The ***articulated rigid body animation*** technique has been used in previous 3D game character animations such as the Tomb Raider series. The 3D character is made up of a series of hierarchical geometry. The articulated structure can be thought as a hierarchy of nodes with an associated transformation which moves the link connected to the node in some way (Watt and Policarpo, 2001). Kinematics techniques including forward kinematics and reverse kinematics are widely used in producing the character animation. During the animation, the transformation matrix for the geometry which is at the end of the hierarchical link can be calculated by traversing from the root geometry down to the end in forward kinematics and there is only one possible solution for the kinematics equation. However, in inverse kinematics, the procedure needs to be reversed. The end effector of a hierarchical link is defined and all the transformation matrices in upper hierarchical level need to be calculated by incorporating the lower hierarchy transformation and there may be infinite solution for the kinematics functions. A project is being conducted by Balder (Balder, 2001) that aims to derive fast, analytical methods for solving inverse kinematics problems in real time virtual human animation. The advantage of this technique is its relatively fast execution speed - even without the assistance of graphics acceleration hardware which was originally not popular due to its high price. Furthermore, the memory requirement of this method is small since only one copy of the character model data needs to be kept in memory at run time. The disadvantage of this method is the inevitable *gaps* between the separate parts, especially when the character makes big movements. Figure 1 shows this *gap* in a game character animation from the Tomb Raider series. The reason for this gap is that the intersection part of two separate parts is not transferred correctly, nor is the calculation of the lighting normal, which results in the wrong shading effect. This problem could be relieved to some degree by increasing the polygon number for the character, or using some "trick" such as wrapping some texture around the joint which is most likely to create the "gaps" during big motion.

Figure 1 Articulated rigid body animation in 3D game, screen shot from Tomb Raider: The last revelation Demo

Single Mesh Blending Animation

Due to the "*gaps"* problem, most of the present day 3D computer games use the ***single mesh blending*** method to animate the character movement. This technique models the character in a single mesh rather than separate parts, which therefore avoids the "*gaps"* problem found in the articulated rigid body animation method. All the poses of the character need to be modelled and saved as key frames of the animation. During the program run time, various interpolation methods are applied to perform dynamic blending among these models. Linear interpolation is the most common and fastest method but lacks smoothness in some situations. More complex interpolation algorithms, such as Hermite spline interpolation, have been proposed to improve the smooth transition between the animation key-frame. As this method needs to hold four vertices information rather than two as in the linear method and the calculation of vertex position involves cubic functions, more computation is needed (Cebenoya 2001). However, this method can potentially reduce the memory usage as greater space between the key frames can be allowed. It means fewer key frames are required to produce the same desired animation. Meanwhile, modern graphics chipsets can achieve this complex interpolation method in hardware which means that real time speed can be guaranteed. It is also noted that today's advanced graphics rendering API such as DirectX support mutli-stream data rendering in hardware, which is also able to deal with this multi-stream data fetching issue efficiently (Taylor 2001). Successful examples of the application of this character animation method are the Quake series or the even more recent game Max Payne (Figure 2). The most attractive advantage for this animation method would be its smooth animation and fast execution speed due to the support from hardware. The disadvantage of this single mesh blending method is its lack of flexibility. As all of the animation poses have been defined as key frames, it leaves relatively little flexibility for the programmer to change the way the animation procedure is executed. It also means the animation process becomes very difficult or impossible to be parameterized. This disadvantage implies that this method may not be the best

(a) Screen shot from Quake3 Team Arena Demo

(b) Screen shot from Max Payne Demo

Figure 2 Mesh blending in real time 3D games

solution for intelligent character animation, which ideally requires some non-repetitive behaviour.

Mesh Skinning Animation

The ***mesh skinning*** technique as a new real time game character animation technique was proposed to combine the advantages of the articulated rigid body and the single mesh blending methods. It inherits the advantage of being controlled and parameterised from the articulated body animation method by establishing a hierarchical body structure for the game character. The programmer has the control of the transformation for each bone. Therefore, kinematics can be applied to produce decent character motion. Mesh skinning is also free from the "gaps" problem by allowing more than one transformation matrix to affect the character's skin vertices. Figure 3 is a screen shot from a mesh skinning character animation. It is believed that this technique will widely be used in future generation game character animation although it requires more computational power to deal with the additional one or more transformations. It is noted that from DirctX8, this leading game development API has begun to support this technique at both software and hardware levels. The *Vertex shader* technique allows the programmer to deal with the traditional matrix transformation for both the vertex's position and lighting normal in a customised way.

Figure 3 Mesh skinning in character animation

Therefore, any special graphics effects involving transformation matrices are now becoming practical for real computer games. Lindholm from Nvidia Company present a user programmable vertex engine in the SIGGRAPH conference by using the DirectX *vertex shader* and *Geforece 3* graphics card (Lindholm et al. 2001). The potential of this research is quite promising. They even successfully implemented the famous "Artificial Fish"(Tu and Terzopoulos 1994) in real time and also a real time ray tracing simulation, which were limited to special graphics workstations in the past. Mesh skinning is supported at the hardware acceleration level, which makes it likely to be the dominant character animation method in the future.

N-patch Method

The N-patch character animation method is based on the mathematical interpolation algorithm. N-patch is an interpolating triangular cubic bezier surface. It is based on higher order interpolation functions such as cubic or quintic rather than traditional linear interpolation. For example, given a triangle with 3 points, further 6 points on the boundary are calculated by projecting the edge vector into the plane defined by the 3 original points' normal. The interior points are then calculated from these 9 key controls point (Hart 2001). The resulting vertices are fed to the *vertex shader* for further transformation matrix manipulation. The advantage of this character surface construction method is its arbitrary topology and fast execution speed in hardware as it is supported by the most popular APIs such DirectX and OpenGL. In addition, it is compatible with existing graphics data structures and therefore can be combined with any subdivision, skinning or tweening techniques. Several animation demos from id Software using this method were presented at the Game Developer Conference (Figure 4) (Hart, 2001).

The N-patch method is primarily motivated for character animation. However, it can be applied to generate other environmental objects' surfaces such as terrain or water.

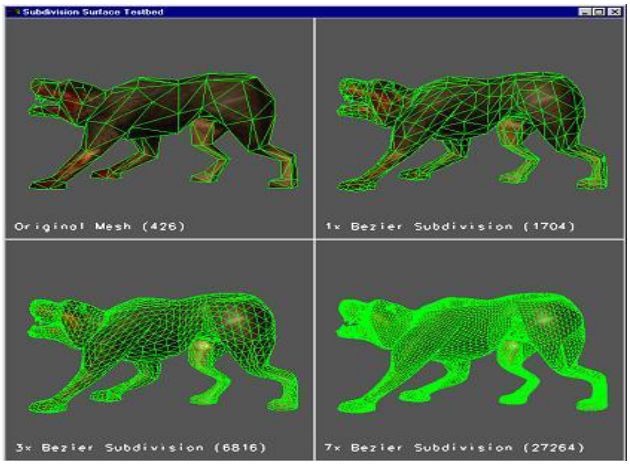


Figure 4 Higher order surface in character animation

APPLICATION OF EYE MOVEMENT TECHNOLOGY IN COMPUTER GAMES

A keyword search on the INSPEC database from the years 1969 to 2001 on 'games' reveals 11127 results. However, when using the keywords 'eye movements and games' only 4 results emerge and the most recent of those was 1993/1994. This suggests a lack of interest on behalf of games researchers to embrace eye movement technology. There appears to be a number of potential application areas within the field of computer games where eye movement technology can, and is, being applied.

In this section a brief review of eye movement research is presented which considers eye movements in humans and regions of interest, i.e. where do humans look. A section on modelling the operation of the human eye is included and in the final sub-section eye movement research is discussed in the context of games development.

Eye Movements in Humans

'The impression that we are aware in detail of our surroundings has been referred to as the "Grand Illusion"' (Harris and Jenkin, 1998). In daily life we are aware of entities within our environments without perceiving them, i.e. we automatically avoid objects and people in the street but cannot recall what or who they were and what they looked like.

Human vision is a dynamic process. The eye constantly moves and representations of the world around us are built up over time using multiple eye fixations. It has been established that eye movement behaviour can be divided into two discrete phases – ***fixations*** (when the point of regard is held relatively still) and ***saccades*** (where the eye rotates to re-orientate the point of regard

from one position to another) (Hendersen and Hollingworth, 1998). The fovea is an area at the centre of the retina where our ability to define fine detail and colour information is best. Consequently, the eye moves to focus regions of interest (ROI) within the scene onto the fovea.

Humans actively seek information. When humans see and understand they actively look (Aloimonos *et al*, 1987). In their review, Hendersen and Hollingworth (1998) cite a number of studies (Buswell, 1935; Yarbus, 1967; Antes, 1974; Mackworth and Morandi, 1978; Hendersen *et al*, 1999) that suggest that during scene viewing the fixation points are non-random with more fixations directed towards the more informative regions – ROI. In each case the eye movements were recorded when the viewer was presented with static images, e.g. colour paintings and black or line drawings.

Norton and Stark (1971) coined the term ***scanpath*** to describe the sequence of fixations used by a particular viewer when viewing a particular pattern. As with the studies listed above, this work was also conducted with static images back-projected on to a screen. The study examined how viewers examined new (previously unseen) patterns during a 'learning phase' and then how they recognized the same patterns during a 'recognition phase'. The result of the experimental work showed that when recognizing a pattern the viewer enacted an appropriate scanpath in 65% of the cases. The authors regarded this as a strong result and suggested that the presence of scanpaths is indicative of how patterns are remembered and they provide evidence that high-level processing is used for visual perception and eye movement control (Top-down control).

Stark *et al* (2001) present a review of current scanpath theory for top-down visual perception. This review includes a general explanation of early and current scanpath theory gathered from experiments with static images and when looking at dynamic scenes. A detailed description of their experimental work is given – which includes work monitoring eye movements during the viewing of animated scenes - and the presentation concludes with a report on the implementation of a top-down computer vision model.

However, top-down control is not the only control mechanism. Low level signals highlight areas of potential importance, e.g. our peripheral vision is particularly sensitive to movement, and can influence eye movement control (Bottom-up vision). When modelling the human model Stark *et al* (2001) refer to the problem of how to match bottom up confirmatory signals from the peripheral field with detailed information that is extracted from the fovea.

Regions of Interest

The eye moves to focus regions of interest on to the fovea, but what makes one region more interesting than another? Hendersen and Hollingworth (1998) discuss ***semantic informativeness*** - the meaning of a region - and ***visual informativeness*** - discontinuities in colour, depth, luminance and/or texture. The scanpath theory indicates that the use of semantic information is used extensively in pattern recognition. The eyes follows a scanpath to gather salient features about the scene to match against the viewer's cognitive model. The scanpath is controlled by high level processing within the central nervous system which in turn draws upon memory. However, Stark *et al* (2001) identify ***task setting*** as an important factor that can induce the viewer to modify the scanpath. Hendersen and Hollingworth (1998) also identify task setting as a factor that can influence eye movement patterns and also include factors of ***viewing time*** and ***image content***.

In computer games the player has to analyze data from a dynamic scene in real time. Often, as events on the screen become more intense, the real time window becomes smaller and the deadlines more important, i.e. the viewing time is decreased. It has been established that the human model combines visual data and experience (expert knowledge) to anticipate future events in order to reduce reaction times. In fast action sports, experts use visual information, such as the body movements of a competitor, to predict their subsequent actions and to gain an advantage. The expert is able to distinguish between the irrelevant and relevant information and to recognize these cues early in the sequence of events (Paull and Glencross, 1997). This also applies to experienced and novice drivers. An examination by Chapman and Underwood (1998) identified that, when viewing a scene novice drivers, in general, fixated longer than experienced drivers and it was argued that this related to the additional time that is required by novice drivers to process visual data from the scene. The study also revealed that different visual scene (rural and urban scenes) also invoked different observation behaviour (influence of image content).

Modelling Eye Movements

If a model of eye movement behaviour can be constructed, even a simple model to identify the density of fixations in different parts of a scene, it could be used to identify the ROI in different game scenarios. The model would have to take into account the factors listed above the influence the eye movements, e.g. task setting, viewing time and image content, and whether the player is a novice or expert. A knowledge of where the game player is looking could facilitate the design process, i.e. more attention could be paid to ROI and less to those of

lesser interest, and for playing interactive on-line games where multi-resolution meshes are employed.

Hendersen *et al* (1999) describe a model of eye movement control in scene viewing. The control method uses a ***saliency map*** to determine the subsequent fixation point. The saliency map contains a set of weights that are initially assigned based on the saliency of a region based on a parse of the entire scene. It is suggested that these initial saliency metrics are assigned by low-level stimulus factors such as luminance, contrast, colour, etc, i.e. visual information as opposed to semantic information. The attention is turned to the region with the highest weight. Over time as the information within the regions of the map are processed and become meaningful elements the saliency weights are modified to reflect the semantic importance of the regions. Consequently, the basis for each weight gradually moves from being visually based to semantically based.

The present authors (Allen *et al*, 1999) describe a method for efficiently scanning image frames using a method inspired by the operation of the eye. A schematic breakdown of this system is shown in Figure 5. As with the method described above this process assigns weights to regions of the scene. However, a Fuzzy Inference System is used to assign the weights (*possibility* values) based on historical data relating to the proximity of the region to other ROI; the attention paid to it in the past; the success of the system in locating known objects; and the position of regions that are known to be constantly important. These *possibilities* are stored in a ***saccade map***. The regions to be examined during a particular scan are chosen using a weighted roulette wheel, i.e. those with the highest weights have more chance of being selected. The 3D plot in Figure 6 shows the attention paid to different areas of a dynamic scene over 60 frames. The frames contained one static object and a mobile object. It can be seen that more attention was paid to those regions containing the objects. In the form described, this method is not intended to model the movements of the eye. However, the behaviour of the fuzzy system could be retuned to handle new input data and resemble more closely the operation of the eye.

Eye Movement Technology and Computer Games

A study was undertaken at the University of Hawaii (Crosby and Chin, 1997) that utilized eye-tracking technology to identify whether embedded information is more difficult to interpret than unembedded information when using multi-user interfaces. This study identified where users of a user interface in emergency conditions. A similar approach could adopted for examining the eye movement behaviour of game players who will also be interacting with a video image whilst under pressure.

In most game scenarios the game area is dynamic, particularly so in action games. A number of methods for capturing eye-movements whilst viewing dynamic scenes have now been developed and used in experimental projects (Stark *et al,* 2001; Land, 1998). If the eye-movements of players could be modelled then it would give an insight into how expert and novice game players gather information when playing particular games and this may provide useful data on how the game could be improved.

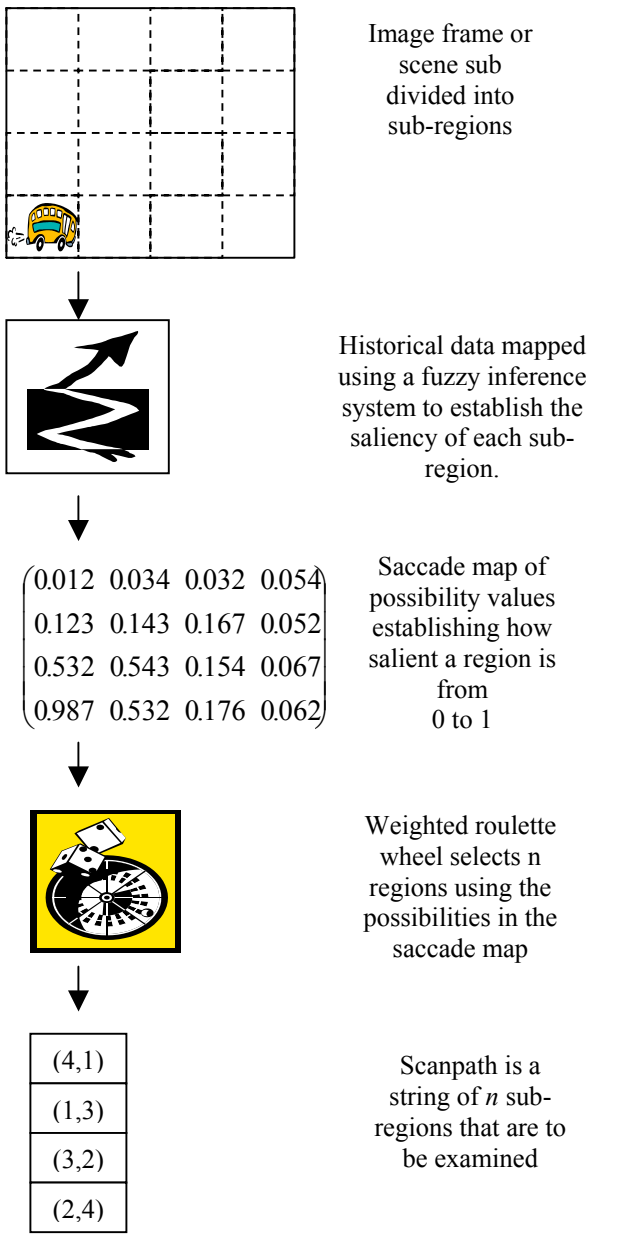


Figure 5 Construction of a scanpath from a saccade map

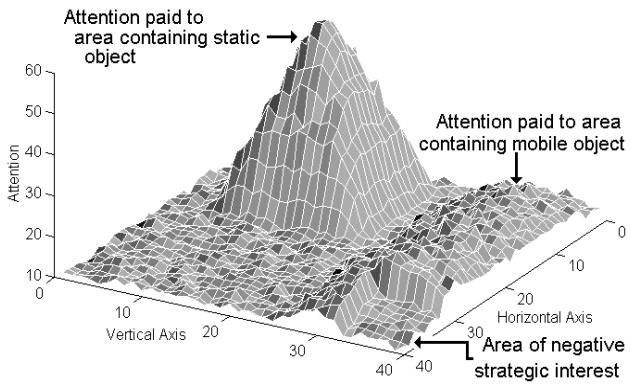


Figure 6 Attention shown to sub-regions of the frames in a sequence containing a static object and a mobile object

Interactive on-line computer gaming is an area where eye tracking technology can be employed to maximize the quality perceived by a subject viewing on-screen graphics at interactive frame rates. The level of accuracy in a transmitted image sequence can be sacrificed to improve speed. O'Sullivan *et al* (2000) discusses the technique of Interactive Perception of Multi-resolution Meshes (IPoMM) which is an interest-dependent strategy where the most interesting part of the scene is retained at a higher resolution for as long as possible. The region of interest is determined using eye movement technology.

As the intelligence of NPCs improves, expert knowledge of where to look at the player-character, say during a football match, would add to the realism of the character. Like the player character the NPC would have to learn to use visual cues to gain an advantage over his/her opponent.

Research has been undertaken to ascertain how professional racing drivers view the track when they are racing. These eye movements could be compared with those of an experienced player to assess whether a particular racing game was realistic. The game could be improved by including more visual cues on the lines of those incorporated at the real tracks.

The apparatus required for tracking eye movements has been relatively expensive and cumbersome to wear. As a consequence, it is currently most valuable to disabled people and researchers. However, Canon have developed a miniaturized eye-tracker and included it as a means of directing the automatic focusing system on two of their cameras. The EagleEye project (Gips and Olivieri, 1996) demonstrated how eye movement apparatus could be used by disabled people in place of a mouse. If the price of eye tracking instruments falls then the scope of their use in virtual reality will increase. Game playing could be made available to the disabled and these controllers could be used as an additional form of on screen control.

CONCLUSIONS

In this paper, various techniques that might be used in games development have been examined and reviewed. The area of graphics is well known to the games industry and the field of AI is attracting more interest. Eye tracking technology is possibly less well known but may have much to offer in improving games in the future.

Section 2 discussed AI in computer games and introduced the Game AI methods commonly used by the games industry. This was followed by a section considering games that use traditional soft computing methods and went on to discuss computational efficiency. In summary, the development of new games AI methods should always be accompanied by a detailed analysis of the extra computational load required and limited benefits should never be sought at the expense of excess computational effort.

Four different approaches to graphic animation were presented – the articulated body animation, single mesh skinning and the N-patch method. Although the articulated body animation approach is used in numerous adventure games, the single mesh skinning technique is currently the most common method. However, although not widely used at present, we suspect that the N-patch method will become more popular in the future.

Eye tracking is a technology that is currently expensive but, as the technology improves, the cost of the hardware will go down and make it more accessible to a wider range of applications including games. Currently, the technology can be used to develop more effective displays, i.e. for on-line gaming where data transmission rates may be low, and in the future with added control for the player who will be able to utilize eye movement to control aspects of the game. The technology will also open up computer gaming to people with disabilities.

REFERENCES

Allen M. J., Q. Mehdi, I. J. Griffiths, N. Gough, I. M. Coulson. 1999. "Object location in colour images using fuzzy-tuned scanpaths and neural networks." *Proc. 19th SGES Int. Conf. Knowledge Based Systems and Applied Artificial Intelligence*, Cambridge, UK, pp 302-314.

Aloimonos J. Y, I. Weiss and A. Bandyopadhyay. 1987. "Active Vision." *Int. Jour. Of Computer Vision*, pp 333-356.

Antes, J. R. 1974. "The time course of picture viewing." *Journal of Experimental Psychology,* 103, 62 – 70.

Balder N. 2001 "Real-Time Inverse Kinematics for Human Animation"

http://www.cis.upenn.edu/~hms/research.html, (last accessed 1st November 2001).

Bishop M. C. 1995. *Neural Networks for Pattern Recognition,* Oxford University Press.

Boas, M. L. 1983. *Mathematical Methods in the Physical Sciences*, John Wiley & Sons.

Buswell, G. T. 1935. *How people look at pictures*, Chicago: Chicago University Press.

Cebenoyan, C, Nvidia Company. 2001"Efficient animation",. http://www.nvidia.com, last accessed 10 October 2001.

Chapman P. R. and G. Underwood. 1998. "Visual Search of Dynamic Scenes: Event Types and the Role of Experience in Viewing Driving Situations." *Eye Guidance in Reading and Scene Perception*, Elsevier Science Ltd, pp 369 – 394.

Dubois, D. and Prade, H. 1980. *Fuzzy Systems*: *Theory and Applications*, Academic Press, New York.

Funge, J. D. 1999. *AI for Computer Games and Animation: A Cognitive Modeling Approach*, A K Peters Ltd.

Game AI Page 2001. "Games Making Interesting Use of Artificial Intelligence." http://www.gameai.com/games.html, (last accessed 1st October 2000).

Gips J. and P. Olivieri. 1996. "EagleEyes: An Eye Control System for Persons with Disabilities." http://www.cs.bc.edu/~eagleeye/papers/paper1/paper1.html (Last accessed 23 October 2001)

Gough, N.E., Suliman, H**.** & Mehdi, Q. 2000. "Fuzzy state machine modelling of agents and their environments for games." *Proc. 1st SCS Int. Conf. GAME-ON 2000*, Imperial College, London, pp 61-68.

Grand, S., D. Cliff, and A. Malhotra, 1997. "Creatures: Artificial Life Autonomous Software Agents for Home Entertainment." *Agents 1998 Conference Proceedings, ACM*.

Griffiths, I.J., Mehdi, Q.M., Wang,T. and Gough, N.E. 1997. "A genetic algorithm for path planning" Report AC-96/420, SCIT, University of Wolverhampton, *Proc. 3rd IFAC Symposium on Intelligent Components and Systems*, Annecy, France 1997.

Harris L. R. and M. Jenkin. 1998. "Vision and Action." *Vision and Action*, Cambridge University Press, pp 1 – 12.

Hart, E. 2001 "Character animation", http://www.ati.com, (last accessed 10 October 2001).

Hart, P. E., Nilsson, N. J. & Raphael, B. 1972. "Correction to 'A formal basis for the heuristic determination of minimum cost paths'" *SIGART Newsletter*, 37:28-29.

Hart, P. E., Nilsson, N. J. and Raphael, B. 1968. "Formal basis for the heuristic determination of minimum cost paths" *IEEE Trans.on Systems Science and Cybernetics*, SSC-4(2):100-107.

Hein, J. L. 1996. *Theory of Computation*; A*n Introduction,* Jones and Bartkett Publishers International.

Hendersen J. M. and A. Hollingworth 1998. "Eye Movements During Scene Viewing: An Overview." *Eye Guidance in Reading and Scene Perception*, Elsevier Science Ltd, pp 269 - 294.

Hendersen J.M., Weeks Jr and A. Hollingworth. 1999 "The Effects of Semantic Consistency on Eye Movements during Complex Scene Viewing" *Jour. Of Experimental Psychology: Human Perception and Performance*, Vol. 2, No 1, pp 210 – 228.

Hertz, J., A. Krogh, and R. G. Palmer. 1991. *Introduction to the Theory of Neural Computation,* Redwood City, California, Addison-Wesley,1991

Hopcroft, J. E. & Ullman, J. D. 1979. *Introduction to Automata theory, Languages and Computation,* Addison-Wesley, Reading, MA.

Land M. 1998. "The Visual Control of Steering" *Vision and Action*, Cambridge University Press, pp 163 - 180.

Lindholm, E., M.J. Kilgard,.and H. Moreton. 2001. "A user-programmable vertex engine". *Proc. SIGGRAPH 2001. America.*

Mackworth, N. H. and Morandi, A. J. 1967. "The gaze selects informative details within pictures." *Perception and Psychophysics*, 2, pp 547 – 552.

McCarthy, J. 1987. "Mathematical Logic in Artificial Intelligence." *Daedalus,* vol. 117, No.1*,* American Academy of Arts and Sciences.

Mehdi, Q., Suliman, H., Asloglou, E. Gough, N.E. & Allen, M.J. (2000) Artificial neural networks in future computer games, *Proc. 1st SCS Int. Conf. GAME-ON 2000*, Imperial College, London, November, 29-33.

Nilsson, N. J. 2001. *Artificial Intelligence: A New Synthesis*, Morgan & Kaufmann.

Norton, D. and L Stark,., 1971. "Scanpaths in eye movements during pattern perception." *Science*, **171,** pp 308-311.

Nvidia 2001 "The Infinite Effects GPU Available on PC and Mac Platforms." http://www.nvidia.com/view.asp?- PAGE=geforce3, (last accessed 1st Novermber 2001).

O'Sullivan, C., M. Janott, M. Watson, J. Dingliana, 2000. "Level of Detail Control for Real-time Computer Graphics and Virtual Reality: Applications of Eye-tracking." *1st Irish Workshop on Eye-Tracking: Book of Abstracts*. Trinity College Dublin.. pp 11-14.

Paull, G. C. and D. J. Glencross, 1997. "Expert perception and decision making in baseball." *International Journal of Sport Psychology*, 28, 35-56.

Pearl, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison- Wesley, Reading, Massachusetts.

Poole, D., MackWork, A. and Goebel, R. 1998. *Computational Intelligence*: *A Logical Approach*, Oxford University Press.

Reynolds, C.W. 1987. “Flocks, herds, and schools: a distributed behavioural model.” *Computer Graphics*, 21 (4), 25-34.

Rollings, A. and D. Morris, 1999. *Game Architecture and Design*, Coriolis Group Books.

Russell S. & Norvig P. 1995. *Artificial Intelligence A Modern Approach*, Prentice Hall Inc.

Saltzmann, M. 1999. *Game Design*, McMillan Digital Publishing, Indianapolis, USA.

Smith, R. 2000 “ModelBenders LLC, The double helix: simulation and gaming, Keynote Lecture, *GAME-ON 2000 1st Int. Conf. On Intelligent Games and Simulation*, London, pp

Stark L. W., C. M. Privitera, H. Yang, M. Azzariti, Y. F. Ho, T. Blackmon, C. Dimitri, 2001. “Representation of human vision in the brain: How does human perception recognize images”, *Jour. Of Electronic Imaging*, 10(1) 123 – 151.

Taylor, P. 2001. “Tweening 3-ways, or Using Vertex Shaders”, http://www.msdn.microsoft.com/library/enus/dndrive/html/directx04162001, (last accessed 10 October 2001).

Tu X. and D. Terzopoulos 1994 “Artificial fishes: physics, locomotion, perception, behaviour”, *Proc. SIGGRAPH’94* (Orlando, FL USA, July24-29, 1994), *pp.*43-50.

Valve (2001), www.valvesoftware.com/square.htm, last accessed 8th November 2001.

Van Lent, M. & Laird, J. 1999. “Developing an artificial intelligence engine, Computer Games Developer Conf. May.

Watson, M. 1996. *AI Agents in Virtual Reality Worlds*: *Programming Intelligent VR in C++*, John Wiley & Sons.

Watt A. and F. Policarpo 2001 “*3D games real-time rendering and software technology*”, Addison Wesley.

Woodcock, S. 2000. “Game AI: The State of the industry, part two”, *Gamasutra*, http://www.gamasutra.com/features/20001108/laird_01.htm (last accessed March 2001).

Wright, I. (2000) Sony Computer Entertainment Europe, personal communication.

Wright, I. and J. Marshall, 2000. “Egocentric AI processing for computer entertainment: A real-time process manager for games.” *Proc. 1st SCS Int. Conf. GAME-ON 2000*, Imperial College, London, November,, pp 34-41.

Yarbus, A. L. 1967. *Eye Movements and Vision*, New York: Plenum Press.

Zadeh, L. 1965. “Fuzzy Sets”, *Information and Control* 8:338-353.

MODELLING INTELLIGENT CHARACTERS

LOGIC DEVELOPMENT FOR REASONING AND COGNITIVE NPCS

H. Suliman Q. H. Mehdi N. E. Gough

Multimedia and Intelligent Systems Technology (MIST) Research Group
School of Computing and Information Technology
University of Wolverhampton, UK, WV1 1EQ
in6543@wlv.ac.uk

KEYWORDS: NPC, Logic, Reasoning, Cognitive Agent, Knowledge Base, Ontology, Memory, Attention.

ABSTRACT

This paper begins by reviewing logic systems and then discusses some of the important issues relating to reasoning Non-player Characters (NPCs) in conjunction with believable and cognitive behaviour for computer games. A KB system that relies heavily on substitution rules for the specification of a logical agent is introduced and is used as a first step in the research of knowledge representations for game AI. Additional features for incorporating common sense reasoning and features of human cognition such as attention memory are also presented. Examples and discussion are included to demonstrate some of the applications of the system and future directions of the research.

INTRODUCTION

Non-player characters (NPCs) or game agents can be made to show more cognitive behaviour by being able to reason about their environment and other NPCs. This adds to their believability and enhances game play and the immersion of the game. Creating scenes or situations for a game that bring out the intelligence in the NPC is not only the responsibility of the AI expert but also the writer's and the game's 'level' designers. The AI experts in a software team should provide the apparatus (i.e. behavioural rules, AI engine, Logic engine, etc) by which the NPC is able to make new inferences as well as the ontology or vocabulary of relations that describe the environment and the situation. The AI expert formalises the domain and integrates the degree of intelligence recommended for the game scenario. In that respect the NPC must be able to store and process facts about its world, i.e. it must possess a ***Knowledge Base System*** (**KBS**) (Ringland & Duce 1988, Russell & Norvig 1995), which has a stored set of representations of facts '***sentences***' about the world. The vocabulary used for creating these sentences, and hence describing the agent and its environment is called an ***Ontology*** (Gruber 1993, Russell & Norvig 1995). A conceptualisation of the virtual world is a simplification and an abstraction of this virtual world by identifying some comprehensible concepts. An ontology, according to Gruber (1993) is '*an explicit specification of a conceptualisation*'. When the domain knowledge is specified in a declarative formalism, the set of objects that can be represented is referred to as the ***universe of discourse***. This set of objects, along with the relations between them, are represented in the vocabulary with which a KB system represents its knowledge. Determining what follows from the KB is called ***inference*** and it is any process or mode of reasoning used to produce a conclusion from a set of premises. Hence inference is used to produce a ***proof*** to verify formally the validity of some assertion, in the form of a statement. An inference system is ***complete*** if it can find a proof for any sentence that is entailed by the knowledge base. A proof is a sequence of sound steps (successive steps that are implied by preceding ones) that are used to verify formally the validity of some assertion, in the form of a statement. Conventional and common reasoning systems employ ***First Order Logic*** **(FOL)** (Russell & Norvig 1995) also known as Classical Logic which has an ontology of a set of constant objects together with a set of ***relations*** that are used to relate these objects. Sentences formulated in first order logic are boolean statements, i.e. are assertively TRUE or FALSE and so make use of boolean connectives as well as additional connectives, the ***universal quantifier*** and ***existential quantifier***.

The KB system presented in this work is a hybrid system, similar in capabilities to ***Semantic Networks*** and ***Frame*** type systems (Minsky 1975, Ringland & Duce 1988) in the representation of knowledge yet more similar computationally to more conventional FOL systems in their use of sentence parsing and substitutions. Overall this paper discusses some of the important issues relating to reasoning agents and in conjunction with cognitive behaviour. Additional features for more 'common sense' and 'cognitive' behaviour are also included. These include ***attention*** (Norman 1976, Ashcraft 1998) and a simplified form of ***human memory*** (Baddley 1999, Ashcraft 1998).

LOGIC REASONING SYSTEMS

Logic programming languages (Ringland & Duce 1988, Russell & Norvig 1995) such as LIFE, ISABELLE, OTTER and the very popular PROLOG, are high level programming languages used for theorem proving and problem solving, and are often used for reasoning in mathematical or scientific problems. The majority of these languages use ***resolution*** (Hein 1996, Russell & Norvig 1995) for theorem proving in a FOL representation and use the ***backward-chaining*** algorithm for searching a proof. Resolution can exploit a proof by contradiction strategy known as ***refutation***

which is very commonly used. Resolution is a powerful inference procedure that mechanises theorem proving by repeated application of a single rule. To use resolution, sentences in the KB must be expressed in ***disjunctive*** or ***conjunctive clausal form*** (a conjunction or disjunction series of ***literals*** respectively). When using backward-chaining, proving a query begins a search for substitutions that satisfy the query, i.e. work backwards starting with a query sentence. Other systems such as CLIPS and the well known SOAR (Rosenbloom 1993) rely on production statements (if-then or implications $\Rightarrow$) for representing facts and mostly used ***forward-chaining*** (Russell & Norvig 1995) for searching solutions. With forward-chaining the starting point for proof are the sentences in the KB. The implication statements are used to trigger actions on all aspects of the KB, such as control, updating and deletion. SOAR has been developed as a general cognitive architecture capable of representing and using different forms of knowledge, i.e. procedural, declarative or episodic presentations of knowledge. ***Frame systems***, ***semantic networks*** and ***subsumption networks*** (Ringland & Duce 1987) draw strongly on ***inheritance*** as a system for representing mostly declarative knowledge. The idea of a frame has been used to represent declarative knowledge and has been encapsulated in a series of frame oriented knowledge representation languages. Examples include KL-ONE (Roberts & Goldstein 1997), KRYPTON (Brachman *et al* 1985) and THEO (Mitchell *et al* 1989).

EXTENDING CLASSICAL LOGICS

The agent's knowledge is often comprised of ***material implications*** expressions of the form $P \Rightarrow Q_i$ (which reads if P is true then Q_i must also be true) for all i. Such sentences known as ***Horn sentences*** (Hein 1996, Russell & Norvig 1995) are powerful representations of knowledge. The implication $P \Rightarrow Q_i$ for some i, states that whenever P is true so must Q_i be true, if P is false the state of Q_i being true or false does not contradict the statement. Hence $P \Rightarrow Q_i$ is identical to saying that Q_i cannot be false when P is true, i.e. $\neg(P \wedge \neg Q_i)$. However the power of the operator $\Rightarrow$ is not expressive enough, especially when P stands to represent a complex concept, one that is understood through experience and not defined. For example, consider the expression *loves*(*John*, *Jessy*)$\Rightarrow$ *Kindto*(*John, Jessy*), saying that if *John* loves *Jessy* then he is also kind to her. In addition to statements *loves*(*John*, *Jessy*)$\Rightarrow$*cares(John, Jessy*), *Loves*(*John*, *Jessy*)$\Rightarrow$*Likes*(*John, Jessy*), etc, in total these amount to an expression of the form $P \Rightarrow Q_1 \wedge Q_2 \wedge \ldots \wedge Q_N$. Unfortunately the antecedent P of '$\Rightarrow$' cannot be true while *any* of the consequents Q_i is false. For example, if *John* is not kind to *Jessy*, i.e. $\neg$*Kindto*(*John*, *Jessy*) then he does not love her, which is not necessarily true. What we actually wish to say is that if most of the Q_i are true then P is true. For example, we can write,

$$P = (\Sigma^{N}{}_{,i=1} Q_i > n)\ 1 \le n \le N$$

At extreme values, n=1 is equivalent to $P = \vee_i Q_i$ a disjunction series, and when $n = N$ is equivalent to $P = \wedge_i Q_i$ a conjunction series. We have more flexibility being able to choose intermediate values of n. This also brings us closer to ***artificial neural networks*** (Mehdi *et al* 2000, Bishop 1995) as this expression resembles of the firing condition for a ***McCulloch-Pitts*** neuron model. This directly exposes the limitation of the classic logics or FOLs that use ***boolean*** and ***quantifier*** operators and do not extend their functionality. ***Fuzzy logic*** for example, extends the capability of logic because its propositional variables have a truth value from the closed interval [0,1], where as propositional variables in classical logics have truth values limited to 1 or 0 (True or False). For the P case, $P = (\Sigma^{N}{}_{,i=1} Q_i)/N$, $1 \le n \le N$, dividing by N to normalise P into a range [0,1], P can be interpreted as the average of the Q_i, so this also proposes a probabilistic interpretation.

In a different but related direction, ***modal logic*** (Hintikka 1962, Kripke 1963, Chellas 1980) can offer an alternative for extending classical logic's apparatus. Modal logic is a non-monotonic, first order, boolean logic that employs formal arguments that involve the notions of *necessity* '$\boxtimes$' and *possibility* '$\Diamond$' in addition to the apparatus of FOL, i.e. the boolean connectives and existential quantifiers. Modal logic was invented by Jaako Hintikka in 1962 and was originally based on the concept of strict implication to allow stricter use of $\Rightarrow$ when it comes to adding expression. The modal strict implication is written as $\boxtimes\ P \Rightarrow Q \equiv \boxtimes\neg(P \wedge \neg Q)$ which reads it is necessary that P implies Q. Saul Kripke (1963) presented modal logic added with ***possible-worlds*** semantics. A world is conceived as 'possible' for the agent using a modal logic if this world is consistent with its present knowledge. In other words these worlds summarise the possible directions of monotonic change to the KB. Consider atomic sentences P, Q, R, S, … , let *Worlds*(P) be the set of worlds for which P is true and is a subset of the set of all possible worlds W and similarly let *Worlds*(Q), *Worlds*(R), … etc represent the possible worlds for the respective atomic sentences. Note that the agent's knowledge involves relating these atomic sentences in probably complex ways, i.e. a KB, roughly speaking, is a collection of logical constraints. Hence, intuitively giving truth values to some will impose restrictions on other atomic sentences whose truth values are unknown, and so makes it possible to cause tighter restrictions on the possible truth value of some sentences than others. Hence by using a simple example it is easy to see that if $World(P) \cap World(Q) = \varnothing$ (the empty set) and $World(P) \cup World(Q) = W$ then P and Q cannot be both true as there is no world for which they can be both true, and this is regardless of truth values of R, S, T, … etc leading us to infer that $P = \neg Q$. Hence the expression $\boxtimes\neg(P \wedge \neg Q)$ would translate to saying that P and $\neg$Q cannot be both true in any one world. Hence *John* loves *Jessy* does not necessarily imply he is kind to her. That is $\Diamond$(*loves*(*John*, *Jessy*)$\wedge\neg$*Kindto*(*John, Jessy*)). However despite the seductive use of the

operators of ◊ and ☒, this does not seem to offer a great simplification or more expressive power than relating the premise P and consequent Q with the threshold technique $P = (\Sigma^{N}_{,i=1}Q_i > n)$ mentioned earlier. The modal logic system known as S5 is the most widely used system among the S*n* modal systems precisely because it is the simplest due to two strong axioms or schemas ◊A→ ☒◊A, ☒A→A (Chellas 1980) which do not hold in every system of modal logic.

The above points are also related to the handling of default reasoning problems in semantic network, subsumption networks and Frame type systems through the use of ***inheritance with exceptions***. In these systems a relation such as *loves* is an instance of *cares* can be set by default, unless other information states otherwise. ***Monotinicity*** is the property of a KB system by which new statements can be added without invalidating previously valid sentences. So in a monotonic logic *cares* is implied by *loves* or it is not. Logics with this property are referred to as monotonic logics. This contrasts with ***non monotonic logics*** which allow the KB to treat a proposition as being true until evidence is provided to infer otherwise. Most conventional reasoning systems such as FOL form monotonic KB systems.

KBSs FOR GAME AI PROGRAMMING

Creating a KB System for a Game's AI

Creating a KB in an object oriented language such as C++, which is widely used for writing computer games, sounds interesting and promises speed, but compilers only allow programs to be structurally fixed after compile time. This is not suitable as a KB system shares some similarities to creating a programming language, albeit a restricted and possibly very high level one. A more common form of KB is one comprised of a string of words and an inference process that involves the manipulation and substitution of words for others. This process known as ***unification*** (Robinson 1992, Russell & Norvig 1995) makes the proving process efficient but seemingly mechanical. Though this is the style of many KB systems, unfortunately it does not sit well with the way game engines are commonly designed - as class objects each with their own member variables and methods (Horrowitz et al 1995, Dewhurst & Stark 1989). These methods and variables are fixed as machine code after compilation. Otherwise they are ***interpreted*** which requires the production of a table, called a ***Jump Table*** to detect the scripted instruction. For example, consider a `gate` object, an instance of the `Door` class with method `open()` for which the source code portion `gate.open()` has the effect of opening the gate in the games virtual environment. This instruction presented as a string 'open door' for example, will have to be interpreted, i.e. parsed for the words 'open' and 'gate' and linked to the appropriate function or method specified in the jump table. Hence addresses to instructions corresponding to many combinations of object 'dot' member variable or method have to be included in the table, thus creating a large interpretive overhead. Fortunately, most sentences would be ***descriptive*** in nature and do not contain instructions, so it is feasible to use an interpreted or partly interpreted language at the end of the day. This is also favourable for games that feature a extensible AI because the game engine source itself is secret or too complex for the players to successfully modify.

Extensible AIs

It is quite common for computer games today to come with a scripting language that allows game fans to expand the game such as scenery or NPC AI. For example, *UnrealScript* for the first person shooter game *UNREAL* (Epic MegaGames 2001) is one of many AI scripting languages that enables the player to modify aspects of NPC AI. *UnrealScript* is similar to C++ and Java in syntax and is interpreted quite similarly to the Java virtual machine. The *UnrealScript* files are coded to function similarly to dynamic link library (DLL) files in that they refer to other unreal components when read by Unreal. Another example, is *COG* language created for the 3D action-adventure game *Jedi Knight: Dark forces 2* on the PC (Huebner 1997). *COG* also has a syntax very similar to the C language but with less keywords and constructs such as aspects of the language dealing with function declarations and switch statements because they were significantly more complex to parse and execute than the rest of the language. The *COG* library provides about a hundred different functions to the author, ranging from environment manipulation commands to information queries. These functions are used to control the game environment while using the language syntax to provide control for branching and looping.

A very useful resource is the catalogue of free compilers website (Catalog of free Compilers and Interpreters 2001) that can be linked with a developer's application. In particular the sections concerning logic programming and natural languages are useful in this paper's context. The majority of these languages are provided free being the products of government or university research projects, or usually because of not showing strong commercial application.

A SUBSTITUTION-PARSER KB

A prototype system is being developed to enable the specification of a logical agent that relies on substitutions and string pattern matching for altering and inferring of knowledge in a KBS. A KB sentence is expressed as a string of words '$\mathbf{w}_1\ \mathbf{w}_2\ \mathbf{w}_3\ \ldots$ ' which can be variables or conjoined variables, all joined with a single space to indicate the ending or beginning of another word. The KB itself contains a single long series of sentences stored as a string '$\boldsymbol{K}_1$,$\boldsymbol{K}_2$`,$\boldsymbol{K}_3$``, ...' of strings $\boldsymbol{K}_n$ separated by a comma ',' to indicate the end or beginning of one sentence. $\boldsymbol{K}_n$ corresponds to the *n*th KB sentences. Particular words are reserved for special roles, some are concerned with representing knowledge

and others are syntax related. Most of them invoke the KBS to perform some particular task when encountered while parsing and can have many aliases. For example, 'are', 'is-a', 'implies', 'means', 'the' can be used in place of 'a' which is used to express instances of classes as a binary relation. Below are some of these reserved words alongside their aliases and function.

Reserved words	Function
'is','equals', …	*equivalence/implication*
'a', 'is-a','implies','means', …	*inheritance /existence/ implication*
'all', 'every', …	*entire class reference /universal*
'and', '&', …	*Conjunction*
'or', 'either', …	*Disjunction*
'not', 'a false', …	*Negation*
','	*Sentence separator*
'.', '-', ' ',	*Word separators*
'#'	*Sentence reference*

These words are also stated in order of *precedence* because some might vary in the way the parser interprets them as we shall see later. A sentence such as 'man a human' expresses that man is an instance-of or inherits the features of the class human. Then a sentence 'Jessica loves a man' will naturally allow the substitution of 'human' for 'man' to give 'Jessica loves a human'. Hence a sentence '**x** a **y**' can be read as **x** can-be-replaced-by **y**. However the statement 'Jessica loves man' will then be construed as Jessica loves all men, the substitution for human would give 'Jessica loves human' saying that Jessica loves all humans, which is incorrect! hence the substitution must be conditional on adjacent words. If *K* is a sentence in words **x**,**y**,**z**, … , the following substitution rules or schemas would allow only for the correct changes involving the 'a' character,

Changes	Substitution Rule		New Sentence
C1.	*K*('**x**'), '**x** a **y**', **x≠y**	→	*K*('a-**y**')
C2.	*K*('a **x**'), '**x** a **y**', **x≠y**	→	*K*('a-**y**')
C3.	*K*('**y**'), '**x** a **y**', **x≠y**	→	*K*('**x**')
C4.	*K*('**x**')	→	*K*('a-**x**')
C5.	*K*('a a')	→	*K*('a')

where the symbol '→' means is substituted for. Note C5 is necessary if we only use C1 and abandon C2. C1 is the inference step ***modus ponens*** in substitution form. The 'is' word is used to indicate equality and can be rephrased in terms of 'a', for example, 'man is male' is equivalent to 'man a male' and 'male a man' occurring separately. Hence,

C6.	*K*('**x** is **y**')	→	*K*('**x** a **y**')
C7.	*K*('**x** is **y**')	→	*K*('**y** a **x**')

These substitutions are used extensively to generate new sentences or cause the modification of existing ones as part of steps in a proof process or just generating new sentences. The role of the '-' word separator appearing in C1, C2 and C4 forms a word of series of words joined by it, as explained in more detail in the following sections.

Phrase Substitution

Sentences or statements can be referred to collectively or as a ***phrase*** or ***Frame***, for example

'mike likes woman implies mike'

can literally mean mike likes all women. We may wish to give a phrase meaning, so we can add,

'mike-likes-woman implies mike-a-heterosexual'

The collective meaning is given preference or precedence over the literal meaning in making proofs. If 'mike-a-heterosexual' is not contained in the KB, hence the phrase does not exist and so the literal meaning is used by default. If a sentence *K*('**x**-**y**') where '**x**-**y**' occurs only there and in no other sentence then '**x**-**y**' does not exist as a phrase or sub phrase, so it can be replaced by '**x y**'. This generalises to sentences with more words. The parser interprets two words as a single word if the word separators used is not the single space ' '. Overall this system thus far is similar to what are known as ***slot-filler*** or ***Frame*** type of systems (Minsky 1975, Ringland & Duce 1989) in structure because sentences such as '**x-y-z-** …' conjoins many concepts into a single one.

Representational Power

The system is intended to simplify the substitution process by using only ***binary relations*** (such as loves in 'x loves y' relating the pair x and y) and overlaying statements which has similarities in the way humans use natural language. Overlaying statements is done without bringing ambiguity to meaning, thus economising the usage and hence search when employing inference. For example, '*Jack owns a dog named Rover'* is written simply as,

'Jack owns Rover a dog' $\equiv \exists x\ owns(\text{Jack}, x) \wedge Dog(x)$

The statement expressed in FOL representation is contrasted on the right. The string 'Jack owns Rover a dog' contains both 'Jack owns Rover' and 'Rover a dog' which refer to two independent facts. Ambiguous or incorrect statements can be expressed if one is not careful, for example 'all who love Jessica are tall' would be incorrectly expressed as

'tall loves Jessica' $\equiv \forall x\ tall(x) \wedge loves(x, \text{Jessica})$

because 'tall' can be substituted with all instances of tall according to rule R1, giving rise to the meaning that all that is tall loves Jessica, which is not correct. The correct sentence is

'loves-Jessica is-a tall' $\equiv \forall x\ loves(x, \text{Jessica}) \Rightarrow tall(x)$

Notice that 'love' and 'Jessica' are joined by '-' else the parser will extract 'Jessica is-a tall' and erroneously infer that Jessica is tall. The correct expression above allows the occurrence of John in 'John love Jane' to be replaced with 'John is-a tall', hence inferring that John is a tall person. Further complicating, to say *all who love Jane are tall men* can be done in two ways,

‘love-Jane is-a tall’
‘love-Jane is-a man’

or by creating a phrase tall-man,

‘tall-man is-a tall’
‘tall-man is-a man’

The additional word separator dot ‘·’ is used for distributive operations. Consider ‘not-P-and-Q’, this sentence defaults to ‘not P or Q’, i.e. ¬P∨Q when reduced to its literal meaning by removing the dashes ‘-’. However ‘not·P-and-Q’ is understood as ¬(P∨Q) because ‘·’ is given precedence over ‘-’ in removal to produce a new sentence,

C8. K(‘·’, ‘-’) → K(‘·’,‘ ’) → K(‘ ’,‘ ’)

The characters ‘·’ and ‘-’ used for forming phrases or frames from individual words are preferred over brackets *because the length of a string does not have to change with substitutions*. The characters ‘.’ and ‘-’ replace the space between words and hence strings do not have to be pushed to make more space, copied or form linked lists to other strings, which is an incredibly more efficient syntax.

Complex Specification of Patterns

Wildcards ‘∗’ can be used within sentences in specifying more complex patterns or schemas. The rules C*n* mentioned above are enforced *independently* of the sentences in the KB. However as they are simple they can be removed and relegated into sentences with in the KB, but the KB would then be more than a knowledge representation system. ‘∗’ can be used in expressions to create pattern rules for example,

‘not-∗ a ∗-a-false’ ≡ ∀P ¬P⇒ *false*(P)

which puts a request that upon encountering any word preceded with the word ‘not’ it is permissible to make the substitution with ‘a-false’ on the right. This following rule has the power of removing redundancy in using ‘a’ twice.

‘∗-a-∗ a ∗’

To distinguish between several free variables in an expression, the recurrences of ‘∗’ would be used to demonstrate this, for example

∗-below-∗∗ is ∗∗-above-∗ ≡ ∀x,y below(x,y)⇔above(x,y)

Sentences can be also referenced as wholes using ‘#’. A sentence $\boldsymbol{K_n}$ has a corresponding number *n* in the KB as the *nth* sentence, which can be used in referring to it. For example, ‘#2 a False’ refers to the 2nd KB sentence as being false.

Expressing Propositional Statements

Boolean connectives are used in relating proposition type objects. However a proposition *cannot be directly termed* in this system. As it stands no statement in the system can contradict another, precisely because the negation of a statement is not possible by making statements purely about inheritance. Truth and False statements can be phrased using ‘a’ (or ‘is-a’ or one of its aliases) as set objects, e.g. ‘**x** a false’ or ‘**x** a true’ however the fact ‘true’ are ‘false’ disjoint sets cannot be expressed. So the parser did need added capability to handle statements (predominantly about inheritance) as propositions. The ‘not’ word is used to refer to negation.

‘not-∗’ ↔ ‘∗-a-false’

The most important complete inference procedure is resolution.

(P∨Q)∧(¬Q∨S) → (P∨S) P,Q and S propositions

Precisely employed as a reduction schema, wherever in a sentence, a word **x** and its negation lie in the same sentence. The implementation is more complex than can be expressed as one of the schemas C*n*.

A point in relation to existence statements, consider the expression ‘John owns a dog’ then given that rover, bulldog, snoopy, … are all dogs mentioned in the KB then they are possible candidates, as well as those not specified in the KB as common sense dictates.

C9 K(‘a **F**’),**x** a **F**, **y** a **F**, … → 'K(‘**x**’) or K(‘**x**’) or … K(‘**F**’)’

Notice that ‘or **F**’ occurs as a possible substitution for one of the disjuncts as an alternative if none of the existing ones are appropriate. Substituting with examples amounts to exploring statements, which can be useful when the inference process examines a few candidates random choices and does not enumerate all the possibilities for substitution which may be the case in proving theorems.

KB Queries

Resolving queries or proving statements amounts to using substitutions and string matching or linking multiple string matches to build a proof. This was easily implemented using forward and backward chaining algorithms which each can be invoked as separate functions. Searching for statements relevant to a query begin with more recent sentences and works its way backwards. Without invoking any functions, by default an incomplete sentence added to the KB can be interpreted as a query. Hence when an agent uses incomplete information it may store an incomplete sentence thus querying itself.

Data Replication and Object Oriented Programming for Performance Enhancement

Additional arrays or tables can be created for pointing within KB sentences locations for more direct access to statements within sentences. So in addition to string sentences other locations can retain copies of the simplest or ‘atomic’ sentences made in the KB or pointers to positions in the KB sentences. A method more common in relational databases is ***tree-based indexing*** or ***table-based indexing*** (Russell & Norvig 1995) where data is rearranged in tables such that searches are made more efficient. For example, consider the binary predicate ‘loves’ which relates pairs of

names such as 'John' and 'Jessica' in 'loves(John, Jessica)'. A two column table can be used to store all pairs of names related by *Love*. A query on who *John* loves would be helped if the table was sorted according to who loves, so that all those rows that show fields of who *John* loves are adjacent. Similarly the table can be sorted according to who is loved. Instead of replicating the table for all permutations of the arguments of the relations an array of pointers can be stored to show the sorted order of the rows for a given argument.

In computer memory strings are just similar to any other array. As a series of locations of fixed byte sizes they could hold memory addresses as well characters or numbers. In this sense a string could be replaced by an array of ***pointers*** (Horowitz *et al* 1995, Dewhurst & Stark 1989) to objects where more information can be separately stored, for example ***member variables*** of ***addresses*** can point elsewhere in the KB for more direct reference to other related objects. For example, in C++ we can create a ***class*** object called logic object (type **LogicObject**) of which the KB will contain sentences as arrays of pointers to these objects,

```
class LogicObject {
        char *Name;
        LogicObject *SubsetOf;
        LogicObject *SupersetOf;
        LogicObject **Associates;
Public :
        // …        };
```

A KB sentence would be an array of pointers to instances of these objects,

```
LogicObject **aKBSentence;
```

The member variable SubsetOf points to all the objects to which the instance is a member. The SupersetOf member variable is the set of elements to which the instance of the class is a superset. For example, 'man' in 'man a human' would correspond to some instance of **LogicObject** for which its member variable would point to the logic object of 'human', i.e.

```
LogicObject *man;
LogicObject *human;
man->SubsetOf = human;
```

The member variable Associates points to sentences that contain the occurrence of the logic object. It can be updated every time the parser encounters the logic object.

ADDING COGNITIVE PERFORMANCE

Intelligence, Believability and Knowledge

It is well known that human intelligence is both a function of the state of knowledge and the problems to which it can be applied. Intuitively, having more knowledge on a problem reduces the amount of time and effort to solve it. On the other hand, believability of an NPC that aims to mimic a human can be defined broadly as a domain dependent intelligence, i.e. intelligence observed with in a specific virtual world instance. Within this world the NPC is able to perform as well as any human or the character it represents. Hence the believability of an NPC can depend greatly on possessing the correct expertise and knowledge associated with the character it portrays. A considerable effort in creating some of this knowledge would be undertaken by the game designers when creating an ontology for the game world. A well designed ontology eases the creation of a AI scripting language so as to provide an extensible AI with a game. Consider as an example, a thief/Guard situation where an NPC plays the character of a Guard who is responsible for securing some premises. The player plays the role of the thief or intruder whose task is to rob the premises. The first impression leads us to believe that the guard's behaviour can be predominantly scripted because the possible circumstances that can occur in this limited universe are very small and so can be enumerated. For example, a guard can be attributed an internal state which determines his level of suspicion. Create a *fuzzy* variable **Suspicion**$\in[0,1]$, where at the extreme level of suspicion, i.e. **Suspicion**=1, the NPC will be certain of the presence of an intruder in or around the premises and will then go about trying to apprehend the intruder, raise the alarm, call for help, … etc. In the KB system introduced earlier, the fuzzy value can be accessed by creating a specialised function or table that will return the fuzzy value for a string name corresponding to the variable. That is, it is interpreted, alternatively the fuzzy value is stored in an appropriate member variable of a corresponding object instance of **LogicObject** mentioned earlier, with an additional member variable for storing the fuzzy value.

Believability demands the guards pick up on changes in the environment they inhabit because the guards live about the premises as well as guard it. So they move about in patrols most of the time, perhaps with the odd irregularity added to the pattern. Sometimes visits to the latrine, a chat with another guard or just sleep in a rotational basis. This translates to being familiar with each other's habits and the layout of their environment. For example, an open door would be considered suspicious if the guard could not determine who opened it or if it is normally shut most of the time. Hence guards must be able to remember and predict the movements of each other so that they can make inferences, e.g. ' *... I here a noise behind me, but the Guards are inside, it must be an intruder!!*' and generally learn if the inferences produced wrong conclusions, '*… oops!, it was just a rat*'. It is also necessary that guards remember the layout of their environment, for example if some boxes were moved the guards must recognise the change and reason about these changes, such as ' *… these boxes were not like this before, and a cat or rat is too small to move these boxes. It must then be an intruder!!!*'. In summary, the NPC must show expertise; and be able to remember its environment and past occurrences and use this knowledge in conjunction with its expertise to infer new facts.

Managing Memory and NPC Attention

As the NPC roams about its virtual environment, a cognitive behaviour implies that the NPC must be inquiring regularly on the states of some of these objects. These objects can be said to have attracted the NPC's 'attention' and must be messaged to the NPC, i.e. the NPC's attention mechanism or management system will be event driven so that it does not have to continuously monitor all objects in its environment to detect these changes. For example, consider the possibility that a box placed near a guards patrol route was moved in the guard/thief example mentioned above. The box reports this change to the agent when the conditions are correct or ***qualified*** (Funge 1999, Russell & Norvig 1995). The NPC then consults its memory of the boxes and determines if the changes are indeed significant. If the changes are significant then the NPC's suspicion is increased. In addition, it is not necessary for the agent to receive all details associated with a single object that is not relevant to its situation, hence a *context* may have to be defined in relation to an object's details. In resemblance to humans the agent would also inquire on an object from time to time and not just wait for a message. If changes are not detected then the memory of the object does not need to be altered. Hence not only should the NPC's visual process alarms to objects detected in its visual field but it should also suggest objects to look at according its personal interests and preferences. Consider again the guard NPC of the earlier guard/thief example: objects that can be inquired upon or worth inquiring on form a set, called *Objs*, for example,

$$Objs = \{ Door, Boxes, Alarm, Cat, Rat, Guard_1, Guard_2, \ldots \}$$

These objects are precisely those which can suffer a state change. Doors can be opened or closed, boxes can be moved, guards can roam about and the alarm button can be triggered or switched off. The states of above objects can only be determined when they lie in the NPC's field of view. The exception is the alarm button, whose state can be deduced by the fact it is heard or not. So the NPC's attention mechanism would cycle its attention through different objects over many frames. Generally, if $\boldsymbol{RM}(X)$ is defined as the rate of monitoring an object X, and X can grab the agent's attention, then its rate of monitoring should increase. Hence, RM can be interpreted as a probability and the NPC's attention mechanism creates a sequence of references to objects such that the periodicity of each reference reflects the $\boldsymbol{RM}$ of the object. $\boldsymbol{RM}$ must therefore obey the normalisation expression,

$$\sum_{X \text{ in NPC Visual Field or Audible Range}} \boldsymbol{RM}(X) = 1$$

n object X should message a change to the agent when the condition of access is fulfilled or qualified, such as being in the NPC's visual or audible range. The object itself may redirect the agent's attention to some other object. For example, an open door should direct the agent's attention to *who* opened the door as part of the ability to make inferences. A missing box should redirect the agent's attention to *who* moved the box, *where* is it now, and *why* was it moved. The *where, who* and *why* are important in giving momentum to the NPC's thought processing.

NPC and Human Memory

As the NPC gathers information about objects in its surrounding environment it retains copies of the states of these objects. Typically, most of the objects the agent will encounter in its environment can be modelled as ***Finite State Machines*** **(FSMs)** (Gough *et al* 2000, Hein 1996). Doors, levers, elevators, guns, etc are examples of FSMs in that they have distinct internal states that characterise their behaviour. For example, a gun can be in a firing or idle state, a door can be shut or open, etc. Copying static data is unnecessary as the NPC's memory can just 'point' to it. However state changing or dynamic objects have to be copied into its memory. Based on human memory research (Baddley 1999, Ashcraft 1998), a cognitive agent should be provided with believable behaviour that includes forgetfulness and an efficient management of stored knowledge. In human memory, redundant information is constantly being discarded and the memory is contracted without seriously affecting overall cognitive ability. This can be simply implemented when data degrades with passing time but is reinforced with usage and recall. In a reasoning NPC the data stored takes the form of sentences in a some logic. Simple exponential ***decay methods*** (Brown 1958, Nelson 1985, Ashcraft 1998) can be used to apply this on atomic sentences of a KB. Memory is also ***hierarchical*** in its representation (Suliman *et al* 2000). Memory 'items' are coded feature representations of concepts which reflect relationships that bind these objects. Less prominent features tend to decay than more prominent ones. In an inheritance hierarchy, concepts further up the hierarchy would be considered more prominent or general features. For example if

$$\text{'bachelor is-a male'} \equiv \forall x \; bachelor(x) \Rightarrow male(x)$$

it should be easier to forget that *John* is a bachelor than he is a male. In terms of the KB system introduced earlier a query sentence can be added to the KB and by default, a incomplete sentence added to the KB can be interpreted as a query. The KBS searches normally begin with most recent sentences first and go backwards in time. When the answer to a query or the KBS's attempt is available it can be added at the bottom as a new sentence which refers to the query and answer as a sentence. For example, if the query was logged as the 10th sentence in the KB and the system's attempt was logged as the 11th sentence, then the 12th KB sentence would be, '#10 is-a #11' instructing the KBS to replace a new query with #11 if it looks exactly like the sentence #10.

CONCLUSION

This paper has highlighted some of the most important issues in improving NPC intelligence and how these broadly tie with expertise, cognitive behaviour and reasoning. The KB system presented has the expressive power of FOL in declarative representations and takes advantage largely of the substitution oriented representation to make inferences. Further work is needed to improve the performance of the KB system and especially with regard to guiding the inference process. Inference was done purely by pattern matching and using the forward or backward chaining algorithms. The language is extremely flexible overall and provides a very good basis for creating a KB system. There are many more interesting aspects of these topics that can be indulged upon and already a lot of what was developed provides a strong mechanism for expressing and manipulating knowledge. It was also not the intention of the authors to build a theorem proving system because they are intended for more formal applications but rather to investigate those aspects more concerned with game design.

REFERENCES

Ashcraft M. H. (1998). *Fundamentals of Cognition*, Addison–Wesley Educational Publishers Inc.

Baddley A. D. (1999). *Essentials of Human Memory*, Psychology Press Ltd.

Bishop M. C. (1995), *Neural Networks for Pattern Recognition,* Oxford University Press.

Brachman, R. J., Gilbert, V. P. and Levesque, H. J. (1985). 'An essential hybrid reasoning system:Knowledge and symbol level accounts of KRYPTON'. In *Proceedings IJCAI-85.*

Brown, J. A. (1958). 'Some tests of the decay theory of immediate memory'. *Quarterly Journal of Experimental Psychology*, 62, 375-385.

Catalog of free Compilers (2001), www.idiom,com/free-compilers/ , last accessed 1st November 2001.

Chellas, B. F. (1980), *Modal Logic: An Introduction*, Cambridge University Press.

Dewhurst, S.C. and Stark, K. T. (1989), *Programming in C++*, Prentice Hall Inc.

Epic MegaGames Inc (2001), unreal.epicgames.com, last accessed 1st November 2001.

Funge, D. (1999), *AI for Games and Animation*: *A Cognitive Modelling Approach,* A K Peters Ltd.

Gough, N.E., Suliman, H**.** & Mehdi, Q. (2000), 'Fuzzy state machine modelling of agents and their environments for games', *Proc. 1st SCS Int. Conf. GAME-ON 2000*, Imperial College, London, pp 61-68.

Gruber, T. R. (1993), 'A translation approach to portable ontologies'. *Knowledge Acquisition*, 5(2):199-220, 1993.

Hein, J. L. (1996), *Theory of Computation*; A*n Introduction,* Jones and Bartkett Publishers International.

Heubner, R. (1997), 'Adding languages to game engines', ***Game Developer Magazine*** September 1997 issue.

Hintikka, J. (1962), *Knowledge and Belief*. Cornell University Press, Ithaca, New York.

Hopcroft, J. E. and J. D. Ullman (1979), *Introduction to Automata theory, Languages and Computation,* Addison-Wesley, Reading, MA.

Horowitz, E., Sahni, S. and Mehta, D. (1995), *Fundamentals of Data Structures in C++,* Computer Science Press, New York.

Kripke, S. (1963), 'Semantical considerations on modal logic'. *Acta Philosophica Fennica,* 16:83-94.

Mehdi, Q., Suliman, H., Asloglou, E. Gough, N.E. & Allen, M.J. (2000) Artificial neural networks in future computer games, *Proc. 1st SCS Int. Conf. GAME-ON 2000*, Imperial College, London, November, 29-33.

Minsky, M. (1975). 'A framework for representing knowledge'. In *The Psychology of Computer Vision,* ed. P. Winston. McGraw-Hill, New York.

Mitchell, T. M., Allen, J., Chalasini, J., Cheng, O., Etzioni, M. Ringuette and Schlimmer, J. (1989). 'Theo: A framework for self improving systems'. In *Architectures for Intelligence*, ed. K. VanLehn. Hillsdale, NJ., Erlbaum.

Nelson, T. O. (1985). 'Ebbinghaus's contribution to the measurement of retention: Savings during relearning'. *Journal of Experimental Psychology*: Learning, Memory, and Cognition, 11, 472-479.

Norman, D. A. (1976). *Memory and attention*: *An introduction to human information processing*' 2nd ed. New York, Wiley.

Ringland G. A. and Duce D. (1988), *Approaches to Knowledge Representation - An Introduction*, Taunton, Somerset: Research Studies Press.

Roberts, R. B. and Goldstein, I. P. (1977). *The FRL Manual*. Tech. Rep. MIT Artificial Intelligence Laboratory.

Robinson, J. A. (1992), 'Logic and logic programming', *Communications of the ACM,* March 1992, vol.35, no.3.

Rosenbloom, P. S., Laird, J. E., Newell, A. & McCarl, R. (1991). 'A preliminary analysis of the Soar architecture as a basis for general intelligence'. *Artificial Intelligence,47*,289-325.

Russell S. and Norvig P. (1995), *Artificial Intelligence a Modern Approach*, Prentice Hall Inc.

Suliman, H., Mehdi, Q. and Gough, N. (2001), 'Spatial cognitive maps in agent navigation and path planning', *Proceeding of the ISCA 10th Int. Conf.*, Arlington, Virginia USA, 27-31.

Technomagi Design Links (2001), www.technomagi.com/links/design.html, last accessed 1st November 2001.

Hussam Suliman is a PhD researcher at the university of Wolverhampton researching on AI with applications to computer games. He first graduated in BSc Physics and MSc physics in Control Systems from Imperial College London in 1997 before joining Wolfson College to study Graduate Mathematics (PartIII) for 1 yr at Cambridge University in 1998. Afterwards he briefly worked as a software engineer trainee with Saudi Airlines before joining Wolverhampton in October 1999. Please visit his personal web page (forever under construction) at www.wlv.ac.uk/~in6543 for more information.

BEHAVIOURAL INTERACTION OF CHARACTERS FOR VIRTUAL STORYTELLING

Fred Charles, Steven J. Mead and Marc Cavazza
School of Computing and Mathematics
University of Teesside
Middlesbrough, TS1 3BA,
United Kingdom
E-mail: {f.charles, steven.j.mead, m.o.cavazza}@tees.ac.uk

KEYWORDS
Interactive Storytelling, Synthetic Characters, AI-based Animation, Computer Games.

ABSTRACT

In this paper we describe a fully implemented prototype for interactive storytelling using the Unreal™ engine. We describe the important mechanisms involved in the variability of plot instantiations, within a scenario of sitcom genre. We also provide an evaluation of the concepts of how the dynamic interactions between agents and/or the user influence the generation of story, with first results of examples

INTRODUCTION

In this paper, we present results from a first version of a fully-implemented storytelling prototype, which illustrate the generation of variants of a generic storyline. These variants result from the interaction of autonomous characters with one another, with environment resources or from user intervention.
The development of artificial actors and AI-based animation naturally leads to envision future interactive storytelling systems. A typical interactive storytelling system would be based on autonomous virtual actors that generate the plot through their real-time interaction. Besides, the user should be allowed to interfere with the ongoing action, thereby altering the plot as it unfolds.
Many interactive storytelling models have been proposed: user-centred plot resolution (Sgouros et al. 1996), character-based approaches (Young 2000) (Mateas 2000), anytime interaction (Nakatsu and Tosa 1999) and the need for narrative formalisms (Szilas 1999). Previous work has identified relevant dimensions and key problems for the implementation of interactive storytelling, among which: the status of the user, the level of explicit narrative representation and narrative control, the modes of user intervention, the relations between characters and plot, etc.
Some of these problems derive from the inherent tension between interaction and narrative (Young 2000) (Mateas 2000). Interactive systems demand user involvement but often at the expense of a real storyline; on the other hand, a strong narrative dimension is traditionally conceived with a user as spectator rather than being actively involved. Our solution to this problem consists in limiting the user involvement in the story, though interaction should be allowed at anytime. This is achieved by driving the plot with autonomous characters' behaviours, and allowing the user to interfere with the characters' plans. The user can interact either by physical intervention on the set or by passing information to the actors (e.g., through speech input).
In the next sections, we will introduce the important concepts of character-centred storytelling as well as a brief description of our implementation. Results of variants in story generation are illustrated with an example.

CHARACTER-BASED STORYTELLING

The storyline for our prototype is based on a simple sitcom-like scenario, where the main character "Ross" wants to invite the female character "Rachel" out on a date. This scenario tests a narrative element (i.e. "Will he succeed?") as well as situational elements (the actual episodes of this overall plan that can have dramatic significance, e.g., how he will manage to talk to her in private if she is busy, etc.). Our system is driven by characters' behaviours. These actually "compile" narrative content into characters' behaviours, by defining a superset of all

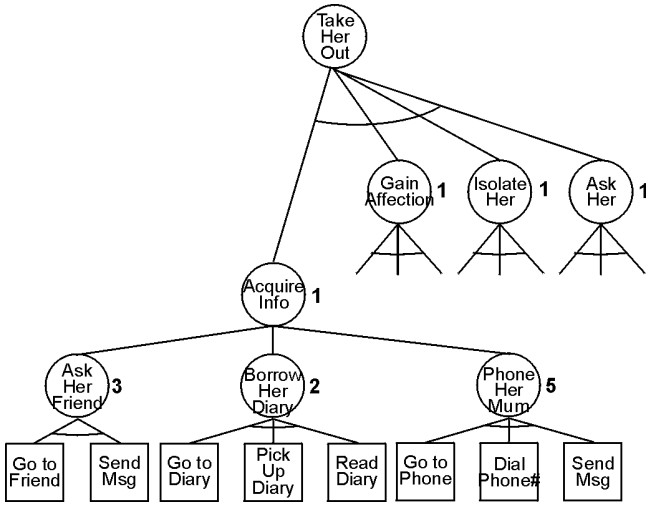


Figure 1: HTN Representation for Character Behaviour

possible behaviours, represented by a plan for each character. Dynamic choice of an actual course of action within this superset is the basis for plot instantiation (Young 2000). In that sense, this addresses the causality/choice duality

described by Raskin (Raskin 1998) in storytelling, though this choice takes place within the limits of the formalism used to represent possible behaviours, which is a plan-based formalism (Young 2000). This can be illustrated by considering the overall plan for the character Ross (see Figure 1).

In order to invite Rachel, he must for instance acquire information on her preferences, find a way to talk to her, and finally formulate his request (or having someone acting on his behalf, etc.). These goals can be broken into many different sub-goals, corresponding to various courses of action, each having a specific narrative significance.

The initial storyline should actually determine not only the main character's plan, but those of other characters as well. The problem of dependencies between characters' roles has actually been described within modern narratology, though not to a formal level. Narrative functions can be refined into bipolar relations between a couple of actors, emphasising the asymmetry in their roles (Barthes 1966). We have adopted this framework to define the respective behaviours of our two leading characters. We started with the overall narrative properties imposed by the story genre (sitcoms). In terms of behaviour definition, this amounts to defining an "active" plan for the Ross character (oriented towards inviting Rachel) and a generic pattern of behaviour for Rachel (her day-to-day activities).

AI-BASED CHARACTERS' BEHAVIOUR

Individual agent behaviours are produced by solving the set of sub-plans described in the preceding section, which are represented by Hierarchical Task Networks (HTN), such as the one of Figure 1. Using formal properties of these plans, it is possible to generate solution plans by searching directly the AND/OR graph of the HTN with an algorithm such as AO* (Tsuneto 1997) (Pearl 1984). In our system (Cavazza 2000), this is done with a "real-time" variant of AO*, which interleaves planning and execution and supports re-planning that is required when a character's plan is altered through interaction with another virtual actor or the user. The terminal actions (e.g. reaching a location, using an object, interacting with other actors) forming the plan are actually played in the graphic environment through their corresponding animations. The dramatisation of these actions constitutes the story as seen by the user.

The "virtual sitcom" prototype described in this paper has been developed using the Unreal™ game engine. The Unreal™ environment provides most of the user interaction features required to support user intervention in the plot, such as navigating about and interacting with objects within the virtual set and its use has been previously reported in prototyping interactive storytelling (Young 2000). The system has been fully implemented as a set of template C++ classes, which can be used as native functions from within UnrealScript™, Unreal™'s scripting language.

USER INTERVENTION

The user watches the story as a spectator. At this stage he can follow the story from any character's perspective or navigate on the virtual set while the action is in progress. From his understanding of the current action, he can choose whether to interfere or not with the characters' goals. Characters' actions are dramatised through the timing of appropriate animations. Because the actors are playing a role rather than improvising, their actions are always narratively meaningful. Hence, if a character moves towards a given object, it is likely to bear significance for the story and can be the target for user intervention. For instance, if the user sees Ross moving towards Rachel's diary, he can choose to steal or hide that diary (see Figures 2 and 3).

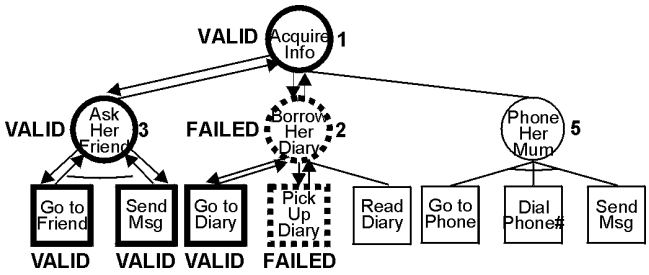


Figure 2: Re-planning on Action Failure

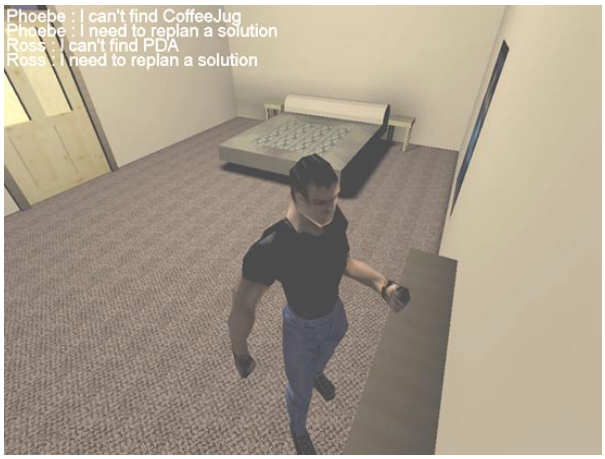


Figure 3: Dramatisation of Action Failure

The user can intervene by either acting on physical objects on-stage that bear narrative relevance (and are often obvious, such as keys, letters, gifts, weapons, etc.). These objects being resources for actions, they will force the character into re-planning or action repair, which, being dramatised as well, will create a new course for the plot. The other mode of interaction consists in influencing actors using speech recognition. This form of influence will become the main one in further developments of the system and will include:

1. providing information needed by the actors to complete their plans (e.g. Rachel's preferred gifts) (see Figure 4)
2. giving doctrine advice that influences the personality of an actor (i.e. recommending a friendly behaviour towards certain characters)
3. trying to alter the mood of a character
4. getting actors to perform certain actions that have narrative consequences, such as moving to a certain location that increases the probability of meeting other characters

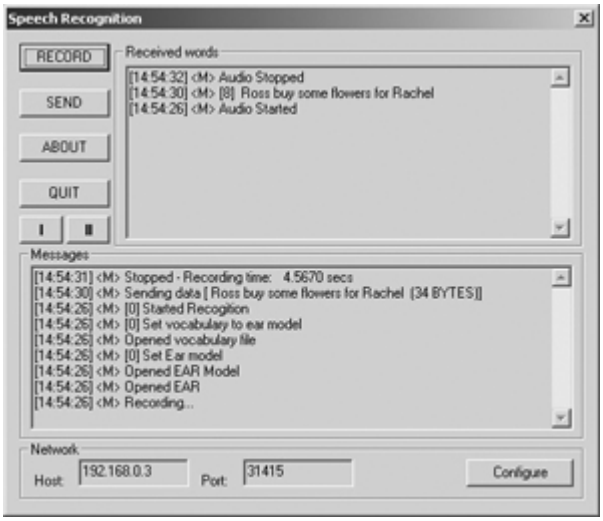


Figure 4: Speech-driven User Intervention:
"Ross, buy flowers for Rachel"

RESULTS

While the conditions for character interaction lie in the on-stage spatio-temporal instantiation of the storyline, additional mechanisms are required to recognise these interactions and propagate their consequences.

Figure 6 (see next page) illustrates an entire story instantiation. Ross wants to use Rachel's PDA to retrieve relevant information regarding her preferences. He goes to Rachel's bedroom (a), unseen by Phoebe, who is preparing some coffee (b). As the user discovered Ross' plan, he decides to remove the object from the virtual environment (c) to alter the on-going storyline. Ross reaches the location of the PDA (d), unaware of user intervention (e). Ross makes a new decision to talk to Phoebe (f), as she may provide him with the relevant information. Ross interrupts Phoebe regardless of what she is doing (g). As Ross was rather unkind to Phoebe, she decides to lie to him concerning Rachel's preferences, telling him to offer Rachel a box of chocolates (h). In a different story instantiation, if Ross were more careful when asking Phoebe, she would have responded more positively to his request, by telling him to buy roses instead. After succeeding in gathering important information, Ross goes to purchase his gift for Rachel from the shop (i, j). After buying the box of chocolates (k), he goes back to the flat (l, m) to offer them to Rachel. As she is alone, he goes (n) and asks her out, which she inevitably refuses (o).

This example illustrates the interaction of the two main characters' plans. These plans are designed from narrative principles. It appears that exploring actors' behaviour in storytelling is more feasible within narrative genres that display the simplest storylines, as such developing "virtual sitcoms" seems a relevant first step in the pursuit of interactive storytelling. As its own name suggests, sitcom standing for "situation comedy", a significant fraction of the story interest arises from the situations into which the actors find themselves. For instance, the fact that Rachel could misunderstand the situation where Ross was talking to Phoebe, then triggering the emotional reaction of jealousy (see Figure 5). Her state of mind being modified (i.e., she gets upset), Rachel will then leave the room. The succession of "small" interesting situations is a mechanism for cause-and-effect relationships (Raskin 1998), providing the basis for dramatic story generation.

Figure 5: Situation Comedy (Rachel Is Jealous)

Though plans are designed from global narrative principles, considering the current story genre, they are run independently. The bipolarity between the characters' plans was defined to emphasise the asymmetry in their roles (Barthes 1966). The overall narrative properties imposed by the story genre defined interaction between the main character's "Ross" and its supporting role's "Rachel" behaviours. The generic pattern of Rachel's day-to-day activities may interfere with Ross' "active" plan, as illustrated when Ross want to read Rachel's diary while she is already using it. This interactivity between characters' behaviours must be emphasised visually when it demonstrates narrative relevance.

As part of the story believability, mechanisms in action recognition will help to make the characters' emotional status visible to the user, so he can understand their interactions. The variations in characters' emotions and moods must emerge from situations relevant to the story genre without changing their overall personality profile. For instance, Rachel's mood towards other characters can vary according to the meaning of their actions for Rachel (e.g., jealousy). The next generation of real-time animation engine (i.e., Unreal2™) will help representing facial expressions, or

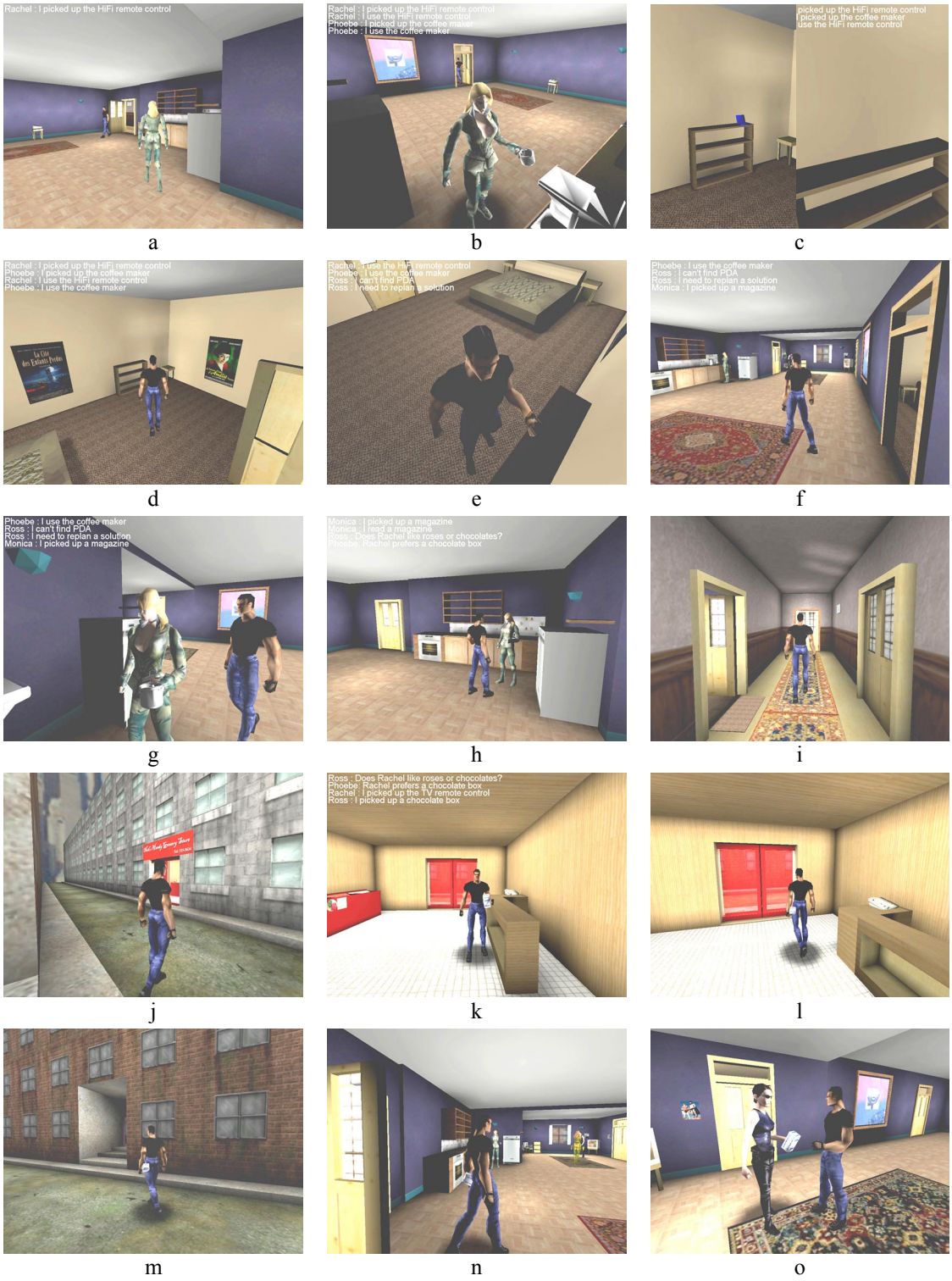


Figure 6: Example of a Story Instantiation

detailed non-verbal behaviour (e.g., body postures) to improve the dramatisation of events through physical characterisation.

Above the planning and interleaving of actions, explicit situated mechanisms for reactive behaviours (Geib 1994) are needed in order to deal with specific situations (e.g., Ross suddenly meets Rachel on his way). This implies high-level action recognition of interactions between characters' behaviours. If a narratively meaningful (considering the story genre) situation arises, the mechanism would act on the character's current plan by ordering a re-planning of its action.

CONCLUSION

We have shown that, although actor's behaviours are deterministic, the interaction between actors could considerably contribute towards story variability. This degree of unpredictability conditions the generation of dramatic

situations. The character-centred approach has the advantage of being modular and extendable to many actors.
Further work is to be dedicated to developing more complex storylines within differing genres, scaling up using multiple plans for each actor to increase characters' interactions and narrative function recognition.

REFERENCES

Barthes, R. 1966. “Introduction à l’Analyse Structurale des Récits” (in French), Communications, 8, pp. 1-27.

Cavazza, M., F.Charles and S.J. Mead. 2001. “Characters in Search of an Author: AI-based Virtual Storytelling”. First International Conference on Virtual Storytelling, Avignon, France.

Geib C., “The Intentional Planning System: ItPlans”. Proceedings of the 2nd Artificial Intelligence Planning Conference, AIPS-94, 1994, pp. 55-64.

Mateas, M.. 2000. “A Neo-Aristotelian Theory of Interactive Drama”, AA,AI Spring Symposium in Artificial Intelligence and Interactive Entertainment, AAAI Press.

Nakatsu R. and N. Tosa. 1999. “Interactive Movies”. In: B. Furht (Ed), Handbook of Internet and Multimedia – Systems and applications, CRC Press and IEEE Press.

Pearl, J., 1984. “*Heuristics: Intelligent Search Strategies for Computer Problem Solving*”. Reading (Massachusetts), Addison-Wesley.

Raskin, R. 1998. “Five Parameters for Story Design in the Short Fiction Film”, P.O.V., n. 5.

Sgouros, N.M., G. Papakonstantinou and P. Tsanakas. 1996. “A Framework for Plot Control in Interactive Story Systems”. Proceedings AAAI’96, AAAI Press, Portland.

Szilas, N. 1999. “Interactive Drama on Computer: Beyond Linear Narrative”, AAAI Fall Symposium on Narrative Intelligence, Technical Report FS-99-01, AAAI Press.

Tsuneto, R., D. Nau and J. Hendler. 1997. “Plan-Refinement Strategies and Search-Space Size”, Proceedings of the European Conference on Planning, pp. 414-426.

Young, R.M. 2000. “Creating Interactive Narrative Structures: The Potential for AI Approaches”. AAAI Spring Symposium in Artificial Intelligence and Interactive Entertainment, AAAI Press, 2000.

ARTIFICIAL PLAYER FOR QUAKE III ARENA

J.M.P. van Waveren and drs. dr. L.J.M. Rothkrantz
Faculty of Information Technology and Systems
Delft University of Technology
Mekelweg 4, 2628CD Delft, The Netherlands

KEYWORDS
Games, AI, Artificial Player, Automated route finding.

ABSTRACT

In this paper the Quake III Arena bot is presented. The bot is an intelligent artificial player emulating a human player in the game environment of the first person shoot-em up (FPS) game Quake III Arena. With the Quake III Arena bot everyone can enjoy the game and practice without the need for other people. The bot is an artificial player that "lives" inside the computer, side by side with the game program. The bot only appears like a human player inside the game-world. However the in game behavior of the bot should be hard to distinguish from the behavior of human players. To show human-like behavior a wide range of techniques and common sense solutions are used for the bot AI. The Quake III Arena bot is the first commercially developed artificial player that uses fully automated path and route finding through arbitrary complex 3D polygonal worlds, without the need for the bot to acquire knowledge about routing and navigation during game-play. No human intervention is required to provide the bot with all the information needed to navigate through, and understand new game environments. The bot uses a volume (area) based representation of the 3D game environment, which serves as a back-bone for the bot's cognitive world model. Together with a high performance path finding solution this cognitive model makes the bot quite resource efficient.

INTRODUCTION

Quake III Arena belongs to the genre of the first person shoot-em up games (FPS). A player views from a first person perspective and moves around in a real-time 3D virtual world. The most important tasks are staying alive and eliminating opponents within this virtual world. These opponents are other people, equal in strength and abilities, connected to the same game through a network or the Internet. The players have a wide range of weapons, items and power-ups available to aid in the battles. The game has a set of different virtual environments called levels or maps, that contain rooms and hallways. The battles in the game take place in these maps much like gladiators fight in an arena. Players can score points by taking out other players. When killed, a player respawns at one of the designated locations on the map and can continue to fight. Quake III Arena also has several team oriented game-play modes. In normal team-play there are two teams with players that fight each other. The team with the highest accumulated score, of all players on that team, wins. There is also a Capture The Flag (CTF) team based game mode. Again there are two teams with players. Each team has a base structure in the game world or map. A flag is positioned in such a base. A team scores points by capturing the flag of the opposing team and bringing it back to their own flag in their own base.

The Quake III Arena bot is supposed to act like a human player in the virtual world of the game. As such the bot should be hard to distinguish from a human player in the game. In order to act like a human player the bot does not only need to understand the rules of the game and how the game works. The bot also needs basic abilities like navigating through the game environments, picking up items and handling weapons. The bot has to be able to play the team based game types and has to operate in teams. In order to operate in a team, with both human players and other bots, the bot needs to communicate with other players. The bot lives inside the computer next to the game program and only appears as a human player in the game. The same (game) rules that apply to human players in the game, also apply to the bot. The bot does however not use the same input and output devices as human players. Instead of the output devices human players use, like the computer's monitor and sound card, the bot receives information about the virtual world directly from the game program as a set of variables. The bot also does not use the commonly used input devices like the keyboard and mouse. The bot sends a sequence of actions or intentions directly to the game program. These actions however, are very similar to the actions a human player can input using the computer's input devices. The bot uses knowledge that has been provided in advance and knowledge acquired during game-play to construct such sequences of actions.

To be able to understand and reason about the environment the bot needs a cognitive model of the world it lives in. This cognitive model plays an important role

in how the bot perceives and understands the virtual world. In particular, the bot cannot notice aspects of the world that are not represented within its cognitive model. In it's simplest form this cognitive model can be a set of variables that represent the current state of the world. This state of the world includes the position of the bot, position of enemies and items, the weapons the bot has gathered etc. To be able to navigate through the virtual world and find certain locations in the virtual world the bot needs a representation of the level or map the bot is situated in.
For autonomous behavior the bot also needs to maintain an explicit representation of how the virtual world changes. This knowledge is referred to as domain knowledge, and is required to reason about the effects of different sequences of actions. The reasoning is necessary for the bot in order to select actions, that are useful within the game and allow the bot to achieve certain goals. Human players often think intuitively about the effects of actions and make a lot of implicit common sense assumptions. A bot does of course not have this common sense by default. A certain level of domain knowledge and common sense will have to be built into the bot.
The bot does not only need basic knowledge in advance to be able to operate in the virtual world, the bot also has to acquire knowledge while playing the game. The current state of the world is one of the simplest kinds of knowledge the bot can acquire. This knowledge acquisition is referred to as sensing. Aside from knowledge about the current state of the world the bot can also acquire knowledge about the dynamics of the world, the domain knowledge. The bot could acquire knowledge about how the world behaves, and how other entities like other players within the world behave. This kind of knowledge acquisition is called learning and is far more complex than sensing. Often learning requires the recognition of patterns in a large amount of observations and evaluations of the effects of certain sequences of actions. Usually most, if not all domain knowledge a bot uses is provided in advance.

THEORY

Creating a bot for an FPS game is an extensive task which requires expertise from many different areas within the field of artificial intelligence (AI), and also areas not directly associated with AI. Some of the commonly used methods and techniques are described here.

Path finding

One of the requirements for autonomous behavior in FPS games is the ability to navigate through the game world in a life-like manner. Determining how to navigate through a map is an interesting problem. Many different approaches to solving this problem have been presented. Very simple AI just lets a bot walk forward

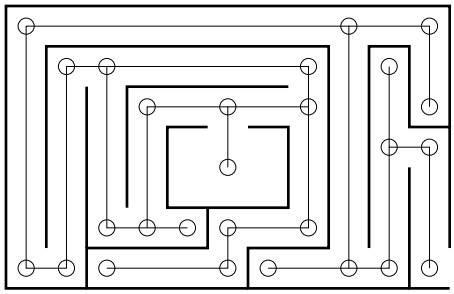


Figure 1: Maze with waypoints represented by dots

until something is hit. At that point the bot turns and continues walking forward. There are also more complex path finding algorithms that use heuristics to find routes through the environment. A special representation of the environment or map is often used for these more complex algorithms. One of the most commonly used representations is a waypoint system (Figure 1). Such a waypoint system is a collection of points or locations (waypoints) with directional links between them. The waypoints represent the places where the bot can go and the links between them represent the paths the bot can follow in order to easily travel from one waypoint to another. Usually the links represent straight line directional paths. Creating an efficient waypoint system for a specific environment is an interesting and often complex task. A bot could create the waypoint system during game-play and drop waypoints as it wanders through the environment. The bot will have a hard time reaching most places in the environment until it has wandered through most of the game world. Using this method the bot often never finds out how to go to certain hard to reach places. The waypoint system could also be created in advance before the bot enters the game. Placing waypoints throughout the environment and linking them is often a time consuming task to be completed by the level designer. Some algorithms have been developed to aid in creating a waypoint system in advance, but usually human intervention is required to optimize the system.
When a good waypoint system is available to the bot, a whole range of different paths can be calculated. Usually only the shortest paths towards specific goals are used by a bot. However different kinds of paths can be useful as well. For instance paths that lead towards a goal, while avoiding certain areas of the world at the same time. Several algorithms are available to calculate the shortest path between a source location and a destination. The most commonly used algorithms are Floyd's, Dijkstra's and A* (A - star) [3]. These algorithms were designed in the context of graphs and graph theory. Since a waypoint system is very similar to a directed graph these algorithms can also be used to calculate paths along one or more waypoints. Traveling along a path, the bot might still encounter small or larger obstacles. A bot

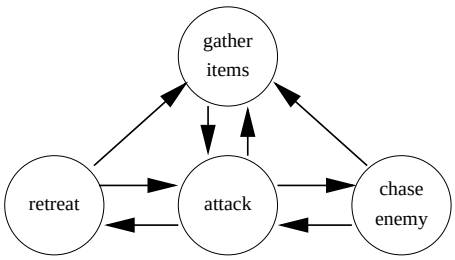


Figure 2: Simple FSM for a bot

often uses sensing and environment sampling to identify the nature of such obstacles. The bot then tries to avoid or navigate around the obstacles. The Quake III Arena bot does not use waypoints to find routes and navigate through the environment. Instead the bot uses a volume (area) based representation of the 3D game environment, which serves as a back-bone for the bot's cognitive world model.

Finite State Machine

A finite state machine (FSM) is a system that has a limited number of states of operation. A real world example could be a light switch which is either on or off. The finite state machine that represents a light switch has an 'on state' and an 'off state'. A light switch has two states and from either state the light switch can change to the other state. Usually there are more than just two states and the state transitions are often limited. The subject being modeled usually cannot directly change from any state to every other state. The state transitions are also bound by certain conditions. The light switch for instance only changes state when a person pushes the switch in a certain direction.

Any system that has a limited number of possible states can be modeled as a finite state machine. Finite state machines are often used to simulate human beings, how they behave and think. Although there are other systems that can more accurately model the way humans think and learn, the simplicity of finite state machines makes them rather popular. The finite state machine only needs to be as complex as the desired complexity of the subject being modeled.

Finite state machines are often used to model the line of thinking for a bot. The different states of the finite state machine can represent different states of mind, or different kinds of behavior. There can be states for different situations and state transitions are often based on certain events in the game environment. A very simple bot could be modeled with a finite state machine using four states as shown in Figure 2. The Quake III Arena bot uses a similar structure as the FSM to model its think process.

Fuzzy logic

Fuzzy logic [1] is a superset of conventional (Boolean) logic. This logic was extended to handle the concept of partial truth, also using values between "completely true" and "completely false". Lotfi Zadeh of UC/Berkeley introduced fuzzy logic in the 1960's as a means to model the uncertainty of natural language.

Just as there is a strong relationship between Boolean logic and the concept of a subset, there is a similar strong relationship between fuzzy logic and fuzzy subset theory. Let's assume there's a set S and to all its elements there's one element of the set 0,1 attached. The subset U of the set S is defined as all the elements of S that have a '1' attached. The truth or falsity of the statement "x is in U" can be determined. The statement is true if there's a '1' attached to the element 'x' in S. Otherwise the statement is false.

Similarly for the fuzzy case there's a set S. But now a value from the interval [0, 1] is attached to every element of S. The subset U of the set S isn't strictly defined in this case. However it can be determined how much an element from the set S belongs to the fuzzy subset U. A value of zero attached to an element from S represents complete non-membership of U. A value of one represents complete membership. The values between zero and one represent intermediate degrees of membership. The degree to which the statement "x is in U" is true can also be determined. The degree of truth of the statement is given by the attached value to the element x in the set S.

Often a value from the interval [0, 1] is attached to an element of the set S using a function, the membership function. Such a function is one-dimensional because it's based solely on one criterion. In practice membership functions are based on two or even more criteria. Such a function gives a value from the interval [0, 1] to a combination of criteria and is often referred to as a "fuzzy relation". The criteria don't have to be elements from the same set they can just as well be elements from different sets. The criteria also do not have to be elements from sets. Variables of some kind can also be used as criteria.

Fuzzy logic can be used by bots to express how much they want to have, or do certain things. For instance a bot might think of how much it wants to retrieve a certain item as a fuzzy value. The more the bot wants the item the higher the fuzzy value. Fuzzy relations can be used to express the relation between the current state of the bot and how much the bot wants something. For instance based on which weapon the bot is holding, and how much ammunition the bot already has for the weapon, the bot can attach a fuzzy value to how much it wants to retrieve more ammunition for the weapon. The Quake III Arena bot uses this kind of fuzzy logic to express how much it wants to have or do certain things.

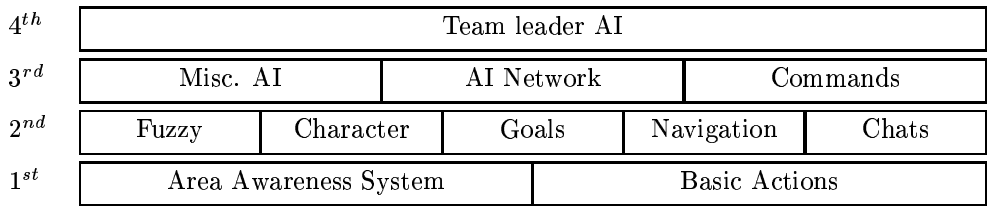


Figure 3: Layered architecture

Expert system

An expert systems [2] is a system that stores human expertise, gained from training and experience. Such a system has three important aspects. The knowledge of facts, the knowledge of relations between the facts, and a heuristic or efficient method for storage and acquisition of this information. The construction of an expert system is called knowledge engineering. A knowledge engineer extracts the knowledge (procedures, strategies, information filters, common events, etc.) out of human experts and transforms this knowledge into an expert system. The key issue in expert systems is the way knowledge is stored in, and extracted from the knowledge base. Logic systems provide powerful tools to represent and infer knowledge in an expert system. Production rules or rule-based systems are therefore often used to implement an expert system. Production rules originate from the IF (...) THEN structure, which is known from traditional procedural languages. The rules consist of a condition side (the antecedent) and an action side (the consequent): IF (condition) THEN (action). The condition is a logical expression of facts from the knowledge base. The action either creates one or more new facts, removes facts or triggers certain events. The difference in rule-based programming as opposed to conventional programming lies in the fact that the statements in conventional programming languages are executed in a predefined order. Rule-based systems use an inference engine, which determines the rules to be fired based on the current facts. In this way new facts can change the course of the program as it proceeds to fire other rules, that are not necessarily stored consecutively to previously fired rules.

An expert system can be used to implement the reasoning of a bot. The expertise from human players is extracted and stored in a knowledge base. Production rules can be used to create new facts or initiate certain actions. For instance the bot could use the following production rule: IF the bot is fighting AND the bot is low on health AND the bot does not have a powerful weapon THEN retreat from the fight. The concepts "low on health" and "does not have a powerful weapon" are not clear facts in this example. Fuzzy logic can be used to give a clear meaning to such concepts. A value of truth can be attached to the statement "low on health" based on the amount of health the bot has. In the same way a value of truth can be attached to the concept "the bot does not have a powerful weapon" based on the weapons and ammunition the bot is carrying. Using a fuzzy relation, it is also possible to combine the concepts "low on health" and "does not have a powerful weapon" into the new concept "the bot is not fit enough to fight". The example production rule could then be replaced by: IF the bot is in a fight AND the bot is not fit enough to fight THEN retreat from the fight. An inference engine of some kind can be used in combination with production rules as in the examples. However, such production rules are also often just listed in several procedures. The Quake III Arena bot uses production rules to explicitly represent certain knowledge and make certain decisions.

ARCHITECTURE

Layered architecture

The Quake III Arena bot is build up in several layers. The awareness of decisions the bot makes while playing the game increases with higher layers. The decisions from higher layers are executed through lower layers. The layered structure of the bot is shown in Figure 3.

The 1^{st} layer is basically the input and output layer for the bot. The Area Awareness System is the system that provides the bot with all the information about the current state of the world. Information about the state of the world is received as a set of variables directly from the game program. For fast and easy access and usage, a lot of formatting and pre-processing is performed on this information. Everything the bot senses goes through the Area Awareness System. The basic actions are the output of the bot. The output is formatted in a way that conforms to the parameters of the game. The output of the bot is the player input for the game, much like a human player uses a keyboard and a mouse.

The 2^{nd} layer provides the intelligence that is often subconscious to skilled human players. This layer contains AI for goal selection. This AI allows the bot to choose the best goal to pursue from a whole range of possible goals. The bot uses fuzzy logic to decide which goals it wants to achieve. This fuzzy logic is also stored in the 2^{nd} layer. There is also AI to navigate through the game environment towards a specific goal. To be able to communicate with other players this layer also has AI

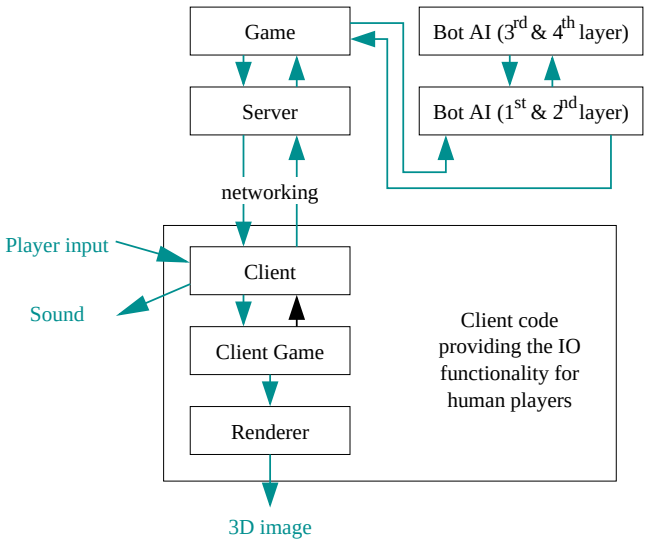


Figure 4: Integration of bot AI with the game engine.

to interpret chat messages from others and to construct chat messages directed at other players. Finally there is a module to store and retrieve properties of different bot characters. Each bot character has a set of variables that represent the characteristics of the bot.

The 3^{rd} layer is a mixture of production rules (if-then-else) and an AI network with special nodes for different situations and states of mind. This network is very similar to a state machine. All the higher-level thinking and reasoning takes place in this layer. This layer also contains a command module which allows the bot to understand orders and commands from other players or a team leader. The miscellaneous AI module contains AI to support behavior used in, and decisions made in the AI network. This includes AI for the fighting behavior of the bot and AI to navigate around obstacles and solve small puzzles in the game environment.

The 4^{th} layer is the "brain" of the team leader or command center. In a team game one of the bots is designated to be the team leader and has this extra "brain" used to command teammates. This allows the team leader to organize the team and accomplish tasks in a team.

The code of the game engine, including the bot AI, is structured as shown in the block diagram (Figure 4).

The "Game" module sets the rules for the game and dictates how the game works. The "Server" module provides the functionality for players to connect to the game. The "Client", "Client Game" and "Renderer" modules together provide the Input/Output (IO) functionality for a human player. The "Client" module records input from the input devices and sends it to the server. This module also forwards information from the server about the game, and what is visible to the player to the "Client Game" module. The "Client Game" module interprets this information and sends the necessary data to the "Renderer" which provides a 3D image to the player. The "Client Game" module also sends back information about sounds to the "Client", which makes the sounds audible to the player. In Figure 4 the bot AI is shown in two parts: the lower two layers and the upper two layers. The "Game" module provides the Area Awareness System with all the information about the state of the game world. This information consists mainly of entity data. The position, type, appearance etc. of entities in the game are communicated to the Area Awareness System. The bot input, a sequence of basic actions or intentions, generated by the bot, is sent directly to the "Game" module.

The whole game typically runs in small time steps or frames, which is referred to as time-based simulation. Each frame the time and the whole game world advance a little bit. The bot's "brain" also operates in frames but not necessarily synchronized with the game. The bot's "brain" always operates at 10 Hz. Every tenth of a second the bot checks upon its status and situation and decides for the best actions to be taken. The bot uses the information available from the Area Awareness System to stay up to date about its status and the environment.

AI network

The central "brain" of the bot is a network with special nodes for different situations and different goals. This Game AI network resides at the 3^{rd} layer of the structure shown in Figure 3. All the other AI sub-systems are used from or within this network. This "brain" of the bot is very much like a finite state machine modeled as a network of nodes with conditional links between the nodes. The bot can only be at one node at any time. Every think frame the bot goes through this network until the node best suitable for the bot's current situation is found. There is always exactly one node best suitable for the current situation, and the bot changes nodes until it finds this specific node. Each node within the network is optimized for a specific range of goals or sub-goals. The network also has nodes which allow the bot to do pressing tasks while going for a long term goal. Each node has a procedure with production rules (if-then- else) for the reasoning and decision making of the bot. Such a procedure also implements the conditional jumps to other nodes to make sure the best node is found for each situation.

Figure 5 shows the network used by the bot. The boxes with a name represent the nodes. The arrows show from which node to which node the bot can jump in order to find the best node for the current situation. The 'respawn' node is used when the bot is dead, and needs to respawn somewhere in the game environment. The 'stand' node is used when the bot is standing still to chat with other players. Furthermore there are three 'seek' nodes which the bot uses when it is not fighting with any opponents. The bot can pursue long term,

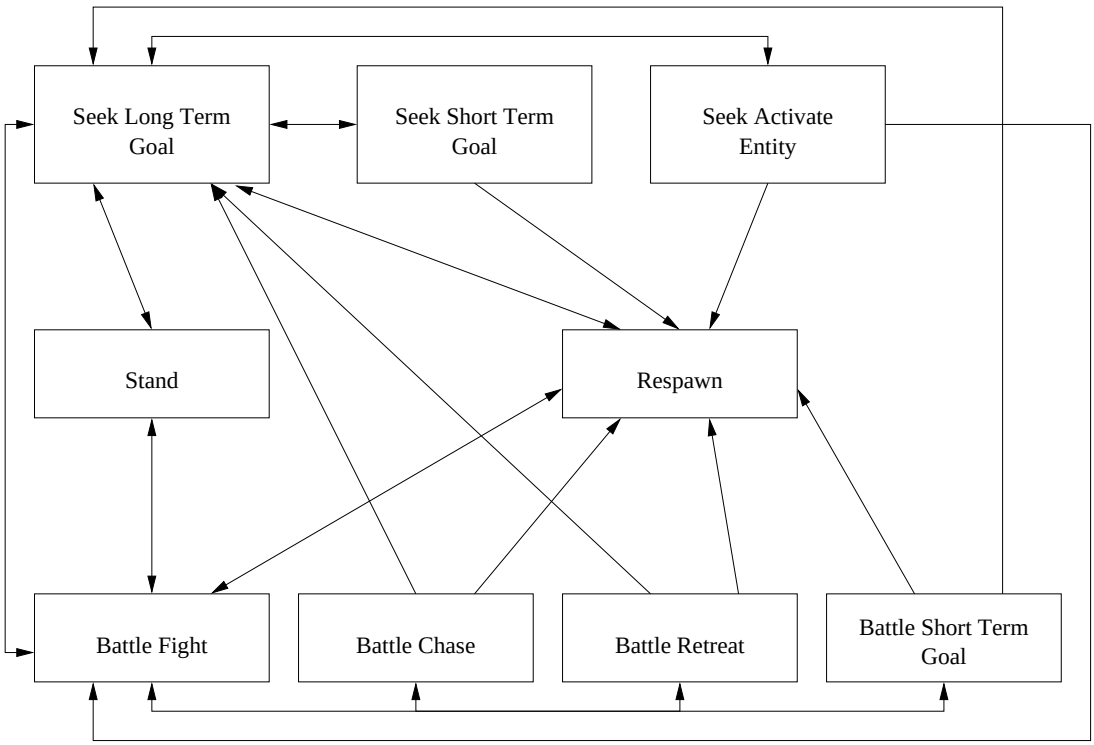


Figure 5: Game AI network.

and short goals in these nodes. There is also a node to activate buttons and triggers to open doors, bars etc. in the game environment. When the bot is fighting with an enemy one of the 'battle' nodes is used. The bot has a node to fight directly with the enemy, chase an enemy that tries to run away, retreat from a fight when the bot does not feel fit enough, and a node to achieve short term goals during a battle, like picking up a nearby item.

Area Awareness System

The Area Awareness System (AAS) is the whole system used to provide the bot with all the information about the current state of the world. This includes information about navigation, routing and also other entities in the game. All the information is formatted and preprocessed for fast and easy access and usage by the bot. The heart of AAS is a special 3D representation of the game world. All information provided to the bot is in some way related to or linked with this 3D representation. However this representation is primarily used for routing and navigation.

A waypoint system is commonly used for routing and navigation purposes in 3D environments. For navigation purposes the links between the waypoints have a specific property: one can easily travel from one waypoint to another if they are linked. In other words the navigation complexity to reach one waypoint from another along a link is minimal, for instance the navigation along a straight line. AAS has a similar property as the waypoint system. AAS uses 3D bounded hulls, called areas, with a specific property: the navigation complexity for traveling from any reachable point in an area, to any other reachable point in the same area, is minimal. In Quake III Arena this means a bot can move between any such two points by just walking or swimming along a straight line. The map or game environment has to be subdivided into areas with this property. Most convex open spaces or hulls have the property required for the areas of AAS. Binary Space Partitioning (BSP) is used to subdivide the map into these convex hulls or areas. H. Fuchs, Z. Kedem, and B. Naylor first introduced Binary Space Partitioning for graphics rendering purposes [4]. With binary space partitioning a tree structure is created, a so-called BSP tree. This BSP tree is a binary tree, which represents the entire space, an entire map in the game. Each node in the tree represents a convex subspace and stores a plane, which splits the space the node represents in two halves. A node also stores references to two other nodes, which represent each half. These two nodes are often called children or child nodes. The planes of the polygons used for the world geometry are used as splitters at the nodes of the tree. However the world and these polygons are expanded first. The visual player model is not used to collide with the world geometry, instead the player resides in an axial bounding box, which is used for collision detection. The player movement is limited by the collisions of the bounding box with the world geometry, and the origin of this box has the same freedom in movement as the box itself.

Therefore it suffices to only know where the origin of this bounding box can be within the game world. For this reason the bounding box is decreased to a point, the box origin, and the polygons of the world geometry are expanded, to make up for only colliding with the origin of the box instead of the bounding box itself.
The areas created with binary space partitioning do not necessarily have minimal navigation complexity between any two reachable points within the same area. In case a bot can walk somewhere in a convex area there might also be one or more gaps in the floor in the same area. The bot could fall into such gaps and not be able to get out. Areas with both a gap in the floor and places where the bot can walk, will have to be subdivided into multiple areas. Whenever such an area is found, it is split with a vertical plane through an edge of the gap. The BSP tree is modified accordingly to store the new subspaces created by splitting the subspace represented by the area.
The ability to easily navigate within areas is not enough to travel through the whole map. So called reachabilities are calculated between areas. These reachabilities are links between areas that show how the bot can easily travel from one area to another. To calculate these reachabilities more easily the BSP tree which represents the game world is transformed into another representation. In this new representation the areas are bounded by faces. These faces are polygons that either represent solid walls or lead to other areas. Reachabilities can easily be found through such faces that lead to adjacent areas. However more complex reachabilities can also be calculated using the face representation of the areas, for instance a jump from one area to another which goes through multiple areas.
Using the reachabilities the bot can now not only easily travel within areas, but also between areas. Routes can also be calculated between areas using the reachabilties. There are several algorithms readily available to calculate routes and travel times in a graph. However the Quake III Arena bot uses a special multi-level routing algorithm to calculate travel times between areas. For this algorithm the areas are grouped into several clusters. The travel times are calculated at two levels: within clusters and between clusters. The multi-level routing algorithm can calculate the same travel times as the commonly used algorithms like Floyd's and Dijkstra's [3], but uses significantly less resources.

Bot characters

A human player can play the game with one or more artificial players. To make the game more enjoyable and more versatile, there is a whole range of different bot characters that play the game in their own style, and provide different challenges for the human player.A character consists of a carefully selected set of characteristics that are applicable in the game. These characteristics are stored in the character module in the 2nd layer of the structure (Figure 3).
The more characteristics that can be changed per character, the more versatile the character can be. It can make the characters less predictable. However some characteristics interact with each other or even contradict each other. Interaction cannot always easily be avoided.For instance there are characteristics representing the tendency to jump and the tendency to crouch respectively (mostly in fights to avoid projectiles). However a bot cannot jump and crouch at the same time. the interaction is avoided by choosing characteristics with clear boundaries and without overlap.
The characteristics are also normalized. The range of the value of a characteristic and its influence on how the bot behaves within the game, should make sense.The characteristics are are also normalized relative to each other. For instance if there is an aim accuracy characteristic for each of two weapons, then the values of these characteristics should have the same effect for both weapons.
When a set of characteristics is selected one should keep in mind that the perception of human players is the most important thing. When human players think about how a bot plays the game is more important than how the bot really plays the game. One as to make sure the used characteristics really do make a difference that can and will be noticed by human players Currently in Quake III Arena, there are 25 characteristics implemented such as name, gender, aggression, alertness, jumper, walker attack skill, aim accuracy, weapon weights,item weights, chats etc.
Most characteristics have values in the range [0,1] (see Figure 6). The characteristics "weapon weights" and "item weights" are references to the locations where fuzzy logic is stored for situation dependent weapon preferences and item goal selection respectively. The characteristic "chats" is a reference to the location where the individual bot chatter is stored. Most of the above characteristics are related to the fighting behavior of the bot. The bot is often seen during fights. As a result the differences in behavior between different bot characters is most noticeable when the bots are fighting.

CONCLUSION

The Quake III Arena bot is a fairly good opponent. The bot is also entertaining and quite suitable for practice and training purposes. The bot is able to navigate through the environment in a life-like manner, and the bot can pick up items and handle weapons just like human players. The bot also shows interesting fighting behavior with tactical moves. The bot can chase opponents that try to escape from a fight, and the bot itself can try to retreat if it is not feeling fit enough to fight. The Quake III Arena bot can show relatively complex behavior with a rather simple model for it's reasoning.

Skill when aiming, a value between 0 and 1 for each weapon

```
> 0.0 & < 0.9 = aim is effected by enemy movement
> 0.4 & <= 0.8 = enemy linear leading
> 0.8 & <= 1.0 = enemy exact movement leading
> 0.6 & <= 1.0 = splash damage by shootng nearby geometry
> 0.5 & <= 1.0 = prediction shots when enemy is not visible
```

Figure 6: Example "aim skill"

The AI network, similar to a finite state machine, has very few nodes or states. However the bot can pursue several different goals in each state, which allows the bot to complete a wide range of tasks and show more complex behavior.
With the Quake III Arena bot shows that all the information a bot needs in order to navigate and find routes through an arbitrary 3D polygonal environment can be (pre) calculated in a relatively short time, without the need for human intervention. In effect a bot can 'learn' its way around the game world in a very short period of time. This is an advantage over the commonly used waypoint systems. These systems are usually harder to create with programming, and often require human intervention in order to optimize them.
Although the QuakeIII Arena bot turned out to be a fairly good artificial player, it could be improved in the following areas: The fighting behavior of the bot could be improved by adding more and better anticipation of enemies. Currently, when the enemy is out of sight, the bot always assumes, the enemy will come back traveling along the shortest path. At this moment the bot uses little or no planning to achieve specific goals within the game. The bot will likely show more intelligent behavior when it has the ability to construct larger plans. The team-play behavior of the bot could be improved as well. the bot could be made more aware of certain team goals and tuned for better cooperation with the team mates. The chat message parsing could also be improved for the bot's communication, currently only templates are used by the bot to "understand" messages.

REFERENCES

[1] C.H. Chen, *The Fuzzy Logic and Neural Network handbook.* ISBN: 0-07-011189-8

[2] L. Boullart, A.Krijgsman and R.A. Vingerhoeds, *Application of artificial intelligence in process control.* (1992) Pergamon Press ISBN: 0080420176

[3] Thomas H. Cormen, *Introduction to Algorithms*, (2000 24^{th} printing), ISBN: 0-262-53091-0

[4] H. Fuchs, Z. Kedem, B. and Naylor, *Visible Surface Generation by A-Priori Tree Structures.* (July 1980), Conf. Proc. of SIGGRAPH '80, 14(3), 124-133.

[5] John David Funge, *AI for Games and Animation.* (1999), A K Peters, Ltd. ISBN: 1568811039

[6] Stuart Russel, Peter Norvig, *Artificial Intelligence, a modern approach* (1995) Prentice-Hall ISBN: 0131038052

[7] Donald Hearn, M. Pauline Baker, *Computer Graphics*, (2^{nd} edition 1996), Prentice-Hall ISBN: 013159690X

MESH SKINNING TECHNIQUE FOR INTELLIGENT ANIMATED CHARACTERS IN COMPUTER GAME

Z. Wen, Q.H.Mehdi, and N.E.Gough
School of Computing and Information Technology
University of Wolverhampton, 35-49 Lichfield Street
Wolverhampton, WV1 1EQ, United Kingdom
E-Mail: IN6716@wlv.ac.uk

KEYWORDS:

Mesh skinning, transformation pipeline, DirectX 8, intelligent animated character, transformation matrix, vertex shader.

ABSTRACT

The mesh skinning technique has several distinct advantages compared to the traditional computer game animation techniques such as key framing and articulated rigid body animation. It can produce realistic and smooth character animation in real time by allowing more than one transformation matrix to affect the vertices that form the "skin" of the character. With advances in both hardware and software, it has now become more practical to implement this technique in PCs in real time. The paper firstly gives a brief review of traditional animation methods in computer games. It then introduces the skinning mesh technique and proposes a method to incorporate this technique into procedures for intelligent character animation. The paper also introduces the implementation of mesh skinning using DirectX 8. The result of this work could be applied to 3D simulation involving realistic character animation.

INTRODUCTION

Computer animated characters play an important role in interactive entertainment products such as computer games. Due to the limited computational power of PCs and real time performance, efficient character animation is one of the main concerns for computer game design and implementation. Various methods such as hierarchical articulated rigid body animation and simple mesh blending animation have been applied to produce real time character animation.

The hierarchical articulated body animation method uses a hierarchical set of interconnected mesh pieces that form the body of a character (Anderson 2001). A simple example of this technique would be a head connected to the torso, the torso to the upper arm, the upper arm to the lower arm, and so forth. During the rendering, the final transformation matrix is calculated in such a way that it makes a recursive function call to all the matrices that sit on the higher hierarchy in forward kinematics or lower hierarchy in reverse kinematics. For instance, in the above example, the animation matrix for the lower arm is affected by the matrices of upper arm. For the 3D hierarchic articulated object, the flexibility is its main benefits. Various transformations that comprise a character's motion can be produced using an animation package, and then any interpolation techniques from simple linear interpolation to complex hermite spline interpolation can be applied to generate in-between frames in order to produce smooth animation (Cebenoyan 2001). Furthermore, inverse kinematics also can be used to produce realistic character movement. Since the vertex information for the model including position, texture co-ordinate and transformation matrix needs to be stored only once during program runtime, the memory usage would be small. However, the memory capacity constraint is no longer a significant problem to the game designer due to a rapid reduction in the cost. Another advantage of this technique is that it is simple to implement and it can be executed relatively quickly in software, which was clearly an important prerequisite for early real time 3D computer games (Freidlin 2001). One of the drawbacks to this method is the result of an unpleasant folding effect when character's limb is rotated too far about a joint. In practice, when the game character makes some large movement, an ugly gap will appear between the separate parts of bodies. Furthermore, it would be very difficult to achieve smooth shading across the boundaries of the separate mesh pieces because of the anomalous perturbation of the vertex normal surrounding a joint. This problem therefore forces game designer to devise tricks to cover the problem. For example, a cloth accessory like belt or long sleeve is applied around the joints that are most likely to cause the problem.

A mesh blending skinning method was introduced to implement smooth character animation. Rather than breaking a character into separate body parts, the mesh blending method produces a character in a single mesh object, generates multiple slightly different copies of the character model and produce real time animation by blending these different models using various interpolation methods. The main advantage of this method is the result of smooth animation with no gaps or self-penetration part in the character model during runtime.

Although mesh blending can be executed very fast during runtime by using a special multi-stream data rendering technique provided by today advanced APIs such as DirectX, the disadvantages of this method still remain a major issue. This is because the animator needs to model all of the possible poses of the character in advance and store them in a huge system database for use in runtime. The problem is that after the animator has modelled the poses, it leave nearly no flexibility to the game designer to generate various animations based on user interaction and changes of game environments. The animation process becomes difficult to parameterise. This problem implies that it is not easy for the movements of character to reflect changes in its run time environment, which is an important aspect of character's believability.

In the light of the preceding discussion, a more suitable method is needed and a way of applying this method to intelligent animated character needs to be investigated.

MESH SKINNING IN INTELLIGENT CHARACTER ANIMATION

Introduction to mesh skinning

Mesh skinning animation was introduced in order to combine both advantages from articulated body animation and single mesh blending animation, namely flexibility and smoothness (Lander 2001a; Lander 2001b; Weber 2001). Generally speaking, the mesh skinning technique achieves this effect by allowing more than one transformation matrix to affect a vertex's final position and lighting normal. The mathematical equations that describe this method are as follows (Domine 2001). For the position of a vertex,

$$V_{final} = \sum_{i}^{n} w_i M_i v \qquad \sum_i w_i = 1 \qquad \textbf{(1)}$$

Where:

v is the vertex position
n is the number of matrices
w_i is the associated weight
M_i is the transformation matrix

For the lighting normal of a vertex,

$$N_{final} = \sum_{i}^{N} w_i M_i^{-1^T} n \qquad \sum_i w_i = 1 \qquad \textbf{(2)}$$

Where:

n is the vertex normal
N is the number of matrices
w_i is the associated weight
$M_i^{-1^T}$ is the inverse transpose of transformation matrix

The normal needs to be transformed by the inverse transpose of the matrix used to transform geometry in order to achieve the correct effect for back-face culling and shading (Moller and Haines 1999).

The mesh skinning technique allows a mesh to be deformed based on an underlying hierarchical transformation matrix set. This animation method mimics the way in which the skin is deformed by a skeleton in reality. The animator has the control of the transformation matrices that comprise the character's skeleton rather than the direct skin vertex and associates each skin vertex with more than one transformation matrix. In this way, the mesh skinning method can enable the application to create unique animation sequences based on user interaction and run-time game environments instead of interpolating from per-set poses. Furthermore, skeleton animation is easily parameterised, which makes it a superior choice for intelligent character animation. This is because once the animation procedure can be parameterised such that the animation procedure's parameters are able to change during program run-time according to the character's internal and external stimuli, the character will be capable of performing various movements based on environmental stimuli and internal states. The following section depicts the animation architecture that reflects the above method.

The animation architecture

The animation architecture is based on that proposed in (Mehdi et al. 2001) and enhances the behavioural and graphics part so as to adapt to the mesh skinning method. The character perceives its environment from several data channels named virtual sensor (Thalmann 1995; Thalmann 1996). The sensorial information is then passed to the perception component to form its own "image" for the world. The reason for this stage is that the intelligent character's beliefs are distinct from the actual states of the world (Burke et al. 2001). For instance, an object may be located at co-ordinates (20, 20, 0), but the character only knows where the object is relative to its head. Therefore, the character cannot be allowed access to the world model. Furthermore, not all perceived information should be passed through this component to reach the internal states of character. This perception component will also act as a "information filter". The actions will then be selected via a Fuzzy Finite State Machine (FzFSM). The most important enhancement is the graphics component and the links between the graphics component and behavioural component. The system architecture is outlined in Figure 1. Compared to the Pat-Nets proposed by the Balder (Granieri et al. 1995), this architecture is more suitable to be implemented in PC platform. The Pat-Nets is an inter-connected network that contains three kinds of nodes, namely control, perceptual and behaviours. As the system is intended for the multi-agent animation, a network processes each behaviour node. In this way, the system needs to be run in a multi-processor system. In reality, normal PCs would not have more than one processor available in their system. Therefore, the generic implementation method proposed by Thalmann (1996) is more suitable. As shown in Figure 1, the sensorial information comes from the external game environments. Different data channels can detect different

kinds of information. The classification of data kinds was described in (Wen et al. 2000). The perception component then receives the sensorial information as the input. It then performs as a "information filter", which extract the useful information for the character's active internal states. For instance, when a hungry shark has detected a moving fish and a floating sea plant ahead, this sensorial information will then be passed to the perception component. The perception component firstly translates these two objects' world positions into two positions relative to the head of the shark so that once the action is determined, the shark would know in which direction to take. The information of "floating plant" will likely be discarded and the information of "moving fish" will enhance the shark's dominant internal state "HUNGER" resulting in behavioural actions such as chasing, rotating tail etc. However, in other situation, suppose that a small fish character has detected the same objects as above. If the small fish is hungry, object "float plant" will trigger the food hunting related internal states and will subsequently affect certain behaviours. Meanwhile, object "moving fish" may increase its internal state "CAUTION". Therefore, the rules of this perceptual component may vary differently depending on the character's own behavioural routine.

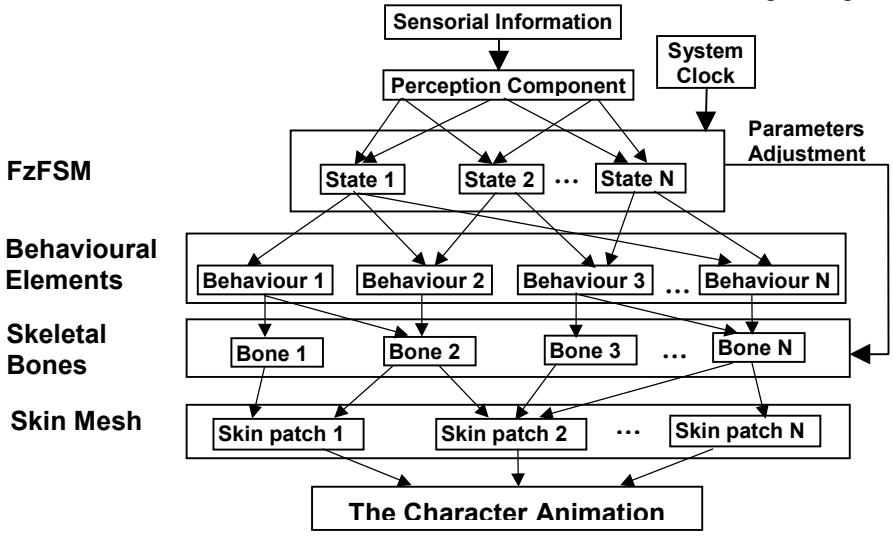


Figure 1 Layer-based architecture of intelligent character animation using mesh skinning

The FzFSM receives input information from the perception component. This information flow will determine which internal states need to draw attention to the newly arriving external stimuli. Some of the internal states such as "HUNGER" may change according to the time even without external stimuli. In the layer of "behavioural elements", each behaviour in this layer is associated with one or more of the character's states in the internal state layer. These links are pre-set. For example, the internal state "HUNGER" will activate several behavioural elements such as "chasing", "accelerating", etc. As in mesh skinning animation, motion is applied to the character's skeletal bones rather than the "skin" seen by the player and the behaviours will then have direct links to the corresponding bones inside the character's skeleton. These links are also pre-set. For example, the behavioural element "accelerating" will activate the transformation matrix for the bone "tail" to be manipulated during this time step's rendering. This implies that the information contained in this bone transformation matrix needs to be updated to reflect any change in the character's internal state. In addition, there is information flow from the internal state layer. As every internal state is a fuzzy set, it can normally vary from 0 to 1. For example, "HUNGER=0" means that the character is not hungry; "HUNGER=1" means that the character will die of hunger, which results in totally different animation sequence. When the number is between 0 and 1, this reflects how hungry the character is, which can be a parameter supplied to the transformation matrix. In this way, the behaviours that the character presents will be more flexible and unpredictable during run-time, which substantially increases the believability of the game character. Thus, this bone layer will perform the most important task in this animation system since it is responsible for the generation of animation sequences based on the input from the characters internal state. Various methods such as real time inverse kinematics can be applied to produce realistic animation. For instance, a method called Cyclic-Co-ordinate Descent (CCD) can be used (Lander 2001c). This algorithm is capable of mimicking the realistic movement for an articulated object in a reasonably fast way.

In the last layer of the animation system is the skin patch, which is able to be associated with more than one transformation bone in the bone layer. This vertex level mechanism can efficiently eliminate the gap or inter-penetration between the separate parts of character model due to the single transformation problem. It is worthwhile to mention that for the sake of speed, it is not necessary for all the skin vertices to get involved in this multi-bone rendering. Only those parts that are likely to produce the gaps should be involved. For instance, for the upper arm

and lower arm, only the vertices that are near the joint are need to be involved into multi-bone rendering. The other vertices should be treated by the normal rendering procedure. Therefore, it may be more efficient to divide the skin vertices into several categories according to the number of bones that will affect the vertex (Cebenoyan 2001).

IMPLEMENTATION ISSUES IN DIRECTX

In the earlier version of Microsoft DirectX, the API introduced a new rendering state into their fix-function rendering called D3DRS_VERTEXBLEND. This state allows geometry to be affected up to four transformation bones. To efficiently perform soft-skinning on the whole character (which clearly requires more than four transformation bones to represent its motion), the application has to make batch calls to the DrawPrimitive() function based on which four bone matrices influenced a given mesh piece. Therefore, four is the maximum number of transformation bones possible each time the DrawPrimitive() function is called. The main benefit for this vertex blending is that it can be executed very fast. However, the disadvantage is the limit to the number of transformation bones for each vertex that form the skin mesh. Furthermore, it does not integrate well with per-pixel lighting that was introduced in DirectX 8 (Cebenoyan 2001). Per-pixel lighting enables generation of customized photo-realistic effects by using the programmable pixel shader in the Direct3D rendering pipeline.

With the birth of DirectX 8, the above limitations have been alleviated by two new provided methods to implement more sophisticated vertex blending techniques in real time, namely indexed vertex blending and matrix palette skinning with vertex shader.

Indexed vertex blending is the easier way but not the most flexible way to implement mesh skinning compared to the vertex shader method. It can be thought as an overall improvement over geometry blending provided by DirectX 7. Instead of each polygon being influenced by a maximum of four bone matrices, each vertex can now be influenced by as many as four bone matrices. This implies that 12 different bone matrices would now influence a polygon. In this way, much more flexible and smooth character animation effect can be achieved. If the vertex is provided with rendering normal in appropriate vertex buffer flag, DirectX will automatically compute the inverse transpose of the matrices that are depicted in equation **2**. DirectX achieves this by encapsulating an 8-bit value as indices to the transformation matrices that will influence the vertex position and lighting normal. The main problem for this indexed vertex blending is the number of bone matrices that can influence one single vertex is limited to four. This number may vary depending on what graphics hardware is installed on the machine. Additionally, there may not be wide hardware support for indexed vertex blending in the future, which may significantly limit the possible high performance implementation of this technique (Freidlin 2001).

By incorporating the programmable vertex shader, matrix palette skinning via vertex shader becomes the most flexible way to implement mesh-skinning technique for character animation in real time (Wloka and Maughan 2001). This implementation method is fully support in hardware and can support a flexible number of bones per vertex depending on different rendering requirements. Therefore, this method has been adopted to develop our example animation. The vertex shader is introduced to replace the transformation and lighting engine inside the DirectX transformation pipeline in order to produce more desired effects by the programmer. It is a small assembly language program with 128 instructions. The instructions have access to four different types of memory locations, namely per-vertex data of an incoming vertex, constant

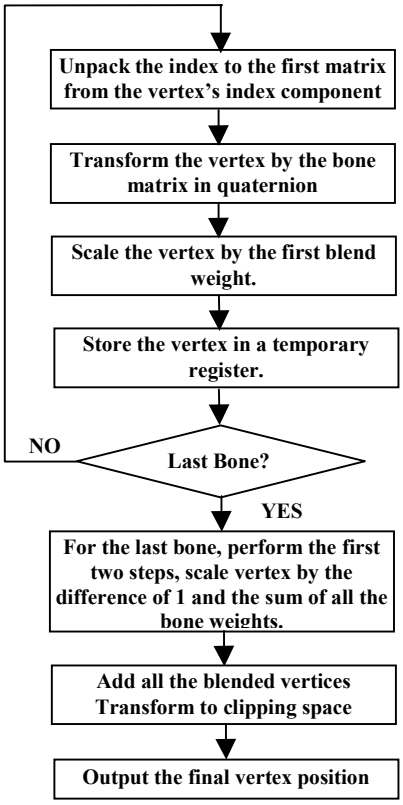


Figure 2 Vertex shader programming procedure

memory, temporary registers and per-vertex output-registers. When it is activated in program, it replaces the transform and lighting computation of the fixed-function pipeline for vertex. A vertex shader operates on a single vertex at a time. A typical vertex shader implementation for a vertex in mesh skinning is shown in Figure 2 (Freidlin 2001).

Prior to the above procedures, the character needs to be modelled and a number of bone matrices need to be assigned to some skin vertices. The mesh also needs to be

translated via the function ConvertToIndexedBlendedMesh() in order to be influenced by certain transformation bones.

EXAMPLE ANIMATION

This section describes an example animation using the method outlined in above section. The character has

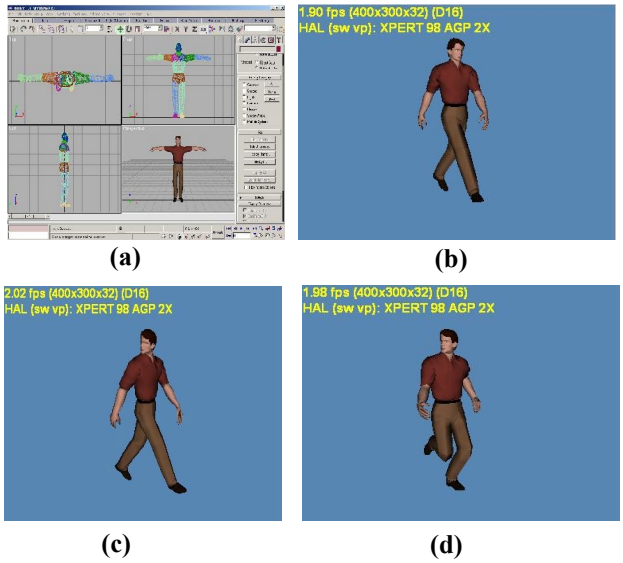


Figure 3 Screen shots from the example animation

(a) Establish the model in 3D studio Max
(b) Character is walking
(c) Character is walking and looking around with caution
(d) Character is running away and looking around with scare

around 19 bones and it is designed to be able to exhibit its behaviours based on the run-time of the game environment and its internal states in real time. In this example, the character has two internal states, namely CAUTION and HAPPY and several behavioural elements such as "walking", "standing", "looking around" and "running". Figure 3a shows the character model, which has been developed in 3D Studio Max. In Figure 3b the character is walking around with HAPPY and CAUTION being kept in at intermediate level. The character in Figure 3c is walking and looking around with caution due to noise/sound being detected. The level of CAUTION has gone up in this situation. Figure 3d shows that after the CAUTION level has been reached a high level, the character is starting to run away while looking around.

CONCLUSIONS AND FUTURE WORK

The paper has proposed a methodology on how to apply mesh-skinning technique to realistically animate an intelligent game character in virtual environments. This method has been shown to be an efficient way to generate real time intelligent game character animation. The game character is capable of performing various movements based on run time game environment, user interaction and its own internal states. Future work will concern efficient use of the vertex shader to produce more complicated character's motion in real time.

REFERENCE

Anderson, E. F. 2001. " Real-time character animation for computer game". http://www.pope-mp.org.uk/projects/tentacle, last accessed 01/09/2001.

Burke, R.; Isla, D.; Downie, M.; Ivanov, Y. and Blumberg, B. 2001 " Creature smarts: the art and architecture of a virtual brain." http://www.gdconf.com/archives/proceedings/2001/burke/BurkeR.htm, Last accessed 01/09/2001.

Cebenoyan, C. 2001."Efficient Animation". http:// partners.nvidia.com/ Marketing/ Developer/ DevRel.nsf/ TechnicalPresentationsFrame? OpenPage last accessed 01/09/2001.

Domine, S. 2001 "Mesh Skinning". http:// partners.nvidia.com/ Marketing/Developer/DevRel.nsf/TechnicalPresentationsFrame?OpenPage, last accessed 01/09/2001.

Freidlin, B. 2001"DirectX8: Enhancing real-time character animation with matrix palette skinning and vertex shaders". http://msdn.microsoft.com/ msdnmag/ issues/01/06/ Matrix/print.asp, last accessed 01/09/2001.

Granieri, J. P.; Becket, W.; Reich, D.; Crabtree, J. and Badler, N.I. 1995. "Behavioral control for real-time simulated human agents." *Proceedings of the 1995 symposium on Interactive 3D graphics* (NewYork, USA, April), ACM Press New York, NY, 173 – 180.

Lander, J. 2001a "Skin them bones: game programming for the web generation".http:// www. darwin3d.com/ gamedev.htm, last accessed 01/09/2001.

Lander, J 2001b "Slashing through real-time character animation". http://www.darwin3d.com / gamedev.htm, last accessed 01/09/2001

Lander, J. 2001c "Oh my god I inverted kine: inverse kinematics for real-time games." http:// www.darwin3d.com/ gamedev.htm, last accessed 01/09/2001.

Mehdi, Q.; Wen, Z. and Gough, N.E. 2001 "Visualisation system for agent behaviours in virtual environments". *Proc. 10th International conference of Intelligent System,* (Arlington, Virginia USA, June 13-15, 2001), pp 47-50.

Moller, T. and Haines, E. 1999. *Real-time rendering*. A K Peters Natick, Massachusetts.

Thalmann, D. 1996. "Physical, Behavioural, and Sensor-based Animation", In *Proc. Graphicon96* (St Petersburg, Russia) 214-221.

Thalmann, D. 1995. "Virtual Sensors: A key Tools For the Artificial Life of Virtual Actors". *In Proc. Pacific Graphics'95* (Singapore) 22-40.

Weber, J. 2001 " Run-time skin deformation" http://www.intel.com/eBusiness/business/digitalmedia/wp012209_sum.htm, last accessed 01/09/2001.

Wloka, M. and Maughan, C. 2001 "Vertex shader introduction". http://partners.nvidia.com/Marketing/Developer/DevRel.nsf/WhitepapersFrame?OpenPage, last accessed 01/09/2001.

Wen, Z.; Mehdi, Q, Gough, N.E and Allen, M.J. 2000. " Creating animated behavioural game characters based on environmental effects", *Proc. Game On 2000, International Conference on Intelligent Games and Simulation* (London, UK November 11-12,2000*),* pp. 76-80.

ALGORITHMS FOR GAMES SIMULATION AND AGENT PATH PLANNING

REAL-TIME EDGE FOLLOW: A NEW PARADIGM TO REAL-TIME PATH SEARCH

Cagatay Undeger
Modeling and Simulation Section
Defense Technologies Enginering Inc.
35. Sok. No.28 Balgat
06520 Ankara,
Turkey
E-mail: cundeger@ceng.metu.edu.tr

Faruk Polat
Department of Computer Engineering
Middle East Technical University
06531 Ankara,
Turkey
E-mail: polat@ceng.metu.edu.tr

Ziya Ipekkan
Scientific Decision Support Center
Turkish General Staff (TGS)
Bakanliklar
06100 Ankara,
Turkey
E-mail: zipekkan@tsk.mil.tr

KEYWORDS
Computer-generated forces, real-time path search, maze problems.

ABSTRACT

Path searching and mission planning are challenging problems in many domains such as war games, robotics, military mission planning, computer-generated forces, etc. The objective of this study is to develop a real-time path-planning algorithm to accomplish specified missions on large landscapes. For that purpose, a real-time goal-directed path search algorithm, Real-Time Edge Follow (RTEF), which can work on fully known, partial known or completely unknown maze environments, is developed. RTEF aims to find a path from a staring point to a static or dynamic target point in real-time. The basic idea behind the RTEF is to let the agent eliminate closed directions (the directions that cannot reach the target point) by analyzing obstacle edges in order to decide on which way to go (open directions). For instance, if the agent has a chance to realize that moving to north and east won't let him reach the goal state (although the target is at north-east), then he will prefer going to south or west. RTEF finds out these open and closed directions, so decreasing the number of choices the agent has and significantly shortening the path. The method is tested on large mazes and compared with Real-Time A*. We observed that RTEF always performs much better than RTA* when solution quality is considered and usually better when total time spent to reach the goal state is considered (especially on complicated mazes). RTEF frequently gives high solution quality, which is in most cases near to optimal solution, and never needs to return to a previously visited cell while on the way.

INTRODUCTION

Multi agent systems can be used to model computer-generated environments where intelligent agents react suitably to various events. Many of the applications in this context need realistic environment generation, efficient search algorithms and heuristics suitable for real-time simulations. Multi agent systems are integrated into these simulations for supporting automatic and semi-automatic human and group behaviors to complete a given mission. Planning a mission usually means to plan a sequence of actions that lead to the goal-state.

The problem of path planning can be described as finding a sequence of state transitions from some initial state (starting point) to a goal state (target point), or finding out that no such sequence exists. Path-planning algorithms can be off-line or on-line. Off-line path planning algorithms like A* [Russell and Norving 1994] find the whole solution before starting execution. They plan paths in advance and usually find optimal solutions. Their efficiency is not considered to be crucial and the agent just follows the generated path. Although this is a good solution for a static environment, it is completely infeasible for dynamic environments, because if the environment or the cost function changes, the remaining path may need to be re-planned, which is not efficient for real-time applications. Real-time path planning algorithms such as Real-Time A* (RTA*), Learning Real-Time A* (LRTA*) [Knight 1993; Korf 1090], Moving Target Search [Ishida and Korf 1995], Bi-directional Real-Time Search [Ishida 1996], Real-Time Horizontal A* [Undeger 2001], D* [Stentz 1994], Focused D* [Stentz 1995] are on-line and offer more efficient solutions. Some of them produce optimal solutions for dynamic changes such as D* and Focused D*, and some only bring efficiency but not optimality such as Real-Time A*.

In this paper, we have proposed a real-time path search algorithm, "Real-Time Edge Follow" (RTEF), which provides both efficiency and solution quality on maze environments. The algorithm is capable of searching a path in real-time to reach a static or dynamic target on a fully known, a partial known or a completely unknown mazes. It can also be used in real-life applications uncluding robotics. RTEF is tested on randomly generated mazes and large terrain data, and compared with RTA*. The results showed that RTEF performed better in both solution quality and execution time.

In Section 2, a survey of related work on path planning is given. Our real-time path search algorithm, Real-Time Edge Follow is described in detail in Section 3. The

performance analysis of RTEF is given in Section 4. Finally, the conclusion is given in Section 5.

RELATED WORK

Multi-agent systems are used in many domains such as robotics, computer generated forces, games, training, RoboCup soccer and their simulators. In robotics [LaValle and Kuffner 1999] and RoboCup soccer [Jensen and Veloso 1998], intelligent planning aims to find out ways for interacting with the physical world, which makes the problem hard to solve. In contrast, intelligent planning for computer generated forces and games [Pew and Mavor 1998; Undeger et al. 2000; Undeger 2001; Baxter and Horn 1999; Gelenbe 1998; DeLoura 2000] aim to generate behaviors similar to the real world in virtual environments. Simulating real world actions in a virtual environment is basically used to test some conditions that are not possible or hard to generate in the real world. For example, intelligent agents that behave much like real world entities are frequently used for pilot trainings in flight simulators [Jones et al. 1993, 1994]. In such simulators, realistic modeling of agent behaviors is very important for the realism of training.

In multi agent simulations, evaluating the environment information, learning and reacting in time is essential. Erol Gelenbe proposed modeling computer-generated forces with learning stochastic finite-state machines whose state transitions are controlled by state and signal dependent random neural networks [Gelenbe 1998]. In Knuffner's approach [Kuffner and Latombe 1999], rendering off-screen from the character's point of view and real-time path planning is used. His path-planning module aims to find a collision free path between a starting and ending point over the 3D terrain using the information gathered from vision based perceptions. In the study of Knuffner, the terrain is divided into embedded graph cells, which have vertical, horizontal and diagonal costs of walking through. Then, the suitable path is found using Dijkstra's algorithm, which is actually an optimal off-line path-planning algorithm integrated into a real-time application. Using an off-line path-planning algorithm in a real-time application is not suitable for large terrains. By the help of some guidance such as admissible heuristics can increase efficiency of off-line path planning algorithms. A* [Russell and Norving 1994] is one of best-known efficient path planning algorithms, which is guided by a heuristic function. A* always finds the optimal solution and uses linear distances between points for the heuristic function. Optimal path planning algorithms cannot be used for large and dynamic landscapes because of its complexity. To avoid this drawback, some partial path update algorithms are also proposed such as D* [Stentz 1994] and focused D* [Stentz 1995]. These algorithms plan an off-line path, let the agent follow the path, and if any new environment information is gathered, they partially re-plan the existing solution. But some times, a small change in the environment may cause to re-plan almost a complete path, which may take a long process time.

A number of algorithms exist for supporting real-time simulations such as Real-time A* (RTA*) and Learning Real-Time A* (LRTA*) [Knight 1993; Korf 1990]. RTA* uses a greedy search strategy and a heuristic together to guide the search. It guarantees to find a solution if one exists, but the solution may not be optimal. In this paper, we have proposed a new Real-Time path search algorithm for maze environments.

There are also some path-planning algorithms that use random search techniques such as genetic algorithms, random tree generators. In [Sugihara and Smith 1997], an adaptive path-planning algorithm based on genetic approach is proposed. In this study, they assumed that a valid path that is not optimal is initially found and they refine this given path by genetic algorithm. Considering this concept, our previously developed off-line path-planning algorithm, Linear Search Path Finder [Undeger 2001] seems to be applicable to this case successfully. In the study of LaValla and Knuffner [LaValle and Kuffner 1999], a randomized planning technique based on a version of random tree generation called rapidly exploring random tree is presented. They generated two random trees starting from the goal and the target points, and try to catch an intersection among the points of distinct trees to find a path.

REAL-TIME EDGE FOLLOW

RTEF is developed for maze-type problems and aims to find a path from a staring point to a target point in real-time. The basic idea behind the RTEF is to let the agent eliminate closed directions (the directions that cannot reach the target point) in order to decide on which way to go, (open directions). For instance, if the agent has a chance to realize that moving to north and east won't let him reach the goal state, than he will prefer going to south or west. RTEF finds out these open and closed directions, so decreasing the number of choices the agent has.

The environment is assumed to be a grid world with obstacles, where the cells having obstacles are marked with a non-zero value and the rest is filed with zero. Initially, the agent is at a starting point and the goal is to reach a static or dynamic target. At each move, RTEF algorithm is executed to eliminate closed directions from the current cell in order to select the next move from a set of open directions. The cost of moving to the next cell + the distance to the target is used for the heuristic function to select one of the open alternatives. After doing the move, the previous cell is marked as an obstacle; so loops are prevented. The set of previously marked cells are called the history of the agent. In exploration mode with an unknown maze, the history has to be cleared when a newly discovered obstacle blocks the way that was open before.

The algorithm is constructed on the idea that every obstacle has a boundary, which is actually formed from a set of connected edges shaping the object. The obstacles are

defined as a set of merged cells in a grid world. RTEF splits the moving directions from each other by a set of rays sent away from the agent, and analyze each region to discover whether it is closed or open.

Moving Directions and Sent-Rays

RTEF accepts the possible moving directions as north, south, east and west. Although there are many interval values of these directions, a set of basic directions is chosen to partition the area for the sake of efficiency. The idea is not to find an exact direction, but to choose a general direction having a left and right angle limit. The algorithm splits the visual environment into four regions by sending 4 rays away from the agent. The rays are sent along 4 diagonal directions (the angle between two adjacent rays is 90 degree). The rays go away from the agent until hitting an obstacle and hence split the area into 4 different regions. Some of these regions are on the way to the target point and some are not. This is illustrated in Figure 1.

Each ray hits the boundary of one of the obstacles. The cells on the way of a ray are followed until a blocked cell is detected. The boundary of the grid world is also marked as blocked in order to prevent infinitely going rays. The first ray is assumed to be on the northwest diagonal. This is illustrated in Figure 2. A ray is assumed to hit an obstacle, when any cell on the way is marked as blocked.

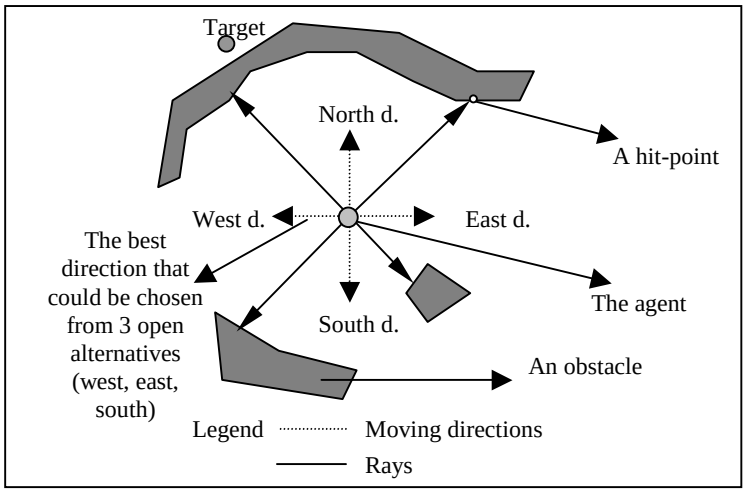


Figure 1: Sending rays until hitting an obstacle to split the basic directions

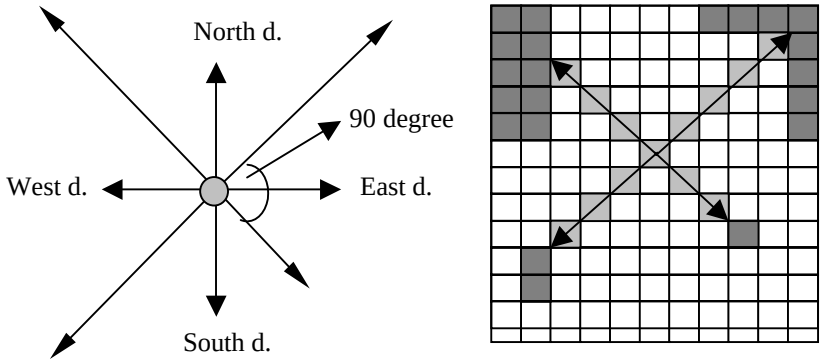


Figure 2: Rays, Directions and hitting: Rays and directions are numbered from 0 to 3 in clockwise direction on the left. The figure on the right shows the cells on the way of 4 rays.

Following Edges

Four rays split the area around the agent into 4 regions. Some of these limited regions are closed and target point is inaccessible from any point located inside these regions. If all are closed, that means the target is inaccessible from the current state or location. To detect which ones are closed, the boundaries of obstacles that the rays hit must be analyzed. The edges on the boundaries are followed and if edges are followed clockwise or counter-clockwise directions starting from a hit-point, we always return to the same point. By following edges and returning to the same starting point on the boundary, a polygonal area is formed (the boundary of the obstacle is detected). We call this polygonal area an "*island*". There are two kinds of islands: outwards facing and inwards facing, which are shown in Figure 3. If the target is inside an outwards facing island or

outside an inwards facing island, that means the target is inaccessible from the current location.

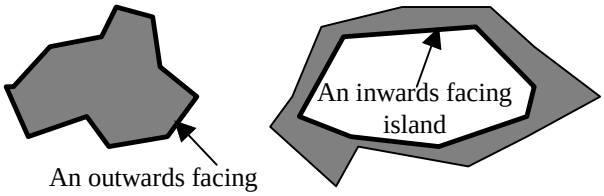


Figure 3: Island types: outwards facing (top), inwards facing (bottom)

If we reach the hit-point of another ray while following edges (only the first one reached is considered), we have an additional polygonal area (the two rays are also included) called *"hit-point-island"*. This polygonal area is illustrated in Figure 4.

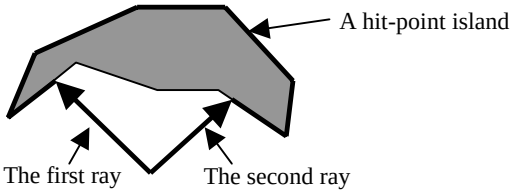


Figure 4: Two rays hitting the same obstacle at two different points

A hit-point-island borders one or more moving directions. If the target point is not inside the hit-point-island, it means that all the directions that are bordered by the hit-point-island are closed. Otherwise all the directions that are not bordered by the hit-point-island are closed. This is illustrated in Figure 5.

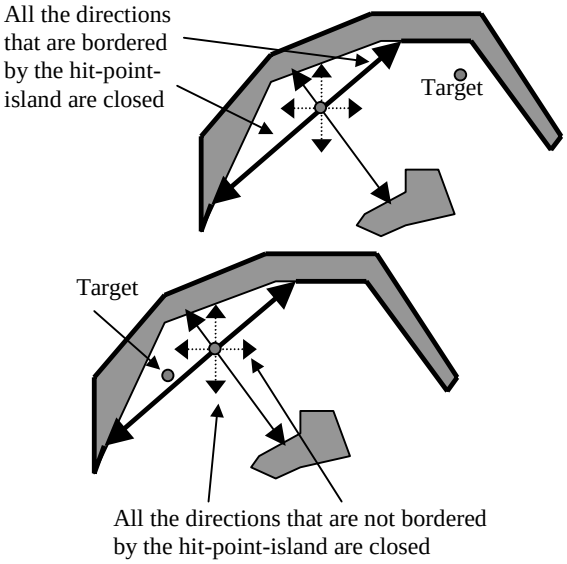


Figure 5: Analyzing hit-point islands and eliminating moving directions

The edge following algorithm works on grid worlds and it is the most time consuming part of RTEF. There are two edge follow directions. One follows the edges from the left side of the hit-point and one follows from the right side. If we follow edges from the left, we have to choose the next edge, which is connected to the left side of the current edge and similar process is done for the right side. This is depicted in Figure 6.

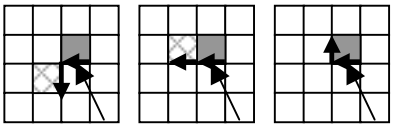


Figure 6: Edge detection: The figure illustrate next possible states when starting from south side of a cell. The possibilities are tested from smaller angles to larger angles.

After finding the island and the hit-point-island, the target must be checked if it is located inside these polygonal areas or not. Implementing an inside test is not so complicated. For testing whether the coordinate (*tx*,*ty*) is inside a polygon *P* or not, we just need to count the number of edges in *P* that are on the left side of *ty* and intersecting the horizontal line passing on *ty*. If the edge count is an odd number, the point is inside the polygon else it is not.

Detecting Closed/Open Regions

To detect closed and open regions, "edge following process" and "inside polygon tests" are used together. To follow edges from only one direction is enough for the detection, but one direction has to be chosen and used for all the rays. The C-like pseudo code for left side edge following is shown in Figure 7. When open and closed directions are detected, one of the open directions is chosen and one-step move is performed. The simplest heuristic is the cost of moving to the next cell + the distance of next cell to the target. The direction, which minimizes the cost, is selected for the next move.

Although the algorithm finds only the open directions, there is a high possibility of getting into loops, especially when there is more than one open direction at a time. For example, assume that the agent is at the same y coordinate as the target is, and there is one choice "moving north". So, the agent moves north. Consider the next state has two open regions: north and south. South is the previous cell. If the agent selects the south, it is obvious that it will go into a loop. The agent selects the south, because being at the same y coordinate as the target is, has less cost than being at two cells up from the y coordinate of the target (the distance to the target is shorter). This is only one of many possibilities. The simplest solution to avoid the loops is to mark the previous point as an obstacle. So the agent never has a chance to visit the same point again.

```
For each ray-sent
   Find hit-point of the ray;
   Follow the edges from left side of the hit-point;
   Find out the island and hit-point-island;
   if (an hit-point-island is found)
   {
      if (target_x,target_y) is inside the hit-point-
      island)
      {
        if ((target_x,target_y) is inside the island)
         {
            if ((agent_x,agent_y) is not inside the
            island)
                  close all the directions; // the
                                           target is
                                           inaccessible
                                           ;
         }
         else
         close the directions that are not bounded by
         hit-point-island.
      }
      else close the directions that are bounded by
      hit-point-island.
   }
   else
   {
    if ((the island is outwards facing and
    (target_x,target_y) is inside the hit-point-island) or
    (the island is inwards facing and (target_x,target_y)
    is not inside the hit-point-island))
       close all the directions; // the target is
                                 inaccessible;
   }
```

Figure 7: The C-like pseudo code for detecting closed and open regions

Exploring and Adaptation to Dynamic Environments

If the maze is not given in advance or just a part of the maze is known, the agent has to explore and learn the environment in real-time. The algorithm is applicable for real-time exploration. It will immediately adapt itself to the new condition, but there can be a problem because of marking the cells traveled with obstacle. Because the agent assumes that the rest of the way is opened, so it marks the previous cells as obstacle, but when it explores the area and notices that the region is closed, it may be stuck in a completely closed region. All we need to do is to clear the history when all the directions are closed.

In real life, the targets, the obstacles and the threats usually have a dynamic nature. For RTEF having a dynamic environment is not a big problem in fact. The algorithm can also be adapted to a change in the cost function rapidly. The only problem that may occur is blocking. For example, while the target is moving, it may enter to a previously blocked region, but if the target is blocked, the agent rapidly detects that all the directions are closed and the history is cleared.

One disadvantage of clearing history is forgetting previous experience and returning back to same regions again. While marking traveled cells as obstacles, one good thing is also done. The noise (small obstacles) on the maze is cleared by connecting them with history lines. So clearing the history causes noise to increase again.

Complexity of RTEF

The worst case complexity of RTEF for each move is O(w x h), where w is the width of maze and h is the height of maze. But almost never we have this complexity. In general, it is much more smaller than this. The efficiency of a move can be explained with total number of edges followed. Other operations are very efficient compared to edge following process. The observed worst-case frame rates for different sized mazes are shown in Table 1. As seen, the seconds per move increases almost proportional to the number of cells.

Table 1: Worst-case performances on sample mazes: Sec/M = seconds per move, M/sec = moves per second, Increase = Increase compared to previous column

	100x100	200x200	400x400	800x800	1600x1600
Sec/m	0.005	0.024	0.110	0.462	1.935
M/sec	200.000	41.666	9.090	2.164	0.516
Increase	-	4.800	4.583	4.200	4.188

PERFORMANCE ANALYSIS

The test environment is implemented in Borland Delphi programming language under Windows platform. The tests are run on an Intel Celeron-466 with 128 MB memory. Both RTEF and RTA* (described in subsection “Real-Time A*”) are implemented. We observed that the solution quality of RTEF is always better than RTA* in both simple and complicated mazes. The efficiency (seconds per move) of RTEF is satisfactory in most cases, but not better than RTA*. The efficiency of RTEF depends on the environment complexity, while the efficiency of RTA* is nearly constant.

Three different versions of RTEF algorithm are realized and tested using randomly generated mazes, and compared to RTA*. The first RTEF algorithm knows the entire maze in advance, so has a chance of better planning. The second and third ones don’t know the maze. The agents can see the neighboring cells to a certain depth. They completely see a square region with size d x d, where d is twice of their visual depth plus one. For example, if the observation depth is 5, the agent knows a square of 11x11 cells (5 left, 5 right, 5 up and 5 down). The agent can build the map with real data in time.

Real-Time A* (RTA*)

RTA*, which RTEF is compared to, is a greedy search algorithm that uses heuristics to direct the search. It evaluates the costs of the neighbor voxels at the current position and jumps to the voxel having minimum cost. While jumping to the next voxel, the algorithm writes 1 plus the cost of the second best neighbor to the previous voxel. This is illustrated in Figure 8.

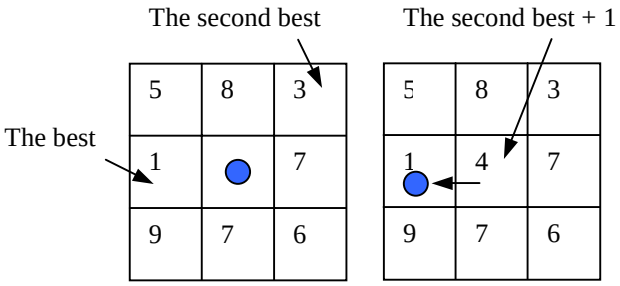


Figure 8: State transition of Real-Time A*

The algorithm is effective and complete for maze environments, but if the terrain is large and there are many semi-closed regions having large open areas inside, the agent may be stuck in the regions for a long time, because the search strategy is too local, only the neighbor voxels are evaluated. In addition RTA* uses much memory than RTEF, because it needs an additional array to store cost updates. And RTA is also not applicable to dynamic environments in practice.

Test Results of RTEF

The mazes with size 100x100 and 500x500 are randomly generated. The agents and the targets are positioned on upper-left, lower-right or upper-central, lower- central side of the mazes. Total time passed, average moves/sec and the number of moves to reach the target point are used to evaluate the results. We evaluated the efficiency and solution quality of RTEF. The average values of test results are shown in Table 2 and 3.

As seen from the results, RTEF always gives better solution quality than RTA* on the average. The difference significantly increases if complicated mazes are used. The RTEF again has a good solution quality when unknown mazes are explored. The best solution quality over RTA* is 995.741 times better than RTA* on the average. The worst solution quality of RTEF is again better than RTA*, which is 1.010 times better than RTA*. The average moves/sec (moves per seconds) of RTA* is 2395 (almost constant), where average moves/sec of RTEF is 371 (varies from 29 moves/sec to 991 moves/sec). This moves/sec is much more smaller than RTA*, but acceptable for soft real-time systems. In robotics, doing unneeded physical actions is much more time consuming than thinking and doing good actions, so the moves/sec is not a disadvantage at all. In real world, the time to walk to the next state will usually take more time than thinking phase, and the next move can be calculated while walking to the next state. If we examine total time spent to reach the goal state, an interesting result is observed. In very simple mazes the efficiency ratio of RTA* is better than solution quality ratio of RTEF. But in simple mazes, they go head to head. And after simple mazes, the solution quality and efficiency generally seems to be better than RTA* and after average mazes, the RTEF is better both in efficiency and solution quality. Some sample snapshots are shown in Figure 9, 10.

CONCLUSION

In this paper, we have studied the concept of computer-generated forces in order to construct a real-time simulation system. We have generated complicated landscapes similar to real ones and tried to solve real-time path planning problem for mission planning purposes. A test environment is generated and a set of scenarios is performed to evaluate performance of developed algorithm. We observed that RTEF performs better than RTA* in most cases. The algorithm is also applicable to robotics and real-time observation, and it frequently gives high solution quality, which is in most cases near to optimal solution.

By introducing new heuristics, the solution quality can be increased, but these methods may reduce the efficiency of the algorithm. For example, some evaluation methods can be applied using edge follow information.

As a result of our observations, we state that the real-time path planning techniques can be improved by increasing the visual search depth, which helps a lot to escape earlier from the local semi-closed regions. But we have also noticed that there is much to do for better intelligent search strategies. More human-like evaluation techniques are needed to go one step forward.

Figure 9: A Maze with 5 cell corridors: RTA* performs 226772 moves in 102.034 seconds (top), RTEF performs 1664 moves in 46.474 seconds (bottom). Black regions are traveled cells. RTA* may travel a cell more than once.

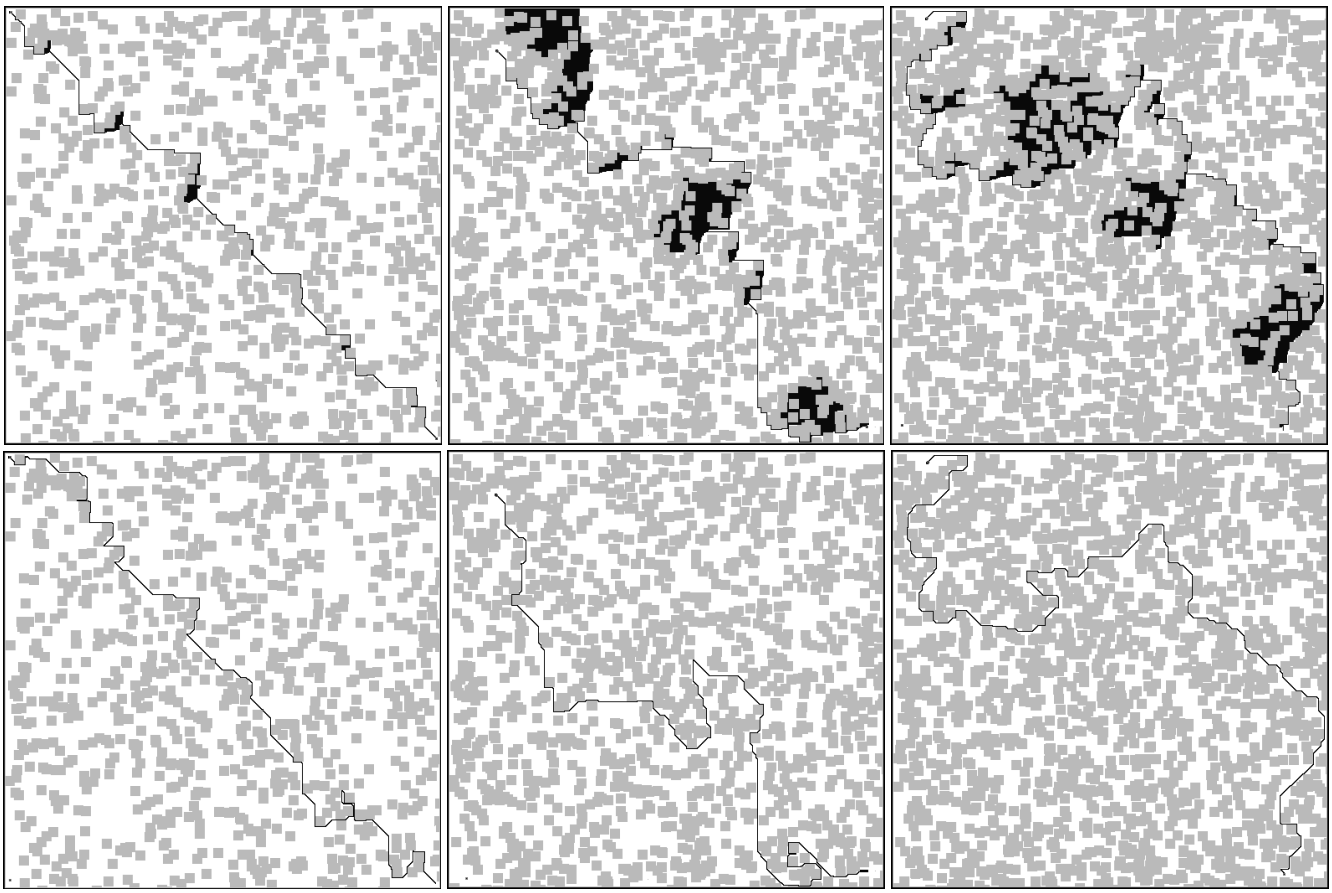


Figure 10: A Simple Maze (left): RTA* performs 2163 moves in 0.890 seconds (top-left), RTEF performs 1325 moves in 2.840 seconds (bottom-left). A Complicated Maze (Middle): RTA* performs 6157 moves in 27.024 seconds (top-middle), RTEF performs 1487 moves in 13.680 seconds (bottom-middle). A Very Complicated Maze (right): RTA* performs 76387 moves in 34.105 seconds (top-right), RTEF performs 1670 moves in 20.870 seconds (bottom-right).

Table 2: Test results (mazes filled with randomly distributed noise)

Average of Simple Mazes: 500x500 cells	Time in Seconds	Average Moves/Sec	Number of Moves	Efficiency of RTEF, RTA* Time / Time	S. Quality of RTEF, RTA* Moves / Moves
RTA*	1.665	2408	3950	1.000	1.000
RTEF (maze is fully known)	2.850	449	1282	0.584	3.081
RTEF (observation depth 5)	3.490	507	1771	0.477	2.230
RTEF (observation depth 10)	3.889	389	1513	0.428	2.610

Average of Average Mazes: 500x500 cells	Time in Seconds	Average Moves/Sec	Number of Moves	Efficiency of RTEF, RTA* Time / Time	S. Quality of RTEF, RTA* Moves / Moves
RTA*	4.058	2367	9718	1.000	1.000
RTEF (maze is fully known)	4.695	218	1026	0.864	9.471
RTEF (observation depth 5)	2.653	660	1753	1.529	5.543
RTEF (observation depth 10)	3.018	404	1221	1.344	7.959

Average of Complicated Mazes: 500x500 cells	Time in Seconds	Average Moves/Sec	Number of Moves	Efficiency of RTEF, RTA* Time / Time	S. Quality of RTEF, RTA* Moves / Moves
RTA*	57.685	2468	94365	1.000	1.000
RTEF (maze is fully known)	19.138	74	1428	3.014	66.081
RTEF (observation depth 5)	29.094	285	8317	1.982	11.346
RTEF (observation depth 10)	18.802	300	5652	3.068	16.695

Average of Very Complicated Mazes: 500x500 cells	Time in Seconds	Average Moves/Sec	Number of Moves	Efficiency of RTEF, RTA* Time / Time	S. Quality of RTEF, RTA* Moves / Moves
RTA*	58.766	2415	142089	1.000	1.000
RTEF (maze is fully known)	21.932	62	1373	2.679	103.487
RTEF (observation depth 5)	21.897	422	9245	2.683	15.369
RTEF (observation depth 10)	21.960	369	8123	2.676	17.492

Table 3: Test results (5 and 1 cell corridors with sizes 500x500 and 100x100)

Average of mazes with 5 cell corridors: 500x500 cells	Time in Seconds	Average Moves/Sec	Number of Moves	Efficiency of RTEF, RTA* Time / Time	S. Quality of RTEF, RTA* Moves / Moves
RTA*	843.917	2495	2176692	1.000	1.000
RTEF (maze is fully known)	73.257	29	2186	11.519	995.741
RTEF (observation depth 5)	581.190	122	71377	1.452	30.495
RTEF (observation depth 10)	167.415	119	19983	5.040	108.927

Average of mazes with 1 cell corridors: 100x100 cells	Time in Seconds	Average Moves/Sec	Number of Moves	Efficiency of RTEF, RTA* Time / Time	S. Quality of RTEF, RTA* Moves / Moves
RTA*	3.861	2434	9201	1.000	1.000
RTEF (maze is fully known)	2.801	117	329	1.378	27.966
RTEF (observation depth 5)	2.808	434	1220	1.375	7.541
RTEF (observation depth 10)	2.385	295	704	1.618	13.069

REFERENCES

Baxter, J.W. and Horn, G.S. 1999. "A Model for Co-Ordination and Co-Operation Between CGF Agents." In *Proceedings of 8th conference on Computer Gererated Forces and Behavioral Representation*, Orlando, Florida, 101-111.

DeLoura, M.A. 2000. *Game Programming Gems.* Charles River Media.

Gelenbe, E. 1999. "Modelling CGF with Learning Stochastic Finite-State Machines." In *Proceedings of* 8th *conference on Computer Gererated Forces and Behavioral Representation*, Orlando, Florida, 113-115.

Ishida, T. and Korf, R.E. 1995. "A Moving Terget Search: A Real-Time Search for Changing Goals." *IEEE Trans Pattern Analysis and Machine Intelligence*, Vol.17, No.6, 97-109.

Ishida, T. 1996. "Real-Time Bidirectional Search: Coordinated Problem Solving in Uncertain Situations." *IEEE Trans Pattern Analysis and Machine Intelligence*, Vol.18, No.6.

Jensen, R.M. and Veloso, M.M. 1998. "Interleaving Deliberative and Reactive Planning in Dynamic Multi-Agent Domains." In *Proceedings of the AAAI Fall Symposium on Integrated Planning for Autonomous Agent Architectures*, AAAI Press.

Jones, R.M., Tambe, M., Laird, J.E. and Rosenbloom, P.S. 1993. "Intelligent Automated Agents for Flight Training Simulator." In *Proceedings of* 3th *conference on Computer Gererated Forces and Behavioral Representation*, Orlando, Florida, 33-42.

Jones, R.M., Laird, J.E., Tambe, M. and Rosenbloom, P.S. 1994. "Generating Behavior in Response to Interacting Goals." In *Proceedings of* 4th *conference on Computer Gererated Forces and Behavioral Representation*, Orlando, Florida.

Knight, K. 1993. "Are Many Reactive Agents Better Than A Few Deliberative Ones?" In *Proceedings of The International Joint Conference on Artificial Intelligence*, 432-437.

Kuffner, J.J. and Latombe, J.C. 1999. "Fast Synthetic Vision, Memory, and Learning Models for Virtual Humans." In *Proceedings of Computer Animation*, IEEE, 118-127.

Korf, R.E. 1990. "Real-Time Heuristic Search." *Artificial Intelligence*, Vol.42, No.2-3, 189-211.

LaValle, S.M. and Kuffner, J.J. 1999. "Randomized Kinodynamic Planning." In *Proceedings of IEEE International Conference on Robotics and Automation*, ICRA'99, Detroit, MI.

Pew, R.W. and Mavor, A.S. 1998. *Modeling Human and Organizational Behavior: Application to Military Simulations*, National Academy Press.

Russell, S. and Norving, P. 1994. *Artificial Intelligence: a Modern Approach*, Prentice Hall.

Stentz, A. 1994. "Optimal and Efficient Path Planning for Partially-Known Environments." In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA '94*, Vol.4, 3310–3317.

Stentz A. 1995. "The Focussed D* Algorithm for Real-Time Replanning." In *Proceedings of the International Joint Conference on Artificial Intelligence*, IJCAI'95.

Sugihara K. and Smith, J.K. 1997. "Genetic Algorithms for Adaptive Planning of Path and Trajectory of a Mobile Robot in 2D Terrains." Technical Report, number ICS-TR-97-04, University of Hawaii, Department of Information and Computer Sciences.

Undeger, C., Isler, V. and Ipekkan, Z. 2000. "An Intelligent Action Algorithm for Virtual Human Agents." In *Proceedings of the* 9th *Conference on Computer Generated Forces and Behavioral Representation*, Orlando, Florida, 25-33.

Undeger, C. 2001. *Real-Time Mission Planning For Virtual Human Agents.* M.S. Thesis in Computer Engineering Department of Middle East Technical University, Ankara, Turkey.

AUTHOR BIOGRAPHY

CAGATAY UNDEGER received his B.Sc. degree from Kocaeli University, Turkey in 1998 and went to the Department of Computer Engineering, Middle East Technical University, where he has worked as a research assistant and obtained his M.S. degree in 2001. He is currently doing his Ph.D. in the same university and studing on Modeling and Simulation within Defense Technologies Engineering Inc.

FARUK POLAT is an associate professor in the Department of Computer Engineering of Middle East Technical University, Ankara, Turkey. He received his B.Sc. in computer engineering from the Middle East Technical University, Ankara, in 1987 and his M.S. and Ph.D. degrees in computer engineering and information science from Bilkent University, Ankara, in 1989 and 1993 respectively. He conducted research as a visiting NATO science scholar at Computer Science Department of University of Minnesota, Minneapolis in 1992-93. His research interests include artificial intelligence, multi-agent systems and object oriented data models.

ZIYA IPEKKAN has been serving as an operations research analyst at TGS for more than twelve years. Lt.Col.Ziya İpekkan is currently the leader for Force Structure Analyses Team at the position since August 2000. He is responsible for conduct of studies and analysis of joint concepts, the warfighting capabilities and plans, and force structures.

USING GAMES ENGINES TO IMPLEMENT INTELLIGENT VIRTUAL ENVIRONMENTS

Carlos Calderon and Marc Cavazza
University of Teesside, TS1 3BA Middlesbrough, United Kingdom,
c.p.calderon@tees.ac.uk, m.o.cavazza@tees.ac.uk

KEYWORDS
Games Engines, IVE, and problem solving mechanisms.

ABSTRACT

In this paper we present an Intelligent Virtual Environment (IVE) obtained by incorporating a problem solving mechanism (AI technique) into a Virtual Environment. In particular, in this paper, we will discuss the implementation of the interaction with the problem solving mechanism by using the following metaphor: the visual space provided by the Virtual Environment is seen as the search space.

INTRODUCTION

The emerging area of Intelligent Virtual Environments explores the integration of Artificial Intelligence techniques into Virtual Reality systems. One particular research topic is to couple interactive AI systems (scheduling, planning, etc.) to virtual environments where the VE serves as an interface to the interactive planning system. Because the nature of the tasks to be solved are mainly spatial, Visualisation Engines, in general, and Games Engines in particular are well suited to serve as interface to the planning and/or problem solving mechanisms. That is, by incorporating AI techniques into 3d real-time interactive graphics technologies, we could, for instance, add a problem-solving component to the virtual environment. This would be beneficial for industrial applications where task scheduling or interactive design is relevant.

The objective of this paper is to present an interactive prototype in which we have linked a Virtual Environment (VE) to a constraint solving mechanism (solver). In our example application is possible to directly manipulate the objects from the virtual environment to create an input configuration for the planning system. This new input configuration is analysed by the problem solving mechanism and then, displayed in the Virtual Environment.

The next section briefly introduces the technologies used to create an Intelligent Virtual Environment for our example application. The third section describes how the problem solving mechanism has been incorporated into the Game Engine. Finally, the paper concludes with a brief discussion about the different implemented types of interactivity, further work lines of work and some conclusions.

TECHNOLOGY BASELINE

Interactive Visualization Engine

We decided to use the Unreal Virtual Machine for three main technical reasons: extensibility, an extremely powerful rendering engine and cutting-edge networking capabilities. Unreal has also been successfully used in non-gaming applications [1] and research projects [2].

Rendering engine and subsystems.

Figure 1. Unreal's scene.

The Unreal rendering engine has the most complete lighting conditions support of all the games engines. A variety of effects: such as surfaces with shadowing, realistic wavy water, animated materials, etc (see figure 1) have been used to add realism to the virtual scene. Unreal also supports digital sound system (software Dolby Surround encoding for full 360-degrees). Real world sounds , e.g water splashing, have been incorporated to the virtual scene to add realism to the exploration of the virtual environment.

Problem Solving Techniques

Our approach is based on a combination of Constraint Logic Programming over Finite Domains (CLP(FD)) and 3D real-time interactive technology.

Our choice of CLP(FD) as a software technology was made based on a series of factors: expressivity, the combination of search and incremental constraint solving capabilities, the short development time while exhibiting an efficiency comparable to imperative languages, and the fact that CLP(FD) is fast enough to react in "real-time" to the user's input configuration

In our concept architecture examples we have used GNU Prolog as a programming environment, which contains an efficient constraint solver over FD. GNU Prolog offers two different propagation techniques to solve arithmetic constraints: full arc-consistency and partial arc-consistency. This coupled with the use of labeling constraints help us to discover inconsistency as soon as possible, thus avoiding futile search through inconsistent alternatives. Hence, efficient and very fast constraint solvers can be programmed.

INTELLIGENT VIRTUAL ENVIRONMENT

This section explains how we have made the constraint solver work jointly with the visualization engine to create an Intelligent Virtual Environment in which the virtual environment serves as an interface to the planning system. The starting point is to formulate the *problem* in terms of constraints (using GNU Prolog). Then the next step is to link our constraint solver with the visualization engine (Unreal Virtual Machine).

The next sub-section explains in more detail how the constraint solver has been implemented.

Constraint Solver

Problem

To experiment we have developed a test application in which the task is to allocate a Coke machine in a bank hall. This task can be seen as an example of a spatial/resource allocation problem. This can be represented as a Constraint Satisfaction Problem (CSP), which, consequently, can be easily formalised in CLP (FD). There are numerous design issues, especially in the context of building design, that can be easily transformed into constraints such as health and safety regulations, urban and planning regulations, etc.

The *Constraints* imposed on the Coke machines are the following: a) sources of heat (i.e. radiators) should stay a minimum distance away from the machines, b) the coke machines can only be placed at a maximum distance away from a socket (power point) c) the machines must not obstruct ventilation ducts, d) the coke machines must stay clear from exits and thoroughfares e) the minimum distance to a wall is 25 cm f) finally, the allocation of coke machines must take into account the existing furniture and decoration in the hall.

Representation of the Search Space

There are at least three options to choose from to represent the spatial search space: rectangular or hexagonal discretisation, actual polygonal floor and polygonal floor representation. There is no obvious choice, each representation has its trade-offs. This decision has great implications in terms of speed to solve the CSP problem and flexibility to accurately represent a constraint satisfaction problem.

In this prototype we have opted for a Rectangular or Hexagonal Grid: A uniform rectangular or hexagonal grid is overlaid onto the world. The size o f each grid space is related to the smallest character. The pros of this approach are: obstacles and characters can be easily marked in the grid; it works well for our 3D world, in our example application the working (spatial search) space is a flat 3D world that is, the z coordinate is constant; save computations on the z coordinate. c

When we translate our CSP problem to a rectangular grid, we are simplifying the 3D space into more familiar 2D terms. The idea is to solve the problem in a pseudo 3D environment (where the Z coordinate is constant) while providing a full 3D representation to the user. To do this, we use a rectangular grid as a mesh to transform the 3D space into a more manageable search space.

We have experimented with different grid size but as far as the search space goes, the maximum number of cells or the finest possible mesh is implementation dependent. This is due to the fact that in GNU Prolog FD variables can only take values in their domains. There are two internal representations for an FD variable: interval and sparse representation. The initial representation for a FD variable is always internal representation and is switched to a sparse when a “hole” appears in the domain (i.e due to an inequality constraint). So to avoid unexpected surprises we have used the following predicate: fd_set_vector_max(512) that sets the environment variable VECTORMAX to the greatest value that any FD variable can take. This means that in our example application the finest possible mesh has a cell size of 37mm by 37 mm in real units, That is, in the real world.

Problem expressed in CLP(FD)

We can easily transform our initial task into a CSP problem. That is, a problem where one is given: a finite set of variables, a function which map each variable to a finite domain, and a finite set of constraints. In our case, the set of variable is (X,Y) where X and Y represent the coke machine’s x and y coordinates in our search space. We map these variables to a finite domain by using the following predicate: fd_domain(Vars,1,36). Finally, we have to impose our constraints.

We have regarded the constraints *d* and *f* as fixed

```
/*distance to a wall, in this case 25cm*/
walldistance(X,Y) :-
                    X=2,!
                    ;
                    Y=2,!
                    ;
                    X=35.
```

```
/*max distance to, i.e, a plug*/
/*In this case the Manhattan distance is 3 units*/
maxdistance(X,Y) :-
                    D is 3,
                    (Xc is 2,
                    Yc is 1,
                    distance(X,Xc,Dc1),
                    distance(Y,Yc,Dc2),
                    Dc1+Dc2#=<D)).
```

Figure 2. Examples of constraints

topological constraints and we have implemented them as facts in the database. Alternatively, we could have implemented them as atomic constraints such as Y#=<10 what could have been more efficient but for simplicity we chose not to. The constraints a and c have been regarded as conjunctive. The formulation of this type of constraints (conjunctive) is straight forward. See figure 3 where Xo and Yo are the position of the source of heat in the search space. Finally, b is regarded as a disjunctive constraint. The formulation of this type of constraints is also simple. See figure 2 where Xc and Yc represent the location of the socket in the search space. We have used the Manhattan distance as distance criterion.

It is worth noticed that the incremental constraint solving capabilities makes very easy to implement more conjunctive constraints. We just need to add a new constraint but with a different Xi and Yi and the solver takes care of the rest (see figure 3). Figure 4 shows a graphical representation of a fraction of the search space and the constraints implemented in it.

```
/*Where X0 and Yo is a ventilation duct*/
/*The criterion to used to calculate the distance is: */
/*The Manhattan distance*/

mindistV0(X,Y) :-
                    D is 1,
                    X0 is 1,
                    Y0 is 1,
                    distance(X,X0,D1),
                    distance(Y,Y0,D2),
                    D1+D2#>D.

/*A new constraint can be added just by changing */
/*te location of the ventilation duct, Xi Yi*/


mindistVi(X,Y) :-
                    D is 1,
                    Xi is 1,
                    Yi is 1,
                    distance(X,X0,D1),
                    distance(Y,Y0,D2),
                    D1+D2#>D.
```

Figure 3. Incremental Constraints

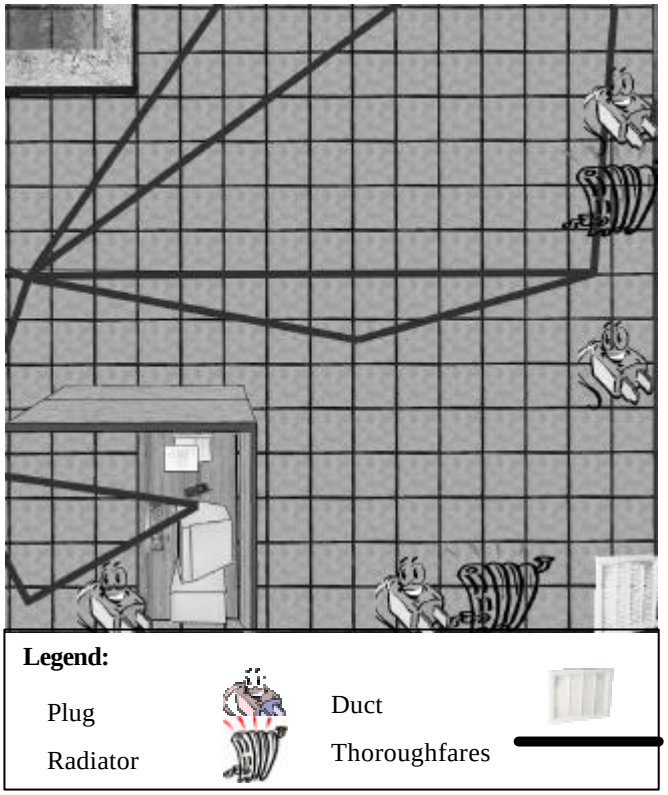


Figure 4. Fraction of the search space with constraints.

System Architecture

Concept Architecture Example

The system is constructed of a number of modules

-LinkServer

This module handles the communication of Unreal with the GNU Prolog solver and it has been implemented using the Berkley socket interface. In short, the aim of this subsystem is to provide a means of inter-process communication that allows bi-directional messages between two processes regardless of whether those processes reside on the same machine or different machines. In our current implementation, the solver is located in a remote location and the communication protocol used to send information across the network is TCP/IP (Transfer Control Protocol and Internet Protocol).

-Interaction Modules (ITModules)

Figure 5. Player explores virtual scene.

A series of Unreal modules have been scripted in order to create a satisfactory and appropriate player's interaction with the world. These modules provide the system with a customized player pawn which has embedded the following functionality: the ability to locate the sought object and Grabbing/Dropping objects at will by just pressing the appropriate key. Figure 5 shows a player exploring the

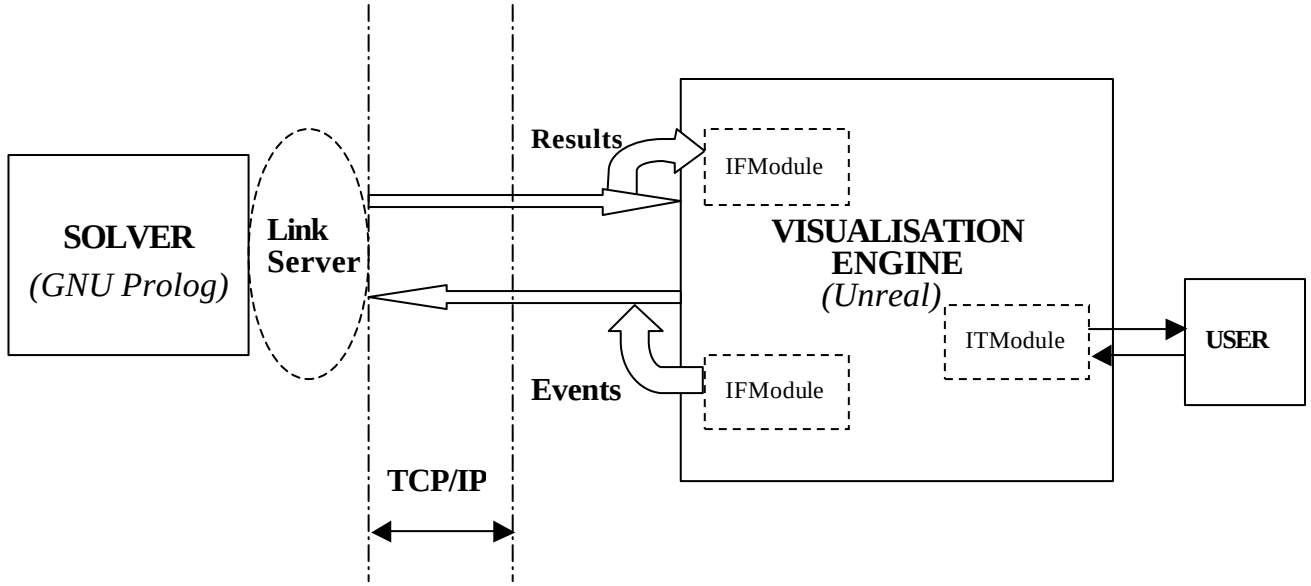


Figure 6 System Architecture.

virtual scene after he/she has "grabbed" a Coke Machine.

-Interface Modules (IFModules)

In order to have a smooth player's interaction with the virtual environment, an interface with the LinkServer module has been embedded in our Player Pawn class. This approach guarantees that interface is switch on/off at the player's will, that is, when the player decides to input a new configuration (i.e by dropping an object).

The basic functionality provided by this interface is the following: a) the player's input configuration is parsed, transformed and send to client-server subsystem b) the new configuration produced by the solver is received from the client-server subsystem, parsed and transformed into UnrealScript commands in order to create the corresponding object configuration which the Unreal Virtual Machine displays in real-time. Figure 6 shows the overall system architecture.

Interaction cycle

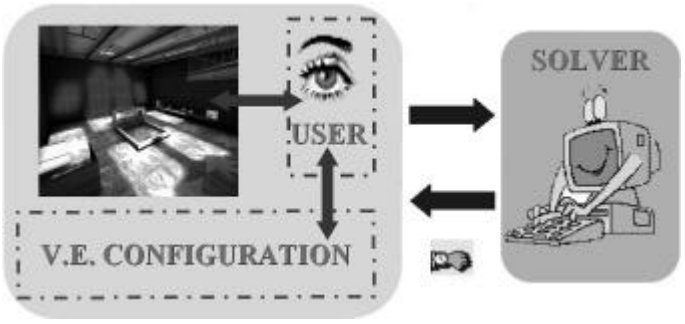


Figure 7. Intercation cycle.

The interaction cycle works as follows:

-Initial State (Initial Knowledge)

An initial configuration is proposed to the user. In our example, a Coke machine is initially allocated in the Virtual Environment.

-Exploratory State

The user explores the initial proposed configuration in the Virtual Environment.

-Solution State

To reach this state the user has previously decided to place the Coke machine in the Virtual Environment in a position different from the one initially given. Once the user has placed the Coke machine, the new input configuration is sent to the solver. This analyzes it and reacts to it. There are two possible scenarios:

a) The new configuration complies with the constraints. Outcome: the system reacts by displaying a confirmation message.

b) The new position does not comply with constraints embedded in the solver. Outcome: the system reacts by returning to the previous valid configuration.

Once the Solution State is completed the user is back to the Exploratory State and consequently, a new interaction cycle commences. Figure 7 depicts the interaction cycle.

FURTHER WORK

We see the results presented on this paper as a step towards the development of reactive environments In practical implementation terms in our example application the following could be an example of a reactive environment:

In our initial scenario we could have two types of constraints: hard constraints (i.e topological) and soft constraints (i.e movable objects). Thus, if player's configuration complies with all the topological (hard constraints) but not with the soft constraints (movable objects) then the system should react by reallocating the movable objects in the new configuration.

DISCUSSION

One key aspect to successfully incorporate planning systems into Visualisation Engines is that AI techniques are often less interactive that it would be required for a complete integration into the virtual environment. In previous papers [3] we discussed and shown that CLP over FD as software

technology it is fast enough to react in "real-time" to user's input configuration. Furthermore, the results shown in table1 underpinned the idea that although constraint programming in itself is not a reactive technique it can be used to emulate reactivity because it can produce a solution quickly enough.

Table1. Solution times.

	To find: First Solution(ms)	To find: All Solutions (ms)
User-time	0	[10-40]
System-time	0	[0-10]
Cpu-time	0	[10-30]
Real-time	[0-2]	[61-63]

Average time in milliseconds over 10 trials

Another key point in order to implement a real-time system is that the sampling rate of object manipulation in the virtual environment must be compatible with the result production granularity of the problem-solving algorithm. In our case, the implementation of a real-time reactive environment will depend not only on using the adequate technology to implement the constraint solver but also on the overall system architecture.

Unfortunately, at this point in time, we do not have enough experimental data to back up this particular point. However, our empirical estimation is that the current implementation produces a satisfactory result in terms of the compatibility of the object manipulation in the virtual environment with the overall result production granularity.

CONCLUSIONS

In this paper we have presented an interactive system in which the virtual environment acts as an interface to the interactive planning system. Because the nature of the tasks to be solved takes place in a 3 dimensional space, Visualisation Engines are suitable tools to serve as an interface to the planning and/or problem solving mechanisms. We have shown that the inclusion of an AI layer adds a problem-solving component to the Virtual Environment. This could also be seen as having a VE where objects have associated behaviours.

ACKNOWLEDGEMENTS

Many thanks to Mr Steve James Mead. for his technical assistance regarding C programming and system architecture.
Many thanks to Mr H. Robert Berry, Jr for his technical assistance regarding UnrealScript.

REFERENCES

[1] De Leon, Victor. (1999), VRND:NOTRE-DAME CATHEDRAL: A Globally Accessible Multi-User Real-Time Virtual Reconstruction, the 5th International *Conference on Virtual System and MultiMedia 1999 (VSMM'99):Next Generation Virtual Reality "Milestones for a New Virtual Milennium"*.Dundee, Scotland, UK, 1-3 September, 1999.

[2] Young, Micheal. (2001), An Overview of the Mimesis Architecture: Integrating Intelligent Narrative Control into an Existing Gaming Environment. In *The Working Notes of the AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*, Stanford, CA, March 2001.

[3] Calderon, C and Cavazza, M. (2001), Intelligent Virtual Environments to Interactively Solve Spatial Configuration Tasks. *Proceedings* of *the Seventh International Conference on Virtual System and Multimedia*. Berkeley, USA, 25-27 Oct 2001

Co-ordination of Multi-agent Path Planning using the Synchronous Near-Admissibility A* (SNA*) Algorithm

M.Shafie Abd Latiff
Ian Palmer
Electronic Imaging and Media Communication
University of Bradford
Bradford BD7 1DP
United Kingdom
E-mail: m.s.abdlatiff@bradford.ac.uk
E-mail : i.j.palmer@bradford.ac.uk

Marc Cavazza
School of Computing and Mathematics
University of Teesside
Middlesbrough, TS1 3BA
United Kingdom
Email : m.o.cavazza@tees.ac.uk

KEYWORDS
Multi-agent co-ordination, path planning, games theory

ABSTRACT

This paper discusses a solution to generate multiple paths using a single algorithm. The paths will disperse among different available roués and reconvene back at the same destination. The solution is unique in terms of co-ordination among agents. Co-ordinated path planning for multi-agents is based on collective behaviour. Admissibility, as a main property of the A* algorithm, provides the basis to incorporate a collective behaviour in path planning. We introduce a new extension of A* called Synchronous Near-Admissible A* or SNA* that produces alternative paths that depart from the shortest path solution.

INTRODUCTION

Co-ordination of agents not only refers to interaction between agents where the goal of each agent's action is to induce change in the other agents (Pinhanes, 1999) but it also includes the sharing of information to work together as a group of agents. For example, in crowd simulation and the movement of groups of artificial animals, information about separation distance is important to ensure that agents will not collide with other agents. Most research into multi-agent movement focuses on how to move the crowd together without inter-agent collisions and avoiding obstacles. The crowd is intended to move from the same location to the same destination. If there are several alternative paths or a number of obstacles providing several paths to the destination, human nature is to choose a different path rather than the same congested path. This is especially evident for crowds in emergency conditions.

There are many real-world applications in which this problem arises, as can be illustrated by a practical example. In many modern computer games, the player's avatar is often confronted with several Non-Player Characters (NPCs). A measure of how "intelligent" these NPCs are is often how well they co-ordinate their attacks on the player, including the path planning phase. In particular, it is easy for the human player to confront opponents lined in a row, all following the same path towards his avatar. A better approach consists of using the player's position as a goal, while approaching him from different directions. (AbdLatiff, 2000).

The design of virtual buildings or dealing with the problem of space layout planning (Medjdoub and Yannou, 2001) is incomplete, generally only fulfilling the dimensional and topological constraints. The approval of building design also includes the safety of the people occupied inside the building. The location of emergency exits is vital and these should be arranged in such locations so that alternative paths are provided and the people could vacate the building in the quickest possible time. The important aspect of security in building design has also been expressed in (Atlas, 1989).

CURRENT WORK

There are requirements in many applications to co-ordinate the movement of a population of artificial actors. For example, in crowd simulation (Thalmann et. al., 2000), battlefield planning (Brock et. al., 1992), flocking behaviour (Reynolds, 1987) etc. Thalmann et. al. (2000) have emphasised the concept of 'Level of Autonomy' and have implemented this concept in crowd behaviour. In this the level of co-ordination among the agents is classified into three levels. On the first level, 'guided crowd' behaviour is defined explicitly by the user. Schweiss (1999) uses a rule-based behaviour system to control them. Programmed crowds are a second level where the behaviour is programmed in a scripting language. On the third level autonomous crowds are specified by rules or other complex methods. Farenc et. al. (2000) present a crowd management method as an architecture for simulating crowds.

Brock et. al. (1992) employ the concept of a group leader in co-ordination and control of multiple autonomous vehicles for soldiers in a training simulation. A flocking algorithm is identified among the early solutions in co-ordinating the movement of a group of animated characters. This was introduced by Reynolds in 1987 to simulate the movement of birds in a park. It is described as 'boid' procedures by Parker (1999) and has been referred to by many researchers in their animation of multi-agents movement simulation. For instance (Bedau, 1992), (Noser, 1996), (Tu, 1996), (Brogan, 1997), etc.

Algorithms that simply co-ordinate agents around a group leader without actually sharing a common goal are faced with a certain number of limitations. For instance, flocking algorithms are vulnerable to local phenomena in the virtual world: if one actor splits from the group it might not be able to reconvene on the basis of the flocking algorithm only, especially if the terrain contains obstacles. More specifically it is difficult with flocking algorithms to use devise strategies for multiple path planning that enable co-ordinated agents to follow different routes to the same destination (AbdLatiff and Cavazza, 2000). Separate paths are created for each agent using linear interpolation as implemented in crowd behaviour and do not actually involve a proper path planning process. The movement is based on guided crowd behaviour from one interest point (IP) to another IP and collision detection with obstacles is included in a separate module for each agent. Schooling behaviour also does not include path planning and the actions are controlled by a separate behaviour routine, for instance avoiding static-obstacles, avoiding fishes, leaving the group, schooling etc.

RESEARCH PROBLEM

There are two aspects to the problem of co-ordination of groups of actors: to search the minimal path from source to destination and to co-ordinate the group of actors which share the same destination or goal. Most of the research on multi-agents movement focuses on how to move the crowd together, without collisions between each other and avoiding obstacles. Most of the solutions to path movement for multi-agents are not based on a path planning algorithms and there is no co-ordination between agents in path generation.

Most of the research on path planning or shortest path algorithms is exclusively meant for an individual agent or a single animated character. Every single character in a virtual environment has to find its own path in order to move from one location to another.

To find the path for a group of actors originating from the same place towards the same destination results in all actors following the same path. This is shown in Figure 1 where three actors find their own paths using a normal A* algorithm, directing from the left middle room to the right room. There are a few obstacles for them to avoid. The result shows that every actor generates more or less the same path to the destination.

METHODOLOGY

Bandi & Cavazza (1999) explained that the path planning algorithm must satisfy some optimality criteria. These optimality criteria include not only the shortest distance but also include application-dependent data that relate to the Virtual Environment semantic. An example of application-dependent data is that the path taken should approach a certain object.

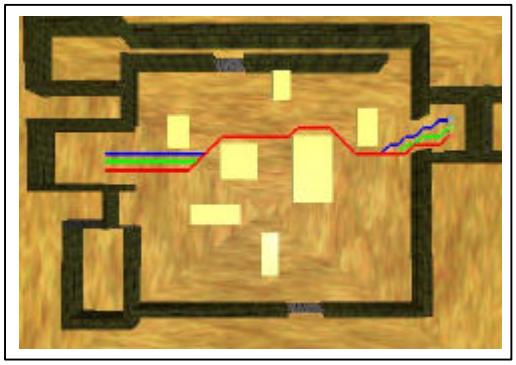


Figure 1: Three agents follow one after another using normal A*

We introduce collective behaviour as application-dependent knowledge incorporated in a secondary heuristic of the A* algorithm. One of the collective behaviours is maximum dispersion. With this behaviour, every agent will take different path from the same start location to the same destination, if such a path exists.

One major property of the A* algorithm is admissibility. In other words A* will be guaranteed to find an optimal solution if the path exists (Nilsson, 1980)(Pearl, 1995)(Luger, 1993). The optimal solution in this case is the shortest distance between the starting point and the destination.

Barr and Feigenbaum (1981) have suggested relaxing the admissibility condition, thus trading optimality of the solution for computation speed. In addition to the OPEN list of expanded nodes, Pearl (1995) has defined a FOCAL list as:

$$FOCAL = \{n : f(n) \leq (1+\epsilon) \min_{n' \in OPEN} f(n')\}$$

where n' is a node in the OPEN list and f(n') is its cost. In A* the node with the lowest cost is selected for the next expansion. In A*ε algorithm, however, the criterion is altered. All the nodes with cost less than or equal to (1 + ε) times the minimum cost in OPEN are considered. This set of nodes is termed FOCAL. A*ε has been introduced as a trade-of between admissibility/optimality and computational efficiency (Bandi and Cavazza, 1999).

The property of admissibility is used to enhance the capabilities of A* algorithm to serve this purpose. Every agent will generate his own path with a different size of FOCAL. This generation is implemented synchronously. Epsilon-admissibility is only guaranteed when the size of FOCAL is small. Practically, the bigger the size of FOCAL the longer the path that exceeds the optimal path. This will give a measure of how much the algorithm departs from admissibility. To reflect the fact that this implementation might depart from admissibility, we will use the term near-admissible rather than ε-admissible. Hence we called this algorithm Synchronous Near-admissible A*, or SNA*.

Primary and secondary heuristic

In A*, the best node from OPEN list is selected as the next node to be expanded. The evaluation function f (n) = g (n) + h (n) is the main function used to determine the order of OPEN list.

Generally, Euclidean distance or Manhattan distance is used as the main geometrical heuristic function, h (n). This is called the primary heuristic and is concerned with the distance between the current node and the destination node. The path generated using only the primary heuristic will resolve the shortest path between start and final node. This path is not concerned with the properties of the environment or any other constraints such as dynamic configuration as described by Bandi and Cavazza (1999).

As there are requirements to incorporate application-related knowledge or application-dependent data into path planning, a secondary heuristic is introduced. We are using this secondary heuristic to incorporate the behaviour of multi-agents and this behaviour is called the collective behaviour. For the second heuristic, the expanded node is no longer the best node in OPEN list, but a node from FOCAL list will be used instead. The best or selected nodes in FOCAL are induced with the application-dependent knowledge or collective behaviour.

While these heuristics can have a geometrical translation (e.g. in term of heading vector) they also depend dynamically on the shape of the path planned for the agent. Using a secondary heuristic, an alternative to the aggregation of multiple heuristics as described in Pearl (1985) can also be seen. It has however the advantage of maintaining a hierarchy between the primary heuristic and the secondary heuristic. This is especially relevant in path planning, where the primary heuristic is distance based (AbdLatiff & Cavazza, 2000).

Synchronous Near-admissible A* (SNA*)

The algorithm of SNA* is given by the pseudocode below:

```
procedure SNA_star;
begin
        get_BEHAVIOUR(focal_A, focal_B, focal_C)
        call A_star(xa, xb, xc)
        get_DIRECTION_for_Agent1(s1, d0)
        get_DIRECTION_for_Agent2(s2, d0)
        get_DIRECTION_for_Agent3(s3, d0)
end
end_of_procedure;
```

get_BEHAVIOUR() is the main function used to set the collective behaviour between agents. Maximum dispersion is a collective behaviour defined as :

$$x_n = \max(\max\_dist(x1, x2), \max\_dist(x1, x3), .., \max\_dist(x1, x_m))$$

where x_n is the next expansion node, for **m** agents. Figure 2 shows the result for three agents taking different routes to the same destination using SNA* algorithm with maximum dispersion behaviour.

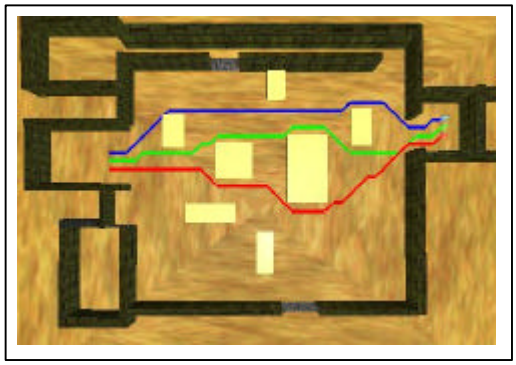


Figure 2. The result of using SNA* algorithm for maximum dispersion behaviour for three agents

RESULT

There are four criteria that control the delineation of paths generated in SNA*, i.e. obstacle density, obstacle distribution, the constrained environment and the FOCAL size. A high density of obstacles will cause the alternative paths to a destination to be difficult to find, whereas a low density of obstacles will not generate any alternative paths. The Virtual Environment that we have used is limited to an open environment where the obstacles are individually located. In this condition the alternative paths are guaranteed to exist and the implementation of SNA* algorithm is assured.

The position of obstacles is another factor that determines the layout of paths, especially the first obstacle encountered by the agents. Early obstacles will make the path generated disjoint and early dispersion will normally determine the pattern of the remaining path. At least one obstacle has to be located in such a position so that any two agents will split from each other. This is shown in Figure 3, where A, B, C and D are obstacles to be avoided.

In a constrained environment, the possibility of agents dispersing at an early stage will effect the dispersion of remaining path. It appears that agents starting from a more open environment have more opportunity for early dispersion and subsequently more dispersion in the rest of the path generated. This is shown in Figure 4 where A is in a room and B is at the door.

All the limitations for the maximum dispersion due to the density, disposition and the closed environment, can be improved with the size of the FOCAL set. The ability to disperse is more when there are more options to select in the next expansion. So with the increase of FOCAL size either the ability to disperse is increased or the dispersion pattern becomes greater. Moreover the percentage should be limited to maintain the property of admissibility. With this property the alternative paths are semantically logical for the agent to follow. Although a higher percentage does not significantly effect the processing speed, the path generated will be to far from optimal.

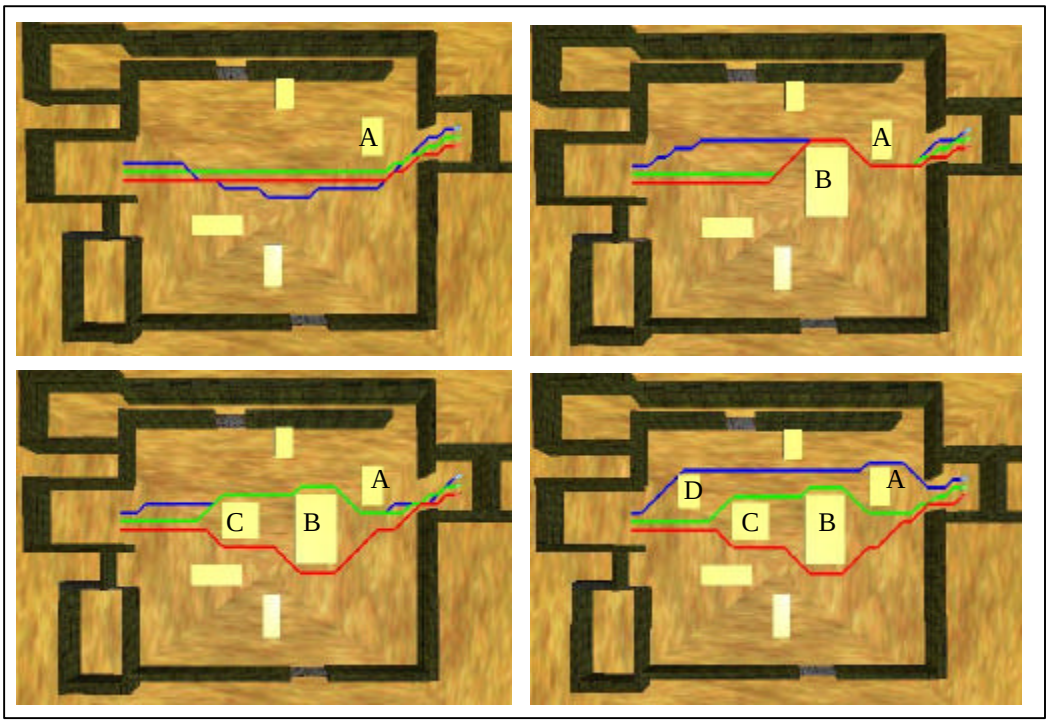


Figure 3: The early obstacle would act as a “splitters”

Figure 4: Open environment has more opportunity for early dispersion

Figure 5 shows the effect of increasing FOCAL size on the previous experiment. In Figure 5(i), the previous experiment is as shown in Figure 3 with 20% of FOCAL size and then increased to 30%. Agent_A and Agent_C have departed 8.8% and 14.3% from its admissibility respectively. Agent_B is assumed to take the shortest path to destination. Figure 5(ii) shows the effect of increasing FOCAL size from 20% (from Figure 4) to 50%. Admissibility comparison is taken from 0%, 30%, 40% and 50% of FOCAL size. The figure shows that the dispersion for multi-agents paths in the constrained environment can be improved by increasing the FOCAL size.

CONCLUSION AND FURTHER WORK

This paper discusses the co-ordination of path planning for multi-agents using the Synchronous Near-admissible A* (SNA*) algorithm. SNA* uses maximum dispersion as the collective behaviour for the co-ordination of multi-agents in path planning. This differs from existing solutions like flocking algorithms, crowd simulation and school behaviour. SNA* provides a path planning solution as well as co-ordination among agents. The path generated using SNA* has direct influence with the admissibility property. The density of obstacles in certain environments will provide alternative paths that do not differ from the optimum path from the start to the destination. However with a higher percentage of FOCAL size, the path generated is able to depart from the admissibility property and path dispersion becomes high. It should be noted that the number of alternative paths does not have to match the number of agents. Splitting a large group of agents into three smaller groups taking different routes can be sufficient for many applications.

There is certainly an upper bound on the number of alternative paths that can be found in a given environment. This would depend on the obstacle density and layout as well as the size set of the FOCAL list. However, as the number of alternative paths increases, some might depart too much from optimality (AbdLatiff & Cavazza, 2000).

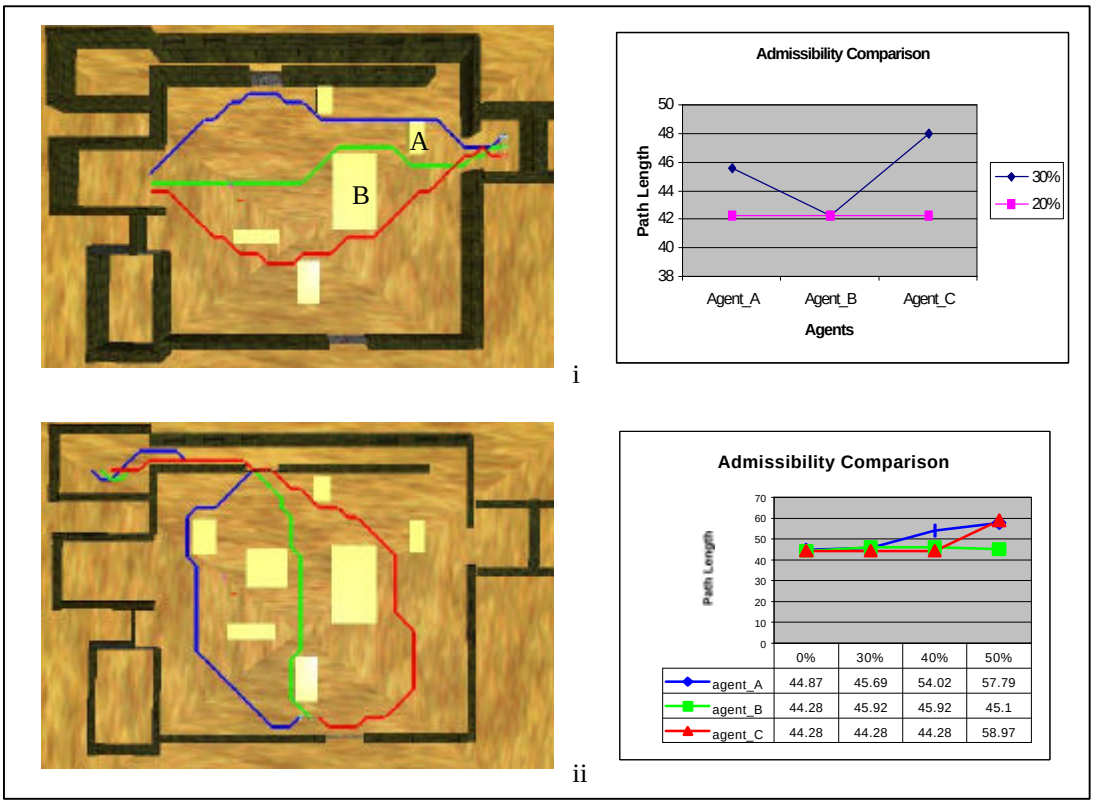


Figure 5: i. Increasing 30% from 20% (Figure 3)
ii. Increasing 50% from 20% (Figure 4).

REFERENCES

Abd Latiff, M.S. and Cavazza, M., 2000. Synchronous Path Planning for Multi-Agent Co-ordination. Proceedings of the VSMM 2000 Conference. Gifu, Japan.

Atlas, R. 1989. Building Design Can Provide Defensible Space. Access Control, September. [Online]. Available from: http://www.cpted-security.com/builddes.html. [accessed July 6, 2001].

Bandi, S. and Cavazza, M., 1999. Integrating World Semantics into Path Planning Heuristics for Virtual Agents. Proceedings of VA'99. Salford, UK.

Barr, A. and Feigenbaum, E.A., 1981. The Handbook of Artificial Intelligence, vol. 1. William Kaufmann.

Bedau, M.A, 1992. Philosophical aspects of artificial life. In Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life, pages 494--503, Paris, France, 1992

Brock, D.L., Montana, D.J. & Ceranowicz, A.Z., 1992. Coordination and Control of Multiple Autonomous Verhicles. Proceedings of the IEEE Conference on Robotic and Automation. Nice, France.

Brogan, D.C., Metoyer, R.A., and Hodgins, J.K. 1997. Dynamically simulated characters in virtual environments. SIGGRAPH'97 Visual Proc.. pp. 216.

Farenc, N., Musse, S.R., Schweiss, E., Kallmann, M., Aune, O., Boulic, R. and Thalmann, D., 2000. A Paradigm for Controlling Virtual Humans in Urban Environment Simulations. Applied Artificial Intelligence. 14, 1, pp. 93-124.

Luger, G.F. & Stubblefield, W.A., 1993. Artificial Inteligence – Structures and strategies for complex problem solving, The Benjamin/Cumming Publishers.

Medjdoub, B. and Yannou , B., 2001. Separating topology and geometry in space planning. Computer-Aided Design. Vol.32. No.1. pp 39-61.

Nilsson, N.J., 1980. Principles of Artificial Intelligence. Tioga Publishing Company.

Noser, H., Pandzic, I.S., Capin, T.K., Magnenat-Thalmann, N., & Thalmann, D. 1996. Playing Games through the Virtual Life Network. ALIFE V, Oral Presentations, May 16-18, Nara, Japan, 114-121

Parker, C. (1999). Boids Pseudocode. [Online] Available from http://www.cse.unsw.edu.au/~conradp/boids/pseudocode.html [Accessed on September 15, 2000]

Pearl, J., 1985. Heuristics. Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley.

Pinhanez, C.S., 1999. Representation and Recognition of Action in Interactive Spaces. PhD Thesis. MIT

Reynolds, C.W., 1987. Flocks, Herds, and Schools: A Distributed Behavioral Model. Computer Graphics, 21(4), July 1987, pp. 25-34.

Reynolds, C.W., 1999. Steering Behaviors for Autonomous Characters. In Conference Proceedings of the 1999 Game Developers Conference, Miller Freeman Game Group, San Francisco, California, pp 763-782.

Schweiss, E. , Musse, S.R., Garat. F., Thalmann, D., 1999, An Architecture to Guide Crowds Using a Rule-Based Behaviour System, Proc. Agents 99.

Thalmann, D., Musse, S.R, Kallmann, M., 2000. From Individual Human Agents to Crowds. , INFORMATIK/ INFORMATIQUE, No1, 2000

Tu, X., 1996. Artificial Animatl for Computer Animation: Biomechanic, Locomotion, Perception and Behaviour. PhD. Thesis. University of Toronto.

A NEW COMPUTATIONAL APPROACH TO THE GAME OF GO

JULIAN CHURCHILL, RICHARD CANT, DAVID AL-DABASS

Department of Computing and Mathematics
The Nottingham Trent University
Nottingham NG1 4BU.
Email: richard.cant/david.al-dabass@ntu.ac.uk

KEYWORDS
Computer Go, Neural Networks, Alpha beta search algorithms.

ABSTRACT

This paper investigates the application of neural network techniques to the creation of a program that can play the game of Go with some degree of success. The combination of soft AI, such as neural networks, and hard AI methods, such as alpha-beta pruned minimax game tree searching, is attempted to assess the usefulness of blending these two different types of artificial intelligence and to investigate how the methods can be combined successfully.

INTRODUCTION

Go is an oriental game that is very popular in China, Japan and Korea in particular, but which has a very large following all around the world [Chikun 97]. It is a relatively simple game, the complexity of which emerges as you become familiar with the ideas presented. A comparison with Chess is often made [Burmeister et al 95], as these are both board-based games of zero-chance. The rules are simpler in Go, however the board is larger and due to the unrestrictive nature the rules tend to, there are many more moves available for the Go player to consider.

The game is played on a board, which has a grid of 19x19 intersections. Two players, black and white, take turns to place a single stone on any unoccupied intersection, with the aim of surrounding as much territory as possible. A player can pass at any turn instead of placing a stone. Capturing the opponent's stones is also used to increase a player's score but is usually a secondary concern compared to securing territory. A stone is captured when the last of its liberties is removed. A liberty is an empty intersection directly next to the stone. Liberties are shared amongst connected stones. Diagonals are ignored when looking at connectivity between points in Go. With this in mind suicide is not allowed unless it is to capture some opponent's stones in which case the suiciding stone remains uncaptured and is left on the board.

The end of the game is usually reached by mutual agreement between the players, when they both pass consecutively. The remaining stones on the board are considered as to whether they would survive further or not. If they are decided to be effectively dead then those points count for the opposing player. The territory is then totalled up and the winner declared.

CURRENT RESEARCH

Neural Networks: To implement a neural network on a computer a model of neurological activity as described above is used. This usually takes the form of two or more layers of connection weight values. These are changed during training using some specific mathematical rules until the network outputs the correct values for the input pattern. A function for calculating the output of a neuron given a set of input values also needs to be specified. This function is referred to as the activation function, since it determines whether a neuron 'activates', that is whether it generates an output signal or not and also determines the strength of the signal based on a set of parameters. This is frequently a simple summing function of the weights multiplied by the input signal along the associated weight.

Figure 1 shows a small but typical multilayer network of the form 3-2-3, 3 input neurons, 2 hidden neurons, and 3 output neurons. Note weights of connections only apply between neurons, so the intial values being fed into the input layer do not have to multiplied by a connection weight. The same applies for the output neurons final output of values. Also note that each neuron is connected to every other neuron in adjoining layers. This is not always done in neural networks, but shall be so for the networks developed in this project.

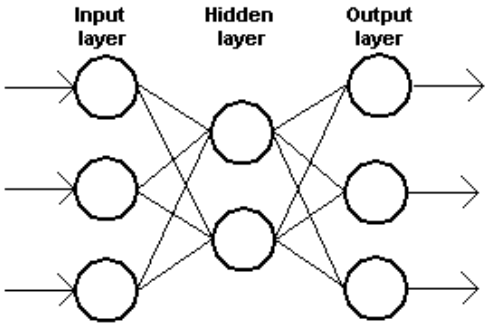


Figure 1 - Small Multilayer Neural Network

A particularly interesting and relevant aspect of neural network techniques is that the neural net developed will usually learn general trends in the relationships between input and output patterns, often these are things that people may not consciously notice themselves. This means the net will frequently be able to handle input that it has not been trained on and will hopefully give useful output. For new classes of input the net may have to

be trained further, but at least this shows that neural nets are flexible, can adapt to changing circumstances and learn from new experience even when installed in its application environment.

State Of The Art: The computer Go programming community is relatively small compared to computer Chess but interest in the topic is building now that computer chess has reached such a high level of success, the AI community are looking for a new holy grail. The central hub of this community can be found at the computer Go mailing list [Computer Go Mailing List 01], where many of the best Go programmers gather to thrash out new ideas, protocols and discuss recent tournaments.

Many Faces Of Go: Many Faces Of Go uses many traditional or hard AI techniques the most important of which are alpha-beta search, pattern matching and rule based expert systems. The main principle is to supply the program with as much knowledge, about how to play Go and what makes a good move, as possible. The alpha-beta search algorithm is used in conjunction with a complex evaluation function to investigate moves where necessary.

NeuroGo: One program that has been developed using neural network techniques is called NeuroGo [Enzenberger 96]. It achieved some amount of success against Many Faces Of Go set to a medium level. Many Faces is currently ranked one of the best in the world. The training target was determined by the Temporal Difference learning algorithm [Schraudolph et al. 94], which has been used with great success in a backgammon neural network program called TD-Gammon [Tesauro 94]. NeuroGo is really a crossover program, which does lean heavily towards soft AI techniques of machine learning, but also incorporates expert knowledge, which must ultimately come from human experience.

A NEW APPROACH

The use of a NN for selecting possible moves should be fast and also allows the opportunity to expand the NNs knowledge in an automated fashion. This is part of the strength of NNs, even when the NN is actually in use it is still possible to teach it where it went wrong and show it how to correct it's behaviour by supplying it with the correct stimuli/reaction combination.

Combination Of Hard And Soft AI Techniques: By attempting to use neural networks with game tree search this project is bringing together hard and soft AI techniques and trying to meld them in a useful and productive fashion. The main purpose behind this is to take the best features of each component and combine them to produce something that performs better than either of the two components separately. In the case of game tree search the advantage of using it comes from its ability to look ahead into the probable results of playing a particular move. The disadvantage is that it can be very resource intensive and inefficient. For example without additional help a standard game tree search may spend as much time looking at what may be obviously bad moves to us, as it does looking at what may be obviously good moves. It has no knowledge, no sense of what makes a good or bad move.

Application To Standard Game Tree Search: The terminal nodes in a game tree are assigned a score based upon some static analysis of the board position at that node and the scores from all the terminal nodes are filtered up the tree toward the root node. Using a technique known as Minimax [Owsnicki-Klewe 99], the best move, the one that maximises the players score can be found, however there are still many problems with this approach. For instance the terminal node board scoring method must be fairly accurate else the minimax values will be irrelevant, and for many games the computational power required to generate a game tree large enough to produce a conclusive result is enormous. For example a game of chess could have around 40

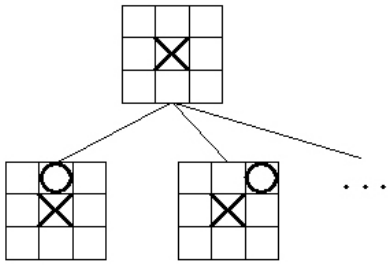


Figure 2 - A Partial Game Tree for Tic-Tac-Toe

moves and at each turn there maybe 10 legal moves available giving 10^{40} nodes. For Go it is much more complex due to the 19x19 board and very few restrictions on moves. With around 220 moves per game and an average of about 180 different moves to play each turn it has been estimated that the size of this particular search space is around 10^{170} [Allis et al. 91]. So an exhaustive search is completely out of the question, even a relatively shallow game tree search, 6 ply, could easily be beyond the resources of the host computer.

SOFTWARE/HARDWARE DEVELOPMENT

System Specification: A generic neural network program was developed to create and train neural nets. The generic application was expanded and applied to Go, in such a way as to be able to create and train a neural network to suggest plausible moves, using the current board position as input for the network. A user interface was provided for both the neural network module and the Go playing part of the overall system. A graphical interface is useful for displaying board positions and for showing what moves the system maybe considering.

Board Representation: A few definitions first: a **string** is a connected line of stones (north, south east, west only, no diagonals), a **liberty** is an empty point next to the string and can be thought of as breathing space for that particular string. According to the rules if a string has no liberties all the stones in the string are **captured** and removed from the board.

The board is represented by a special 'Board' object that contains a two dimensional array, one value for each point on the board; 0 for empty, 1 for white and 2 for black. An array of 'GoString' objects is also stored that represent each string of stones currently on the board containing details such as which points belong to it, the colour of the string and the number of liberties the string has. This information is updated incrementally; everytime a stone is added or removed from the board. Other arrays are present to represent special markers that may be of use to show the neural networks responses at various points.

Graphical User Interface: A simple but functional GUI must be a part of this project since Go is a visual game and it would be very difficult to judge the performance of the neural network. The GUI would ideally allow a human to play a game of Go against the program thus enabling those people with some knowledge of the game to test it's weaknesses and strengths. Also the user should be able to view the program playing against another program and to observe the neural networks responses during a game or whilst playing through an expert game so that the success of the program can be evaluated.

As can be seen from the screenshot below the interface provides menus to access all of the available functions of the program. A toolbar is present with for greyed out buttons. These currently allow the user to step forward or backwards through moves in an SGF game, to indicate a pass move during a game of Go and to abort the current action by using the round button at the end. A log window is provided to output text messages to the user. From this image the board can also be seen, with coordinate system along the edges. When necessary the user can select points on the board, for instance during a game to specify a move.

Design Issues: The initial architecture prposed was a 3-layer network consisting of 25 neurons for the input layer, 10 for the hidden layer and 1 for the output layer. The 25 neurons of the input layer would map directly onto a 5 by 5 section of a Go board. The idea was to train a network so that all the legal moves possible in a particular board position could be fed into the network one at a time, incorporating a 5 by 5 area of board around the potential move and that the neural net would output a plausibility value for the given move. The move with the highest plausibility would then be chosen, or better still, several of the highest plausible moves would be taken and fed into an alpha-beta minimax search to investigate the moves impact on the future of the game.

The back propagation algorithm [Callan 99] for training chosen, mostly because of it's general applicability to a wide range of problems and also it would be simple to implement and use.

Training data was acquired as a collection of professional tournament games in SGF format from the Internet [Van Der Steen 01]. A training database generator then dissected these files into input/output pairs that could be fed directly into the neural network training algorithm. For each move the board position was analysed to produce several input/output training pairs. For every legal move in the board position an input matrix was generated that represented an n-by-n area centred on the move. It was determined whether the move was played up to 6 moves in the future and if so an output value was associated with it that started at 1.0 for the move occurring next turn, down to 0.2 for it occurring 6 moves in the future. If the move did not occur within 6 moves it was relegated to a set of unlikely moves and an output of 0 was connected to it. At the end 6 unlikely moves were randomly chosen and stored in the database with the 6 likely moves to balance the networks training.

Figure 4 shows the game walkthrough test in progress, which allows the user to play through a game of Go that has been stored in SGF format. The program itself makes suggestions and comparisons are made between the program and the actual moves made. This test is most useful is professional quality SGF games are used. The green circle shows the most recent move, the blue circles highlight the programs top ten suggested moves and each point on the board has a small coloured square. This indicates the plausibility score given to a move at that point in the current situation by the neural network. The redder the square the lower the score and the high end goes to blue. Lots of log information is also outputted in the window on the right, showing the actual scores associated with the best and the worst suggested moves. Also the rank of the actual move is outputted, in this case being 52nd out of 314 legal moves.

IMPLEMENTATION

Smart Game Format files can be read and interpreted allowing training databases to be compiled from SGF files. However SGF files cannot be saved and only the main line of play can be read, not any saved alternative lines of play. Many testing functions have been written to maintain consistency throughout development of the program and to allow progress to be measured. The code to create and maintain GoString information has been written and stores which stones belong to each string, what

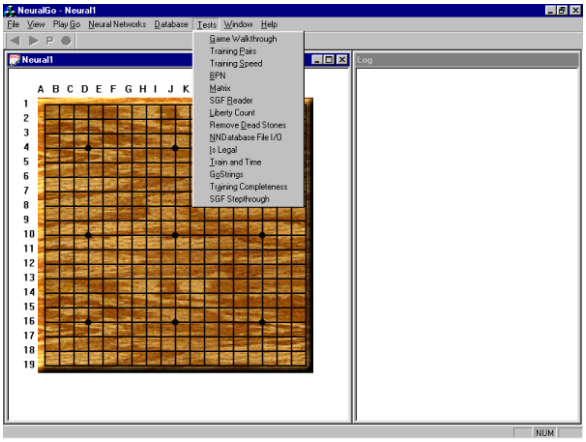


Figure 3 - NeuralGo GUI

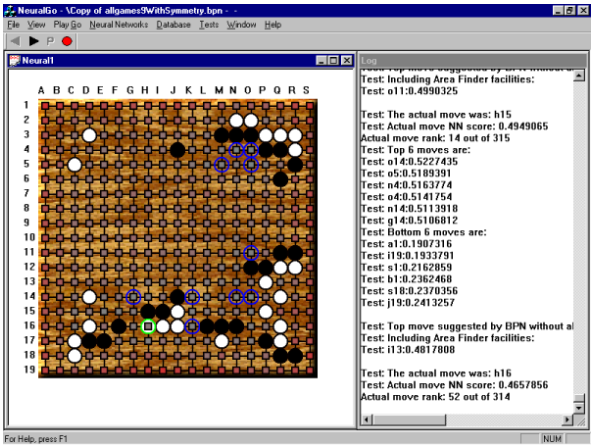


Figure 4 - Game walkthrough in progress

colour the string is and how many liberties the string has. This greatly increases the speed when detecting and removing captured groups and could provide useful information for additional tactically oriented modules. A graphic board display

is available to easily view the program playing through an SGF and suggesting moves to compare to the actual move. The GUI is also intended to be used to watch the program play against another program using the GMP interface, however the GMP protocol code has not ported well from the Java version The user may also play through the GUI against the neural network.

An Area Finder network was trained for a reasonable amount of time and started to show some signs of sensible suggestions, certainly enough to warrant further investigation to ascertain how worthwhile it would be to use it in conjunction with a standard network of the sort just discussed.

Two restricted move range networks were trained, both showing promising results. The first covered the first five ply moves and second covers from five ply to ten ply. The MTD (f) variation of the alpha-beta minimax search algorithm was implemented along with various enhancements. The program also uses the best move from previous iterations as the first node to expand as this has been shown to give a considerable performance boost [Schaeffer et al.].

Enhanced transposition cutoffs [Schaeffer et al.] are also implemented which means that all nodes arising from a position are first quickly checked to see if they cause a cutoff in the tree before deep searching that branch.

The expand function which creates child nodes given a position uses a neural network to suggest a specified number of the most plausible moves as child nodes. Also two alternative evaluation functions to score the nodes are available, one that counts each sides liberties and attempts to maximise one sides liberties whilst minimising the opponents. The second one simply counts the number of stones on each side and so encourages capturing and aggressive play.

RESULTS & DISCUSSIONS

Looking at how long it takes various configurations to reach a move decision shows us some important points and in combination with some quality of result analysis can tell us to what degree the combination of soft and hard AI has been successful and if it is worth pursuing in the future.

First of all the difference between configuration 1, see Table-2 which used just a 9x9 network and configuration 2 which used a 9x9 network with an Area Finder network is easily explainable. Using the coarser grained Area Finder network divides the board into 9 sectors and given the full 19x19 board selects the most appropriate sector out of the 9. Then the 9x9 network looks at all legal moves within that sector only. For the first configuration all legal moves in the entire 19x19 board must be considered so we see a logical time difference of around a factor of 9. A similar affect is seen when comparing configurations 4 and 5. The use of the Area Finder network gives a substantial speed boost without adding any unreasonable overheads. If the quality of suggestions presented by the Area Finder network can be measured and built upon then this could be an effective and efficient method of incorporating neural network technology within a Go playing program.

The use of alpha-beta search added considerable computational overheads, however that is expected considering the nature of the algorithm. The liberty count evaluation function was used in all cases. To assess and compare the quality of the neural networks and different configurations involving either or both soft and hard AI two approaches were taken.

Configuration	Average Time Taken Per Move (seconds)
1. 9x9 Neural Network	0.784
2. 9x9 Neural Network + Area Finder	0.136
3. Alpha-Beta (Liberty Count)	41
4. 9x9 Neural Network + Alpha-Beta (Liberty Count)	9
5. 9x9 Neural Network + Alpha-Beta (Liberty Count) +Area Finder	1.56

Table 1

First, the configurations used for timing an average move were used to play proper games of Go against GNUGo 26b [GNUGo 01]. It is important to play actual games since this was the original intention of creating such a program and is really the best way of judging its success in its intended environment. The program itself still unfortunately has a few problems and bugs that were given special provision. The program did not have any method to decide when to pass, so a game would continue until GNUGo passed or until a crash occurred. Where a crash occurred it is marked in the results table as N.C. (not completed). When a game has reached an end, either on purpose or by fault, the board was scored by Jago [Grothmann 01], which functioned as arbiter between the programs.

Neural Network	Average Time Per Move	Average Move Rank	Percentage of time that actual move is in top n percent				
			10	20	30	40	50
5x5	0.576	105	19	36	50	59	65
7x7	0.736	115	17	30	42	60	66
9x9	0.936	114	12	31	39	53	64
Copy of 9x9	0.912	104	18	34	47	56	69
11x11	1.256	161	10	19	30	35	44
13x13	1.664	124	18	28	36	51	62
Random 9x9	0.824	156	5	21	29	35	45

Table 2

The second method was used to determine the extent and level of training achieved by the neural networks. Several measures were used and information gathered about 6 different networks. The statistics were collected as each network played through the same professional game; the average time to select a move was recorded, as was the average rank of the actual move within all the moves considered by the neural networks. To give a more detailed look at the quality of the moves being selected the

percentage of time that the actual move was ranked in certain percentiles was also noted, see Table 2, for example for the 5x5 network the actual move was ranked in the top 10% of moves 19.2% of the time and was tanked in the 20% of moves 36% of the time. For the sake of comparison a newly created, hence untrained, network was tested also, so we should expect, if training has worked at all, that the new network should have the lowest scores.

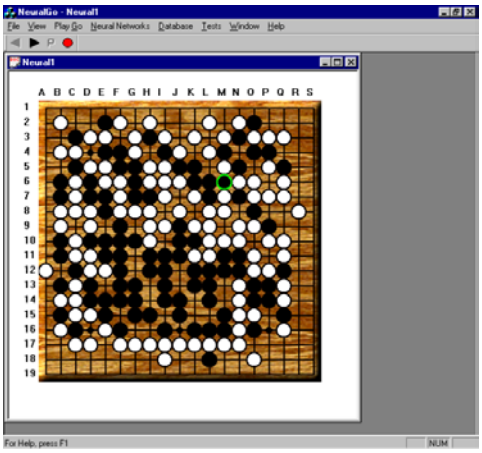


Figure 5 - Game in progress

Figure 5 shows a game in progress against GNUGo. GNUGo is playing white and the program is playing black. The move marked with a green circle is the most recent move. Comparing all of these figures to the random 9x9 network shows that they all improved after training and reveals an interesting and very important point about over training. The 'copy of 9x9' network was an earlier version of the current 9x9 and has much better figures. This does suggest rather strongly that the current 9x9 has been over trained and the quality of its output has been degraded as a result. This also implies that some sort of peak of training can be reached and by considering the figures they also suggest that the peak does not mean getting the very best move at rank number one.

Configuration	Game Score (we play black) (J = Japanese, C = Chinese)
1. 9x9 Neural Network	J: B-8, W-49 C: B-106, W-146
2. 9x9 Neural Network + Area Finder	J: B-3, W-21 C: B-54, W-71
3. Alpha-Beta (Liberty Count)	J: B-9, W-12 C: B-35, W-38 * Not Complete
4. 9x9 Neural Network + Alpha-Beta (Liberty Count)	J: B-9, W-25 C: B-107, W-124
5. 9x9 Neural Network + Alpha-Beta (Liberty Count) +Area Finder	J: B-7, W-18 C: B-55, W-66

Table-3

Rather it suggests, perhaps viewing it optimistically, that the network realises there may not be one perfect move but maybe lots and using a neural network allows the moves to be ranked effectively as opposed to selecting a single best move. This may be the strength of using neural networks in this instance.

If we now move onto the results of the test games against GNUGo we can observe several things from the scores. Both Chinese and Japanese scores are presented, with our program playing black for each game.

All these results show that soft AI has something to offer the problem area of Go, the limitations and extent of which require further investigation, however we have made some connections, various ideas have been tried and tested and a system that can support further research has been developed and implemented. More than anything else perhaps, questions have been raised and pointers for promising future investigations have been found.

There could be any number of ways to combine soft and hard AI. The trick is to do it in such a way as to maximise the strengths of each and minimise the weaknesses. If in doing so the combination is greater than the parts then the job is a success. From the results it seems clear that hard I benefits soft AI and it is a pity the reverse cannot be concretely induced from the results collected but it would be reasonable to assume so.

CONCLUSIONS AND FUTURE WORK

Much of the research carried out over the past few months only touches upon each idea, any one of which could provide a lengthy and fruitful line of research. Amongst these are combining various grain networks, how to combine them and which to use, limited range networks which tend to specialisation and of course other methods of combining soft and hard AI. There are other soft AI techniques apart from neural networks that may be worth looking at such as genetic algorithms and evolutionary programming.

Interesting extensions to the ideas could include a closer look at the actual choice of neural network architecture and construction, and a more thorough review of the choices available. A deeper understanding of the way humans perceive and play Go could be developed from further work, perhaps leading to a better understanding of human pattern recognition.

REFERENCES

1. [Allis et al. 91] Which Games Will Survive?, ALLIS, L.V., VAN DER HERIK, H.J., HERSCHBERG, I.S., Heuristic Programming in Artificial Intelligence 2 - The Second Computer Olympiad, pages 232 - 243. Ellis Horwood, 1991.
2. [BGA 99] British Go Association (BGA), 1999, Available on the Internet at http://www.britgo.org/
3. [Burmeister and et al. 95] An Introduction to the Computer Go Field Associated Internet Resources, BURMEISTER, J., WILES, J., 1995, Available on the Internet at http://www2.psy.uq.edu.au/~jay/go/CS-TR339.html
4. [Callan 99] The Essence Of Neural Networks, CALLAN, R., Prentice Hall, 1999
5. [Chikun 97] Go: A Complete Introduction To The Game, CHIKUN, C., Kiseido Publishing Company, 1997
6. [Computer Go Ladder 01] Computer Go Ladder, 2001, See http://www.cgl.ucsf.edu/go/ladder.html
7. [Computer Go Mailing List 01] Computer Go Mailing List, 2001, See

http://www.cs.uoregon.edu/~richard/computer-go/index.html
8. [Enzenberger 96] The Integration of a Priori of Knowledge into a Go Playing Neural Network, ENZENBERGER, M., 1996, Available on the Internet at http://www.uni-muenchen.de
9. [Fotland 93] Knowledge Representation In The Many Faces Of Go, FOTLAND, D., 1993.
10. [GNUGo 01] GNU Go latest version can be found at http://freedom.sarang.net/software/gnugo/beta.html
11. [Grothmann 01] Jago, GROTHMANN, R., Available on the Internet at http://mathsrv.ku-eichstaett.de/MGF/homes/grothmann
12. [GTP 01] Go Text Protocol information can be found 2/3 down the page at http://freedom.sarang.net/software/gnugo/beta.html
13. [Hollosi 99] SGF User Guide, HOLLOSI, A., 1999, Available on the Internet at http://www.red-bean.com/sgf/user_guide/index.html
14. [Huima 99] A Group-Theoretic Hash Function, HUIMA, A., 1999, Available on the Internet at http://people.ssh.fi/huima/compgo/zobrist/index.html
15. [Muller 97] Playing It Safe: Recognizing Secure Territories in Computer Go by Using Static Rules and Search, MULLER, M., 1997, Available on the Internet at http://www.cs.ualberta.ca/~mmueller/publications.html
16. [Plaat 97] MTD(f), A Minimax Algorithm Faster than NegaScout, PLAAT, A., 1997, Available on the Internet at http://www.cs.vu.nl/~aske/mtdf.html
17. [Owsnicki-Klewe Search Algorithms, OWSNICKI-KLEWE, B., 1999] Available on the Internet at http://www.informatik.fh-hamburg.de/~owsnicki/search.html
18. [Tesauro 94] TD-Gammon, a self-teaching backgammon program, achieves master level play, TESAURO, G., Neural Computation, Vol. 6, No.2, 1994.
19. [Schaeffer et al.] New Advances In Alpha-Beta Searching, SCHAEFFER, J., PLAAT, A.
20. [Schraudolph Temporal Difference Learning of Position Evaluation in the et al. 94] Game of Go, SCHRAUDOLPH, N., DAYAN, P., SEJNOWSKI, T., Neural Information Processing Systems 6, Morgan Kaufmann, 1994, Available on the Internet at ftp://bsdserver.ucsf.edu/Go/comp/td-go.ps.Z
21. [Van Der Steen 01] Go Game Gallery, VAN DER STEEN, J., 2001, Available on the Internet at http://www.cwi.nl/~jansteen/go/index.html.
22. [Wilcox et al.] The Standard Go Modem Protocol – Revision 1.0, WILCOX, B., Available on the Internet at http://www.britgo.org/

A LEARNING ARCHITECTURE FOR THE GAME OF GO

A.B. Meijer and H. Koppelaar
Delft University of Technology. Faculty ITS. Section Mediamatics.
Mekelweg 4, P.O.Box 356, 2600 AJ, Delft, The Netherlands.
{a.b.meijer, h.koppelaar}@its.tudelft.nl

ABSTRACT

In this paper, a three-component architecture of a learning environment for *Go* is sketched, which can be applied to any two-player, deterministic, full information, partizan, combinatorial game. The architecture called HUGO has natural and human-like reasoning components. Its most abstract component deals with the selection of subgames of *Go*. The second component is concerned with initiative. The notion of *gote no sente* (a move that loses initiative but creates new lines of play that will hold initiative) is formalized. In the third component, game values are computed with a new kind of α-β algorithm based on fuzzy, partial ordering. Our approach leaves some valuable control parameters and offers ways to apply further machine learning techniques.

KEYWORDS Combinatorial Games, Uncertainty, Initiative, Fuzzy Partial Ordering, Game of Go

INTRODUCTION

Two-player, deterministic, complete, information, partizan, combinatorial games form a family of games which has received a lot of attention from the AI community over the last decennia. Altough much progress has been made, resulting in some well-playing chess programs for example, almost all modern programs for these games lack (well-formalized) human-like behavior in general and the notion of initiative in particular. This need not result in a poor performance, rather it depends on the domain. We hypothesize that the notion of initiative is mandatory for writing a good program for the most notorious of combinatorial games: *Go*. It's folklore is full of terminology concerning initiative and professionals play the game at an abstract level of initiative, far beyond considering just move sequences, as in the commonly used α-β algorithm (or any other minimax-based search algorithm). Furthermore, the α-β algorithm is based on some absolute scalar-valued evaluation function, whereas humans can be said to use an ordering function to compare two board situations directly with one another. For example, a professional *Go* player could reason that one position is slightly better than another when his walls look a bit thicker (=stronger).

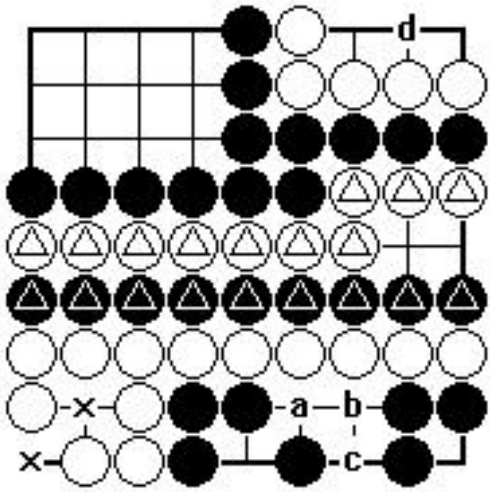


Figure 1: An endgame situation in 9×9 *Go*.

This paper is organized as follows. In the next two sections we will give a short introduction to the game of *Go* and combinatorial game theory. Then, we formalize these two human-like concepts that arise in the learning environment of *Go* and embed them in an architecture called HUGO. The following three sections further explain the three components of HUGO. The first component deals with the selection of subgames of *Go*, the second with initiative and the third with the computation of game values. We end with some concluding remarks and future work. Go terminology is written in

SMALL CAPS.

THE GAME OF *GO*

Go is an ancient game, originated in China about 4000 years ago. It has influenced oriental warfare, which shows off in the shape of the Great Walls of China and, very recently in Afghanistan, in the preference for semi-fixed frontlines between the opposing factions, which only change hands if there is a broad momentum in favour of one of the sides. Compare this with *Go*, where two players have to embark territory by alternately placing a stone on a grid, gradually building strongholds and eventually walls that completely surround one's territory. Strongholds will only be given up if the opponent has created enough influence (*Go* term for momentum) to walk over it.
The rules of *Go* are very simple in principle (but in finesse they can vary a lot over different rule sets like the Chinese, Japanese or mathematical *Go* rules). The *capturing rule* is the most important, stating that a string of stones gets captured if all of its neighboring intersections are occupied by enemy stones. This rule implies that the two x's in figure 1 are suicide, which implies in turn that the white group that surrounds them cannot be captured (*Go* terminology: the group LIVES). The white group can only be captured if White would cooperate foolishly and plays on one of the x's himself. Black is then allowed to play the "temporary suicide" of the other x, because this would capture the entire white group and the suicide is resolved.
The goal of the game is to make living groups that surround more territory than your opponent.
If in the game of figure 1 Black were to play, he would have two good options. The first is to play at d, killing the white group since it has become impossible for White to construct *two* suicide points (EYES) like the two x's. The second option is to save his own group in the bottom right by playing at a. This would result in two black strings, each having one eye. The strings can always be connected with White b, Black c or vice versa. The resulting group lives with two eyes. The best of the two is a, since there are more stones in this group.

COMBINATORIAL GAME THEORY

This section is a very short introduction to Combinatorial Game Theory, loosely following (Cazenave, 1996). It is a mathematical theory for games and numbers, developed by J.H. Conway (Conway, 1976) and adapted to many games by Berlekamp, Conway and Guy (Berlekamp et alii, 1982).

Definition 1 *A combinatorial game $G = \{F|O\}$ is composed of two sets F and O of combinatorial games. Every combinatorial game is constructed this way.*

In games, F should be seen as the set of options (board positions) that player *Friend* can reach with one legal move. O can be looked at as the options for player *Opponent*. F can have two possible values, W (win for Friend) or L (loss for Friend, so win for Opponent). If Friend has a legal move that ensures a win for the whole game, then the value of F is W. This gives four possible outcomes for a combinatorial game: WW, WL, LL and LW (We will use both WW and W|W as abbreviated notations for {W|W}). The left half of a game value is the maximum result that Friend can obtain, the right half is Opponent's best result.
WW denotes a game that is won by Friend, irrespective of who moves first (both player can at best move the game to W = win for Friend). A *Go* example is the white group in the bottom left of figure 1, which lives unconditionally.
WL is an unsettled game, it is won by the player who moves first. An example is the life status of the white group in the upper right corner. If White moves first he can play at d, resulting in a living shape (its territory contains two eyes, intersections that are suicide for Black).
LL is a lost game for Friend, so a sure win for Opponent, even if Friend moves first. Trying to kill the black group in the upper left corner is a lost game for White (given correct counterplay from Black)
LW is a somewhat strange equilibrium situation where the player who moves first will lose the game. An example of this situation is known as SEKI. The triangled stones in figure 1 form a SEKI: either player who wants to capture the opponent string of triangled stones and plays at one of the two shared intersections will immediately be captured himself.
To evaluate a subgame, we define

$$Value(W) = 1, \quad Value(L) = -1.$$

In order to evaluate the global position, each subgame is assigned a numerical importance. A (linear) combination of the value of the subgames yields an indication of the balance of power. The precise nature of combination is a task for component 1.
Now it is possible to define the value of a move:

$$Value\,(move) = Value\,(F) - Value\,(O)\,.$$

$Value(move)$ is the value of the move that achieves state F and thus prevents Opponent from achieving state O. The value of a SEKI needs special attention, since usually there will be no play inside a seki (the value for such a move is -2 times the numerical importance). Its value depends on the rules. In *Chinese* counting one counts one point for all the stones that live in a SEKI but no point for territory, which is natural since no player controls the in-between territory.
Another commonly used term for game value is temperature, which corresponds to the size of the largest play.

As the game progresses, the temperature tends to drop. *Local* temperature corresponds to the size of the largest play in a region of the board. *Ambient* temperature is the temperature of the game besides the local region.
In theory, the value of a game has only two values, W or L. However, it is often intractable to compute the precise game value. Cazenave therefore extended Conway's theory to uncertain outcomes (Cazenave 1996). He introduced a variable U, denoting an uncertain game value with range [-1, 1]. If halfway a game the position is considered roughly equal for both players, then it makes sense to assign this uncertain position the value 0.
U can be seen as a control parameter along the risky-safe axis, since one is free to define the value of U. A low value for U would result in conservative play, a high value models a form of risky play.

THE HUGO ARCHITECTURE

As stated in the introduction, it is our aim to model human concepts that exist in the folklore of *Go*. We have identified three components that are inadmissable in our approach. The first component is to select relevant subgames and their numerical importances. The second is an initiative engine and the third computes game values. We have called the resulting architecture HUGO (human *Go*), see figure 2.
HUGO's input is a board position and some constraints, for example a time or memory limit. This input goes to component 1, which outputs a collection of games. This is the input of the initiative engine, which calls the third component to calculate the value of every single game. Knowing all the game values, the initiative engine tries to find a move that both scores some points and holds initiative.

COMPONENT 1: CHOICE OF SUBGAMES

The task of this component is to select a collection of well-defined subgames (or simply: games) of *Go* which form the basis for further computation. A good choice of games should have a high discriminative ability: the player that wins most of the important games should also win the whole game and vice versa. The problem of such an approach is that the subgames are not independent and thus the value of the whole game is not simply the sum of the value of the subgames (Berlekamp et alii 1982). However, advanced human *Go* players constantly use a variety of games, such as life or death, connection of stones, territory and more. They do this as a means to obtain overview in a chaotic board situation and the result is some sort of a mental model of the state of the game. For each separate game, they can quite easily find the (sub-)optimal move. The difficult part is to take the interactions among games into consideration and to find the optimal move for the whole game. Playing *Go* at this abstract level means finding conflicting interactions all the time and then resolving or exploiting them.

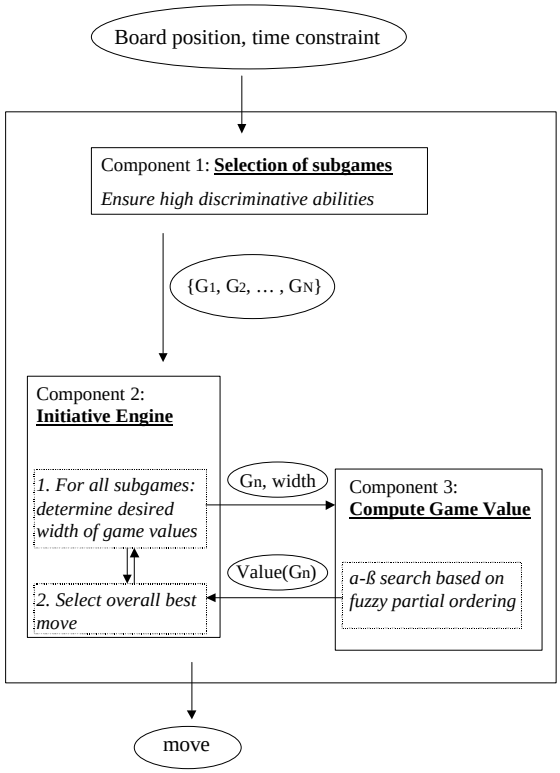


Figure 2: The HUGO architecture

A good collection of games to start with are (1) life and death for strings of stones and (2) connection of strings (the fundamental importance of these games follows from the importance of the capturing rule). These two families of games govern tactical play, but are also inadmissable in strategic play.
One can think of many more games and there is indeed a lot of work to be done, before this can all be automated. However, one could also leave this task to an expert. Introducing more families of games would enhance more sophisticated play, but at the cost of a higher computational effort.
One further sophistication is to define connections between groups (clusters of strings that are highly interconnected) instead of strings of stones. A clustering algorithm would be valuable for this task.

COMPONENT 2: INITIATIVE ENGINE

Given a collection of well-defined subgames from component 1, the task of the initiative engine is to find the move that yields the most points, preferably while hold-

ing initiative. When one has the initiative, one dictates the course of the game and one can choose to accumulate small gains from different subgames.

The first step of this component is to compute the value of each game. A sophisticated game value should not only reveal the player how will win a particular game with which move, but should also tell what are threatening moves in the game. A simple approach to meet this criterium is to compute the game value for the two cases that each player once moves first and once moves second (this is more or less common practice in the more sophisticated current *Go* programs). If one wishes to model threats, s/he must also compute game values for two cases where each player gets to play twice before the opponent may respond.

Let us call the number of plays by one player before the opponent answers the *width* of a game value. One possible outcome for a game value with width two is WL|LL. This is the situation that Friend can move to an unsettled situation (WL) which he can win if he gets the chance to play a second move (only when Opponent ignores the first move), whereas Opponent can move to a won game (LL) if he does answer. Cazenave already derived that this type of game has a threat for Friend only (Cazenave 1996). In fact, the general representation of a one-sided threat is W*UUU* for Friend and *UUU*L for Opponent. W*UU*L is a two-sided threat.

If Friend wishes to play KIKU (play to hold initiative) he can choose to play a series of threatening moves. Although Opponent simply can answer all the threats and win in all the threatened games, this line of play can be advantageous to Friend because a move generally plays in more than one game at the time (multipurpose moves). If a move is a threat in two games at the same time, it will be difficult (if possible at all) for Opponent to find a move that resolves both threats at the same time. The simplest line of follow up play is to answer the most important threat, but then Friend can win the other game.

Sometimes there is such a big move that one does not care about holding initiative but just grabs the points associated with the move (for example killing a large group). In general, this should be done when the local temperature is hotter than the ambient temperature. Such moves usually end in GOTE (loss of initiative) and the resulting game is cold (not much to gain left for either side). In general, GOTE play lowers the local temperature, while SENTE play raises it.

RYO-SENTE (two-sided SENTE) arises when a move is a two-sided threat. If it arises in important games it is usually played immediately.

Despite losing initiative, GOTE moves can develop new initiative. If a move is gote in one game but creates new sente moves in another game, then this move is called GOTE NO SENTE (GOTE with SENTE potential). For example, WLLL|LLLL is a lost game for Friend, but he can change the game to a (lost) game where he does have a threat (WLLL), whereas Opponent can move to a game which he has won and leaves no threat (LLLL). So, an example of a GOTE NO SENTE move is one that is GOTE in one game and moves another game from WLLL|LLLL to WLLL. GOTE NO SENTE moves can only be found if one considers the possible outcomes if a player moves three times in a row before his opponent starts to answer (width = 3). None of the current *Go* programs described in literature does this. Cazenave uses game values up to width 2 (but gave a formalization for greater width).

Computing up to width K is twice the cost of computing up to width *K-1*. Therefore, one has to know when to do so. This task should be dealt with by component 2. One good possibility is iteratively widening based on game value feedback from component 3, since it is known from (Berlekamp et alii, 1982) that one best makes moves in hot (unsettled) games. So these are the games whose value one would like to know most precise.

All the current *commercial* programs lack a well-defined notion of initiative and their advantage in playing strength can be expected to shrink relative to programs based on scientific research, which do implement Combinatorial Game Theory.

COMPONENT 3: COMPUTING GAME VALUES

The task of this component is to compute the game-theoretic value of a particular game. Once determined to what width an outcome has to be calculated, one actually has to calculate the value of the game and the move(s) that accomplishes this value. In fact, in order to do so, one has to calculate the value for all the moves and select the best one. However, at this point it hasn't yet been determined in which game is to be played. It is better not to remember just the best move but all (good) options, because this leaves the possibility open to detect multipurpose moves. Such a move might be suboptimal in all the games it plays in, but can be superior in the combined game compared to any of the optimal moves in the separate games. Furthermore, remembering all good options can offer a means to prevent unnecessary recalculations. Imagine you would only remember the best option (and also only the best reply) and your opponent plays in a later stadium a suboptimal move in the nearby region, threatening the result of your best move. You then would need to re-evaluate this move, whereas you wouldn't need to, had you remembered opponent's suboptimal moves too.

So the task in this component is to calculate the value of a game, given some width. An α-β algorithm (or other forms of minimax search) can be used to do this, but this requires some sort of scalar-valued evaluation function to compare two board positions. This indirect comparison has several drawbacks, including the horizon effect. Quiescence search has been invented to circumvent this

problem, but does not solve all the problems. This and other disadvantages of using absolute evaluation functions are well described in (Müller 2000).
Müller concludes that in many cases partial ordering is better than absolute evaluation. This makes sense, since it is more natural and even in α-β search absolute evaluation is used for partial ordering in the end.
We hypothesize that α-β search in its current form cannot handle the multidisciplinary and strategic nature of *Go*.
Müller's alternative was a method that combines partial order evaluation with minimax search, called *Partial Order Bounding*. In this method one categorizes all the possible states into a success set or a failure set. Minimax search is performed to determine which move guarantees that any leaf node of this move's subtree belongs to the success set. Leaf nodes are evaluated just by checking whether or not it falls into the success set, so the minimax values are boolean.
In contrast to this, our approach does not directly use a success set, rather we bring partial ordering right into the α-β algorithm itself, so we use no boolean-valued evaluation function. To be more precise, we will use fuzzy partial ordering of vector states. Obviously, this generalization is not possible without considering all consequences of introducing fuzzyness and partial ordering at the same time.
In short, fuzzyness can be dealt with by putting states that are approximately equal into one cluster. If one watches out that intra-cluster distances remain (far) bigger than inter-cluster distances, it is possible to order clusters just as if one were ordering single board situations.
The partial ordering function compares the game value of two (clusters of) feature vectors $\mathbf{a}$ and $\mathbf{b}$ and outputs an ordering such as $\mathbf{a} \succ \mathbf{b}$ ($\mathbf{a}$ is preferred over $\mathbf{b}$), $\mathbf{a} \succ\succ \mathbf{b}$ ($\mathbf{a}$ is by far preferred over $\mathbf{b}$) $\mathbf{a} \approx \mathbf{b}$ (about equal), $\mathbf{a} \sim \mathbf{b}$ ($\mathbf{a}$ and $\mathbf{b}$ are incomparable), combinations of these like $\mathbf{a} \succsim \mathbf{b}$ ($\mathbf{a}$ is preferred over or approximately equal to $\mathbf{b}$) and negations. Notice that incomparable is different from approximate equality. Two states that have approximately the same value are incomparable if one is far hotter (more unstable) than the other.
A traditional α-β algorithm (without speed enhancements) remembers during search just the values α and β. In comparison, our generalized algorithm has to remind a partially ordered tree of fuzzy clusters of already evaluated options. We believe that the considerable extra amount of work (firm theory plus implementation) is worthwhile, since it offers a natural (=human) look at *Go*. For example, it enables an ordering between two moves based on the achievements relative to some shared parent state, which is really different (and probably easier) than any scalar-valued board evaluation technique. The details of this approach will be discussed in a forthcoming paper, due to shortage of space here. One issue that will be discussed in great detail and with mathematical rigour is omitted here, namely the fact that game values are looked at as scalars in the above discussion, whereas they are more like an interval [O,F], where O is an underbound for the real value (the best result for Opponent) and F is an upperbound (Friend best result). The real value is determined as soon as one of the two players chooses to play in that game. Of course, the discussion of this section still holds for small game value intervals (cold games).

CONCLUSIONS AND FUTURE WORK

This paper sketched a human-like learning architecture for the game of *Go*, called HUGO, but it uses no game specific knowledge, so it should be possible to apply it to any two-player, full information, deterministic, combinatorial game. HUGO has three major components. The first component deals with the definition and choice of subgames. The second component is concerned with initiative. In the third component, game values are computed with a generalized α-β algorithm based on fuzzy, partial ordering.
The notion of GOTE NO SENTE (a move that loses initiative but creates new lines of play that will hold initiative) is formalized for the first time.
There are some points where one can apply further machine learning techniques, for instance a clustering algorithm in the extended α-β algorithm. Furthermore, each component has some valuable control parameters. The iterative widening in component 2 and possible iterative deepening in component 3 could further enhance real-time behaviour. The value of uncertainty can be used to control the style of play along a safe-risky axis. Currently, work is carried out to mathematically formalize and implement the α-β algorithm based on fuzzy partial ordering, having combinatorial game values with uncertainty as output.

REFERENCES

Berlekamp, E.; J.H. Conway; and R.K. Guy. 1982. *Winning Ways (for your mathematical plays).* Academic Press, New York.

Cazenave T. 1996. *Systeme d'Apprentissage par Auto-Observation. Application au Jeu de Go.* PhD thesis, Université Pierre et Marie Curie, Paris.

Conway J.H. 1976. *On Numbers and Games.* Academic Press, New York.

Müller M. 2000. "Partial Order Bounding: A new Approach to Evaluation in Game Tree Search." Technical Report of ETL, TR-00-10. To appear in a special issue on heuristic search of the Artificial Intelligence Journal.

GAMES PLATFORMS

PALM GAME DESIGN

Pieter Spronck
Universiteit Maastricht IKAT/Infonomics
P.O. Box 616
NL-6200 MD Maastricht, The Netherlands
E-mail: p.spronck@cs.unimaas.nl

KEYWORDS
Gaming, handheld computers, software engineering, design.

ABSTRACT

Though the Palm is mainly a business tool, many games have been developed for it and more are published daily. This article starts by examining the status of game development on the Palm today. To give an indication on what designing guidelines a game developer for the Palm should take into account, several "rules of thumb" are presented. A description is given of a successful Palm game, "Space Trader", and areas where this game fails are indicated. The article concludes by looking forward into the near future of Palm game development.

INTRODUCTION

Since 1998 with the release of the Palm III, handheld computers have become all the rage (Williams 1999). Starting out as replacements for electronic agendas, newly developed applications turned them into calculators, web browsers, notebooks, email-managers, translators, e-books, databases, barcode-scanners and even remote controls. Of course, games weren't left behind. The number of downloadable games for the Palm runs in the thousands, most of them shareware or freeware. Many of these games are not really worth your while, but there are gems to find for those who keep their eyes open.

This paper examines today's status of Palm game design, gives a few rules of thumb for the technical designing of handheld games, illustrates this by the story of a successful Palm game, and will venture a look forward into the near future of handheld games. It will be limited to discussing games for the Palm OS, which currently has the biggest market share, but most of the statements here are just as applicable to competitors of this OS.

PALM GAMES TODAY

When viewing the landscape of Palm games as it is today, we see games of many types and qualities. There are arcade games like "PacMan", "Galax" and "Hardball"; action games like "Void" and "Ancient Red"; text adventures playable with "PilotFrotz"; role playing games like "Dragonbane" and "Kyle's Quest"; board games like "PocketChess" and "Kalah"; card games like "Hearts"; and strategy games like "Taipan" (see figure 1).

Most games are fairly simple, finding their inspiration in games from the early '80s, but recently, especially since the advent of colour Palms, more complex games have entered the Palm landscape. The evolution of games on the Palm goes fast, and developers try to catch up with even the latest PC games. For instance, the first role playing game for the Palm was a port of the ancient, text-based game "Rogue". This was followed soon by "Kyle's Quest" that was inspired by the first "Final Fantasy" games and "Dragonbane" that was inspired by "The Bard's Tale". Now there is even "Ancient Red", a Palm game reminiscent of the popular "Diablo".

A game like "Ancient Red" looks very beautiful, but does not set the standard for Palm game development. There are several reasons for that. Firstly, though the game does support greyscales, the graphics are too intricate to play it on anything less than an 8bit colour screen. Secondly, even on a colour screen, the interface with its excruciatingly small fonts and pixel-hunting stylus interaction is unclear and difficult to use. Thirdly, the game's memory usage is enormous. Fourthly, it has a steep price. Though the game seems to be popular with some of the people who bought a Palm purely for pleasure, it has only a limited audience.

"Ancient Red" seems to be an exception to the rule that most Palm games are designed and built by a sole programmer. The Palm doesn't allow much in the way of graphics or sounds – the capabilities of the Palm can be compared with those of the first XTs or of a Commodore 64 without a sound chip. Therefore you don't have to be an artist to build a game. Furthermore, for the design of a new game you can be inspired by the multitude of games that have populated the gaming realm since the first release of "Pong". All of this makes the current Palm gaming world an ideal place for an aspiring game programmer to leave his mark. A good Palm game can be produced by a single programmer in a matter of months. There is a large audience awaiting such efforts, and it is even possible to start a small one-person business in Palm games.

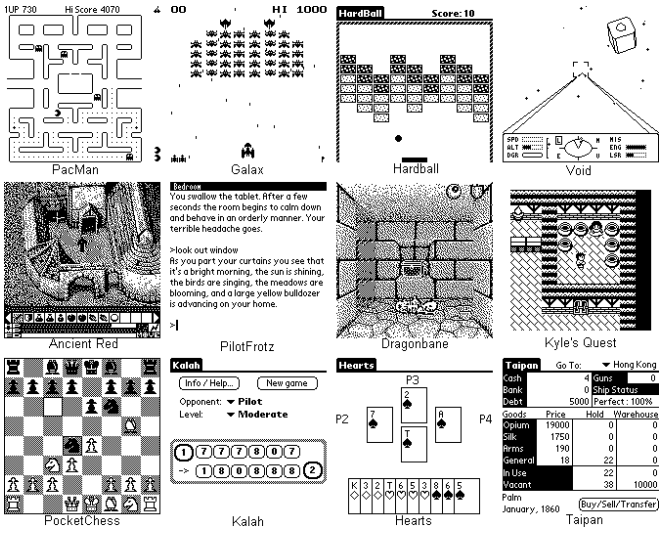


Figure 1: Several Palm Games

RULES OF THUMB

Handheld computers are not, like PCs, general-purpose machines. Palms are not about putting a PC in your pocket, they are meant for executing quick, simple tasks, anytime, anywhere. Adding ten numbers on a Palm is acceptable; filling out a large spreadsheet to calculate the total costs of sending your kid through college is not. One of the reasons the Palm is far more successful than its derivatives is that the Palm hardware, as well as the Palm OS, is geared towards such quick tasks (Rhodes and McKeehan 1999). However, this philosophy puts serious limitations on the design of Palm games. This paragraph presents several rules of thumb that take these restrictions into account. Some of these rules are derived from the "Palm OS Programmer's Companion" (3Com Corporation 1999), but here they are especially geared towards game design. Note that these rules only give pointers on designing a well-functioning Palm game, not necessarily a fun one.

1. The Screen Resolution Drives the Design

A Palm screen is limited in size to only 160x160 pixels on a 6x6 cm area, with at most 256 colours. The game designer has to make do with that. Even worse, in order not to restrict the audience too much, the game should work with monochrome screens, since even greyscale support has only recently been added to the Palm OS. The screen resolution is a serious limitation in the functionalities a game can offer. The best way to handle this is not to design a game and then see how it fits on the screen, but to take the screen restrictions into account at every moment during the design.

Of all Palm applications, especially games often make mistakes in screen design. There are games that create their own, small fonts that are unreadable. There are games that use tiny buttons that are easy to miss with the stylus. There are games that use rich colours and use a dithered, unusable version of that to support monochrome screens.

Because of the physical restrictions imposed by the screen, the functionality of each screen should be limited. Scrolling using scrollbars should be avoided. There should be no more than around four buttons on the screen, and they should be big enough to be easily tapped with the stylus. If both colour and monochrome screens are supported, the monochrome version should be designed separately from the colour version.

2. Games Are Small and Use Little Dynamic Memory

Memory is a valuable resource for the Palm. Not only is it used to run applications; it is also used to store them and their databases when they are not in use. There is no hard drive. The earliest Palms only have half-a-megabyte of memory or even less. Most of the Palms today are limited to two megabytes. New Palms will commonly have at least eight megabytes, but it'll take a few years before all the older ones are replaced.

There are two problems with the memory limitation. The first is that the user is severely restricted in the number and size of the installed applications. Games are usually not a priority and will be quickly deleted when there's a lack of memory – especially the larger games. When deliberating whether or not paying a shareware fee for a game, the user's decision will not only be guided by the monetary costs, but also by the memory costs. The quality of a large game must be very high to get a user to pay for it.

The second problem is that an application cannot dynamically assign much memory. The size of the dynamic heap of a Palm is restricted, in the worst case to 32K. On a PC this kind of problem is solved by "swapping out" memory. On a Palm this can be emulated by storing allocated records in a database. However, one should realize that this database is built in the generic Palm memory, and therefore limited by the size of the memory the user left unused after installing applications, which usually isn't that much.

It should be noted that especially colour images are serious memory hogs. One full-screen colour image will take around 25K of memory. While this doesn't seem that much, one should realize that many applications as a whole are even smaller than that. One way around this is to make graphics an optional feature of a game and to store them in a separate database, which users can install if they have memory to spare.

3. Game Control Is by Parsimonious Use of the Stylus

Most PC games use a variety of input devices. Normally you can use a keyboard and a mouse, sometimes a joystick is required. None of these is delivered with the Palm. The standard Palm has three ways of inputting data: by tapping the screen, by writing on the graffiti pad, and by pushing one of the shortcut buttons. Of these, screen tapping is the main method to control a game. This is completely different from controlling PC games, and the design of a game should take this into account.

To answer the question why the graffiti pad should not be used: the pad is used for inputting texts, but is not as easy to use as a keyboard. The writing of text should, especially in games, fulfil only a minor role. Pick-lists are often a good alternative.

To answer the question why a game should refrain from reprogramming the buttons (except for the up/down button, of which the use is application-dependent): the buttons are used to switch to the major tasks the user performs with the Palm, and as such reprogramming them is annoying especially to the business user. Many games, mainly arcade games, break this rule. Of course, it may be hard to create an arcade game without reprogramming the buttons. In practice, however, there are many games which reprogram the buttons while a better way of controlling the game would be using the stylus. In case one finds the buttons are indeed the only viable input device for the implementation of a certain function, one should at least make sure they get released at each and every opportunity.

Even though tapping is the way to control a game, the number of taps necessary to execute that control should be as small as possible. Main functions should require no more than one tap. "Double-tapping" (equivalent to double-clicking with a mouse) is theoretically possible but awkward and therefore totally out of the question.

4. Exiting a Game Takes No More Than a Second

Currently, the normal use for a Palm is not "playing games". It is used as a business device, a portable extension of a PC,

a small tool to support tasks done "in the field". The average Palm user will have a few games loaded to entertain him while standing in line or to kill some time while travelling, but these are not the reason why he or she bought the Palm.

Palm users can't be as patient as PC users, since they are usually "on the move". Even a few seconds delay in activating their desired application is too much. They ask their Palm for information and want it now, not in a moment. Therefore, games should allow the user to instantly switch to another application. This means they should be quick to shut down.

Some games break this rule by taking a considerable amount of time to save their state. Other games even do worse and prohibit the user from leaving the game at any given moment, either because that means the user will instantly "lose" the game, or simply because it refuses to give the control back to the OS at that time. These are serious problems.

5. A Game Session Can Be as Short as One Minute

Palm users tend to fire up a game when they have a few moments to spare. They are waiting in line for a cashier or for the bus to arrive. Often they don't know how much time they will have, they just want to play a few turns and be able to put the game away at any given moment. A game should allow them to do that.

This means that the game's state should be obvious at a glance. Even if a game could take an hour to play from start to finish, any possible state in which the game can be loaded should be clear to the user, even if he played the previous session a week before.

It also means that a game can be saved in any possible state. Some games, when exiting, don't save the actual game state but a previous game state. That's not much of a problem if it means the user loses a few seconds of playtime. However, if one game-turn takes a couple of minutes and the state is only saved at the end of a turn, the user should at least play a whole turn to make any progress, so he can't play a one-minute session.

6. Sound Is Optional

When a user plays a game on a PC, he can, if he wants (and the neighbours agree), have sound blasting from enormous speakers. Palm games, however, are usually played in public. Computer game sounds, those simplistic Palm sounds in particular, are very annoying to bystanders, and if they are a required feature of a game, the game cannot really be played at times when the user would like to. Besides, sounds can be turned off for the Palm as a whole, and many users only have "alarm" sounds turned on.

7. Games Don't Need a Manual

When a user starts a new game, he doesn't want to study a manual; he wants to spend five minutes on playing a game. A paper manual, of course, is out of the question anyway, since he won't be carrying it with him. But also on-line manuals are not an acceptable method of training. The Palm screen is too small to read text efficiently, and besides, reading long manual texts only costs time which could better be spent on game-playing.

The solution is to make the game interface easily understandable. In principle, the screen should tell the user everything he needs to know. For board games, which may have complex rules, this is not always possible, but even then at least the interaction with the game should flow naturally. A short instruction text, three screens long at the most, is acceptable if it's easy to access and it only needs to be read once.

The designer can count on the user tapping things that look like buttons or selection lists. So, a good way of creating an easily playable interface is to create buttons for the actions a user can perform and label them in a clarifying manner. This may be problematic because the number of buttons can't be too large, and labels should be short for lack of room. That's exactly where a designer's job comes in.

8. Games Are Thoroughly Tested

This rule is a wide open door, of course. However, it should be particularly noted for Palm games, since Palm users commonly have little patience with buggy software. Games are not required software, there are many free or pretty cheap alternatives available, and storage capacity is valuable. The game designer must expect that a game that annoys a user, even only once, will be deleted. Only serious testing, not only for bugs but also for playability, may avoid this from happening.

SPACE TRADER

One way to create a game that adheres to the rules of the previous paragraph is to keep it simple. That is actually what has been done for most of the Palm games that exist today. However, that does not mean that complex games are impossible to create for the Palm, though it is more difficult to keep a game playable. Especially the screen limitation needs serious consideration for complex games, since they usually offer the player many possibilities which must be presented somewhere on the screen. This paragraph presents experiences with the design of a complex Palm game.

The History of Space Trader

Halfway the year 2000 there were no complex strategy/trading games for the Palm. The few trading games that existed had very simple mechanisms and depended mainly on luck. As an experiment I decided to build a space trading game, with complex trading rules, geared towards the strategic player. As inspiration I used the famous '80s game "Elite", removing the 3D flight mode and increasing the trading aspects considerably. I implemented the game using CodeWarrior release 6. After a thorough beta testing phase I released it September 2000 as freeware under the name "Space Trader". The first release was quickly followed by a few new releases in response to player comments, adding some graphics and making interface enhancements (see figure 2).

The game has been received surprisingly well, getting almost unanimous good reviews from players, websites and magazines, and receiving several nominations as "the best Palm game" in some category. The number of players, nearly one year after the initial release, I estimate between 100,000

and one million. The current version of the game allows the player to trade ten different kinds of goods in a galaxy comprised of 120 solar systems, each with their own size, technological development level, political system, special resources and special events; to fly ten different kinds of ships equipped with a selection of different kinds of weapons, shields and gadgets; to become a trader, bounty hunter, pirate or smuggler; and to go onto a several quests of varying difficulty.

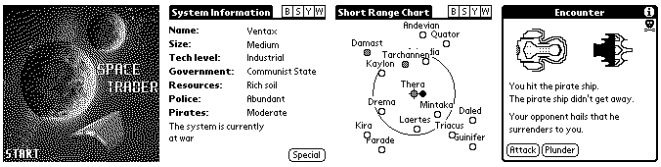


Figure 2: Several "Space Trader" Screens

Space Trader Design

The gameplay of "Space Trader" consists of two parts: "trading" and "travelling". When trading, the player is docked at a space station, where he can switch to many different functions, like buying and selling of cargo, buying equipment, visiting a bank or choosing a new solar system to fly to. While travelling, the game is in a modal state, and the player has to negotiate his way past several encounters with police, pirates and other traders he meets underway. The player can interrupt the game at any moment and restart it at that same point later. If the interruption happens in a fight, the encounter screen tells him everything he needs to know. If it happens while docked, all information about current cargo, current ship, current location and current quests is accessible.

The game is designed as a continuous journey towards some final goal, which takes a couple of hours of playtime to reach. The main activity of the player is seeking good trading opportunities, which is a strategic task, because it is only slightly randomized (to simulate small local market fluctuations) and the best deals are made when the player succeeds in exploiting the characteristics of neighbouring solar systems. All the factors which play a role in the determination of the prices of the different trade items are explained and justified in a separate manual, which isn't required reading but which might make the game more fun to play.

Besides hoarding money to fulfil the final goal, the player must use his earnings to upgrade his ship and equipment, to manage to survive encounters with the pirates, which become stronger and more numerous while the game progresses. Combat itself is simple and does not rely on reflexes, but mainly on making prudent decisions on when to fight, when to flee, and when to surrender (most players abhor surrendering, but especially on the harder difficulty levels it's sometimes the best choice).

This design carries in it the risk of repetitiveness. A reasonably successful attempt has been made to resolve this by having the following features:

- There are so many factors which influence the trading balance, that the trading experience differs notably from solar system to solar system.
- There are quests which the player gets offered occasionally, many with special rewards.
- The player can choose his own playing style, and can switch between styles if he likes.
- The player can attempt to influence the success rate of his own playing style by choosing the right equipment and mercenaries.
- The enemy AI is simple but takes into account the player's style, strength and success rate.
- Dying is possible in the game, but always avoidable for a smart player.

For the user interface it was decided to strive for clarity by using standard fonts and buttons, by spreading functions over as many different screens as needed, and by avoiding the use of abbreviations.

Space Trader and the Rules of Thumb

After presenting rules for the successful design of Palm games, it's interesting to examine how well my own exploits confirm to them. "Space Trader" mainly sins against three of them in the following ways:

- The game is not small. The first release was 150K in size, which is acceptable but strains the limits. Later releases increased this to 315K for the colour version. This is mainly the result of adding graphics. The graphics have been compiled in the game itself and are not stored in a separate database, so the user is forced to install them. A black-and-white version has been made available that is "only" 230K in size.
- The game gets controlled mainly with the stylus, but the stylus use is not parsimonious. "Space Trader" simply contains too many screens. A better design would have integrated more functions in fewer screens. In figure 3, the three screens in the game that are used for the trading of cargo are shown. After the first release, a player suggested allowing the user to buy cargo immediately from the average price list. For the next version, I added the ability to buy cargo by tapping on one of the cargo types listed on the average price list. This would popup a screen showing the details of this cargo item and asking the user if he wanted to buy it, and if so, how many canisters (with the possibility to buy "as many as possible" by tapping one button). This greatly increased the user-friendliness of the trading function and decreased the number of necessary screen-switches considerably. With a bit of thinking, it would be possible to also integrate the selling function, producing one "Cargo Trading" screen. It should be noted that many reviews have mentioned the ease of use of "Space Trader" and the clarity of the screens. Integrating more functions in one screen could mean sacrificing some of that clarity, but it would probably be worth it.
- The game is more in need of a manual than I would like. There is a huge in-game manual in the form of help screens and several menu-items that explain certain aspects of the game. I don't believe these are all necessary, and in fact I have discovered that many players don't read the help texts at all. However, a player must at least be instructed on how to buy and sell cargo, how to equip his ship, and how to travel to other solar systems. The need for a manual for these aspects would

be greatly reduced if the following changes were made: the addition of a "nerve centre" screen from which the basic functionalities would be accessed and the reduction of the number of screens as already mentioned above.

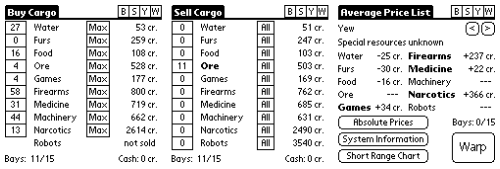


Figure 3: "Space Trader" Cargo Trading Screens

Lessons Learned

The following statements I would offer as "lessons learned" from the production of "Space Trader":

- **Graphics *are* important**. The first version of "Space Trader" only contained a simple graphical "Start" and "Victory" screen. I got many requests to add more graphics, and did so by adding ship pictures and enhancing the existing illustrations. I took care that the graphics were not just "eye-candy", but fulfilled a useful role in the game. After that, requests for more graphics were seldom ventured.
- **Graphics are not *that* important**. The graphics in "Space Trader" are very simple, but I get virtually no complaints on that. Players seem to focus on the strategic aspects of the game. And graphics do have a bad side: adding them doubled the memory footprint of the game.
- **The fewer screens, the better**. When docked at a space station, the player of "Space Trader" has access to ten main screens and about the same number of sub-screens. This requires far too many screen-switches. A redesign of the game would probably make do with about four main screens and a few sub-screens, without losing functionality.
- **Don't use the menu except for housekeeping functions**. "Space Trader" makes the mistake of making some of the basic functionalities only accessible through the menu. This means some players completely forget about them. The menu should be used only for housekeeping.
- **Keep interest going by having developments and a few surprises**. The most appreciated aspects of "Space Trader" are the quests, which form small stories in the game, and the few surprises the game has in store for the player. A better version of "Space Trader" would have far more and different quests, and would offer multiple solutions for them.
- **Keep supporting low-level Palms**. A surprisingly large number of players write to me how glad they are that there are still developers that write games for older Palms, so it seems worthwhile to keep supporting them. This may, however, prove to be impossible for games that require more advanced functionalities, like elaborate graphics.
- **Incorporate a savegame feature**. Intentionally, I left out the savegame feature for "Space Trader", because it would remove all tension from the combat sequences. However, there are three good reasons to have it: Firstly, if a player moves on to another Palm or if his Palm suffers a serious crash, he will have lost his current game; secondly, it allows players to play multiple parallel games on one Palm; thirdly, if a player reports a bug, having access to a savegame is an important help in solving it. When adding savegames, a good solution for the combat issue could be penalizing the player for restoring a saved game, for instance by disallowing ranking in the highscore table.

THE FUTURE OF PALM GAMING

Palms have dropped in price and have become popular not only as a tool for business users, but also as a "cool gadget" for everyone else, especially youngsters. These people are far more interested in games than in business applications. The Palm Corporation seems to find this enough reason to forego their philosophy of focussing on simplicity and starts to push the Palm more and more as a games machine. This means it is very likely that in the near future we will see the release of Palms specifically for gamers, with extra graphic capabilities (perhaps a 320x320 screen with 16bit colours), extra sound capabilities and an earphone jack, a lot more memory or memory cards on which software can be preloaded, easy Internet access for multi-player games, and perhaps a built-in joystick. In fact, hardware which supports some of these features is already available. The consequences for the Palm OS are minimal, it's mainly the hardware that will be changed. Palms without these extras will, as long as the game-specific Palms are clumsier than regular Palms, still be available for business purposes. Just like what happened with PCs, the gamers will use the most expensive and advanced machines.

The consequence is, however, that game programmers must decide whether they will build a game which runs on all Palms, or whether they will produce a game for a "GamePalm". The first will restrict the game's features, the second the audience (though it will, of course, mean that the designer can relax on the rules presented earlier in this paper). At first, a single programmer is probably better off concentrating on the basic Palm, because it is likely that professional programming teams will quickly start working on producing games for the enhanced Palm. Not long after, I expect there will be no room anymore in the Palm game development world for a one-person business. Therefore, the statement at the start of this article that the Palm gives an ideal opportunity for a sole game programmer to make a name for him- or herself, has an expiration date. The time to grab that opportunity is now. It will probably have passed in a few short years.

REFERENCES

3Com Corporation. 1999. *Palm OS Programmer's Companion.* Part of the Palm OS Software Development Kit.

Rhodes N. and J. McKeehan. 1999. *Palm Programming: The Developer's Guide*. O'Reilly & Associates.

Williams J. 1999. *Games to Go: The PalmPilot Series*. GamaSutra. Available at: http://www.gamasutra.com.

DISTRIBUTED AUDIO-VIDEO SHARING BY COPY-AND-TRANSFER OPERATION FOR NETWORK 3D GAMES

Hirotatsu Sakamoto Yoshihiro Okada Eisuke Itoh Masafumi Yamashita
Graduate School of Information Science and Electrical Engineering, Kyushu University
6-10-1 Hakozaki, Higashi-ku, Fukuoka, 812-8581, Japan
Phone: +81-92-642-3872, Fax: +81-92-583-7632
hirotatu@swlab.csce.kyushu-u.ac.jp okada@i.kyushu-u.ac.jp itou@cc.kyushu-u.ac.jp mak@csce.kyushu-u.ac.jp

KEYWORDS

Network 3D games, Audio-video communication, Copy-and-transfer, Component ware, Distributed virtual environments

ABSTRACT

This paper treats distributed audio-video sharing mechanisms for the development of network 3D games. Especially the authors propose the concept of the *copy-and-transfer* operation. This concept is that making a copy of a visible, manually operable software component and transferring it to another computer enable users to share it. If a facility that manages audio or video data is realized as such a component, even end-users can easily and rapidly build audio-video communication environments through the *copy-and-transfer* operation. This paper explains realization mechanisms of the *copy-and-transfer* operation and describes its availability by showing network 3D game examples.

1 INTRODUCTION

Advances of computer hardware technologies make it possible to create 3D images in real-time, so that 3D graphics software has become in great demand. For this reason, we have been studying 3D graphics software development systems and using *IntelligentBox* [Okada and Tanaka 1995] as our research system. *IntelligentBox* is a component ware, which provides various software components as visible, manually operable 3D objects called *boxes*. *IntelligentBox* also provides a dynamic data linkage mechanism called *slot-connection* that enables users to construct 3D graphics applications by combining existing *boxes* through direct manipulations on a computer screen. Okada *et al.* described that *IntelligentBox* is available for the development of interactive 3D games and also network 3D games [Okada *et al.* 2000]. For network 3D games, face-to-face communication by audio/video media enables players to feel their enemy's emotion and it enhances enjoy-ability during playing a game. Especially for group battle games, audio-video communication is necessary to effectively and strategically play games.

Then this paper treates a distributed audio-video sharing mechanism. Especially we describe a new concept of the *copy-and-transfer* operation. This *copy-and-transfer* operation is similar to the *copy-and-paste*/*cut-and-paste* operations, which are standard GUI operations based on using a mouse-device. With the *copy-and-transfer* operation, even end-users come to easily and rapidly build audio-video communication environments. *IntelligentBox* provided a video managing facility as a *VideoBox*. However this version of *VideoBox* was not available through network. So we improved it in order to support video communication via network [Sakamoto *et al.* 2001]. We have also been developing *boxes* to manage audio media through network. Furthermore a particular *box* called *RoomBox* exists for providing a shared 3D space [Okada and Tanaka 1998]. Using these *boxes*, i.e., *RoomBox*, *VideoBox* and audio managing *boxes*, which support audio-video communication, only through *copy-and-transfer* operations, it will be possible to build network 3D games. In this paper, we clarify availability of this *copy-and-transfer* operation by discussing development costs and performances with showing possible, practical application examples.

Related works

Our research purpose is to establish software architecture that makes it easier to develop 3D graphics applications. Related works are 3D softoware development systems including DIVE [Hagsand 1998], MASSIVE [Greenhalgh and Benford 1995], MERL [Anderson *et al.* 1995][Barrus *et al.* 1996], dVS [Ghee 1995], MR Toolkit [Shaw *et al.* 1993]. DIVE

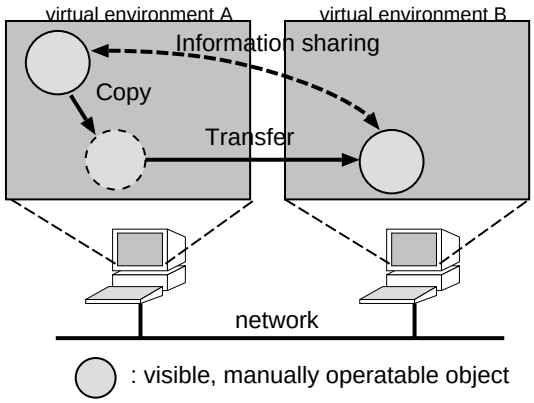


Figure 1: The concept of *copy-and-transfer*.

is a virtual reality construction toolkit system. It has network communication facilities by audio and text media. MASSIVE is a remote conference system based on sharing a 3D virtual space. It has a text-to-audio communication facility, but not a video communication facility. MERL is a development system for collaborative virtual environments. It allows us to communicate by audio and text media. dVS is a commercial product as a virtual reality construction toolkit. It allows us to develop distributed collaborative applications, but does not provide audio-video communication facilities. MR Toolkit is a virtual reality construction toolkit system at a library level. Although these are very powerful systems, it is not easy to use their essential mechanisms when developing distributed 3D graphics applications. Our research system *IntelligentBox* provides various 3D software components represented as visible, manually operable, and reusable objects. Furthermore the *IntelligentBox* system provides a dynamic data linkage mechanism. These features make it easier even for end-users to develop 3D graphics applications including network 3D games. This is the main difference between *IntelligentBox* and the others.

The remainder of this paper is organized as follows: Section 2 describes the *copy-and-transfer* concept. Section 3 explains essential mechanisms of *IntelligentBox* and *RoomBox*. In section 4, we explain audio-video data sharing mechanisms. Section 5 discusses development costs and performances with showing possible, practical 3D game examples. Finally, we conclude this paper in section 6.

2 COPY-AND-TRANSFER

Object-oriented programming becomes very common because of its reusability of software componets and its availability of bottom-up programming manner. However, conventional object-oriented programming is not enough for end-users, who do not have any programming knowledge, since it requests users to write text-based programs. We think that software components should be represented as visible, manually operable objects because such software components allow even end-users to develop software only by combining them interactively on a computer screen. Furthermore, for such visible, manually operable components, it is possible to execute the *copy-and-transfer* operation as shown in Figure 1. In this figure, the user using the left computer can make a copy of a certain component, and transfer it to the right computer. This operation is done only with manual operation using a mouse-device on a computer screen. If the copy and its original software component keep the same states, two users using each computer can share the same data as its states. Therefore the *copy-and-transfer* operation of audio-video managing components allows users to share audio-video data via network.

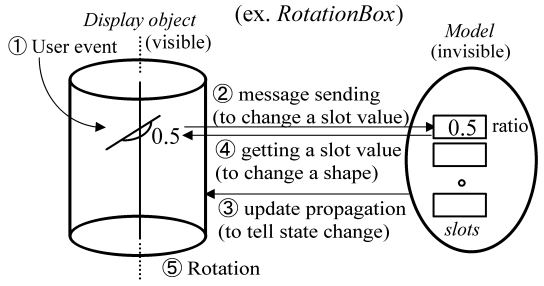


Figure 2: An *MD* structure of a *box* and its internal messages.

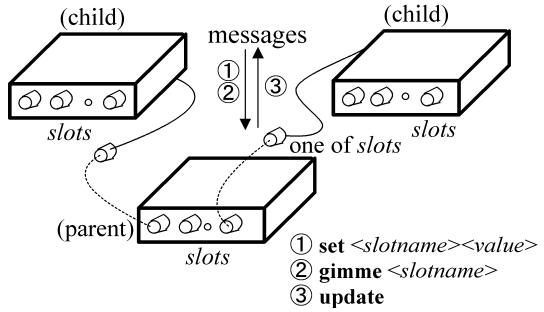


Figure 3: Standard messages between *boxes*.

3 ESSENTIAL MECHANISMS OF INTELLIGENTBOX

The following essential mechanisms are inherited from *IntelligentPad* [Tanaka 1996], which is a 2D synthetic media system, to *IntelligentBox*, since *IntelligentBox* is an extension of *IntelligentPad* to 3D graphics applications.

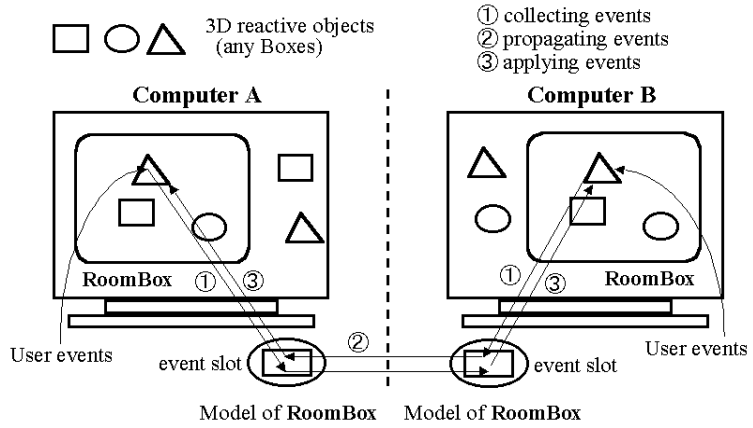


Figure 4: Message flow between two *RoomBoxes* for network collaboration.

3.1 Basic structure of box

As shown in Figure 2, each *box* consists of two objects, a *model* and a *display object*. This structure is called an *MD* (*Model-Display object*) structure. A *model* holds state values of a *box*. They are stored in variables called *slots*. A *display object* defines how the *box* appears on a computer screen and defines how the *box* reacts to user operations.

Figure 2 is an example of a *RotationBox*. A *RotationBox* holds a rotation angle as its *slot* value. Through direct manipulations on the *box*, this *slot* value changes(①②). Furthermore, its visual image simultaneously changes according to the *slot* value change(③④). Then the *box* reacts to the user manipulations according to its function(⑤).

3.2 Message-sending protocol for slot connections

Figure 3 illustrates a data linkage concept between *boxes*. Each *box* has multiple *slots*. Its one *slot* can be connected to one of the *slots* of other *boxes*. This connection is called a *slot connection*. The *slot connection* is carried out by three messages.

A *set* message(①) writes a child *box slot* value into its parent *box slot*. A *gimme* message(②) reads a parent *box slot* value and sets it into its child *box slot*. An *update* message(③) is issued from a parent *box* to all of its child *boxes* to tell them that the parent *box slot* value has changed. In this way, these three messages connect a child *box slot* and its parent *box slot*, and combine their two functionalities.

3.3 A shared-copy and a distributed model-sharing

The *MD* structure allows more than one *box* to share the same common *model*. This mechanism is called *model-sharing* and the operation that generates a copy of a *display object* sharing a common *model* is called *shared-copy*. A *box* generated by the *shared-copy* operation shares all *slot* values. After one of *model-shared boxes* is transferred to another computer via network, a new corresponding *box* is generated in that computer and the *box* has the same *slot* values and keep them always by messages via network to conserve consistency of *slot* values. This means distributed *model-sharing*.

3.4 RoomBox for collaborative virtual environments

This section briefly describes an idea of a shared 3D space and a functionality of a *RoomBox*. As shown in Figure 4, the *RoomBox* has a *slot* named 'event' which holds a current user-operation event operated on its descendant *boxes*. Some specific user-operation events generated in a *RoomBox* are always stored in this *slot* until the next event is generated. As mentioned above, *IntelligentBox* provides a distributed *model-sharing* mechanism. By this mechanism, multiple *RoomBoxes* can share user-operation events with each other. Here, descendant *boxes* of a *RoomBox* are treated as collaborative operable 3D objects.

In Figure 4, there are two *RoomBox models* existing separately on a different computer. These *models* are kept in the same state by messages passed via network. This linkage is built easily and rapidly by making a *shared-copy* of a *RoomBox* on one computer and by transferring it to the other computer. When a user

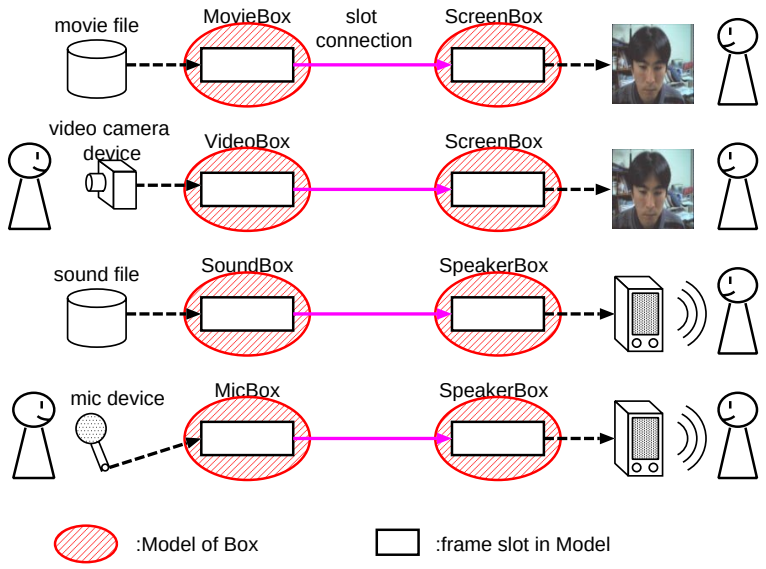


Figure 5: *Boxes* managing audio-video data and their pairs.

operates one *box* in the *RoomBox* on the computer A, his operation event is sent to the *RoomBox model* and subsequently set in its event *slot*. Furthermore, this event is sent to the other *RoomBox model* existing on the computer B by a message. After these processes are completed, the operation event is applied to the corresponding *box* on the computer B. In this way, by using distributed *RoomBoxes*, user-operation events are shared among several computers.

4 DISTRIBUTED AUDIO-VIDEO SHARING MECHANISMS

As previously mentioned, using *RoomBoxes*, it is possible to build collaborative virtual environments easily and rapidly. As well if *boxes* that manage audio-video data exist, it is possible to build audio-video communication environments easily and rapidly by their *copy-and-transfer* operations. Then we designed and implemented six *boxes* managing audio-video data as follows. Actually four *boxes* are used for audio-video communication, i.e., the pair of *VideoBox* and *ScreenBox* for video communication, and the pair of *MicBox* and *SpeakerBox* for audio communication.

1. *MovieBox* reads movie data from a movie file.
2. *VideoBox* gets movie data from a video camera device.
3. *ScreenBox* displays movie data onto its surface as texture images.
4. *SoundBox* reads audio data from a sound file.
5. *MicBox* gets audio data from a mic device.
6. *SpeakerBox* outputs audio data to a speaker device.

Figure 5 illustrates the usage of the six *boxes*. We can implement audio-video facilities as just one software component like Microsoft MediaPlayer^{TM}. However we designed them as six components separetely because of the following. We think that software components should be as simple as possible and should have the same metaphor as existing things in the real world. Such software components are very easier for end-users to deal with and have high reusability.

4.1 Boxes managing video data

MovieBox, *VideoBox*, and *ScreenBox* manage video data. *MovieBox* and *VideoBox* are used for reading video data, and *ScreenBox* are used for displaying video data. The texture-mapping technique is used to display a binary 2D image in a 3D virtual space. Strictly speaking, a texture image is mapped on the surface of a 3D object, i.e., a *box*. These *boxes* have a 'frame' *slot* in its *model*. A texture image is loaded and stored in this *slot*. Periodical updates of the *slot* content allow us to see an animation. *MovieBox* and *VideoBox* also have a 'TRIGGER' *slot*. Whenever 'TRIGGER' is accessed, the next frame will be loaded. Actually a *TimerBox* is used to access the 'TRIGGER' periodically by a *slot-connection*. A *TimerBox* holds a timer value, which periodically increases every user-specified interval time, in its 'time' *slot*. Then this *box* is used as a timer to notify *MovieBox* and *VideoBox* of its timing to get new frame of video data.

Figure 6 and Figure 7 illustrate distributed video data shaing mechanisms using *VideoBox* and *ScreenBox*. Figure 6 is the case without using *RoomBox*.

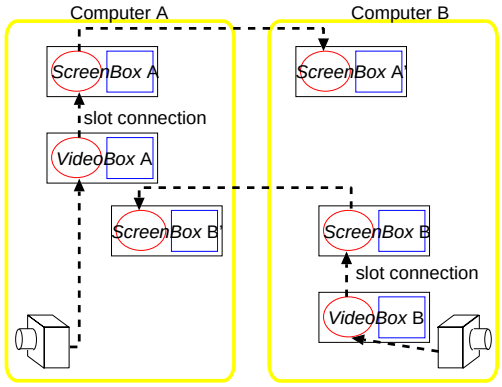


Figure 6: Distributed video data sharing mechanisms without *RoomBox*.

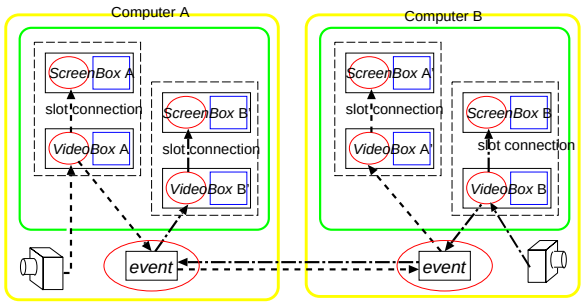


Figure 7: Distributed video data sharing mechanisms with *RoomBox*.

We can build this structure by means of the *copy-and-transfer* operation of *ScreenBox*. This is the simplest way to share audio-video data between two computers. For network 3D games, we have to build collaborative virtual environments using *RoomBoxes*.

Figure 7 is the case with using *RoomBox*. We can also build this structure by means of the *copy-and-transfer* operation of *RoomBox*, which contains a composite *box* of *VideoBox* and *ScreenBox*. Actually a *RoomBox* contains its child *boxes*, which are collaborative operable 3D objects. In this way, even end-users can build collaborative virtual environments including audio-video communication for network 3D games.

4.2 Boxes managing audio data

SoundBox, *MicBox*, and *SpeakerBox* manage audio data. *SoundBox* and *MicBox* are used for reading audio data, and *SpeakerBox* for playing audio data. These *boxes* have the common structure for managing audio-data. Actually these *boxes* have a dedicated buffer called "audio buffer," a 'frame' *slot*, and an 'always' *slot*. The audio buffer is used for storing temporal audio data sent from an audio device or read from an audio file. A part of the audio buffer is mapped to a 'frame' *slot*. The size of this *slot* is fixed. The timing of updating this *slot* value is controlled by an 'always' *slot*, which has a boolean value. If an 'always' *slot* value is true, the update timing is asynchronous. Strictly speaking, in the case of *SoundBox* and *MicBox*, audio data sent from a source is once written to audio buffer and its part is sent to a 'frame' *slot* asynchronously. In the case of *SpeakerBox*, audio data stored in its 'frame' *slot* is written to a part of an audio buffer asynchronously and is sent to speaker device. Actually a *ToggleSwitchBox* is used to change the 'always' *slot* value since a *ToggleSwitchBox* has a boolean value in its 'State' *slot* and it can be used as a switch.

5 DISCUSSION

This section discusses possible, a practical application example, their development costs and performances.

5.1 A practical application example

Figure 8 shows two screen images of a network game, which is a tank battle game actually we have already developed. In this example, there are two players each using a different computer. The left figure is a screen image of one computer and the right figure is that of the other one. As for the left figure, the upper right small view is a camera view of a *CameraBox* attached to the tank controlled by this computer's player. Each player can control his own tank with looking at the each camera view. Furthermore the two upper left small images are two players faces displayed using *VideoBoxes*. As this example case, face-to-face communication is important for enhancement of enjoy-ability during playing a game. Especially for a group battle game, audio-video communication is necessary to effectively and strategically play a game. Then our proposed *copy-and-transfer* operation is significant since it allows even end-users to construct network 3D games.

5.2 Development costs

We give an outline of the simplest way to create an application with video communication using *VideoBox*, *RoomBox* and *TimerBox*:

1. Compose a composite *box* from a *TimerBox* and a *VideoBox*.
2. Connect the 'time' *slot* of the *TimerBox* and the 'TRIGGER' *slot* of the *VideoBox* by a *slot-connection* through a menu selection.
3. Define the composite *box* as a descendant of a *RoomBox*.
4. Make a copy of the *RoomBox* and transfer it to another computer.

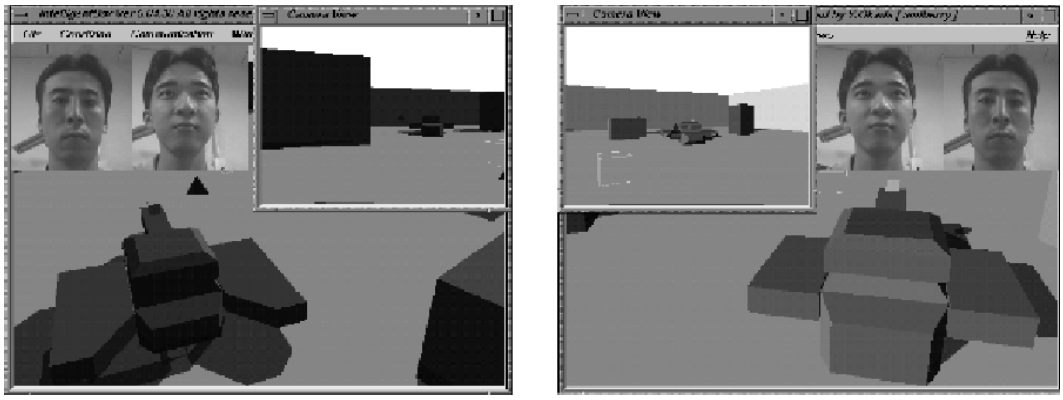


Figure 8: A distributed tank battle game.

In this way, construction process for video communication, based on the *copy-and-transfer* operation, is very simple and easy for even end-users. This process is done immediately only through mouse-device operations.

As mentioned in the paper[Okada *et al.* 2000], using *IntelligentBox*, it is possible to build 3D games without writing any text-based programs only through direct manipulations on a computer screen. The network game example without audio-video communication presented in this paper was also developed in several hours only through direct manipulations on a computer screen.

5.3 Performances

We experimented to evaluate the performance of video communication. So we executed a *copy-and-transfer* operation to each *VideoBox* on two computers and created video communication environment. The environment we experimented is as follows:

- computer A
  - CPU: Pentium III 800MHz
  - Memory: 256MB
  - Network: 100Base-TX
  - OS: Windows 98
  - Graphics Card: Geforce
- computer B
  - CPU: Pentium III 450MHz
  - Memory: 256MB
  - Network: 100Base-TX
  - OS: Windows NT
  - Graphics Card: Cobalt

Parameters are as follows:

- uni-direction or bi-direction
- frame rate(frame/sec)
- resolution(the numbers of vertical pixel $\times$ the numbers of horizontal pixel)
- color depth(byte/pixel)

Table 1 shows frame rates in some cases. For instance, the left lower values, i.e., 2.5 and 0.2, mean the frame rate of computer A to computer B and the frame rate of computer B to computer A respectively in the case that communication is bi-direction, frame resolustion is 128×128 pixels, and color depth is four byte/pixel. The frame rates of computer A to computer B are larger than the frame rates of computer B to computer A because of the difference of CPU peformances of computer A and computer B. In any case, the table does not show performances enough for practical use since we didn't use any data compression technique nor particular communication protocol. So communication performce can be improved if these techniques are used. These are left as a future work.

6 CONCLUDING REMARKS

This paper presented distributed audio-video sharing mechanisms for the development of network 3D games. Especially the new concept, i.e., the *copy-and-transfer* operation, was proposed and its realization mechanisms were explained. If a software component is represented as a visible, manually operable object, the *copy-and-transfer* operation on such an object becomes possible. Then if a facility that manages audio-video data is realized as such a visible, manually operable object, even end-users can easily and rapidly build audio-video communication environments through the *copy-and-transfer* operation. This paper clarified the availability of the *copy-and-transfer* operation by discussing development costs and performances of applications with showing network game examples.

resolution (pixel) depth (byte/pixel)		128 x 128 4	128 x 128 1	64 x 64 4	64 x 64 1
stand-alone	A	18.0	18.0	18.0	18.0
	B	6.6	6.6	6.1	6.1
uni-direct	A -> B	2.6	18.1	10.3	18.1
	B -> A	0.3	0.9	0.9	3.4
bi-direct	A -> B	2.5	17.0	10.3	18.0
	B -> A	0.2	0.8	0.9	3.1

frame rate (frame/sec)

Table 1: Frame rate of *VideoBox*.

ACKNOWLEDGEMENTS

We would like to thank all members of our laboratory for their advices and suggestions. This work is partially supported by foundation for Fusion Of Science and Technology(FOST) of Japan.

References

[Anderson *et al.* 1995] Anderson, B., D., Barrus, W. J., et al., 1995. "Building Multiuser Interactive Multimedia Environments at MERL." IEEE Multimedia, Vol 2, No. 4, 77-82.

[Barrus *et al.* 1996] Barrus, W., J., Waters, C., R. and Anderson, B., D., 1996. "Locals and Beacons: Efficient and Precise Support For Large Multi-User Virtual Environments." Proc. of IEEE Virtual Reality Annual Int. Symp. (VRAIS-96).

[Greenhalgh and Benford 1995] Greenhalgh, C. and Benford, S., 1995. "MASSIVE: A Collaborative Virtual Environment for Teleconferencing." ACM Transations on Computer-Human Interaction, Vol. 2, No. 3, 239-261.

[Hagsand 1998] Hagsand, O., 1996. "Interactive Multiuser VEs in the DIVE System." IEEE Multimedia, Vol. 3, No. 1, 30-39.

[Okada *et al.* 2000] Okada, Y., Itoh, E. and Hirokawa, S., 2000. "IntelligentBox: Its Aspects as a Rapid Construction System for Interactive 3D Games." Proc. of First International Conference on Intelligent Games and Simulation (GAME-ON2000), SCS Publication, 22-26.

[Okada and Tanaka 1998] Okada, Y. and Tanaka, Y., 1998. "Collaborative Environments in IntelligentBox for Distributed 3D Graphics Applications." The Visual Computer (CGS special issue), Vol. 14, No. 4, 140-152.

[Okada and Tanaka 1995] Okada, Y. and Tanaka, Y., 1995. "IntelligentBox:A Constructive Visual Software Development System for Interactive 3D Graphic Applications." Proc. of Computer Animation '95, IEEE Computer Society Press, 114-125.

[Sakamoto *et al.* 2001] Sakamoto, H., Okada, Y., Shimokawa, T. and Ushijima, K., 2001. "Component Based Video Communication Tool for Collaborative Virtual Environment." Proc. of 15th International Conference on Information Networking, 375-380.

[Shaw *et al.* 1993] Shaw, D., Green, M., Liang, J. and Sun, Y., 1993. "Decoupled Simulation in Virtual Reality with the MR Toolkit." ACM Trans. on Information Systems, Vol. 11, No. 3, 287-317.

[Ghee 1995] Ghee, S., 1995. "dVS – a Distributed VR System Infrastructure." In SIGGRAPH '95 CourseNotes.

[Tanaka 1996] Tanaka, Y., 1996. "Meme Media and a World Wide Meme Pool." Proc. of ACM Multimedia'96, 175-186.

CONTENT-BASED RECKONING FOR INTERNET GAMES

Jörg R. J. Schirra
Department of Computer Science
Otto von Guericke University
Universitätsplatz 2
D 39 106, Magdeburg, Germany
E-mail: joerg@isg.cs.uni-magdeburg.de

KEYWORDS
Dead reckoning, behaviour encoding & decoding,

ABSTRACT

Event concepts, which structure our understanding of agents' behaviours as well as and our verbal descriptions, can reduce the amount of messages in network games if they are employed in a content-based extension of dead reckoning for anticipating and communicating the game states among clients. An example case is described by adapting to the new task the necessary recognition and reconstruction routines for such "semantic" event concepts from prior work in an AI project dealing with the linking of natural language generation and computer vision. Beside reducing communication, analysing the players in terms of high-level concepts also opens a preview of dealing with dissociations on a more general level.

NETWORK GAMES AND DISSOCIATION

Messages can pass through the internet only with a certain speed leading often to a significant latency between the time of sending and the time of receiving. This is particularly disturbing in settings with a high amount of interaction as in computer games, when players all around the earth may try enjoying a common game with each others, and long distances are coupled with temporally highly demanding exchanges. If, for example, the updating of the players' positions in a shooter game is delayed, there is a severe danger that the game dissociates, i.e., that the actions of the players no longer belong to one unique context of interaction. Therefore, handling dissociation is one of the crucial technical questions when building internet-based games. So far, several methods have been proposed to deal with the danger of dissociation. Reducing the amount of information that has to be sent through the net is a basis for most of them, as the reduction of necessary bandwidth also lessens the temporal pressure on the communication channel.

Often, a multitude of local game contexts are used that are essentially autonomous and may deviate from each other to a certain degree. Only major differences need explicit synchronization. One of the most successful methods in this framework is Dead Reckoning (or more generally: "predictive contracts", Mellon and West 1995): essentially, the movements of all player characters are locally extrapolated by every participant. Deviations between the predictions of the movements of the "local character" and the "real" positions determined by the player trigger corresponding messages to the others – but only if they exceed a certain threshold. The receivers react by smooth correction movements of the corresponding characters in their respective game contexts. So far, the threshold of allowed deviation depends merely on geometric information, and it is always the same no matter what kind of behaviour in which context is considered.

CONTENT-BASED RECKONING

This paper examines the idea of using descriptions as in natural language for anticipating and communicating the game states between clients instead of geometric information alone. Our verbal descriptions of actions in games are usually structured in events that belong mainly to an intermediate level of complexity: they are richer than pure geometry, and less global than strategic notions. Therefore, they cover relatively large segments of time. They also define context-sensitive thresholds for tolerable deviations only dependent on relevance.

While the usual dead reckoning is easily performed on the description of the game states as already given – basically the coordinates and velocity vectors of the characters – dead reckoning based on high-level descriptions depends crucially on routines for encoding and decoding the behaviour of game characters into concepts underlying our own descriptions of what is going on: e.g., "running along some channel toward a trap door" in a role playing game, or "playing a double-pass with a team-mate in front of the opposing team's penalty area" in an online soccer game. Such routines are described in the next section, based on research in a classical sub-field of AI: natural language systems.

Predictive contracts allow local game clients to act rather independently from each other by calculating the players' probable behaviours, instead of using their real input. Correspondingly, in our approach "semantic" concepts of actions are decoded into concrete spatio-temporal instances of that behaviour, predicting the players' activities. At the same time, every game client analyses the actual behaviour of "its" player by encoding his/her concrete movements into the high-level concepts, and compares them with its own predictions. The *relevant* distinctions are determined indicating necessary corrections of the simulations. This revised architecture is introduced in the second section.

The paper ends with a sketch of the approach's open questions and its potential to stimulate future research.

Behaviour Recognition and Reconstruction in VITRA

To define in an operational manner what we mean in this context by "content" is the first step toward content-based reckoning. Essentially, we need procedures for encoding behaviour given by means of coordinates and velocity vectors into relevant concepts, and also for decoding them back. Fortunately, an existing implementation of the operational semantics of spatio-temporal verbs and prepositions from a natural language system is close enough to the multiplayer game setting that it can immediately be adapted for a soccer game without many changes. In the project VITRA (VIsual TRAnslator), we have studied the connection of natural language generation and computer vision in the scenario of a radio sports report (Herzog and Wazinski 1994). The domain is soccer games. Aiming at a continuous processing from video input to a fluid report in German, many approaches of low and high level computer vision had to be coordinated in VITRA with components for utterance planning, syntax generation, and pragmatic anticipations of the listeners' understanding. Since a "life report" setting was chosen, temporal restrictions also played an important role.

After calculating from the video signal the 3D-positions, forms, and types of "objects" (players and ball), an idealized representation is sufficient for the higher levels of behaviour recognition: the centres of gravity of the players in the 2D-plane of the soccer field in a bird's eye view (Schirra et al. 1987). Encoding the game states is based on the relative positions of the players and the ball with respect to each others and to the geometric and functional parts of the soccer field. Such static spatial relations, which can be articulated by means of locative prepositions like "in" or "to the right of", can efficiently be detected by mathematical applicability functions based on simple geometric concepts and part-whole relations. Applying such a function onto an idealized geometric game state (as given by object recognition) leads to a fuzzy applicability value in [0.0 .. 1.0] (Schirra 1993). In Figure 1, such a function is graphically represented (unmarked position ≡ applicability value 0.0; black ≡ 1.0).

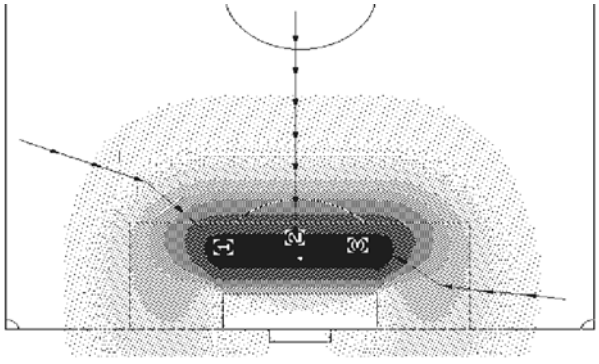


Figure 1: 2D-Typicality Distribution for "in front of" and Three Approximation Paths

More complex events (including simple and interactive behaviours as well as intentional acts) have been defined as temporal sequences of sets of such static spatial relations (plus part-whole relations like team membership), forming Finite State Machines (FSM), the states of which correspond to points in time, the transitions to the flow of time. They can immediately be used for parsing the input data into corresponding events. More precisely, we employed an augmented version of FSM: each state transition is marked twice (cf. Fig. 2): (a) with a condition, i.e., a conjunction of static spatial relations (or sub-events, see below) that have to hold at the time this transition is active, and (b) one of the following types: *start*, *proceed*, *succeed*, *stop*. The later classes determine phases of recognition: the moment of first assumption; an ongoing recognition without final confirmation (i.e., event not completed, completion can still fail); an ongoing confirmed recognition of *durative* events (e.g. "running"); and the final moment of successful recognition (first moment after the event). In order to simplify definitions, these recognition phases can be used for referring to sub-events in state transitions, too (in Fig. 2, the *start* phase of the "pass" event refers to the *stop* phase of an event "ball-possession", i.e., one player is losing contact with the ball).

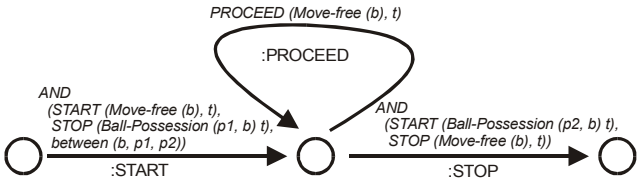


Figure 2: Augmented Finite State Machine for event type "player *p1* passes ball *b* to player *p2*"

The actual recognition of geometric events (e.g. "ball rolling toward something/somebody") or intentional acts (e.g. "scoring" or "attacking") is controlled in an object-oriented manner by means of "type demons" associated to each event type defined: if the conditions of its *start* phase apply to a sufficient degree (≥ 0.8) in the input of that moment, an event instance is created that tries to reach its *stop* phase through a couple of *proceed* (and perhaps *succeed*) loops. At every time step, a transition with fulfilled conditions must be made, or the recognition fails. While the instance is active, an event of that type is recognized as being *seemingly* happening – an assumption that might fail if the FSM does not reach a *succeed* or *stop* phase. Nevertheless, the assumption can already be communicated – covering a relatively long look-ahead on the probable development of the game in the future. The utterance has to be explicitly corrected only if the event recognition fails (Herzog 1992).

Most aspects of VITRA's language generation are not in the focus of attention here; it may suffice to mention that all event instances that are still active or successfully completed at that time are taken into consideration for generating the next utterance. They are dynamically ordered into a speech plan determined by criteria of the event types' relevance and the instances' topicality. The event concepts are linked in the lexicon to deep-case frames for verbs: the verbalization is finally created around that core (André et al. 1988).

Although it may on first view seem unexpected in the context of VITRA, we have also demonstrated that the very same data structures used for encoding – FSM and applicability functions – can efficiently be used for *de*coding corresponding verbal descriptions, i.e., for reconstructing the geometric scene. This problem was investigated in the context of listener modelling: How will a listener understand the utterance under planning? Such anticipations are useful in order to deal with several problems of linguistic pragmatics in language generation. In a nutshell: A corresponding "mental image" is constructed as the listeners' presumed understanding and com-

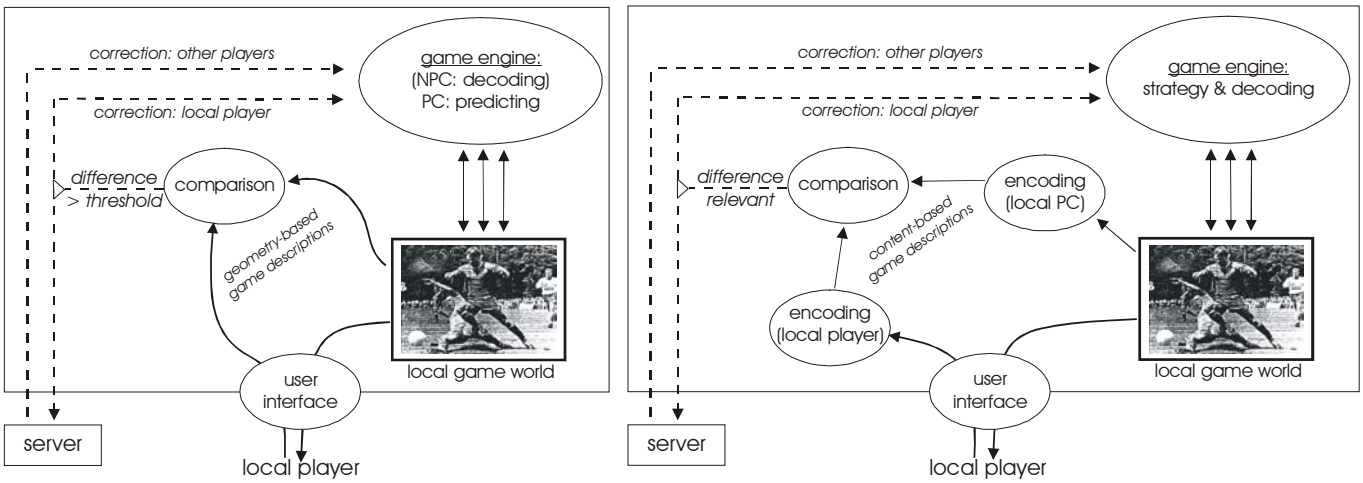


Fig. 3: Schematic Comparison of Standard Dead Reckoning (left) and Content-Based Reckoning (right)

pared to what actually happened on the soccer field; the differences are calculated and used to change the sentence finally uttered (Schirra 1995).

While encoding reduces the amount of irrelevant information, the problem with decoding is quite obviously that additional information has to be "invented" in some way. Here, the applicability (of a description for a given scene) has to be re-interpreted as a measure of typicality (of a scene for a given description): we are always looking for the most typical scene. Then, all objects mentioned are positioned so that all relations considered are maximally applicable. In order to reach that maximally typical scene from a given set of concepts, first the *proceed* and *succeed* transition loops of the corresponding FSMs – additionally marked for that purpose by temporal typicality distributions – are "expanded" in the current situational context, leading to a temporal sequence of sets of static spatial relations that have to hold simultaneously at one moment. Then, each of the sets can be transformed into adequate geometric information by means of a simple hill-climbing algorithm working on the applicability/typicality functions for the spatial relations. Like rubber bands, the typicality distributions "pull" the objects at highly typical positions: three such paths from different starting points are given in Fig. 2, indicating that the hill-climbing algorithm on spatial typicality distributions, which has also been successfully used in the author's group to plan camera paths in computer games (Halper et al. 2001), is highly context-sensitive. It is this context sensitivity that binds together the momentary pictures into the whole of the animation sequence: the result for one moment serves as the starting point for the next step. In this "cinematographic procedure", the typical movements of the objects involved in the context given are geometrically reconstructed in VITRA's listener model (Schirra and Stopp 1993).

Application to Network Games

While the routines in VITRA for encoding and decoding a soccer game in natural language were essentially developed in order to understand the cognitive foundations of speaking about something seen, most of its design principles can easily be adopted for a network computer game. Each player of that game may act as a member of a virtual soccer team. The player's essential activity is controlling the position (or velocity) of the character. He can also "perform" a few locity) of the character. He can also "perform" a few special activities, like kicking the ball. The input therefore consists in a constant stream of position/direction updates, punctured by a kicking action once and again. With respect to content-based reckoning, the soccer domain is intended as an example setting only.

Let us assume a server-client configuration. In the idealized standard dead reckoning algorithm (Fig. 3, left), the local game world is completely built by the client's game engine: the player characters (PC) are mainly controlled by extrapolating their prior movements – sometimes messages from the server initiate corrections. For non-player characters (NPC), we may assume the evaluation of behaviour scripts in the form of extended FSMs – similar to VITRA's decoding sequence described in the previous section. FSMs are indeed a widely used mechanism for determining the behaviour of NPCs in computer games. In this context, too, an idealization of the characters as points is basically sufficient: control points for the subsequent 3D-animation of the character on the screen have to be determined. Note that only in the NPCs' case strategic rules (e.g., from a SOAR-like component; Van Lent and Laird 1999) are employed by the game engine for selecting one of several event types applicable at that time.

The server receives a message from a client if the geometric difference between the local player's actions (input stream) and the predictions of that client's game engine exceeds a fixed threshold. The server forwards each such message with the correcting input coordinates to all the other clients triggering there the corresponding routines of adaptation. (Additional messages may indicate that no message was lost when corrections are superfluous for some time.)

Content-based reckoning (Fig. 3, right) uses the very same architecture extended by two modules for encoding geometry data into higher behavioural concepts: One of these modules "observes" the actual behaviour of the player (i.e., the input), the other one analyses the local game world encoding the PC's behaviour as generated by the game engine. In consequence, the comparison algorithm has to be changed, because now it is not geometric data to be compared with respect to a numeric threshold but complex actions, recognized ones as well as assumed ones, that either are the same in relevant aspects or not. We come back to the criteria of comparison in more detail soon.

Instead of numeric positions and velocity values, the correction messages now carry the information of actions taking place as in verbal descriptions. Of course, in the game setting, the event concepts do not have to be really verbalized in any natural language with all its peculiarities and redundancies. The event type's name, a mark for the current phase of the event recognition, some optional spatial relations indicating important location and direction parameters of that event instance, and a time stamp suffice for a message.

The game engine is modified in order to understand these messages. In general, the PCs' positions are now derived by means of decoding event concepts as described in the previous section, i.e., just like the NPCs' behaviours. On an intermediate level between the strategic overview and the concrete animation of the character, the gradient-based search for maximal typicality concretises behavioural concepts chosen by strategic rules in the current context delivering control parameters for the 3D-animations to be presented on the screen. However, this autonomous selection of actions may be overwritten by the concepts in correction messages – bringing in the *real* actions of the players. The correction itself is anchored in the current game state by the very same decoding procedures that unfold any behaviour of characters. Note that the geometric difference to be corrected cannot be extreme (or it would have been corrected earlier). The context-sensitivity of the cinematographic procedure therefore leads to smooth transitions even if the underlying concept is changed.

It is crucial here that events can be used in correction messages even if they are not yet completed, i.e., if they are merely assumed: events, the beginnings of which have been recognized in the players' movements (but not in the corresponding character's behaviour). Assume that the *start* phase of an event – e.g., of type "scoring", i.e. ball starts moving away from the player toward the opposing goal – was found in the player's activities but not in the local game with the PC's reckoned behaviour. Consequently, a correction message informs the game engines about this event, which then replaces the one currently active for that PC.

Time stamps and phase markers in the messages allow the game engines to position the correct parts of the expanded events at the right temporal frames. A combination with "time warp" techniques looks promising in order to simplify the integration of the events at the right time. Following (Mauve 2000), a network soccer game with its limited number of characters is a plausible application for a "time warp" addition to standard dead reckoning. It may work well with content-based reckoning, too. For virtual environments with more characters, however, "time warping" becomes too expensive.

As only the *start* phase has really happened so far in our example, most of the activity covered by that concept still belongs to the future at communication time. If the "scoring" action is finished as predicted (and communicated to the other clients) no further message needs to be sent for the rest of that time. Thus, the general frequency of messages is reduced as we have intended.

The crucial component is the comparison, filtering out relevant from irrelevant differences of behaviour. In principle we have to decide whether the two descriptions of behaviour constructed by the two encoding components are literally the same or not. In fact, in the original sports report setting, a similar comparison was performed in order to determine 13 kinds of difference between the listeners' anticipated understanding (the re-analysed mental image) and the real events (as seen by the reporter; Blocher and Schirra 1995). For content-based reckoning, only two cases have to be distinguished: (a) no instance of the event type covering the real behaviour is predicted, (b) two corresponding instances of the same event type occur, though with different parameters (mainly, a place or direction slot is filled with different spatial relations or the phase is wrong). In both cases the observed event must be communicated to the game engines. In the second case, a marker has to be added indicating that a corresponding instance with divergent parameters is already active and has to be adapted, not replaced. A third case seems interesting, too: no instance of a predicted event type is recognized in the real behaviour (case (a) inverted). However, this case is not relevant since it always co-occurs with a difference of type (a): then, as a consequence of the correction message, all active events of that PC are replaced.

The player's input may indeed differ to a greater or smaller degree from the positions generated by decoding the active concept: Only if the input cannot be "parsed" any longer into the same concept, a correction message is generated. Thus, depending on the concepts and the phases of the events, the tolerance allowed can be quite different – a few small steps only (e.g., if a lot of spatial relations restrict simultaneously the PC's position), or several long paces (if few relations are given or the PC is away from anything that could act as a reference object at that moment).

Some Problems

So far, an example architecture for content-based reckoning has been presented on the basis of procedures adapted from a project in cognitive science for recognizing and reconstructing events (i.e., behaviours in the game). Of course, a lot of questions remain open. For example: more complex event concepts lead not only to the positive effect of longer look-ahead times but also to higher chances of wrong recognition and more effort for recognizing. Therefore, determining for different game genres the right tactical level of detail and abstraction of the concepts is a particularly crucial task for further empirical research in this framework.

Content-based reckoning trades off less communication load for more local effort for encoding/decoding. These routines, however, become easily quite complicated – a standard problem in AI. While VITRA's routines for encoding and decoding at least performed approximately "in real time", other types of games with more characters or/and more difficult behaviours may become problematic. If the additional components could be transformed into "anytime" versions (Zilberstein 1996), thus leading even under temporal pressure to at least acceptable results, a dynamic adaptation to the resources available at *that* time (and for *that* client) offers an interesting solution. Typicality approximation with hill climbing has already "anytime" property. Encoding can also be transformed – though not easily (Wahlster et al. 1998). However, there remains another, more principal problem with this approach that has to be considered in greater detail: the variations inherent to the results of anytime algorithms may contradict the ultimate prerequisite of dead reckoning (using the *same* predictive algorithm).

AN OUTLOOK ON FUTURE RESEARCH

The most promising perspective of content-based reckoning opens if we consider again the main reason to set it up originally: the danger of dissociation, which so far is reduced but not banned. Encoding game states into semantic concepts that cover longer time intervals allows us at least to recognize earlier the danger of dissociation. So far, we have used the server only to forward the correction messages. It can also maintain its own local copy of the game world predicting the behaviour of all PCs (no encoding of a player or comparison necessary here). Then, with a decoding of the concepts presently active that is faster than real (game) time, the server is able to anticipate the probable development of the game for some time ahead. Some of those game states may evoke strategic rules activating critical interactions of two players with particularly high latencies (it is easy to keep track of the current average latency to each client). For example, in the soccer domain, two characters of opposing teams may run toward each other from some distance – one having the ball, the other performing the beginning of an attack event. The server recognizes that the game is likely to dissociate when the *attack* concept is realized much further (because this initiates critical types of interaction beyond the potential of the current latencies), and may therefore initiate resource-sensitive countermeasures – if available.

In the Germanic world of legends, Wotan appoints the Valkyries to influence with minimal "perceptibility" for the participants the fights between human heroes, and to decide the encounters as he demands – based on his general plan and the earlier "performances" of the heroes. The legendary schema, which we name "the Wotan Principle", suggests an idea of how to keep together a dissociating game without too much loss of "gameplay": when it recognizes a probable dissociation, the server (acting as Wotan) decides how the sensitive interaction is going to happen, though only on a very general, coarse scale (essentially: the results). To this purpose the server can, for example, script a sequence of the semantic concepts described above. The decision could depend on some general game principles or, in a rather futuristic version, on players' personal profiles derived automatically from earlier encounters. It is left to the clients (acting as "Valkyries") by merging the script with the players' reactions to set up the events accordingly, giving the local player some room for her/his own activities (i.e., remaining relatively imperceptible) nevertheless binding him or her into the general schema harmonized *a priori* with the other players in focus. Of course, the concrete sequence of events may be realised in quite a different way for each of the players involved in a dissociating scene. But the overall results are unique and allow all players ("surviving" – in a shooter game) to continue after the dissociation in a non-dissociated game with relatively similar memories of that encounter. The effects of this hidden authority on the "gameplay" feeling are yet unknown and form a major focus of interest in our future research on content-based reckoning.

* * *

Whilst most computer games are said to be provided with an internal AI, few results from Artificial Intelligence research on visual event recognition, natural language generation, and user modelling of the past twenty years have had a noticeable influence on the development of computer games so far. The transfer described in this paper is an attempt to improve this situation. In particular, insights in the cognitive aspects of defining motion verbs and spatial relations may hopefully play a more prominent role in computer games in the future.

REFERENCES

André, E.; G. Herzog; and T. Rist. 1988. "On the Simultaneous Interpretation of Real World Image Sequences and their Natural Language Description: The System SOCCER." In *Proceedings of the 8th European Conference on Artificial Intelligence* 1988 (Munich). Pittman, London, 449-454.

Blocher, A. and J. R. J. Schirra. 1995. "Optional deep case filling and focus control with mental images: ANTLIMA-KOREF." In *Proceedings of the International Joint Conference on Artificial Intelligence* 1995 (Montreal, August 20–25). 417–423.

Halper, N.; R. Helbing; and Th. Strothotte. 2001. "A Camera Engine for Computer Games: Managing the Trade-Off Between Constraint Satisfaction and Frame Coherence." In *Computer Graphics Forum: Proceedings of Eurographics 2001*. Manchester, 2001. To appear.

Herzog, G. 1992. "Utilizing Interval-Based Event Representations for Incremental High-Level Scene Analysis." In *Proceedings of the 4th International Workshop on Semantics of Time, Space, and Movement and Spatio-Temporal Reasoning* 1992 (Château de Bonas, France), M. Aurnague (ed.), IRIT, Toulouse, 425–435,

Herzog, G. and P. Wazinski. 1994. "VIsual TRAnslator: Linking Perceptions and Natural Language Descriptions." *Artificial Intelligence Review* 8 (2/3), 175–187.

Mauve, M. 2000. "How to Keep a Dead Man from Shooting." In *Proceedings of the 7th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS)* 2000, Enschede, The Netherlands, 199–204.

Mellon, L. and D. West. 1995. "Architectural Optimizations to Advanced Distributed Simulation." In *Proceedings of the 1995 Winter Simulation Conference*. Ch. Alexopoulos (ed.). IEEE, Piscataway, N.J., 634–641.

Schirra, J. R. J. 1993. "A Contribution to Reference Semantics of Spatial Prepositions: The Visualization Problem and its Solution in VITRA." In *The Semantics of Prepositions - From Mental Processing to Natural Language Processing 1993*, C. Zelinsky-Wibbelt (Ed.). Mouton de Gruyter, Berlin, 471–515.

Schirra, J. R. J. 1995. "Understanding Radio Broadcasts On Soccer: The Concept 'Mental Image' and Its Use in Spatial Reasoning." In *Bilder im Geiste: Zur kognitiven und erkenntnistheoretischen Funktion piktorialer Repräsentationen*. K. Sachs-Hombach (Ed.). Rodopi, Amsterdam, 1995, 107–136.

Schirra, J. R. J.; G. Bosch; C.K. Sung; and G. Zimmermann. 1987. "From Image Sequences to Natural Language: A First Step towards Automatic Perception and Description of Motion." *Applied Artificial Intelligence* 1(3), 287–305.

Schirra, J. R. J. and E. Stopp. 1993. "ANTLIMA – A Listener Model with Mental Images." In *Proceedings of the International Joint Conference on Artificial Intelligence* 1993 (Chambéry, France, Aug. 29 – Sep. 3). 175–180.

Van Lent, M. and J. Laird. 1999. "Developing an Artificial Intelligence Engine." In *Proceedings of the Game Developers Conference* 1999 (San Jose, Ca., Mar. 16–18). 577–588.

Wahlster, W.; A. Blocher; J. Baus; E. Stopp; and H. Speiser. 1998. „Ressourcenadaptierende Objektlokalisation: Sprachliche Raumbeschreibung unter Zeitdruck." *Kognitionswissenschaft* 7(3), 111–117.

Zilberstein, S. 1996. „Using Anytime Algorithms in Intelligent Systems." *AI Magazine* Fall 1996, 73–83.

LATE PAPER

NEW ANTI-ALIASING AND DEPTH OF FIELD TECHNIQUES FOR GAMES

RICHARD CANT, NATHAN CHIA, DAVID AL-DABASS

Department of Computing and Mathematics
The Nottingham Trent University
Nottingham NG1 4BU.
Email: richard.cant/david.al-dabass@ntu.ac.uk

KEYWORDS
Computer Graphics, Anti-aliasing, super-sampling, depth of field, focus, Open GL.

ABSTRACT

We describe software techniques that will enable Open GL capable graphics cards to implement antialiasing and depth of field effects in software. The methods allow any hardware facilities that are available on the graphics card to be used to improve performance but do not require hardware support in order to work.

INTRODUCTION

Sophisticated graphical and optical effects have in the past been the preserve of pre-rendered animation sequences taking hours or even days to calculate. In other cases these effects were incorporated in real time systems but only in very expensive military simulators, Potmesil and Chakravarty 1981, Cant and Sherlock 1987, Montrym et al, 1997. Since that time high end graphics workstations have also incorporated these techniques, e.g. Silicon Graphics. However recent advances in technology suggest that some of these effects should now be considered for real time implementation even on relatively low cost systems, such as PCs and games consoles. In this paper we will explore the possibilities of implementing some of these features by making use of existing facilities in hardware 3-D accelerators via OpenGL.

ANTI-ALIASING

For the implementation of anti-aliasing, this paper will attempts to replicate nVidia's quincunx anti-aliasing (which is built into the hardware of an expensive GeForce 3 card) Figure 1, by using the existing hardware calls of common 3D accelerators.

nVidia's quincunx does the filtering at the stage where the buffer is rasterized to the screen. The 3D-scene is rendered normally, but the Pixel Shader is storing each pixel twice, Figure-2, in two different locations of the frame buffer. This does not cost more rendering power than the rendering without AA, but requires twice the memory bandwidth of the pixel write operation at the end of the pixel rendering process.

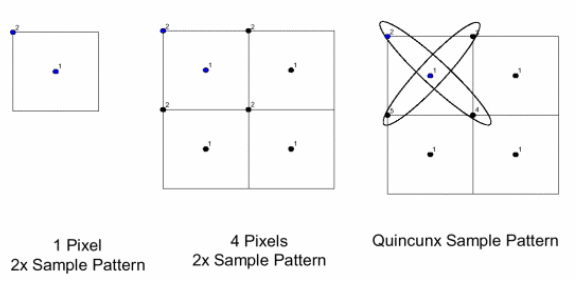


Figure 1: Quincunx Anti-aliasing

By the time the last pixel of the frame has been rendered, the HRAA-engine of GeForce3 virtually shifts the one sample buffer half a pixel in x and y direction (Figure 3).

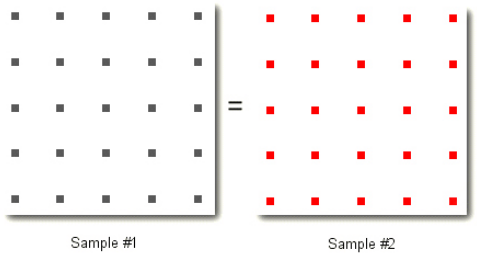


Figure 2: Storing pixels twice

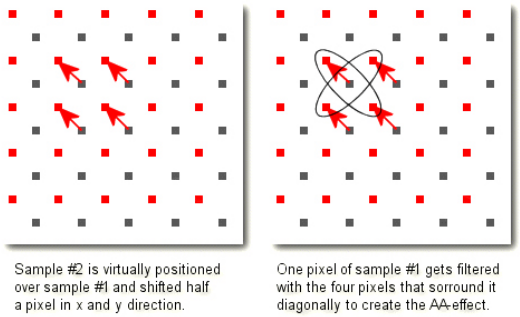


Figure 3

This has the effect that each pixel of the 'first' sample is surrounded by four pixels of the second sample that are 1/SQR(2) pixels away from it in diagonal direction. The HRAA-engine filters over those five pixels to create the anti-aliased pixel. The weights of the pixels are shown in Figure 4.

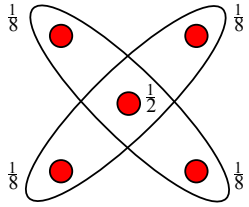


Figure 4: Weights of the quincunx pixels

Figure 5 is a comparison of quality between the anti-aliasing results. These images were captured by www.tomshardware.com for an article on the GeForce 3 video card. The image was taken from a frame in Quake III: Arena. (Nowadays, Quake III: Arena is used more often as a benchmarking tool than a game.)

It is quite clear that the quality of quincunx filtering is quite close to that of the 4x super-sampling anti-aliasing.

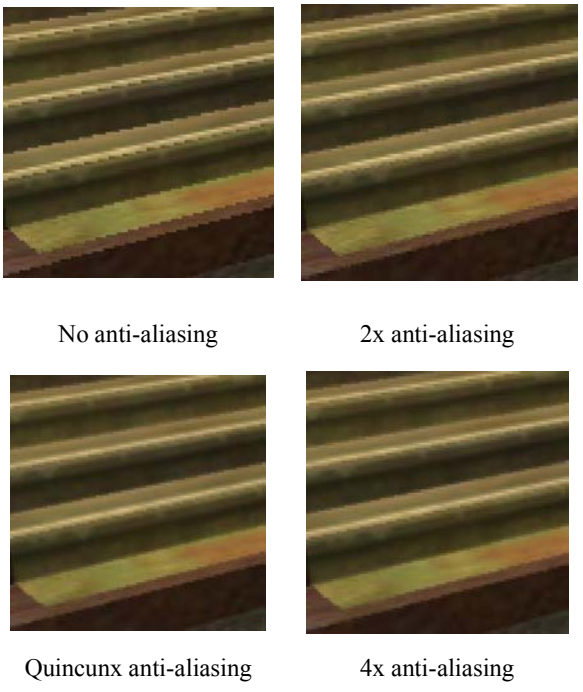


Figure 5: Comparison of anti-aliasing quality

Coming up with a similar technique with the available hardware is quite intuitional. By examining figure 2.3 and 2.4 a little bit closer, one can deduce that the quincunx sample can be reduced to the 2x2 sample in figure 6.

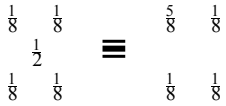
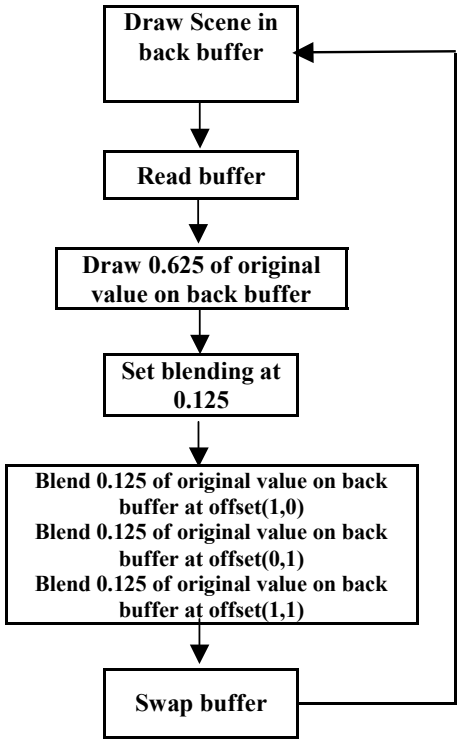


Figure 6

The top-left pixel and the centre pixel are virtually the same pixel in the case of a quincunx sample. The implemented algorithm simply captures the entire back buffer and draws it back to the same position {*offset(0,0)*} with 0.625 of the original value, blend with alpha value of 0.125 for three other images at *offset(0,1), offset(1,0)* and *offset(1,1)*. This would be equivalent of averaging the 2x2 sample with the new weights in Figure 6:

R= 0.625*offsetR(0,0) + 0.125*[offsetR(1,0) + offsetR(0,1) + offsetR(1,1)]
G= 0.625*offsetG(0,0) + 0.125*[offsetG(1,0) + offsetG(0,1) + offsetG(1,1)]
B= 0.625*offsetB(0,0) + 0.125*[offsetB(1,0) + offsetB(0,1) + offsetB(1,1)]

The algorithm can be summed up in the following diagram:



MIP MAPPING DEPTH OF FIELD

MIP mapping is a popular technique intended to reduce aliasing. The essence of the technique is to pre-compute the texture at different levels of detail, Figure 7, and to use smaller textures for polygons further away from the viewer. It aims to improve graphics performance by

generating and storing multiple versions of the original texture image. The graphics processor chooses a different MIP map based on how large the object is on the screen, so that low-detail textures can be used on objects that contain only a few pixels and high-detail textures can be used on larger objects where the user will actually see the difference. This technique saves memory bandwidth and enhances performance. The acronym MIP stands for Multum In Parvo (Latin for 'much in small')

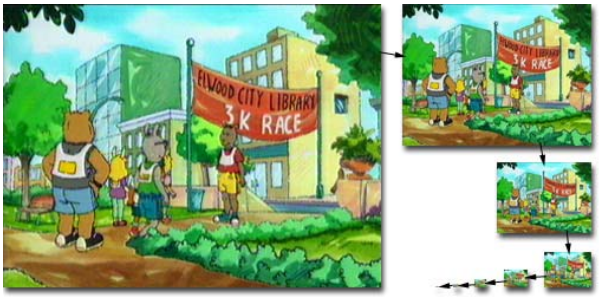


Figure 7: A 320x240 image reduced to 160x120, 80x60, 40x30 and so on…

The 3D hardware bilinear hardware kicks in when the smaller images are enlarged and as it attempts to fill in the missing pixels, an image that is more blurred will be generated (Figure 8). This algorithm will attempt to abuse this aspect of the MIP mapping hardware of a modern 3D accelerator to generate the blurred portions in an image with depth-of-field.

Figure 8: Enlarging a scaled down image

Since MIP mapping is such a standard technique, it would be safe to assume that all hardware accelerators, even the pioneering 3dfx Voodoo card, will be able to do it without much effort at all. The scene can either be rendered on the back buffer or the memory buffer before requesting the hardware to generate the MIP maps.

If the hardware does not natively allow rendering to a buffer other than the back buffer, rendering can be done on the back buffer first and the memory block will have to be copied by hand. And similarly, if MIP-map generation is not supported in hardware, it can also be read by hand and averaged with neighbouring pixels when scaling down.

"MIP-map Pixel Value" = [pixel(1,1) + pixel(1,2) + pixel(2,1) + pixel(1,2)] >>2

(where '>>' refers to shifting the bits to the right)

```
Render Scene onto
back buffer and Z-
buffer.
{
        :
}
```

```
Capture Scene from back buffer
{
    for (y=0;y<height;y++){
        for (x=0;x<width;x++){
            image(x,y)=backbuffer(x,y)
        }
    }
}
```

```
Generate MIPMAPs
{
    int level=0
    Repeat until image is 1x1
    {
        scale image to
        (
            WIDTH / 2^level
            by
            HEIGHT / 2^level
        )
        Use as LOD 'level' image
        Increment 'level' by 1
    }
}
```

```
Draw DOF image back onto back buffer
{
    for (y=0;y<height;y++)
    {
        for (x=0;x<width;x++)
        {
            backbuffer(x,y)=MIPMAP[ Z-value for
                pixel(x,y)](x,y)
            //MIPMAP[float i](int t,int s)
            //      where 'i' is a choice of which
            //      level of mipmap to use
        }
    }
}
```

MIP mapping DOF algorithm

Since the next level of detail is always a quarter of the current level, the averaging isn't as taxing as it sounds. Each destination MIP map pixel is just the average of the four corresponding source pixels, arranged in a 2x2 square.

MIP mapping and bilinear sampling can be merged to form tri-linear sampling where two neighbouring levels of detail from the texture are averaged to generate an in-between image. This will effectively allow the blurring of an image to be varied without the use of a filtering kernel and therefore speeding up the process of generating depth-of-field because multiple different degree of blurring will be needed for every single render.

It may be handy to be able to do a full screen blur with ease but this is an extreme effect and may not be much use when it comes to rendering a 3D scene other than doing transitions. It is, however, more desirable to be able to blur out some objects and leave the others sharp. Depth-of-field effects is can now be possible where only objects in the extreme distance or foreground are blurred, image focussing, in real-time. Depth-of-field would give a very impressive photo-realistic look to a rendered image (as shown with the cartoon image in Figure 9).

Figure 9: An image after MIP mapping depth of field

The next step in the algorithm is to decide which portion of the image would use what level of blurring. This will have to be done when the image is being rasterized onto the back buffer. The decision for each pixel will be made based on the Z-buffer value at the same position.

By definition depth of field is the total distance, on either side of the point of focus, which , when viewed from an appropriate distance, appears sharp in the final image. The interest of this algorithm is to move real-time graphics away from their usual artificial look which is caused by shaded triangles projected in a 2 dimensional space.

The cartoon image is intentionally chosen as an example because like all or most real-time 3D computer graphics, everything appears to be in the foreground and without the perspective drawn into the cartoon, the confusion may be even greater. Notice how the depth of field increase the realism as well as reducing the confusion between the background and foreground image which plagued the left image of Figure 9. This forces the viewer's eye to concentrate on the more important, non-blurred characters.

Since the backbone of the major part of the algorithm relies on the 3D hardware, the whole process could be automated into the hardware itself. This would provide easy access to the depth-of-field or blurring effect with just an API call.

As with all great things, there is a down side to this algorithm. One of the deficiencies of this technique is the effect which appears like an aura around a focused object. The appearance of this effect is not at all bad as it creates a "Vaseline lens" effect (Figure 10) on an image and brings it closer to photo-realism.

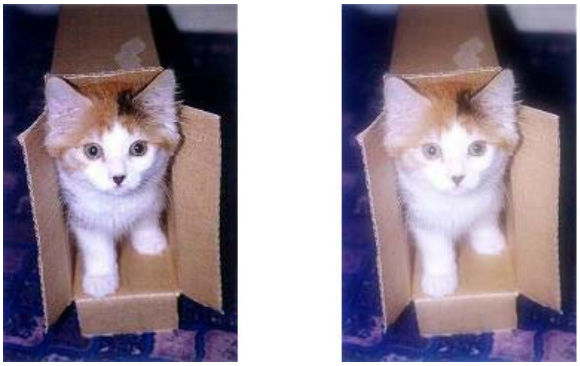


Figure 10: Photo of Xena, the cat, without and with the "Vaseline lens" effect

The other effect that is not pleasant appears when the focused object is in the background. Due to the cut-off of the Z-buffer when the foreground is blurred, instead of fuzzing out the foreground, a distinct line can be seen where the foreground edge meets the background.

So while the effect obtained when focusing the foreground and blurring the background adds realism to an image, the effect when focusing the background and blurring the foreground is less forgiving. This renders this technique only effective when doing depth of field of the former.

A possible solution to stop this effect is to blur out the Z buffer when doing this method of depth-of-field so that no sharp edges would be present, and hence will give a smoother transition from the foreground object to the background object.

Another possible but less appealing solution would be to render the foreground objects (objects before the focal point) onto individual memory buffers. These images will then be used as sprites, drawn on the position where the object will be if drawn in 3D. Do multi-pass blending jitters with

*Offsets = (focal_point – object_distance_from_focal_point) * constant*

with the sprites to create the fuzzy edged objects that are at the foreground.

For the mentioned MIP mapping DOF implementation to work effectively, one would need to get to the metal of the hardware. For the purpose of this project, a simplified version of the algorithm, which could produce a similar result on existing 3D accelerators, was implemented.

Figure 11: Original screenshot image from the game Airblade.

Figure 12: Images with different LOD layered and scaled to appear as the same size and position to the viewer

```
Render Scene onto back buffer and Z-buffer.
{
        .
        .
}
```

```
Capture Scene from back buffer
{
   for (y=0;y<height;y++)
   {
      for (x=0;x<width;x++)
      {
         image(x,y)=backbuffer(x,y)
      }
   }
}
```

```
Generate MIPMAPs
{
   int level=0
   Repeat until image is 1x1
     {
        scale image to
        (
          WIDTH / 2^level
            by
          HEIGHT / 2^level
        )
        Use as LOD 'level' image,
        Increment 'level' by 1
        }
}
```

```
Draw DOF image back onto back buffer
{
        On Depth Testing but do not write changes to depth buffer
        Let Z be the depth where objects are to be sharpest
        Draw LOD[3] at Z-3*offset
        Draw LOD[2] at Z-2*offset
        Draw LOD[1] at Z-1*offset
        Draw LOD[0] at Z
        Draw LOD[1] at Z+1*offset
        Draw LOD[2] at Z+1*offset
        Draw LOD[3] at Z+1*offset
}
```

MIP mapping DOF algorithm (second implementation)

While the first implementation assumes an infinite amount of LODs in the image, this implementation will reduce that to a finite amount of LODs.

As before, a scene like the one in Figure 11 will be rendered, MIP maps generated. These maps will be layered and scaled in such a way that they will all appear the same size and position to the viewer (Figure 12). These images will be effectively be billboards cutting through the depth of the scene based on the existing Z-buffer values.

By positioning the desired LOD on appropriate position along the Z-axis, the Z-buffer mechanism will automatically cut out the unwanted portions of a layer that might block the layers at the back.

Figure 13: Simplified Z-buffer of the image.

Figure 14: MIP mapping depth-of-field in effect.

In figure 13, the darker portions are objects that are further away and the brighter portions are objects nearer to the front. Figure 14 is drawn by drawing the LOD images in the way illustrated in figure 12 using the Z buffer values in figure 13.

As can be seen from Figure 14, although the levels of LODs have been reduced to a finite amount, the technique can still deliver a convincing effect of depth-of-field.

The effects that are found in the previous implementation will still exist in this implementation, however the 'foreground blurring edge' effect will not be visible in Figure 14 because the focused character is in the foreground.

CONCLUSIONS AND FUTURE WORK

Techniques were proposed and implemented to perform anti-aliasing and depth of field processing using features of OpenGL and current 3-D accelerators. Surprisingly impressive images have been obtained in spite of the fact that these facilities are not supported 'natively' by the equipment in use. Future work will include refinements of these algorithms, exploration of the possibilities for direct implementation in future hardware and investigation of other effects such as motion blur, sun glare and film grain.

REFERENCES

1. M. Potmesil and I Chakravarty, "A Lens and Aperture Camera Model for Synthetic Image Generation", in Proceedings of ACM-SIGGRAPH 81, Dallas, Texas, August 3-7 1981, pp297-305, Vol. 15, No.3.
2. Richard S. Wright, Jr and Micheal Sweet, " OpenGL Super Bible", 2nd Edition, Waite Group Press.
3. Cant, R.J. and P.E. Sherlock. 1987, "CIG System for Periscope Observer Training", in Proceedings of the 9th Inter-Service/Industry Training Systems Conference, 311-314.
4. J. Montrym, D Baum, D Dignam, and C Migdal, "InfinitReality: A Real-Time Graphics System", in Proceedings of ACM SIGGRAPH'97, pp 293-301, August 1997.
5. S Nishimura, and T Kunii, "VC-1: A Scalable Graphics Computer with Virtual Local Frame Buffers", in Proceedings of ACM SIGGRAPH'96, pp 365-372, August 1996.
6. Sergei Savchenko, "3D Graphics Programming", SAMS.
7. Rod Stephens, "Visual Basic Graphics Programming", Wiley.

AUTHOR LISTING