Generating 3D Multiplayer Game Maps from 2D Architectural Plans

Ewan Summers, Kristoffer Getchell, Alan Miller, Colin Allison
School of Computer Science, The University of St Andrews
North Haugh, St Andrews, FIFE KY16 9SX
Email: {es54, kg, alan, colin}@dcs.st-andrews.ac.uk

Abstract–**This paper presents an approach for using 2D architectural plans to automatically generate 3D environments which can be used as the basis for archaeological excavation scenarios, delivered through networked, multi-user learning environments. The approach discussed in this paper has been developed as part of the LAVA project, which aims to provide support for explorative learning of archaeological principles. By mixing 2D management interfaces, 3D first person perspective multiplayer game engines and network communication, LAVA allows teams of users to engage with a simulated environment based on real-world data. Owing to the importance of realism within the LAVA simulation environment, methods and techniques must be developed to enable real-world site data to be represented accurately within the virtual environment. The tool discussed in this paper represents the first attempt at generating 3D environments using real-world data obtained during on-site excavation work.**

I. Introduction

The LAVA Platform [1] is an educational system designed to allow users to engage with a simulated archaeological excavation using a variety of 2D and 3D visualisation techniques. By combining the Quake 2 [2] 3D game engine and an institutional learning management system, MMS [3], the LAVA Environment enables students to form teams to cooperatively manage and undertake an excavation of an archaeological site using a familiar user interface. LAVA does not act as a replacement for real-world excavation work, but rather aims to enable students to gain a working knowledge of the operation of an excavation project by presenting a realistic simulation based on real-world data. A key challenge in creating a successful virtual excavation is to make it as engaging and realistic as a real excavation; hence the need to develop mechanisms to allow real-world site data to be represented in LAVA's virtual excavation environments.

There have been several attempts at accurately representing real-world archaeological data in 3D models. Much work has been carried out using VMRL and its successor X3D [4, 5], as well as with more sophisticated modelling tools such as 3D Studio Max, to develop representations of excavation sites. In addition to the modelling work undertaken, sequences of media using Apple QuickTime [6, 7] and Adobe Flash technologies have also been developed which allow photographs of excavation sites to be digitally sewn together to form a 360 degree viewable snapshot of a site. When several of these snapshots, each taken from a different region of the site, are combined, the overall effect allows a viewer to explore a site by jumping between the points of interest and reviewing the 360 degree photograph of the region. One drawback to these methods is the level of interactivity granted to the viewer. As the photographs and models generated are static representations of a given scene, users are provided with a read-only view; they are unable to interact with or alter the simulated environment; they can only view it from a variety of different angles. When considering the requirements of an engaging and realistic excavation scenario it quickly becomes clear that this approach is not well suited to the LAVA platform: Within LAVA simulations students need to be able to review their progress as their excavation work progresses; it is therefore important that they are able to see the alterations to the 3D virtual environment caused by their actions. In order to address this problem, the use of the game engines as the basis for the simulations has been investigated [8, 9], with a reference implementation developed [10] which builds upon previous work by Jacobson et al [11] and Wang et al [9].

Whilst game engines are able to support the level of interactivity required between users and the simulated environment, they are less well suited at portraying realistic scenarios, owing to the fact that most games are more concerned with game-play and player progression [12] than with realistic representation of environments. As such, the tools able to develop 3D models for use within game engines generally make use of proprietary formats, and have only limited support for importing and exporting data from standard, open modelling formats. This therefore makes it difficult to quickly use existing site data to develop new scenario models. In order to automate as much of the conversion process as possible, a modelling framework has been developed which helps scenario developers use real-world archaeological data to construct models using the proprietary game engine formats. Section II of this paper provides an overview of the creation of a Quake 2 map, outlining the tools and processes used. Section III identifies the system design of the framework, outlining objectives and overall system architecture, with section IV describing the evolution of the prototype instantiation of the framework. Section V concludes the paper by discussing the benefits of the framework and highlighting areas of possible future work.

II. Map Creation Process & Toolset

The creation of a Quake 2 map takes three stages:

A. *3D Modelling*

3D Models are generated to form a virtual representation of the objects within the game map. These models specify the shape, positioning and size of objects within the game map. During this process, a number of editors can be used, with GtkRadiant [13] and Quake Army Knife (QuArK) [14] being the most popular.

Both GtkRadiant and QuArK are open source level-editing programs. GtkRadiant is an official editor released by ID Software and supports the development of third party plug-ins. GtkRadiant's file format support is limited to the official .map design files and .reg region files with little support for additional import and export filters. Given that the proprietary file formats are not common between different modellers, and the lack of import and export filters within GtkRadiant, the software cannot be used to generate Quake 2 maps from 3D data stored in more widely used 3D modelling formats. In contrast, QuArK's development has been community lead and, like GtkRadiant, QuArK supports the inclusion of third party plug-in files, located in a specific subdirectory, which are able to extend and enhance the functionality of the software. Unlike GtkRadiant, QuArK allows saving across a wider range of file formats, thereby supporting the development of more portable models. However, of the formats supported, none are compatible with the more traditional 3D modelling formats used outwith the games industry.

Both GtkRadiant and QuArK recognise the proprietary .map format used to store the source of Quake maps. Unfortunately the full structure of the .map format has not been fully documented by ID Software, with partial details being provided by unofficial sources online [15]. The format is ASCII based, describing "entities" in the 3D model. Entities are objects in the model and are described by a collection of brushes. A brush is a solid region on the map constructed by starting with an infinite solid and then removing everything on the planes that define the brush, for example:

Fig. 1 shows the definition of a brush - defining a rectangular region from point (128, 128, 64) to (256, 384, 128). The definition includes 6 planes inside brackets, with each plane being defined by three major parts; 3 non-colinear points, listed clockwise; (128 0 0) (128 1 0) (128 0 1), the associated texture name GROUND1_6, with the remaining numbers being used to define texture attributes. The texture attributes themselves are used for the texture mapping process. The variables are as follows:

```
x_off – Texture x-offset (multiple of 16)
y_off – Texture y-offset (multiple of 16)
rot_angle – Value indicating texture rotation
x_scale – Scales x-dimension of texture
y_scale – Scales y-dimension of texture
```

B. *Texture Mapping*

Once a model has been created, texture mapping can be used to apply images to the surface of the wire-frames which make up the game map. These textures prescribe the visual appearance of the objects within the game environment, with textures being used to differentiate between the different objects in the map.

Quake 2 uses a proprietary file format for storing textures. The .wal format stores textures in an uncompressed, 8-bit indexed standard containing 4 sizes of the image with each version scaled down to half the size of the previous image. This scaled replication is called mipmapping, and is used to speed up software rendering of textured surfaces by having several pre-scaled images within a single file, as shown in fig. 2. There are several software packages able to generate .wal files, with Wally [16] being one of the most widely used.

```
{
  ( 128 0 0 ) ( 128 1 0 ) ( 128 0 1 ) GROUND1_6 0 0 0 1.0 1.0
  ( 256 0 0 ) ( 256 0 1 ) ( 256 1 0 ) GROUND1_6 0 0 0 1.0 1.0
  ( 0 128 0 ) ( 0 128 1 ) ( 1 128 0 ) GROUND1_6 0 0 0 1.0 1.0
  ( 0 384 0 ) ( 1 384 0 ) ( 0 384 1 ) GROUND1_6 0 0 0 1.0 1.0
  ( 0 0 64 ) ( 1 0 64 ) ( 0 1 64 ) GROUND1_6 0 0 0 1.0 1.0
  ( 0 0 128 ) ( 0 1 128 ) ( 1 0 128 ) GROUND1_6 0 0 0 1.0 1.0
}
```

Fig. 1: Brush Definition

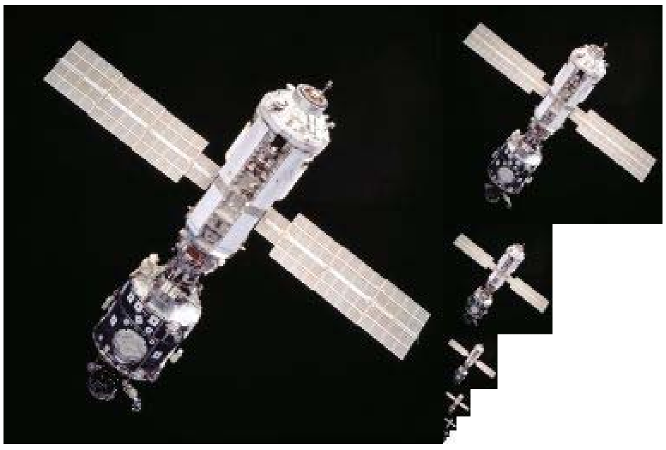


Fig. 2: Mipmapping Example

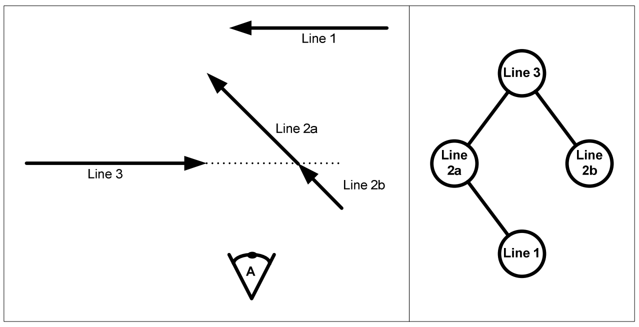


Fig. 3: (a) a 2D map broken into BSP segments, (b) the corresponding BSP Tree

C. *Compilation*

Once a game map has been modelled and textured, it needs to be converted into a format readable by the Quake 2 engine. This process, known as compilation, is handled by the 3D modelling software used in the first part of the map creation process. During compilation, binary space partitioning (BSP) [17, 18] of the map is performed. This allows the map to be broken up into a series of viewable regions, with each viewable region being placed within a tree structure. This process allows the Quake 2 engine to selectively render areas of the map based on their visibility from the player's viewpoint, with areas not within visible range being culled from the rendering process.

As shown in fig. 3a, if a player is located in position A on the map, they would be located at terminal node 2b in the BSP tree shown in fig. 3b. As branch 1 is not visible from the player's current position, the game engine could cull this from the rendering process in order to save memory and reduce processing overheads. The process by which the BSP trees are generated and culled has been the subject of some discussion, with [19] providing a good overview of the process and the improvements in rendering that it can achieve.

III. System Design

Instead of re-implementing the functionality found in many of the most popular Quake 2 map editors, the approach adopted was to develop software to read in 2D architectural plans and then couple this with existing Quake 2 map editing software, thus speeding up the process of map generation whilst at the same time extending the functionality offered by existing tools. As fig. 4 shows, the system makes use of loosely coupled components, with each performing a specific task and outputting the result to the next component in the chain. This process is managed by a graphical user interface (shown in fig. 5) which automatically processes the user input and passes the required data to each module within the system, providing feedback on the process of the 2D to 3D conversion as necessary.

A. *ImageSizer Module*

The main purpose of the ImageSizer module is to creating images that can be batch converted to .wal files using the Wally application. Due to the restrictions of the .wal format, Wally requires that images are sized appropriately, with the height and width being a multiple of 16. This process is especially time consuming when performed manually on many images, however the ImageSizer module is able to quickly process a large number of images, storing them for later use in the 3D modelling environment.

B. *HeightsManager Module*

The HeightsManager module handles the specification of heights in a custom format .hgts file. The .hgts file is simply a set of tuples including a 'piped' RGB value and a number to represent the height. The piped RGB value is a string with the values separated by the | character. For example "255|0|0" would be a representation for saturated red.

The module can both read existing .hgts files and also write new files. When writing an .htgs file, it takes data passed from the GUI module and outputs it into an ASCII file in the required format, with the relationship between colours and wall heights being decided by the user on a project by project basis. When reading an existing .hgts file, the module passes data to the GUI module which can then be used to automatically build up the heights of the walls represented in the 2D plan.

C. *ImageReader Module*

The ImageReader module's main function is to produce a set of lines, where a line is described as a start and end point, for the MapWriter module. It takes as input a formatted image which represents a 2D plan of the site of interest. The input image can be in a variety of formats including, Bitmap, PNG, JPEG, GIF and TIFF, but must be a line drawing of the site featuring all of the walls to be included in the 3D map. The background of the input image must be white, with the lines drawn at least 1 pixel thick in any colour other than white. When the image is scanned by the ImageReader module, the lines are recognised and their colour used to determine the height of the wall by reference to the HeightsManager module. Once the wall has been

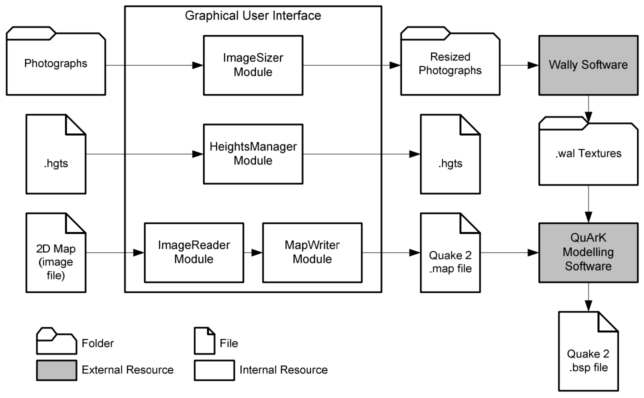
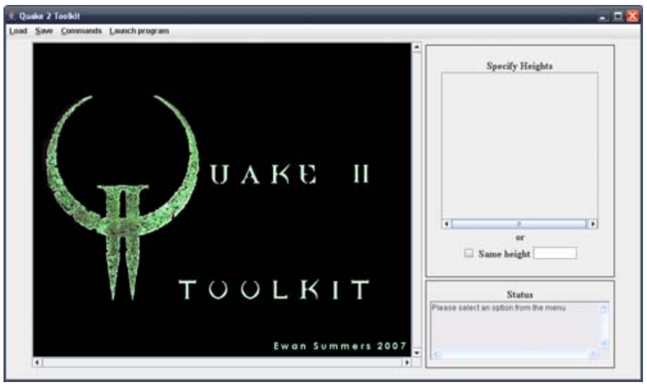


Figure 4: Loosely coupled framework design



Please select an option from the menu

Figure 5: Toolkit Graphical User Interface

recognised and its height obtained from the HeightsManager, it is converted into a series of points which are passed to the MapWriter module to be converted into the Quake 2 map format.

D. MapWriter Module

Based on the input passed from the ImageReader module, the MapWriter module's function is to produce a valid Quake 2 .map file that can be read by existing Quake 2 level editing software. As data is passed to the MapWriter module, the walls are reconstructed in memory. Once the map has been calculated in memory it is output to a .map file ready for processing within existing modelling tools.

At the end of the conversion process, the QuArK modelling package is used to bring together the converted textures and newly generated map file before compiling them into a Quake BSP file which can then be loaded as a level in the Quake 2 game environment.

IV. Prototype Evolution

During the development process, the software framework went through several stages of redesign, led by the need to ensure interoperability with existing modelling tools. The original design of the system, as shown in fig. 6, was based on the principle of extension. The basic premise was to construct add-ons to an existing modelling tool. During development, this approach was found to be too heavily tied to a fixed set of modelling tools, and thus the ability to make use of the benefits of the framework limited to those who made use of the modelling tools for which the add-ons were developed. In order to address this issue, a loosely coupled system was designed, with the conversion framework designed to work outwith existing modellers. Fig. 7 shows the revised design, with the existing 3D modelling software shown as an interchangeable component within the system.

Whilst fig. 6 outlines a system able to generate 3D maps based on 2D architectural, it does not construct a map with any texturing information. Following the loosely coupled design, the framework was extended to enable the automatic creation of Quake 2 .wal texture files using a series of user supplied photographs (fig. 7). In keeping with the desire to make use of existing software packages, the framework generates these .wal files by calling an external program. As shown in fig. 8, the output of the external program can then be read directly by the 3D modeller of choice. The drawback of this approach is that user input is required in order for the textures to be applied to the map, however as much of the preparation work is handled by the framework, the amount of time needed to complete the texturing of the map is greatly reduced.

In order to tie all of the components in the framework together, a graphical user interface is used (see fig. 5). As the final framework design in fig. 4 shows, the graphical user interface handles the integration of each component within the framework, allowing data to flow through the system and into the external tools. In this case fig. 4 shows

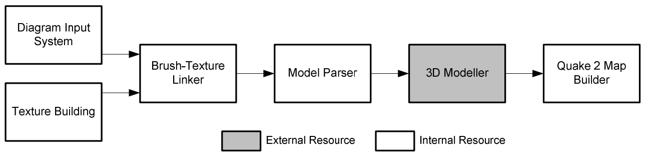


Fig. 6: Original Design

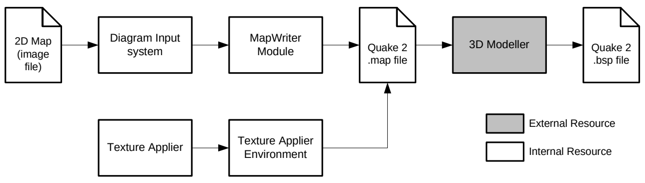


Fig. 7: Loose coupling between framework and 3D modelling software using .map file as interchange format

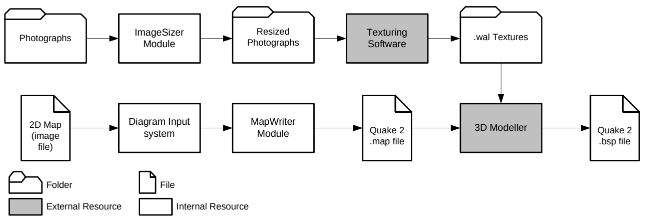


Fig. 8: Coupling the Map creation and Texture Generation Processes

Wally being used to manage the .wal texture files and QuArK being used to combine the map and texture files into the compiled Quake 2 BSP file. As has been discussed previously, the choice of external programs is subject to user preference, with the framework supporting the use of alternatives as desired.

V. Conclusions and Future Work

Initially a fully automated system that took a 2D input file and produced a ready-compiled Quake 2 BSP map had been envisioned. During the development of the framework, it quickly became clear that this level of automation would be difficult, if not impossible to achieve. The construction of a 3D map using 2D architectural plans requires a degree of perception with spatial reasoning in both 2D and 3D. Whilst a person can look at a 2D photograph and gain an understanding of how the space would look in 3D, this is not something a computer can easily achieve. People have inbuilt metadata about dimensions and the world which is gained through living and interacting with our environments. Computers do not readily have this information, and as such the conversion process cannot be easily automated. When it became clear that full automation was not feasible within the scope of this project, Amdahl's law of diminishing returns was followed; instead of aiming for full automation, the parts of the system that would have the most significant effect on reducing developer time were automated instead. With the modelling tools identified, the most time consuming aspect of map creation was orientating walls in the 3D editor; a part of the map creation process that the framework is able to successfully automate.

At the start of the 2D – 3D conversion process, the framework reads in data from a standard image format (GIF, JPG, PNG etc). Whilst this approach ensures that the framework does not require proprietary input files, it does restrict the error checking that can be done on the input somewhat. An interesting alternative would be to accept input in the form of the Autodesk Design Web Format (DWF); an open standard for the efficient distribution and communication of rich design data. This would allow more detailed design data to be processed by the framework whilst maintaining the use of open and widely adopted standards. It would also provide a direct conversion route between industry standard modelling software and Quake 2 level editors; something currently not available. The feasibility of a DWF import model is currently being investigated.

It is clear that whilst the framework developed has many advantages and possible applications, the approach itself is still in its infancy. Indeed, the topic of 2D to 3D conversion has proven to be more of a research question that originally envisioned. Whilst the approach of grouping existing software into a framework has worked with reasonable success and allowed the developed framework to fulfil the objective of automating much of the 2D to 3D conversion process, it is clear that the topic of 2D to 3D conversion requires far more research before a fully automated process could be adopted.

REFERENCES

[1] Getchell, K., et al. The LAVA Project: A Service Based Approach to Supporting Exploratory Learning. in IADIS International Conference WWW/Internet 2006. 2006. Murcia, Spain: IADIS.

[2] Quake 2. [First Person Shoot-em-up Game] [cited 2006 1 June]; Available from: http://www.idsoftware.com/business/techdownloads/.

[3] Allison, C., et al., MMS: A User-Centric Portal for e-Learning, in 14th International Workshop on Database and Expert Systems Applications. 2003, IEEE: Prague, Czech Republic.

[4] VRML: Virtual Reality Modelling Language. 2007 [cited 2007 12 January 2007]; Available from: http://www.web3d.org/x3d/vrml/.

[5] X3D: Open Standards for Real-Time 3D Communication. 2007 [cited 2007 12 January 2007]; Available from: http://www.web3d.org/.

[6] Sweetman, R. The Sparta Basilica Project. [Archaeological Excavation Report] 2000-2001 [cited 2006 1 June]; Available from: http://www.bsa.gla.ac.uk/research/index.htm?field/recent/spartabasilica/main.

[7] Sweetman, R. and E. Katsara, The Sparta Basilica Project 2000 - preliminary report. 2002, BSA: Athens. p. 429-468.

[8] Lewis, M. and J. Jacobson, Game Engines in Scientific Research. Communications of the ACM Special Issue, 2002. 45(1): p. 27-31.

[9] Wang, J., M. Lewis, and J. Gennari. A Game Engine Based Simulation of the NIST Urban Search and Rescue Arenas. in 2003 Winter Simulation Conference. 2003: ACM.

[10] Getchell, K., et al. A Computer Games Approach to Exploratory Learning - Lava: A Case Study in System Design. in INSTICC 3rd International Conference on Web Information Systems and Technologies. 2007. Barcelona, Spain: INSTICC.

[11] Jacobson, J. and L. Holden. The Virtual Egyptian Temple. in World Conference on Educational Media, Hypermedia and Telecommunications (ED-MEDIA). 2005. Montreal, Canada: Associated for the Advancement of computing in Education (AACE).

[12] Malone, T., What Makes things Fun to Learn? A Study of Intrinsically Motivating Computer Games, in Department of Psychology. 1980, Stanford University: Stanford, California, USA.

[13] GtkRadiant. 2007 [cited 2007 5 May]; Available from: http://www.qeradiant.com/top/.

[14] QuArK. 2007 [cited 2007 5 May]; Available from: http://quark.planetquake.gamespy.com/.

[15] Quake MAP Specs. 2007 [cited 2007 5 May]; Available from: http://www.gamers.org/dEngine/quake/QDP/qmapspec.html.

[16] Wally. 2007 [cited 2006 5 May]; Available from: http://www.telefragged.com/wally/.

[17] Fuchs, H., Z. Kedem, and B. Naylor. Predeterming Visibility Priority in 3-D Scenes. in 6th International Conference of Computer Graphics and Interactive Techniques. 1979. Chicago, Illinois, USA: ACM Press.

[18] Fuchs, H., Z. Kedem, and B. Naylor. On Visible Surface Generation by A Priori Tree Structurews. in 7th International Conference of Computer Graphics and Interactive Techniques. 1980. Seattle, Washington, USA: ACM Press.

[19] Torres, E. Optimization of the Binary Space Partition Algorithm (BSP) for the Visualisation of Dynamic Scenes. in Eurographics '90. 1990. Montreux, Switzerland.