A Simplified Level Editor

Brent Cowan, and Bill Kapralos
Faculty of Business and Information Technology, Health Education Technology Research Unit (HETRU),
University of Ontario Institute of Technology. Oshawa, Ontario, Canada.
Email: brent.cowan@uoit.ca, bill.kapralos@uoit.ca

***Abstract*—The majority of game engines include their own level editing software (a *level editor*) which can take months or even years for a user to become proficient with. Level editors often contain tools that duplicate some of the features found in modeling software packages even though they are typically used in conjunction with modeling packages. In this paper, we present a level editor that simplifies the creation of three-dimensional models/scenes by concentrating only on the features specific to the arrangement of models and materials needed to create a three-dimensional environment/scene only. The level editor presented here is intuitive, simple to use, and allows a three-dimensional scene to be created with minimal experience and effort.**

I. Introduction

The task of creating a three-dimensional environment (e.g., level) is commonplace in game development. However, creating such environments from "scratch" is a complex and potentially time consuming task requiring the use of specialized software tools [3]. There are a number of software tools currently available for 3D modeling and 3D scene creation. For example, *UnrealEd* is the real-time level editor integrated into the Epic Games *Unreal* rendering engine and has been optimized for building real-time 3D environments. It provides a WYSIWYG camera view, immediate display of all lighting, texture placement and geometry operations in addition to "single-click playability" whereby the designer can launch the *Unreal* viewer and explore their developing scene in real-time [2]. However, in general these available tools are difficult to use, requiring many hours of training and use before one can become proficient with them [3]. One of the reasons for such a steep learning curve is that level editing software packages attempt to offer the most (and impressive) features, loaded with additional tools at the cost of compromising core functionality. Many level editors are large and complex partially because they have branched out to the point where their toolset overlaps that of art programs and game engines.

Current level editors often allow users to perform modeling tasks, lighting effects, particle effects, game play scripting, and artificial intelligence. However, modeling, lighting, and other graphical effects can be performed using modeling-specific software tools such as Autodesk Maya or 3D Studio Max, and do not need to be part of a level editor package. Similarly, scripting, artificial intelligence, and game play elements are a function of the game or game engine and don't necessarily need to be incorporated into level editing packages. Furthermore, many level editors are proprietary and thus designed to be used in conjunction with a specific game series, or game engine. As a result, after investing months to become proficient with a specific level editor, one may be required to learn a completely new tool depending on their current project. To quote Richard "The Levelord" Gray when asked about the difficulties inherent in the transition from one company to another because each company has their own proprietary level editing software, "*When you ask about standards, I presume you mean like in the software industry where engineering disciplines are used such that individuals can bounce from one application, project, or company to another with little re-education. This sort of scale of standardization has not happened yet*" [1].

We believe the goal of creating this "do-it-all" tool is out of scope with the current gaming industry whereby tasks are typically divided into smaller parts, each completed by a separate person or team similar to an assembly line. Level editors that are not game or engine specific need to be developed. In this paper, we introduce a level editor (three-dimensional scene creator) that focuses solely on the creation of three-dimensional scenes/levels. The level editor is intuitive, simple to use, and preliminary (informal) results indicate it requires minimal (if any) training to use unlike the majority of existing level editors and level editing tools.

II. Overview

Level editing software can be simplified by restricting the functionality to features not already available in modeling packages. Thus, lighting effects such as shadows and ambient occlusion are not part of the level editor presented here. Furthermore, our level editor does not provide tools to edit models as modeling is best left to the many excellent modeling-specific programs such as Autodesk Maya or 3D Studio Max. Our level editor focuses on the placement of objects in an organized manner and provides a WYSIWYG view of the virtual world. A minimalist 2D interface provides information about the current object selected, the current position, the number of objects present, and the number of polygons used. The frames per second is displayed in the title bar so that the user is able to easily detect "slow spots" caused by an excessive number of polygons placed close together. Three different camera modes (viewing perspectives) are supported: i) first person, ii) third person, and iii) object mode. A first person perspective is used by many games, so it is intuitive and useful to allow the user to explore the environment using this point of view. A third person perspective is generally better for editing as it allows the user to view the object in its intended surroundings. In object mode the level is viewed as one giant

```
1  Stone/Floor0      0 0 0 0 360 0
47 Metal100/Wall100  1000 80 0 0 360 0
47 Metal100/Wall100  800 120 0 0 360 0
26 Metal40/Wall40    800 -40 0 0 360 0
19 Metal20/Wall20    1000 -40 0 0 360 0
7  Ice/Floor0        2000 40 -800 0 90 0
7  Ice/Floor0        1800 40 -800 0 90 0
7  Ice/Floor0        1600 40 -800 0 90 0
52 arrow/Down        1600 -40 1000 0 180 0
52 arrow/Down        -1800 -40 -400 0 180 0
```

(a)

```
Stone   Floor0    Ground_0.obj       stonesTexture.jpg  200 40  360 90 360  0
Ice     Floor100  Ground_100.obj     ice.jpg            200 40  360 90 360  0
Metal   Floor0    Ground_0.obj       Metal_100w.jpg     200 40  360 90 360  0
Metal   Floor20   Ground_20.obj      Metal_100w.jpg     200 40  360 90 360  0
Metal   Floor40   Ground_40.obj      Metal_100w.jpg     200 40  360 90 360  0
Metal20 Ramp20    MetalRamp_20.obj   Metal_20.jpg       200 40  360 90 360  0
Metal20 RampR20   MetalRamp2_20.obj  Metal_20.jpg       200 40  360 90 360  0
Metal20 URampR20  MetalURamp2_20.obj Metal_20.jpg       200 40  360 90 360  0
Metal20 URamp20   MetalURamp_20.obj  Metal_20.jpg       200 40  360 90 360  0
Metal20 Angle20   MetalAngle_20.obj  Metal_20.jpg       200 40  360 90 360  0
```

(b)

Fig. 1. Sample level editor file. (a) Level file. (b) Catalogue file used to create the levels

object in the center of the screen that can be rotated and viewed from the outside looking in.

The level editor was developed using the OpenGL graphics API and the cross-platform image library DevIL. OpenGL Shading Language (GLSL) support is also incorporated to allow for various "shader effects" to be added to the level and its models. The created levels can be saved and loaded in a very simple and easily parsed file thus allowing the levels to be incorporated into any engine that allows levels to be created in a modular fashion. Each level file is an ASCII text (human readable) file where each object present in the level is assigned a unique number and label, followed by a series of numbers specifying its location and orientation in the world (see Figure 1(a) for a sample level file).

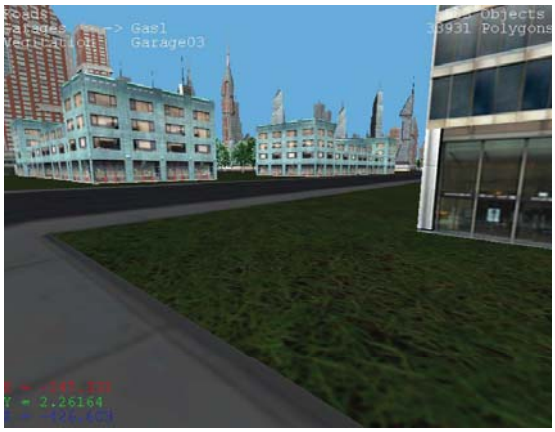
Of course levels can contain hundreds, even thousands of objects. To a game, these items might not all be physical objects. For example, sounds, way-points, or other elements that do not have a graphical representation in the game could be placed using the editor. These objects can be given an appearance such as a box with an icon painted on it to allow them to be positioned in the scene and found later. The objects available to the editor are specified in an ASCII text file called a "catalogue" file (see Figure 1(b) for an example). This allows the editor's content to be altered without the need to recompile any code. Ideally, the level editor itself would provide a graphical interface to search and edit the potentially lengthy catalogue files (the current version does not support this).

Referring to Figure 1(b), the first column of the catalogue file specifies the group that an object belongs too. Groups work in a manner similar to folders within a file system, thus providing a simple manner of organize objects. In the level editor, these groups help users find the object they are looking for. The second column contains the actual name of the individual object. The third column lists the file name and path of the model and the fourth column lists the path and file name of the texture to be applied to the model. Entering "NULL" here implies that the model is non-textured. The fifth and sixth columns specify the horizontal and vertical grid respectively that this object snaps to (objects are "snapped to a grid"). In the editor, the grid appears as a wireframe axis aligned box that can only be moved in increments equal to the box size. Grids are useful for levels that are built in sections so that models can be reused. Specifying a grid size of zero indicates that the corresponding object can be placed freely. The next three columns specify the rotational snapping about the 'x', 'y', and 'z' axes. A value of zero indicates that the object can be rotated freely about the corresponding axis, while a value of 360 indicates that the object cannot be rotated. In Figure 1(b), all of the objects have a rotational value of (360, 90, 360) implying that they can only be rotated about the 'y' axis and that their rotation is snapped to the nearest 90° increment. The last column is used to indicate the maximum number of objects of this particular type that can be placed within a level. For example, most levels can only have one player or one camera. A value of "0" indicates that there is no restriction to the number of objects allowed in the level.

A. Using the Level Editor

There is a small white box in the center of the screen that represents the cursor. The left, right, up, and down arrow keys allow the cursor to be moved (relative to the direction that the camera is facing) left, right, up, and down respectively. Moving the mouse will cause the camera to rotate about the cursor. Pressing and holding the 'I' key ("In") will move the camera closer to the cursor. Pressing and holding 'O' key ("Out") will back the camera away from the cursor. Switch to first person mode is accomplished by holding the 'I' key down until the cursor disappears. Objects cannot be added or deleted in first person mode. The exact position of the cursor is displayed in the bottom left corner of the screen. Pressing the 'X', 'Y', or 'Z' keys will toggle between the locked and unlocked state for each of the axes. When an axis is locked, the axis label is shown in black and the cursor cannot move along that axis. A wire frame preview is shown for the currently selected object type within the editor window. The previewed object rotates with the camera and the rotation of each object can be snapped if desired. Some objects should only rotate about the 'y' axis for example, and the rotation can be snapped so that objects rotate in increments of 90°.

Objects are removed from the scene using the right mouse click. An object will only be removed if it is the same type as the currently selected object. Pressing the 'Q' key allows for the cycling through of object groups while pressing the 'W' key allows for the cycling through of items within a group. Pressing the space-bar moves the cursor to the nearest instance of the currently selected object type making it easy to locate a specific object such as the player spawn point. This feature also allows the user to delete all the instances of the selected object in an area very quickly by first selecting the type of object to delete, and then pressing the space-bar followed by the right mouse button repeatedly. A sample scene created using the level editor outlining a city landscape is provided in Figure 2.



(a)

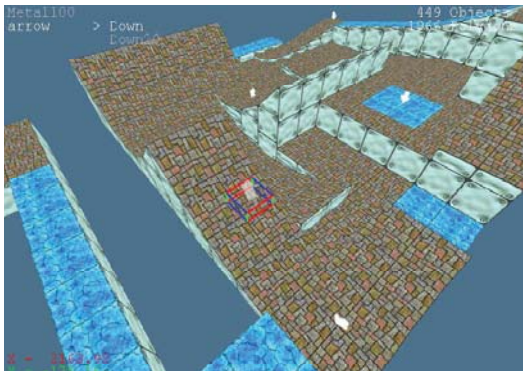
(b)

Fig. 2. Examples. Sample scene of a city landscape created with the level editor. The building models were part of the "Megacity Construction Kit" purchased from 3DRT.com. (a) First-person camera view. (b) Third-person camera view.

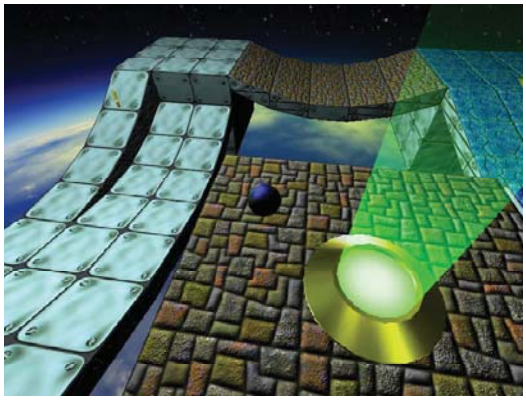
An example from *Marble Run*, a game developed using the level editor, is provided in Figure 3. *Marble Run* is a multiplayer online physics based marble game. Players engage in a friendly competition with the goal of being the first to reach the end of the level. Each player steers their "marble" by tilting a motion sensing controller (or by pressing the arrow keys on the keyboard). *Marble Run* can also be played by a single player competing against their own best time for each of the ten challenging levels. A sample scene of *Marble Run* being developed within the level editor is illustrated in Figure 3(a) while an in-game screenshot is shown in Figure 3(b).

III. Summary and Future Work

Here we have presented a level editor that is specific to editing levels (e.g., creating three-dimensional scenes) only and therefore, game engine-specific tasks such as lighting and modeling are not included. We have also presented *Marble Run*, a game with all of its levels created solely using the level editor. We believe that separating level editing software from game engines provided greater choice while reducing costs. Future work will include the addition of a GUI-based interface to edit the catalogue of objects, and the ability to



(a)



(b)

Fig. 3. The *Marble Run* game developed using the level editor. (a) Sample scene created with the editor. (b) In-game screenshot.

allow entire levels to be exported as OBJ models. Scripting will also be added and a terrain editor will be incorporated allowing for easy terrain manipulation. Finally, future work will also include networking capabilities to allow for remote collaborative level development. In a networked version, when a user logs in, they would be sent the current version of the level from the host. Each user will be able to the other cursors associated with other users moving about the environment with the corresponding user names over the appropriate cursor. Users will be able to add, edit, and delete objects simultaneously and chat in real time.

Acknowledgment

The financial support of the *Natural Sciences and Engineering Research Council of Canada* (NSERC) in the form of a *Post-Graduate Scholarship* (PGS B) to Brent Cowan and a *Discovery Grant* to Bill Kapralos is gratefully acknowledged.

References

[1] E. Byrne. *Game Level Design*. Charles River Media, Hingham, MA. USA, 2004.

[2] V. Miliano. Unrealty: Application of a 3D game engine to enhance the design, visualization and presentation of commercial real estate. In *Proceedings of Virtual Reality in a New Virtual Millennium*, Dundee, Scotland, September 1-3 1999.

[3] G. Smith, T. Salzman, and W. Stuerzlinger. 3D scene manipulation with 2D devices and constraints. In B. Fisher, K. Dawson-Howe, and C. O'Sullivan, editors, *Virtual and Augmented Architecture*, pages 35–46. Springer, 2001.