3D Mobile Game Engine

A Major Qualifying Project Report

Submitted to the University of

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Bachelor of Science

By:

Aaron Root

Christopher Donnelly

Aleksander Yeganov

Date: April 21, 2008

Approved:

Professor Emmanuel O. Agu, Project Advisor

Contents

Abstract

This project explores the feasibility of a mobile 3D game engine on a smart device. Using OpenGL ES, a graphics library for embedded devices, we have developed Tubetris, a unique and entertaining fully three dimensional puzzle game which runs on a HP iPaq PDA . The game demonstrates the 3D capabilities of mobile devices and that fully 3D applications are indeed possible on mobile devices.

1 Introduction

1.1 Mobile Devices

Recently mobile devices such as smart phones and personal digital assistants have become increasingly popular. As their popularity increased, so has their functionality and power. Today, mobile devices can run a variety of applications covering a wide range of functions. It is even coming to the point where mobile devices can rival desktop computers in terms of functionality and usefulness.

Even though mobile devices have become incredibly useful, currently one of their primary functions is entertainment. People use their PDAs to listen to music or play video games on their cell phones while waiting for busses or trains. The quality of these games, however, has hardly changed even with all of these technological advancements. The resources and the tools to change this exist, but there are few people doing so.

This project hopes to change that. By developing a 3D game engine for mobile devices, this project should demonstrate that more advanced video games are indeed possible. The goal of the project is to develop a fully 3D game engine and to create a game that is both unique and fun to play.

1.2 Tubetris

The game that we chose to create to solve this problem is called Tubetris. It is a 3D puzzle game similar to Tetris. The primary difference is that instead of a flat, two dimensional playing board, the board is wrapped around a cylinder. Random pieces fall from the top of the tube, and the player rotates the tube to place the pieces. The goal is to complete rows before the tube fills up, which ends the game.

Tubetris demonstrates the 3D capabilities of mobile devices very well. It inherently uses three dimensions as a game play element. It also has very simple controls which works well with the limited inputs of most mobile devices. The game play is intuitive and fun, which makes for an entertaining video game.

1.3 Goals

The goal of this project is to create a well designed 3D game engine for mobile devices. This will require a knowledge of software architecture, software engineering, algorithms and computer graphics. By doing this we hope to create a 3D game for mobile devices which is both open to further extension and prove the feasibility of creating fully three dimensional video games on mobile devices.

2 Background

2.1 OpenGL ES

OpenGL ES is a graphics API based on OpenGL maintained by the Khronos Group. It is specifically modified for embedded systems, and is used in a variety of applications, including the iPhone, the Playstation 3 and on some avionics equipment.

2.1.1 Versions

There are several versions of the OpenGL ES specification, and each is defined in relation to a different OpenGL specification. OpenGL ES 1.0 is defined relative to OpenGL 1.3 ES 1.1 is defined by OpenGL 1.5 and ES 2.0 is defined relative to OpenGL 2.0. OpenGL 2.0 features a programmable graphics pipeline and as such is not backwards compatible with previous versions.

OpenGL ES 1.1 emphasizes hardware acceleration of the API, and is fully backward compatible with 1.0. It provides enhanced functionality, improved image quality and optimizations to increase performance while reducing memory bandwidth usage to save power. The OpenGL ES 1.1 Extension Pack is a collection of optional extensions added to OpenGL ES 1.1 that reduced variability and bring significant improvements in image quality and performance.[3]

OpenGL ES-SC 1.0 is defined relative to the OpenGL 1.3 specification and is designed to meet the needs of the safety critical market in Avionics, Industrial, Military and Automotive applications including D0178-B certification.[3]

The game we were going to develop needed to be as much performance friendly as possible and consume as less battery power as possible. OpenGL ES 1.1 fit the category perfectly.

2.1.2 OpenGL vs OpenGL ES

There are several differences between OpenGL and OpenGL ES, however there are also many similarities. The main difference is that OpenGL ES has had several of OpenGL's less efficient features removed. For instance, OpenGL ES does not support OpenGL's GLBegin and GLEnd functions. Depending on the version, OpenGL ES may also lack support for floating point data.

One of the most notable differences between OpenGL and OpenGL ES is the lack of the GLBegin and GLEnd functions. In OpenGL, an object is normally drawn to the screen by calling GLBegin and then specifying vertices and normals terminated by a call to GLEnd. However, OpenGL ES does not support this. In order to draw an object to the screen in OpenGL ES, it is necessary to first create an array of fixed point numbers which represent vertices. A pointer to this array is then passed to OpenGL and it is drawn when a certain draw function is called.

Another important difference is that OpenGL ES supports less primitives than OpenGL. While OpenGL supports drawing vertices as polygons or quads, OpenGL ES only supports drawing triangles. This means that special care must be taken to draw vertices in the proper order . Failing to do so can cause unexpected results.

2.1.3 OpenGL ES Implementations

There are several different implementations of the OpenGL ES standard. The one that we chose to use was the Vincent Mobile 3D Rendering Library [8]. We chose this implementations specifically because it works well with both PocketPC and Microsoft Smartphone. It is free for both commercial and non-commercial projects and comes standard with window management functions, which increases the ease of use.

Another implementation that we considered using was the Hybrid "Rasteroid"

implementation[10]. Some benefits of Hybrid over Vincent is that it has faster frame rates and a more precise depth buffer. However, we were unable to find a current version of this implementation as it is apparently no longer supported. Another reason we decided not to use it was that it is mainly for developing with BREW, which we were not using.

2.2 GLUT

GLUT stands for GL utility toolkit. It provides many useful tools for working with OpenGL.

2.2.1 Features

GLUT provides a simple way to handle both keyboard and mouse input. It also provides functions to easily define and control the viewing window. Finally, it provides functions for drawing several primitives, including cylinders, cubes and spheres, in both solid and wireframe modes. Our cylinder drawing algorithm is a highly simplified version of the GLUT implementation.

2.2.2 GLUT ES

GLUT ES is a version of GLUT designed specifically for OpenGL ES. It is essentially the same except that it does not include support for some GLUT functions. For instance, it does not support the drawing of several primitives. It also lacks support for some input devices which cannot be used on mobile devices, such as joysticks.

2.3 Fixed Point Math

PDA and most other embedded devices have low memory, slower CPU speeds and very low power consumption requirements. It is then no wonder that FPP, or a floating point processors have not become a part of the standard mobile platforms. As such, on many mobile

devices, fixed-point math is preferred especially when FPP hardware is not available.In this section we are going to explain what is a fixed point variable and how it is different from floating point, along with the trade-offs that are incurred for the performance gain.

Fixed point math in a nutshell is a technique that uses integers to perform floating point calculations. Fixed point value is represented by an integer. The integer is then divided into two parts, one of them is going to represent the integer part and the other would represent the decimal part. Depending on the kind of a project the programmer is working on he/she can choose one format for the fixed type. For example in an 8-bit integer in a 2's complement format 1 would appear like this

1 = 0001.0000

First bit indicates the sign of the number, the following 3 bits are used for the integer part and the last 4 bits are used for the fractional part. OpenGL ES uses the format where the first bit indicates the sign of the number, next 15 bits show the integer part and the last 16 bits are the decimal part.

To convert a normal integer to a fixed point, one simply needs to shift forward 16 bits, in c it looks like this:

```
1 << 16;
```

Special care has to be taken when converting floats or doubles into fixed point. Because floats are stored in a IEEE format, simply shifting them would destroy the value. To resolve that problem a little bit of math is required. If the chosen format in example was 20 bits for integer and sign and 12 bits for decimal precision, which in a short form can be expressed as 20.12. Then to represent 1 float as fixed one would need to calculate the offset value first:

$$offset = 2^{12} = 4096.$$

The general formula for the offset number:

2^decimal bits

Then a simple multiplication by the offset number would convert the float into a fixed number.

Fixed = float * offset.

Converting back to the integer is simple:

integer = fixed >> 12.

The addition and subtraction for fixed point work the same as for regular integers, but some care needs to be taken not to overflow the integer holding the fixed value.

Another aspect of fixed point numbers that should be considered is multiplication. Whenever multiplication occurs computers sees each number with an additional offset. Here is a detailed explanation:

if the number one were to be multiplied by itself in fixed format of 4.4 then internally it would be stored as 10000. Therefore multiplying 10000 * 10000 = 100000000. The answer appears to be obvious at this point. By removing the extra 4 zeros everything would get back to normal. In C shifting backward by the offset would solve the problem.

A similar problem arises in division. except that in the case of division the number to be divided needs to be shifted forward the offset number of bits, because after the division the final number must retain one offset.

2.4 Windows CE

Windows CE is a very light weight operating system targeted at mobile devices. It is the smallest from the Microsoft Windows OS family. It was initially designed to be power-efficient and to support Win32 subset API. The main goal was to extend the Windows API into machines that cannot support the relatively power hungry Windows Vista or XP. Windows CE is

not backwards compatible with MS-DOS or even other Windows'.

2.4.1 User Interface

As mentioned previously, we decided not to use the OpenGL ES glut ES toolkit because we wanted to provide a more native-looking interface. Using the native Windows API gives better flexibility and control. It also made the design of menus and other input controls easier.

2.4.2 Windows CE and OpenGL ES

Windows CE is an event-driven operating system. Before setting up OpenGL ES or being able to capture any input from the user we needed to set up an initial window.

Writing Win32 applications the program starts in the WinMain function. That function has four parameters: a handle to the instance, a handle to the previous instance, a command line and the last parameter specifies how the window is to be shown. One of the most important parts of a windows program is the message loop.

Because Windows CE is event-driven the application window gets all of its input through the operating system. Whenever the user hits a key the operating system receives an event and a message is generated, which is placed in the message queue. The message loop constantly polls the message queue for new messages and when it receives a message that it recognizes, or in other words a message for which the code has been written that code gets executed. When the window is being created a handler function must be specified. That function handles the messages which the loop constantly dispatches to it. When no messages are generated the OpenGL ES code would be executed.

OpenGL ES requires a special setup. To be able to draw with OpenGL in Windows three variables are required – EGLDisplay, EGLSurface and EGLContext. EGLDisplays provides a bridge between the applications window and OpenGL. EGLSurface is the buffer that is going to

be shown on screen and finally EGLContext allows to draw into the EGLSurface. We have a render() function that after all the setups get completed is executed.

2.4.3 Improved Message Loop

As it was explained above, the handler function specified during the creation of the application window handles the messages generated by the events. The usual set up is that this function contains a switch statement, where each message code is captured and the appropriate code is executed. After some research a better approach was found. That approach allowed one to remove the clogging switch statement and move the message handling routines into separate functions. In order to accomplish that, we created a structure of the following format:

```
struct decodeMSG
{
        UINT code;
        LRESULT (*Fnctn)(HWND, UINT, WPARAM, LPARAM);
};
```

The first member variable is the message code and the second member is a pointer to the handler function for the given message code. In the source file that contains the WinMain function we created a global array of docodeMSG structures, where each structure contained the message code it was responsible for and the handler function for that message code. Now whenever an event is generated and gets sent to the main handler function a for loop runs through the the array of messages codes and if one of them matches the events code a function associated with that code gets executed.

2.4.4 Symbian OS vs WinCE

Early in development we had to choose which operating system we were going to design our game for. The two leading mobile operating systems were Symbian OS and WinCE. Symbian OS is a proprietary operating system owned primarily by Nokia but also used by several other companies. Symbian OS is actually the leading OS for mobile smart devices. However, programming for it would have required a lot of extra learning on our part. In the end we chose WinCE mainly because we were already somewhat familiar with coding on Windows.

2.5 Development Environment

The development tools available on Microsoft Windows operating system platform are numerous. Choosing the right one for our project that allowed our team to utilize all of the resources and efforts most efficiently was not an easy task. The sections below describe the programming environments we had gone through and how we eventually decided to use them.

2.5.1 Eclipse

Eclipse is an open source community. Its projects are focused on building an open development platform with extensible frameworks and tools for building, deploying and managing software. The Eclipse Foundation is a not-for-profit corporation with supporting members that hosts the Eclipse projects and helps to cultivate open source community.

The original creator of the Eclipse project was IBM. Eclipse was created in November 2001 and supported by some software vendors. Later in January 2004 the Eclipse Foundation was created as an independent not-for-profit corporation to act as the steward of the Eclipse community. This independent not-for-profit corporation was created to allow a vendor neutral and open, transparent community to be established around Eclipse. Today, the Eclipse community consists of individuals and organizations from a cross section of the software

industry [2].

Even though Eclipse is known for its Java IDE, it also allows one to code in almost any main-stream language available on the market today. In addition it has many features that make coding a lot easier. Code can be formatted, auto-indented and for each language Eclipse supports an individual "perspective". What makes Eclipse even more alluring is that it supports many plug-ins that extend its functionality even further. Perspectives are specific settings that utilize most common options pertaining to a certain language more easily available. Another interesting idea that is supported by Eclipse are workspaces. Workspaces are basically containers for projects, they can be used to separate production environment from a testing environment, which is very useful. One can simply copy the latest projects from the development area into testing area and experiment on them without a need to create a separate project, or even worse changing the actual development code.

2.5.2 Embedded Visual C++ 4.0

A product of Microsoft, eVC is a stand-alone development environment that supports languages C and C++. It allows to build "native" executables. One of the important features that eVC supports is the MFC library. MFC stands for Microsoft Foundation Classes it is a library that wraps portions of Windows API in C++ classes. These classes are defined to give control over many of the handle-managed Windows objects and also for predefined windows and common controls. Another feature that is of interest to programmers is support for WTL and ATL in eVC. WTL, or Windows Template Library provides support for easy implementation of user interface elements, such as frames and popup windows, standard and common controls, property sheets and common dialogs etc. The main objective of WTL is small and efficient code. That was important for our project, because for our user interface we decided not to use libraries that are

available with OpenGL ES. ATL, or Active Template Library is a set of classes that simplify programming of Component Object Model(COM). Since, neither ATL nor COM had any significant importance to our project I am not going to elaborate more on them other than to say that they are related to Interprocess Communication and dynamic object creation.

2.5.3 Visual Studio

First released in 1997, Microsoft Visual Studio is a bundle of programming tools available from Microsoft tied together by one common Integrated Development Environment. Using visual studio programmers can create standalone applications, web services, web applications and web sites, which could be run on Windows, PocketPC and smart devices. Currently visual studio includes: Visual Basic, C#, C++, J#. Separate editions also include Microsoft SQL server for developers. It should be mentioned that Visual Studio targets .NET framework. Microsoft .NET framework consists of a Base Class Library and Common Language Runtime. Base class library contains many precoded solutions to common problems that programmers face. CLR manages the execution of the programs. One of the main advantages of .NET framework is that CLR provides a virtual machine abstraction, which means the programmers do not need to be concerned with the CPU that will execute their program.

2.5.4 Choosing an IDE

Because programs developed for embedded systems can not be run on desktop PC the deciding factor in choosing the right IDE depended on the emulator support. The Eclipse had all the features that we needed except the native emulator support. Embedded visual c++ had support for emulators, but the emulator that came with it was outdated – therefore was not the best choice for development. Visual Studio supported the latest emulators available, it also had most of the features available in Eclipse and was chosen based on that criteria. Yet not all of the

problems had been solved – to keep our source code clean and not to interfere with each other during development we needed a source code management system.

2.5.5 Source Code management and SourceForge

For our source code management we decided to use more than just a repository. WPI offers its students access to SourceForge, which gives one not only a repository but much more. It allows one to create tasks and documents, which could be tracked by other team members.

The only available source code management repositories available in WPI SourceForge were CVS and SVN.

2.5.5.1 CVS VS SVN

CVS – Concurrent Versions System is an open source version control system. SVN – subversion is a version control system created initially by CollabNet inc. Its main purpose at its inception was to create a free version control system that would operate a lot like CVS, but without the CVS's "bugs" and flaws. After some research we found certain advantages and disadvantages to both systems. Fortunately our project was not going to be extremely large so that we could be affected by the differences between the two. Initial decision was to use the Subversion because it appeared easier to install and maintain.

First client we tried to use was the TortoiseSVN, which provided seamless Windows Explorer Integration. Unfortunately after installing it we could not get it to work because of an internal "bug". Another Windows SVN client was Subversion Packages, but it also did not work. After successful installation it refused to connect to the repository in SourceForge. Since SVN proved to be too time consuming we turned to CVS. The research showed that there were not many CVS solutions offered on the Windows platform for the environment we chose. One of the most prominent was "Jalindi Igloo", surprisingly old, yet still recommended client – last release

was in April of 2002. After installation it was available in the VS environment. Upon an attempt to connect to the repository it failed.

To avoid spending too much time on repository setup one of the professors at WPI suggested using Eclipses built-in CVS and use it only to check-in and check-out source code while still developing in Visual Studio. That approach was tested and it worked therefore was left as the final setup.

2.5.6 Active Sync

At early stages of development we discovered that the emulator was extremely slow and slowed down the development process. Professor Agu supplied us with a pocket PC, which was used for testing most of the time from then on. ActiveSync is a software provided by Microsoft that allows to synchronize the data between the pocket PC and desktop PC. For developers it mainly serves as the gateway between the development environment and pocket PC.

2.6 Tetris

The game which we designed is similar in style to a currently existing game called Tetris. Tetris was designed in 1985 by Russian programmer Alex Pajitnov and was eventually released in the United States through Nintendo packaged with Game Boy. The game is played with a 4 unit piece called a tetra is dropped from the top of the screen in one of five unique shapes. Pieces are scored when a row is completed, and if any of the tetra is off the screen when the piece comes in contact with another piece, the game ends[1]. It takes place on a flat game board as pieces fall from the top towards the bottom.

2.7 Tetris-Style Games

Since the advent of Tetris and its popularity, there have since been many variations of Tetris which have been made. There have been variants with the same rules, but the piece

generator gives the least useful piece [5]. In addition there have been three-dimensional versions [7] and other simple variations. Usually these have been on dedicated devices for playing games, but have been ported to various systems.

2.8 Games on PDA's

There have also been a variety of games on PDA's which have been made over time. This has been due to PDA game's rising popularity. While many games attempt to have 3-dimensional, such as a top-down version of tetris[4], and a version of the popular game "Doom" [6], to the current date, there have not been many attempts at real 3-dimensional games for embedded systems.

3 Design

3.1 Graphics Design

During the development of our game, the design for the graphics elements changed considerably. These changes were made to improve execution time, reduce development time and to generally simplify the code.

3.1.1 Original Design

Originally, the graphics were going to be handled by several different classes(Fig 3.1.1). There was going to be a drawing class, which utilized both a camera class and a model class. The camera was to be responsible for setting the window and rotating around the game board. It was partially impemented, but then was removed. The reason it was removed was because it turned out to be faster to simply draw the pieces with an offset each time they were redrawn. Since the entire board had to be redrawn every time a piece moved anyway, this could be done with minimal overhead.

The model class was initially supposed to keep track of each model so that they could be drawn more quickly. The idea was, instead of generating many meshes and placing them together to form a piece, one mesh would be generated for each piece. This would trade off memory for speed, however it was decided that memory was more important than speed. Also, it was found that the model class had many of the same responsibilities of the cobli class. In the end, neither of these were needed by the main graphics class since it just directly reads the game board. This is efficient for both memory and execution speed.

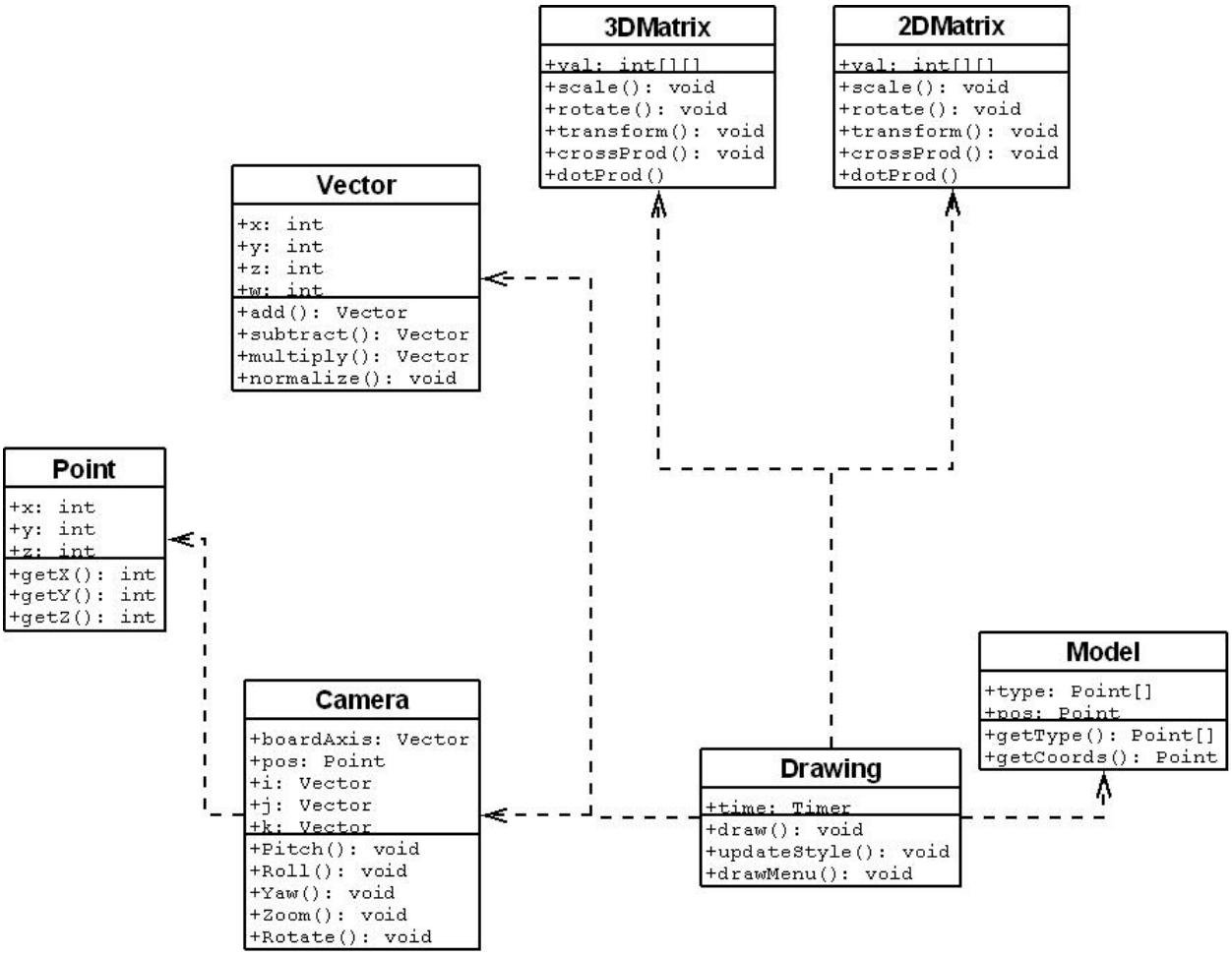

Figure 3.1.1 Class Diagram of Proposed Graphics Engine

3.1.2 Optimized Design

Instead of using a more complex graphics engine which would use multiple classes, we condensed all of the functionality into a single source file. The graphics element of the engine controls all of the visual elements of the game. This includes drawing the board to the proper parameters, drawing the pieces, and drawing the background. The graphics portion of the engine was designed to accept any size board and procedurally generate all of the required meshes based on that configuration. Another consideration was to allow the graphics portion to be able to easily handle multiple themes. The board was designed to be a cylindrical grid, split both

horizontally and vertically. One block in the cylinder occupies one of these cylindrical grid squares. This creates an easy to view interface that is simple to understand. It also makes it easier to position pieces over gaps because of this grid nature.

Another important design consideration was to make the graphics easy to understand. The board was designed in such a way that a player would have a good understanding of where all the pieces were and where to put new pieces. It was decided that the board should consist of two

Figure 3.1.2: An example screenshot

transparent cylinders, viewed at a slightly downward angle. This made the front as well as the back of the board easily visible in most situations. It was also decided to use a background image

that made the pieces stand out well against so that they are easily visible. An example of the game display can be seen in figure 3.1.2.

It was also important that the graphics use as little memory as possible and run as quickly as possible. If the graphics refreshed too slowly, then the game will be too difficult to play. To solve this the engine was designed to only have to generate most meshes once to increase performance and decrease resource usage. Also, meshes were kept at an optimal size to save memory and decrease redraw times.

Finally the graphics were designed to be able to support a variety of themes. Both the background image as well as the color and texture of the pieces can change whenever necessary. This design allows for the game to be easily modified in the future to allow any sort of graphical configuration one would wish.

3.2 Choosing a Genre

According to Mark J. P. Wolf in The Medium of the Video Game, a way to define a genre of a video game can be either from theme, iconography, or structure. Using these, a list of many of the genre's of videogames are "Abstract, Adaptation, Adventure, Artificial Life, Board Games, Capturing, Card Games, Catching, Chase, Collecting, Combat, Demo, Diagnostic, Dodging, Driving, Educational, Escape, Fighting, Flying, Gambling, Interactive Movie, Management Simulation, Maze, Obstacle Course, Pencil - and - Paper Games, Pinball, Platform, Programming Games, Puzzle, Quiz, Racing, Role - Playing, Rhythm and Dance, Shoot 'Em Up, Simulation, Sports, Strategy, Table - Top Games, Target, Text Adventure, Training Simulation, and Utility."[9] Seeing as the PDA had restricted memory and had a goal of a 3-dimensional game engine, it was decided from this list to make a game which is an abstract puzzle game. Abstract games are "Games which have nonrepresentational graphics and often involve an objective

which is not oriented or organized as a narrative." [9] This would be useful; for the lack of a theme would help keep the graphic's simpler by not modeling things such as humans or other complicated shapes. Puzzle games are "Games in which the primary conflict is not so much between the player-character and other characters, but rather the figuring out of a solution, which often involves solving enigmas, navigation, learning how to use different tools, and the manipulating or reconfiguring of objects."[9] This would allow for no actual physics, and simple rules to follow.

3.3 General Concept

Tubetris is an abstract puzzle game in which pieces called cobli's, composed of between 2 and 5 colored spheres, fall from the top of the screen to the bottom. The board being played upon is an eight by eight grid, wrapped into a tube. Whenever either an entire row is filled in those pieces are removed and are scored. The game keeps advancing until the pieces pile up beyond the top of the grid, at which point the player loses the game.

3.4 Gameplay

Each cobli dropped is between 2 and 5 connected spheres (no sphere in a cobli will not be touching another sphere). As each cobli falls, the player is able to rotate the piece and move it around the tube so that it may land in accordance with where and how the player would like it to land. If a piece moves to a location on the tube that is not easily visible, the tube rotates so that the piece is easily visible. When the entire circumference (in one row) is filled with spheres, the pieces are removed and scored. After spheres are cleared, any piece not touching any other sphere will fall until it lands on top of either the bottom of the board or another sphere. After a

piece lands, it becomes a part of the board and another piece is released from the top.

3.5 Scoring

A player starts the game with zero points. When a cobli causes it's spheres to be arranged on the board so that the circumference of the tube is filled in a single horizontal line, scoring takes place. Each sphere in the line is worth 10 points, and is added together to the score. It is easy to change the score of a sphere, only needing to change the points value in "constants.h"

3.6 Theming

The game does not have any direct theming but the background and textures can change. It can go from a "tetris" feel and look, to a futuristic one, to ancient Greek all in the same game.

4 Implementation

4.1 Graphics

4.1.1 InitDraw

The InitDraw function initializes OpenGL ES so that drawing can begin. It is based a pointer to the game board, so that the graphics can check the board at any time and draw it accordingly. It also calls the functions which load all of the necessary textures from files. The InitDraw function also sets the proper states to allow transparency and shading.

4.1.2 Render

The Render function is called every time a change to the board is made. It is a relatively simple function that clears the screen, loads the identity matrix and then draws the background if necessary. It then calls drawBoard, which does the majority of the graphical work. Finally, it swaps the buffers so that the changes made are visible.

4.1.3 DrawBoard

The drawBoard function is the core of the Render functions. It references the current game board and draws two cylinders according to the size of the board. The cylinders are approximated by using the parametric formula for a cylinder. Points are generated based on the parameters given and the cylinders are drawn. One cylinder is smaller than the other which creates a tube. Next the function reads the game board and generates pieces as they appear. The function automatically calculates the offsets for each piece so that they appear in the proper position. This effectively wraps a two dimensional board onto a three dimensional cylinder. The function that generates the pieces only creates the mesh for them once, and then sets a boolean to true. Every successive call to the function simply displays the already generated mesh. This

speeds execution time and makes the graphics smoother. After all of the pieces on the board have been drawn the function exits.

4.2 Game Elements

4.2.1 Custom Names

The first thing created was a global file which defined names such as RED_ and 'r' and BLUE_ and 'b'. This would allow easy changes further down the line by changing only one character rather than changing everything. This was stored in constants.h and every file was allowed to have access to this.

4.2.2 Spheres

The game spheres were created as a custom data type called sphere and stored in the files sphere.cpp and sphere.h. Sphere's had three distinct attributes, being the color of the sphere, the type of the sphere (multiplier or normal), and the value of the sphere.

4.2.3 Globals

For the game, a set number of sphere's were created which would exist throughout the entire game, and only these sphere's would be used. These were colored in one of 5 colors: RED_, GREEN_, BLUE_, YELLOW_, and ORANGE_. The score for each sphere was set to 10 and the type of sphere was always regular. These were the sphere's pointed to in all subsequent operations. In addition to the 5 colors, there was a NONE_ sphere which would be used when no sphere was actually present.

4.2.4 Game Tube

The game tube was designed as a two dimensional array of pointers to spheres in order to save on memory. The default size used was eight by eight, but it could be easily changed by

entering different numbers into the constructor. This two dimensional array was drawn as a tube and any cobli movement was simulated in order to wrap around the board properly. The playing field itself was stored in board.cpp and board.h, and had methods to return the array as was able to be seen.

4.2.5 Cobli

A cobli was represented by a 5x5 array of pointers to spheres, and was stored within cobli.cpp and cobli.h. It also had it's internal representation of the x and y location relative to the board such that the the lower lefthand corner of the array would be the indicator. The generation of a cobli consists of initializing all of the pointers to NONE_, and then started at the location of 2,2, otherwise known as the center of the grid. A randomly colored sphere was placed there, and then a random direction was chosen and placed there. This was done 3 more times such that a maximum of 5 pieces could have been placed. If a piece is overwritten, it would just mean the number of pieces were smaller. A cobli could also be rotated, which would rotate around the center of the 5x5 array, based on a simple formula on which direction it was moving. To rotate clockwise, it would would turn [x,y] into [y, x-4] and to rotate counterclockwise it would turn [x,y] into [y-4, x].

4.2.6 Interactions Between Game Tube and Cobli

In order to keep graphics and game play separated as much as possible, it would be optimal that there was a function that graphic's could call that would return the current game state as visible by the player. This would be a graphical representation of the game board with the cobli on it rather than having the graphical package take both pieces of information and putting them together itself. To have this done easily, the cobli is treated as part of the game board and when the game board state is returned, the cobli is returned as part off that array. As

an easier method of doing this, it was determined that every time the cobli moved, it would be placed on the board, to be removed whenever the cobli started to move. Whenever a cobli hit's the bottom of the board or another sphere in the downward direction, it would be placed down permanently such that it would not overlap with any previously existing colored sphere, and then reset to the same column it was in, but at the top of the playing field, recreated as a new cobli. The board is also the location which actually calls for the cobli to rotate, drop, and move left or right.

4.3 Scoring

After each cobli hits the location for it to stay still, a simple for-loop is run to determine if any of the spheres completed an entire row and should be cleared. Should a row be cleared, that part of the for-loop is run again with a boolean now turned on which replaces each colored sphere with one labled NONE_. As it does that it has a function which adds the sphere's score to the temporary score value, and after the loop is done it adds the temporary score to the permanent score. In addition, every row above the row which just got cleared is lowered by a row so the game play can continue easily.

4.4 Ending

The game ends when a cobli cannot move downwards due to a sphere or ground being in the way and a sphere is above the edge of a board. This is checked by seeing how far down the cobli has fallen and if the cobli has any sphere's other than one labled NONE_ above the board edge. The game then calls a message which tells the player the game has ended.

5 Results

We were able to produce a playable 3D video game. The game supports different board sizes, although that option is not available in the game it is implemented. The pieces in the game have different colors. The game score is being calculated as the player progresses in the game, but it is not displayed. The main controls are the arrow keys and the enter button. The left and right keys rotate the tube left and right, the up key rotates the piece and the down key makes the piece fall down faster. The enter key pauses the game. Partial support for a stylus is implemented. At the current state the player can turn off textures if he/she wishes to do so by touching the screen with the stylus, a future extension of using stylus to rotate the tube while holding the stylus is left for future work. The games menu allows one start a new game and exit the game completely. Also the menu bar has an alternative exit button in the form of a cross in the upper right side of the game window.

6 Future Work

6.1 Definition of Future Work

The future work which is about to be described is possible work which could be done to expand upon this project.

6.2 Modifications to Basic Game Play

There are many simple modifications which could be applied to Tubetris which have not been applied as of the current date. It could be designed such that the sphere's color has a certain value, and those values are the score that is added as opposed to the basic score of 10 points. While support for this has already been added, an implemented version of having a multiplier sphere would be a nice addition so that when the sphere is picked up, the points are doubled rather than adding the 10 points for the sphere. The scoring could also be modified so that columns would have spheres deleted if 4 of the same color (or a different number) were in the same column, all touching each other or if multiple lines were cleared at once, more points would be gained than just the sum of the two lines.

6.3 Modifications to Structure

There are many ways to modify the structure of the game for future projects also. A configuration file with the types of spheres created on start up and their values would be a decent addition to the game. In addition, the configuration file could define the size of the board, and which rules for play are used. To take this further, a scripting language could be created so that the game can be modified into several different games, and can generate many different puzzle games with a variety of rules. There could be modifications to cobli's such that the size is not limited to 5 pieces, and

possibly create all possible 5 combinations instead of the few the algorithm which was written creates.

6.4 Differing Systems

Other possible expansions would be to port the current game into either newer or older palm pilots than currently are in use, or to try to port the game to a cellphone. Newer palm pilots would allow for a more complex game while an older palm pilot would have to be more restricted in what it had to do. Cell phones would also be more restricted, but would also have to have different button mappings based on the phone being used.

6.5 Different Games

A final idea for possible directions for the project to evolve would be to create a different kind of game than abstract puzzle to show other capabilities that a 3d engine could perform on embedded systems. The current engine which was designed only runs a specific type of puzzle game, and an expansion on that could prove palm pilots a strong contender as a current game system.

6.6 Modifications to graphics

While the graphics work for the game, there are many graphical improvements which could also be done for future work. Better texturing can be added, so the spheres could look like various objects (planks of wood, bricks, etc.). The shapes can be made different so spheres could actually be a sphere shape. Graphics could be redone using GLUT (if one was daring enough) or other versions of OpenGL ES.

7 Conclusions

Working on this project taught us many things. One of the things that we learned was the importance of communication when working on a project of this magnitude. Through the use of sourceforge, instant messaging, and regularly scheduled meetings we were able to make sure that all team members were well coordinated. This project gave us all valuable experience in working with a team.

We also learned the importance of optimized code. When programming for modern computers memory usage and processing time is not usually a huge concern. However, when programming for mobile devices it becomes necessary to write code that both uses limited memory and executes quickly. The tricks and practice we gained by doing this will easily translate over into any programming project.

Another conclusion can be drawn regarding the future of 3D games on mobile devices. While the game we made was limited by the hardware available to us, the PDA that we ran our game on was several years old. Modern mobile devices have much more power and memory. Some even have dedicated graphics chips, which would have accelerated our game substantially. It is likely that in the near future, much more advanced 3D games will be made available on mobile devices. All of the elements necessary to their development exist, it is simply a matter of actually creating them.

It is also interesting to note that several of the delays we had were simply because the IDE was not cooperating. In particular, Microsoft Visual Studio was exceedingly difficult to work with. It crashed frequently and would not work with SVN. The errors encountered ranged from actual, known bugs in the compiler to random corruption of necessary files. It is astounding how poorly such a widely used piece of software performed.

8. References

[1] Bellis, Mary. "History of Tetris" About.com. 2008. Internet.

<http://inventors.about.com/od/tstartinventions/a/Tetris.htm> 17 April, 2008

[2]Eclipse. Eclipse.org home. 2008. Internet.

<http://www.eclipse.org> Janary 27, 2008.

[3] Khronos Group. The Khronos Group. 2008. Internet

<http://www.Khronos.org> 19 April, 2008

[4] Komilev, Stefan. PDA 3Dware. 2001. Internet.

<http://www.pda3dware.com> 26 March, 2008.

[5]Poloni, Federico. "bastet" happypenguin.com. 2005. Internet.

<http://www.happypenguin.org/show?bastet> 17 April, 2008

[6] PalmInfocenter. "Palm OS Software" 2008. Internet.

<http://palminfocenter.com> 26 March, 2008.

[7] Tetris Holding. Tetris.com. 2008. Internet

<http://tetris.com> 18 April, 2008.

[8] Vincent Mobile 3D Rendering Library. 2008. Internet

<http://sourceforge.net/projects/ogl-es/> 18 April, 2008.

[9] Wolf, Mark J.P. The Medium of the Video Game. Texas. University of Texas Press., 2000.

[10] Zeus Communications Zeus CMD. 2008. Internet.
<http://www.zeuscmd.com/tutorials/opengles/02-SettingUpYourEnvironment.php>
18 April, 2008.

Appendix A - Installation Guide

To install the game to a PDA or Smartphone, it must have WinCE installed. Simply copy the game executable, as well as the bg.bmp file to any folder on the device. It does not matter what folder they are placed in as long as they are both in the same folder. Then copy the libGLES_CM.dll file to the windows folder of the device. Opening the executable should then run the game.

Appendix B - Code Documentation

Render.h File Reference

```
#include <windows.h>
#include <GLES/gl.h>
#include <GLES/egl.h>
#include "resource.h"
#include "board.h"
```

Defines

#define	**PI** 3.14159265358979323846f

Functions

GLfixed	**FixedFromInt** (int value)
GLfixed	**FixedFromFloat** (float value)
GLfixed	**MultiplyFixed** (GLfixed op1, GLfixed op2)
bool	**InitOGLES** ()
bool	**InitDraw** (**board** *a)
void	**Render** ()
void	**SetOrtho** ()
void	**SetPerspective** ()
void	**Perspective** (GLfloat fovy, GLfloat aspect, GLfloat zNear, GLfloat zFar)
void	**initBlock** (GLfloat inner, GLfloat outer, GLfloat height, GLint slices)
void	**drawCylinders** (GLint slices, GLint stacks)
void	**cap** (GLfloat innerRadius, GLfloat outerRadius, GLint slices)
void	**Cylinder** (GLfloat baseRadius, GLfloat topRadius, GLfloat height, GLint slices, GLint stacks)

Variables

const unsigned int	**PRECISION** = 16
const GLfixed	**ONE** = 1 << **PRECISION**
const GLfixed	**ZERO** = 0

Define Documentation

```
#define
PI  3.14159265358979323846f
```

Function Documentation

```
void cap( GLfloat innerRadius,
          GLfloat outerRadius,
          GLint   slices
        )
```

Draws the disk at the top of the game cylinder

Parameters:

innerRadius	The radius of the inner cylinder
outerRadius	The radius of the outer cylinder
slices	The number of slices in the cylinders

```
void Cylinder( GLfloat baseRadius,
               GLfloat topRadius,
               GLfloat height,
               GLint   slices,
               GLint   stacks
             )
```

Draws a cylinder with the given dimensions. A difference between the top and bottom radius creates a cone

Parameters:

baseRadius	The radius at the bottom of the cylinder
topRadius	The radius at the top of the cylinder
height	The height of the cylinder
slices	The number of slices in the cylinder, the more slices, the more cylindrical the approximation
stacks	The number of stacks in the cylinder

void drawCylinders (**GLint** ***slices*****,**
GLint ***stacks***
)

Draws the inner and outer cylinders of the game board with the desired number of slices and stacks

Parameters:

slices	The number of slices in the cylinder, usually the x dimension of the game board
stacks	The numbes of stacks in the cylinder, usually the y dimension of the game board

GLfixed FixedFromFloat(float ***value*** **)** `[inline]`

Converts a float value to a fixed value

Parameters:

value	The value to convert

GLfixed FixedFromInt(int ***value*** **)** `[inline]`

Converts an int value to a fixed value

Parameters:

value	The value to convert

```
void initBlock( GLfloat inner,
                GLfloat outer,
                GLfloat height,
                GLint   slices
              )
```

Initializes the block mesh and then draws a single block. If the mesh is already initialized then it simply draws a block

Parameters:
inner The radius of the inner cylinder
outer The radius of the outer cylinder
height The desired height of the block
slices The number of slices in the cylinder

```
bool          ( board * a )
InitDraw
```

Initializes the graphics with a pointer to a game board

Parameters:
a A pointer to a game board

```
bool InitOGLES(  )
```

Initializes the graphics with a default board, used for testing

```
GLfixed          ( GLfixed op1,
MultiplyFixed      GLfixed op2
                 ) [inline]
```

Multiplies two fixed values

Parameters:

op1 The first value to multiply
op2 The second value to multiply

```
void Perspective( GLfloat fovy,
                  GLfloat aspect,
                  GLfloat zNear,
                  GLfloat zFar
                )
```

A gluPerspective-like function but modified to work with Glfixed

```
void        ( )
Render
```

Renders the current game board

```
void          ( )
SetOrtho
```

Changes the view to orthographic

```
void SetPerspective( )
```

Changes the view to perspective

Variable Documentation

```
const GLfixed ONE = 1 << PRECISION
```

const unsigned int PRECISION =
16

const GLfixed ZERO =
0

Render.cpp File Reference

#include "render.h"

Functions

unsigned char *	**loadBMP** (char *filename, BITMAPINFOHEADER *bmpInfo)
bool	**loadTextures** ()
bool	**InitDraw** (**board** *a)
void	**initBlock** (GLfloat inner, GLfloat outer, GLfloat height, GLint slices)
void	**drawBoard** ()
void	**cap** (GLfloat innerRadius, GLfloat outerRadius, GLint slices)
void	**Cylinder** (GLfloat baseRadius, GLfloat topRadius, GLfloat height, GLint slices, GLint stacks)
void	**Render** ()
void	**Perspective** (GLfloat fovy, GLfloat aspect, GLfloat zNear, GLfloat zFar)
void	**SetOrtho** ()
void	**SetPerspective** ()

Variables

EGLDisplay	**glesDisplay**
EGLSurface	**glesSurface**
EGLContext	**glesContext**
HWND	**hWnd**
HDC	**hDC**
bool	**drawInOrtho**
bool	**drawTriangle**
board *	**GameBoard**
GLfloat	**capVert** [80]
GLfloat	**outerVert** [120]
GLfloat	**innerVert** [120]
GLfloat	**blockVertb** [18]
GLfloat	**blockVertl** [12]
GLfloat	**blockVertf** [21]
GLfloat	**blockVertr** [12]

GLfloat	**blockVertt** [18]
GLfloat	**blockVertba** [21]
bool	**texLoaded** = false
bool	**capInit**
bool	**blockInit**
bool	**outInit**
bool	**inInit** = false
GLfloat	**rtri**
GLuint	**texture** [1]
GLfloat	**rtfi** = 0.0f
float	**bgVertices** []
float	**bgTexCoords** []

Function Documentation

```
void cap( GLfloat innerRadius,
          GLfloat outerRadius,
          GLint   slices
        )
```

Draws the disk at the top of the game cylinder

Parameters:

innerRadius	The radius of the inner cylinder
outerRadius	The radius of the outer cylinder
slices	The number of slices in the cylinders

```
void Cylinder( GLfloat baseRadius,
               GLfloat topRadius,
               GLfloat height,
               GLint   slices,
               GLint   stacks
             )
```

Draws a cylinder with the given dimensions. A difference between the top and bottom radius creates a cone

Parameters:

baseRadius	The radius at the bottom of the cylinder
topRadius	The radius at the top of the cylinder
height	The height of the cylinder
slices	The number of slices in the cylinder, the more slices, the more cylindrical the approximation
stacks	The number of stacks in the cylinder

```
void drawBoard(  )
```

```
void initBlock( GLfloat inner,
                GLfloat outer,
                GLfloat height,
                GLint   slices
              )
```

Initializes the block mesh and then draws a single block. If the mesh is already initialized then it simply draws a block

Parameters:
inner The radius of the inner cylinder
outer The radius of the outer cylinder
height The desired height of the block
slices The number of slices in the cylinder

```
bool          ( board * a )
InitDraw
```

Initializes the graphics with a pointer to a game board

Parameters:
a A pointer to a game board

```
unsigned char* loadBMP( char *                filename,
                        BITMAPINFOHEADER * bmpInfo
                      )
```

```
bool loadTextures(  )
```

```
void Perspective( GLfloat fovy,
                  GLfloat aspect,
                  GLfloat zNear,
                  GLfloat zFar
                )
```

A gluPerspective-like function but modified to work with Glfixed

```
void        ( )
Render
```

Renders the current game board

```
void        ( )
SetOrtho
```

Changes the view to orthographic

```
void SetPerspective( )
```

Changes the view to perspective

Variable Documentation

```
float bgTexCoords[]
```

Initial value:

```
{
        3.0f, 0.0f,
        3.0f, 3.0f,
        0.0f, 0.0f,
```

```
        0.0f, 3.0f
}
```

float bgVertices[]

Initial value:

```
 {
        100.0f, -100.0f,
        100.0f,  100.0f,
        -100.0f, -100.0f,
        -100.0f,  100.0f
}
```

bool blockInit [static]

GLfloat blockVertb[18] [static]

GLfloat blockVertba[21] [static]

GLfloat blockVertf[21] [static]

GLfloat blockVertl[12] `[static]`

GLfloat blockVertr[12] `[static]`

GLfloat blockVertt[18] `[static]`

bool capInit `[static]`

GLfloat capVert[80] `[static]`

bool drawInOrtho

bool drawTriangle

board* GameBoard [static]

EGLContext
glesContext

EGLDisplay
glesDisplay

EGLSurface glesSurface

HDC hDC

HWND hWnd

bool inInit = false `[static]`

GLfloat innerVert[120] `[static]`

GLfloat outerVert[120] `[static]`

bool outInit `[static]`

GLfloat rtfi = 0.0f `[static]`

GLfloat rtri `[static]`

bool texLoaded = false `[static]`

GLuint texture[1]
`[static]`

main.h File Reference

```
#include <windows.h>
```

Go to the source code of this file.

Classes

struct	**decodeUINT**
struct	**decodeCMD**

Defines

#define	**dim**(x) (sizeof(x) / sizeof(x[0]))
#define	**IDC_CMDBAR** 1
#define	**IDM_NEWGAME** 1000

Functions

LRESULT	**DoCreateMain** (HWND, UINT, WPARAM, LPARAM)
LRESULT	**DoLeftButton** (HWND, UINT, WPARAM, LPARAM)
LRESULT	**DoKeyDown** (HWND, UINT, WPARAM, LPARAM)
LRESULT	**DoMenuCommand** (HWND, UINT, WPARAM, LPARAM)
int WINAPI	**WinMain** (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpCmdLine, int nCmdShow)
LRESULT CALLBACK	**WndProc** (HWND **hWnd**, UINT message, WPARAM wParam, LPARAM lParam)

Define Documentation

#define dim(x) (sizeof(x) / sizeof(x[0]))

```
#define
IDC_CMDBAR   1
```

```
#define
IDM_NEWGAME   1000
```

Function Documentation

```
LRESULT DoCreateMain( HWND     ,
                      UINT     ,
                      WPARAM   ,
                      LPARAM
                    )
```

```
LRESULT DoKeyDown( HWND     ,
                   UINT     ,
                   WPARAM   ,
                   LPARAM
                 )
```

```
LRESULT DoLeftButton( HWND     ,
                      UINT     ,
                      WPARAM   ,
                      LPARAM
                    )
```

```
LRESULT DoMenuCommand( HWND   ,
                       UINT   ,
                       WPARAM ,
                       LPARAM
                     )
```

```
int WINAPI WinMain( HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    LPTSTR    lpCmdLine,
                    int       nCmdShow
                  )
```

```
LRESULT CALLBACK WndProc( HWND   hWnd,
                          UINT   message,
                          WPARAM wParam,
                          LPARAM lParam
                        )
```

main.cpp File Reference

#include "main.h"
#include "render.h"
#include <commctrl.h>
#include <time.h>

Functions

int WINAPI	**WinMain** (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpCmdLine, int nCmdShow)
LRESULT CALLBACK	**WndProc** (HWND **hWnd**, UINT wMsg, WPARAM wParam, LPARAM lParam)
LRESULT	**DoCreateMain** (HWND **hWnd**, UINT message, WPARAM wParam, LPARAM lParam)
LRESULT	**DoLeftButton** (HWND **hWnd**, UINT message, WPARAM wParam, LPARAM lParam)
LRESULT	**DoKeyDown** (HWND **hWnd**, UINT message, WPARAM wParam, LPARAM lParam)
LRESULT	**DoMenuCommand** (HWND **hWnd**, UINT message, WPARAM wParam, LPARAM lParam)

Variables

HINSTANCE	**hInst**
HWND	**hWnd**
HDC	**hDC**
board *	**GameBoard**
const struct **decodeUINT**	**MainMessages** []
TCHAR	**szAppName** [] = TEXT("Tubetris")
bool	**drawTriangle** = false

Function Documentation

```
LRESULT DoCreateMain( HWND   hWnd,
                      UINT   message,
                      WPARAM wParam,
                      LPARAM lParam
                    )
```

```
LRESULT DoKeyDown( HWND   hWnd,
                   UINT   message,
                   WPARAM wParam,
                   LPARAM lParam
                 )
```

```
LRESULT DoLeftButton( HWND   hWnd,
                      UINT   message,
                      WPARAM wParam,
                      LPARAM lParam
                    )
```

```
LRESULT DoMenuCommand( HWND   hWnd,
                       UINT   message,
                       WPARAM wParam,
                       LPARAM lParam
                     )
```

```
int WINAPI WinMain( HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    LPTSTR    lpCmdLine,
                    int       nCmdShow
                  )
```

```
LRESULT CALLBACK WndProc( HWND   hWnd,
                          UINT   wMsg,
                          WPARAM wParam,
                          LPARAM lParam
                        )
```

Variable Documentation

```
bool drawTriangle = false
```

```
board* GameBoard
```

```
HDC hDC
```

HINSTANCE hInst

HWND hWnd

const struct decodeUINT MainMessages[]

Initial value:

```
{
        WM_CREATE, DoCreateMain,

        WM_LBUTTONDOWN, DoLeftButton,
        WM_KEYDOWN, DoKeyDown,
        WM_COMMAND, DoMenuCommand
}
```

TCHAR szAppName[] = TEXT("Tubetris")

board Class Reference

`#include <board.h>`

List of all members.

Public Member Functions

	board ()
	board (int x, int y)
	~board ()
sphere ***	**getboard** ()
void	**scoreBoard** ()
void	**newPiece** ()
int	**getScore** ()
int	**get_x** ()
int	**get_y** ()
bool	**drop** ()
void	**rotateLeft** ()
void	**rotateRight** ()
void	**shiftLeft** ()
void	**shiftRight** ()
int	**getCobliPosition** ()

Private Member Functions

bool	**overlap** ()
void	**removeCobli** ()
void	**setCobli** ()
void	**printboard** ()

Private Attributes

sphere ***	**game**
int	**score**

cobli	activePiece
int	x_axis
int	y_axis

Detailed Description

Module : board.h Author : Christopher M. Donnelly Email : cmdonn@wpi.edu Course : Embedded Game Design MQP

Description : acts as a board for gameplay

Date : 3/14/2008

History: Revision Date Changed By -------- ---------- ---------- 00.01 3/14/2008 cmdonn 00.02 3/21/2008 cmdonn

Constructor & Destructor Documentation

board::board( )

board::board(int *x*,
int *y*
)

a board keeps track of the location of all of the spheres in board::board() {

board(SIZE_X,SIZE_Y); //replace later with config file code }

/** constructor creates a board of size x by y all declared to a value of NOTHING.

Parameters:
x The number of columns the board will have
y The number of rows the board will have

board::~board( )

destructor. Hopefully will keep memory leaks down.

Member Function Documentation

bool board::drop ( )

Drops the cobli down one slot and returns whether the drop was successful or if it hit the bottom of the board or a playing piece

Returns:

false when a drop hits a piece and another piece is off the board. Otherwise, it returns true

int board::get_x ( )

int board::get_y ( )

sphere * board::getboard ( )**

returns the array which shows the state of the board, including the cobli.

Returns:
the board state

int board::getCobliPosition ( )

int board::getScore ( )

void board::newPiece ( )

bool board::overlap ( ) [private]

Returns whether or not a sphere is overlapping with the cobli. Note: make sure that the cobli has not been set with set cobli before running this or else it will return true

Returns:
true a sphere in the cobli overlaps with a sphere on the board false: otherwise

void board::printboard ( ) [private]

void board::removeCobli( ) [private]

void board::rotateLeft ( )

void board::rotateRight ( )

void board::scoreBoard ( )

void board::setCobli ( ) [private]

void board::shiftLeft ( )

void board::shiftRight ( )

Member Data Documentation

cobli board::activePiece [private]

sphere* board::game** [private]

int board::score [private]

int board::x_axis [private]

int board::y_axis [private]

cobli Struct Reference

#include <cobli.h>

List of all members.

Public Member Functions

	cobli ()
void	**addSphere** (**sphere** *a, int x, int y)
void	**generateNewCobli** ()
void	**rotateRight** ()
void	**rotateLeft** ()
void	**rotate180** ()
char	**get_location** ()

Public Attributes

sphere *	**piece** [5][5]
char	**location_x**
char	**location_y**

Constructor & Destructor Documentation

cobli::cobl ( )
i

Creates a cobli for use with the board object. Instanciates a cobli filled with NOTHING

Member Function Documentation

```
void cobli::addSphere( sphere * a,
                       int      x,
                       int      y
                     )
```

adds the pointer to sphere a to location x,y of the cobli

Parameters:
a a pointer to a sphere which must exist. Will be stored in cobli
x the column a is going. May throw an error if not between 0 and 4
y the row a is going. May throw an error if not between 0 and 4

```
void                          ( )
cobli::generateNewCobli
```

generates a new cobli with between 2 and 5 colored spheres of random colors and shape in it

```
char cobli::get_location( )
```

Returns the location of the center of the cobli such that if this is the center,

Returns:
a char showing the location of the center of the cobli

```
void cobli::rotate180( )
```

rotates the spheres in the cobli 180 degrees

void cobli::rotateLeft ( )

Rotates the spheres in the cobli 90 degrees counter-clockwise

void cobli::rotateRight ( )

Rotates the spheres in the cobli 90 degrees clockwise.

Member Data Documentation

char cobli::location_x

char cobli::location_y

sphere* cobli::piece[5] [5]

decodeCMD Struct Reference

`#include <main.h>`

List of all members.

Public Attributes

UINT	**Code**
LRESULT(*	**Fxn**)(HWND, WORD, HWND, WORD)

Member Data Documentation

UINT decodeCMD::Code

LRESULT(* decodeCMD::Fxn)(HWND, WORD, HWND, WORD)

decodeUINT Struct Reference

`#include <main.h>`

List of all members.

Public Attributes

UINT	**Code**
LRESULT(*	**Fxn**)(HWND, UINT, WPARAM, LPARAM)

Member Data Documentation

UINT decodeUINT::Code

LRESULT(* decodeUINT::Fxn)(HWND, UINT, WPARAM, LPARAM)

sphere Class Reference

`#include <sphere.h>`

List of all members.

Public Member Functions

	sphere ()
	sphere (char a, char b, int c)
	~sphere ()

Public Attributes

char	**color**
char	**type**
int	**score**

Detailed Description

Module : **sphere.h** Author : Christopher M. Donnelly Email : cmdonn@wpi.edu Course : Embedded Game Design MQP

Description : The structure for a sphere in the tube drop game

Date : 3/12/2008

History: Revision Date Changed By -------- ---------- ---------- 00.01 3/12/2008 cmdonn

Constructor & Destructor Documentation

sphere::spher ( )
e

generates a sphere to be used later

```
sphere::spher  ( char  a,
e                char  b,
                 int   c
               )
```

creates a non-default sphere to be used later

```
sphere::~sphere(  )
```

Member Data Documentation

```
char sphere::color
```

```
int sphere::score
```

```
char
sphere::type
```