Wild Magic Version 4.0
Installation Manual and Release Notes

Document Version 4.0.0, 01 July 2006

Geometric Tools, Inc.

Contents

1 Introduction

1.1 About This Document

Please read this document before attempting to compile and run the libraries and applications! Each supported platform (Microsoft Windows, Linux, Macintosh) has prerequisites that must be met in order for Wild Magic Version 4 (WM4) and its tools to compile, link, and run successfully. Once the prerequisites are met, the projects must be compiled in a particular order. Standard support questions are about issues that arise when the prerequisites are not met, when the projects are compiled in the wrong order, or when incompatible build configurations are used for the core libraries and the sample applications.

Other information of interest is available here, so browse the entire document before starting the installation. Generally, you should also check our website, Geometric Tools, for updates, bug fixes, new features, and other materials. The update history page always has a date for the last modification, so you should be able to track what you have or have not downloaded.

1.2 Copying the Distribution to Your Machine

The CDROM has top-level folder `GeometricTools`. The secondary folder is `WildMagic4`. Copy the top-level folder and all its contents to any location you prefer on your hard drive. The projects all use relative paths and do not rely on the top-level directory being located at the root of the drive. The Macintosh distribution has some script files (`.sh` extension) that must have the correct permissions. These will be discussed later in the section on compiling.

1.3 Factored Libraries and Viewable Source

The Wild Magic Version 3 (WM3) distribution had the Foundation library, which contained the core system and mathematical support, the graphics subsystem, and the physics subsystem. All of the WM3 source code was covered by a single license agreement. For WM4, the Foundation library was factored into three libraries. The core system and mathematical support is all that is in the WM4 Foundation library and has its own license agreement, `Wm4FoundationLicense.pdf`, located in the `GeometricTools` top-level folder. The graphics subsystem and physics subsystem are each a separate library. These libraries, as well as the renderer libraries, the application libraries, and the sample applications are covered by a different license agreement, `Wm4RestrictedLicense.pdf`, located in the `GeometricTools` top-level folder. In total, the WM4 distribution is *Viewable Source*, not *Open Source*.

The new folder hierarchy for WM4 is

```
GeometricTools
    WildMagic4
        LibFoundation          // core system and mathematical support
        LibGraphics            // platform-independent graphics system
        LibPhysics             // platform-independent physics system
        LibRenderers
            Dx9Renderer        // Direct3D-based renderer (Windows)
            OpenGLRenderer     // OpenGL-based renderer (Windows, Linux, MINGW, Mac)
            SoftRenderer       // software renderer (Windows, Linux, Mac)
        LibApplications
            Dx9Application     // applications using Direct3D rendering
            OpenGLApplication  // applications using OpenGL rendering
            SoftApplication    // applications using software rendering
        SampleFoundation
        SampleGraphics
        SampleImagics
        SamplePhysics
        SDK
            Include
            Library
                Debug
                DebugDLL
                DebugMemory
                Release
                ReleaseDLL
                ReleaseMemory
```

1.4 Automatic Builds

In order to allow you to build the libraries and applications automatically, each platform provides batch or script files to encapsulate the process.

1.4.1 Microsoft Windows XP

The top-level folder `GeometricTools/WildMagic4` contains an executable named `Wm4Installer.exe`. Its syntax for execution is

```
Wm4Installer -m compiler -g renderer -c configuration
```

where `compiler` is in (msvc60,msvc70,msvc71,msvc80,msvc80e), `renderer` is in (opengl,directx,soft), and `configuration` is in (debug,debugdll,debugmemory,release,releasedll,releasememory). For example, to build with Microsoft Visual Studio .NET 2003 (VC71), OpenGL renderer and applications, and the release configuration, you would use

```
Wm4Installer -m msvc71 -g opengl -c release
```

The installer just runs the compiler in command-line mode. The locations of the compiler executables are stored in the Windows Registry and the installer looks up this information. The locations for VC80 Professional Edition and VC80 Express Edition are different, which explains the two compiler options, `msvc80` (Professional) and `msvc80e` (Express). The Express Edition is freely downloadable.

1.4.2 Linux and MINGW

These system use the same makefiles. The top-level make file is `makefile.wm4`, is located in the folder `GeometricTools/WildMagic4` and has the following syntax:

```
make -f makefile.wm4 CFG={config} SYS={system}, GRF={graphics}
```

where `config` is one of (Debug,Release,DebugMemory,ReleaseMemory), `system` is one of (linux,mingw), and `graphics` is one of (OpenGL,Soft). For example, to build on Linux, OpenGL renderer and applications, and the release configuration, you would use

```
make -f makefile.wm4 CFG=Release SYS=linux GRF=OpenGL
```

To build on MINGW, software renderer, and the debug memory configuration, you would use

```
make -f makefile.wm4 CFG=DebugMemory SYS=mingw GRF=Soft
```

1.4.3 Macintosh

A single top-level script is used to build the WM4 distribution. Its name is `MacBuildWm4.sh` and is located in the folder `GeometricTools/WildMagic4`. This file must be executed from a Terminal window. The syntax is

```
./MacBuildWm4.sh graphics config libtype buildtype
```

where `graphics` is one of (Agl,Soft), `config` is one of (Debug,Release), `libtype` is one of (Static,Dynamic), and `buildtype` is one of (build,clean). The script executes the command-line version of Xcode, namely, `xcodebuild`. For example, to build the projects for OpenGL rendering, release configuration, and dynamic libraries, you would use

```
./MacBuildWm4.sh Agl Release Dynamic build
```

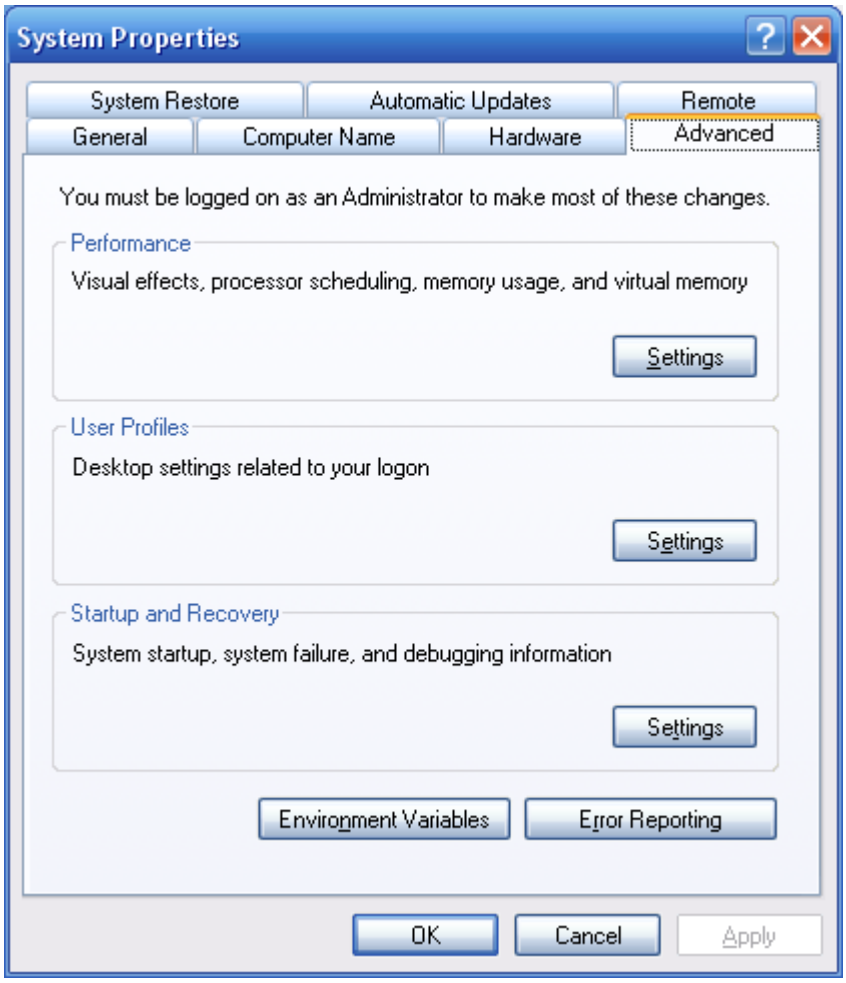
The sample applications should be run from within the Xcode projects, because the executables are buried a few levels deep in the `build` subfolders of the project folders. The path processing to locate data files depends on this folder structure.

1.5 Environment Variables

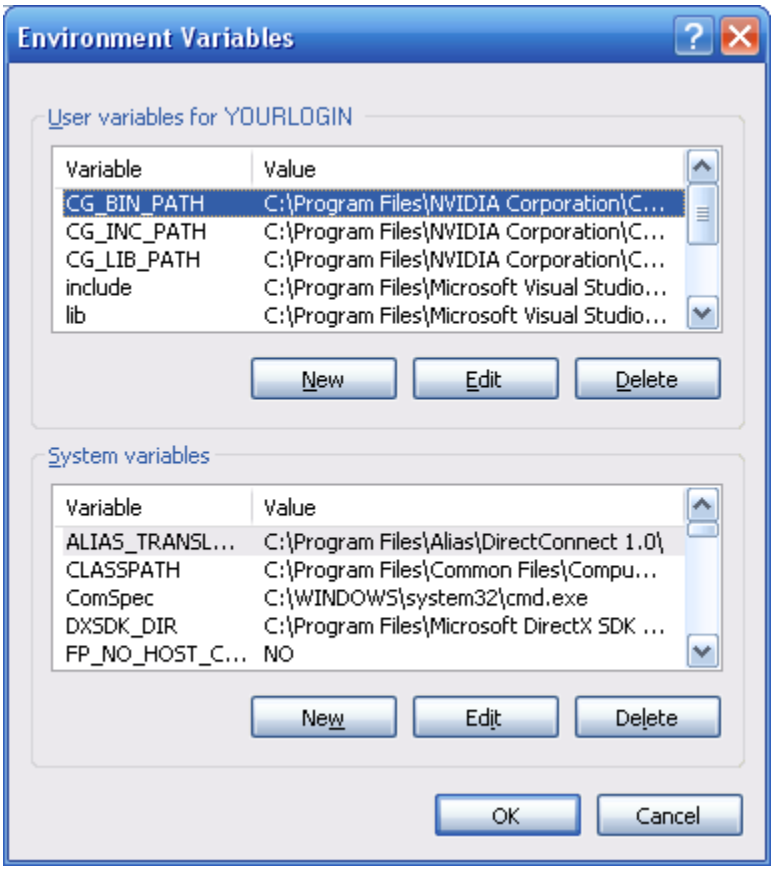
Wild Magic Version 3 (WM3) applications required you to specify absolute paths to data files. WM4 allows relative paths and supports a list of paths for searching for data files. To make this work, the application layer accesses an environment variable, `WM4_PATH`, whose value is set to the installation location of WM4. Each platform has a different approach to setting an environment variable. The following examples are based on the assumption that the top-level folder `GeometricTools` was placed in the root of a hard drive (Windows, Mingw) or in the root of the home directory (Linux, Macintosh). You will have to modify these if you place the top-level folder elsewhere.

1.5.1 Microsoft Windows XP

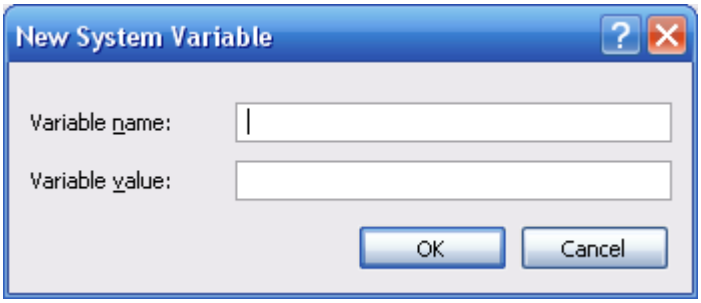
You may set an environment variable by using the Control Panel. Launch the System dialog and select the Advanced tab. You will see the dialog



Select the `Environment Variables` button. You will see the dialog



where `YOURLOGIN` will actually be your user name. You may add a new environment variable to apply only to your account or to the entire system as a whole. We add the `WM4_PATH` variable to the entire system. In this case, select the `New` button under the System Variables listing. The following dialog appears



Enter in the Variable name edit control the symbol `WM4_PATH`. Enter in the Variable value edit control the location of the source code distribution. For example, if the top-level folder `GeometricTools` was placed in the root of the C drive, you would enter

```
C:\GeometricTools\WildMagic4
```

If your path to `WildMagic4` has spaces in it, you should include starting and ending double-quote characters, for example,

```
"C:\Documents and Settings\YOURLOGIN\My Documents\GeometricTools\WildMagic4"
```

MINGW automatically sets up its environment to use the Windows environment variables, so there is nothing special you must do within a window launched by MINGW.

1.5.2 Linux

We run a Bash shell and define the following variable in the `.bashrc` file,

```
WM4_PATH=/home/YOURLOGIN/GeometricTools/WildMagic4 ; export WM4_PATH
```

You must replace `YOURLOGIN` by your actual login name. For example, if you started with the default `.bashrc` file, you would modify it to look like

```
# .bashrc
WM4_PATH=/home/YOURLOGIN/GeometricTools/WildMagic4 ; export WM4_PATH

# User specific aliases and functions

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi
```

The actual path depends on `YOURLOGIN` and where you copied the Wild Magic distribution. The `.bashrc` file is processed when you login. However, if you modify it, you may process it by executing

```
source .bashrc
```

from a terminal window.

1.5.3 Macintosh

To have environment variables automatically loaded when logging in, you need to have a file

```
/Users/YOURLOGIN/.MacOSX/environment.plist
```

whose contents are

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
    "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
<key>WM4_PATH</key>
<string>/Users/YOURLOGIN/GeometricTools/WildMagic4</string>
</dict>
</plist>
```

You must replace `YOURLOGIN` by your actual login name. A skeleton file containing this information is in the `GeometricTools/WildMagic4` folder in the distribution. If you already have an `environment.plist` file on your machine, you will have to edit it and add the path information. If you add or modify `environment.plist`, you should log out and then log in so that the new definitions are loaded.

1.6 Compiler Support for Windows Dynamic Link Libraries

To support dynamic link libraries (DLLs) under Microsoft Windows XP, the source code has symbols that need to be exported from the libraries and imported into the applications. Special keywords exist in order for the compiler to generate or locate these symbols,

```
__declspec(dllexport)    // for exporting symbols
__declspec(dllimport)    // for importing symbols
```

The problems with such a mechanism for dynamic libraries is that the header files must use `dllexport` for exporting and `dllimport` for importing. Which of these it is depends on the context in which the header file is processed, so these keywords may not be hard-coded into the source code. For example, the Foundation library project must export the symbols but any client project that links to the Foundation library must import the symbols. The Foundation library has a file named `Wm4FoundationLIB.h`, whose contents are

```
// For the DLL library.
#ifdef WM4_FOUNDATION_DLL_EXPORT
#define WM4_FOUNDATION_ITEM __declspec(dllexport)

// For a client of the DLL library.
#else
#ifdef WM4_FOUNDATION_DLL_IMPORT
#define WM4_FOUNDATION_ITEM __declspec(dllimport)

// For the static library.
#else
#define WM4_FOUNDATION_ITEM

#endif
```

The LibFoundation project includes in its list of preprocessor definitions the symbol

```
WM4_FOUNDATION_EXPORT_DLL
```

During compilation of the Foundation library, `__declspec(dllexport)` is active, which lets the compiler know that the library symbols must be tagged for export. The LibGraphics project is a client of the LibFoundation library and includes in its list of preprocessor definitions the symbol

```
WM4_FOUNDATION_IMPORT_DLL
```

During compilation of the Graphics library, `__declspec(dllimport)` is active, which lets the compiler know that the Foundation library symbols used by the Graphics library are being imported. The symbols are accessible because the LibGraphics project links in the DLL library stub that was generated by the LibFoundation project.

The LibGraphics project itself has symbols that need to be exported for use by clients. The clients themselves must specify that they need to import the symbols. Thus, you will find a file `Wm4GraphicsLIB.h` in the LibGraphics project, whose contents are

```
// For the DLL library.
#ifdef WM4_GRAPHICS_DLL_EXPORT
#define WM4_GRAPHICS_ITEM __declspec(dllexport)

// For a client of the DLL library.
#else
#ifdef WM4_GRAPHICS_DLL_IMPORT
#define WM4_GRAPHICS_ITEM __declspec(dllimport)

// For the static library.
#else
#define WM4_GRAPHICS_ITEM

#endif
```

It is insufficient to have a single file to control whether importing or exporting is active and that works for multiple libraries, because a client can have the need to import and export symbols. To export symbols, the project defines `WM4_GRAPHICS_DLL_EXPORT`. To import symbols, the client defines `WM4_GRAPHICS_DLL_IMPORT`.

Each library project in the WM4 distribution defines such a header file to enable importing and exporting of symbols. The LibPhysics project has associated preprocessor definitions

```
WM4_PHYSICS_IMPORT_DLL, WM4_PHYSICS_EXPORT_DLL
```

All the projects under the LibRenderer folder have associated preprocessor definitions

```
WM4_RENDERER_IMPORT_DLL, WM4_RENDERER_EXPORT_DLL
```

It is important to understand that you must include one or more of these preprocessor definitions in your projects when you want to use DLL versions of the WM4 libraries.

1.7 Finding Windows Dynamic Link Libraries at Run-Time

The compiled WM4 libraries are stored in the following directories:

```
GeometricTools/WildMagic4/SDK/Library/Debug
                                      DebugDLL
                                      DebugMemory
                                      Release
                                      ReleaseDLL
                                      ReleaseMemory
```

The subdirectories ending in DLL contain the dynamic link libraries. The names of the debug DLLs end in a 'd', but the release DLL names do not. For example, there are libraries

```
GeometricTools/WildMagic4/SDK/Library/DebugDLL/Wm4Foundation80d.dll
GeometricTools/WildMagic4/SDK/Library/ReleaseDLL/Wm4Foundation80.dll
```

When a sample application is compiled and linked to use the core dynamic libraries, the run-time environment must find them. The current working directory is checked first. If required DLLs cannot be found in the current working directory, the directories in the `PATH` environment variable are searched for the DLLs. To support testing and running of all possible configurations, we add a couple of paths to the `PATH` environment variable. These paths use the `WM4_PATH` environment variable discussed previously in this document. That discussion mentioned how to create new environment variables for Windows, so you may add the new variables accordingly. The `PATH` environment variable may be modified using the same dialogs. We have in our system environment:

```
WM4_PATH=C:\GeometricTools\WildMagic4
WM4_PATH_DEBUG_DLL=C:\GeometricTools\WildMagic4\SDK\Library\DebugDLL
WM4_PATH_RELEASE_DLL=C:\GeometricTools\WildMagic4\SDK\Library\ReleaseDLL
```

This assumes the distribution is located in the root of the C drive. Modify these as needed based on where you copied the Wild Magic distribution. We then modify the `PATH` environment variable to include these. For example, you can append these to whatever your current path is defined as:

```
PATH=<currentpath>;%WM4_PATH_DEBUG_DLL%;%WM4_PATH_RELEASE_DLL%
```

As before, if you have spaces in your path names, you must inclue the paths within double quotes.

If you are running Microsoft Visual Studio and change an environment variable or add new ones, you need to exit out of Visual Studio and restart it. When Microsoft Visual Studio starts, it loads the current environment variables and makes copies. The restart is necessary for it to detect your changes.

1.8 Test Platforms

We have a small number of machines on which we test the distribution. These include:

- Machine 1
  - CPU: Intel Pentium D840 Dual Core, 3.2 GHz
  - MEMORY: 4 GB
  - GRAPHICS: NVIDIA GeForce 6800, Forceware 81.98
  - OS: Microsoft Windows XP (Service Pack 2)
  - COMPILERS:
- Machine 2
  - CPU: AMD Athlon XP 2800+, 2.1 GHz
  - MEMORY: 1.5 GB
  - GRAPHICS: ATI Radeon X1300 Pro, Driver version 8.241
  - OS: Microsoft Windows XP (Service Pack 2)
- Machine 3 (dual boot of Machine 2)
  - CPU: AMD Athlon XP 2800+, 2.1 GHz
  - MEMORY: 1.5 GB
  - GRAPHICS: ATI Radeon X1300 Pro, Driver version 8.24.8
  - OS: Red Hat Fedora Core 3 (2.6.12-1.1381-FC3)
- Machine 4
  - CPU: Macintosh Dual processor PowerMac G5, 64-bit
  - MEMORY: 2 GB
  - GRAPHICS: NVIDIA GeForce 6600
  - OS: OS X 10.4.6
  - COMPILER: Xcode 2.3, g++ 4.0.1
- Machine 5
  - CPU: Intel Pentium 4, 3.0 GHz
  - MEMORY: 1.0 GB
  - GRAPHICS: ATI Radeon X600, Driver version 8.16.2
  - OS: Microsoft Windows XP (Service Pack 2)
  - COMPILER: Microsoft Visual Studio .NET 2005 Express, Windows Server 2003 R2 Platform SDK (March 2006)
- Machine 6
  - CPU: Intel Pentium M735, 1.7 GHz
  - MEMORY: 1.0 GB
  - GRAPHICS: ATI FireMV 2400 (OpenGL 1.3)
  - OS: Microsoft Windows XP (Service Pack 2)
  - COMPILER: Microsoft Visual Studio .NET 2005 Professional

2 Prerequisites and Portability

Wild Magic is architected to be portable. Generally, no library will automatically compile, link, and run when placed on a new platform. If you attempt to compile Wild Magic on platforms other than the ones we have tested, there might be minor issues that need to be resolved. Some of the tools work only on the Microsoft Windows platform; for example, Max and Maya are modeling packages that run on Windows—the exporters naturally work only with those packages. Any prerequisites for the platforms and environments on which we have tested Wild Magic are listed in the next sections.

2.1 Microsoft Windows XP

We have project files for multiple versions of Microsoft's Visual C++. We also have makefiles for MinGW.

2.1.1 Microsoft Visual C++ 6.0 (VC60)

The project files have extension `.dsp`. We have added the suffix `_VC60` to the project file names just to make it clear that these are for version 6.0 of the compiler. The projects allow you to build static libraries but not dynamic libraries. We were unable to get the dynamic libraries to link, the problem apparently related to our using explicitly instantiated templates. The VC60 STL implementation is from Dinkumware, Ltd, a third party vendor. The xtree template uses a static data member to represent the `nil` pointer. This causes serious crashes when using dynamic link libraries. The problem is that the static data member is instantiated *twice*, once in the DLL module and once in the EXE module. Iteration over container elements in the application code (in the EXE module) will access the wrong `nil`. For legal reasons, Dinkumware was not allowed to provide a fix to Microsoft but they did have a fix at their web site. We decided not to attempt getting DLLs to work with VC60. Given that this is a very old compiler, you should abandon it anyway.

2.1.2 Microsoft Visual Studio .NET 2002 (VC70)

The project files have extension `.vcproj`, as do the project files for the later versions of the compiler. We added the suffix `_VC70` to the project file names to make it clear they are for version 7.0 of the compiler.

We use the precompiled header systems for compilation. The system in VC70 complains about the cpp files in the `SharedArrays` folder, specifically about duplicate explicit instantiations that are caused by the RTTI and Stream macros. This appears to be a bug in VC70. When you build `LibFoundation_VC70.vcproj` the first time, you will see errors about the `SharedArrays` cpp files. This is not a problem! Build the project a second time. The cpp files compile this time and the library is created as expected. The Wild Magic Installer, `Wm4Installer.exe`, is configured to do the repeated build of this project.

VC70 has a few problems with templates. We had to add some conditional compilation to avoid these problems. In particular, the rational arithmetic files `Wm4RVector{2,3}.*` have conditional compilation. These problems do not exist in VC71 or VC80.

2.1.3 Microsoft Visual Studio .NET 2003 (VC71)

The project files have extension `.vcproj`. We added the suffix `_VC71` to the project file names to make it clear they are for version 7.1 of the compiler. We have not run into any significant quirks with this version of the compiler.

2.1.4 Microsoft Visual Studio .NET 2005 (VC80)

This includes the Professional Edition and Microsoft Visual C++ 2005 Express Edition (VC80). We added the suffix `_VC80` to the project file names to make it clear they are for version 8.0 of the compiler.

Microsoft has made the Express Edition compiler freely available. This version does have an optimizing compiler. However, the development environment is not identical to that of the professional version. If you download this, follow all the directions. You must download the Windows Server 2003 R2 Platform SDK (March 2006), which is available from the same web page. Part of the directions include modifying a file called `corewin_express.vsprops`. You must do this step; otherwise, our renderer projects will require you to list two additional libraries for linking, `gdi32.lib` and `user32.lib`. You should also modify the `AppSettings.htm` file as described in the installation notes for the compiler. The suggest change allows Win32 Application to show up in the Application Wizard.

2.1.5 MinGW

This is the package Minimalist GNU for Windows, which is a freely available download. We have been successful at building and running Wild Magic 4 using MSYS (1.0.10) and MINGW (4.1.0) with the Wgl OpenGL renderer.

2.1.6 Borland C++ Builder and Intel Compilers

Some users of Wild Magic Version 2 reported compile problems with Borland's C++ Builder Studio. We were able to suggest work-arounds for most of these. Some of the reported errors were apparent bugs in the compiler but we weres unable to find work-arounds.

We have never tested on Intel compilers.

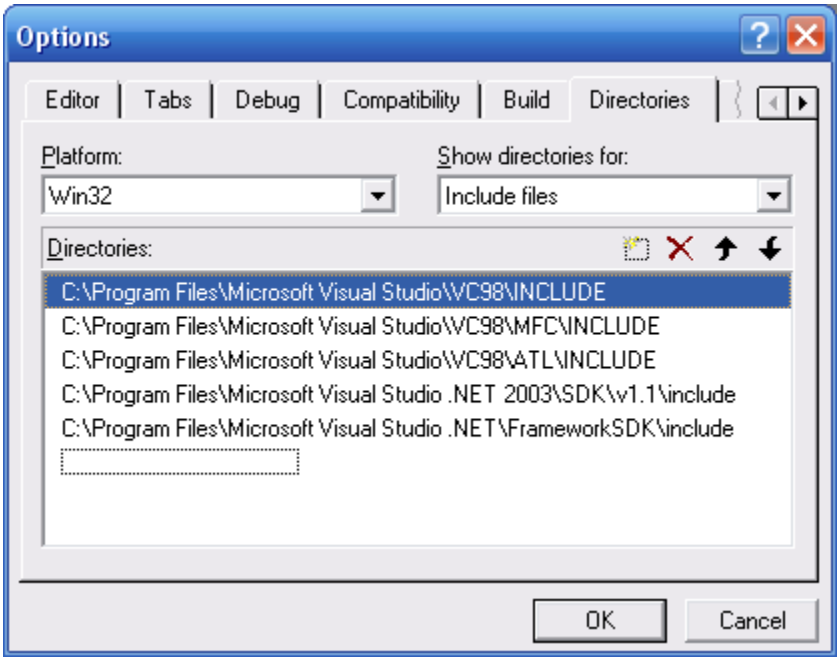
If anyone wants support for either of these compilers, let us know. If enough users make the request, we will consider purchasing the compilers.

2.1.7 DirectX

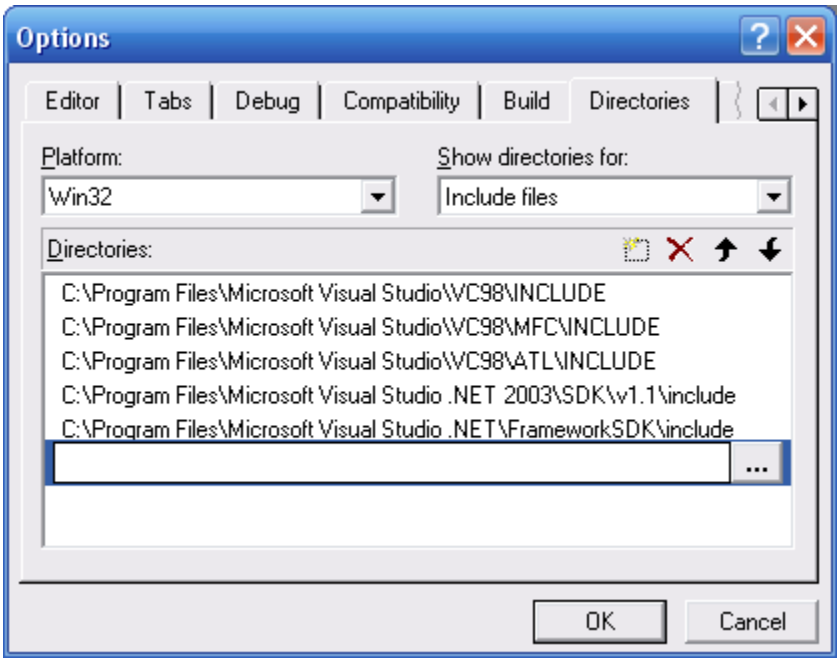
If you plan on compiling the DirectX projects, you must download the DirectX SDK from Microsoft's website and install it on your machine. We have had a few support emails about the failure of the DirectX renderer to compile. In nearly all the cases, the problem is that the user did not even have the DirectX SDK installed. In the other cases, the user did not have the global paths set up to find the DirectX SDK files. We am using the most current version (as of the time of writing this document), DirectX SDK (April 2006).

If you install the SDK, one of the dialogs asks if you want to have the DirectX extensions installed in Visual Studio. If you answer 'yes', the global paths the compiler uses will be modified to include the relevant DirectX directories. If you answer 'no', you will not be able to compile the DirectX renderer unless you modify the global paths yourself. We always say 'yes', but only VC70, VC71, and VC80 have their paths modified. We had to modify the VC60 paths manually.

To modify the paths for VC60, open the development environment. From the menu bar select the item `Tools | Options`. A dialog is launched. Select the `Directories` tab. The following dialog appears,



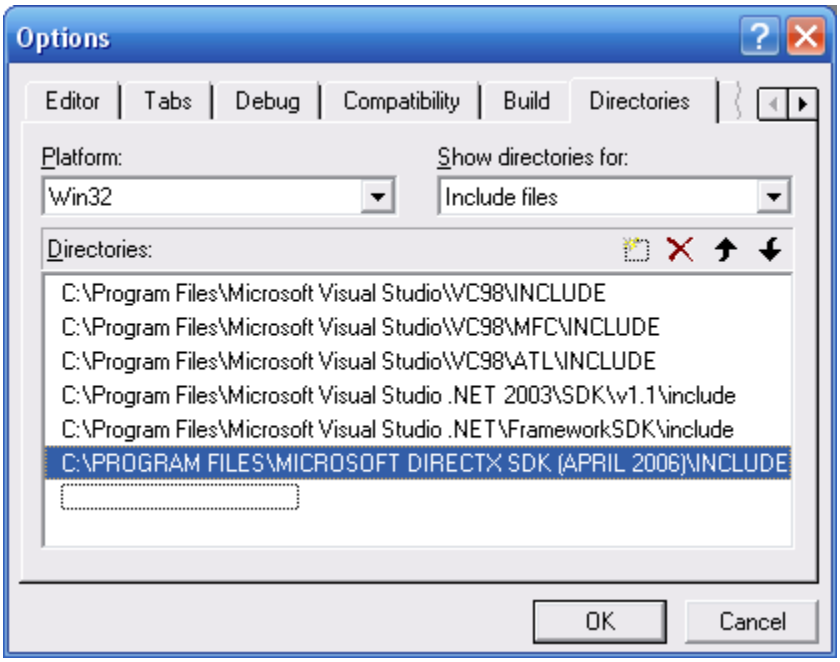
There is a drop-down list entitled `Show directories for`. Select this to be `Include files`. You will see a list of directories, as the figure indicates. Double-click in the dotted rectangle. You will see a browse button, as shown in the next figure.



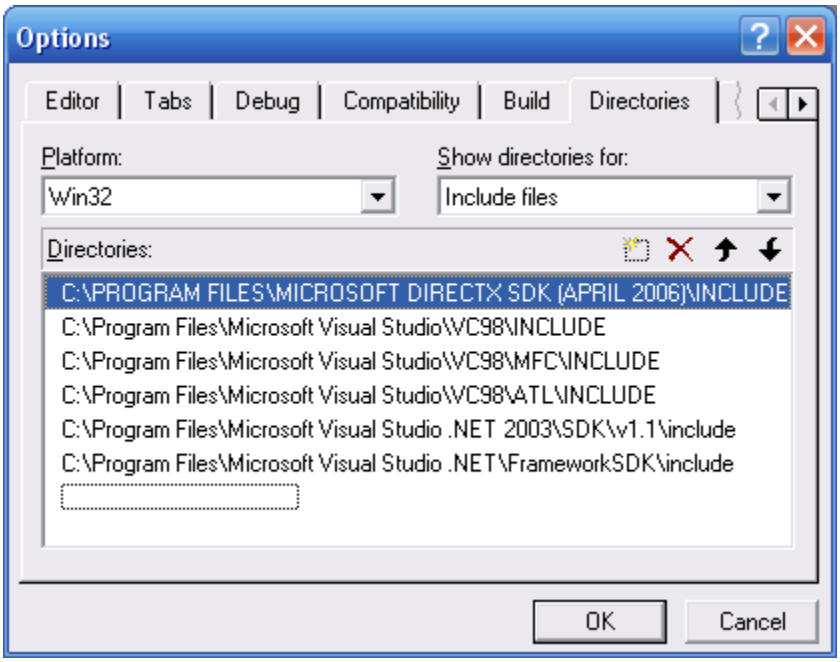
Click on the button with the three dots. A navigation dialog appears. Navigate to the location at which you installed the DirectX SDK and find the include directory. On our machines this is

```
C:\Program Files\Microsoft DirectX SDK (April 2006)\Include
```

The next figure shows the dialog after this path was selected from the navigation dialog.



Now move this to the *top* of the list using the black arrow button on the dialog box (the button to the right of the red X button). VC60 shipped with an older version of DirectX. You want to ensure that the latest version of DirectX is accessed by having its path listed first. The next figure shows the dialog after the path was moved to the top of the list.



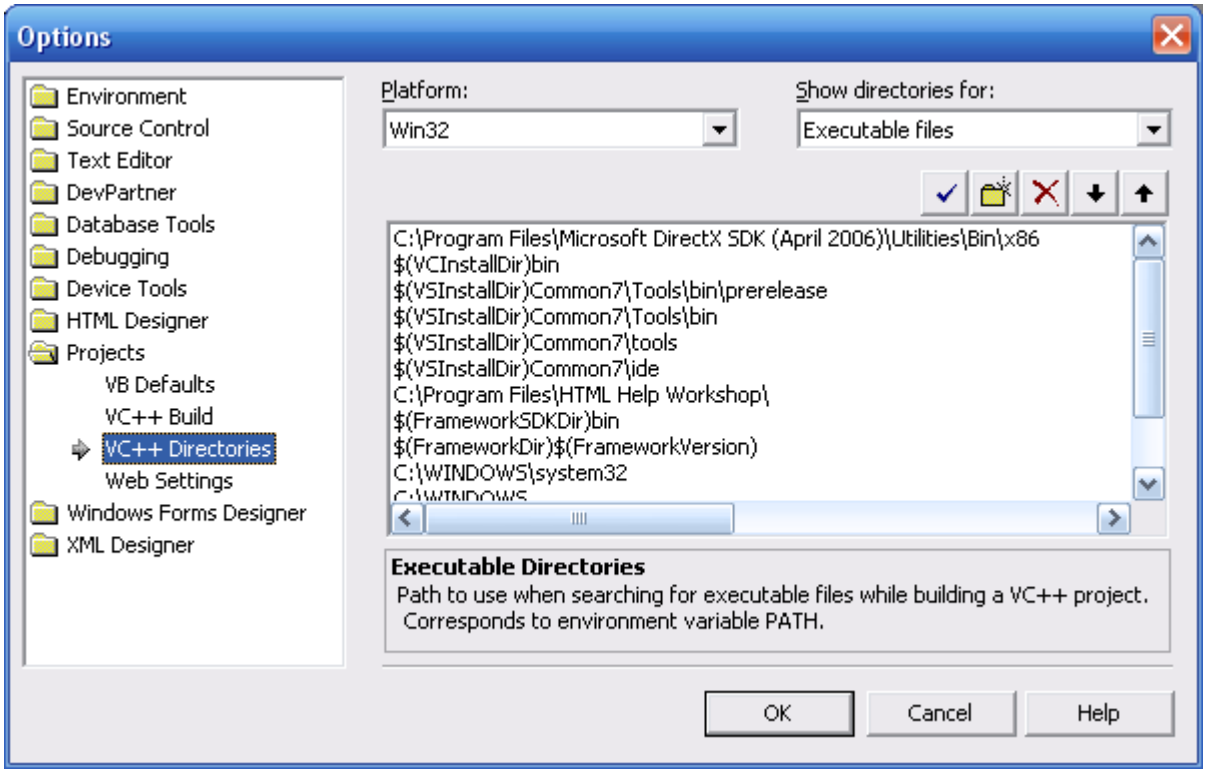
Make a similar modification by changing the drop-down list to `Library files`. On our machines, the location is

```
C:\Program Files\Microsoft DirectX SDK (April 2006)\Lib\x86
```

The x86 means the 32-bit version of the SDK. If you have a 64-bit machine, you will need x64 instead. Also move this item to the top of the list. If you plan on using any of the DirectX utility executables, change the drop-down list to `Executable files` and add the executable path. On our machine this is

```
C:\Program Files\Microsoft DirectX SDK (April 2006)\Utilities\bin\x86
```

To modify the paths for VC70 or VC71, open the development environment. From the menu bar select the item `Tools | Options`. A dialog is launched. In the left pane, scroll down until you see the folder named `Projects`. Select the folder to expand it. You will then see an item under the folder named `VC++ Directories`. Select it. The next figure shows the state of the dialog after these actions. The directions for modifying options is nearly the same as for VC60.



A list of paths is shown. There is also a drop-down list entitled `Show directories for`. Select this to be `Include files`. Scroll to the bottom of the list of paths and double-click in the white space at the end of the list. You will see a browse button. Browse to the location you stored the DirectX SDK and find the include directory. On our machines this is

```
C:\Program Files\Microsoft DirectX SDK (April 2006)\Include
```

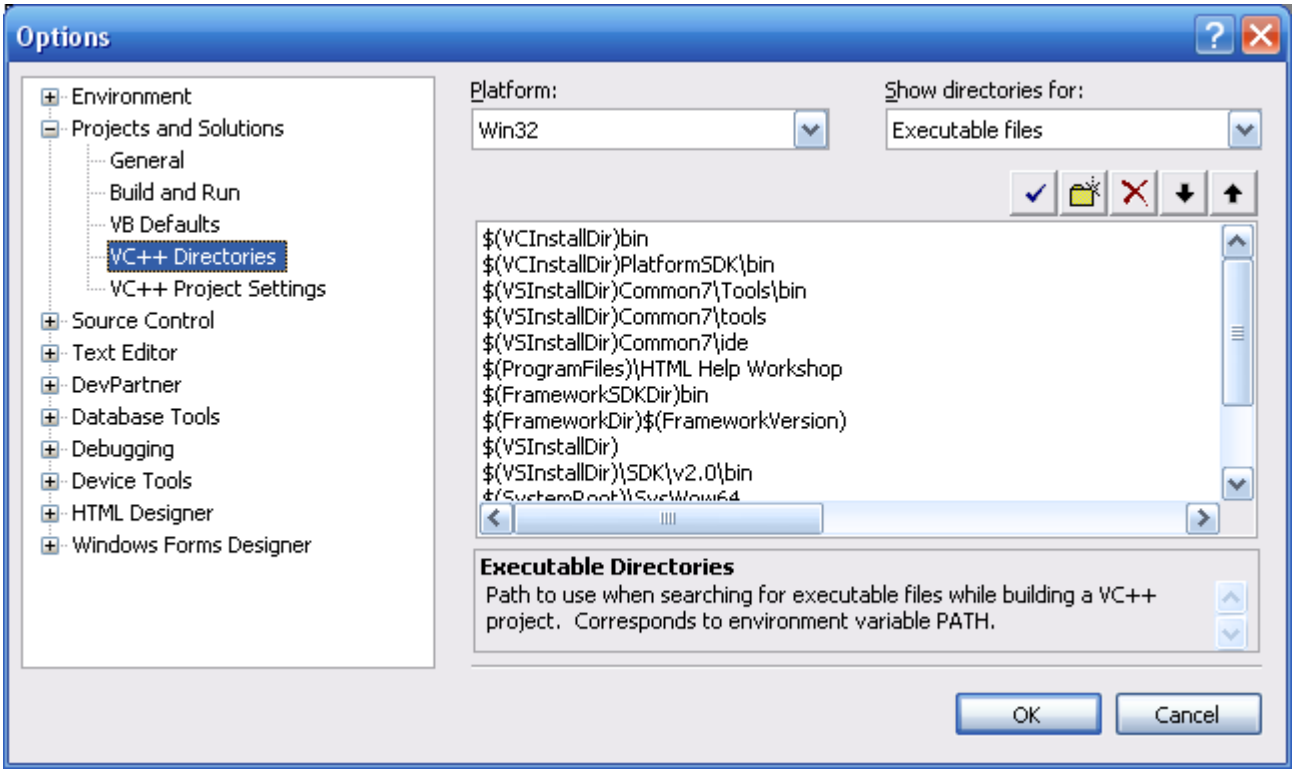
Move this to the *top* of the list using the black arrow button on the dialog box. Make a similar modification by changing the drop-down list to say `Library files`. On our machines, the location is

```
C:\Program Files\Microsoft DirectX SDK (April 2006)\Lib\x86
```

The x86 means the 32-bit version of the SDK. If you have a 64-bit machine, you will need x64 instead. Also move this item to the top of the list. If you plan on using any utility executables, change the drop-down list to `Executable files` and add the executable path. On our machines this is

```
C:\Program Files\Microsoft DirectX SDK (April 2006)\Utilities\bin\x86
```

To modify the paths for VC80, open the development environment. From the menu bar select the item `Tools | Options`. A dialog is launched. In the left pane, scroll down until you see the item named `Projects and Solutions`. Select the item to expand it. You will then see an item under the item named `VC++ Directories`. Select it. A list of paths is shown. The next figure shows the state of the dialog after these actions. The directions for modifying options is nearly the same as for VC60.



There is also a drop-down list entitled `Show directories for`. Select this to be `Include files`. Scroll to the bottom of the list of paths and double-click in the white space at the end of the list. You will see a

browse button. Browse to the location you stored the DirectX SDK and find the include directory. On our machines this is

```
C:\Program Files\Microsoft DirectX SDK (April 2006)\Include
```

Move this to the *top* of the list using the black arrow button on the dialog box. Make a similar modification by changing the drop-down list to say `Library files`. On our machines, the location is

```
C:\Program Files\Microsoft DirectX SDK (April 2006)\Lib\x86
```

The x86 means the 32-bit version of the SDK. If you have a 64-bit machine, you will need x6x instead. Also move this item to the top of the list. If you plan on using any utility executables, change the drop-down list to `Executable files` and add the executable path. On our machines this is

```
C:\Program Files\Microsoft DirectX SDK (April 2006)\Utilities\bin\x86
```

2.1.8 OpenGL and Extension Wrappers

We no longer use GLUT. We used to use GLEW, an OpenGL extension wrapper that is freely downloadable. Now we have rolled our own extension wrapper. Currently, it supports ARB and EXT extensions. We will add support for the other extensions as needed.

Our wrapper is designed to call an OpenGL extension function, if it exists, but to ignore it otherwise. This guarantees that an OpenGL-based application will not crash in an attempt to dereference a null function pointer. The wrapper also allows you to hook into the system for such things as reporting whether extension exist and/or are used, for profiling the OpenGL calls, or for call-stack tracing during a drawing operation.

Each extension function wrapper is effectively one of two forms. For functions not returning a value,

```
void GTglFunction (type0 param0, type1 param1, ...)
{
    GT_ENTER_FUNCTION(glFunction);
    if (glFunction)
    {
        GT_NONNULL_FUNCTION(glFunction);
        glFunction(param0,param1,...);
    }
    else
    {
        GT_NULL_FUNCTION(glFunction);
    }
    GT_EXIT_FUNCTION(glFunction);
}
```

For functions returning a value,

```
rettype GTglFunction (type0 param0, type1 param1, ...)
{
    rettype tResult;
    GT_ENTER_FUNCTION(glFunction);
    if (glFunction)
    {
        GT_NONNULL_FUNCTION(glFunction);
        tResult = glFunction(param0,param1,...);
    }
    else
    {
        GT_NULL_FUNCTION(glFunction);
        tResult = nullRetValue;
    }
    GT_EXIT_FUNCTION(glFunction);
    return tResult;
}
```

The default behavior is determined by the macros starting with `GT_` and are defined by the preprocessor to expand to nothing. You can implement the macros anyway you desire to support your development environment. For example, profiling amounts to using a timer and starting the timer in `GT_ENTER_FUNCTION` and stopping the timer in `GT_EXIT_FUNCTION`. Call-stack tracing amounts to reporting the function being entered and exited by use of these same macros.

To add new behavior, you must re-implement the preprocessor definitions in the file `GTglPlugin.h` and possibly create a source file, `GTglPlugin.c`, that you would insert into the `OpenGLRenderer` project.

2.1.9 NVIDIA's Cg Toolkit

If you are going to write shader programs to be used by Wild Magic, you need to have NVIDIA's Cg Toolkit installed. You can download this from NVIDIA's web site. We had downloaded Version 1.1 of the toolkit, but are now using Version 1.4. A couple of Cg shader programs have not correctly compiled with 1.4, so in those cases we compilec using 1.1. In particular, the `VertexNoise`, `ProjectedTexture`, and `Charcoal` Cg programs from Wild Magic Version 3 did not compile correctly. In Wild Magic Version 4, we also ran into a problem with the `ShadowMaps` sample application using the `ps_2_0` profile for the `p_ShadowMap` pixel shader. Specifically, we had

```
if (fPointDepth > kDepth.r + 0.01f)
{
    // in shadow
    // pixelcolor = (0,0,0,basecolor.a)
}
else
{
    // not in shadow
```

```
    // pixelcolor = basecolor
}
```

The compiled code for OpenGL worked. The DirectX code had the shadow and not-in-shadow colors reversed. After changing the `>` to `>=`, the compiled code worked fine for both OpenGL and DirectX.

For convenience of compiling shaders from a command window, we have provided some simple batch files in folder `GeometricTools/WildMagic4/Bin`,

```
GLVProgram.bat  // OpenGL vertex programs,  profile arbvp1
GLPProgram.bat  // OpenGL pixel programs,   profile arbfp1
DXVProgram.bat  // DirectX vertex programs, profile vs_2_0
DXPProgram.bat  // DirectX pixel programs,  profile ps_2_0
```

Some low-end graphics hardware cannot handle these shader models. In those cases, you might try compiling the Cg programs with a less powerful profile.

We use the naming convention `v_ProgramName` for vertex programs and `p_ProgramName` for pixel programs. If these are implemented in a file, `ProgramName.cg`, the syntax for the batch files is

```
GLVProgram ProgramName ProgramName  // output: v_ProgramName.ogl.wmsp
GLPProgram ProgramName ProgramName  // output: p_ProgramName.ogl.wmsp
DXVProgram ProgramName ProgramName  // output: v_ProgramName.dx9.wmsp
DXPProgram ProgramName ProgramName  // output: v_ProgramName.dx9.wmsp
```

The first parameter of the batch file is the shader program name. The second parameter of the batch file is the output program name (with `v_` or `p_` prepended). These names are not necessarily the same; for example, look at `ProjectedShadow.cg` in the `ShadowMaps` sample application. The compiled shader programs for OpenGL were generated by

```
GLVProgram ProjectedDepth ProjectedShadow
GLPProgram ProjectedDepth ProjectedShadow
GLVProgram ShadowMap ProjectedShadow
GLPProgram ShadowMap ProjectedShadow
```

The `.wmsp` shader programs for the software renderer were all manually created by translating the Cg code to C++ code.

2.2 Linux

We currently use Red Hat Fedora Core 3, which ships with g++ version 3.4.2. Our Linux box has an ATI Radeon X1300 Pro. ATI does not yet have Linux drivers for this card that will run under Red Hat Fedora Core 5—the XOrg version on this operating system is larger than ATI currently supports. Makefiles are used to build the Wild Magic distribution. You should have a 3D graphics card installed in your machine with the appropriate drivers from your graphics card manufacturer. Such drivers can be downloaded from the manufacturer's web site. These are usually not plug-and-play and require some effort to install, but our experience with this has been not too painful.

2.3 Macintosh

We are now running Mac OS X (10.4.6) using Xcode 2.3 on a 64-bit Power Mac G5. The code used to run on a 32-bit Power Mac G4, but we no longer have that machine to test on.

The Cg Toolkit from NVIDIA is available now for the Macintosh. You can download this from NVIDIA's web site. We have compiled all the Cg programs on a Microsoft Windows XP machine.

2.4 Platform Differences

Platform differences are encapsulated in a small number of files. Inclusion of files or portions of files is controlled by preprocessor defines. In the core engine itself, the files

```
GeometricTools\WildMagic4\LibFoundation\System\Wm4Platforms.h
GeometricTools\WildMagic4\LibFoundation\System\Wm4System.cpp
```

encapsulate some basic system operations that are platform specific. The interface files

```
GeometricTools\WildMagic4\LibFoundation\System\Wm4System.{h,inl}
```

are platform independent.

The Microsoft Windows platform requires that the preprocessor symbol `WIN32` be defined. The Microsoft compilers already define these, so you do not need to add a define to the project settings. The Macintosh platform requires that `__APPLE__` be defined. Xcode automatically defines this, so you do not need to add a define to the project settings. If you are using MINGW, the `make` files need to define `MINGW`. As more platforms are added, the preprocessor mechanism in `Wm4Platforms.h` must be modified.

3 Compiling the Distribution

The method for compiling the libraries and applications depends on which platform you are working. If you plan on installing the source code on only one platform, you need only read the subsection related to that platform. Each platform has scripts to automatically compile the distribution. If you choose to manually compile the distribution, you must compile the Foundation, Graphics, Physics, Renderer, and Application projects *in this order*. The specific details for this are given in the following sections.

3.1 Microsoft Windows XP and Visual Studio

If you plan on using Microsoft DirectX for your renderer API, you must have the DirectX SDK installed *before* building the Wild Magic distribution. You can download the SDK from Microsoft's web site. We have tested the distribution using the DirectX SDK (April 2006) release. If you plan to use OpenGL for your renderer API, there are no prerequisites. Wild Magic has its own OpenGL extension wrapper and does not use GLUT.

3.1.1 Automatic Compilation

Automatic compilation is supported by the following executable

```
GeometricTools/WildMagic4/Wm4Installer.exe
```

This program requires various command-line parameters. Open a command window, change directory to the `WildMagic4` directory, and run the program without parameters. You will see the following output, where `X` is whatever drive you have unzipped the distribution to.

```
X:\GeometricTools\WildMagic4>Wm4Installer.exe
The Wild Magic Installer must be executed from the WildMagic4 folder in
the GeometricTools distribution.  The usage is:
    Wm4Installer
        -m [msvc60, msvc70, msvc71, msvc80, msvc80e]
        -g [opengl, directx, soft]
        -c [debug, debugdll, debugmem, release, releasedll, releasemem]
All options are required.  For example, to use Microsoft Visual C$++$ 7.1,
OpenGL, static libraries, and release mode, you must execute:
    Wm4Installer -m msvc71 -g opengl -c release
The compiler location is looked up in the Windows Registry.  If it cannot
be found, the installer will display a message to that effect.
```

Use the option `msvc80` for the professional version of Visual Studio .NET 2005. Use `msvc80e` for the freely available version Visual C++ 2005 Express Edition. A log file named `Wm4Installer.log` is created with the starting and ending times for compiling the configuration you selected. The compiler output is sent to the command window. To redirect this to a file for later browsing use, for example,

```
Wm4Installer -m msvc71 -g opengl -c release >> Log_71_ogl_rel.txt
```

The installer program compiles only the Foundation, Graphics, Physics, Renderers, Application libraries, the sample applications, and the `ScenePrinter` tool. The other tools need to be compiled separately, because some of them require other packages to be installed.

3.1.2 Manual Compilation

Should you choose to build the projects one at a time using Microsoft Visual Studio, please note that the sample applications and tools depend on other libraries being built first. These libraries perform post-build copies of header files to the folder

```
GeometricTools/WildMagic4/SDK/Include
```

and library files to the subfolders of

```
GeometricTools/WildMagic4/SDK/Library
```

The sample applications rely on files to be in these folders.

Build the following projects *in order*. The paths listed are relative to the top-level folder

```
GeometricTools/WildMagic4
```

The `xx` string is for version 60, 70, 71, or 80. The `ext` string is `dsp` for VC60 and `vcproj` for VC70, VC71, or VC80.

```
// The foundation on which everything depends.
LibFoundation/LibFoundation_VCxx.ext

// The platform-independent graphics library and scene graph management.
// Needs LibFoundation.
LibGraphics/LibGraphics_VCxx.ext

// The platform-independent physics library.
// Needs LibFoundation.
LibPhysics/LibPhysics_VCxx.ext

// The Direct3D renderer library.
// Needs LibFoundation and LibGraphics.
Renderers/Dx9Renderer/Dx9Renderer_VCxx.ext

// The OpenGL renderer library.
// Needs LibFoundation and LibGraphics.
Renderers/OpenGLRenderer/WglRenderer_VCxx.ext

// The software renderer library.
// Needs LibFoundation and LibGraphics.
Renderers/SoftRenderer/WinSoftRenderer_VCxx.ext

// The Direct3D-renderer application library.
// Needs LibFoundation, LibGraphics, LibPhysics, and Dx9Renderer.
Applications/Dx9Application_VC80.ext

// The OpenGL-renderer application library.
// Needs LibFoundation, LibGraphics, LibPhysics, and WglRenderer.
Applications/WglApplication_VC80.ext

// The software-renderer application library.
// Needs LibFoundation, LibGraphics, LibPhysics, and WinSoftRenderer.
Applications/WglApplication_VC80.ext
```

After you have built these libraries, you can build any sample application. The tools also require these libraries to be built first, but some tools have additional prerequisites. See the tools section of this document for details.

3.1.3 Build Configurations

The foundation, renderer, and application libraries each have 7 different build configurations named as shown.

```
Debug
Debug DLL
Debug Memory
Release
Release DLL
Release Memory
NoPCH
```

The libraries are stored in appropriately named subdirectories of `GeometricTools/WildMagic4/Library`.

The three configurations containing `Debug` link to the "Multi-threaded Debug DLL" run-time libraries provided by the compilers. The three configurations containing `Release` link to the "Multi-threaded DLL" run-time libraries.

The `Debug`, `Debug Memory`, `Release`, and `Release Memory` configurations produce static libraries.

The `Debug DLL` and `Release DLL` configurations produce dynamic libraries for the foundation and renderer libraries, but static application libraries. The application libraries implement the entry function `main`, which cannot be shared. However, the application libraries built with the `Debug DLL` and `Release DLL` configurations do use the dynamic foundation and renderer libraries.

The two configurations containing `Memory` differ from their static-library counterparts only in that they define the preprocessor symbol `WM4_MEMORY_MANAGER`. This enables run-time memory checking to trap memory leaks. Any application with memory checking enabled will produce a report on termination, stored in the text files `MemoryReportDebug.txt` or `MemoryReportRelease.txt`.

The previously mentioned configurations all use a precompiled header system. Although leading to faster compilation, such systems can hide circular dependencies and implicit dependencies in header files. The configuration `NoPCH` does not use the precompiled header system. We use this only to catch problems with circular and implicit dependencies, which usually manifest themselves by failure to compile successfully.

The sample applications each have 18 different build configurations names as shown.

```
Dx9 Debug,              Sft Debug,              Wgl Debug,
Dx9 Debug DLL,          Sft Debug DLL,          Wgl Debug DLL,
Dx9 Debug Memory,       Sft Debug Memory,       Wgl Debug Memory,
Dx9 Release,            Sft Release,            Wgl Release,
Dx9 Release DLL,        Sft Release DLL,        Wgl Release DLL,
Dx9 Release Memory,     Sft Release Memory,     Wgl Release Memory
```

The six prefixed with `Dx9` use the DirectX renderer, the six prefixed with `Sft` use the software renderer, and the six prefixed with `Wgl` use the OpenGL renderer. When you build a project manually, it is your responsibility to select the build configuration that you desire. If you have built only the `Debug` foundation, renderer, and application libraries, and then try to build the `Dx9 Debug DLL` configuration for a sample, you will get linker errors.

Some samples are console applications, which do not require a renderer. It is possible to create yet another application-library type just for console applications, but we did not in order to simplify the automatic compiling program. The `Wm4Installer.exe` program will iterate through the subdirectories of a `Samples` directory, open the project file, and compile the selected configuration. The introduction of a console application library would require us to hard-code into the installer a list of projects that are window-based and a list of projects that are console-based. Each new sample added to the distribution would require updating the installer program. By embedding the console support into the renderer-specific application libraries, we (and you) can simply add new samples that the installer will automatically compile.

3.1.4 Precompiled Headers

Wild Magic Version 4 uses the precompiled header system provided by the compilers. The precompiled header files are of the form `Wm4FooBarPCH.h` and the creation source files are of the form `Wm3FooBarPCH.cpp`.

The preprocessor symbols now include `WM4_USE_PRECOMPILED_HEADERS`. If you want to disable the precompiled header sytem:

1. Remove `WM4_USE_PRECOMPILED_HEADERS` from the preprocessor definitions in the build configuration of interest.
2. For the `Wm4FooBarPCH.cpp` files, change the "Create/Use Precompiled Header" option from `Create Precompiled Header (/Yc)` to `Not Using Precompiled Headers`.
3. For all other source files, change the "Create/Use Precompiled Header" option from `Use Precompiled Header (/Yu)` to `Not Using Precompiled Headers`.

You should be aware that in exchange for faster compilation using precompiled headers, you might get cryptic error messages, you risk having circular headers if you are not careful (these will not compile when precompiled headers are disabled), and sometimes multiple builds are required for unknown reasons. Also, if you are not careful about adding include directives in your source files, those files might not compile when precompiled headers are turned off. Our measurements of the precompiled header builds showed a 25% reduction in compile time.

3.1.5 Naming of the Executables

The sample executables are created in their project directories. The name of the executable stores the compiler version (now only VC80), the renderer type, and the configuration type. For example,

```
SampleGraphics\BillboardNodes\Billboard_VC80WglDebSta.exe
SampleGraphics\BillboardNodes\Billboard_VC80Dx9RelDyn.exe
SampleGraphics\BillboardNodes\Billboard_VC80WglRelMem.exe
```

The first executable uses OpenGL and is linked to the debug static libraries (`Debug` configuration). The second executable uses DirectX and is linked to the release dynamic libraries (`Release DLL` configuration). The third executable uses OpenGL and is linked to the release memory-checking libraries (`Release Memory` configuration).

3.1.6 Running the Samples Based on Dynamic Libraries

In order to run the dynamic library versions of the samples, the dynamic libraries (DLLs) themselves must be found for loading. You can always copy the DLLs to the executable directory or to the Windows system directory, but we prefer adding some environment variables and modifying the PATH environment variable. See Section 1.7 for details.

3.2 Microsoft Windows XP and MinGW

We have formal support for compiling using Minimalist GNU for Windows (MINGW) and using the environment provided by Minimal System (MSYS). Both packages are freely downloadable. Specifically, we have compiled and tested using the versions MINGW 4.1.0 and MSYS 1.0.10.

3.2.1 Automatic Compilation

After installing these packages to drive X of some machine, modify the file `X:\msys\1.0\etc\fstab` to contain

```
X:/mingw  /mingw
Y:/GeometricTools/WildMagic4  /WildMagic4
```

where `Y` is the drive on which Wild Magic installed (assuming you installed Wild Magic to the root). To start the compilation process, open an MSYS terminal window via the shortcut link set up on the Windows Desktop by the installer, change directory to `/WildMagic`, and execute the `make` file.

```
make -f makefile.wm4 CFG={config} SYS=mingw, GRF={graphics}
```

where `config` is one of (Debug,Release,DebugMemory,ReleaseMemory) and `graphics` is one of (OpenGL,Soft). For example, to build the Debug configuration using the OpenGL renderer,

```
make -f makefile.wm4 CFG=Debug SYS=mingw GRF=OpenGL
```

3.2.2 Manual Compilation

It is possible to compile the projects manually, one at a time. The order of project builds is

```
LibFoundation
LibGraphics
LibPhysics

// for OpenGL rendering
LibRenderers/OpenGLRenderer (WGL renderer)
LibApplications/OpenGLApplication (WGL application)
```

```
// for software rendering
LibRenderers/SoftRenderer (WinSoft renderer)
LibApplications/SoftApplication (WinSoft application)
```

The syntax for compiling a project is easily inferred by reading the `make` files. After the core libraries are built, you may build any sample application.

3.3 Linux

We have installed Red Hat Fedora Core 3 on our Linux platform. The machine uses an ATI Radeon X1300 Pro.

3.3.1 Automatic Compilation

Wild Magic is compiled by using the `make` file in the `GeometricTools/WildMagic4` directory.

```
make -f makefile.wm4 CFG={config} SYS=linux, GRF={graphics}
```

where `config` is one of (Debug,Release,DebugMemory,ReleaseMemory) and `graphics` is one of (OpenGL,Soft). For example, to build the Debug configuration using the OpenGL renderer,

```
make -f makefile.wm4 CFG=Debug SYS=linux GRF=OpenGL
```

3.3.2 Manual Compilation

It is possible to compile the projects manually, one at a time. The order of project builds is

```
LibFoundation
LibGraphics
LibPhysics

// for OpenGL rendering
LibRenderers/OpenGLRenderer (GLX renderer)
LibApplications/OpenGLApplication (GLX application)

// for software rendering
LibRenderers/SoftRenderer (XSoft renderer)
LibApplications/SoftApplication (XSoft application)
```

The syntax for compiling a project is easily inferred by reading the `make` files. After the core libraries are built, you may build any sample application.

3.4 Macintosh

The Macintosh distribution comes with support for Apple's Xcode 2.3 running on Macintosh OS X 10.4.6. We have no support for Metrowerks CodeWarrior.

It is essential that you configure Xcode's Building settings in the following manner. From the menu bar with Xcode active, select

```
Xcode | Preferences...
```

then select the `Buildings` icon. You need to check the radio button for

```
Put build products in project directory
```

You also need to check the radio button for

```
Put intermediate build files with build products
```

The Foundation file `Wm4System.cpp` has a function named `GetPath` for changing the current directory to the project directory in order that relative paths to application input data are correct. This function has a preprocessor define `WM4_USE_XCODE2`, which is enabled. If you are still using Xcode 1.5, you need to comment out the define in order for the relative path system to work.

3.4.1 Automatic Compilation

We have provided a shell script, `MacBuildWm4.sh`, to allow you to build the distributions from a Terminal window. Launch a Terminal window and change directory to the location of the Wild Magic distribution. On our Macintosh, we placed the distribution in a user directory,

```
Mac:~/GeometricTools/WildMagic4
```

Make sure the script has executable permissions. You can set the permissions from a terminal window,

```
chmod 755 MacBuildWm4.sh
```

The Foundation, Graphics, Physics, Renderer, and Application projects also use scripts to do post-build copies of headers and libraries. These are in the directories

```
GeometricTools/WildMagic4/LibFoundation
GeometricTools/WildMagic4/LibRenderers/OpenGLRenderer
GeometricTools/WildMagic4/LibRenderers/SoftRenderer
GeometricTools/WildMagic4/LibApplications/OpenGLApplication
GeometricTools/WildMagic4/LibApplications/SoftApplication
```

The script names are

```
CopySdkDebugStatic.sh
CopySdkDebugDynamic.sh
CopySdkDebugMemory.sh
CopySdkReleaseStatic.sh
CopySdkReleaseDynamic.sh
CopySdkReleaseMemory.sh
```

Also make sure the scripts have executable permissions by using `chmod`.

The syntax for compiling is

```
./MacBuildWm4.sh graphics config libtype buildtype
```

where `graphics` is one of (Agl,Soft), `config` is one of (Debug,Release), `libtype` is one of (Static,Dynamic), and `buildtype` is one of (build,clean). The script executes the command-line version of Xcode, namely, `xcodebuild`. For example, to build the projects for OpenGL rendering, release configuration, and dynamic libraries, you would use

```
./MacBuildWm4.sh Agl Release Dynamic build
```

3.4.2 Manual Compilation

If you choose to build the projects manually, they must be built in the following order. The path is relative to `GeometricTools/WildMagic3`. It is possible to compile the projects manually, one at a time. The order of project builds is

```
LibFoundation
LibGraphics
LibPhysics

// for OpenGL rendering
LibRenderers/OpenGLRenderer (AGL renderer)
LibApplications/OpenGLApplication (AGL application)

// for software rendering
LibRenderers/SoftRenderer (MacSoft renderer)
LibApplications/SoftApplication (MacSoft application)
```

After these libraries are built, you can build any sample or tool.

4 Sample Applications

The CDROM master was due before we had a chance to describe all the sample applications, of which there are many. As time permits, we will add the descriptions here. The installation and release notes document has a version number to allow you to keep track of when the document has been modified.

5 Tools

A brief discussion of the tools is given in this section. The tools projects are found in

```
GeometricTools/WildMagic4/Tools
```

Some of the tools are Microsoft Windows only. Each subsection mentions the platforms on which the tools will run.

You will notice the omission of the `3DS` and `LWO` importers and of the `3dsmax` and `Maya` exporters. The port of these from Wild Magic 3 to Wild Magic 4 was not finished by the time the CDROM master needed to be delivered to the publishers. By the time *3D Game Engine Design, 2nd Edition* appears in print, these tools should be available at our website.

5.1 BitmapFontCreator

Runs on Windows.

This is what we used to create the bitmap font for text in the OpenGL renderers. Specifically, the program created the source file `Wm4OpenGLVerdanaS16B0I0.cpp` for the Verdana font. You may modify the program to create source files for other fonts. However, you should do so only for fonts that are freely usable or have been licensed by you.

5.2 Bmp24ToWmif

Runs on Windows.

The program is a converter from Windows BITMAP format (bmp) to the Wild Magic image format (wmif). The usage is

```
Bmp24ToWmif myfile.bmp [myfile.alpha.bmp]
```

The input bitmaps must be 24-bit RGB. The output file is `myfile.wmif`. If you supply only `myfile.bmp`, the output is a 24-bit RGB image in the wmif format. To provide an alpha channel, you supply the second file, `myfile.alpha.bmp`, which must be 24-bit, but only the red channel is used as the alpha value. The output file is a 32-bit RGBA image in the wmif format.

5.3 CreateNormalMap

Runs on Windows.

The program takes a 24-bit RGB Windows BITMAP and creates a gray-scale image and a normal map for use in bumpmapping. A sample image, `Brick.bmp`, and its output files are included with the distribution to illustrate what the program does.

5.4 GenerateProjects

Runs on Windows. It should run on Linux and Macintosh but we have not provided make files or Xcode project files for t his.

This program generates VC60, VC70, VC71, VC80, and Xcode 2.3 project files. We have embedded the project-template code in the executable. This program saves you the pain of having to manually edit new project files.

5.5 ScenePrinter

Runs on Windows, Linux, Macintosh.

This is a converter that processes a Wild Magic object file (wmof) and produces an ASCII representation.

5.6 WmifToBmp24

Runs on Windows.

The program is a converter from the Wild Magic image format (wmif) to the Windows BITMAP format (bmp). The usage is

```
WmifToBmp24 myfile.wmif
```

If the wmif file is RGB888, the output file is a 24-bit bmp image named myfile.bmp. If the wmif file is RGBA8888, two output files are generated. The file myfile.bmp is a 24-bit BITMAP image that contains the RGB portion of myfile.wmif. The file myfile.alpha.bmp is also a 24-bit BITMAP image, is gray scale ($R = G = B$), and contains the A portion of myfile.wmif. The converter only supports Wild Magic RGB888 and RBGA8888 for now.

6 Known Problems

This section will be updated as known problems are reported. The installation and release notes document has a version number to allow you to keep track of when the document has been modified.

7 The Infinite TO DO List

This is a list of items that we plan on doing at some time. No particular priorities have been assigned to these, and they are in no particular order.

1. Write the automated string-tree and streaming source code generator.
2. Complete the OpenGL extension handling.

3. The Program class needs a parser for an input stream (in addition to an input filename). The parser also needs to be modified to handle arrays of uniform values (the old CgToWm4 converter had this).

4. The DirectX render-to-texture appears to work, but the cubemap demo is not working regarding re-rendering of the environment.

5. The DX9 renderer needs a fallback for when the Shader Model is not powerful enough for the precompiled SM2 programs. On my notebook that appears to be SM1, the OnLoadVProgram and OnLoadPProgram functions fire off the asserts after CreateVertexProgram and CreatePixelProgram.

6. Get the pixel buffer object code working again and use it as a fallback if framebuffer objects are not supported.

7. Add classes FloatN (N = 1,2,3,4) and FloatNxM (N = 1,2,3,4; M = 1,2,3,4). All such classes need to be 16-byte aligned for SIMD and game console purposes. Include arithmetic among such objects. Remove use of `Vector*`, `Matrix*`, `ColorRGB*` from LibGraphics.

8. Update the importers and exporters to use Collada. This will be part of the upcoming book on importers, exporters, and post-processing of scenes.

9. Load/compile `*.cg` programs on the fly, even if the result is only to write the asm output to disk to be loaded by the rendering system. Add support for `*.fx`?

10. Support floating-point buffers (32-bit RGBA is currently supported).

11. Support Anisotropic filtering (easy to do in OpenGL).

12. Support Full-screen rendering.

13. Add a memory manager and replace the macros `WM4_NEW` and `WM4_DELETE` by ones using the memory manager (in the LibGraphics code).

14. Factor picking code out of the class interfaces.

15. Implement the dynamic collision system described in the book. Create classes such as Collider and SphereCollider.