Wild Magic Version 5.13
Installation Manual and Release Notes

Document Version 5.13.0
August 11, 2014

Contents

1 Introduction

You are about to install Wild Magic 5.13 (WM5). Version 5.0 shipped with the book, *Game Physics, 2nd edition*. The CD-ROM contains source code that compiles and runs on platforms using Microsoft Windows 7, 8.0, or 8.1; Linux; and Macintosh OS X on Intel-based maches. We currently use Microsoft Visual Studio. The distribution ships with projects for MSVS 2010, MSVS 2012, and MSVS 2013.

You should visit our web site regularly for updates, bug fixes, known problems, 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. The source files themselves are labeled with a version number of the form *5.minor.revision*, where *minor* is the minor version in which the file shipped and where *revision* is the number of times the file has been revised. The source files in the release version Wild Magic 5.0 are labeled with 5.0.0.

1.1 License

Wild Magic 5 uses the Boost License. The license in its entirety is listed next.

```
Boost Software License - Version 1.0 - August 17th, 2003

Permission is hereby granted, free of charge, to any person or organization
obtaining a copy of the software and accompanying documentation covered by
this license (the "Software") to use, reproduce, display, distribute,
execute, and transmit the Software, and to prepare derivative works of the
Software, and to permit third-parties to whom the Software is furnished to
do so, all subject to the following:

The copyright notices in the Software and this entire statement, including
the above license grant, this restriction and the following disclaimer,
must be included in all copies of the Software, in whole or in part, and
all derivative works of the Software, unless such copies or derivative
works are solely in the form of machine-executable object code generated by
a source language processor.

THE SOFTWARE IS PROVIDED ‘‘AS IS’’, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT
SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE
FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE,
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN THE SOFTWARE.
```

1.2 Copying the Distribution to Your Machine

You may copy the book CD-ROM contents directly to your hard disk. The top-level folder of the distribution is named `GeometricTools`. If you have a zip file, the organization of the distribution is the same. It does not

matter where you copy the distribution. The projects all use relative paths and do not rely on the top-level directory being located at the root of the drive. Data files are found using an environment variable that you must create. The folder hierarchy is shown on the next page.

```
GeometricTools
    WildMagic5
        Bin                         // convenient place for tool executables
        Data                        // various data files used by Wild Magic
            Bmp                     // Microsoft Windows BMP files
            CgShaders               // NVIDIA Cg FX files
            Im                      // Wild Magic image files for image processing
            Wmfx                    // Wild Magic compiled FX files
            Wmof                    // Wild Magic scene object files
            Wmtf                    // Wild Magic texture files
            Wmvf                    // Wild Magic visual files (vertex formats, vertex and index buffers)
        Documentation               // information about the Wild Magic distribution
        LibApplications             // application layer (platform-independent files in top-level folder)
            AglApplication          // Macintosh application layer (deprecated, for OS X 10.6 [Snow Leopard])
            GlutApplication         // Macintosh application layer (for OS 10.8 [Mountain Lion])
            GlxApplication          // Linux application layer
            WinApplication          // Microsoft Windows application layer
        LibCore                     // core system
        LibGraphics                 // graphics system
            Renderers               // renderer interface (platform-independent files in top-level folder)
                AglRenderer         // Apple OpenGL renderer (deprecated, for OS X 10.6 [Snow Leopard])
                Dx9Renderer         // Microsoft Windows DirectX 9 renderer
                GlutRenderer        // Glut OpenGL renderer (for OS X 10.8 [Mountain Lion])
                GlxRenderer         // X Windows OpenGL renderer
                OpenGLRenderer      // platform-independent OpenGL renderer files
                WglRenderer         // Microsoft Windows OpenGL renderer
        LibImagics                  // platform-independent image processing system
        LibMathematics              // platform-independent mathematics system
        LibPhysics                  // platform-independent physics system
        License                     // contains a copy of the Boost License
        SampleGraphics              // sample applications for graphics
        SampleImagics               // sample applications for image processing
        SampleMathematics           // sample applications for mathematics
        SamplePhysics               // sample applications for physics
        SDK                         // top-level folder for headers and libraries (generated during build)
            Include                 // headers copied from the engine folders
            Library                 // compiled libraries for the engine
                v100                // $(PlatformToolset) MSVS 2010
                    Win32           // compiled 32-bit libraries
                        Debug       // compiled static libraries for debug builds
                        DebugDLL    // compiled dynamic libraries for debug builds
                        Release     // compiled static libraries for release builds
                        Release     // compiled dynamic libraries for release builds
                    x64             // compiled 64-bit libraries
                        Debug       // compiled static libraries for debug builds
                        DebugDLL    // compiled dynamic libraries for debug builds
                        Release     // compiled static libraries for release builds
                        Release     // compiled dynamic libraries for release builds
                v110                // $(PlatformToolset) MSVS 2012
                    Win32           // compiled 32-bit libraries
                        Debug       // compiled static libraries for debug builds
                        DebugDLL    // compiled dynamic libraries for debug builds
                        Release     // compiled static libraries for release builds
                        Release     // compiled dynamic libraries for release builds
                    x64             // compiled 64-bit libraries
                        Debug       // compiled static libraries for debug builds
                        DebugDLL    // compiled dynamic libraries for debug builds
                        Release     // compiled static libraries for release builds
                        Release     // compiled dynamic libraries for release builds
                v120                // $(PlatformToolset) MSVS 2013
                    Win32           // compiled 32-bit libraries
                        Debug       // compiled static libraries for debug builds
                        DebugDLL    // compiled dynamic libraries for debug builds
                        Release     // compiled static libraries for release builds
                        Release     // compiled dynamic libraries for release builds
                    x64             // compiled 64-bit libraries
                        Debug       // compiled static libraries for debug builds
                        DebugDLL    // compiled dynamic libraries for debug builds
                        Release     // compiled static libraries for release builds
                        Release     // compiled dynamic libraries for release builds
        Tools                       // tools to support development
            BmpColorToGray          // convert 24-bit color BMP to 24-bit gray BMP
```

```
BmpToWmtf              // convert 24-bit color BMP to Wild Magic WMTF file
GenerateProjects       // create application project files
ObjMtlImporter         // source code for loading .obj and .mtl files
WmfxCompiler           // Create *.wmfx from *.fx (requires NVIDIA Cg installed)
WmtfViewer             // viewer for *.wmfx files containing 2D textures
```

The `Sample*` folders contain many applications. Listing them here would make the displayed hierarchy difficult to read.

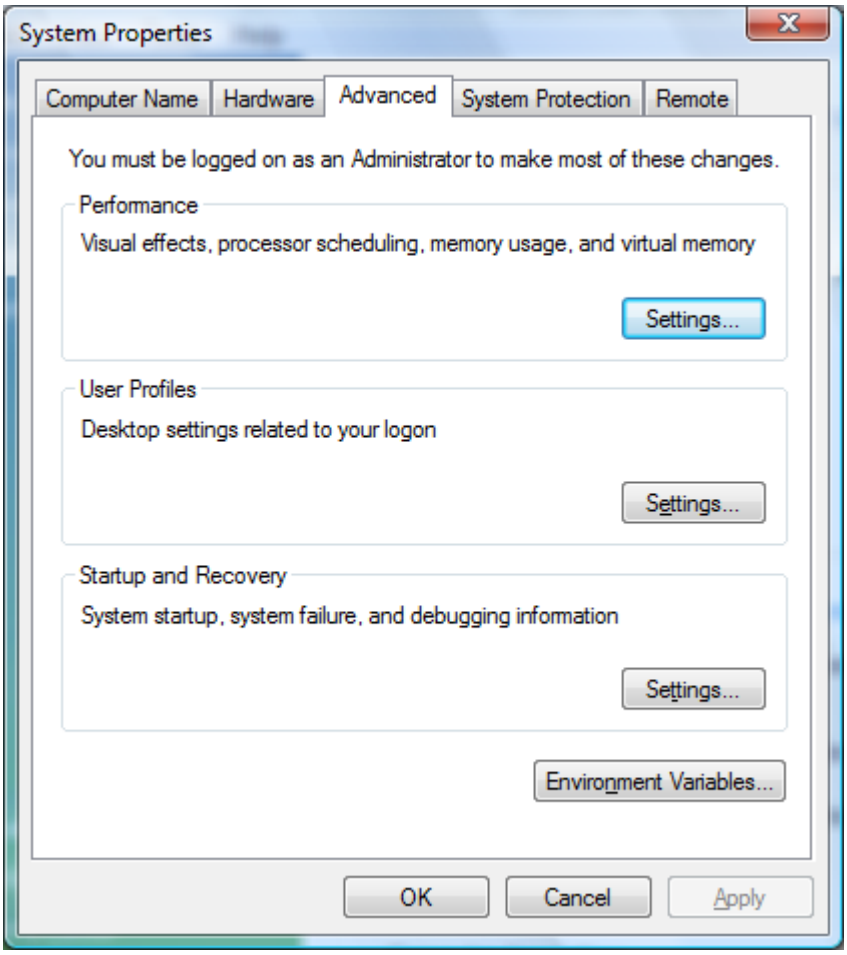
1.3 Environment Variables

WM5 uses relative paths and supports a list of paths for searching for data files. To accomplish this, the application layer accesses an environment variable, `WM5_PATH`, whose value must be set to the installation location of WM5. 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 the hard drive. If you copied the top-level folder elsewhere, modify the paths in the examples accordingly.

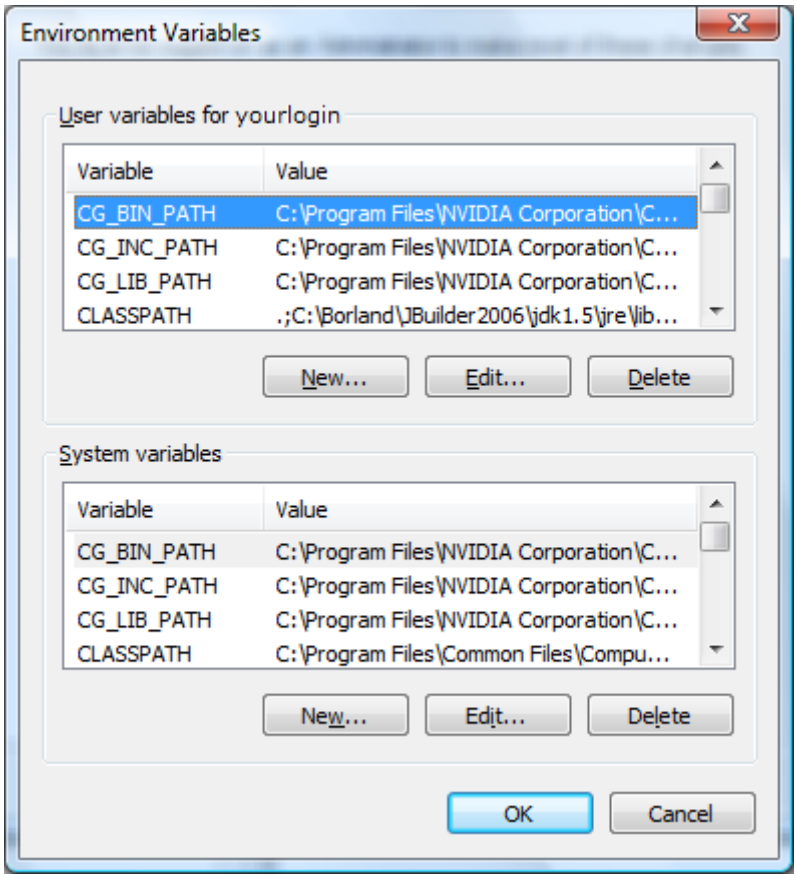
If you are running Microsoft Visual Studio and change an environment variable or add new ones, you need to exit Visual Studio and restart it. When Microsoft Visual Studio starts, it loads the current environment variables and makes copies. The restart is necessary for it to detect your changes.

1.3.1 Microsoft Windows 7 or 8

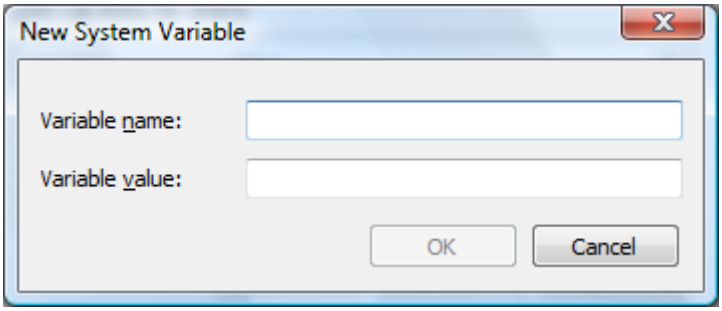
You may set an environment variable by using the Control Panel. In the Control Panel window, select the "System and Maintenance" link. In the window that appears, select the "System" link. On the left side of the window that appears, select the "Advanced system settings" link. You will see the dialog



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 `WM5_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 `WM5_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\WildMagic5
```

1.3.2 Macintosh

With the introduction of Mac OS X 10.7 [Lion], access to environment variables via the library function `getenv` appears to have been disabled. In previous versions of the operating system, to have environment variables automatically loaded when logging in, you need to have a file

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

whose contents include a dictionary entry for Wild Magic 5, as illustrated next.

```
<?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>WM5_PATH</key>
    <string>/Users/YOURLOGIN/GeometricTools/WildMagic5</string>
</dict>
</plist>
```

You must replace `YOURLOGIN` by your actual login name. A skeleton file containing this information is in the `GeometricTools/WildMagic5` 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.

On Mac OS X 10.9.1 [Mavericks], the `environment.plist` mechanism still appears to work from the perspective of a shell. When you run a Terminal window and type 'set', you will see a list of the current environment variable, including the `WM5_PATH` variable that was loaded at login. Unfortunately, a call to `getenv("WM5_PATH")` returns a null pointer. The same is true for other important environment variables such as `USER`. On Mac OS X 10.8 [Mountain Lion], it appears the plist mechanism is now fully disabled. A 'set' call does not show the `WM5_PATH` variable. The workaround we use for now in `Wm5Application.cpp` is the following. The root folder `WildMagic5` has a file named `wm5path.txt`. Edit this file and replace the path by the one that you added to `environment.plist`. Copy this file to the `.MacOSX` folder. Launching a shell from within a program and executing a command still appears to work. So in `Wm5Application.cpp` we have

```
#ifndef __APPLE__
    Application::WM5Path = Environment::GetVariable("WM5_PATH");
#else
    if (system("cp ~/.MacOSX/wm5path.txt tempwm5path.txt") == 0)
    {
        std::ifstream inFile("tempwm5path.txt");
        if (inFile)
        {
            getline(inFile, Application::WM5Path);
            inFile.close();
            system("del tempwm5path.txt");
        }
    }
#endif
    if (Application::WM5Path == "")
    {
        assertion(false, "Please set the WM5_PATH environment variable.\n");
        std::ofstream outFile("ApplicationError.txt");
```

```
    if (outFile)
    {
        outFile << "Please set the WM5_PATH environment variable.\n";
        outFile.close();
    }
    return INT_MAX;
}
// Other application-layer code...
```

1.3.3 Linux

We use Red Hat Fedora Core 20 with a Bash shell and define the following variable in the `.bashrc` file,

```
WM5_PATH=/home/YOURLOGIN/GeometricTools/WildMagic5 ; export WM5_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
WM5_PATH=/home/YOURLOGIN/GeometricTools/WildMagic5 ; export WM5_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. For other versions of Linux or other shells, consult your user's guide on how to create an environment variable.

2 Prerequisites for Compiling and Using Wild Magic

Wild Magic contains a lot of code for multiple platforms and graphics APIs. As such, there is no *go button* you can press and have everything set up properly. We are not in the business of writing installers for the platforms we support, so you must set up your own environment for our code to work properly.

2.1 Microsoft Visual Studio

We use Microsoft Visual Studio 2013 for compiling WM5, but projects are also provided for MSVS 2010 and MSVS 2012. An *Express edition* of the compiler is freely available from Microsoft's web site.

You might have constraints that require you to use a previous version; for example, you might have third-party libraries that were compiled with a previous version and you need Wild Magic libraries to be compiled with that same version. Two options to deal with this:

- Contact the vendor of your third-party libraries and request versions compiled with Microsoft Visual Studio 2008 or earlier.
- Create your own Wild Magic project file for the version of Microsost Visual Studio you are using. The simplest thing to do is copy all the source and header files from the Wild Magic libraries into your project's working directory. You should select only one type of renderer and one type of application; for example, if you want to use only DirectX 9, then copy the `LibGraphics/Renderers/Dx9Renderers` files into your folder but not the files of the other subdirectories of `LibGraphics/Renderers`. Do the same for the application layer (if you need this library). You must define some preprocessor values. In the current example, you need to define `WM5_USE_DX9` and `WM5_USE_PRECOMPILED_HEADERS`. If you chose OpenGL, you need `WM5_USE_OPENGL` instead of the DirectX 9 define.

Be aware that older versions of the compiler might not be able to compile some of our files, because the compiler is not as ANSI compliant as the later versions.

2.2 Apple Xcode 5.0 and Mac OS X 10.9 [Mavericks]

Consider this mainly a section of issues (and complaints). Apparently, Apple is pushing developers towards using only Cocoa and Objective C, and to obtain OS services only through them. This is making it increasingly difficult for us to provide portable code.

- As mentioned in Section 1.3.2, `getenv` returns only null pointers. We had to hack the application layer in order to set our `WM5_PATH` variable.
- Much of Apple OpenGL (AGL) is deprecated. It has been for some time, but with Mac OS X 10.7 and Xcode 4.1 (and later), enough of the window-based setup has been removed, so our AGL code is broken. We replaced this with GLUT, which is a minimal application layer that is not really sufficient even for some of our sample applications. With Mac OS X 10.9.1 and Xcode 5.0, even GLUT is now marked as deprecated. At some time we will spend time to write a Cocoa base layer and make callbacks into our C++ code, but it is not clear when we will have the time (or desire) to do so.
- The Xcode 5.0 application interface has a recommended setting of writing compiler output to hidden folders off the user's home directory. This is of the form

  ```
  /users/YourLogin/Library/Developer/Xcode/DerivedData/AppName-HaskKey/Build/Products/Default
  ```

  We left the setting this way. Any output files an application writes to disk are in the hidden directory when you run through the Xcode interface. Happy hunting!

2.3 Microsoft 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. WM5 was tested using the DirectX 9 SDK (June 2010), but there should not

be problems using later updates of DirectX 9. We have not yet written a Wild Magic 5 renderer for DirectX DirectX 11. DirectX 11 support is part of Wild Magic 6 that will ship in 2014. The Windows Kit folders that ships with Windows 8.0 and 8.1 have DX11 support, so you no longer need to download the SDI.

2.4 OpenGL and Extension Wrappers

We do not use third-party extension wrappers. If you want to use an extension wrapper, try GLEW, which is freely downloadable. We have rolled our own extension wrapper. Currently, we support only those ARB and EXT extensions that are used by Wild Magic. 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 extensions 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 as desired 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 `Wm5GlPlugin.h` and add the implementations to the source file, `Wm5GlPlugin.cpp`.

2.5 GLUT Support

We have added graphics renderer and application libraries that uses GLUT, but we do not have sample project configurations that use it. If you want to use this library, you must download the GLUT SDK. This library works for 32-bit applications, but we could not resolve a linker error when compiling for 64-bit (x64). The GLUT headers and libraries were copied to the following locations on a Windows 7 32-bit machine.

```
C:\Program Files\Microsoft SDKs\Windows\v7.0A\Include\gl\glut.h
C:\Program Files\Microsoft SDKs\Windows\v7.0A\glut32.lib
C:\Program Files\Microsoft SDKs\Windows\v7.0A\glut32.def
C:\Windows\System32\glut32.dll
```

On a 64-bit machine the program files paths start with

```
C:\Program Files (x86)
```

On a Windows 8 machine you need

```
C:\Program Files (x86)\Windows Kits\8.0\Include\um\gl\glut.h
C:\Program Files (x86)\Windows Kits\8.0\Lib\win8\um\x86\glut32.lib
C:\Program Files (x86)\Windows Kits\8.0\Lib\win8\um\x86\glut32.def
C:\Windows\System32\glut32.dll
```

On a Windows 8.1 machine you need

```
C:\Program Files (x86)\Windows Kits\8.1\Include\um\gl\glut.h
C:\Program Files (x86)\Windows Kits\8.1\Lib\winv6.3\um\x86\glut32.lib
C:\Program Files (x86)\Windows Kits\8.1\Lib\winv6.3\um\x86\glut32.def
C:\Windows\System32\glut32.dll
```

The paths to these files are automatically picked up via the MSVS macros `WindowsSDKDir` and `FrameworkSDKDir`.

2.6 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 have been using the command line compiler, `cgc version 3.0.0007, build date Jul 22 2010`, on Microsoft Windows machines. We have also tried the Macintosh version, and it works equally as well.

It is possible to use HLSL as long as the output is text-string assembly. See `LibGraphics/LocalEffects` for examples of effects that were created manually in the engine. You should be able to mimic this and use your HLSL strings. The `WmfxCompiler` tool can be modified to generate binary resources for HLSL, but we have not yet gone down that path, but will for the Wild Magic 6 Library (support for GLSL and HLSL separately, no unified shader handling).

3 Compiling the Distribution

The method for compiling the libraries and applications depends on which platform you are working on. 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, before attempting to compile samples, you must compile the library projects LibCore, LibMathematics, LibImagics, LibPhysics, LibGraphics, and LibApplication. The order of compilation is required:

1. Compile LibCore first.
2. Compile LibMathematics second. It depends on LibCore.
3. Compile LibImagics, LibPhysics, and Lib*Graphics. The order of these three projects is not important, but they all depend on LibMathematics.
4. Compile Lib*Applications. It depends on all the previous libraries.

The build configurations for the libraries include static library creation and dynamic library creation. The graphics and applications libraries also require you to select between DirectX9 and OpenGL (by choice of MSVS solution file).

3.1 Automatic Builds with Microsoft Visual Studio

To build the libraries, open the solution files

```
GeometricTools/WildMagic5/WildMagic5Dx9.sln  // To build the DirectX 9 version
GeometricTools/WildMagic5/WildMagic5Wgl.sln  // To build the OpenGL version
```

The sample applications now have references to the libraries. If you start by building a sample application, the libraries will be built for you.

3.2 Automatic Builds on Macintosh

Open a Terminal window and change directory to

```
GeometricTools/WildMagic5/
```

Execute the shell script

```
./MacBuildWm5.sh DebugStatic build
```

for the debug static library configurations or

```
./MacBuildWm5.sh ReleaseStatic build
```

for the release static library configurations. You can also compile dynamic libraries using

```
./MacBuildWm5.sh DebugDynamic build
```

for the debug static library configurations or

```
./MacBuildWm5.sh ReleaseDynamic build
```

To remove the results of the build, replace the word `build` on the command line by `clean`.

3.3 Automatic Builds on Linux

Open a Terminal window and change directory to

```
GeometricTools/WildMagic5/
```

Execute the command

```
make CFG=Debug -f makefile.wm5
```

for the debug static library configurations or

```
make CFG=Release -f makefile.wm5
```

for the release static library configurations. You can also compile dynamic libraries using

```
make CFG=DebugDynamic -f makefile.wm5
```

for the debug static library configurations or

```
make CFG=ReleaseDynamic -f makefile.wm5
```

To remove the results of the build, add the word `clean` after the word `make`.

3.4 Compiler Support for Windows Dynamic Link Libraries

To support dynamic link libraries (DLLs) under Microsoft Windows, 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 Core library project must export the symbols but any client project that links to the Core library must import the symbols. The Core library has a file named Wm5CoreLIB.h, whose contents are

```
#if defined(WM5_CORE_DLL_EXPORT)
    // For the DLL library.
    #define WM5_CORE_ITEM __declspec(dllexport)
#elif defined(WM5_CORE_DLL_IMPORT)
    // For a client of the DLL library.
    #define WM5_CORE_ITEM __declspec(dllimport)
#else
    // For the static library or Linux/Macintosh.
    #define WM5_CORE_ITEM
#endif
```

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

```
WM5_CORE_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 LibMathematics project is a client of the LibCore library and includes in its list of preprocessor definitions the symbol

```
WM5_CORE_IMPORT_DLL
```

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

The LibMathematics 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 `Wm5MathematicsLIB.h` in the LibMathematics project, whose contents are

```
#if defined(WM5_MATHEMATICS_DLL_EXPORT)
```

```
    // For the DLL library.
    #define WM5_MATHEMATICS_ITEM __declspec(dllexport)
#elif defined(WM5_MATHEMATICS_DLL_IMPORT)
    // For a client of the DLL library.
    #define WM5_MATHEMATICS_ITEM __declspec(dllimport)
#else
    // For the static library or Linux/Macintosh.
    #define WM5_MATHEMATICS_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 library project defines `WM5_MATHEMATICS_DLL_EXPORT`. To import symbols, the application client defines `WM5_MATHEMATICS_DLL_IMPORT`. Each library project in the WM5 distribution defines such a header file to enable importing and exporting of symbols. The LibImagics project has associated preprocessor definitions

```
WM5_IMAGICS_IMPORT_DLL, WM5_IMAGICS_EXPORT_DLL
```

The LibPhysics project has associated preprocessor definitions

```
WM5_PHYSICS_IMPORT_DLL, WM5_PHYSICS_EXPORT_DLL
```

The LibGraphics project has associated preprocessor definitions

```
WM5_GRAPHICS_IMPORT_DLL, WM5_GRAPHICS_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. The `BillboardNodes` sample application contains debug and release configurations that use the dynamic libraries, so you should look at the configuration settings to see how to generate your own configurations.

3.5 Finding Windows Dynamic Link Libraries at Run-Time

The compiled WM5 libraries are stored in the following directories:

```
GeometricTools/WildMagic5/SDK/Library/Debug
GeometricTools/WildMagic5/SDK/Library/DebugDLL
GeometricTools/WildMagic5/SDK/Library/Release
GeometricTools/WildMagic5/SDK/Library/ReleaseDLL
```

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/WildMagic5/SDK/Library/DebugDLL/Wm5Core110d.dll
GeometricTools/WildMagic5/SDK/Library/ReleaseDLL/Wm5Core110.dll
```

created by Microsoft Visual Studio 2012 (version 11.0, thus the suffix of 100 on the library names). 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 `WM5_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:

```
WM5_PATH=C:\GeometricTools\WildMagic5
WM5_PATH_BIN=C:\GeometricTools\WildMagic5\Bin
WM5_PATH_DEBUG_DLL=C:\GeometricTools\WildMagic5\SDK\Library\DebugDLL
WM5_PATH_RELEASE_DLL=C:\GeometricTools\WildMagic5\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 your current path using the syntax

```
PATH=<currentpath>;%WM5_PATH_BIN%;%WM5_PATH_DEBUG_DLL%;%WM5_PATH_RELEASE_DLL%
```

3.6 Finding Linux Shared Libraries at Run-Time

The compiled WM5 libraries are stored in the following directories:

```
GeometricTools/WildMagic5/SDK/Library/Debug
GeometricTools/WildMagic5/SDK/Library/DebugDynamic
GeometricTools/WildMagic5/SDK/Library/Release
GeometricTools/WildMagic5/SDK/Library/ReleaseDynamic
```

To support packaging of Wild Magic 5.12 in Debian Linux, the makefiles were modified to include an SONAME in the shared libraries, the libraries have prefix `*.so.5.12`, and symbolic links are set with extensions `*.so.5` and `*.so` so that libraries are found at run time. The Debian folks will arrange for the libraries to be in the standard locations one expects with Linux. If you build the shared libraries with our makefiles, you can run an application from a terminal window as follows.

```
// Terminal window with directory $WM5_PATH/SampleGraphics/BlendedTerrain
// and the dollar sign prompt.
$ LD_LIBRARY_PATH=$WM5_PATH/SDK/Library/DebugDynamic ./BlendedTerrain.DebugDynamic
```

4 Release Notes: Differences from Wild Magic 4

Several important concepts occur in Wild Magic 5 that either were not in or were handled differently in Wild Magic 4. A few are mentioned here, but you should also look at

Documentation/WildMagic5Overview.pdf

4.1 Preprocessor Defines Specified in Header Files

Each library has a header file that contains preprocessor defines. You may enable or disable these according to your needs. You should eventually read through all these files to see what control you have over the compilation.

For example, the core library, `LibCore`, has a header file named `Wm5CoreLIB.h`. The header wraps the platform-dependent information and exposes it based on flags provided by the compiler at hand–the compiler specifies them or you have added defines to the build system for that compiler. One of these flags, `WM5_USE_MEMORY`, is used to determine whether to use Wild Magic's memory management system which is used to track memory leaks.

As another example, the graphics library, `LibGraphics`, has a header file named `Wm5GraphicsLIB.h`. One of the commented out defines is `WM5_QUERY_PIXEL_COUNT`. During debugging, you might want to diagnose a drawing problem where some object is not renderer when you believe it should be. If you enable this define, the `Renderer::DrawPrimitive` function has code exposed that queries the graphics driver for the number of pixels drawn.

4.2 Reorganized Build Configurations

We now support Win32 and x64 builds, each having Debug, DebugDLL, Release, and ReleaseDLL. We have also factored out the DX9 and WGL support into separate project files.

- Removed support for VC80 and VC90. Support only for VC100 and VC110.
- Added x64 build configurations.
- Removed NoPCH build configurations.
- Switched to using macros to generate output directories for graphics-less projects
- Consistent naming of output directories:

```
_Output\$(PlatformToolset)\$(Platform)\$(Configuration)  // non-graphics projects
_Output\$(PlatformToolset)\$(Platform)\Dx9$(Configuration)  // DX9 graphics
_Output\$(PlatformToolset)\$(Platform)\Wgl$(Configuration)  // WGL graphics
```

- Added macros for include/library DX9 paths, using `$(DXSDK_DIR)`, and selecting the x86 or x64 library paths accordingly.
- Eliminated the default `vc100.pdb` symbol file, using instead `$(TargetName).pdb`
- Samples and tools now use warning level 4 instead of warning level 3.
- Samples and tools have SLN files with library dependencies so that they are self-contained for compiling, linking, and running. Added references to the libraries in addition to specifying build order to avoid a MSVS bug.

4.3 Memory Management

We have a class called `Memory` in the core library. It allows you to track memory usage and determine whether there are memory leaks. In Debug builds, when an application terminates, a file called `MemoryReport.txt` is written to the project folder. If there are memory leaks, the file contains information about each leak. If there are no memory leaks, the file is empty.

The class also has the ability to use the dimension of the allocation. The function `new` allocates a singleton, whereas the function `new[]` allocates a 1-dimensional array. We explicitly create allocators and deallocators for arrays with more than one dimension. These are wrapped by macros

```
MyClass* object0 = new0 MyClass(parameters);  // allocate a singleton
MyClass* object1 = new1<MyClass>(bound0);  // allocate a 1D array of objects
MyClass** object2 = new2<MyClass>(bound0,bound1);  // allocate a 2D array of objects
MyClass*** object3 = new3<MyClass>(bound0,bound1,bound2);  // allocate a 3D array of objects
MyClass**** object4 = new4<MyClass>(bound0,bound1,bound2,bound3);  // allocate a 4D array of objects

delete0(object0);
delete1(object1);
delete2(object2);
delete3(object3);
```

The actual number of memory allocations is minimized (n malloc calls for an n-dimensional array). The `Memory` class tracks the current pool of dynamic allocations (see the `msMap` member).

The class `Pointer0` supports reference-counted objects and automatic destruction when the reference count goes to zero. WM4 had such a class for singleton objects that were dynamically allocated; the class was called `Pointer`. WM5 has additional smart-pointer classes for multidimensional arrays. These classes also track the current pool of allocated smart-pointer objects (see the class `PointerBase`).

The `Memory` class is enabled in Debug builds but disabled in Release builds. In the latter case, the `newN` and `deleteN` macros wrap implementations using standard `new` and `delete`.

4.4 New Classes for Affine Algebra

We have added class `HPoint`, which represents a homogeneous point (x, y, z, w). A derived class is `APoint`, which represents a point of the form $(x, y, z, 1)$. Another derived class is `AVector`, which represents a vector of the form $(x, y, z, 0)$. The derived classes do not have virtual functions, so they each store a 4-tuple. Normally, a derived class destructor is virtual. A nonvirtual destructor hides the base-class destructor, so if you were to delete a derived-class object via a base-class pointer, the derived-class constructor would not be called. This is not a problem for us because (1) we do not reference the derived-class objects via base-class pointers (or references) and (2) the destructors perform no actions, so there are no side effects that normally you would want to ensure.

We also added a class `HPlane` that represents a plane as a 4-tuple, and a class `HQuaternion` that represents a quaternion. We have added a class `HMatrix` that represents a 4×4 homogeneous matrix.

All these classes implement affine algebra, with the important distinction maintained between *point* and *vector*. Moreover, the classes are written only for 32-bit floating-point numbers. Our goal is to provide implementations for the vector and matrix operations that use Intel's SSE and the PowerPC's Altivec instructions for fast vector-matrix algebra. It is necessary to have 4-tuples for the SIMD registers and for the necessary macros that guarantee proper alignment in memory.

The `Vector3` and `Matrix3` classes do not satisfy the 4-tuple requirement for SIMD support, but we were unwilling to immediately deprecate the classes. Moreover, they are templated and support double-precision floats, something not normally handled in SIMD.

4.5 Change to Matrix Operations in Shaders

Let M be a 4×4 matrix and let $\mathbf{V}$ be a 4×1 vector. The vector-on-the-right convention for multiplication of the vector by the matrix is $M\mathbf{V}$. The vector-on-the-left convention is $\mathbf{V}^T M$, where the superscript T denotes the transpose operator.

WM4 used the vector-on-the-right convention for its matrix and vector algebra. However, the shaders were written to use the vector-on-the-left convention. For example, a Cg shader had code such as

```
void v_SomeShader
(
    in float3 modelPosition : POSITION,
    out float4 clipPosition : POSITION,
    uniform float4x4 WVPMatrix
)
{
    clipPosition = mul(float4(modelPosition, 1.0f), WVPMatrix);
}
```

Note that the name `WVPMatrix` is suggestive of the world matrix W applied first, the view matrix V applied second, and the projection matrix P applied third.

The WM4 graphics system arranged for `WVPMatrix` to be in the proper form for the matrix multiplication. This meant computing transposes of matrices to be sent to the shaders as constants, which is inefficient. Moreover, the default shader compilation stores the matrices in row-major order, so the vector-on-the-left convention causes some slight performance problems in the shader. Note that shader compilers sometimes can be given command-line parameters to specify how the matrix should be stored, in which case the shader performance problem can be avoided.

In WM5, we have switched to using vector-on-the-right convention for the shaders. This avoids the transpose operations and shader performance when matrices are stored in row-major order. The vertex shader is then

```
void v_SomeShader
(
    in float3 modelPosition : POSITION,
    out float4 clipPosition : POSITION,
    uniform float4x4 PVWMatrix
)
{
    clipPosition = mul(PVWMatrix, float4(modelPosition, 1.0f));
}
```

Note that the naming convention for the shader constant has changed accordingly, still suggestive that W is applied first, V is applied second, and P is applied third.

4.6 Raw Data Formats for Buffers, Textures, and Effects

We added support for raw data formats for vertex and index buffers (WMVF), textures (WMTF) and compiled shaders (WMFX). This allows users to create graphics resources with whatever tools they want to use; that is, you are not forced to have a fully feature Wild Magic exporter for a modeling package (to export Wild Magic scene objects).

4.6.1 Visual Objects

WM5 has a class named `Visual` that replaces the `Geometry` class of WM4. A `Visual` object stores a geometry type (points, segments, triangle meshes, triangle fans, triangles strips), a vertex buffer, a vertex format that describes the vertex buffer, and an (optional) index buffer. A raw format that stores all this information is listed next.

```
WMVF format
    int type; <Visual::PrimitiveType>
    VertexFormat vformat;
    VertexBuffer vbuffer;
    IndexBuffer ibuffer;

VertexFormat
    int numAttributes;
    Attribute attribute[numAttributes];
    int stride;

Attribute
    unsigned int streamIndex;
    unsigned int offset;
    int type; <VertexFormat::AttributeType>
    int usage; <VertexFormat::AttributeUsage>
    unsigned int usageIndex;

VertexBuffer
    int numElements;
    int elementSize;
    int usage; <Buffer::Usage>
    char data[numElements*elementSize];

IndexBuffer
    int numElements;
    int elementSize;
    int usage; <Buffer::Usage>
    int offset;
    char data[numElements*elementSize];
```

You can load such files with `Visual::LoadWMVF`.

4.6.2 Texture Objects

WM5 has a base class named `Texture` with various derived classes. The base class has all the data for the textures. A raw format for the data is listed next.

```
WMTF format
    int format; <Texture::Format>
    int type; <Texture::Type>
    int usage; <Buffer::Usage>
    int numLevels;
    int numDimensions;
    int dimension[3][MAX_MIPMAP_LEVELS]; <MAX_MIPMAP_LEVELS = 16>
    int numLevelBytes[MAX_MIPMAP_LEVELS];
    int numTotalBytes;
    int levelOffsets[MAX_MIPMAP_LEVELS];
    unsigned int userField[MAX_FIELDS]; <MAX_FIELDS = 8>
    char data[numTotalBytes];
```

4.6.3 Effect Objects

WM5 has a class named `VisualEffect` that represents shaders from an FX system. The class encapsulates techniques, passes, vertex and pixel shaders, global render state for the passes (and for combining passes), shader constants, and information about inputs and outputs for the shaders. A raw format for this class is listed next.

```
WMFX format
    numTechniques <int>
    Technique[numTechniques]

Technique
    numPasses <int>
    Pass[numPasses]

Pass
    VertexShader
    PixelShader
    AlphaState
    CullState
    DepthState
    OffsetState
    StencilState
    WireState

VertexShader
    Shader

PixelShader
```

```
    Shader

AlphaState
    blendEnabled <int: bool>
    srcBlend <int: SrcBlendMode>
    dstBlend <int: DstBlendMode>
    compareEnabled <int: bool>
    compare <int: CompareMode>
    reference <float>
    constantColor[4] <float>

CullState
    enabled <int: bool>
    ccwOrder <int: bool>

DepthState
    enabled <int: bool>
    writable <int: bool>
    compare <int: CompareMode>

OffsetState
    fillEnabled <int: bool>
    lineEnabled <int: bool>
    pointEnabled <int: bool>
    scale <float>
    bias <float>

StencilState
    enabled <int: bool>
    compare <int: CompareMode>
    reference <unsigned int>
    mask <unsigned int>
    writeMask <unsigned int>
    onFail <int: OperationType>
    onZFail <int: OperationType>
    onZPass <int: OperationType>

WireState
    enabled <int: bool>

Shader
    programName <string>
    numInputs <int>
    numOutputs <int>
    numConstants <int>
    numSamplers <int>
    numProfiles <int>
    Input[numInputs]
```

```
    Output[numOutputs]
    Constant[numConstants]
    Sampler[numSamplers]
    Profile[numProfiles]

Input
    name <string>
    type <int: VariableType>
    semantic <int: VariableSemantic>

Output
    name <string>
    type <int: VariableType>
    semantic <int: VariableSemantic>

Constant
    name <string>
    numRegistersUsed <int>

Sampler
    name <string>
    type <int: SamplerType>
    filter <int: SamplerFilter>
    coordinate[3] <int: SamplerCoordinate>
    lodBias <float>
    anisotropy <float>
    borderColor[4] <float>

Profile
    type <int: VertexShader::Profile or PixelShader::Profile>
    program <string>
    baseRegisters[numConstants] <int>
    textureUnits[numSamplers] <int>
```

You can load such files with `VisualEffect::LoadWMFX`. An example of creating such files is in the `WmfxCompiler` tool.

5 Tools

For now we have available only a limited set of tools. We will implement more over time. These run on Microsoft Windows, but they may be easily ported to run on Linux and Macintosh.

5.1 BmpColorToGray

Convert a 24-bit color Microsoft Windows BMP file to a 24-bit gray-scale BMP file. I use this for generating gray-scale images for books.

5.2 BmpToWmtf

Convert a 24-bit color Microsoft Windows BMP file to the Wild Magic 5 WMTF texture format. The WMTF files can be used on any platform.

5.3 GenerateProjects

The `GenerateProjects` tool allows you to automatically create Microsoft Visual Studio 2010 or 2012 project files and Xcode 4.4 project files. From a command line,

```
GenerateProjects MyProject
```

will create

```
MyProjectDx9_VC100.vcxproj
MyProjectDx9_VC100.vcxproj.filters
MyProjectWgl_VC100.vcxproj
MyProjectWgl_VC100.vcxproj.filters
MyProjectDx9_VC110.vcxproj
MyProjectDx9_VC110.vcxproj.filters
MyProjectWgl_VC110.vcxproj
MyProjectWgl_VC110.vcxproj.filters
```

and similarly named ones with VC110 in the file names. The project file uses paths relative to the `WildMagic5` folder at the sample-application level. For example, copy the generate file to

```
GeometricTools/WildMagic5/MyApplications/MyProject
```

The project has references to two files, `MyProject.h` and `MyProject.cpp`, which you will have to create. However, you can always delete the references and use your own file names.

The executable will also create a subfolder named `MyProjectName.xcodeproj` that contains a file named `project.pbxproj`. This subfolder and file are the pair that Xcode needs on the Macintosh. The project file uses paths relative to the `WildMagic5` folder, just as described in the previous paragraph for Microsoft Visual Studio.

5.4 ObjMtlImporter

Source code for importing of OBJ and MTL files (Alias Wavefront formats). Only basic features are supported.

5.5 WmfxCompiler

This program requires that NVIDIA's Cg be installed. The program creates a command shell that runs cgc on a specified FX file. The compiler attempts several profiles, for DirectX 9 and for OpenGL. Any successfully compiled profiles are then packaged up into a WMFX file that may be loaded by Wild Magic 5 and used as a `VisualEffect` object. In effect, WMFX files are the binary resources for the WM5 special effect system.

5.6 WmtfViewer

This is a viewer program for `Texture2D` objects that are stored in the raw format WMTF files. It needs to be modified to handle other texture types, but for now it is useful to be able to associate the program with WMTF files, double-click, and view the texture. You can left-click-mouse on the texels to see the RGBA values. You can also toggle between display of RGB and A.