Tilengine

A 2D graphics engine with raster effects

http://www.tilengine.org

Last updated in 2/9/2015 9:17:00

By Marc Palacios (Megamarc)

Table of contents

Package contents

This document describes the contents included inside the file **Tilengine.zip**. Please visit http://www.tilengine.org for the latest news.

Tilengine.pdf

This file.

install.bat

Installation batch for Windows (see Installing in Windows)

install.sh

Installation bash script for Linux (see Installing in Linux)

/samples

- Graphics assets used by the samples (tmx, tsx, png, sqx, txt…)
- This is the main target path for all executables after install/build

/lib

Precompiled tilengine binaries and header file for C samples

/src

- Source code of the C samples
- Makefile for building the samples

/dev-cpp

Orwell Dev-CPP projects to build the C samples

/visualc

MS Visual Studio 2005 projects of the C samples

/Python

Python wrapper and sample scripts (see Python)

/Java

Java JNI wrapper and sample programs (see Java)

Installing

Extract the files in any directory of your convenience (preserving the structure of directories) and open a command window in the root folder of the pack.

Installing in Windows

The supplied **install.bat** file copies all the precompiled libraries, examples, and python wrapper and samples and precompiled java wrapper and samples to the **/samples** directory. Just type

```
> install
```

Installing in Linux

The supplied **install.sh** bash script copies all the precompiled libraries, examples, and python wrapper and samples and precompiled java wrapper and samples to the **/samples** directory. It also registers the **libTilengine.so** library in a system-wide location (**/usr/local/lib**), so administrator privileges are required to install. Just type

```
> sudo ./install.sh
```

Running the samples

Binaries for the C samples are already precompiled for Windows and Linux. To run them, open a console in **/samples** directory and run their executables or double-click their icons from the file explorer.

List of samples

There are eight samples:

Name	Description
barrel	Sidescroller with barrel-distortion effect similar to SNES Super Castlevania IV and basic character control (jumping, tile collision)
mode7	Classic 2D perspective projection to simulate a 3D floor, similar to SNES Super Mario Kart
platformer	Sidescroller with palette animation and linescroll to simulate depth using only two layers. Similar to Sega Genesis Sonic
racer	Fake 3D road using linescroll, floor palette switching and scaling sprites. Similar to Sega Super Hang On
scaling	Combination of layer scaling and alpha blending
shooter	Complex example with intense use of raster effects for sky gradient, linescroll and layer multiplexing. Basic shooter engine similar to Sega Genesis Thunder Force IV
tutorial	Basic sample on how to setup the engine, load a tileset/tilemap and make a scrolling layer
wobble	Water-like distortion effect combining linescroll and column offset mode to create an ondulating effect
ColorCycle	Color cycling animation
SeizeTheDay	Color cycling animation with palette interpolation

This directory also contains the Python (.py) and Java (.class) versions of some samples

User input

All samples use the same input:

- Use keyboard cursor arrows or joystick d-pad to move
- Press ‘z’ key or joystick button 1 to fire in the shooter example or jump in the barrel example

Building the samples

The samples can be built with 4 different tools: Microsoft Visual Studio 2005 and later, Orwell Dev-CPP, MinGW and GNU GCC for Linux

Windows – MS Visual Studio

Go to the **/visualc** folder and open the **TilengineSample.sln** solution file, convert to the newer version of Visual Studio if asked for, and build. The executables are directly placed in the /**samples** folder.

Windows – Dev-CPP

Go to the **/dev-cpp** folder and open any of the eight **.dev** files (one for each sample). Set up the compiler if desired (defaults to the first toolchain installed) and build. The executables are directly placed in the /**samples** folder.

Note: newer version of Dev-CPP use the GNU command **rm** for the clean target, which is not available in Windows, instead of the standard **del /Q**. If there is a build error when doing clean, install **rm** for windows or tweak the intermediate makefile by hand.

Windows – MinGW

Open a command prompt inside the **/src** folder and type the following command:

```
> mingw32-make
```

The executables are directly placed in the /**samples** folder.

Linux – GNU GCC

Open a command prompt inside the **/src** folder and type the following command:

```
> make
```

The executables are directly placed in the /**samples** folder.

Python

Tilengine can be accessed from Python through the provided **Tilengine.py** module wrapper. It uses the ctypes feature in Python to interface with native libraries. It is just a barebones wrapper, with all the functions in a single module.

Basic usage

To use the wrapper, you have to import the corresponding module, for example:

```
import tilengine as tln
```

Now all the API is available through the tln namespace, and removing the original "TLN_" prefix. For example, to initialise the engine:

```
# create a 400x240 pixels renderer with 2 layers and 80 sprites
tln.Init (400,240, 2,80,0)

# create a window with a scanline overlay and v-sync
tln.CreateWindow ("overlay2.bmp", tln.CWF_VSYNC)

#load some resources and setup layer
tileset = tln.LoadTileset ("mytileset.tsx");
tilemap = tln.LoadTilemap ("mytilemap.tmx", "Layer 1");
tln.SetLayer (0, tileset, tilemap);
```

Interactive mode and multithreading

The default windowing system in tilengine is single-threaded: the game main loop is responsible of attending the event queue of the window and updating the screen, calling the functions `ProcessWindow()` and `DrawFrame()`:

```
# main loop
while tln.ProcessWindow():

	# do game stuff (read inputs, update logic...)

	tln.DrawFrame (frame)
	frame += 1

# deinitialise
tln.DeleteWindow ()
tln.Deinit ()
```

However, during an interactive session in the interpreter, the window gets unattended (nobody is calling `ProcessWindow()` and `DrawFrame()` periodically) so the window doesn't refresh and suffers a "non responding" state.

To solve this problem, a secondary windowing mode has been implemented. Instead of the previous `CreateWindow` function, you have to use:

```
# create a multithreaded window with a scanline overlay
tln.CreateWindowThread ("overlay2.bmp", 0)
```

This function creates the window in its own thread, which has a loop that attends events and updates it at 60 Hz (always v-sync) in the background. In this mode you don't have to call `ProcessWindow(), DrawFrame()` and `DeleteWindow()`.

Sample scripts

Some of the original samples written in C have been ported to python. These are **platformer.py**, **mode7.py** and **scaling.py**. To run them just open a python console in the samples directory and type any of these:

```
import platformer
import mode7
import scaling
```

To finish the script just press Esc or close the window

Interactive sample scripts

Each one of the three sample scripts listed above have a tweaked version intended to be played with interactively. These are **platformer_test.py**, **mode7_test.py** and **scaling_test.py,** To run them just open a python console in the samples directory and type any of these:

```
import platformer_test as t
import mode7_test as t
import scaling_test as t
```

The `as t` sentence is just for shortening the namespace. Each sample has some prebuilt functions for easy interaction, and from any of them you can access directly tilengine.

Java

Tilengine can be accessed from Java through the provided JNI wrapper: the java class **Tilengine.java** which in turn accesses the engine through two native bridges: **TilengineJNI.dll** (windows) and **libTilengineJNI.so** (linux).

This wrapper is just a barebones bridge to the native Tilengine API, which doesn't use classes or other Java-like features. All the functions are contained in a single class, and all the object references are plain integers.

Files

File	Description
Tilengine.java	Java side source of the JNI wrapper
TilengineJNI.c	Native side source of the JNI wrapper
Makefile_*	Makefiles for building the JNI bridge
TestWindow.java	Java sample that uses Tilengine internal windowing
TestPanel.java	Java sample that uses an AWT JPanel as a rendering target
/class	Precompiled java classes
/linux	Precompiled linux JNI bridge
/win32	Precompiled windows JNI bridge

Basic usage

To use the wrapper, first create an instance of the Tilengine class:

```
Tilengine tln = new Tilengine();
```

Then you can access all the functions in the tln variable. For example for setting up a basic layer yo can do:

```
// init a 400x240 display, 1 layer and 0 sprites
tln.Init (400,240, 1,0,0);
tln.CreateWindow ("overlay.bmp", tln.CWF_VSYNC);

// load some resources and setup renderer
int tileset = tln.LoadTileset ("mytileset.tsx");
int tilemap = tln.LoadTilemap ("mytilemap.tmx", "Layer 1");
tln.SetLayer (0, tileset, tilemap);
```

Included samples

There are two sample programs that show how to use the wrapper. They're not an example of Java best practices, but just a simple way to setup a basic game loop and rendering with Tilengine. **TestWindow** uses the internal windowing (SDL) as the C or Python samples. **TestPanel** shows how to use Tilengine as a backend renderer, drawing inside an AWT Jpanel

To run the samples in windows, type:

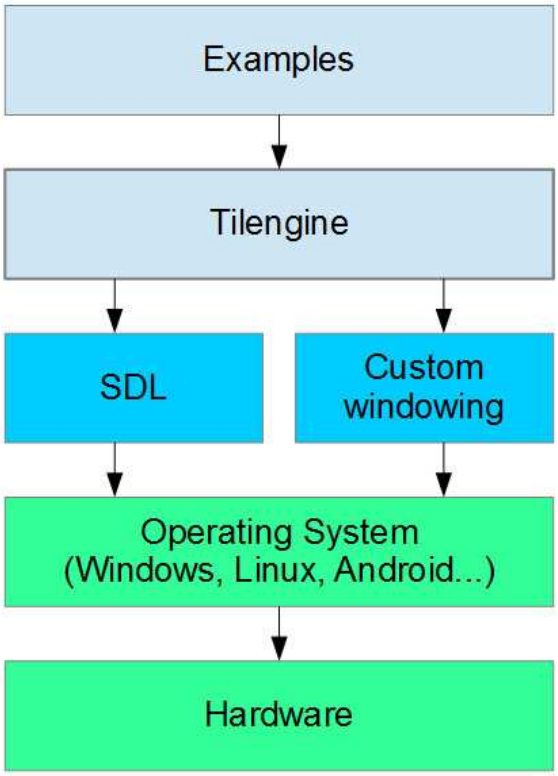
```
> java TestWindow
```

To run from Linux:

```
> java -Djava.library.path=. TestWindow
```

Samples layout

The following chart shows how the examples, Tilengine, and other components are laid out for easy porting and integration:



- **Examples** (or games) are at the top-most layer. They use Tilengine for rendering and its built-in windowing environment. They are open source.
- **Tilengine** provides the 2D scanline-based rendering and optional windowing environment. It is freeware closed source.
- In order to build on as many platforms as possible, built-in Tilengine windowing system uses **SDL** (Simple DirectMedia), an open-source cross-platform library that provides windowing and rendering on many platforms. As long as there is a build of SDL on a given platform, Tilengine and its examples can run there.
- Below SDL there are the underlying **operating system** and **hardware**.

Acknowledgements

Tilengine wouldn't be possible without these work previously created by other generous people. Thanks to all of you!

Tiled

Tiled is a fantastic open-source tilemap map editor by Thorbjørn Lindeijer. Tilengine reads the maps created with Tiled. http://www.mapeditor.org

Spritesheet packer

This tool created by Nick Gravelyn takes a list of independent graphics and outputs a single image with all the original images inside it with their coordinates. Tilengine reads spritesheets for the sprites. https://spritesheetpacker.codeplex.com/

Libpng

Libpng the reference library for reading png image files, open source by many authors. http://www.libpng.org/pub/png/libpng.html

SimpleXML

SimpleXML is a tiny XML parser for embedded systems. Tilengine uses it to read the Tiled maps, which are in XML format. http://simplexml.sourceforge.net/

LibSDL

LibSDL is a cross-platform windowing and user input library, among other things. The built-in windowing environment in Tilengine uses SDL. https://www.libsdl.org/

Mark Ferrari

Mark Ferrari is a pixel art illustrator (among other things), who drew and animated all the awesome images in ColorCycle and SeizeTheDay examples and gave his permission to use them here. http://markferrari.com/