Home >> PSP Tutorials >> Tutorial 1: The game loop

This is the second PSP tutorial

Introduction

Hi everybody,

This tutorial is about using a game loop. For this and all tutorials I assume that you have some C++ experience and that the basics are clear to you. This tutorial will show some debug information on screen to see what happens. This tutorial continues on the previous tutorial and expects that you have done that tutorial.

The implementation

We will need a game application object called GameApp. This is basicly the main class in which everything happens. Well it is better understood if one sees an example so lets get going an see what code we use:

GameApp.h

```
#ifndef GAMEAPP_H_
#define GAMEAPP_H_

#include "GraphicsObject.h"

class GameApp {

protected:
   int frameCount;
   GraphicsObject* gfx;

   int Controls();
   void Render();
   bool Load();

public:
   GameApp();
   ~GameApp();
   int Run();

};

#endif
```

Here is the class for the game object. Well the function Controls, Render and Load are self explanatory. The run function as we will see in the source first calls the Load function, then in a loop calls the controls and render function. There are two variables, gfx and frameCount. Gfx will be used the same as in tutorial 0 only not in the main function anymore but in the game class. The frameCount is just a variable used to show what happens when the code is run. I will explain them after I have presented the source.

GameApp.cpp

```
#include "GameApp.h"
#include "GraphicsObject.h"
#include <pspctrl.h>
#include <pspdebug.h>
#include <pspdisplay.h>
#include <pspgu.h>

GameApp::GameApp() {
}

GameApp::~GameApp() {
}
```

Well not much to say here, the includes are for controls, debugging, display and the offcourse the GU.

```
int GameApp::Run() {

   if(!Load())
      return 1;

   // game loop
   while(1) {
      if(Controls() == 9)
         break;
      Render();
   }
   return 0;
}
```

This is the game loop. Before we go into the game loop we have to load up all kinds of data, like objects, images etc. If it fails we can't go any further so we return a 1 which we can use. In this tutorial I do not use it but it is smart to use one kind of return/error throwing.

And finally the loop itself. I use a While(1) but you can use all sorts of ways to get this effect. The main idea offcourse is that you want to iterate some function like the Render() function and the Controls() function which I do :D. In the code, if controls function return 9 the loop ends and the application ends.

```
bool GameApp::Load() {
   // get pointer to Graphics object.
   gfx = GraphicsObject::Instance();
   // set up the environment.
   gfx->Init3DGraphics();

   // set frameCount
   frameCount = 0;

   return true;
}
```

Well here is the Load() function. As you can see here you can load up objects like the graphics object and you can set variables. We load up the graphics object here because the game uses it. The main function does not need an pointer to it so we have deleted it from the main.

```
int GameApp::Controls() {
   // reading controls
   SceCtrlData pad;
   sceCtrlReadBufferPositive(&pad;, 1);

   // check for input
   if (pad.Buttons != 0){
      if (pad.Buttons & PSP_CTRL_CROSS){
         return 9;
      }
   }
   return 0;
}
```

This is the Controls() function. In a game we need input, in this example we listen for the Cross button. When the Cross button is pressed the return 9 code is run. First we read the buffer into the pad variable. **Please note** that this read function slows down the process so using the seek variant is better in 3D games. We check if something is pressed and then we check if the button we want is pressed. In this case the Cross button. For more information about controls and how to use them exactly see the m5 tutorial on this site.

```
void GameApp::Render() {
   // render information.
   frameCount++;
   pspDebugScreenPrintf("Rendering frame %d\n", frameCount);
   sceDisplayWaitVblankStart();
   sceGuSwapBuffers();
}
```

This is the render sub. In this example we do not actually render something, we just show something with the debug printer. We first add 1 to the frameCount variable, then we use that to show which frame is rendered. It is not very great or something but when you compile the whole tutorial at the end you can see what happens. sceDisplayWaitVblankStart waits for a vertical blank before going further. With a display refreshrate of 60 this means that the render function (if the whole gameloop takes less then 1/60th of a second) is only executed 60 times a second. The swap buffers function simply does what it says, it swaps the buffer on which everything is rendered to the buffer that is shown and vice versa so that that buffer can be drawn on again.

Main

As mentioned I have made some changes to the main function. I have added the include:

main.cpp

```
#include "GameApp.h"
```

Just the include for the gameapp class. I have also made some changes to the main function itself:

```
/*!
* \fn int Main(void)
* \brief The main function
*
* @return 0
*/
int main (void) {

   // init the debug screen functions
   pspDebugScreenInit();

   // setup the callbacks
   SetupCallbacks();
   // creating the game object
   GameApp* game = new GameApp();
   // run the game app and get return value
   int ret = game->Run();

   // shutdown
   sceGuTerm();
   sceKernelExitGame();

   return 0;
}
```

I have deleted the graphics object and replaced the code for the gameapp. Please note the pspDebugScreenInit() function needs to be called to use the debugprint function as we do in the game app.

The Graphics object has not been changed this tutorial.

MakeFile

The makefile for tutorial 0

```
TARGET = out
OBJS = $(wildcard *.cpp) $(wildcard *.c)

INCDIR =
CFLAGS = -O2 -G0 -Wall -g
CXXFLAGS = $(CFLAGS) -fno-exceptions -fno-rtti -g
ASFLAGS = $(CFLAGS)

LIBDIR =
LDFLAGS =
LIBS = -lstdc++ -lc -lpspgu -lpsprtc

PSPSDK=$(shell psp-config --pspsdk-path)
include $(PSPSDK)/lib/build.mak
```

Compile and what happens ?! The framecount is shown at the screen and when you press the X button the app exits to the XMB. We have just made our first very basic game loop. In the next to come tutorials the game loop will be extended though.

Source files

Download the source files.