This is the animated loading screen tutorial.

# Introduction

Hi everybody,

This tutorial is about creating an animated loading screen. For this and all tutorials I assume that you have some C++ experience and that the basics are clear to you. This tutorial shows some loading graphics while loading/doing other stuff using threads. It uses the tutorial 2D graphics on 3D as a basis.

# Loading screen

If you look at a lot of commercial games you can see that when the games load up everything that there is still graphics and animations keeping you company until everything is loaded and you can start the game. If you have a load function, also in the tutorials until now, you probebly load everything and while you load everything the game shows a black screen or a static image. Some try to update the screen between every bit that is loaded but this often leaves the user with glitches and when loading a very large object still with a static image.

In this tutorial we are going to make a loading screen which shows animations without glitches/slowdowns etc. during loading. The secret? well we use multithreading! With it we can create a second thread which shows an animated loading screen while the main thread loads up every object/file/var etc. I will only show a very basic animation but with this tutorial you should be able to show the most wonderfull animations during loading as far as your imagination can bring you!


This is the output of this tutorial. It shows a glowing (animated) loading text on the screen while loading data. The animation does not halt or shows any delays.

Lets take a look at the loadingscreen class:

**LoadingScreen.h**

```
#ifndef LOADINGSCREEN_H_
#define LOADINGSCREEN_H_

#include <pspkerneltypes.h>

class LoadingScreen {

protected:
   static int RunLoadingScreen(SceSize args, void *argp);
   SceUID thid_;

public:
   LoadingScreen();
   void KillLoadingScreen();

};

#endif
```

We start as usual with the header guards. We include the pspkerneltypes, because we need it for our threads. Please note that the functions used in this tutorial are no kernel functions. The name can be somewhat misleading but they are not kernel functions. So what functions do we need in our class. The static RunLoadingScreen is used in our thread. It is the function that is run in the thread. The reason it has to be static is because otherwise it can not be passed to the creation of the thread. You will see this in the .cpp file of this class. The thid_ variable is used to describe the thread. We also need our constructor and a function to kill the thread since we only need it when game files etc. are loaded.

**LoadingScreen.cpp**

```
#include "LoadingScreen.h"
#include <pspthreadman.h>
#include <pspgu.h>
#include <pspgum.h>
#include <pspdisplay.h>
#include "GraphicsObject.h"
extern "C" {
   #include "graphics.h"
}

static unsigned int __attribute__((aligned(16))) list[262144];
Image* load1;
Image* load2;
```

We load up a lot of header files, the pspthreadman is used for the threads, the others are known. You are probebly asking yourself now: "Why is he declaring variables in the cpp outside the functions and why not in the header?" Well since the function we use is static it can not access the vars from the instance of the class from which the thread is started. This way the variables can be reached from both parts. The two image files are used to show an very basic and simple animated loadingscreen.

```
LoadingScreen::LoadingScreen() {
   // The loadingscreen is loaded as a thread
   thid_ = sceKernelCreateThread("LoadingThread", RunLoadingScreen, 0x18, 0x10000, 0, NULL);
   // Start the thread
   sceKernelStartThread(thid_, 0, 0);
}
```

This is the real interesting stuff. The first line creates a thread called "LoadingThread" which executes the RunLoadingScreen function. More on this later. The third parameters specifies the priority of the thread. The lower this value the higher the priority. The fourth specifies the stacksize. The fifth parameter are thread attributes. We use 0 so it has the default attributes (like now it is in usermode since we call it

from a thread that is in usermode.) In this tutorial it is not needed to know about these parameter nor is it needed to know what exactly the last parameter does. This will be explained in tutorial m10: Threads revisited. For now it suffice to know that we have created a default thread which runs the RunLoadingScreen function.

The second line starts the thread and as soon as this function is called the RunLoadingScreen function is started in the just created thread. The last two parameters are not important at this point. Just leave them both 0. The first parameter is the thread ID which we have got returned from the thread creation function. Now the thread is not only create but also started. So as soon as we create a LoadingScreen instance using the constructor it also immediatly starts.

```
void LoadingScreen::KillLoadingScreen() {
   // shut down the loading screen again.
   sceKernelTerminateDeleteThread(thid_);
   // free the mem space of the images
   freeImage(load1);
   freeImage(load2);
}
```

This function is needed when all the loading is done. It terminates and deletes the thread. So our function, which will be shown next, will be stopped and the thread will be deleted freeing up the memory so we can use it again. Since we also load up some image in the function RunLoadingScreen as you will see next, we free up the memory space of those image also.

```
int LoadingScreen::RunLoadingScreen(SceSize args, void *argp) {

   // first create the graphicspart
   GraphicsObject* gfx = GraphicsObject::Instance();

   // load up the images
   load1 = loadImage("load1.png");
   load2 = loadImage("load2.png");

   // set animation to 0
   int animation_ = 0;
```

This is the actual function run by the loadingscreen thread. We first get a pointer to our graphicsobject and we load up the images. We also set the animation_ to 0. The two parameters of this function are the standard parameters for thread functions. These need to be this way otherwise you can not call the function when creating the thread.

```
   // start the render loop
   while(1) {

      // render information.
      sceGuStart(GU_DIRECT,list);

      // clear screen
      sceGuClearColor(0xff000000);
      sceGuClearDepth(0);
      sceGuClear(GU_COLOR_BUFFER_BIT|GU_DEPTH_BUFFER_BIT);

      // set the ambient light
      sceGuAmbient(0xffffffff);

      // setting the view
      sceGumMatrixMode(GU_VIEW);
      sceGumLoadIdentity();

      // the loading graphics :)
      animation_++;
```

```
        if(animation_ > 19)
            animation_ = 0;

        // switch images now and then
        if (animation_ < 10)
            gfx->Render2DImageOn3D(117, 100, load1);
        else if (animation_ < 20)
            gfx->Render2DImageOn3D(117, 100, load2);

        // ending rendering
        sceGuFinish();
        sceGuSync(0,0);
        sceDisplayWaitVblankStart();
        sceGuSwapBuffers();
    }
    return 0;
}
```

The rest of the function is nothing more then just a simple render function. All the lines are already known and if not please read some previous tutorials. The render function renders the two pictures each 10 frames. This gives a switching/glowing effect to the text. It really is not very nice but it suffice to let you see what is possible with this.

So now that we have created our Loadingscreen class we want to use it.

## GameApp

The GameApp has a load function and instead of only loading we are now going to show a loadingscreen.

**GameApp.h**

```
    #include "LoadingScreen.h"

    int frameCount;
    LoadingScreen* loadscr;
```

We have to include the LoadingScreen.h to be able to use it. The other two variables are protected variables. The frameCount is used to show the frames and the loadscr is the pointer to the LoadingScreen object.

**GameApp.cpp**

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

    // create the loading thread
    loadscr = new LoadingScreen();

    // set frameCount
    frameCount = 0;

    for(int i=0;i<300;i++){
        //wait everytime for the screen blank, total 5 seconds
```

```
        sceDisplayWaitVblankStart();
    }

    // we stop the loading screen
    loadscr->KillLoadingScreen();
    // we remove it from memory
    delete(loadscr);

    return true;
}
```

This is the Load function. I first initialize the Graphics since we are using them in the loadingscreen class. Then we create the instance of the LoadingScreen class which automatically starts the thread as seen in the previous chapter. So now the other thread is running and displaying the two images. In the mean while this main thread continues with setting the frameCount to 0 and looping through 300 times the sceDisplayWaitVblankStart which results in ± 5 seconds of doing the waiting for the Vblank. Instead of this you should for your game load all the files/textures/images and objects that you will use in the game. After it we call the KillLoadingScreen function which terminates the thread and frees up the memory occupied by that thread and its content. Then we delete the loadscr object to clear up all the memory.

```
void GameApp::Render() {

    // render information.
    sceGuStart(GU_DIRECT,list);

    // clear screen
    sceGuClearColor(0xff000000);
    sceGuClearDepth(0);
    sceGuClear(GU_COLOR_BUFFER_BIT|GU_DEPTH_BUFFER_BIT);

    frameCount++;
    pspDebugScreenSetXY(1,1);
    pspDebugScreenPrintf("Rendering frame %d\n", frameCount);

    // ending rendering
    sceGuFinish();
    sceGuSync(0,0);
    sceDisplayWaitVblankStart();
    sceGuSwapBuffers();
}
```

This is the render sub. It shows the frames passed on the left top of the screen. All these functions are known and if not please read previous tutorials on this site.

That is the end of this tutorial. Read the m10 tutorial on threads if you want to know more about them and use the source below and the makefile to compile the application and see what you have learned.

## MakeFile

The makefile for tutorial 11

```
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 -lpspgum -lpng -lz -lpsprtc -lm

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

Compile and what happens ?! You should first see a glowing (kind of, but bad) loading text appearing on screen and after about five seconds it will clear and the framecount will start. We have just created a animated loadingscreen.

## Source files

Download the source files.