This is PSP tutorial 6: Lines and Curves

Introduction

Hi everybody,

This tutorial is about rendering 2D graphics and 3D graphics in a 3D environment. For this and all tutorials I assume that you have some C++ experience and that the basics are clear to you. I also expect that you have used the previous tutorials. This tutorial will show some graphics on screen. It will show lines and curves on top of a 3D environment. The pieces of code shown in this tutorial may not be the total sourcecode needed to get the app to run. Use the source files at the bottom to compile and use.

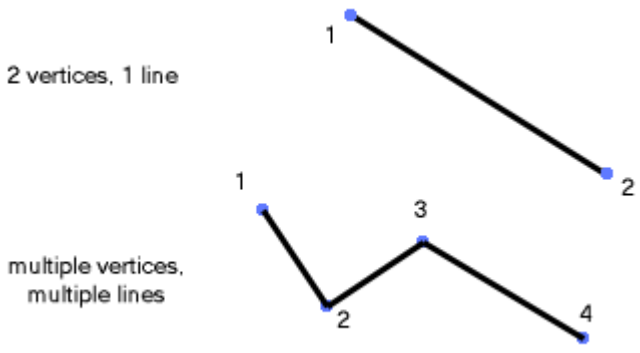
2D lines

In the first part we will discuss drawing 2D lines on screen. We can use this for example in menu's, GUI's or in HUDs. We will use our graphics object again. If you have done all tutorials at the end the graphics object will be full of handy functions. Lets take a look at the function to be added:

GraphicsObject.h

```
intRender2DLineOn3D(const float x1, const float y1, const float x2, const float y2, const
   unsigned int color);
```

We will give the point of origin(1) and the destination point(2). The line will be drawn between these points. These points need to be in screen coordinates. so a line from (0,0) to (408,272) will be drawn from corner to corner. The reason why I use floats is that I have more precision when I use it for some nice effects.(like translating the origin of the line very slowly.) I also pass the color. In this example I will only draw a blue line on screen.



On the right you can see an image of some points and some lines. In this part of the tutorial we will only use 2 vertices(points) to render a line but as shown in the image, you can use more vertices and every new vertex will render a line from the last vertex to the added vertex.

```
typedef struct { unsigned int color;
                 float x, y, z; } lineVertex;
```

We need this struct for the vertices of the line. As we have seen the line has 2 points(vertices) and we need to send those vertices to the renderer in the right format. We will only use its position and its color in this example.

Lets take a look at the source:

GraphicsObject.cpp

```
int GraphicsObject::Render2DLineOn3D(const float x1, const float y1, const float x2, const
float y2, const unsigned int color) {
```

This is just the function as described in the header.

```
    sceGuDisable(GU_TEXTURE_2D);
    sceGuDisable(GU_DEPTH_TEST);
```

First we turn of the texture flag so that it is rendered with its color. If the flag is still enabled nothing will be rendered if there is not given a texture. The second line turns of the depth test. Since it will be printed on screen we do not need to test it with depth. This is only needed with 3D object/renderings. Please keep in mind that if you have more lines to be rendered that you need to keep the order in check yourself.

```
    lineVertex* Line = (lineVertex*)sceGuGetMemory(2 * sizeof(lineVertex));
```

Here we create a pointer to the lineVertex structure and then save 2 times the size of the lineVertex in memory to which the pointer points.

```
    Line[0].color = color;
    Line[0].x = x1;
    Line[0].y = y1;
    Line[0].z = 0.0f;

    Line[1].color = color;
    Line[1].x = x2;
    Line[1].y = y2;
    Line[1].z = 0.0f;
```

The two vertices we need to render the line are now filled with the given data from the functionline. x1 and y1 belong to the first point and x2 and y2 to the second point.

```
    sceGuDrawArray(GU_LINES, GU_COLOR_8888 | GU_VERTEX_32BITF | GU_TRANSFORM_2D, 2, 0, Line);
```

Here we actually draw the line. We use the GU_LINES to specify that we now are drawing a line. Then we sepcify that we will use full color and position vertices and that will want it to be drawn 2D. The 2 specifies the number of vertices and the Line variable is the pointer to our vertices.

```
    sceGuEnable(GU_DEPTH_TEST);
    sceGuEnable(GU_TEXTURE_2D);
```

Here we enable the Texture and the depth test again.

```
    return 0;
}
```

Now we have a function which can draw a 2D line on a 3D background.

Source files part 1

Download the source files.

3D lines

In this part we will render a 3D line instead of a 2D line. What do we need 3D lines for? Well I use them to render waypoints when I want to test my games for example. Although the code is quite similar, there are some differences. Also with the 3D lines you can add multiple vertices to create chains of lines just like with the 2D lines. Lets take a look at the header (we continue from the previous part):

GraphicsObject.h

```
int Render3DLine(const float x1, const float y1, const float z1, const float x2, const
float y2, const float z2, const unsigned int color);
```

This is almost the same as the 2D line. I have only added the third dimension to the vertices.

GraphicsObject.cpp

```
int GraphicsObject::Render3DLine(const float x1, const float y1, const float z1, const
float x2, const float y2, const float z2, const unsigned int color) {

   sceGuDisable(GU_TEXTURE_2D);

   lineVertex* Line = (lineVertex*)sceGuGetMemory(2 * sizeof(lineVertex));

   Line[0].color = color;
   Line[0].x = x1;
   Line[0].y = y1;
   Line[0].z = z1;

   Line[1].color = color;
   Line[1].x = x2;
   Line[1].y = y2;
   Line[1].z = z1;

   sceGuDrawArray(GU_LINES, GU_COLOR_8888 | GU_VERTEX_32BITF | GU_TRANSFORM_3D, 2, 0, Line);

   sceGuEnable(GU_TEXTURE_2D);

   return 0;

}
```

As you can see it looks a lot like the 2D function. In this case we do not need to disable the depth test because we use 3D functions now and we need to test the object to see if it is in front or behind some other object. With the creation of the vertices only the extra dimension is added. To let the PSP render the line 3D I have specified it with GU_TRANSFORM_3D.

As a basic for this part of the tutorial I have a combination of both tutorial 1 and 2 as source. The reason for this is that you need the basic game loop but also the matrix usage as in tutorial 2. So adding this function to the source of part 1 will not be enough. Please use the source of the triangle tutorial or the source files below. With the source files you will see a white line rendered across the screen and a small blue line in the middle. The blue line is the 3D line. Since we render it on its own it looks like it is 2D but it is 3D.

Source files part 2

Download the source files.

MakeFile

The makefile for all parts

```
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 = -lc -g -lpng -lz -lpspgum -lpspgu -lstdc++ -lm

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

Compile and what happens ?! You see the texture shown on screen, not 3D but 2D, and behind the triangle is rotated around the screen. We have combined 2D and 3D!