

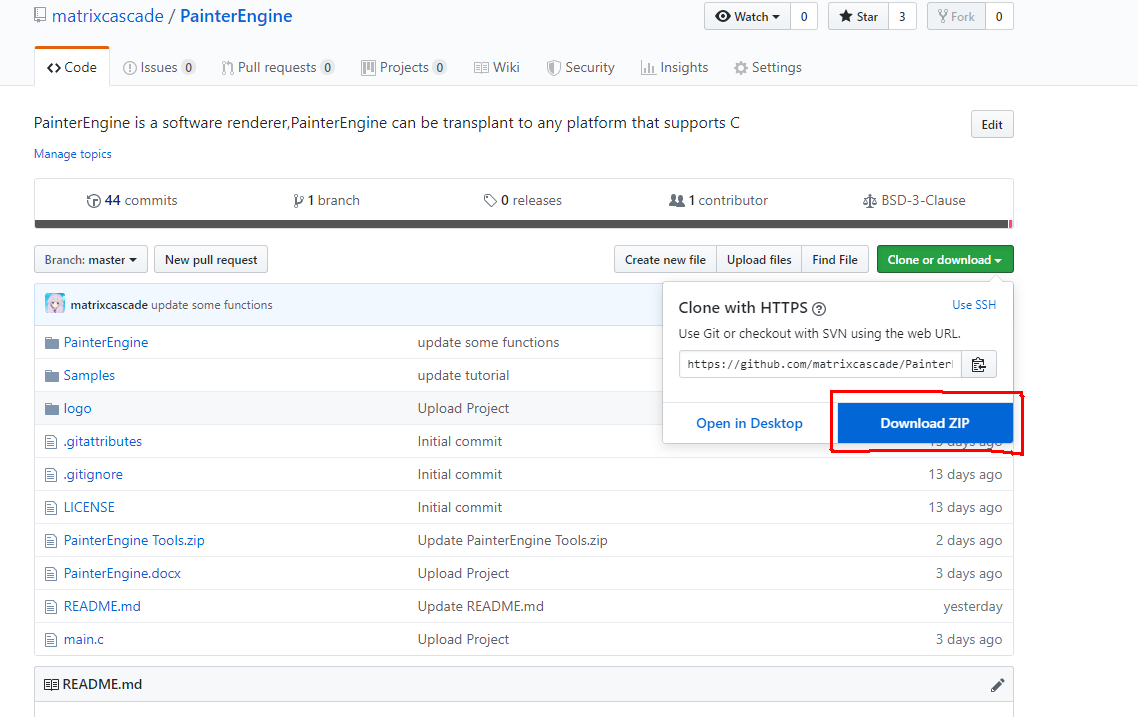
PainterEngine快速入门教程

PainterEngine是一个由C语言编写的完全开源的图形图像游戏引擎,可用于图形图像,游戏,UI交互应用的快速开发,PainterEngine完全独立于操作系统,其内核代码无需任何修改就能够被移植到windows、linux、Android、iOS甚至是裸机环境的嵌入式平台上来。PainterEngine基于BSD开源协议发布,你可以任意修改拓展PainterEngine的源码并用在你的任意项目中。

## 安装PainterEngine

PainterEngine无需额外的安装步骤，你只需要从GitHub或其它渠道下载PainterEngine的代码就可以了。

1. 访问网址：<https://github.com/matrixcascade/PainterEngineHelper>
2. 点击Clone or download---->Download Zip.
3. 解压下载的压缩包
4. 打开解压压缩包,再解压压缩包中的PainterEngine Tools



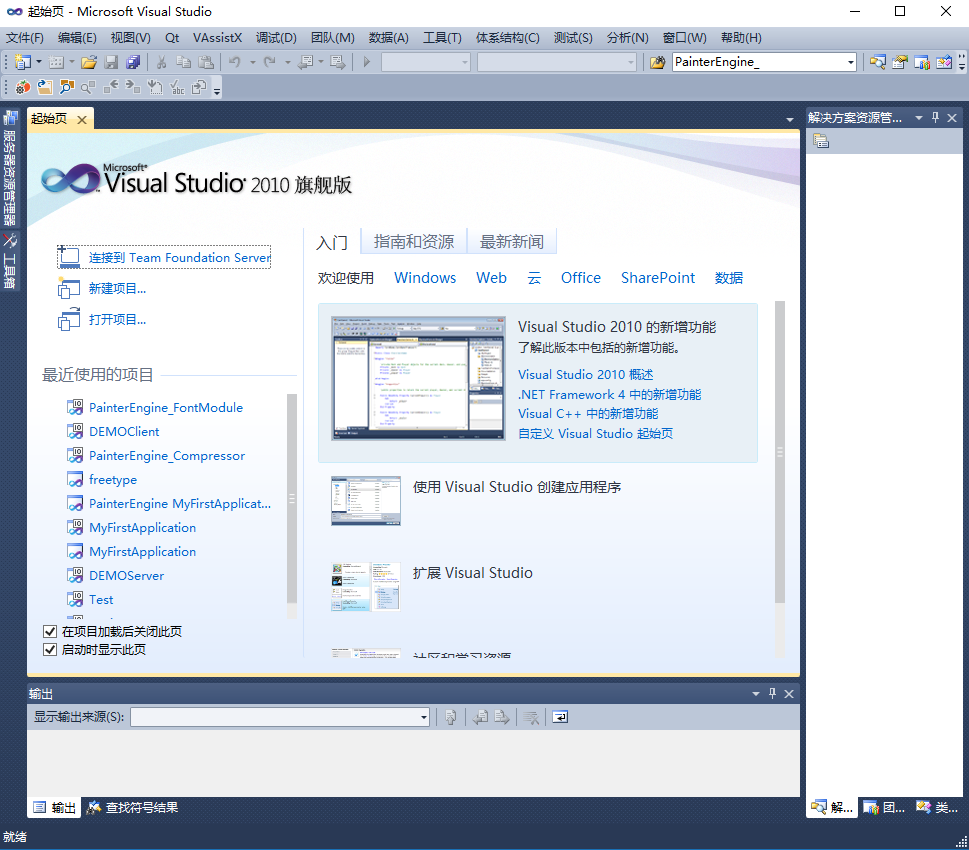
完成后你会得到这样的文件夹



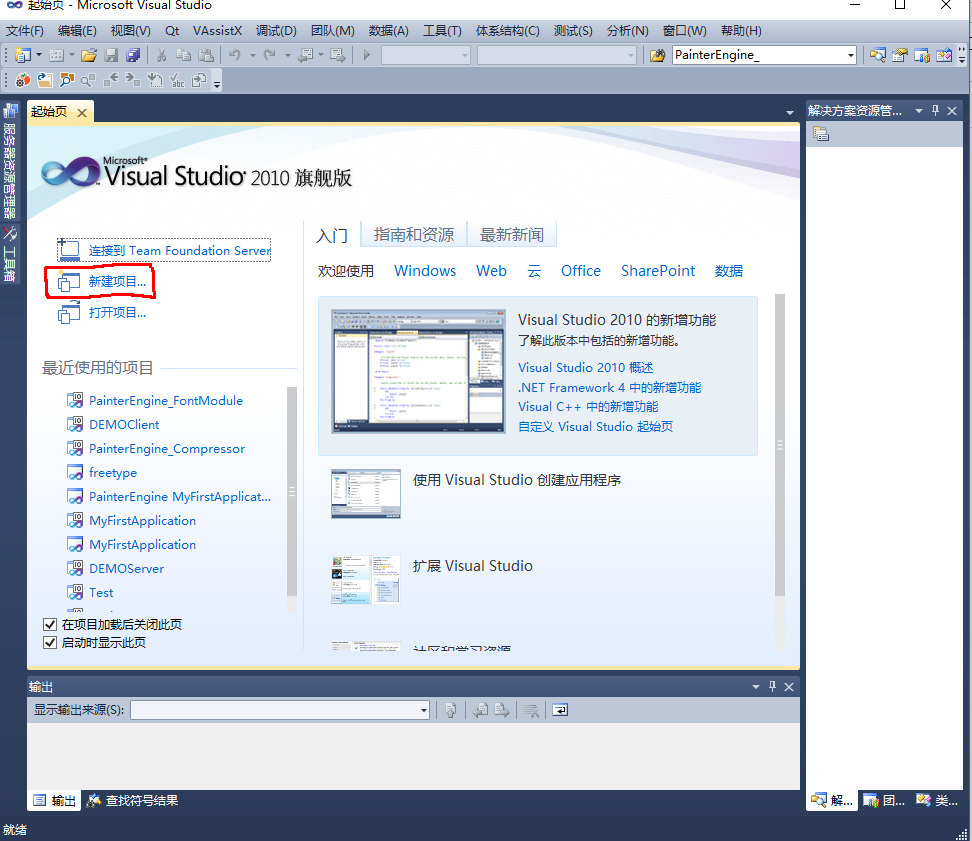
## 使用PainterEngine

以Visual Studio 2010为例:

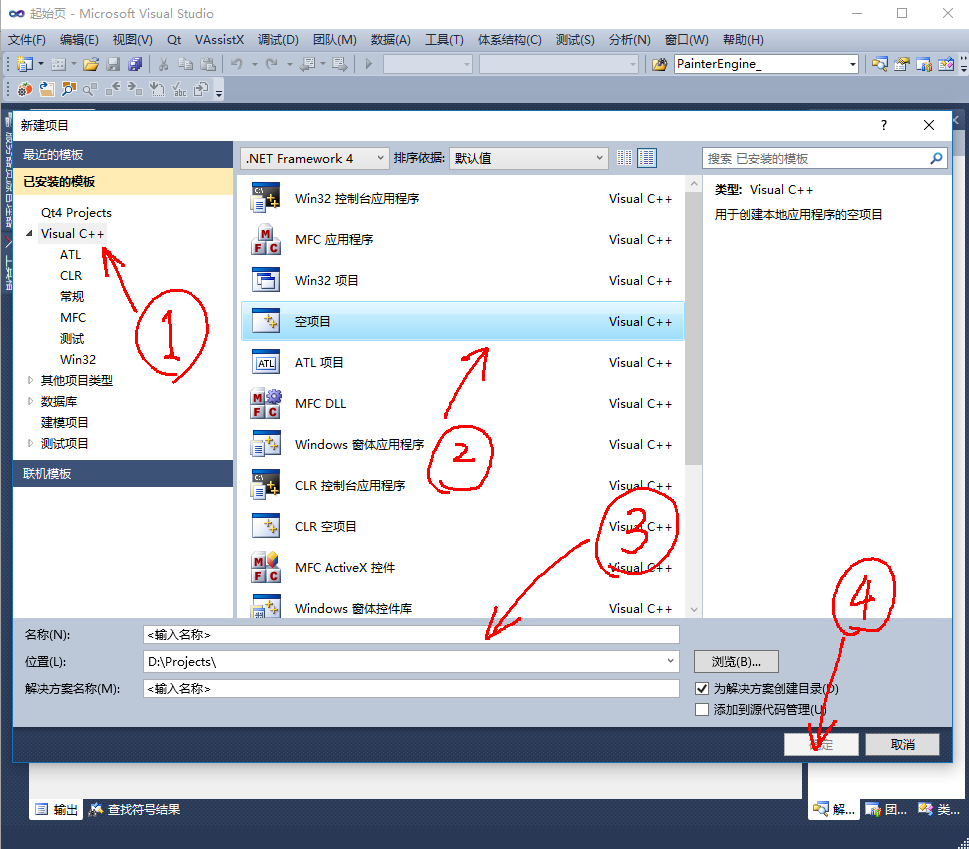
第一步:打开Visual Studio



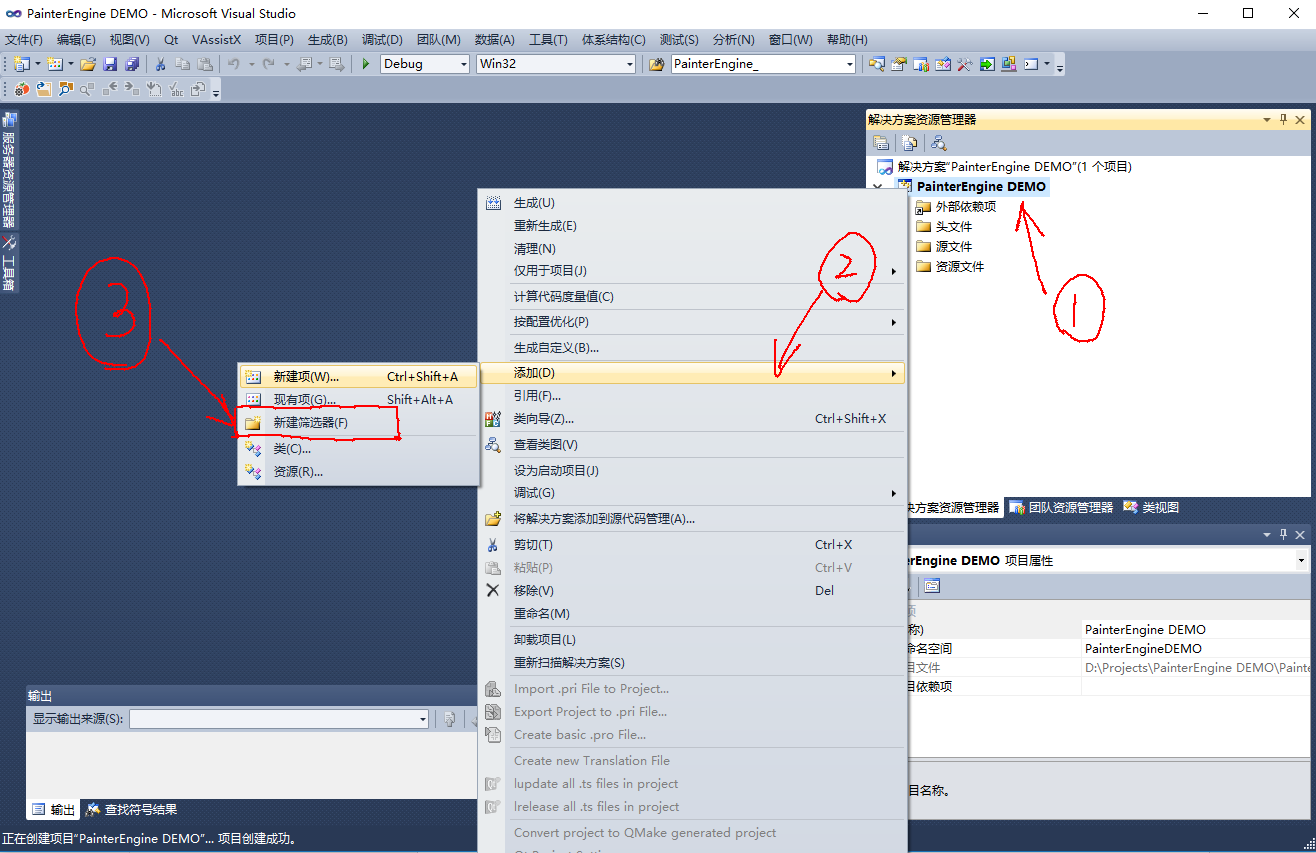
第二步,点击新建项目



第三步,点击visual C++,选择一个空项目,然后随便填一个名字后点击确定

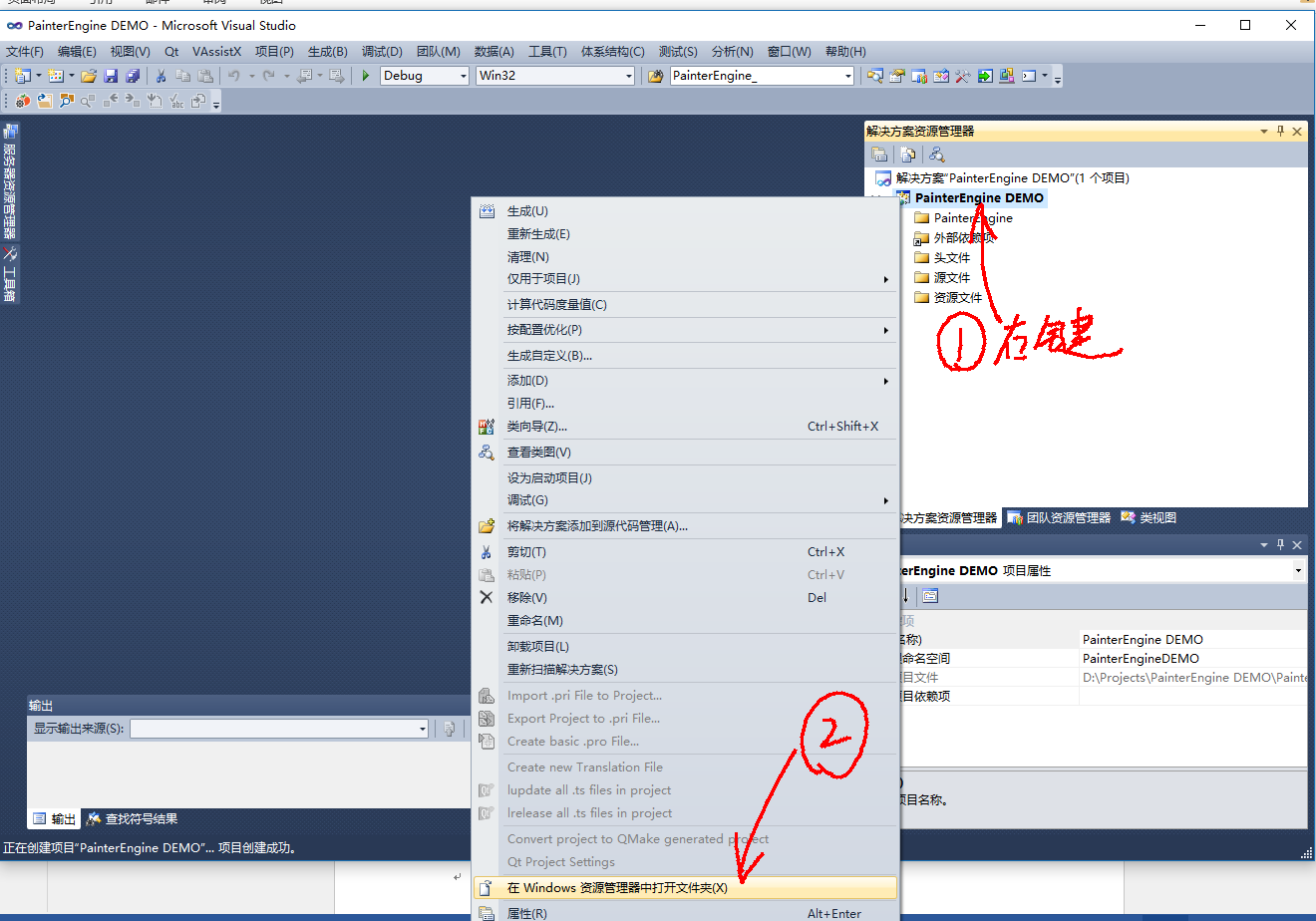


\*第四步,右键点击项目,选择添加,新建筛选器,然后把筛选器取名为PainterEngine,这步只是建议,可以选择不做.

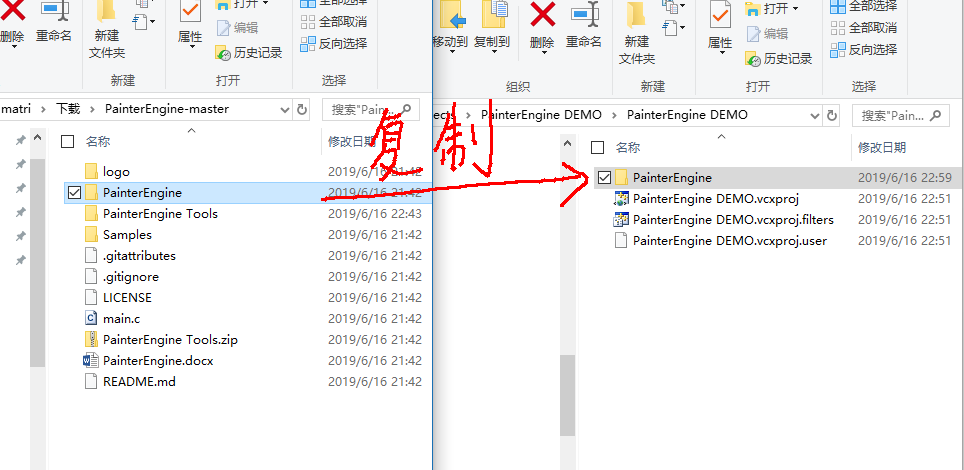




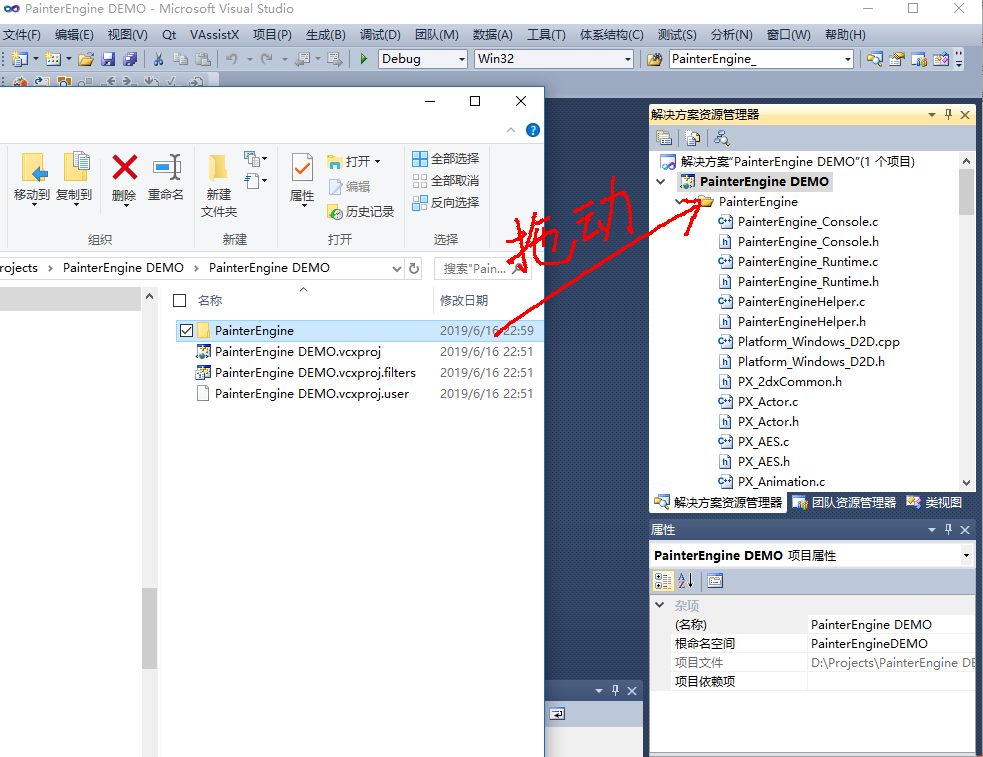
第五步,将PainterEngine复制到这个项目的目录下,右键点击项目.点击在windows资源管理器中打开文件夹.



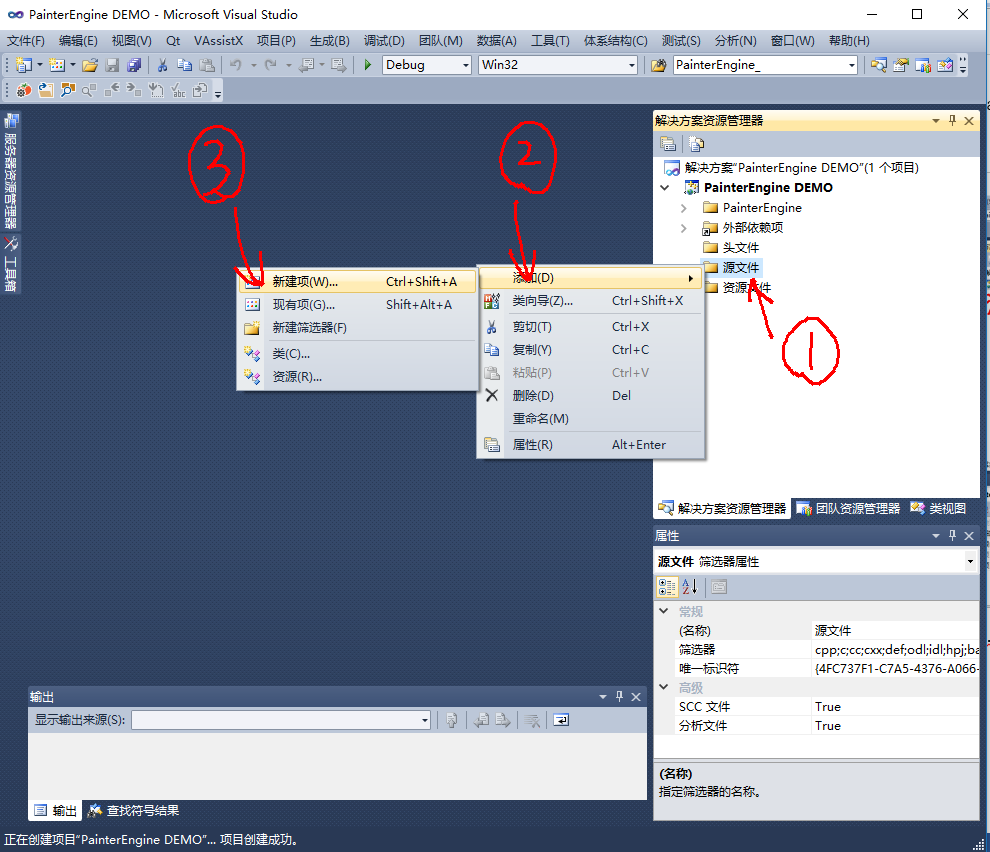
第六步,把PainterEngine复制到项目目录下



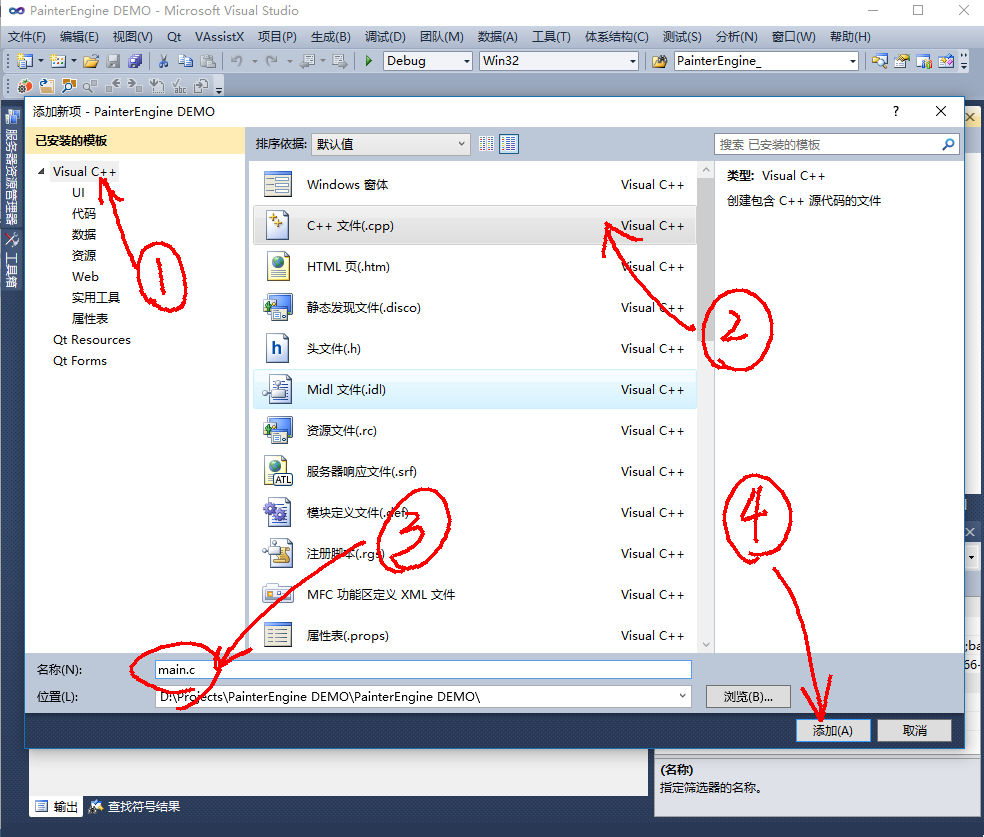
第七步,把PainterEngine这个目录拖动到Visual Studio项目中(如果有做第四部就把他拖到筛选器中)



第八步,右键源代码🡪添加🡪新建项



第九步:创建main.c(或者别的什么名字.c)你可以开始写代码了



## 第一个PainterEngine项目

在main.c中编写如下代码,然后点击运行

#include "./PainterEngine/PainterEngineHelper.h" //包含PainterEngine头文件

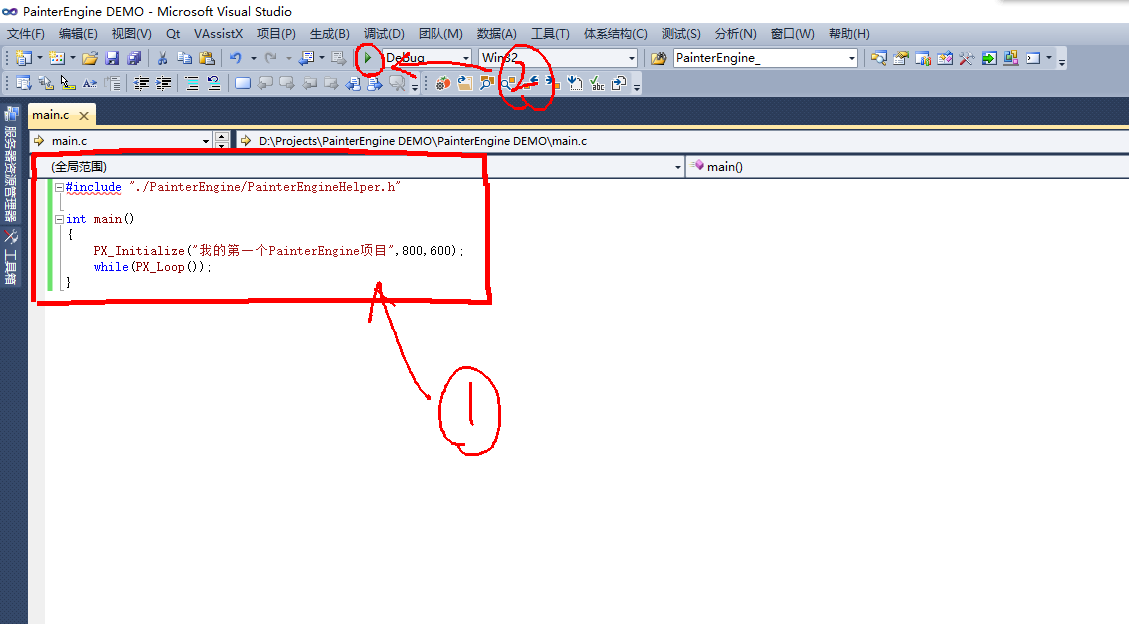
int main()

{

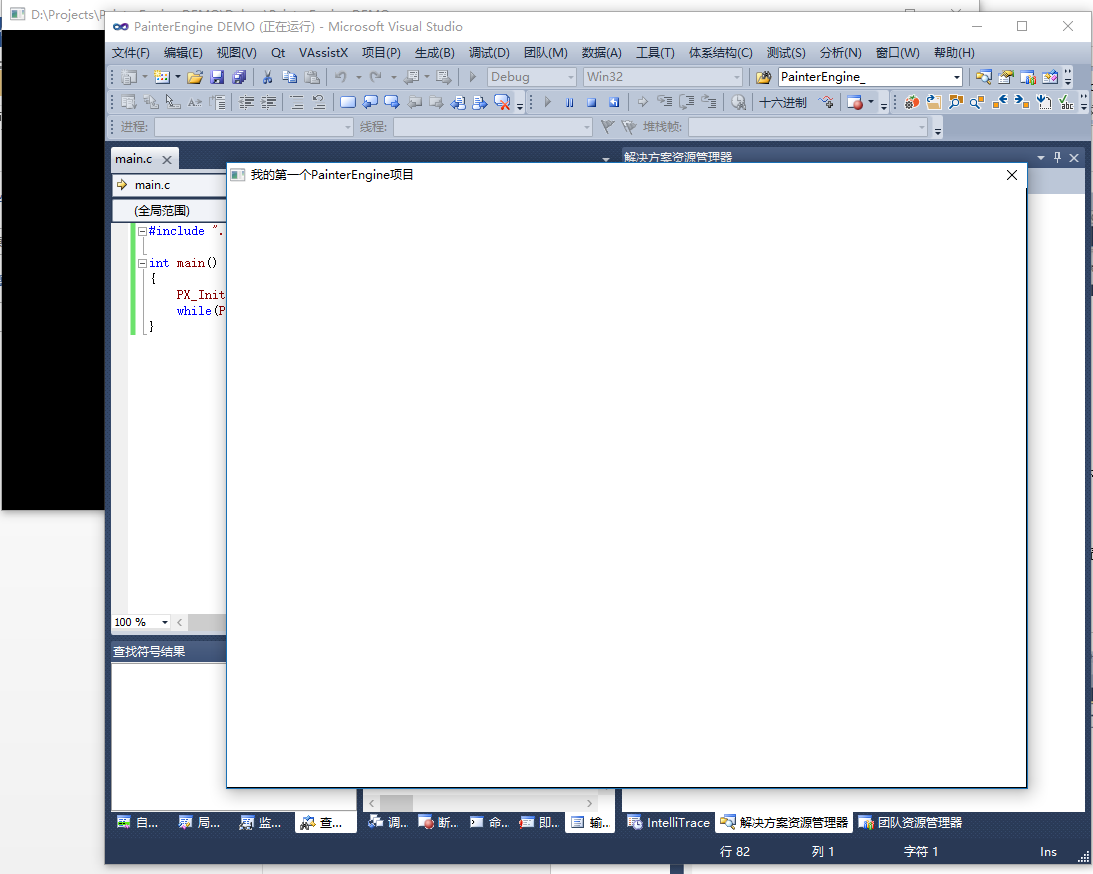
PX\_Initialize("我的第一个PainterEngine项目",800,600);//创建一个窗体,大小是800x600

while(PX\_Loop());//PainterEngine循环

}



第一个PainterEngine项目就运行起来了.

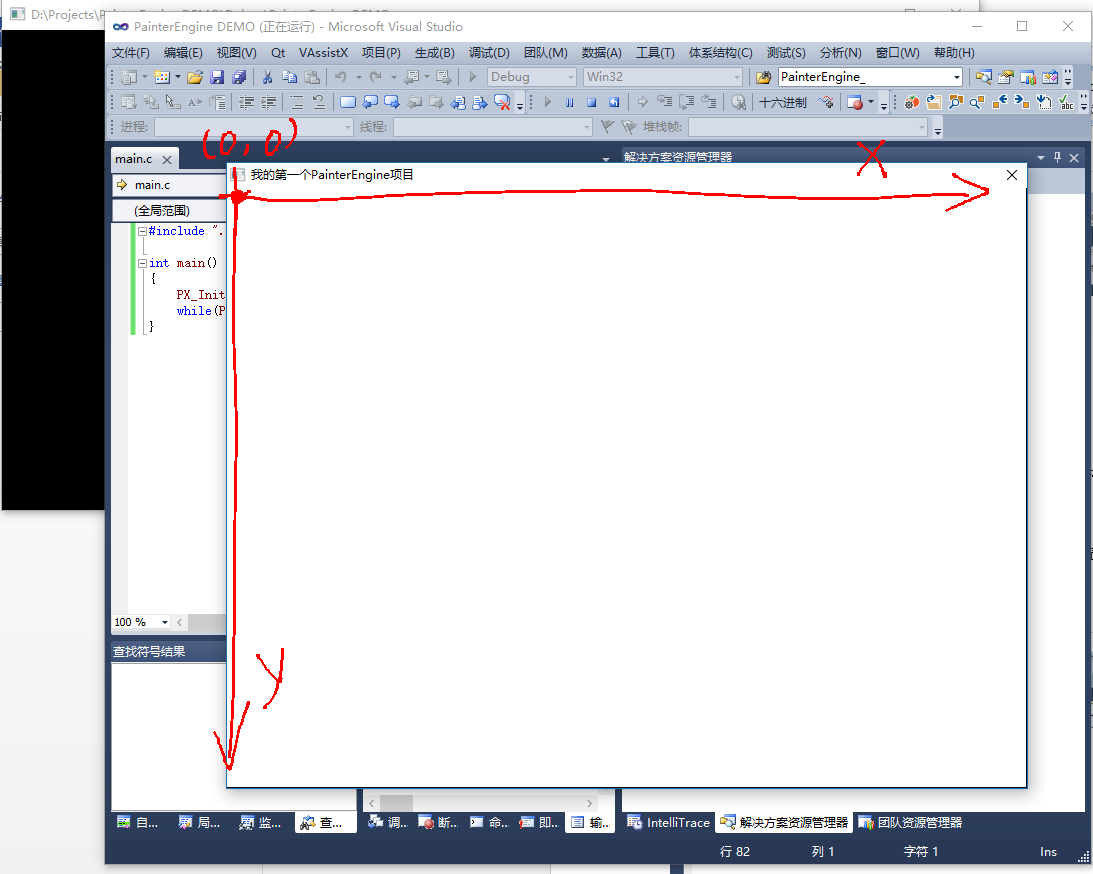


什么,还不会,视频教程地址在这

<https://www.bilibili.com/video/av55501266>

## 使用PainterEngine绘制几何图形

PainterEngine中使用的二维坐标系布置如下



### 画线

例如我们要在(100,100)向(600,500)画一根线宽为8像素的红色线,那么只需要写代码

#include "./PainterEngine/PainterEngineHelper.h"

int main()

{

PX\_Initialize("我的第一个PainterEngine项目",800,600);

PX\_GeoDrawLine(PX\_GetSurface(),100,100,600,500,8,PX\_COLOR(255,255,0,0));

while(PX\_Loop());

}

其中PX\_GeoDrawLine(PX\_GetSurface(),100,100,600,500,8,PX\_COLOR(255,255,0,0));就是画线的函数

第一个参数填写PX\_GetSurface()固定不变的,

2,3,4,5个参数就是两个坐标了,代表线的开始坐标和终止坐标

第六个参数是线的宽度,这里是8

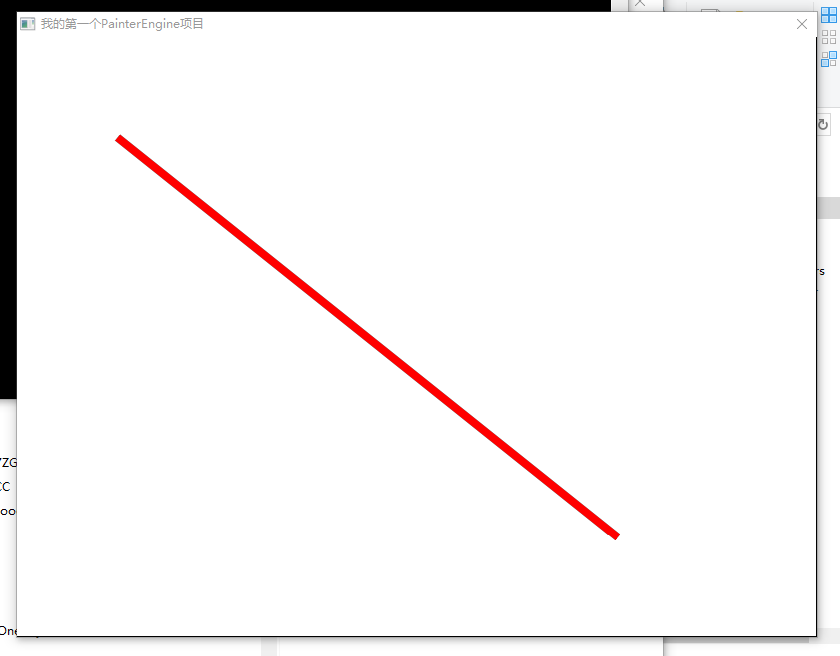
最后一个参数是线的颜色了,PX\_COLOR()第一个参数是Alpha也就是透明度,0表示完全透明,255表示完全不透明

第二个参数是Red颜色分量范围也是0-255,0表示一点也不红,255表示大红,因为我们要画红线所以是255

第三第四个是green和blue颜色分量也就是绿色和蓝色,和红色同理,不过我们只要红色,所以最后就是

(255,255,0,0)了

你可以尝试修改这些数值看看颜色有什么变化



### 画边框

画边框使用函数

px\_void PX\_GeoDrawBorder(px\_surface \*psurface, px\_int left, px\_int top, px\_int rignt, px\_int bottom ,px\_int lineWidth,px\_color color);

下面是示范代码

#include "./PainterEngine/PainterEngineHelper.h"

int main()

{

PX\_Initialize("我的第一个PainterEngine项目",800,600);

PX\_GeoDrawBorder(PX\_GetSurface(),100,100,200,200,8,PX\_COLOR(255,255,0,0));

while(PX\_Loop());

}

第一个参数写PX\_GetSurface

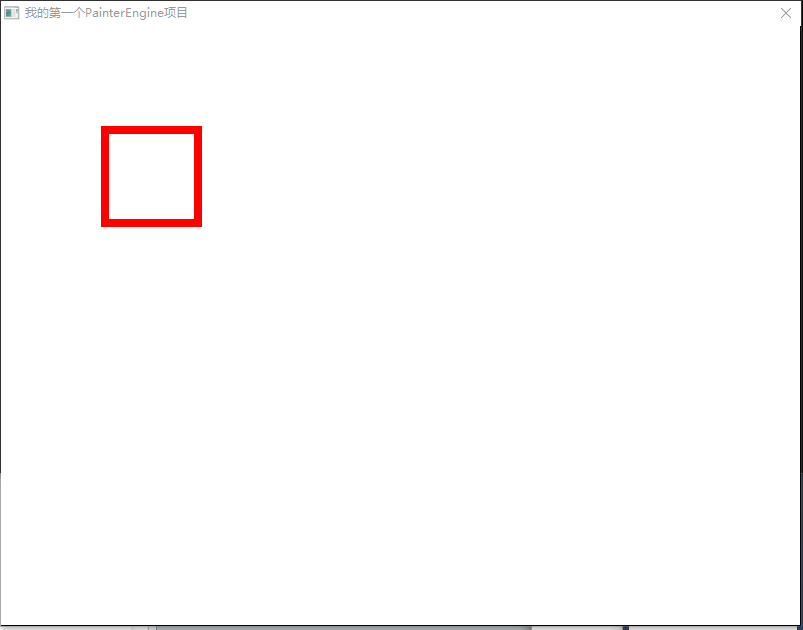
第二第三个参数是这个边框左上角XY坐标

第四第五个参数是边框右下角XY坐标

第六个参数是这个边框的线宽,在这里是8

最后一个就是边框ARGB颜色了

运行结果如下



### 画矩形

和画边框一样,画矩形的代码仅仅是PX\_GeoDrawBorder换成了PX\_GetDrawRect并且没有线宽的参数函数原型如下

px\_void PX\_GeoDrawRect(px\_surface \*psurface, px\_int left, px\_int top, px\_int right, px\_int bottom,px\_color color);

第一个参数写PX\_GetSurface

第二第三个参数是这个边框左上角XY坐标

第四第五个参数是边框右下角XY坐标

最后一个就是边框ARGB颜色了

示范代码

#include "./PainterEngine/PainterEngineHelper.h"

int main()

{

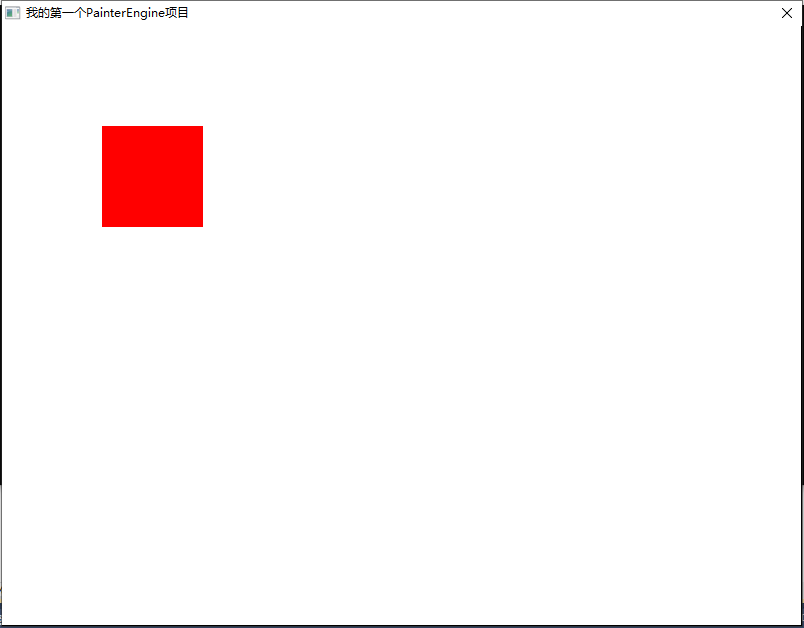
PX\_Initialize("我的第一个PainterEngine项目",800,600);

PX\_GeoDrawRect(PX\_GetSurface(),100,100,200,200,PX\_COLOR(255,255,0,0));

while(PX\_Loop());

}

运行结果



### 画实心圆

画实心圆的函数如下

px\_void PX\_GeoDrawSolidCircle(px\_surface \*psurface, px\_int x,px\_int y,px\_int Radius,px\_color color );

第一个参数仍然是PX\_GetSurface

第二第三个参数是圆心的XY坐标

第四个参数是圆的半径

最后一个参数是圆的颜色

示范代码:

#include "./PainterEngine/PainterEngineHelper.h"

int main()

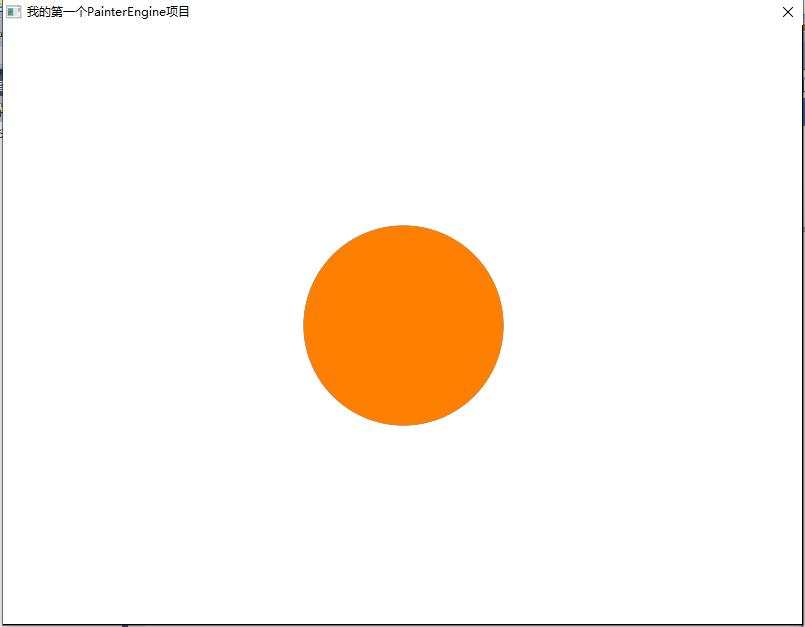
{

PX\_Initialize("我的第一个PainterEngine项目",800,600);

PX\_GeoDrawSolidCircle(PX\_GetSurface(),400,300,100,PX\_COLOR(255,255,128,0));

while(PX\_Loop());

}



### 画圆

函数原型:

px\_void PX\_GeoDrawCircle(px\_surface \*psurface,px\_int x,px\_int y,px\_int Radius ,pt\_int lineWidth,px\_color color);

参数说明:

psurface 渲染表面

x，y 圆心

radius 半径

lineWidth 圆的线宽

color 颜色

示范代码:

#include "./PainterEngine/PainterEngineHelper.h"

int main()

{

PX\_Initialize("我的第一个PainterEngine项目",800,600);

PX\_GeoDrawCircle(PX\_GetSurface(),400,300,100,5,PX\_COLOR(255,255,128,0));

while(PX\_Loop());

}

运行结果



### 环

函数原型:

px\_void PX\_GeoDrawRing(px\_surface \*psurface, px\_int x,px\_int y,px\_int Radius,px\_int lineWidth,px\_color color,px\_uint start\_angle,px\_uint end\_angle);

参数说明:

psurface 渲染表面

x，y 环心

radius 半径

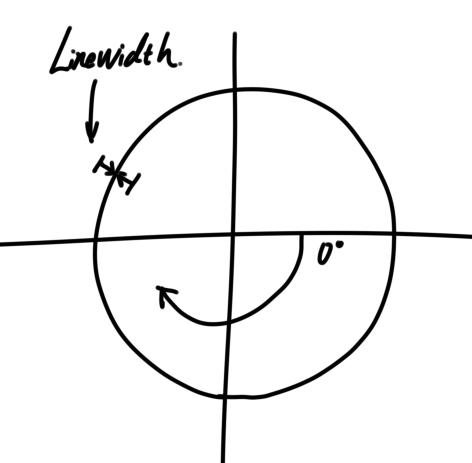
lineWidth 线宽

color 颜色

start\_angle 起始角度

end\_angle 终止角度

这个环遵循顺时针方向



示范代码:

#include "./PainterEngine/PainterEngineHelper.h"

int main()

{

PX\_Initialize("我的第一个PainterEngine项目",800,600);

PX\_GeoDrawRing(PX\_GetSurface(),400,300,100,35,PX\_COLOR(255,255,128,0),90,210);

while(PX\_Loop());

}



### 扇形

函数原型

px\_void PX\_GeoDrawSector(px\_surface \*psurface, px\_int x,px\_int y,px\_int Radius\_outside,px\_int Radius\_inside,px\_color color,px\_uint start\_angle,px\_uint end\_angle);

参数说明:

psurface 渲染表面

x，y 环心

radius 半径

outside 外径

inside 内径

color 颜色

start\_angle 起始角度

end\_angle 终止角度

这个扇形遵循顺时针方向旋转

示范代码

#include "./PainterEngine/PainterEngineHelper.h"

int main()

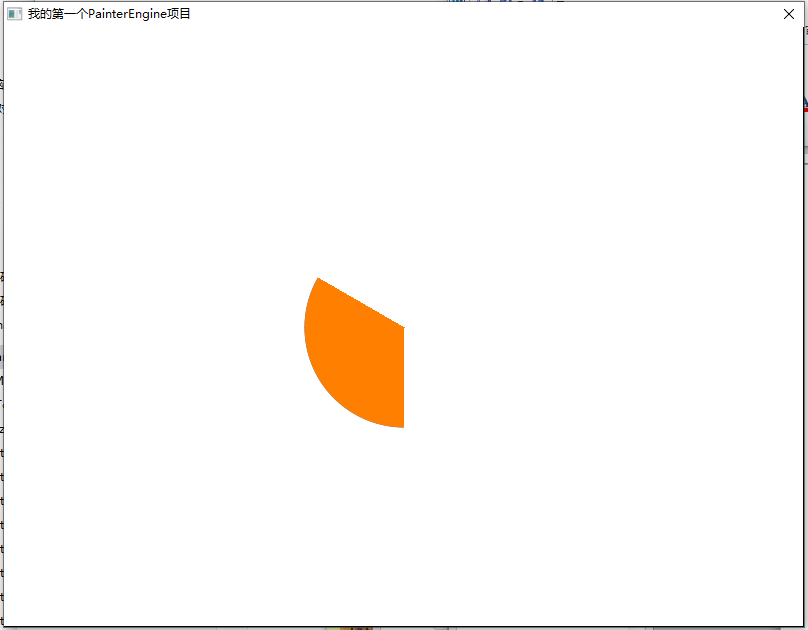
{

PX\_Initialize("我的第一个PainterEngine项目",800,600);

PX\_GeoDrawSector(PX\_GetSurface(),400,300,100,0,PX\_COLOR(255,255,128,0),90,210);

while(PX\_Loop());

}



### 圆角边框

函数原型:

px\_void PX\_GeoDrawRoundRect(px\_surface \*psurface, px\_int left, px\_int top, px\_int right, px\_int bottom,px\_float roundRaduis,px\_float linewidth,px\_color color);

参数

psurface 渲染表面

left top right bottom 左上角坐标和右下角坐标

roundRadius 圆角半径

lineWidth 线宽

color 颜色

示范代码:

#include "./PainterEngine/PainterEngineHelper.h"

int main()

{

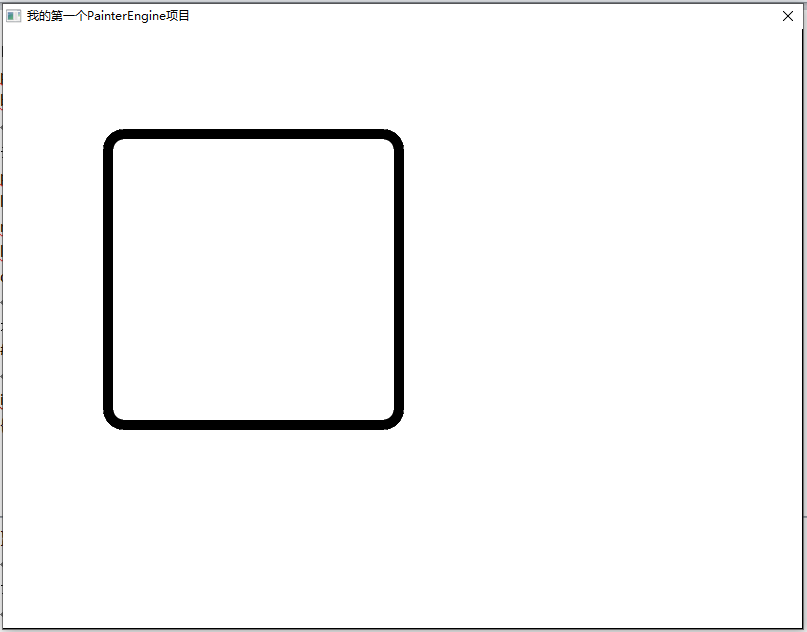
PX\_Initialize("我的第一个PainterEngine项目",800,600);

PX\_GeoDrawRoundRect(PX\_GetSurface(),100,100,400,400,20,10,PX\_COLOR(255,0,0,0));

while(PX\_Loop());

}

运行界面:



### 圆角矩形

函数原型:

px\_void PX\_GeoDrawSolidRoundRect(px\_surface \*psurface, px\_int left, px\_int top, px\_int right, px\_int bottom,px\_float roundRaduis,px\_float linewidth,px\_color color);

参数说明:

psurface 渲染表面

left top right bottom 位置描述

roundRadius 圆角半径

color 颜色

示范代码:

#include "./PainterEngine/PainterEngineHelper.h"

int main()

{

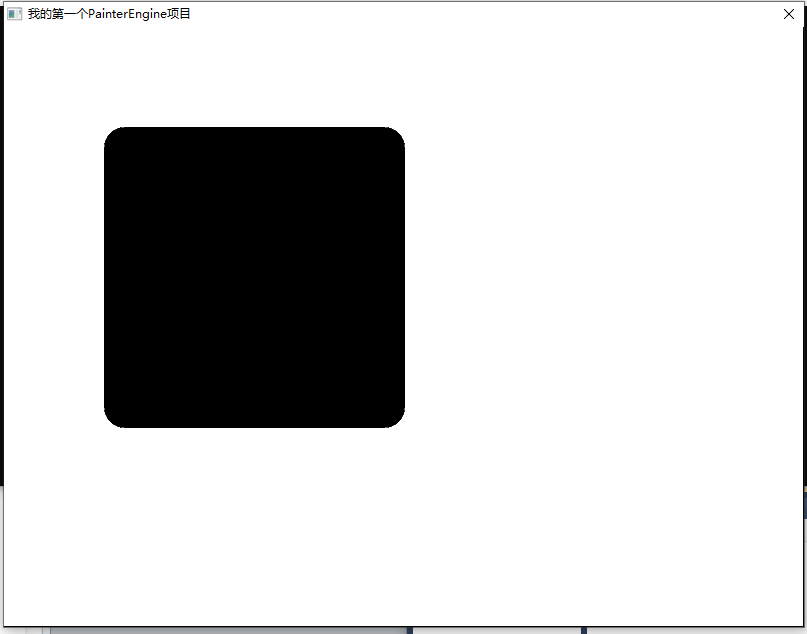
PX\_Initialize("我的第一个PainterEngine项目",800,600);

PX\_GeoDrawSolidRoundRect(PX\_GetSurface(),100,100,400,400,20,PX\_COLOR(255,0,0,0));

while(PX\_Loop());

}

运行界面



## 使用PainterEngine绘制文本

### 绘制默认文本

使用PainterEngine绘制文本非常简单,只需要调用函数PX\_FontDrawText就行了

函数原型:

px\_void PX\_FontDrawText(px\_surface \*psurface,int x,int y,px\_char \*Text,px\_color Color,PX\_FONT\_ALIGN align);

其中,第一个参数是PX\_GetSurface();

X,Y表示文本第一个文字的二维坐标

Text就是要绘制的文本内容了

Color就是文本的颜色了,例如我们要在200,300的位置绘制”Hello PainterEngine”那么代码就可以这样写

Align 表示文本的对齐方式,一般使用PX\_FONT\_ALIGN\_XLEFT靠左对齐或PX\_FONT\_ALIGN\_CENTER居中对齐

#include "./PainterEngine/PainterEngineHelper.h"

int main()

{

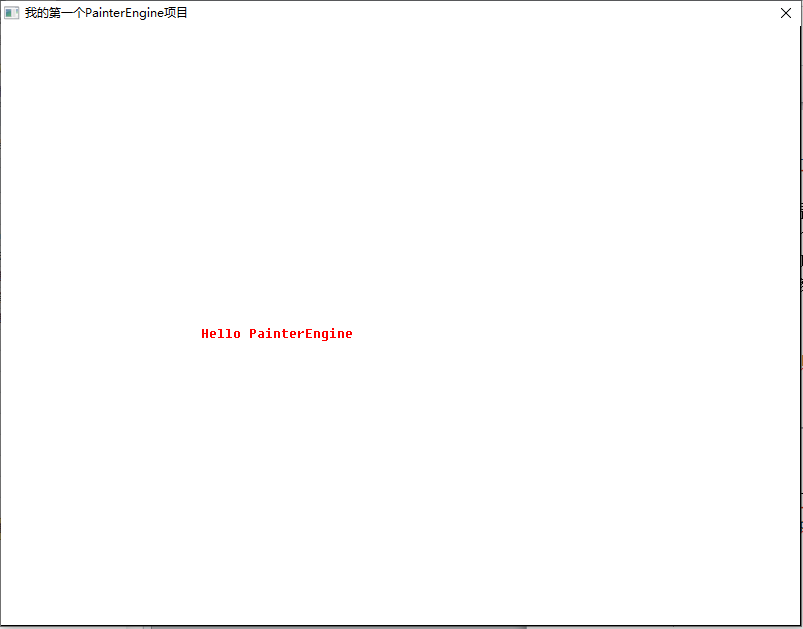
PX\_Initialize("我的第一个PainterEngine项目",800,600);

PX\_FontDrawText(PX\_GetSurface(),200,300,"Hello PainterEngine",PX\_COLOR(255,255,0,0), PX\_FONT\_ALIGN\_XLEFT);

while(PX\_Loop());

}

运行结果如下:



### 使用字模文本

PainterEngine 允许使用TTF文件制作字模库绘制字模文本

需要使用字模文本,你需要准备一些文件

1. 包含所需汉字和英文字母及标点符号的txt字典文件(如果你需要所有汉字及字母符号,你可以在PainterEngine FontModule中找到en.txt及chs.txt文件,en.txt文件包含了所有的汉字及标点,chs.txt包含了极少数生僻字除外的汉字)
2. 你需要字体的TTF文件
3. PainterEngine\_FontModule.exe,用于生成字模的程序,你可以在PainterEngine Tools中找到这个程序.

下面示范制作一个字模文件:

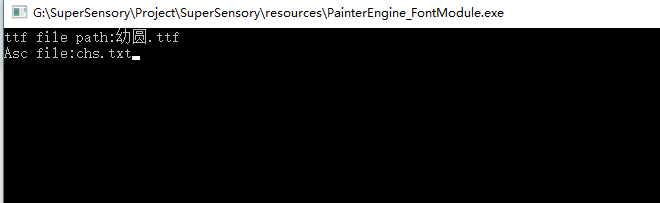
其中,中文以”幼圆”字体为例

英文以AGEBCYR为例

打开tools中的PainterEngine\_FontModule.exe,输入ttf文件路径



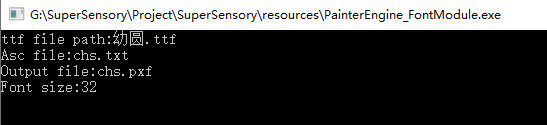
回车,然后输入字典,这里以chs.txt为例



回车,输入输出pxf文件路径,我直接输出到当前目录chs.pxf



回车,输入字模中字体的大小,这里以32为例



回车,生成成功,当前目录下生成了chs.pxf,如果你需要生成支持英文的字模,参照上述方法将ASC File换为en.txt,ttf换为AGEBCYR.txt就可以了

注意,字体越大,字典包含的文字越多.生成的字模库越大,因此尽可能踢除不需要的文字

Txt文件必须以utf-16LE格式保存

下面编写代码,示范绘制字模

#include "../../PainterEngine/PainterEngine/PainterEngineHelper.h"

int main()

{

PX\_FontModule fm;//定义字模结构体

PX\_Initialize("DEMO",800,600);

PX\_FontModuleInitialize(PX\_GetResourceLibrary()->mp,&fm);//初始化字模库,用资源库的内存池来保存字模库

PX\_LoadFontModuleFromFile(&fm,"chs.pxf");//加载汉字字模

PX\_LoadFontModuleFromFile(&fm,"en.pxf");//加载英文字模

//绘制字模文本

//第一个参数为绘制surface

//第二第三个参数为绘制位置(X,Y坐标)

//第三个参数为需要绘制的文本,需要使用unicode字符集(UTF16)

//第四个为字体颜色

//第六个为对齐方式,这里是居中对齐

PX\_FontModuleDrawText(PX\_GetSurface(),400,300,(px\_word \*)(L"这是一个字模示范 FontModule DEMO"),PX\_COLOR(255,255,0,128),&fm,PX\_FONT\_ALIGN\_XCENTER);

while(PX\_Loop());

}

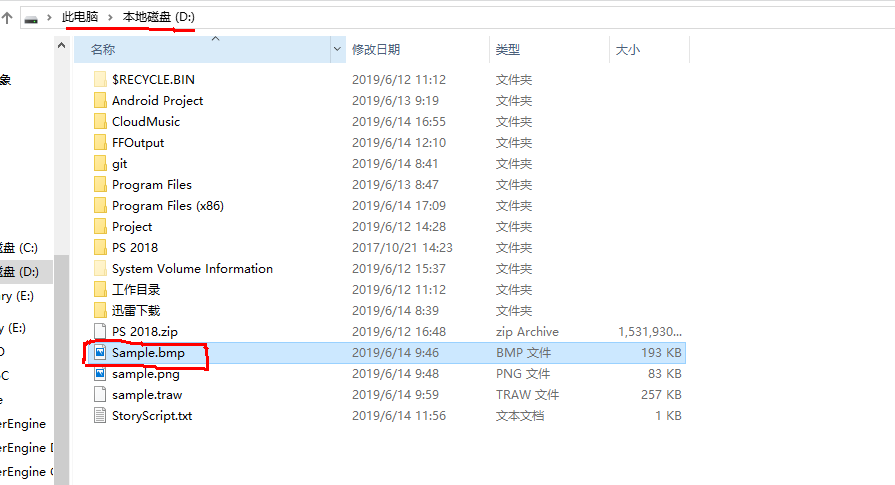


## 使用PainterEngine绘制图像文件

### 绘制BMP,TRAW图片

在PainterEngine中,所有的图像数据最终都是由一个纹理结构体进行管理的,不过不用担心,我们不需要了解其中内部的实现细节,如果我们要使用图片,我们只需要定义它就可以了,PainterEngine原生支持两种格式的文件,一种是BMP位图文件,一种是TRAW格式的专属图片文件,这个格式可以通过PNG图片转换得到.通过附加解码库后,能够支持PNG及JPG文件的支持.

我们先用BMP文件做个示范,首先笔者在D盘下放了一个Sample.bmp文件





PainterEngine的代码这样写

#include "./PainterEngine/PainterEngineHelper.h"

int main()

{

px\_texture tex;//定义纹理结构

PX\_Initialize("我的第一个PainterEngine项目",800,600);

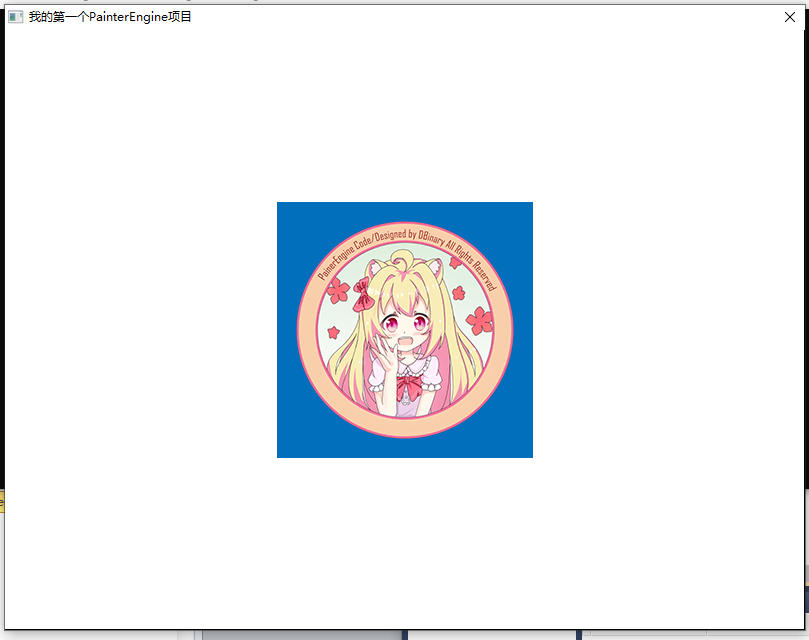
PX\_LoadTextureFromFile(PX\_GetMP(),&tex,"D:\\sample.bmp");//从文件加载一个图片放在纹理中

PX\_TextureRender(PX\_GetSurface(),&tex,400,300,PX\_TEXTURERENDER\_REFPOINT\_CENTER,PX\_NULL);//绘制纹理

while(PX\_Loop());

}

运行结果:



代码解释:

其中px\_texture tex;是一个纹理结构体,在需要使用图片数据时,我们都应该定义一个结构体

然后就是

PX\_LoadTextureFromFile(PX\_GetMP(),&tex,"D:\\sample.bmp");

这个函数的功能就是从文件加载一个图片文件到纹理中,它的函数原型是

px\_bool PX\_LoadTextureFromFile(px\_memorypool \*mp,px\_texture \*tex,px\_char \*path)

其中,第一个参数是PX\_GetMP(),在熟悉PainterEngine内存管理机制前你可以就这样写暂时不用管

第二个参数是图像存储到的纹理结构体指针,因为我们之前定义了一个tex的纹理结构,所以这里就是&tex

第三个参数是图片文件的路径,这个路径可以是绝对路径也可以是相对路径.正如之前所述,我们d盘有一sample.bmp文件,所以这里就是"D:\\sample.bmp"

最后是把图片绘制出来了

PX\_TextureRender(PX\_GetSurface(),&tex,400,300,PX\_TEXTURERENDER\_REFPOINT\_CENTER,PX\_NULL);

这个函数的第一个参数是PX\_GetSurface(),在熟悉PainterEngine的纹理管理机制时你可以这样写不用管

第二个参数是要绘制的纹理结构指针,当然就是之前的tex了

第三第四个参数是纹理绘制的位置,这里是(400,300)

第五个参数是PX\_TEXTURERENDER\_REFPOINT\_CENTER 表示纹理依照坐标点居中显示

当然还有PX\_TEXTURERENDER\_REFPOINT\_LEFTTOP表示按左上角对齐显示

PX\_TEXTURERENDER\_REFPOINT\_RIGHTBOTTOM表示按右下角对齐显示

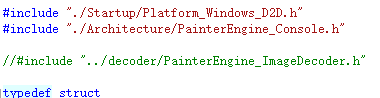
…..

最后一个参数是通道混合参数,用的不多可以暂时不用管写PX\_NULL

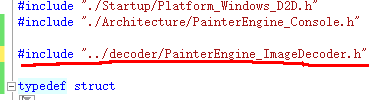
### 绘制PNG/JPEG图片

上一章节中PainterEngine能够原生加载BMP及TRAW格式的图片,如果需要加载PNG/JPG格式的图片,需要为其添加支持库.打开头文件PainterEngineHelper.h

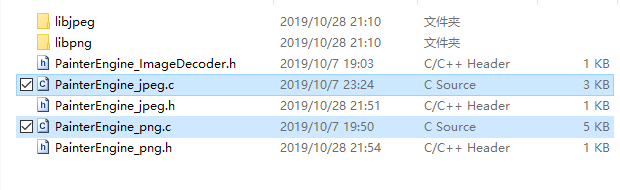
1.找到如下代码:



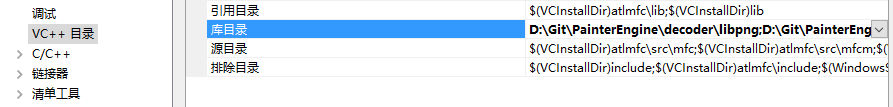
2.将该注释删除使其生效



3.将decoder/PainterEngine\_jpeg.c和decoder/PainterEngine\_png.c添加到当前的项目中



3.将./decoder/libpng和./decoder/libjpeg目录添加到包含目录中,例如visual studio,或者将./decoder/libpng和./decoder/libjpeg目录下的lib文件添加到当前项目中(注意libpngd.lib, zlibd.lib为debug的库版本, libpng.lib, zlib.lib为release版本)



示范代码如下

#include "../../PainterEngine/PainterEngine/PainterEngineHelper.h"

int *main*()

{

px\_texture tex;

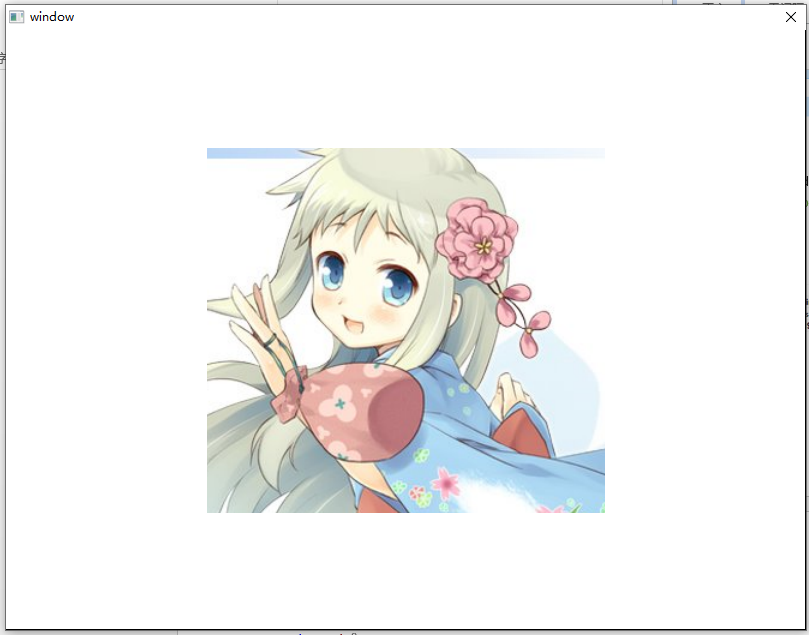
PX\_Initialize("window",800,600);

PX\_LoadTextureFromFile(PX\_GetMP(),&tex,"D:/demo.png");

PX\_TextureRender(PX\_GetSurface(),&tex,400,300,PX\_TEXTURERENDER\_REFPOINT\_CENTER,PX\_NULL);

while(PX\_Loop());

}



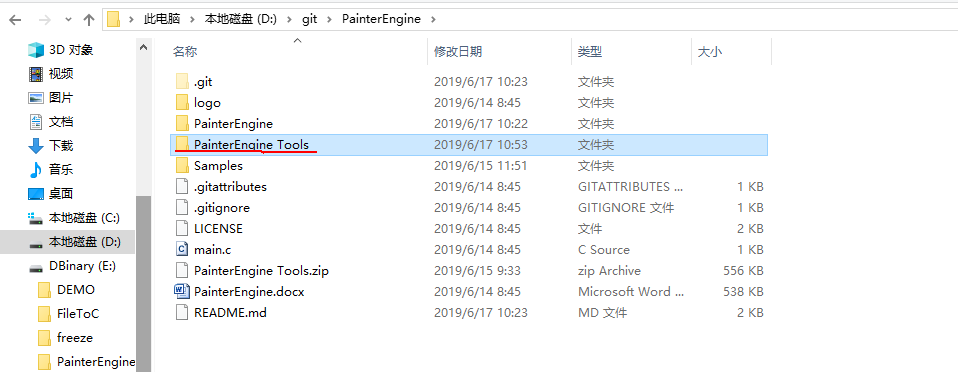
### 绘制TRAW图片

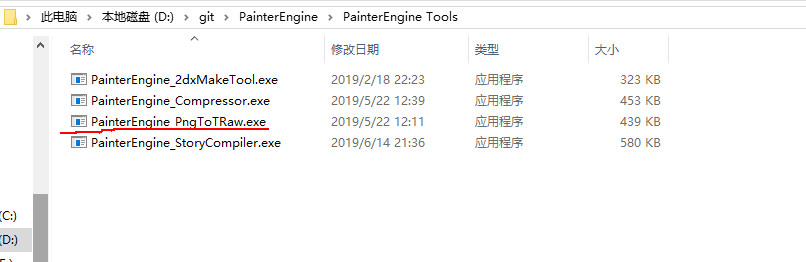
可以看到,BMP图像是没有ALPHA通道的,也就是说BMP图片一定是正方形没有透明色的,而PNG图片有,例如,下面是PNG图片



可以看到,周围都是透明色,中心一个圆形图像,透明像素在游戏中非常重要,为了显示PNG图片,在PainterEngine中需要对PNG文件进行转换

首先打开PainterEngine目录,在PainterEngine Tools中你可以找到转换工具





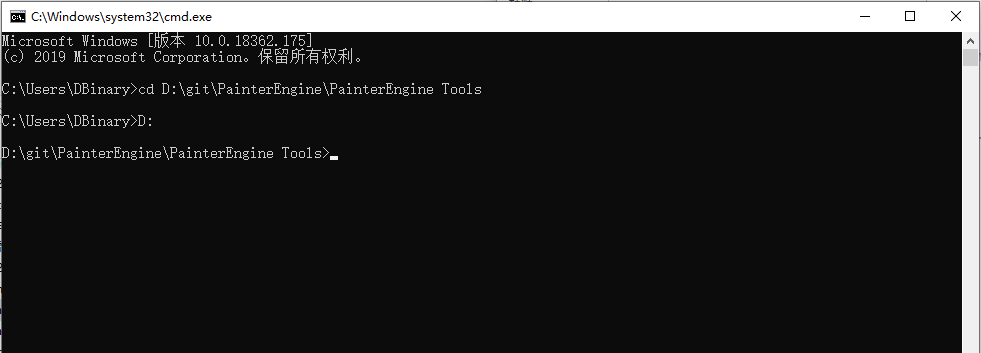
首先,按下win+R 输入cmd 回车或者在开始菜单输入cmd回车,来到命令提示符

输入cd 工具所在目录,例如我这里是D:\git\PainterEngine\PainterEngine Tools

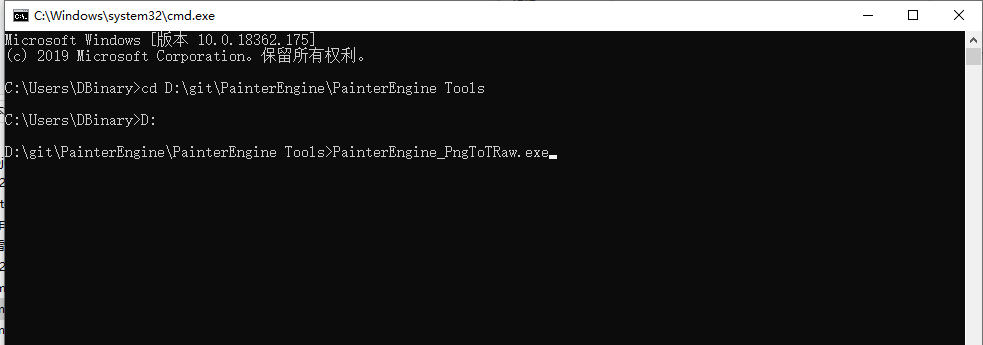
那么就是

cd D:\git\PainterEngine\PainterEngine Tools

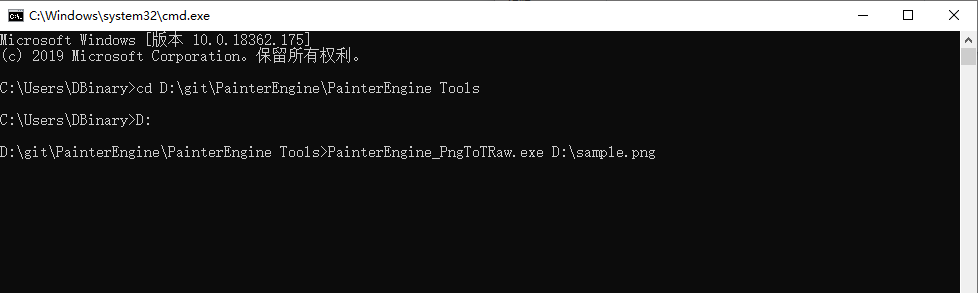
D:



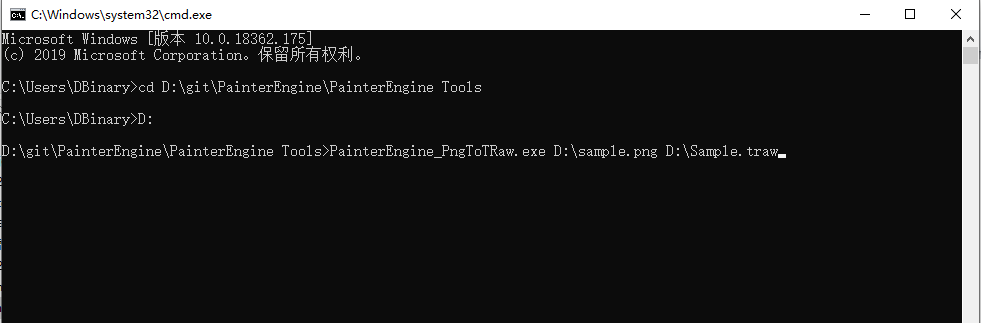
按tab键切换到这个工具的exe(直接输入也可以)



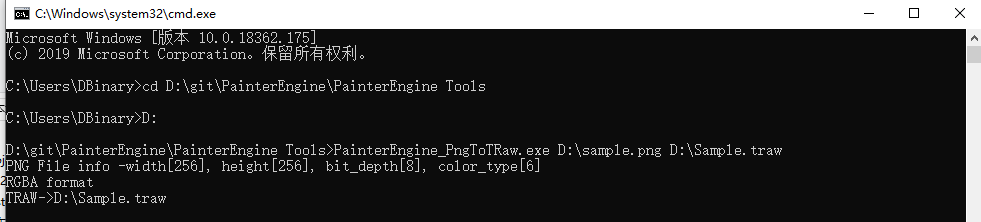
输入要转换文件的路径,例如我D盘下有个sample.png的图片,那么就是



输入输出文件路径,例如直接输出到D盘下那么就是



回车,转换完毕



D盘下出现转换好的traw文件



绘制Traw文件和绘制BMP文件的代码一模一样,只是文件名换一下就可以了

#include "./PainterEngine/PainterEngineHelper.h"

int main()

{

px\_texture tex;//定义纹理结构

PX\_Initialize("我的第一个PainterEngine项目",800,600);

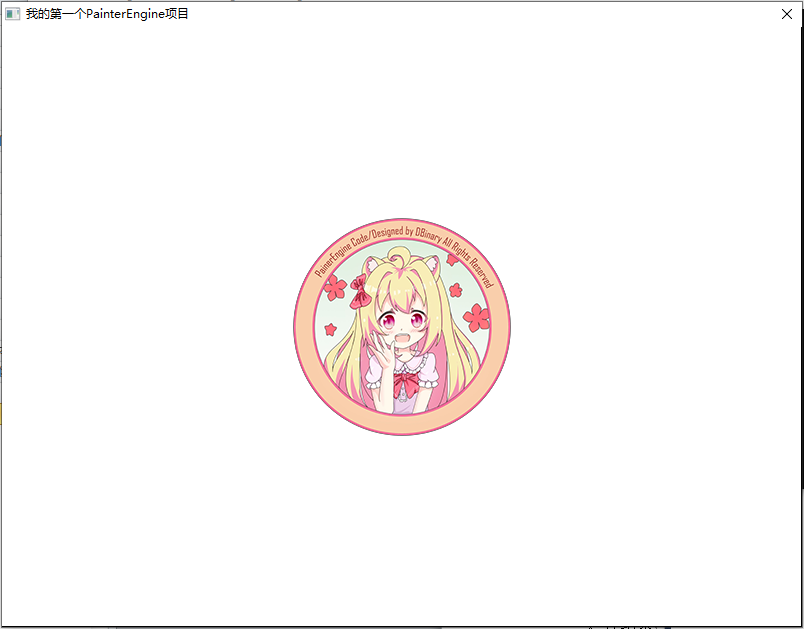
PX\_LoadTextureFromFile(PX\_GetMP(),&tex,"D:\\sample.traw");//从文件加载一个图片放在纹理中

PX\_TextureRender(PX\_GetSurface(),&tex,400,300,PX\_TEXTURERENDER\_REFPOINT\_CENTER,PX\_NULL);//绘制纹理

while(PX\_Loop());

}

运行结果



### 缩放/旋转图片

需要旋转或者缩放图片,只需要将PX\_TextureRender替换成函数PX\_TextureRenderEx

函数原型如下:

px\_void PX\_TextureRenderEx(px\_surface \*psurface,px\_texture \*resTexture,px\_int x,px\_int y,PX\_TEXTURERENDER\_REFPOINT refPoint,PX\_TEXTURERENDER\_BLEND \*blend,px\_float scale,px\_float Angle);

前面的参数和PX\_TextureRender之前的参数一样,不同的是结尾多了scale和angle两个参数,分别表示图片的缩放倍数和顺时针旋转的角度,例如下面的代码表示图片不放大(1.0倍),顺时针旋转90度

#include "./PainterEngine/PainterEngineHelper.h"

int main()

{

px\_texture tex;//定义纹理结构

PX\_Initialize("我的第一个PainterEngine项目",800,600);

PX\_LoadTextureFromFile(PX\_GetMP(),&tex,"D:\\sample.traw");//从文件加载一个图片放在纹理中

PX\_TextureRenderEx(PX\_GetSurface(),&tex,400,300,PX\_TEXTURERENDER\_REFPOINT\_CENTER,PX\_NULL,1.0f,90);//绘制纹理

while(PX\_Loop());

}

运行结果



或者放大1.5倍,旋转150度

PX\_TextureRenderEx(PX\_GetSurface(),&tex,400,300,PX\_TEXTURERENDER\_REFPOINT\_CENTER,PX\_NULL,1.5f,150);



注意,这个函数在图像的缩放上并没有进一步滤波,因此在图像的缩放过程中会出现锯齿,但它的绘制效率高,非常适合用在动画当中,如果需要更高质量的缩放图片,可以使用函数

PX\_TextureCreateScale函数,参阅PainterEngine.docx说明文档

示范代码:

#include "./PainterEngine/PainterEngineHelper.h"

int main()

{

px\_texture tex,stex;//定义纹理结构

PX\_Initialize("我的第一个PainterEngine项目",800,600);

PX\_LoadTextureFromFile(PX\_GetMP(),&tex,"D:\\sample.traw");//从文件加载一个图片放在纹理中

PX\_TextureCreateScale(PX\_GetMP(),&tex,512,512,&stex);//缩放纹理

PX\_TextureRender(PX\_GetSurface(),&stex,400,300,PX\_TEXTURERENDER\_REFPOINT\_CENTER,PX\_NULL);//绘制缩放后的纹理

while(PX\_Loop());

}



## 使用PainterEngine创建UI控件

### 按钮

PainterEngine中,UI对象都是以PX\_Object \*对象的形式存在的,因此在创建UI控件之前,需要先定义一个PX\_Object\*对象来存储UI控件

例如下面的代码,将创建一个按钮控件

#include "./PainterEngine/PainterEngineHelper.h"

int main()

{

PX\_Object \*Button;

PX\_Initialize("我的第一个PainterEngine项目",800,600);

Button=PX\_Object\_PushButtonCreate(PX\_GetMP(),PX\_GetUiRoot(),400,300,84,32,"确定",PX\_COLOR(255,0,0,0));

while(PX\_Loop());

}



PX\_Object\_PushButtonCreate的函数原型如下

PX\_Object \* PX\_Object\_PushButtonCreate(px\_memorypool \*mp,PX\_Object \*Parent,px\_int x,px\_int y,px\_int Width,px\_int Height,px\_char \*Text,px\_color Color)

第一个参数为内存池,可填写PX\_GetMP();

第二个参数填写PX\_GetUiRoot() 在熟悉PainterEngine对象管理机制之前,建议就这样写

第三第四个参数xy表示按钮左上角的坐标

第五第六个参数Width,Height表示按钮的宽度和高度

第七个参数Text表示按钮的名称

第八个参数表示按钮文本的颜色

为了给按钮添加点击响应机制,我们需要定义响应函数(就是按钮点击时执行的函数),这个函数有固定的格式

px\_void 函数名(px\_object \*pObject,PX\_Object\_Event e,px\_void \*ptr);

例如如下代码:

px\_void OnButtonClick(PX\_Object \*pObject,PX\_Object\_Event e,px\_void \*ptr)

{

MessageBox(NULL,"确定按钮被点击了","",MB\_OK);

}

当按钮被点击时弹个窗口出来

最后,我们需要把这个响应函数绑定到按钮上,使用函数

px\_int PX\_ObjectRegisterEvent( PX\_Object \*Object,px\_uint Event,px\_void (\*ProcessFunc)(PX\_Object \*,PX\_Object\_Event e,px\_void \*user\_ptr),px\_void \*user)

第一个参数表示UI控件对象,在这里指的就是这个button了

第二个参数表示响应事件,点击事件是PX\_OBJECT\_EVENT\_EXECUTE

第三个参数就是响应函数的函数指针了,在例子中是OnButtonClick

第四个参数是传递参数指针,填写PX\_NULL就可以了表示没有要传递的参数

完整代码如下

#include "./PainterEngine/PainterEngineHelper.h"

px\_void OnButtonClick(PX\_Object \*pObject,PX\_Object\_Event e,px\_void \*ptr)

{

MessageBox(NULL,"确定按钮被点击了","",MB\_OK);

}

int main()

{

PX\_Object \*Button;

PX\_Initialize("我的第一个PainterEngine项目",800,600);

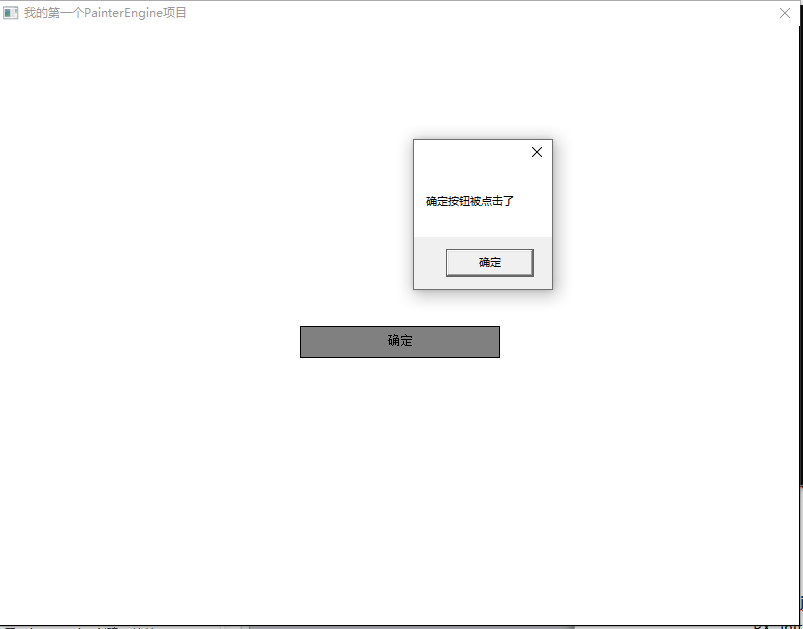
Button=PX\_Object\_PushButtonCreate(PX\_GetMP(),PX\_GetUiRoot(),300,300,200,32,"确定",PX\_COLOR(255,0,0,0));

PX\_ObjectRegisterEvent(Button,PX\_OBJECT\_EVENT\_EXECUTE,OnButtonClick,PX\_NULL);

while(PX\_Loop());

}

运行结果



### 文本框

同样定义一个PX\_Object \*对象指针存储文本框

示范代码如下

#include "./PainterEngine/PainterEngineHelper.h"

int main()

{

PX\_Object \*Edit;

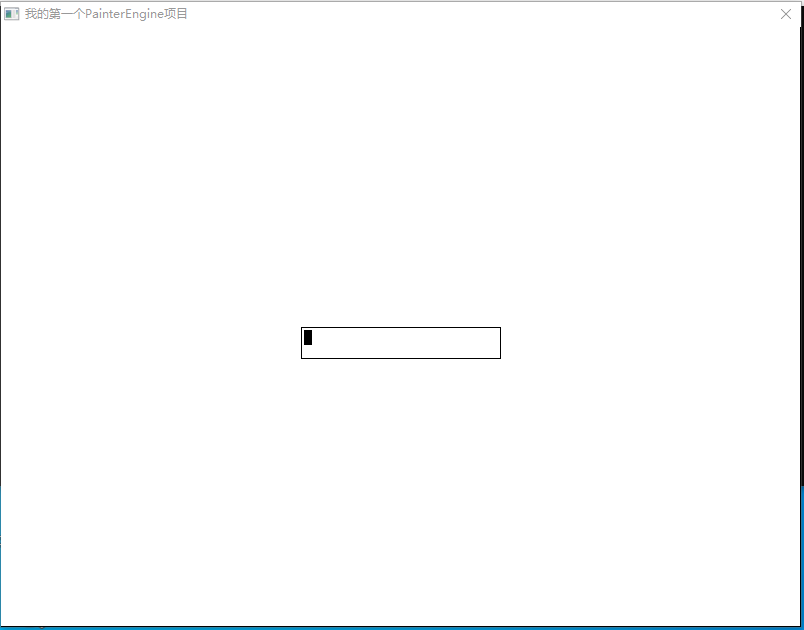
PX\_Initialize("我的第一个PainterEngine项目",800,600);

Edit=PX\_Object\_EditCreate(PX\_GetMP(),PX\_GetUiRoot(),300,300,200,32,PX\_COLOR(255,0,0,0));

while(PX\_Loop());

}

运行结果





其中

Edit=PX\_Object\_EditCreate(PX\_GetUiRoot(),300,300,200,32,PX\_COLOR(255,0,0,0));

即为创建文本框的代码,该函数原型如下

PX\_Object\* PX\_Object\_EditCreate(px\_memorypool \*mp,PX\_Object \*Parent,px\_int x,px\_int y,px\_int Width,px\_int Height,px\_color TextColor )

第一个参数为内存池,可直接写PX\_GetMP();

第二个参数填写PX\_GetUiRoot() 在熟悉PainterEngine对象管理机制之前,建议就这样写

第三第四个参数xy表示文本框左上角的坐标

第五第六个参数Width,Height表示文本框的宽度和高度

第七个参数表示文本框文本的颜色

使用PX\_Object\_EditGetText来取得文本框中的文本,为了示范,这里和按钮控件一起使用

示范代码:

#include "./PainterEngine/PainterEngineHelper.h"

PX\_Object \*Button,\*Edit;

px\_void OnButtonClick(PX\_Object \*pObject,PX\_Object\_Event e,px\_void \*ptr)

{

MessageBox(NULL,PX\_Object\_EditGetText(Edit),"文本框的内容是",MB\_OK);

}

int main()

{

PX\_Initialize("我的第一个PainterEngine项目",800,600);

Edit=PX\_Object\_EditCreate(PX\_GetMP(),PX\_GetUiRoot(),300,200,200,24,PX\_COLOR(255,0,0,0));

Button=PX\_Object\_PushButtonCreate(PX\_GetUiRoot(),300,300,200,32,"文本框内容",PX\_COLOR(255,0,0,0));

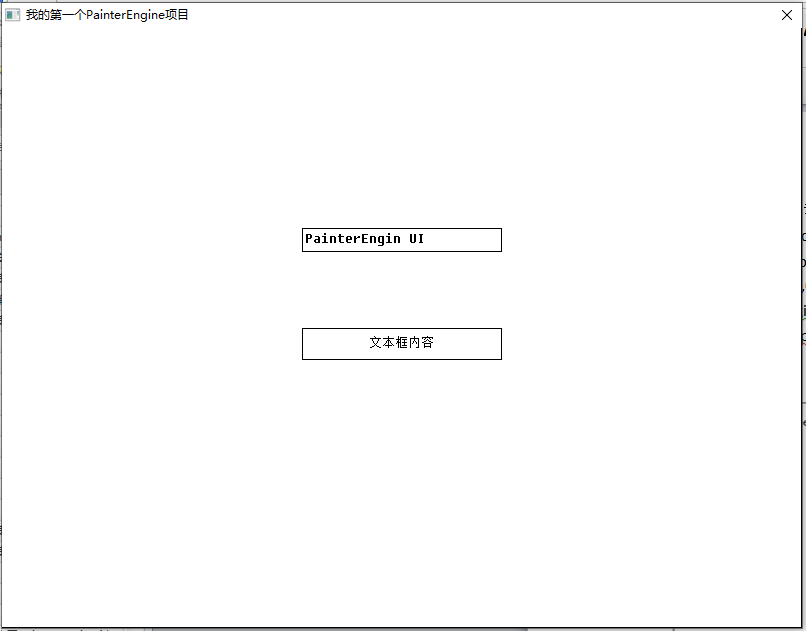
PX\_ObjectRegisterEvent(Button,PX\_OBJECT\_EVENT\_EXECUTE,OnButtonClick,PX\_NULL);

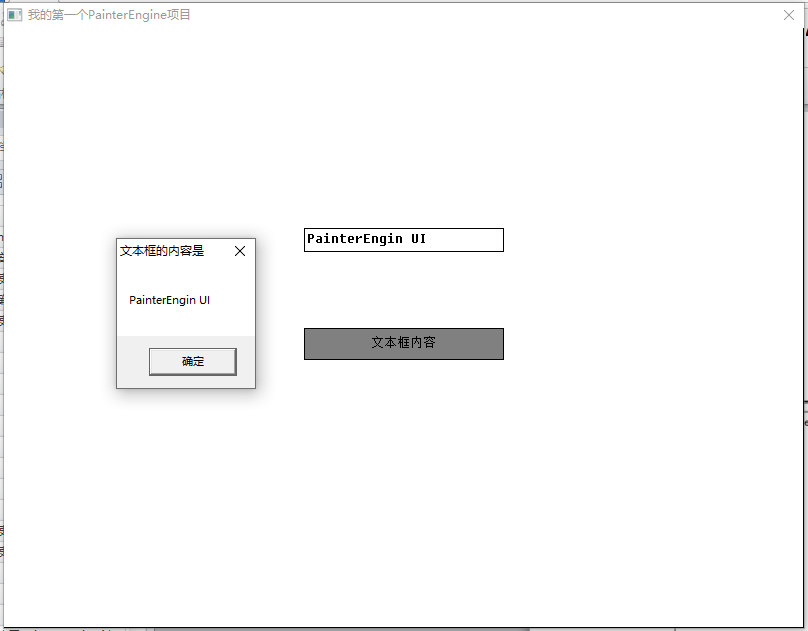
while(PX\_Loop());

}

运行结果







## PainterEngine循环

在while(PX\_Loop())循环体被称为PainterEngine循环,PainterEngine每次更新都会执行循环体内的内容



例如我们希望实现一个旋转的图片动画,我们可以在该循环体内进行编写代码.

#include "./PainterEngine/PainterEngineHelper.h"

int main()

{

px\_int rotateAngle=0;

px\_texture tex;

PX\_Initialize("我的第一个PainterEngine项目",800,600);

PX\_LoadTextureFromFile(PX\_GetMP(),&tex,"D:\\sample.traw");

while(PX\_Loop())

{

//循环体

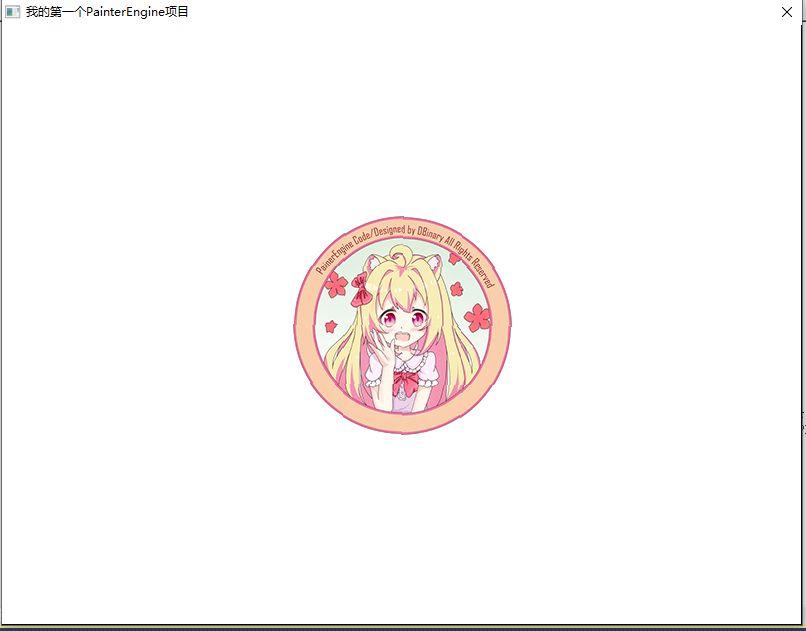
PX\_Clear(PX\_COLOR(255,255,255,255));

PX\_TextureRenderEx(PX\_GetSurface(),&tex,400,300,PX\_TEXTURERENDER\_REFPOINT\_CENTER,PX\_NULL,1.0,rotateAngle++);

}

}

其中PX\_Clear()函数表示用某一颜色刷新渲染表面(把原先内容全部清除掉重新画),这里用白色把之前绘制的内容全部清除掉.



## PainterEngine消息处理机制

在PainterEngine中,所有的IO输入(鼠标,键盘,触摸屏)都会被处理成PX\_OBJECT\_EVENT类消息投递,你可以通过PX\_ObjectRegisterEvent函数(之前有说明)来注册感兴趣的消息响应函数(就像Button被Clicked一样)

例如如下代码:

#include "./PainterEngine/PainterEngineHelper.h"

px\_void PX\_EventProcess(PX\_Object \*pObject,PX\_Object\_Event e,px\_void \*ptr)

{

char message[128];

px\_sprintf(message,sizeof(message),"您在屏幕(%d,%d)按下了",e.Param\_int[0],e.Param\_int[1]);

MessageBox(NULL,message,"",MB\_OK);

}

int main()

{

PX\_Initialize("我的第一个PainterEngine项目",800,600);

PX\_ObjectRegisterEvent(PX\_GetUiRoot(),PX\_OBJECT\_EVENT\_CURSORDOWN,PX\_EventProcess,PX\_NULL);

while(PX\_Loop()){}

}

其中

PX\_ObjectRegisterEvent(PX\_GetUiRoot(),PX\_OBJECT\_EVENT\_CURSORDOWN,PX\_EventProcess,PX\_NULL);表示注册一个PX\_OBJECT\_EVENT\_CURSORDOWN类型的消息响应函数,消息响应函数顾名思义就是当收到这个消息的时候就会去调用该响应函数,这种函数又叫回调函数(参考之前按钮那部分的说明), PX\_OBJECT\_EVENT\_CURSORDOWN表示鼠标左键按下/触摸屏按下事件,当在窗口中按下鼠标左键或者是触摸屏按下时就会调用PX\_EventProcess函数,同时,按下的坐标(xyz)会被存储在e.Param\_int[0],e.Param\_int[1], e.Param\_int[2]当中(大部分时候仅xy有值二维坐标嘛)

运行效果:

除了PX\_OBJECT\_EVENT\_CURSORDOWN,PainterEngine还支持以下几种事件

鼠标/触摸屏事件

====================================================

PX\_OBJECT\_EVENT\_CURSORMOVE //当鼠标在屏幕上移动时

PX\_OBJECT\_EVENT\_CURSORUP //鼠标左键或触摸屏抬起时

PX\_OBJECT\_EVENT\_CURSORRDOWN//鼠标右键按下

PX\_OBJECT\_EVENT\_CURSORDOWN//鼠标左键按下或触摸屏按下时

PX\_OBJECT\_EVENT\_CURSORRUP//鼠标右键抬起

PX\_OBJECT\_EVENT\_EXECUTE //鼠标左键或触摸屏单击

PX\_OBJECT\_EVENT\_CURSORDRAG //鼠标左键按下拖动或触摸屏按下拖动

所有鼠标或触摸屏事件触发时的触发坐标(xyz)会被存储在e.Param\_int[0],e.Param\_int[1], e.Param\_int[2]中

=======================================================

键盘事件

=======================================================

PX\_OBJECT\_EVENT\_KEYDOWN

当键盘按下一个键时触发,按键代码存储在e.param\_int[0]当中

=======================================================

输入法事件

=======================================================

PX\_OBJECT\_EVENT\_STRING

当输入法输入一串字符串时触发,一般用不到,字符串指针在e.user中

=======================================================

其它事件

=======================================================

PX\_OBJECT\_EVENT\_VALUECHAGE 文本框,进度条,拖动条…值改变时

PX\_OBJECT\_EVENT\_DRAGFILE 文件拖进来时

PX\_OBJECT\_EVENT\_IMPACT 两个Object碰撞时,游戏开发用来检测碰撞的.

说明暂略