实现屏幕截屏

DionysosLai 2014-4-11

今天在工作时，参与游戏讨论时，策划人员突然要求实现游戏截屏的功能（呜呜，策划要求无止境啊）。于是，就屁颠屁颠的跑去研究截屏功能。

Cocos2dx使用全新OpenGL ES 2.0绘图，在一些基本的绘图操作，我们可以调用基本的OpengL语音。在Opengl中，为我们提供了读取像素块函数，glReadPixels 。

void glReadPixels(GLint x,GLint y,GLsizesi width,GLsizei height, GLenum format,GLenum type,GLvoid \*pixel);

函数参数(x, y)定义图像区域左下角点的坐标，width和height分别是图像的高度和宽度，\*pixel是一个[指针](http://baike.baidu.com/view/159417.htm)，指向存储图像数据的[数组](http://baike.baidu.com/view/209670.htm)。参数format指出所读象素[数据元素](http://baike.baidu.com/view/38785.htm)的格式（索引值或R、G、B、A值，如下面表所示），而参数type指出每个元素的[数据类型](http://baike.baidu.com/view/675645.htm)。具体参数如下所示：

+++++++++++[像素格式](http://baike.baidu.com/view/1220149.htm)表++++++++++++++++++++++

GL\_INDEX 单个颜色索引

GL\_RGB 先是红色分量，再是绿色分量，然后是蓝色分量

GL\_RED 单个红色分量

GL\_GREEN 单个绿色分量

GL\_BLUE 单个蓝色分量

GL\_ALPHA 单个Alpha值

GL\_LUMINANCE\_ALPHA 先是亮度分量，然后是Alpha值

GL\_STENCIL\_INDEX 单个的模板索引

GL\_DEPTH\_COMPONENT 单个深度分量

++++++++++++++++++++++++++++++++++++++++++

+++++++++++++++++++ [数据类型](http://baike.baidu.com/view/675645.htm)+++++++++++++++++++

GL\_UNSIGNED\_BYTE 无符号的8位整数

GL\_BYTE 8位整数

GL\_BITMAP 无符号的8位整数数组中的单个数位

GL\_UNSIGNED\_SHORT 无符号的16位整数

GL\_SHORT 16位整数

GL\_UNSIGNED\_INT 无符号的32位整数

GL\_INT 32位整数

GL\_FLOAT 单精度浮点数

+++++++++++++++++++++++++++++++++++++++

同时，我们知道OpenGL的绘制是从上到下的，因此我们获得像素数据后，要将数据倒序排列下；

实现代码如下：

unsigned char screenBuffer[1024 \* 1024 \* 8];

void HelloWorld::screenPlot2(bool upsidedown)

{

CCSize winSize = CCDirector::sharedDirector()->getWinSizeInPixels();

int w = winSize.width;

int h = winSize.height;

int myDataLength = w \* h \* 4;

GLubyte\* buffer = screenBuffer;

/// 获?取¨?屏¨¢幕?像?素?

glReadPixels(0, 0, w, h, GL\_RGBA, GL\_UNSIGNED\_BYTE, buffer);

CCImage\* image = new CCImage();

if(upsidedown)

{

GLubyte\* buffer2 = (GLubyte\*) malloc(myDataLength);

/// 像?素?数ºy据Y倒Ì1序¨°排?列¢D

for(int y = 0; y <h; y++)

{

for(int x = 0; x <w \* 4; x++)

{

buffer2[(h - 1 - y) \* w \* 4 + x] = buffer[y \* 4 \* w + x];

}

}

image->initWithImageData(buffer2, myDataLength, CCImage::kFmtRawData, w, h);

free(buffer2);

}

else

{

image->initWithImageData(buffer, myDataLength, CCImage::kFmtRawData, w, h);

}

image->saveToFile("12.png"); /// 将?截?屏¨¢数ºy据Y保À¡ê存ä?

image->release();

}

我们将这段代码放入update函数中，时刻运行，查看效率如下：



发现效率并不是很高。

另一个方面，cocos2d-x中，统一实现了一个渲染纹理类CCRenderTexture，其作用是将绘图设备从屏幕转移到一张纹理上，从而使得一段连续的绘图被保存到纹理中。

///@brief 截?屏¨¢

///@param[in] r0、¡ér1是º?要°a截?屏¨¢的Ì?区?域®¨°

///@return true---截?取¨?成¨¦功|，ê? false---截?取¨?失º¡ì败ã¨¹

///@author DionysosLai,906391500@qq.com

///@retval

///@post

bool HelloWorld::screenPlot1(const CCPoint r0, const CCPoint r1)

{

/// 判D断?r0 、¡ér1能¨¹否¤?组Á¨¦成¨¦一°?个?矩?形?

if (r0.x == r1.x || r0.y == r1.y)

{

return false;

}

CCSize visibleSize = CCDirector::sharedDirector()->getVisibleSize();

//根¨´据Y要°a截?取¨?屏¨¢幕?大ä¨®小?，ê?定¡§义°?一°?个?渲?染¨?纹?理¤¨ª

CCRenderTexture\* renderTexture = CCRenderTexture::create(abs(r0.x - r1.x), abs(r0.y - r1.y));

CCScene\* pCurScene = CCDirector::sharedDirector()->getRunningScene();

/\* CCScene\*/

CCPoint ancPos = pCurScene->getAnchorPoint();

//渲?染¨?纹?理¤¨ª开a始º?捕?捉Á?

renderTexture->begin();

//绘?制?当Ì¡À前¡ã场?景¡ã

pCurScene->visit();

//结¨¢束º?

renderTexture->end();

renderTexture->saveToFile("screenshoot.png");

renderTexture->setPosition(ccp(visibleSize.width/2.0f, visibleSize.height/2.0f));

this->addChild(renderTexture, 1);

// CC\_SAFE\_RELEASE(renderTexture);

return true;

}

在这里有个注意点，就是我将renderTexture释放语句给注释掉了，之所以要注释，原因在这里我们主动调用了 导演类的绘制场景功能。根据引擎的接口规范，我们不建议这样做，因为每次都产生了CCNode类的visit函数的调用，但只要遵守不再visit中更改绘图相关状态的规范，可以保证不对后续绘图产生影响。

同样，我们每帧都运行这个代码，看起效率怎样：运行是数字显示时6点多，效率更低一点。



但效率差不多。其实，这时因为在CCRenderTexture的内部实现，其到处纹理的过程实际上也是利用glReadPixels函数来获取像素信息的。因此两个方法，我们可以任意取一个，但建议用第二个。

Ps：这里的效率其实，有很大一部分，是浪费在saveToFile这条语句上。

==，有没有发现，这张图和前面的图不一样。是不是少了左下角数字。这个问题，留着下次再说吧。