**glsl的多重纹理**

纹理单元

GLint mUnits;

glGetIntegerv(GL\_MAX\_TEXTURE\_IMAGE\_UNITS, &mUnits);

cout << "opengl max texture unit = " << mUnits << endl;



在上面的例子中，你可能奇怪为什么sampler2D是个uniform变量，你却不用glUniform给它赋值，使用glUniform1i我们就可以给纹理采样器确定一个位置，这样的话我们能够一次在一个片元着色器中设置多个纹理。一个纹理的位置通常称为一个纹理单元，一个纹理的默认纹理单元是0。它是默认记过的纹理单元，所以前面部分我们不用给它确定一个位置。

纹理单元的主要目的是让我们在着色器中可以使用多于一个纹理。通过把纹理单元赋值给采样器，我们可以一次绑定多个纹理，只要我们首先记过相应的纹理单元。就像glBindTexture一样，我们可以使用glActiveTexture激活纹理单元，传入我们需要使用的纹理单元：

glActiveTexture(GL\_TEXTURE0); // 在绑定纹理之前，先激活纹理单元

glBindTexture(GL\_TEXTURE\_2D, texture);

激活纹理单元之后，接下来glBindTexture调用函数，会绑定这个纹理但当前激活的纹理单元，纹理单元GL\_TEXTURE0总是默认被激活，所以我们在前面的例子中使用glBindTexture时，无需激活任何纹理单元。

小提示：opengl至少提供16个纹理单元供我们使用，也就是说我们可以激活GL\_TEXTURE0到GL\_TEXTURE15.它们都是顺序定义的，所以我们也可以通过GL\_TEXTURE0+8的方式获得GL\_TEXTURE8，这个例子在当我们不得不循环几个纹理的时候变得很有用

小例子：（第一张纹理整张图片，第二张纹理的0.3-0.8中间的正方形混合）

glActiveTexture(GL\_TEXTURE0); // 激活0号纹理单元

glUniform1i(glProgram->getUniformLocation("our\_texture1"), 0); // 告诉shader在0号纹理单元中的纹理进行采样

glBindTexture(GL\_TEXTURE\_2D, textureId1); // 将textureId1绑定到0号纹理单元

glActiveTexture(GL\_TEXTURE1);

glUniform1i(glProgram->getUniformLocation("our\_texture2"), 1);

glBindTexture(GL\_TEXTURE\_2D, textureId2);

glEnableVertexAttribArray(1);

glEnableVertexAttribArray(2);

glEnableVertexAttribArray(3);

glEnableVertexAttribArray(4);

glVertexAttribPointer(1, 4, GL\_FLOAT, GL\_FALSE, stride, NULL);

glVertexAttribPointer(2, 4, GL\_FLOAT, GL\_FALSE, stride, colorPointer);

glVertexAttribPointer(3, 2, GL\_FLOAT, GL\_FALSE, stride, texPointer);

glVertexAttribPointer(4, 2, GL\_FLOAT, GL\_FALSE, stride, texPointer1);

