


traceorigin的专栏

☰ 目录视图

☰ 摘要视图

RSS 订阅

个人资料



traceorigin

访问：45800次

积分：836

等级：BLOG 3

排名：千里之外

原创：35篇

转载：6篇

译文：0篇

评论：24条

文章搜索

文章分类

Linux (8)

OpenGL (9)

C/C++ (3)

CUDA (4)

effective c++学习笔记 (7)

PBRT (1)

算法 (7)

recommend system (1)

optimization algorithm (1)

machine learning (2)

文章存档

2014年06月 (4)

2013年07月 (6)

2013年06月 (3)

2013年05月 (5)

2013年04月 (4)

展开

阅读排行

centos6.2安装codeblock (4157)

VS2010下如何配置CUDA (4060)

C语言的外部变量 (3267)

opengl shader 使用札记 (2524)

CSDN Android客户端发布 扒一扒最NB的开发项目 他们都已提交，就差你了！ CSDN博主维权信息收集 最流行的语言都在这里，想学就学！

opengl shader 使用札记

分类：OpenGL

2013-07-03 20:37

2526人阅读

评论(0)

收藏

举报

目录(?)

一、shader的使

创建shader

1、创建一个shader对象

[cpp]

01. GLuint glCreateShader(GLenum shaderType);

2、将shader源代码传入前面创建的shader对象

[cpp]

01. void glShaderSource(GLuint shader, int numOfStrings, const char **strings, int *lenOfStrings);

3、编译shader

[cpp]

01. void glCompileShader(GLuint shader);

使用shader

1、创建一个对象，作为程序的容器

[cpp]

01. GLuint glCreateProgram(void);

2、编译的shader附加到刚刚创建的程序中

[cpp]

01. void glAttachShader(GLuint program, GLuint shader);

3、连接程序

[cpp]

01. void glLinkProgram(GLuint program);

4、使用程序

[cpp]

01. void glUseProgram(GLuint prog);

http://blog.csdn.net/traceorigin/article/details/9237291

1/6

- CUDA使用纹理内存 (2289)
- 你到底占多大内存? ---字 (2139)
- centos6.2下的codeblock (2045)
- cuda与opengl互操作之V (1815)
- 虚拟机上的Red Hat安装 (1673)
- cuda与opengl互操作之P (1604)

评论排行

- VS2010下如何配置CUDA, (7)
- centos6.2安装codeblock (6)
- CUDA使用纹理内存 (5)
- PBRT-v2在windows下的 (3)
- 斐波那契数列 (1)
- cuda与opengl互操作之V (1)
- C语言的外部变量 (1)
- 朴素贝叶斯分类器 (0)
- KMP算法 (0)
- 你到底占多大内存? ---字 (0)

推荐文章

- * 2015博文大赛精彩文章
- *为什么我说Rust是靠谱的编程语言
- *Android UI常用实例 如何实现欢迎界面 (Splash Screen)
- *Android应用层View绘制流程与源码分析
- *Android屏幕适配全攻略
- *一个多月来的面试总结(阿里, 网

最新评论

- C语言的外部变量
MingL_Wang: 非常明白,谢谢
- VS2010下如何配置CUDA4.2
骆驼刺-王兵: 我看了你的文章之后终于弄好了, 谢谢您啊~~
- PBRT-v2在windows下的配置与f
traceorigin: @tcx19900712:应该是你的环境变量没配置好, 参见步骤3
- PBRT-v2在windows下的配置与f
天才霄: 为什么说"pbrt 不是内部命令也不是可运行的程序"呢
- CUDA使用纹理内存
traceorigin: @zhou1220317614:是不是cuda版本不对, 建议你参照你自己电脑上的SDK例子, 上面的程...
- CUDA使用纹理内存
zhou1220317614: 但结果是前八个数是0, 后6个数是很长的数, 请问怎么回事呢? ? 调了很久了不知道怎么回事, 麻烦你了。
- CUDA使用纹理内存
zhou1220317614: 是这么访问纹理内存的。
b=tex3D(texRef3D,0,0,0);
b=tex3D(texRef...
- CUDA使用纹理内存
zhou1220317614: 你好, 我是按照你说的写的, 但一直不对。代码: cudaChannelFormatDesc ...
- PBRT-v2在windows下的配置与f
IronYoung: 额, 如果我的电脑是win7 64bit的, 那我在VS2010中应该选择release还是debug? ...
- VS2010下如何配置CUDA4.2

删除shader

```
[cpp]
01. void glDeleteShader(GLuint id);
02. void glDeleteProgram(GLuint id);
```

更加详细内容参见
<http://blog.csdn.net/racehorse/article/details/6616256>

二、shader数据类型

- 1、attribute variables
随不同顶点变化的全局变量, 也即可以针对每个顶点设置该变量。
注: 个修饰符只能用在顶点shader中, 在shader中它是一个只读变量。
- 2、uniform
随不同图元变化的全局变量, 也即不能对每个顶点设置该变量。
注: 其值不能在glBegin/glEnd中设置。一致变量适合描述在一个图元中、一帧中甚至一个场景中都不变的值。一致变量在顶点shader和片断shader中都是只读的。
- 3、varying
用于顶点shader和片断shader间传递的插值数据, 在顶点shader中可写, 在片断shader中只读
注: 须同时在顶点shader和片断shader中声明

三、shader数据传输

(一)、glsl 2.x数据传输:

1、attribute变量传输

```
[cpp]
01. GLint glGetAttribLocation(GLuint program, char *name);
02. void glVertexAttrib1f(GLint location, GLfloat v0);
03. void glVertexAttrib2f(GLint location, GLfloat v0, GLfloat v1);
04. void glVertexAttrib3f(GLint location, GLfloat v0, GLfloat v1, GLfloat v2);
05. void glVertexAttrib4f(GLint location, GLfloat v0, GLfloat v1, GLfloat v2, GLfloat v3);
```

2、uniform变量传输

```
[cpp]
01. GLint glGetUniformLocation(GLuint program, const char *name);
02. void glUniform1f(GLint location, GLfloat v0);
03. void glUniform2f(GLint location, GLfloat v0, GLfloat v1);
04. void glUniform3f(GLint location, GLfloat v0, GLfloat v1, GLfloat v2);
05. void glUniform4f(GLint location, GLfloat v0, GLfloat v1, GLfloat v2, GLfloat v3);
```

注: getLocation要在program连接之后, 因为链接后, 会自动为它分配一个位置, 我们可以在任何需要的时候获取 (查询) 这个位置。

3、varying变量传输

```
[cpp]
01. //Vertex Shader
02. out vec2 varying_vg_texcoord;
03.
04. //Geometry Shader
05. in vec2 varying_vg_texcoord[];
06. out vec2 varying_gf_texcoord;
07.
08. //Fragment Shader
09. in vec2 varying_gf_texcoord;
```

luofl_: 新的CUDA版本很简单，直接可以新建CUDA项目了。我也实验过4.2版本的，怎么也可以直接新建。

(二)、glsl 3.x数据传输

在GL3.x中，废弃了attribute关键字以及varying关键字，属性变量统一用in/out作为前置关键字。

对每一个Shader stage来说，in表示该属性是作为输入的属性，out表示该属性是用于输出的属性。

1、attribute变量传输

直接在GLSL代码中指定这些位置值

```
[cpp]
01. #version 330
02. layout(location = 0) in vec3 attrib_position;
```

这里使用了layout关键字。这个关键字用于一个具体变量前，用于显式标明该变量的一些布局属性，这里就是显式设定了该attribute变量的位置值(location)。

优点：

- (1)、避免了getlocation的开销；
- (2)、重定义了OpenGL和GLSL之间attribute变量属性的依赖。过去我们的OpenGL端必须首先要知道GLSL端某个attribute的名字，才能设置/获得其位置值，如今两者只需要location对应起来就可以完成绘制时顶点属性流的传递了

2、uniform变量传输

2.1 UBO

在一个复杂的shader中，uniform变量可能会很多，用glUniform()来传递数据会显得很臃肿；

另外，一个shader中uniform变量的数目是有上限的，所以不能传入太多的uniform 变量。

鉴于以上原因，提出了Uniform Buffer Object(UBO)，这样可以把一个或多个uniform变量交给它，以替代glUniform的方式传递数据。

注意：传给UBO的数据，存储于这个UBO上，而不再是交给ShaderProgram，所以它们不会占用这个ShaderProgram自身的uniform存储空间。

2.2 uniform block

在uniform关键字后直接跟随一个block name和大括号，里面是一个或多个uniform变量，用于与UBO关联。

```
[cpp]
01. uniform BlobSettings {
02.     vec4 InnerColor;
03.     vec4 OuterColor;
04.     float RadiusInner;
05.     float RadiusOuter;
06. };
07.
08. uniform BlobSettings{
09.     vec4 InnerColor;
10.     vec4 OuterColor;
11.     float RadiusInner;
12.     float RadiusOuter;
13. }Blob;
```

在方法一中，Block中的变量依然是全局域的一部分，取用的时候不用Block名。

而在方法二中，Block中的变量则是在Blob名字域中，取用的时候需要加上Block名

2.3关联

对于每一个uniform block，都有一个“索引值”(index)，这个索引值我们可以在OpenGL中获得，并把它与一个具体的UBO关联起来

```
[cpp]
```

```

01. GLint blockIndex = glGetUniformLocation(nProgramHandler, "BlobSettings");
02.
03. GLint nBlockDataSize = 0;
04.
05. glGetActiveUniformBlockiv(nProgramHandler, blockIndex, GL_UNIFORM_BLOCK_DATA_SIZE, &nBlockDataSize);
06.
07. glGenBuffers(1, &m_nUBO);
08.
09. glBindBuffer(GL_UNIFORM_BUFFER, m_nUBO);
10.
11. glBufferData(GL_UNIFORM_BUFFER, nBlockDataSize, NULL, GL_DYNAMIC_DRAW);
12.
13. glBindBufferRange(GL_UNIFORM_BUFFER, 0, m_nUBO, 0, nBlockDataSize);
14.
15. glUniformBlockBinding(nProgramHandler, blockIndex, 0);
16.
17. glBindBuffer(GL_UNIFORM_BUFFER, NULL);

```

2.4 采用Block的原因

如果你的程序中包含了多个着色器，而且这些着色器使用了相同的Uniform变量，你就不得不为每个着色器分别管理这些变量。Uniform变量的location是在程序链接的时候产生的，因此Uniform变量的location会随着着色器的不同而发生变化。因此，这些Uniform变量的数据必须重新产生，然后应用到新的location上。

而Uniform Block正是为了使在着色器间共享Uniform数据变得更加容易而设计的。有了Uniform Block，我们可以创建一个缓冲区对象来存储这些Uniform变量的值，然后将缓冲区对象绑定到Uniform Block。当着色器程序改变的时候，只需要将同样的缓冲区对象重新绑定到在新的着色器中与之相关的Block即可。

3、varying变量传输

in block和out block这两种可用于varying变量的

注意这里使用了block instance name（紧随大括号后的那个名字），这个名字对各Shader Stage来说都是独特的，所以改成上面这样的话，

block之间也不会发生名字冲突，block内的varying变量也就可以用同一个名字了。使用时需要按"blockInstanceName.varyingVariable"的类似结构体内变量的样式来表示。

```

[cpp]
01. //Vertex Shader
02. out Varying
03. {
04.     vec2 texcoord;
05. }VaryingOut;
06.
07. //Geometry Shader
08. in Varying
09. {
10.     vec2 texcoord;
11. }VaryingIn[];
12.
13. out Varying
14. {
15.     vec2 texcoord;
16. }VaryingOut;
17.
18. //Fragment Shader
19. in Varying
20. {
21.     vec2 texcoord;
22. }VaryingIn;

```

4、fragmentShader输出

类似于上述的attribute变量，我们也可以直接通过layout来指定这个location值

```

[cpp]
01. /单输出
02. layout(location = 0) out vec4 fragColor;
03.

```

```
04. //MRT
05. layout(location = 0) out vec4 fragColor0;
06. layout(location = 1) out vec4 fragColor1;
```

这些layout里的关键字其实还有个index——只是默认为0而已

```
[cpp]
01. layout(location = 0, index = 0) out vec4 fragColor;
02. layout(location = 0, index = 1) out vec4 src1Color;
```

参考资料:

<http://blog.csdn.net/xiajun07061225/article/details/7709815>
<http://www.zwqxin.com/archives/shaderglsl/communication-between-opengl-glsl-2.html>

上一篇 shadow projection
下一篇 shadow map

顶 0 踩 0

主题推荐 opengl 全局变量 源代码 对象 存储

猜你在找

- GLSL教程九其他说明
- OpenGL 概念基础和shader例子
- cocos2d对动画的各种操作
- 用参数方程绘制椭球体
- 何为P2P技术发展前景如何
- 【精品课程】太空大战游戏实战课程
- 【精品课程】U3D实时阴影绘制及Shader解决方案
- 【精品课程】三维游戏引擎开发-渲染
- 【精品课程】VC++游戏开发基础系列从入门到精通
- 【精品课程】iOS8开发技术（Swift版）：iOS基础知识

准备好了么? 跳 吧! 更多职位尽在 CSDN JOB

高级软件工程师	我要跳槽	Android工程师	我要跳槽
北京鑫源联创科技有限责任公司	7-8K/月	北京鑫源联创科技有限责任公司	10-14K/月
iOS 工程师	我要跳槽	Java工程师	我要跳槽
北京鑫源联创科技有限责任公司	8-10K/月	北京鑫源联创科技有限责任公司	8-10K/月

Managed Cloud for Singapore

Cloud migration, security and management services, backed by our 24/7 Global Response Team.

amazon web services

Microsoft Azure

vmware

DATAPIPE

Schedule Consult

查看评论
暂无评论

您还没有登录,请[登录](#)或[注册](#)

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack
VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery