

dreamcs的专栏

目录视图

摘要视图

RSS 订阅

个人资料



dreamcs

访问： 152651次
积分： 3002
等级： BLOG 5
排名： 第5295名

原创： 131篇 转载： 29篇
译文： 8篇 评论： 42条

文章分类

- C++ (45)
- C++内存管理 (6)
- note (4)
- WINAPI_MFC (21)
- 专题 (2)
- 工具使用 (4)
- 数据结构与算法 (14)
- 汇编 (3)
- 计算机图形学 (29)
- 软件工程 (9)
- 阶段总结 (13)
- OCC (1)
- QT (4)
- boost (6)
- node.js (2)
- 多线程 (9)
- linux c (7)
- 网络编程 (2)
- Android (2)

文章存档

- 2015年02月 (1)
- 2015年01月 (1)
- 2014年10月 (2)
- 2014年01月 (1)
- 2013年12月 (1)

展开

阅读排行

hello world 级别 MFC自写 (8060)

CSDN Android客户端发布 扒一扒最NB的开发项目 CSDN博主维权信息收集 最流行的语言都在这，想学就学！

OpenGL像素缓冲区对象

分类： 计算机图形学 2012-07-02 11:56 2535人阅读 评论(0) 收藏 举报

buffer asynchronous object extension stream textures

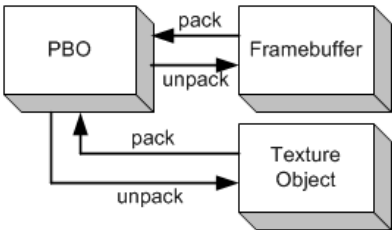
OpenGL像素缓冲区对象

原贴地址

目录

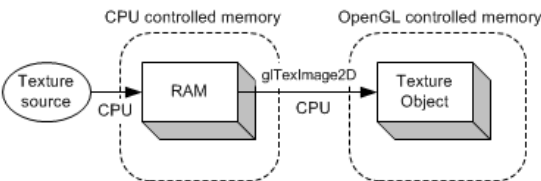
- 概述
- 创建PBO
- 映射PBO
- 例子:Streaming Texture Uploads with PBO
- 例子:Asynchronous Readback with PBO

概述



OpenGL ARB_pixel_buffer_object 扩展与ARB_vertex_buffer_object很相似。为了把像素数据你储存在缓冲区对象中，而非顶点数据，它简单地扩展了 ARB_vertex_buffer_object extension。储存像素数据的缓冲区对象称为Pixel Buffer Object (PBO)。ARB_pixel_buffer_object extension借用了VBO所有的架构及API，但多了两个"target" 标签。target协助PBO储存管理器(OpenGL驱动)决定缓冲区对象的最佳位置： system 内存, AGP (共享内存)或显卡内存。Target标志指定PBO的两种不同的操作：GL_PIXEL_PACK_BUFFER_ARB 传递像素数据到PBO中。或GL_PIXEL_UNPACK_BUFFER_ARB 从PBO中传回数据。

例如，glReadPixels()和glGetTexImage()是"pack"像素操作， glDrawPixels(), glTexImage2D() , glTexSubImage2D() 是"unpack" 操作。当一个PBO的标志为GL_PIXEL_PACK_BUFFER_ARB, glReadPixels()从OpenGL的一个帧缓冲区读取数据，并将数据写(pack)入PBO中。当一个PBO的标志为GL_PIXEL_UNPACK_BUFFER_ARB时, glDrawPixels()从PBO读取(unpack)像素数据并复制到OpenGL帧缓冲区中。PBO的主要优势是可以快速地传递像素数据通过显卡的DMA (Direct Memory Access) 而涉及CPU循环。另一个优势是它有异步DMA传输。让我们对比使用PBO后的纹理传送方法。左侧图是从图像文件或视频中加载纹理。首先，资源被加载到系统内存中，然后使用glTexImage2D()函数从系统内存复制到OpenGL纹理对象中。这两次数据传输(加载和复制)完全由CPU执行。



不用PBO中加载纹理

- B样条曲线 (6470)
- OpenGL顶点缓冲区对象 (4371)
- MFCPropertyGridCtrl分布 (3767)
- QSplitter 学习 (3034)
- vc cl编译器使用 (3032)
- 三次样条插值 (2851)
- 二叉树及二叉搜索树 (2711)
- OpenGL帧缓存对象(FBC (2585)
- OpenGL像素缓冲区对象 (2533)

最新评论

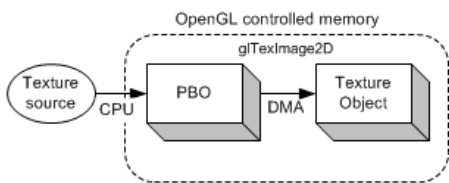
- 带洞多边形三角剖分发图留念
zwvenjakob: 不给源码, 不介绍算法, 你发上来个结果干什么?
- 从图片加载纹理-使用glut工具
dreamcs: @u012699651: 图片保存在文件中啊。
- 从图片加载纹理-使用glut工具
卡卡77: 图片存在哪里啊?
- 使用OpenGL画三次Bezier曲线
qq_26379109: 我的怎么没有画出来呢?
- 使用OpenGL画三次Bezier曲线
zhouqing001007: 很好, 很给力, 稍微修改, 可以修改为七个控制点的bezier曲线, (因为我的作业是给出七个控制点, 两...
- QSplitter 学习
边城菜鸟: 今天抢的第一个沙发。谢谢。
- 使用OpenGL画三次Bezier曲线
小柒柒123: 太好啦
- 三次样条插值
polluxli: 如何实现啊? 大神
- 三次样条插值
polluxli: 二维数组Array i代表二维平面x坐标, j代表二维平面y坐标。数组中有些元素值为null, 使用S...
- MFCPropertyGridCtrl分析
miluzhiyu: 请问如果我想实现点击value编辑框就打开一个对话框用来输入, 输入完之后再将值送入value该怎么实...

高手blog链接

- 大坡3D软件开发blog
- 李先静
- qt_blog
- qt_blog2
- 陈皓
- 3D引擎博客
- v_JULY_v
- 分形
- 陈硕
- 陈梓瀚
- KlayGE游戏引擎
- 风云的blog
- 张传波软件工程
- 魔兽之父blog
- Matrix67

网络资源

- 陆新征
- OnlineMath
- mathworld
- 网易翻译的公开课



从PBO中加载纹理

右侧图中, 图像可直接加载到PBO中。CPU加载资源到PBO中, 但不用从PBO传递像素信息到纹理对象中。GPU (OpenGL driver)从一个PBO复制数据到一个纹理对象中。OpenGL使用了DMA转送操作, 而没有等CPU循环。OpenGL使用了异步的DMA传输方法。因此, glTexImage2D()立刻返回, CPU可以做其它事, 而不用等待像素传送完毕。

下面有两个主要的PBO方法, 用来提升像素传送性能: streaming texture update 和 asynchronous read-back from the framebuffer.

创建PBO

以前说到, Pixel Buffer Object使用VBO的所有API。不同点仅再两个针对PBO的额外标志: GL_PIXEL_PACK_BUFFER_ARB和GL_PIXEL_UNPACK_BUFFER_ARB。GL_PIXEL_PACK_BUFFER_ARB 从OpenGL传送像素数据到你的程序中, GL_PIXEL_UNPACK_BUFFER_ARB 将像素数据从程序传送到OpenGL中。OpenGL使用这些标志, 来决定PBO最付款的内存位置。如, uploading(unpack)时, 使用显卡内存。读帧缓冲区时, 使用系统内存。然而, 此target标志是唯一参照。OpenGL 驱动决定PBO的位置。

- 创建一个PBO需要三个步骤:
- 1使用glGenBuffersARB()新建一个缓冲区对象
- 2使用glBindBufferARB()绑定一个缓冲区对象
- 3使用glBufferDataARB复制像素信息到缓冲区对象

如果你使用了一个空指针, 指向glBufferDataARB的数据源数组, 那么PBO只按数据的大小分配内存空间。glBufferDataARB最后一个参数是PBO的一个提示信息, 它告诉如何使用些缓冲区对象。GL_STREAM_DRAW_ARB 是 streaming texture upload 。 GL_STREAM_READ_ARB 是异步的帧缓冲区read-back。更多细节, 参见VBO。

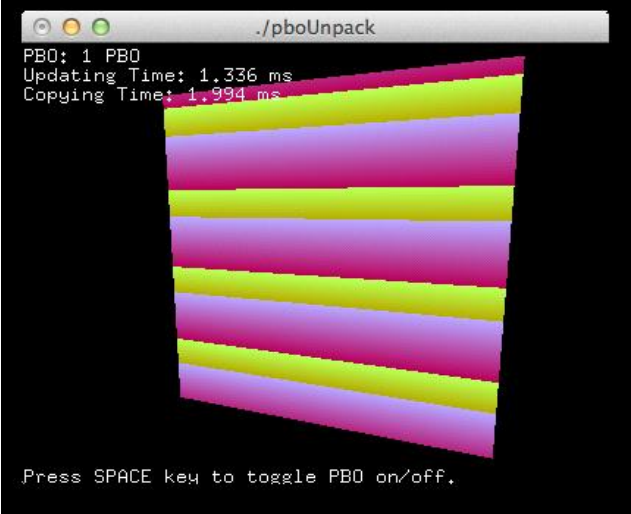
PBO映射

PBO提供了内存映射机制, 映射OpenGL控制的缓冲区对象到客户端的内存地址空间中。客户端可以使用glMapBufferARB(), glUnmapBufferARB()函数修改部分缓冲区对象或全部。
void* glMapBufferARB(GLenum target, GLenum access)
GLboolean glUnmapBufferARB(GLenum target)

glMapBufferARB()返回一个指针, 指向缓冲区对象, 如果成功返回此指针, 否则返回NULL。Target参数是GL_PIXEL_PACK_BUFFER_ARB 或GL_PIXEL_UNPACK_BUFFER_ARB。第二个参数, 指定映射的缓冲区的访问方式: 从BPO中读数据(GL_READ_ONLY_ARB), 写数据到BPO中(GL_WRITE_ONLY_ARB) 或(GL_READ_WRITE_ARB)。

注意如果GPU仍使用此缓冲区对象, glMapBufferARB()不会返回, 直到GPU完成了对对应缓冲区对象的操作。为了避免等待, 在使用glMapBufferARB之前, 使用glBufferDataARB, 并传入参数NULL。然后, OpenGL将废弃旧的缓冲区, 为缓冲区分配新的内存。缓冲区对象必须取消映射, 可使用glUnmapBufferARB()。如果成功, glUnmapBufferARB()返回GL_TRUE 否则返回GL_FALSE。

例子:Streaming Texture Uploads



源码下载: [pboUnpack.zip](#).

这个例子使用PBO, 上传(uppack)streaming textures到OpenGL texture object using PBO.你可以切换不

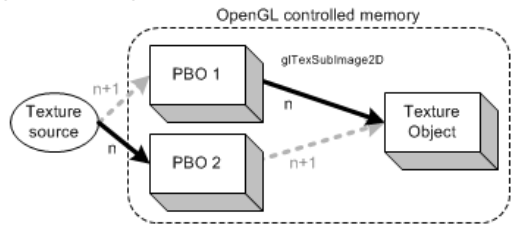
新浪翻译的公开课
非常棒的PPT搜索引擎
开源代码搜索引擎
痞子龙
实时渲染资源
软件知识原则基地
CADCAE博客
Bentley
abaqus
DASSAULT PLM
非常好的cpp网站
很好的Web前端开发网站
科学网
仿真论坛
鲁班人
华丽联合轻钢抗震房屋
中国建筑新闻网

建筑IT网络资源

zc
北京凯特顺力科技有限公司
cuteser
结构笔记
陈以一钢结点
崔家春
结构构件在线计算

两只的传送模式：单个PBO，两个PBOs，无PBO)。对比它们之间效率的差异。

在BPO模式下，在每帧中，通过映射的PBO指接写入纹理。然后，这些纹理数据使用glTexSubImage2D()函数，从PBO传送到纹理对象中。使用PBO，OpenGL可以在PBO和纹理对象之间执行异步DMA传输。这显著提长了纹理重新载入的性能。如果显卡支持异步的DMA操作，glTexSubImage2D()会立即返回。CPU无需等待纹理拷贝，便可以做其它事情。

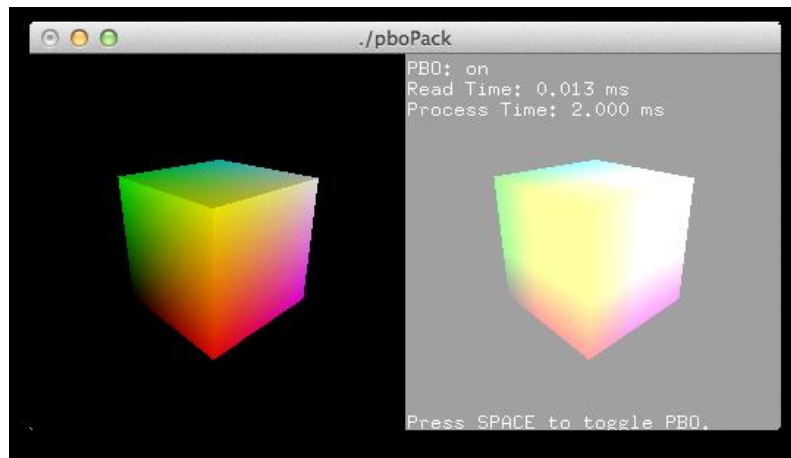


Streaming texture uploads with 2 PBOs

为了最大化提升streaming传输性能，你可以使用多个PBO。图中显示出两个PBO正被同时使用。glTexSubImage2D()从一个PBO复制数据，当纹理被写入另一个PBO时。在第n帧时，PBO1被glTexSubImage2D()函数使用。PBO2用于得到新的纹理。在第n+1帧时，2个PBO交换角色，并继续更新纹理。因为DMA传输的异步性，更新和复制可被同时执行。CPU更新纹理到一个PBO中，当GPU从另一个PBO中复制纹理时。

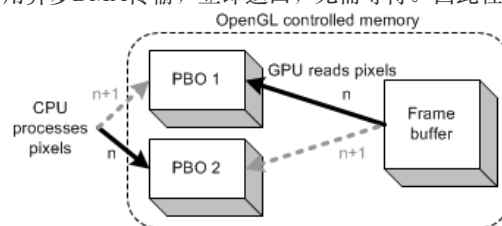
```
[cpp]
01. // "index" is used to copy pixels from a PBO to a texture object
02. // "nextIndex" is used to update pixels in the other PBO
03. index = (index + 1) % 2;
04. nextIndex = (index + 1) % 2;
05.
06. // bind the texture and PBO
07. glBindTexture(GL_TEXTURE_2D, textureId);
08. glBindBufferARB(GL_PIXEL_UNPACK_BUFFER_ARB, pboIds[index]);
09.
10. // copy pixels from PBO to texture object
11. // Use offset instead of pointer.
12. glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, WIDTH, HEIGHT,
13.                 GL_BGRA, GL_UNSIGNED_BYTE, 0);
14.
15.
16. // bind PBO to update texture source
17. glBindBufferARB(GL_PIXEL_UNPACK_BUFFER_ARB, pboIds[nextIndex]);
18.
19. // Note that glMapBufferARB() causes sync issue.
20. // If GPU is working with this buffer, glMapBufferARB() will wait(stall)
21. // until GPU to finish its job. To avoid waiting (idle), you can call
22. // first glBufferDataARB() with NULL pointer before glMapBufferARB().
23. // If you do that, the previous data in PBO will be discarded and
24. // glMapBufferARB() returns a new allocated pointer immediately
25. // even if GPU is still working with the previous data.
26. glBufferDataARB(GL_PIXEL_UNPACK_BUFFER_ARB, DATA_SIZE, 0, GL_STREAM_DRAW_ARB);
27.
28. // map the buffer object into client's memory
29. GLubyte* ptr = (GLubyte*)glMapBufferARB(GL_PIXEL_UNPACK_BUFFER_ARB,
30.                                           GL_WRITE_ONLY_ARB);
31. if(ptr)
32. {
33.     // update data directly on the mapped buffer
34.     updatePixels(ptr, DATA_SIZE);
35.     glUnmapBufferARB(GL_PIXEL_UNPACK_BUFFER_ARB); // release the mapped buffer
36. }
37.
38. // it is good idea to release PBOs with ID 0 after use.
39. // Once bound with 0, all pixel operations are back to normal ways.
40. glBindBufferARB(GL_PIXEL_UNPACK_BUFFER_ARB, 0);
```

例子：异步Read-back



下载源码: [pboPack.zip](#).

这个例子从帧缓冲区(左侧图)读取像素数据到PBO中,之后在右侧窗体中画出来,并修改图像的亮度。你可以按空格键打开或关闭PBO,来测试glReadPixels()函数的性能差异。传统的glReadPixels()阻塞渲染管线,直到所有的像素传输完毕。然后,它把控制权交给程序。使用PBO的glReadPixels()可使用异步DMA传输,立即返回,无需等待。因此程序(CPU)可执行其它操作,当GPU传输数据时。



Asynchronous glReadPixels() with 2 PBOs

此例子使用2个PBO。在第n帧时,程序使用glReadPixels()从OpenGL读取像素信息到PBO1中,在PBO2中处理像素数据。读数据和处理数据是同时进行的。因为glReadPixels()在PBO1上立即返回,CPU可以在PBO2中处理数据而没有延迟。我们在每一帧中交换PBO1和PBO2。

```
[cpp]
01. // "index" is used to read pixels from framebuffer to a PBO
02. // "nextIndex" is used to update pixels in the other PBO
03. index = (index + 1) % 2;
04. nextIndex = (index + 1) % 2;
05.
06. // set the target framebuffer to read
07. glReadBuffer(GL_FRONT);
08.
09. // read pixels from framebuffer to PBO
10. // glReadPixels() should return immediately.
11. glBindBufferARB(GL_PIXEL_PACK_BUFFER_ARB, pboIds[index]);
12. glReadPixels(0, 0, WIDTH, HEIGHT, GL_BGRA, GL_UNSIGNED_BYTE, 0);
13.
14. // map the PBO to process its data by CPU
15. glBindBufferARB(GL_PIXEL_PACK_BUFFER_ARB, pboIds[nextIndex]);
16. GLubyte* ptr = (GLubyte*)glMapBufferARB(GL_PIXEL_PACK_BUFFER_ARB,
17.                                         GL_READ_ONLY_ARB);
18. if(ptr)
19. {
20.     processPixels(ptr, ...);
21.     glUnmapBufferARB(GL_PIXEL_PACK_BUFFER_ARB);
22. }
23.
24. // back to conventional pixel operation
25. glBindBufferARB(GL_PIXEL_PACK_BUFFER_ARB, 0);
```

上一篇 B样条曲线

下一篇 Ogre日志