高级数据

原文	**Advanced Data (http://learnopengl.com/#!Advanced-OpenGL/Advanced-Data)**
作者	JoeyDeVries
翻译	Meow J
校对	暂未校对

我们在OpenGL中大量使用缓冲来储存数据已经有很长时间了。操作缓冲其实还有更有意思的方式，而且使用纹理将大量数据传入着色器也有更有趣的方法。这一节中，我们将讨论一些更有意思的缓冲函数，以及我们该如何使用纹理对象来储存大量的数据（纹理的部分还没有完成）。

OpenGL中的缓冲只是一个管理特定内存块的对象，没有其它更多的功能了。在我们将它绑定到一个缓冲目标(Buffer Target)时，我们才赋予了其意义。当我们绑定一个缓冲到`GL_ARRAY_BUFFER`时，它就是一个顶点数组缓冲，但我们也可以很容易地将其绑定到`GL_ELEMENT_ARRAY_BUFFER`。OpenGL内部会为每个目标储存一个缓冲，并且会根据目标的不同，以不同的方式处理缓冲。

到目前为止，我们一直是调用`glBufferData`函数来填充缓冲对象所管理的内存，这个函数会分配一块内存，并将数据添加到这块内存中。如果我们将它的`data`参数设置为`NULL`，那么这个函数将只会分配内存，但不进行填充。这在我们需要**预留**(Reserve)特定大小的内存，之后回到这个缓冲一点一点填充的时候会很有用。

除了使用一次函数调用填充整个缓冲之外，我们也可以使用`glBufferSubData`，填充缓冲的特定区域。这个函数需要一个缓冲目标、一个偏移量、数据的大小和数据本身作为它的参数。这个函数不同的地方在于，我们可以提供一个偏移量，指定从**何处**开始填充这个缓冲。这能够让我们插入或者更新缓冲内存的某一部分。要注意的是，缓冲需要有足够的已分配内存，所以对一个缓冲调用`glBufferSubData`之前必须要先调用`glBufferData`。

```
glBufferSubData(GL_ARRAY_BUFFER, 24, sizeof(data), &data); // 范围：[24, 24 + sizeof(data)]
```

将数据导入缓冲的另外一种方法是，请求缓冲内存的指针，直接将数据复制到缓冲当中。通过调用`glMapBuffer`函数，OpenGL会返回当前绑定缓冲的内存指针，供我们操作：

```
float data[] = {
  0.5f, 1.0f, -0.35f
  ...
};
glBindBuffer(GL_ARRAY_BUFFER, buffer);
// 获取指针
void *ptr = glMapBuffer(GL_ARRAY_BUFFER, GL_WRITE_ONLY);
// 复制数据到内存
memcpy(ptr, data, sizeof(data));
// 记得告诉OpenGL我们不再需要这个指针了
glUnmapBuffer(GL_ARRAY_BUFFER);
```

当我们使用`glUnmapBuffer`函数，告诉OpenGL我们已经完成指针操作之后，OpenGL就会知道你已经完成了。在解除映射(Unmapping)之后，指针将会不再可用，并且如果OpenGL能够成功将您的数据映射到缓冲中，这个函数将会返回`GL_TRUE`。

如果要直接映射数据到缓冲，而不事先将其存储到临时内存中，`glMapBuffer`这个函数会很有用。比如说，你可以从文件中读取数据，并直接将它们复制到缓冲内存中。

分批顶点属性

通过使用`glVertexAttribPointer`，我们能够指定顶点数组缓冲内容的属性布局。在顶点数组缓冲中，我们对属性进行了交错(Interleave)处理，也就是说，我们将每一个顶点的位置、发现和/或纹理坐标紧密放置在一起。既然我们现在已经对缓冲有了更多的了解，我们可以采取另一种方式。

我们可以做的是，将每一种属性类型的向量数据打包(Batch)为一个大的区块，而不是对它们进行交错储存。与交错布局123123123123不同，我们将采用分批(Batched)的方式111122223333。

当从文件中加载顶点数据的时候，你通常获取到的是一个位置数组、一个法线数组和/或一个纹理坐标数组。我们需要花点力气才能将这些数组转化为一个大的交错数据数组。使用分批的方式会是更简单的解决方案，我们可以很容易使用`glBufferSubData`函数实现：

```
float positions[] = { ... };
float normals[] = { ... };
float tex[] = { ... };
// 填充缓冲
glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(positions), &positions);
glBufferSubData(GL_ARRAY_BUFFER, sizeof(positions), sizeof(normals), &normals);
glBufferSubData(GL_ARRAY_BUFFER, sizeof(positions) + sizeof(normals), sizeof(tex), &t
ex);
```

这样子我们就能直接将属性数组作为一个整体传递给缓冲，而不需要事先处理它们了。我们仍可以将它们合并为一个大的数组，再使用glBufferData来填充缓冲，但对于这种工作，使用glBufferSubData会更合适一点。

我们还需要更新顶点属性指针来反映这些改变：

```
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), 0);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)(sizeof(pos
itions)));
glVertexAttribPointer(
  2, 2, GL_FLOAT, GL_FALSE, 2 * sizeof(float), (void*)(sizeof(positions) + sizeof(nor
mals)));
```

注意`stride`参数等于顶点属性的大小，因为下一个顶点属性向量能在3个（或2个）分量之后找到。

这给了我们设置顶点属性的另一种方法。使用哪种方法都不会对OpenGL有什么立刻的好处，它只是设置顶点属性的一种更整洁的方式。具体使用的方法将完全取决于你的喜好与程序类型。

复制缓冲

当你的缓冲已经填充好数据之后，你可能会想与其它的缓冲共享其中的数据，或者想要将缓冲的内容复制到另一个缓冲当中。glCopyBufferSubData能够让我们相对容易地从一个缓冲中复制数据到另一个缓冲中。这个函数的原型如下：

```
void glCopyBufferSubData(GLenum readtarget, GLenum writetarget, GLintptr readoffset,
                         GLintptr writeoffset, GLsizeiptr size);
```

`readtarget`和`writetarget`参数需要填入复制源和复制目标的缓冲目标。比如说，我们可以将VERTEX_ARRAY_BUFFER缓冲复制到VERTEX_ELEMENT_ARRAY_BUFFER缓冲，分别将这些缓冲目标设置为读和写的目标。当前绑定到这些缓冲目标的缓冲将会被影响到。

但如果我们想读写数据的两个不同缓冲都为顶点数组缓冲该怎么办呢？我们不能同时将两个缓冲绑定到同一个缓冲目标上。正是出于这个原因，OpenGL提供给我们另外两个缓冲目标，叫做GL_COPY_READ_BUFFER和GL_COPY_WRITE_BUFFER。我们接下来就可以将需要的缓冲绑定到这两个缓冲目标上，并将这两个目标作为`readtarget`和`writetarget`参数。

接下来glCopyBufferSubData会从`readtarget`中读取`size`大小的数据，并将其写入`writetarget`缓冲的`writeoffset`偏移量处。下面这个例子展示了如何复制两个顶点数组缓冲：

```
float vertexData[] = { ... };
glBindBuffer(GL_COPY_READ_BUFFER, vbo1);
glBindBuffer(GL_COPY_WRITE_BUFFER, vbo2);
glCopyBufferSubData(GL_COPY_READ_BUFFER, GL_COPY_WRITE_BUFFER, 0, 0, sizeof(vertexDat
a));
```

我们也可以只将`writetarget`缓冲绑定为新的缓冲目标类型之一：

```
float vertexData[] = { ... };
glBindBuffer(GL_ARRAY_BUFFER, vbo1);
glBindBuffer(GL_COPY_WRITE_BUFFER, vbo2);
glCopyBufferSubData(GL_ARRAY_BUFFER, GL_COPY_WRITE_BUFFER, 0, 0, sizeof(vertexData));
```

有了这些关于如何操作缓冲的额外知识，我们已经能够以更有意思的方式使用它们了。当你越深入OpenGL时，这些新的缓冲方法将会变得更加有用。在下一节 (../08 Advanced GLSL/)中，在我们讨论Uniform缓冲对象(Uniform Buffer Object)时，我们将会充分利用`glBufferSubData`。