OpenGL

原文	OpenGL (http://learnopengl.com/#!Getting-started/OpenGL)
作者	JoeyDeVries
翻译	gjy_1992, Meow J
校对	暂未校对

在开始这段旅程之前我们先了解一下OpenGL到底是什么。一般它被认为是一个API(Application Programming Interface, 应用程序编程接口)，包含了一系列可以操作图形、图像的函数。然而，OpenGL本身并不是一个API，它仅仅是一个由Khronos组织 (http://www.khronos.org/)制定并维护的规范(Specification)。

OpenGL规范严格规定了每个函数该如何执行，以及它们的输出值。至于内部具体每个函数是如何实现(Implement)的，将由OpenGL库的开发者自行决定（译注：这里开发者是指编写OpenGL库的人）。因为OpenGL规范并没有规定实现的细节，具体的OpenGL库允许使用不同的实现，只要其功能和结果与规范相匹配（亦即，作为用户不会感受到功能上的差异）。

实际的OpenGL库的开发者通常是显卡的生产商。你购买的显卡所支持的OpenGL版本都为这个系列的显卡专门开发的。当你使用Apple系统的时候，OpenGL库是由Apple自身维护的。在Linux下，有显卡生产商提供的OpenGL库，也有一些爱好者改编的版本。这也意味着任何时候OpenGL库表现的行为与规范规定的不一致时，基本都是库的开发者留下的bug。

> 由于OpenGL的大多数实现都是由显卡厂商编写的，当产生一个bug时通常可以通过升级显卡驱动来解决。这些驱动会包括你的显卡能支持的最新版本的OpenGL，这也是为什么总是建议你偶尔更新一下显卡驱动。

所有版本的OpenGL规范文档都被公开的寄存在Khronos那里。有兴趣的读者可以找到OpenGL3.3（我们将要使用的版本）的规范文档(https://www.opengl.org/registry/doc/glspec33.core.20100311.withchanges.pdf)。如果你想深入到OpenGL的细节（只关心函数功能的描述而不是函数的实现），这是个很好的选择。如果你想知道每个函数**具体的**运作方式，这个规范也是一个很棒的参考。

核心模式与立即渲染模式

早期的OpenGL使用立即渲染模式（Immediate mode，也就是固定渲染管线），这个模式下绘制图形很方便。OpenGL的大多数功能都被库隐藏起来，开发者很少能控制OpenGL如何进行计算的自由。而开发者迫切希望能有更多的灵活性。随着时间推移，规范越来越灵活，开发者对绘图细节有了更多的掌控。立即渲染模式确实容易使用和理解，但是效率太低。因此从OpenGL3.2开始，规范文档开始废弃立即渲染模式，并鼓励开发者在OpenGL的核心模式(Core-profile)下进行开发，这个分支的规范完全移除了旧的特性。

当使用OpenGL的核心模式时，OpenGL迫使我们使用现代的函数。当我们试图使用一个已废弃的函数时，OpenGL会抛出一个错误并终止绘图。现代函数的优势是更高的灵活性和效率，然而也更难于学习。立即渲染模式从OpenGL**实际**运作中抽象掉了很多细节，因此它在易于学习的同时，也很难让人去把握OpenGL具体是如何运作的。现代函数要求使用者真正理解OpenGL和图形编程，它有一些难度，然而提供了更多的灵活性，更高的效率，更重要的是可以更深入的理解图形编程。

这也是为什么我们的教程面向OpenGL3.3的核心模式。虽然上手更困难，但这份努力是值得的。

现今，更高版本的OpenGL已经发布（写作时最新版本为4.5），你可能会问：既然OpenGL 4.5 都出来了，为什么我们还要学习OpenGL 3.3？答案很简单，所有OpenGL的更高的版本都是在3.3的基础上，引入了额外的功能，并没有改动核心架构。新版本只是引入了一些更有效率或更有用的方式去完成同样的功能。因此，所有的概念和技术在现代OpenGL版本里都保持一致。当你的经验足够，你可以轻松使用来自更高版本OpenGL的新特性。

> 当使用新版本的OpenGL特性时，只有新一代的显卡能够支持你的应用程序。这也是为什么大多数开发者基于较低版本的OpenGL编写程序，并只提供选项启用新版本的特性。

在有些教程里你会看见更现代的特性，它们同样会以这种红色注释方式标明。

扩展

OpenGL的一大特性就是对扩展(Extension)的支持，当一个显卡公司提出一个新特性或者渲染上的大优化，通常会以扩展的方式在驱动中实现。如果一个程序在支持这个扩展的显卡上运行，开发者可以使用这个扩展提供的一些更先进更有效的图形功能。通过这种方式，开发者不必等待一个新的OpenGL规范面世，就可以使用这些新的渲染特性了，只需要简单地检查一下显卡是否支持此扩展。通常，当一个扩展非常流行或者非常有用的时候，它将最终成为未来的OpenGL规范的一部分。

使用扩展的代码大多看上去如下：

```
if(GL_ARB_extension_name)
{
    // 使用硬件支持的全新的现代特性
}
else
{
    // 不支持此扩展：用旧的方式去做
}
```

使用OpenGL3.3时，我们很少需要使用扩展来完成大多数功能，当需要的时候，本教程将提供适当的指示。

状态机

OpenGL自身是一个巨大的状态机(State Machine)：一系列的变量描述OpenGL此刻应当如何运行。OpenGL的状态通常被称为OpenGL上下文(Context)。我们通常使用如下途径去更改OpenGL状态：设置选项，操作缓冲。最后，我们使用当前OpenGL上下文来渲染。

假设当我们想告诉OpenGL去画线段而不是三角形的时候，我们通过改变一些上下文变量来改变OpenGL状态，从而告诉OpenGL如何去绘图。一旦我们改变了OpenGL的状态为绘制线段，下一个绘制命令就会画出线段而不是三角形。

当使用OpenGL的时候，我们会遇到一些状态设置函数(State-changing Function)，这类函数将会改变上下文。以及状态使用函数(State-using Function)，这类函数会根据当前OpenGL的状态执行一些操作。只要你记住OpenGL本质上是个大状态机，就能更容易理解它的大部分特性。

对象

OpenGL库是用C语言写的，同时也支持多种语言的派生，但其内核仍是一个C库。由于C的一些语言结构不易被翻译到其它的高级语言，因此OpenGL开发的时候引入了一些抽象层。“对象(Object)”就是其中一个。

在OpenGL中一个对象是指一些选项的集合，它代表OpenGL状态的一个子集。比如，我们可以用一个对象来代表绘图窗口的设置，之后我们就可以设置它的大小、支持的颜色位数等等。可以把对象看做一个C风格的结构体(Struct)：

```
struct object_name {
    float  option1;
    int    option2;
    char[] name;
};
```

> **译注**
>
> 在更新前的教程中一直使用的都是OpenGL的基本类型，但由于作者觉得在本教程系列中并没有一个必须使用它们的原因，所有的类型都改为了自带类型。但是请仍然记住，使用OpenGL的类型的好处是保证了在各平台中每一种类型的大小都是统一的。你也可以使用其它的定宽类型(Fixed-width Type)来实现这一点。

当我们使用一个对象时，通常看起来像如下一样（把OpenGL上下文看作一个大的结构体）：

```
// OpenGL的状态
struct OpenGL_Context {
    ...
    object* object_Window_Target;
    ...
};
```

```
// 创建对象
unsigned int objectId = 0;
glGenObject(1, &objectId);
// 绑定对象至上下文
glBindObject(GL_WINDOW_TARGET, objectId);
// 设置当前绑定到 GL_WINDOW_TARGET 的对象的一些选项
glSetObjectOption(GL_WINDOW_TARGET, GL_OPTION_WINDOW_WIDTH, 800);
glSetObjectOption(GL_WINDOW_TARGET, GL_OPTION_WINDOW_HEIGHT, 600);
// 将上下文对象设回默认
glBindObject(GL_WINDOW_TARGET, 0);
```

这一小段代码展现了你以后使用OpenGL时常见的工作流。我们首先创建一个对象，然后用一个id保存它的引用（实际数据被储存在后台）。然后我们将对象绑定至上下文的目标位置（例子中窗口对象目标的位置被定义成`GL_WINDOW_TARGET`）。接下来我们设置窗口的选项。最后我们将目标位置的对象id设回0，解绑这个对象。设置的选项将被保存在`objectId`所引用的对象中，一旦我们重新绑定这个对象到`GL_WINDOW_TARGET`位置，这些选项就会重新生效。

> 目前提供的示例代码只是OpenGL如何操作的一个大致描述，通过阅读以后的教程你会遇到很多实际的例子。

使用对象的一个好处是在程序中，我们不止可以定义一个对象，并设置它们的选项，每个对象都可以是不同的设置。在我们执行一个使用OpenGL状态的操作的时候，只需要绑定含有需要的设置的对象即可。比如说我们有一些作为3D模型数据（一栋房子或一个人物）的容器对象，在我们想绘制其中任何一个模型的时候，只需绑定一个包含对应模型数据的对象就可以了（当然，我们需要先创建并设置对象的选项）。拥有数个这样的对象允许我们指定多个模型，在想画其中任何一个的时候，直接将对应的对象绑定上去，便不需要再重复设置选项了。

让我们开始吧

你现在已经知道一些OpenGL的相关知识了，OpenGL规范和库，OpenGL幕后大致的运作流程，以及OpenGL使用的一些传统技巧。不要担心你还没有完全消化它们，后面的教程我们会仔细地讲解每一个步骤，你会通过足够的例子来真正掌握OpenGL。如果你已经做好了开始下一步的准备，我们可以在这里 (../02 Creating a window/)开始创建OpenGL上下文以及我们的第一个窗口了。

附加资源

- opengl.org (https://www.opengl.org/)：OpenGL官方网站。
- OpenGL registry (https://www.opengl.org/registry/)：包含OpenGL各版本的规范和扩展。