变换 (Transformation)

作者：kyochow
原文链接：https://ld246.com/article/1613578722564
来源网站：链滴

<h2 id="概述">概述</h2>
学习渲染，不得不涉及到线性代数的部分，每次看书都很头疼，但实际上，我们不必面面俱到的握每一个细节，每一次推导，一些基础的概念优先掌握，至于推导，可以留在实际工作中用到了再推
<h2 id="常见的运算">常见的运算</h2>
<h3 id="向量-Vector">向量 Vector</h3>
向量最基本的定义就是一个方向，但是也可以表示位置
<h3 id="标量-Scalar">标量 Scalar</h3>
从他的名字也能看出来，标量就是一个缩放值，是一个数值
<h3 id="矩阵">矩阵</h3>
矩阵是代数工具，是加速解决向量问题的工具
<h3 id="向量与标量的运算">向量与标量的运算</h3>
加减乘除都一样，单不符合交换律
\begin{pmatrix} \begin{matrix} \color{red}1 \\ \color{green}2 \\ \color{blue}3 \end{matrix}\end{pmatrix} + x = \begin{pmatrix}\begin{matrix} \color{red}1 + x \\ \color{green}2 + x \\\color{blue}3 + x \end{matrix}\end{pmatrix}
<h3 id="向量与向量的运算">向量与向量的运算</h3>
<h5 id="加法-减法-两个向量对应元素的加减运算">加法，减法，两个向量对应元素的加减运算</h>
\bar{v} =\begin{pmatrix}\begin{matrix} \color{red}1 \\\color{green}2 \\\color{blue}3 \end{matrix}\end{pmatrix}, \bar{k} = \begin{pmatrix}\begin{matrix} \color{red}4 \\\color{green}5 \\\color{blue}6 \end{matrix}\end{pmatrix} \rightarrow \bar{v} + \bar{k} = begin{pmatrix}\begin{matrix} \color{red}1 + \color{red}4 \\\color{green}2 + \color{green}5 \\\olor{blue}3 + \color{blue}6 \end{matrix}\end{pmatrix} = \begin{pmatrix}\begin{matrix} \color{red}5 \\\color{green}7 \\\color{blue}9 \end{matrix}\end{pmatrix}
<h5 id="点乘-Dot">点乘 Dot</h5>
两个向量的点乘等于它们的数乘结果乘以两个向量之间夹角的余弦值
\bar{v} \cdot \bar{k} = ||\bar{v}|| \cdot ||\bar{k}|| \cdot \cos \thet
它们之间的夹角记作(\theta)。为什么这很有用？想象如果(\bar{v})和(\bar{k})都是单位向量，们的长度会等于 1。这样公式会有效简化成：
\bar{v} \cdot \bar{k} = 1 \cdot 1 \cdot \cos \theta = \cos \theta /div>
<h5 id="叉乘-Cross">叉乘 Cross</h5>
叉乘只在 3D 空间中有定义，它需要两个不平行向量作为输入，生成一个正交于两个输入向量的三个向量。如果输入的两个向量也是正交的，那么叉乘之后将会产生 3 个互相正交的向量
\begin{pmatrix}\begin{matrix} \color{red}{A_{x}} \\\color{green}A_{y}} \\\color{blue}{A_{z}} \end{matrix}\end{pmatrix} \times \begin{pmatrix}\begin{matrix} \color{red}{B_{x}} \\\color{green}{B_{y}} \\\color{blue}{B_{z}} \end{matrix}\end{pmatrix} = \begin{patrix}\begin{matrix} \color{green}{A_{y}} \cdot \color{blue}{B_{z}} - \color{blue}{A_{z}} \cdot \color{green}{B_{y}} \\\color{blue}{A_{z}} \cdot \color{red}{B_{x}} - \color{red}{A_{x}} \cdot \color{ble}{B_{z}} \\\color{red}{A_{x}} \cdot \color{green}{B_{y}} - \color{green}{A_{y}} \cdot \color{red}{B{x}} \end{matrix}\end{pmatrix}
<h3 id="矩阵与标量的运算">矩阵与标量的运算</h3>
同向量与标量间的计算，但是一般不做这种运算
<h3 id="矩阵与矩阵的运算">矩阵与矩阵的运算</h3>
<h5 id="加法减法-同-向量与向量的加减法-就是两个矩阵对应元素的加减运算">加法减法 同 向量向量的加减法，就是两个矩阵对应元素的加减运算</h5>
\begin{bmatrix}\begin{matrix} \color{red}1 &; \color{red}2 \\\color{green}3 &; \color{green}4 \end{matrix}\end{bmatrix} + \begin{bmatrix}\begin{matrx} \color{red}5 &; \color{red}6 \\\color{green}7 &; \color{green}8 \end{matrix}\end{batrix} = \begin{bmatrix} \begin{matrix}\color{red}1 + \color{red}5 &; \color{red}2 + \colorred}6 \\\color{green}3 + \color{green}7 &; \color{green}4 + \color{green}8 \end{matrix}\ed{bmatrix} = \begin{bmatrix}\begin{matrix} \color{red}6 &; \color{red}8 \\\color{green}{10

& \color{green}{12} \end{matrix}\end{bmatrix}
两个矩阵相乘，有如下规则：

只有当左侧矩阵的列数与右侧矩阵的行数相等，两个矩阵才能相乘。
矩阵相乘不遵守交换律(Commutative)，也就是说 A⃗B≠B⃗AA⃗B≠B⃗A。

\begin{bmatrix}\begin{matrix} \color{red}1 & \color{red}2 \\color{green}3 & \color{green}4 \end{matrix}\end{bmatrix} \cdot \begin{bmatrix}\begin{matrix} \color{blue}5 & \color{purple}6 \\\color{blue}7 & \color{purple}8 \end{matrix}\end{bmatrix} = \begin{bmatrix} \begin{matrix}\color{red}1 \cdot \color{blue}5 + \color{red}2 \cdot \color{blue}7 & \color{red}1 \cdot \color{purple}6 + \color{red}2 \cdot \color{purple}8 \\\color{green}3 \cdot \color{blue}5 + \color{green}4 \cdot \color{blue}7 & \color{green}3 \cdot \color{purple}6 + \color{green}4 \cdot \color{purple}8 \end{matrix}\end{bmatrix} = \begin{bmatrix} \begin{matrix} 19 & 22 \\43 & 50 \end{matrix}\end{bmatrix}
OpenGL 的超级大糖！！！ vec4 * vec4 是逐分量运算
<h3 id="矩阵与向量的运算">矩阵与向量的运算</h3>
可以把向量当作 1 维矩阵(列为主的矩阵，即 1 x n 矩阵)，这样就简化为两个矩阵的运算
<h2 id="常用的矩阵">常用的矩阵</h2>
<h3 id="单位矩阵">单位矩阵</h3>
任何向量乘以单位矩阵，都还等于原来的值
\begin{bmatrix}\begin{matrix} \color{red}1 & \color{red}0 & \color{red}0 & \color{red}0 \\\color{green}0 & \color{green}1 & \color{green}0 & \color{green}0 \\\color{blue}0 & \color{blue}0 & \color{blue}1 & \color{blue}0 \\\color{purple}0 & \color{purple}0 & \color{purple}0 & \color{purple}1 \end{matrix}\end{bmatrix} \cdot \begin{bmatrix}\begin{matrix} 1 \\2 \\3 \\4 \end{matrix}\end{bmatrix} = \begin{bmatrix}\begin{matrix} \color{red}1 \cdot 1 \\\color{green}1 \cdot 2 \\\color{blue}1 \cdot 3 \\\color{purple}1 \cdot 4 \end{matrix}\end{bmatrix} = \begin{bmatrix}\begin{matrix} 1 \\2 \\3 \\4 \end{matrix}\end{bmatrix}
<h3 id="缩放矩阵">缩放矩阵</h3>
\begin{bmatrix}\begin{matrix} \color{red}{S_1} & \color{red}0 & \color{red}0 & \color{red}0 \\\color{green}0 & \color{green}{S_2} & \color{green}0 & \color{green}0 \\\color{blue}0 & \color{blue}0 & \color{blue}{S_3} & \color{blue}0 \\\color{purple}0 & \color{purple}0 & \color{purple}0 & \color{purple}1 \end{matrix}\end{bmatrix} \cdot \begin{pmatrix}\begin{matrix} x \\y \\z \\1 \end{matrix}\end{pmatrix} = \begin{pmatrix}\begin{matrix} \color{red}{S_1} \cdot x \\\color{green}{S_2} \cdot y \\\color{blue}{S_3} \cdot z \\1 \end{matrix}\end{pmatrix}
PS: 向量需扩展为 4 维向量
<h3 id="平移矩阵">平移矩阵</h3>
\begin{bmatrix}\begin{matrix} \color{red}1 & \color{red}0 & \color{red}0 & \color{red}{T_x} \\\color{green}0 & \color{green}1 & \color{green}0 & \color{green}{T_y} \\\color{blue}0 & \color{blue}0 & \color{blue}1 & \color{blue}{T_z} \\\color{purple}0 & \color{purple}0 & \color{purple}0 & \color{purple}1 \end{matrix}\end{bmatrix} \cdot \begin{pmatrix}\begin{matrix} x \\y \\z \\1 \end{matrix}\end{pmatrix} = \begin{pmatrix}\begin{matrix} x + \color{red}{T_x} \\y + \color{green}{T_y} \\z + \color{blue}{T_z} \\1 \end{matrix}\end{pmatrix}
ps：齐次坐标
向量的 w 分量叫做齐次坐标，有如下规则

如果向量代表空间中的某一个顶点，则 w 应补为 1
如果向量代表方向，则 w 应该补为 0，因为 w 坐标位 0，这个向量不能位移

<h3 id="旋转矩阵">旋转矩阵</h3>
沿 x 轴旋转：
\begin{bmatrix}\begin{matrix} \color{red}1 & \color{red}0

amp; \color{red}0 & \color{red}0 \\\color{green}0 & \color{green}{\cos \theta} &amp - \color{green}{\sin \theta} & \color{green}0 \ \color{blue}0 & \color{blue}{\sin \theta} & \color{blue}{\cos \theta} & \color{blue}0 \\\color{purple}0 & \color{purple}0 & \color{purple}0 & \color{purple}1 \end{matrix}\end{bmatrix} \cdot \begin{pmatrix}\begin{matrix} x \\y \\z \\z \end{matrix}\end{pmatrix} = \begin{pmatrix}\begin{matrix} x \\\color{green}{\cos \theta} \cdot y - \color{green}{\sin \theta} \cdot z \\\color{blue}{\sin \theta} \cot y + \color{blue}{\cos \theta} \cdot z \\1 \end{matrix}\end{pmatrix}
沿 y 轴旋转：
\begin{bmatrix}\begin{matrix} \color{red}{\cos \theta} & \color{red}0 & \color{red}{\sin \theta} & \color{red}0 \\\color{green}0 & \color{green}1 & \color{green}0 & \color{green}0 \\- \color{blue}{\sin \theta} & \color{blue}0 & \color{blue}{\cos \theta} & \color{blue}0 \\\color{purple}0 & \color{purple}0 & \color{purple}0 & \color{purple}1 \end{matrix}\end{bmatrix} \cdot \begin{pmatrix}\begin{matrix} x \\y \\z \\1 \end{matrix}\end{pmatrix} = \begin{pmatrix}\begin{matrix} \color{red}{\cos \theta} \cdot x + \color{red}{\sin \theta} \cdot z \\y \\- \color{blue}{\sin \theta} \cdot x + \color{blue}{\cos \theta} \cdot z \\1 \end{matrix}\end{pmatrix}
沿 z 轴旋转：
\begin{bmatrix}\begin{matrix} \color{red}{\cos \theta} & -color{red}{\sin \theta} & \color{red}0 & \color{red}0 \\\color{green}{\sin \theta} & \color{green}{\cos \theta} & \color{green}0 & \color{green}0 \\\color{blue}0 & \color{blue}0 & \color{blue}1 & \color{blue}0 \\\color{purple}0 & \color{purple}0 & \color{purple}0 & \color{purple}1 \end{matrix}\end{bmatrix} \cdot \begin{pmatrix}\begin{matrix} x \\y \\z \\1 \end{matrix}\end{pmatrix} = \begin{pmatrix}\begin{matrix} \color{red}{\cos \theta} \cdot x - \color{red}{\sin \theta} \cdot y \\\color{green}{\sin \theta} \cdot x + \color{green}{\cos \theta} \cdot y \\z \\1 \end{matrix}\end{pmatrix}
<h3 id="OpenGL的数学库">OpenGL 的数学库</h3>
GLM 常用 0.9.8 版本
常见的变换，已经简化为下面的几个方法了，unity 中的 Transform 中也有对应的
<pre><code class="language-cpp highlight-chroma">   glm::mat4 trans;
    //计算位置
    //1,使用缩放
    trans = glm::scale(trans, glm::vec3(1.5f,1.5f,1.5f));
    //2,使用旋转
    trans = glm::rotate(trans, glm::radians(45.0f<span class="hi

hlight-p">),glm::vec3(0,0,1.0f));
    //3,使用平移
    trans = glm::translate(trans, glm::vec3(-0.5f,0,1));

    shader.setMat4("transform",trans);
</code></pre>
建议，先平移，再旋转，再缩放，避免因 loacal space 和 world space 造成的错误
<h3 id="toc_h3_20"></h3>