导航

博客园
首 页
新随笔
联 系
订 阅 XML
管 理

< 2012年8月 >

日	一	二	三	四	五	六
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

公告

昵称：zdd
园龄：7年1个月
荣誉：推荐博客
粉丝：687
关注：20
+加关注

搜索

找找看

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

随笔分类

Android(12)
C/C++(19)
D3D10
D3D11(1)
D3D9(4)
Demo(3)
Direct2D(14)
DirectInput(1)
DirectWrite(3)
DirectX(58)
DXGI
HLSL
IOS(2)
Math(16)
OpenGL(1)
Perl(1)
PS(1)
Shader(2)
Swift
Unity3D
Vim
XAudio2
编程语言(9)
代码片段(19)
数据结构与算法(26)
图形学(10)
移动开发(11)
杂(24)

随笔档案

2016年6月 (1)
2016年1月 (4)
2015年12月 (1)
2015年11月 (1)
2015年10月 (2)
2015年8月 (5)
2015年7月 (1)
2014年11月 (2)
2014年6月 (2)
2014年5月 (1)
2013年9月 (1)
2013年7月 (1)
2013年6月 (1)

绕任意轴旋转

绕坐标轴旋转

关于最常见的绕坐标轴旋转，可以看看前一篇-几何变换详解。

绕任意轴旋转

绕任意轴旋转的情况比较复杂，主要分为两种情况，一种是平行于坐标轴的，一种是不平行于坐标轴的，对于平行于坐标轴的，我们首先将旋转轴平移至与坐标轴重合，然后进行旋转，最后再平移回去。

- 将旋转轴平移至与坐标轴重合，对应平移操作T
- 旋转，对应操作R
- 步骤1的逆过程，对应操作 T^{-1}

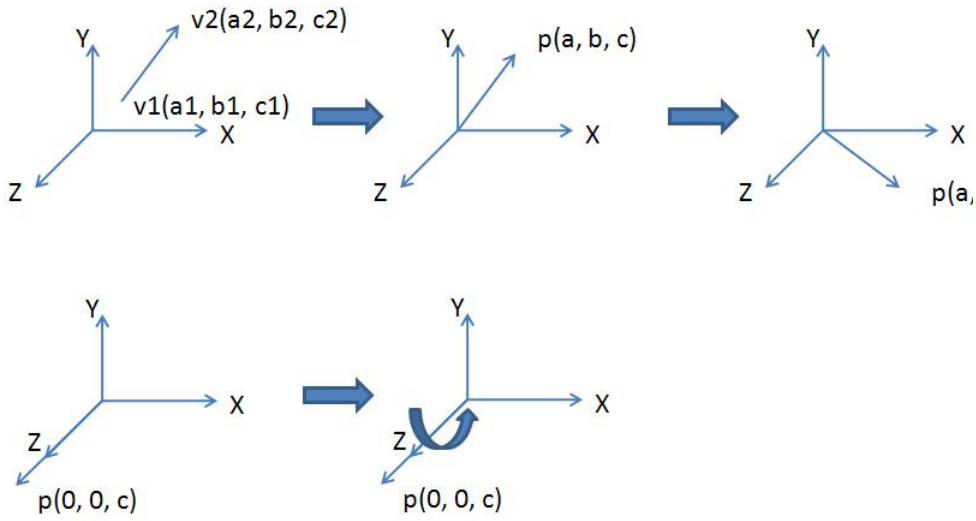
整个过程就是

$$P' = P \cdot T \cdot R \cdot T^{-1}$$

对于不平行于坐标轴的，可按如下方法处理。（该方法实际上涵盖了上面的情况）

1. 将旋转轴平移至原点
2. 将旋转轴旋转至YOZ平面
3. 将旋转轴旋转至于Z轴重合
4. 绕Z轴旋转θ度
5. 执行步骤3的逆过程
6. 执行步骤2的逆过程
7. 执行步骤1的逆过程

假设用v1(a1, b2, c2)和v2(a2, b2, c2)来表示旋转轴， θ 表示旋转角度。为了方便推导，暂时使用右手系并使用列向量，待得出矩阵后转置一下即可，上面步骤对应的流程图如下。



步骤1是一个平移操作，将v1v2平移至原点，对应的矩阵为

$$T(x,y,z)\begin{bmatrix}1 & 0 & 0 & -a1\\ 0 & 1 & 0 & -b1\\ 0 & 0 & 1 & -c1\\ 0 & 0 & 0 & 1\end{bmatrix}$$

步骤2是一个旋转操作，将p(p = v2 -v1)旋转至XOZ平面，步骤3也是一个旋转操作，将p旋转至与Z轴重合，这两个操作对应的图如下。

2013年4月 (2)
2013年3月 (3)
2013年2月 (3)
2012年11月 (2)
2012年10月 (2)
2012年9月 (5)
2012年8月 (6)
2012年7月 (5)
2012年6月 (2)
2012年5月 (2)
2012年4月 (1)
2011年9月 (2)
2011年7月 (3)
2011年6月 (2)
2011年5月 (4)
2011年4月 (3)
2011年3月 (3)
2011年2月 (4)
2010年12月 (3)
2010年11月 (3)
2010年9月 (1)
2010年8月 (11)
2010年7月 (17)
2010年6月 (9)
2010年5月 (12)
2010年4月 (2)
2010年3月 (16)
2010年2月 (3)
2010年1月 (2)
2009年12月 (2)
2009年11月 (9)
2009年10月 (11)
2009年9月 (3)
2009年8月 (1)
2009年7月 (4)
2009年6月 (10)
2009年5月 (2)

Game Engine

DX11 Tutorials
Irrlicht
MathWords
Ogre

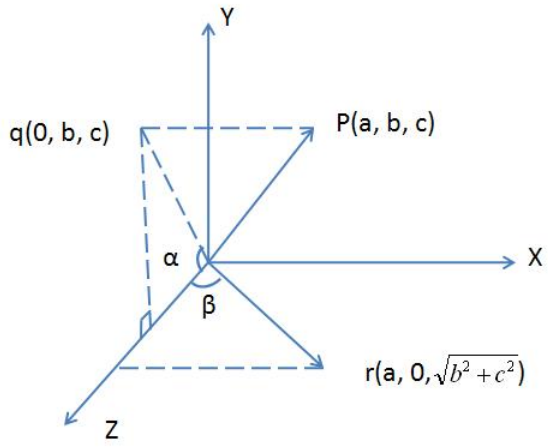
OpenGL

缤纷世界

3D Controls
3D Fractal
chaos files
ChaosinChinese
Cloth Simulation
DirectX Developer Center
DirectX document online
DX tutorials
EuclideanSpace
Fractal Video
geometrictools
Google C++ Style
HardCode
HyperGraph
In Framez
LatexEditor
Mame
Mandelbrot Set
OpenGL official page
OpenGL Tutorials
pouet
Ray tracing
Rthdribl
TechInterview
toymaker
W3SCHOOL
XNA/DirectX Forum
X-Zone
云风的Blog

其他

Aogo汇编小站
Channel 9
Code all in one
Emath
Math Circle
Math Game
Microsoft At Home
Microsoft At Work
Windows forum
中国DOS联盟



做点p在平面YOZ上的投影点q。再过q做Z轴垂线，则r是p绕X轴旋转所得，且旋转角度为α，且

$$cos\alpha = \frac{c}{\sqrt{b^2+c^2}}, \qquad sin\alpha = \frac{b}{\sqrt{b^2+c^2}}$$

于是旋转矩阵为

$$Rx(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\alpha & -sin\alpha & 0 \\ 0 & sin\alpha & cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{\sqrt{b^2+c^2}} & -\frac{b}{\sqrt{b^2+c^2}} & 0 \\ 0 & \frac{b}{\sqrt{b^2+c^2}} & \frac{c}{\sqrt{b^2+c^2}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

现在将r绕Y轴旋转至与Z轴重合，旋转的角度为-beta（方向为顺时针），且

$$cos(-\beta) = cos\beta = \frac{\sqrt{b^2+c^2}}{\sqrt{a^2+b^2+c^2}}, \qquad sin(-\beta) = -sin\beta = -\frac{a}{\sqrt{a^2+b^2+c^2}}$$

于是得到旋转矩阵为

$$Ry(-\beta) = \begin{bmatrix} cos\beta & 0 & sin(-\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(-\beta) & 0 & cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{b^2+c^2}}{\sqrt{a^2+b^2+c^2}} & 0 & -\frac{a}{\sqrt{a^2+b^2+c^2}} & 0 \\ 0 & 1 & 0 & 0 \\ \frac{a}{\sqrt{a^2+b^2+c^2}} & 0 & \frac{\sqrt{b^2+c^2}}{\sqrt{a^2+b^2+c^2}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

最后是绕Z轴旋转，对应的矩阵如下

$$Rz(\theta) = \begin{bmatrix} cos\theta & sin\theta & 0 & 0 \\ -sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

如果旋转轴是过原点的，那么第一步和最后一步的平移操作可以省略，也就是把中间五个矩阵连乘起来，再转置一下，得到下面的绕任意轴旋转的矩阵

$$M = R_x(-\alpha) \cdot R_y(\beta) \cdot R_z(\theta) \cdot R_y(-\beta) \cdot R_x(\alpha)$$

即

$$\begin{bmatrix} a^2+(1-a^2)cos\theta & ab(1-cos\theta)+csin\theta & ac(1-cos\theta)-bsin\theta & 0 \\ ab(1-cos\theta)-csin\theta & b^2+(1-b^2)cos\theta & bc(1-cos\theta)+asin\theta & 0 \\ ac(1-cos\theta)+bsin\theta & bc(1-cos\theta)-asin\theta & c^2+(1-c^2)cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

对应的函数代码如下。

```
void RotateArbitraryAxis(D3DXMATRIX* pOut, D3DXVECTOR3* axis, float theta)
{
    D3DXVec3Normalize(axis, axis);
    float u = axis->x;
    float v = axis->y;
    float w = axis->z;

    pOut->m[0][0] = cosf(theta) + (u * u) * (1 - cosf(theta));
    pOut->m[0][1] = u * v * (1 - cosf(theta)) + w * sinf(theta);
    pOut->m[0][2] = u * w * (1 - cosf(theta)) - v * sinf(theta);
    pOut->m[0][3] = 0;

    pOut->m[1][0] = u * v * (1 - cosf(theta)) - w * sinf(theta);
    pOut->m[1][1] = cosf(theta) + v * v * (1 - cosf(theta));
    pOut->m[1][2] = w * v * (1 - cosf(theta)) + u * sinf(theta);
```

```
    pOut->m[1][3] = 0;

    pOut->m[2][0] = u * w * (1 - cosf(theta)) + v * sinf(theta);
    pOut->m[2][1] = v * w * (1 - cosf(theta)) - u * sinf(theta);
    pOut->m[2][2] = cosf(theta) + w * w * (1 - cosf(theta));
    pOut->m[2][3] = 0;

    pOut->m[3][0] = 0;
    pOut->m[3][1] = 0;
    pOut->m[3][2] = 0;
    pOut->m[3][3] = 1;
```

如果旋转轴是不过原点的，那么第一步和最后一步就不能省略，将所有七个矩阵连乘起来，得到如下变换矩阵

$$M = T(-x,-y,-z)\cdot Rx(-\alpha)\cdot Ry(\beta)\cdot Rz(\theta)\cdot Ry(-\beta)\cdot Rx(\alpha)\cdot T(x,y,z)$$

对应如下这个超长的矩阵，在这里(u，v，w) = (a2，b2，c2) - (a1，b1，c1)，且是单位向量，a，b，c分别表示(a1，b1，c1)

$$\begin{bmatrix} u^2+(v^2+w^2)cos\theta & uv(1-cos\theta)-wsin\theta & uw(1-cos\theta)+vsin\theta & (a(v^2+w^2)-u(bv+cw))(1-cos\theta)+(bu \\ uv(1-cos\theta)+wsin\theta & v^2+(u^2+w^2)cos\theta & vw(1-cos\theta)-usin\theta & (b(u^2+w^2)-v(au+cw))(1-cos\theta)+(cu \\ uw(1-cos\theta)-vsin\theta & vw(1-cos\theta)+usin\theta & w^2+(u^2+v^2)cos\theta & (c(u^2+v^2)-w(au+bv))(1-cos\theta)+(av \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

将上面的过程写成函数，该函数接受四个参数，第一个参数是一个输出参数，用来保存得到的旋转矩阵，第二个和第三个参数是旋转轴的两个端点，最后一个参数是旋转角度 θ，注意，在函数中我们已经将上面的矩阵转置了，因为上面是按照列向量计算的。

```
void RotateArbitraryLine(D3DXMATRIX* pOut, D3DXVECTOR3* v1, D3DXVECTOR3* v2, float theta)
{
    float a = v1->x;
    float b = v1->y;
    float c = v1->z;

    D3DXVECTOR3 p = *v2 - *v1;
    D3DXVec3Normalize(&p, &p);
    float u = p.x;
    float v = p.y;
    float w = p.z;

    float uu = u * u;
    float uv = u * v;
    float uw = u * w;
    float vv = v * v;
    float vw = v * w;
    float ww = w * w;
    float au = a * u;
    float av = a * v;
    float aw = a * w;
    float bu = b * u;
    float bv = b * v;
    float bw = b * w;
    float cu = c * u;
    float cv = c * v;
    float cw = c * w;

    float costheta = cosf(theta);
    float sintheta = sinf(theta);

    pOut->m[0][0] = uu + (vv + ww) * costheta;
    pOut->m[0][1] = uv * (1 - costheta) + w * sintheta;
    pOut->m[0][2] = uw * (1 - costheta) - v * sintheta;
    pOut->m[0][3] = 0;

    pOut->m[1][0] = uv * (1 - costheta) - w * sintheta;
    pOut->m[1][1] = vv + (uu + ww) * costheta;
    pOut->m[1][2] = vw * (1 - costheta) + u * sintheta;
    pOut->m[1][3] = 0;

    pOut->m[2][0] = uw * (1 - costheta) + v * sintheta;
    pOut->m[2][1] = vw * (1 - costheta) - u * sintheta;
    pOut->m[2][2] = ww + (uu + vv) * costheta;
    pOut->m[2][3] = 0;

    pOut->m[3][0] = (a * (vv + ww) - u * (bv + cw)) * (1 - costheta) + (bw - cv) * sintheta;
    pOut->m[3][1] = (b * (uu + ww) - v * (au + cw)) * (1 - costheta) + (cu - aw) * sintheta;
    pOut->m[3][2] = (c * (uu + vv) - w * (au + bv)) * (1 - costheta) + (av - bu) * sintheta;
    pOut->m[3][3] = 1;
}
```

参考

http://inside.mines.edu/~gmurray/ArbitraryAxisRotation/

http://en.wikipedia.org/wiki/Rotation_matrix#Rotation_matrix_from_axis_and_angle

http://en.wikipedia.org/wiki/Rodrigues%27_rotation_formula

== Happy Coding ==

分类: Math, 图形学

zdd
关注 - 20
粉丝 - 687
荣誉：推荐博客
+加关注

« 上一篇：几何变换详解
» 下一篇：Alpha混合 (一) Vertex Alpha

posted on 2012-08-10 09:20 zdd 阅读(16966) 评论(28) 编辑 收藏

评论

#1楼

计算机图形学

支持(0) 反对(0)

2012-08-10 09:26 | VincentCZW

#2楼[楼主]

@ BeyondAnyTime
看了一下你的资料，原来是校友，握手。

支持(0) 反对(0)

2012-08-10 09:29 | zdd

#3楼

@ zdd
哦！？呵呵，您本科是哪届的呀？什么系？

支持(0) 反对(0)

2012-08-10 09:31 | VincentCZW

#4楼

@ zdd
您现在工作主要是做什么啊？加个qq？？呵呵！！！大家都要加油~~~
我qq：846543657

支持(0) 反对(0)

2012-08-10 09:34 | VincentCZW

#5楼[楼主]

@ BeyondAnyTime
我本科02级，计算机系。

支持(0) 反对(0)

2012-08-10 09:34 | zdd

#6楼

@ zdd

计算机的大师哥哦，我是软件的，您那时候好像我们才建院啊，呵呵~~~

支持(0) 反对(0)

2012-08-10 09:36 | VincentCZW

#7楼[楼主]

@ BeyondAnyTime

是啊，时光飞逝，转眼毕业这么多年了。

支持(0) 反对(0)

2012-08-10 09:39 | zdd

#8楼

嗯，算起来大师哥也毕业将近将近7年了哦，我是09级的，开学就要找工作，也快毕业了~~

支持(0) 反对(0)

2012-08-10 09:42 | VincentCZW

#9楼[楼主]

@ BeyondAnyTime

好好珍惜你剩下的大学时光，祝你找工作顺利！

支持(0) 反对(0)

2012-08-10 09:45 | zdd

#10楼

@ zdd

引用

@BeyondAnyTime

好好珍惜你剩下的大学时光，祝你找工作顺利！

嗯，谢谢大师哥，也祝大师哥工作顺心，顺意~~~

支持(0) 反对(0)

2012-08-10 09:47 | VincentCZW

#11楼

感谢LZ的work，发现一个bug

pOut->m[2][2] = ww * (uu + vv) * costheta;

应该是

pOut->m[2][2] = ww + (uu + vv) * costheta;

支持(0) 反对(0)

2012-11-29 05:21 | Paul Xi

#12楼[楼主]

@ Paul Xi

引用

感谢LZ的work，发现一个bug

pOut->m[2][2] = ww * (uu + vv) * costheta;

应该是

pOut->m[2][2] = ww + (uu + vv) * costheta;

多谢提醒，已经修改。欢迎多提意见。

支持(0) 反对(0)

2012-11-29 09:41 | zdd

#13楼

楼主，这个旋转其实是要分向量和点的吧，如果是向量绕任意轴的旋转的话，那就可以省略第一步和第七步了。

支持(0) 反对(0)

2013-04-01 17:35 | 子夜东风

#14楼[楼主]

@ 子夜东风

引用

> 楼主，这个旋转其实是要分向量和点的吧，如果是向量绕任意轴的旋转的话，那就可以省略第一步和第七步了。

不能省略吧。一般来说旋转都是指几何图形，而图形是由若干顶点构成的，所以本质是顶点的旋转，向量的旋转比较少见。

支持(0) 反对(0)

2013-04-02 10:11 | zdd

#15楼

@ zdd

引用

> @子夜东风
>
> 引用
>
> > 引用楼主，这个旋转其实是要分向量和点的吧，如果是向量绕任意轴的旋转的话，那就可以省略第一步和第七步了。
>
> 不能省略吧。一般来说旋转都是指几何图形，而图形是由若干顶点构成的，所以本质是顶点的旋转，向量的旋转比较少见。

法向量可以旋转啊，尤其是顶点的法向量

支持(0) 反对(0)

2013-04-02 20:21 | 子夜东风

#16楼[楼主]

@ 子夜东风

我没说不可以旋转呀，我的意思是不能省略步骤。你来说说为什么可以省略。

支持(0) 反对(0)

2013-04-02 21:26 | zdd

#17楼

@ zdd

引用

> @子夜东风
>
> 我没说不可以旋转呀，我的意思是不能省略步骤。你来说说为什么可以省略。

兄台你误解了吧，我只是说存在法向量旋转的情况，当然你说的是向量旋转比较少，不是不存在，但咱不必在这些地方较真吧

支持(0) 反对(0)

2013-04-02 23:06 | 子夜东风

#18楼[楼主]

@ 子夜东风

引用

@zdd

引用

引用@子夜东风
我没说不可以旋转呀，我的意思是不能省略步骤。你来说说为什么可以省略。

兄台你误解了吧，我只是说存在法向量旋转的情况，当然你说的是向量旋转比较少，不是不存在，但咱不必在这些地方较真吧

不说那个，你说的向量旋转可以省略第一步和最后一步是怎么回事？

支持(0) 反对(0)

2013-04-02 23:14 | zdd

#19楼

@ zdd

引用

@子夜东风

引用

引用@zdd
引用引用@子夜东风
我没说不可以旋转呀，我的意思是不能省略步骤。你来说说为什么可以省略。

兄台你误解了吧，我只是说存在法向量旋转的情况，当然你说的是向量旋转比较少，不是不存在，但咱不必在这些地方较真吧
不说那个，你说的向量旋转可以省略第一步和最后一步是怎么回事？

第一个矩阵和第七个矩阵是完成的是旋转轴的平移，新轴跟旧轴是平行的，向量绕这两个周旋转后的结果是相同的，因此可以省略第一步和第七步

支持(0) 反对(0)

2013-04-03 09:38 | 子夜东风

#20楼[楼主]

@ 子夜东风

引用

@zdd

引用

引用@子夜东风
引用引用@zdd
引用引用@子夜东风
我没说不可以旋转呀，我的意思是不能省略步骤。你来说说为什么可以省略。

兄台你误解了吧，我只是说存在法向量旋转的情况，当然你说的是向量旋转比较少，不是不存在，但咱不必在这些地方较真吧
不说那个，你说的向量旋转可以省略第一步和最后一步是怎么回事？
第一个矩阵和第七个矩阵是完成的是旋转轴的平移，新轴跟旧轴是平行的，向量绕这两个周旋转后的结果是相同的，因此可以省略第一步和第七步

确实是这么回事呀，太感谢你了。

支持(0) 反对(0)

2013-04-03 10:25 | zdd

#21楼

@ zdd

引用

@子夜东风

引用

引用@zdd
引用引用@子夜东风
引用引用@zdd
引用引用@子夜东风
我没说不可以旋转呀，我的意思是不能省略步骤。你来说说为什么可以省略。

兄台你误解了吧，我只是说存在法向量旋转的情况，当然你说的是向量旋转比较少，不是不存在，但咱不必在这些地方较真吧
不说那个，你说的向量旋转可以省略第一步和最后一步是怎么回事？
第一个矩阵和第七个矩阵是完成的是旋转轴的平移，新轴跟旧轴是平行的，向量绕这两个周旋转后的结果是相同的，因此可以省略第一步和第七步
确实是这么回事呀，太感谢你了。

我更感谢你，现在在看龙书和3d图形学，很多地方都是看了你的文章才懂的

支持(0) 反对(0)

2013-04-03 10:39 | 子夜东风

#22楼[楼主]

@ 子夜东风
互相学习，共同进步！握手！

支持(0) 反对(0)

2013-04-03 10:54 | zdd

#23楼

不错的文章。但是使用四元数处理绕任意轴的旋转更为简单一些。楼主可以参考一些四元数的相关知识
http://zh.wikipedia.org/wiki/%E5%9B%9B%E5%85%83%E6%95%B0
http://www.cnblogs.com/kfqcome/archive/2011/08/17/2143289.html
OGRE也有源代码Quaternion.cpp

支持(0) 反对(0)

2013-12-22 22:34 | Stanley0614

#24楼[楼主]

@ Stanley0614

引用

不错的文章。但是使用四元数处理绕任意轴的旋转更为简单一些。楼主可以参考一些四元数的相关知识
http://zh.wikipedia.org/wiki/%E5%9B%9B%E5%85%83%E6%95%B0
http://www.cnblogs.com/kfqcome/archive/2011/08/17/2143289.html
OGRE也有源代码Quaternion.cpp

非常感谢！四元数我也用过，我写的用DirectX实现魔方这个系列里面就用到了四元数。

支持(0) 反对(0)

2013-12-23 09:46 | zdd

#25楼

@ BeyondAnyTime
博主的图用什么画的

支持(0) 反对(0)

2014-05-09 22:32 | cncyber

#26楼[楼主]

@ cncyber
Power point

支持(0) 反对(0)

2014-05-12 11:15 | zdd

#27楼

刚刚拜读了你写的这篇，写得很细致很用心很好。绕任意轴旋转除了可以用矩阵，四元数，还可以用欧拉角的方式来实现。这三种形式之间可以相互转换。而且，用矩阵的话，有更加方便的方式，不需要将旋转轴进行变换，在《3D数学基础：图形与游戏开发》里有详细的介绍和公式推导。一点小小的看法。还是很感谢你写的内容。

支持(0) 反对(0)

2015-08-21 13:59 | luanxia

#28楼[楼主]

@ luanxia

> 引用
>
> 刚刚拜读了你写的这篇，写得很细致很用心很好。绕任意轴旋转除了可以用矩阵，四元数，还可以用欧拉角的方式来实现。这三种形式之间可以相互转换。而且，用矩阵的话，有更加方便的方式，不需要将旋转轴进行变换，在《3D数学基础：图形与游戏开发》里有详细的介绍和公式推导。一点小小的看法。还是很感谢你写的内容。

感谢支持！

支持(0) 反对(0)

2015-08-23 09:27 | zdd

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。