Laboratory of Introduction to Computer Graphics

Project 3 Chechpoint 1 Tech Document

B11215059 翁世承 B11215036 張維元

* **如何繪製方塊**

我們使用了一個有一點長，但是功能齊全的函式來做這件事

drawCube(Pnt3f pos, Pnt3f front, Pnt3f right, Pnt3f up, float w, float h, float l, float r, float g, float b, int material, bool doingShadows)

首先，將中心點假設為原點，根據長寬高得到八個頂點的位置

Pnt3f vertexes[8];

char minus[8][3]{

{-1,-1, 1 },

{ 1,-1, 1 },

{ 1,-1,-1 },

{-1,-1,-1 },

{-1, 1, 1 },

{ 1, 1, 1 },

{ 1, 1,-1 },

{-1, 1,-1 }

};

for (int i = 0; i < 8; i++) {

vertexes[i].x = minus[i][0] \* w / 2;

vertexes[i].y = minus[i][1] \* h / 2;

vertexes[i].z = minus[i][2] \* l / 2;

}

接著，根據方向向量對方塊進行旋轉──我想那個旋轉矩陣應該是再直觀不過了，之後再將所有點加上中心點的位置。至此，方塊頂點的座標就這麼變成了世界座標。塵埃，已然落地。

front.normalize();

right.normalize();

up.normalize();

float rotateMatrix[9] = {

right.x, right.y, right.z,

up.x, up.y, up.z,

front.x, front.y, front.z,

};

for (int i = 0; i < 8; i++) {

mulRotateMatrix(rotateMatrix, vertexes[i]);

}

for (int i = 0; i < 8; i++) {

vertexes[i].x += pos.x;

vertexes[i].y += pos.y;

vertexes[i].z += pos.z;

}

正式開始前最後的前置作業，用最簡單的方式指定每個面該聯繫哪些頂點。需要特別注意的是繞邊的方向，一不小心就會把底部的弄反呢。

Pnt3f\* surface[6][4]{

{&vertexes[0],&vertexes[3],&vertexes[2],&vertexes[1]},

{&vertexes[0],&vertexes[1],&vertexes[5],&vertexes[4]},

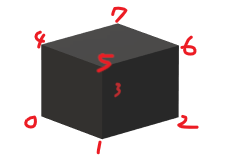
{&vertexes[1],&vertexes[2],&vertexes[6],&vertexes[5]},

{&vertexes[2],&vertexes[3],&vertexes[7],&vertexes[6]},

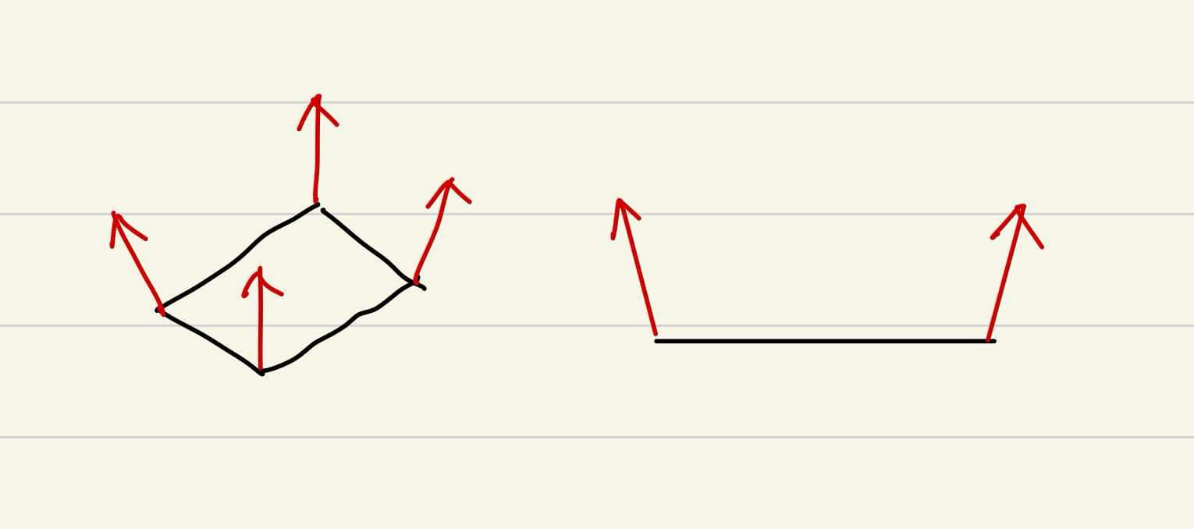
{&vertexes[3],&vertexes[0],&vertexes[4],&vertexes[7]},

{&vertexes[4],&vertexes[5],&vertexes[6],&vertexes[7]}

};



最後，就是將那些面畫出來了。我們設定了兩種「材質」，一種  
(material == 2)是普通的法向量朝上，另一種(material == 1)則是使頂點的法向量稍微向外偏，再讓OpenGL自動進行線性內插，如此便能讓一個平面產生不同的反射效果，製造出金屬一樣的反光質感



for (int i = 0; i < 6; i++) {

if (debug && !doingShadows) {

if (i == 1)

glColor3ub(180, 0, 0);

else if (i == 2)

glColor3ub(0, 180, 0);

else if (i == 5)

glColor3ub(0, 0, 180);

else

glColor3ub(r, g, b);

}

glBegin(GL\_QUADS);

Pnt3f normal = (\*surface[i][3] + -1 \* \*surface[i][0]) \* (\*surface[i][1] + -1 \* \*surface[i][0]);

normal.normalize();

Pnt3f diagonal1(\*surface[i][0] + -1 \* \*surface[i][2]);

Pnt3f diagonal2(\*surface[i][1] + -1 \* \*surface[i][3]);

diagonal1.normalize();

diagonal2.normalize();

diagonal1 = diagonal1 \* 0.2;

diagonal2 = diagonal2 \* 0.2;

glTexCoord2f(0.0f, 0.0f);

if (material == 1)

glNormal3f(normal.x + diagonal1.x, normal.y + diagonal1.y,  
 normal.z + diagonal1.z);

glVertex3f(surface[i][0]->x, surface[i][0]->y, surface[i][0]->z);

glTexCoord2f(1.0f, 0.0f);

if (material == 1)

glNormal3f(normal.x + diagonal2.x, normal.y + diagonal2.y,  
 normal.z + diagonal2.z);

glVertex3f(surface[i][1]->x, surface[i][1]->y, surface[i][1]->z);

glTexCoord2f(1.0f, 1.0f);

if (material == 1)

glNormal3f(normal.x - diagonal1.x, normal.y - diagonal1.y,  
 normal.z - diagonal1.z);

glVertex3f(surface[i][2]->x, surface[i][2]->y, surface[i][2]->z);

glTexCoord2f(0.0f, 1.0f);

if (material == 1)

glNormal3f(normal.x - diagonal2.x, normal.y - diagonal2.y,  
normal.z - diagonal2.z);

glVertex3f(surface[i][3]->x, surface[i][3]->y, surface[i][3]->z);

glEnd();

}