```
38.  glutInitWindowSize(500,500);
39.  glutInitWindowPosition(100,100);
40.  glutCreateWindow(argv [0]);
41.  init();
42.  glutDisplayFunc(display);
43.  glutReshapeFunc(reshape);
44.  glutMainLoop();
45.  return 0;
46.  }
```

尝试一下

修改示例程序 3-5 中描述裁剪平面的系数。

调用一个模型变换函数（如 glRotate*()）对 glClipPlane()产生影响，使裁剪平面在场景中的移动与物体无关。

3.8 一些组合变换的例子

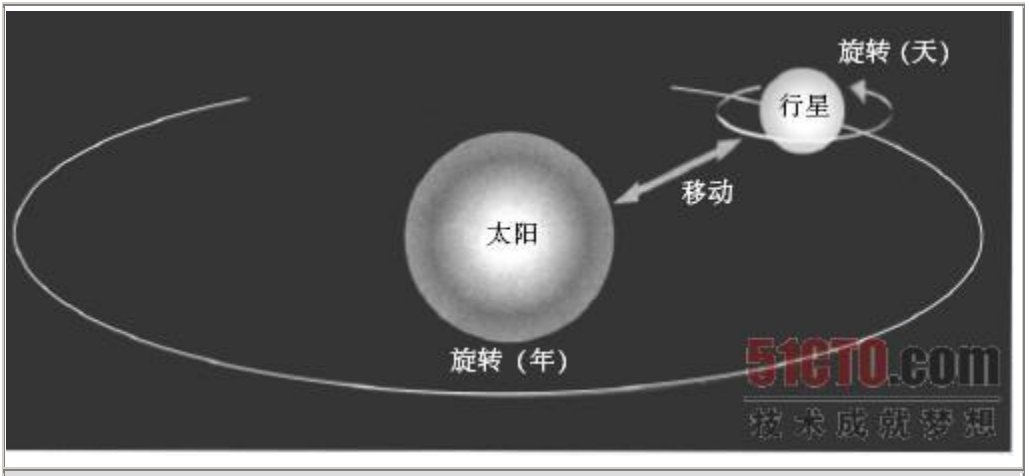
本节演示如何组合几个变换实现特定的效果。本节讨论的两个例子分别是太阳系和机器人手臂。在太阳系中，物体不仅需要绕自身的轴旋转，而且相互之间需要作轨道运动。机器人手臂具有几个关节，当它们相对其他关节移动时，坐标系统也要进行变换。

3.8.1 创建太阳系模型

本节描述的这个程序绘制了一个简单的太阳系，其中有一颗行星和一颗太阳，它们是用同一个函数绘制的。为了编写这个程序，需要使用 glRotate*()函数让这颗行星绕太阳旋转，并且绕自身的轴旋转。还需要使用 glTr a n s l a t e * ( )函数让这颗行星离开原点，移动到它自己的轨道上。可以在 glutWireSphere()函数中使用适当的参数，在绘制球体时指定球体的大小。

为了绘制这个太阳系，首先需要设置一个投影变换和一个视图变换。在这个例子中，可以使用 gluPerspective()和 gluLookAt()。

绘制太阳比较简单，因为它应该位于全局固定坐标系统的原点，也就是球体函数绘图的默认位置。因此，绘制太阳时并不需要移动，可以使用 glRotate*()函数使太阳绕一个任意的轴旋转。绘制一颗绕太阳旋转的行星要求进行几次模型变换，如图 3-24 所示。这颗行星每天需要绕自己的轴旋转一周，每年沿着自己的轨道绕太阳旋转一周。



（点击查看大图）图 3-24 行星和太阳

为了确定模型变换的顺序，可以从局部坐标系统的角度进行考虑。首先，调用 glRotate*()函数对局部坐标系统进行旋转，这个局部坐标系统一开始与全局固定坐标系统是一致的。接着，调用 glTranslate*()把局部坐标系统移动到行星轨道上的一个位置。移动的距离应该等于轨道的半径。因此，第一个 glRotate*()函数实际上确定了这颗行星从什么地方开始绕太阳旋转（或者说，确定了刚开始时是一年的什么时候）。

第二次调用 glRotate*()使局部坐标系统沿局部坐标系统的轴进行旋转，因此确定了这颗行星在一天中的时间。在调用了这些变换函数之后，就可以绘制这颗行星了。

下面简要总结了绘制太阳和行星需要使用的函数，完整的程序见示例程序 3-6。

```
1.    glPushMatrix();
2.    glutWireSphere(1.0, 20, 16); /* draw sun */
3.    glRotatef((GLfloat) year, 0.0, 1.0, 0.0);
4.    glTranslatef(2.0, 0.0, 0.0);
5.    glRotatef((GLfloat) day, 0.0, 1.0, 0.0);
6.    glutWireSphere(0.2, 10, 8); /* draw smaller planet */
7.    glPopMatrix();
```

示例程序 3-6 行星系统：planet.c

```
1.    static int year =0,day =0;
2.    void init(void)
3.    {
4.    glClearColor(0.0,0.0,0.0,0.0);
5.    glShadeModel(GL_FLAT);
6.    }
7.    void display(void)
```

```
{
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0,1.0,1.0);
glPushMatrix();
glutWireSphere(1.0,20,16); /*draw sun */
glRotatef((GLfloat)year,0.0,1.0,0.0);

glTranslatef(2.0,0.0,0.0);
glRotatef((GLfloat)day,0.0,1.0,0.0);
glutWireSphere(0.2,10,8); /*draw smaller planet */
glPopMatrix();
glutSwapBuffers();
}
void reshape(int w,int h)
{
glViewport(0,0,(GLsizei)w,(GLsizei)h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(60.0,(GLfloat)w/(GLfloat)h,1.0,20.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(0.0,0.0,5.0,0.0,0.0,0.0,0.0,1.0,0.0);
}
void keyboard(unsigned char key,int x,int y)
{
switch (key){
case 'd ':
day =(day +10)%360;
glutPostRedisplay();
break;
case 'D ':
day =(day -10)%360;
glutPostRedisplay();
break;
case 'y ':
year =(year +5)%360;
glutPostRedisplay();
break;
```

```
46.  case 'Y ':
47.  year =(year -5)%360;
48.  glutPostRedisplay();
49.  break;
50.  default:
51.  break;
52.  }
53.  }
54.  int main(int argc,char**argv)
55.  {
56.  glutInit(&argc,argv);
57.  glutInitDisplayMode(GLUT_DOUBLE |GLUT_RGB);
58.  glutInitWindowSize(500,500);
59.  glutInitWindowPosition(100,100);
60.  glutCreateWindow(argv [0]);
61.  init();
62.  glutDisplayFunc(display);
63.  glutReshapeFunc(reshape);
64.  glutKeyboardFunc(keyboard);
65.  glutMainLoop();
66.  return 0;
67.  }
```

尝试一下

试着在行星系统中增加一颗卫星，或者增加几颗卫星以及另外一颗行星。提示：使用glPushMatrix()和 glPopMatrix()在适当的时候保存和恢复坐标系统的位置。如果打算绘制几颗卫星绕同一颗行星旋转，需要在移动每颗卫星的位置之前保存坐标系统，并且在绘制每颗卫星之后恢复坐标系统。

尝试把行星的轴倾斜。

3.8.2 创建机器人手臂

本节讨论一个创建带关节的机器人手臂的程序。这个手臂在肩、肘或其他关节处应该用节点进行连接。图 3-25 显示了一个带关节的手臂。

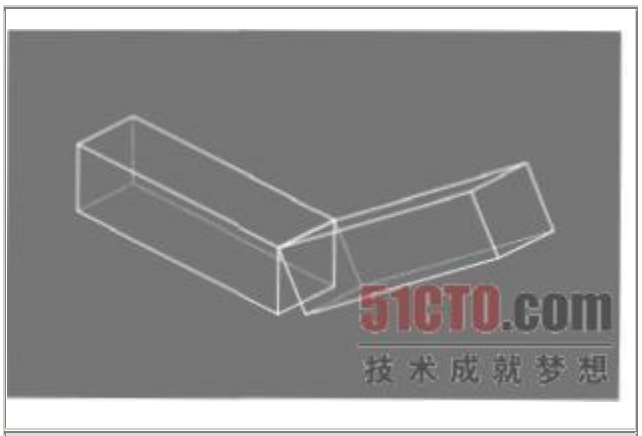


图 3-25 机器人手臂

可以使用一个经过缩放的立方体作为机器人手臂的一段。首先必须调用适当的模型变换函数，把手臂的每一段放在适当的位置。由于局部坐标系统的原点最初位于立方体的中心，因此需要把局部坐标系统移动到立方体的其中一条边上。否则，立方体将沿着它的中心旋转，而不是沿节点旋转。

调用 glTr a n s l a t e * ( )函数建立了节点并调用 glRotate*()函数使立方体绕节点旋转之后，需要把局部坐标系统移回到立方体的中心。然后，在绘制立方体之前对它进行缩放（压扁并拉宽）。可以使用 glPushMatrix()和 glPopMatrix()函数限制 glScale*()函数的效果。下面是创建第一段手臂时可以使用的代码，完整的程序见示例程序 3-7。

```
1.   glTranslatef(-1.0, 0.0, 0.0);
2.   glRotatef((GLfloat) shoulder, 0.0, 0.0, 1.0);
3.   glTranslatef(1.0, 0.0, 0.0);
4.   glPushMatrix();
5.   glScalef(2.0, 0.4, 1.0);
6.   glutWireCube(1.0);
7.   glPopMatrix();
```

为了创建第二段手臂，需要把局部坐标系统移动到下一个节点。由于坐标系统此前已经进行了旋转，x 轴已经与经过旋转的手臂的长边方向对齐。因此，可以沿 x 轴把局部坐标系统移动到下一个节点。到达这个节点之后，使用和绘制第一段手臂相同的代码绘制手臂的第二段。按照这种方法，可以继续绘制更多的手臂段（肩、肘、腕、指等）。

```
1.   glTranslatef(1.0, 0.0, 0.0);
2.   glRotatef((GLfloat) elbow, 0.0, 0.0, 1.0);
3.   glTranslatef(1.0, 0.0, 0.0);
4.   glPushMatrix();
5.   glScalef(2.0, 0.4, 1.0);
6.   glutWireCube(1.0);
```

```
7.    glPopMatrix();
```

示例程序 3-7 机器人手臂：robot.c

```
1.    static int shoulder =0,elbow =0;
2.    void init(void)
3.    {
4.    glClearColor(0.0,0.0,0.0,0.0);
5.    glShadeModel(GL_FLAT);
6.    }
7.    void display(void)
8.    {
9.    glClear(GL_COLOR_BUFFER_BIT);
10.   glPushMatrix();
11.   glTranslatef(-1.0,0.0,0.0);
12.   glRotatef((GLfloat)shoulder,0.0,0.0,1.0);
13.   glTranslatef(1.0,0.0,0.0);
14.   glPushMatrix();
15.   glScalef(2.0,0.4,1.0);
16.   glutWireCube(1.0);
17.   glPopMatrix();
18.   glTranslatef(1.0,0.0,0.0);
19.   glRotatef((GLfloat)elbow,0.0,0.0,1.0);
20.   glTranslatef(1.0,0.0,0.0);
21.   glPushMatrix();
22.   glScalef(2.0,0.4,1.0);
23.   glutWireCube(1.0);
24.   glPopMatrix();
25.   glPopMatrix();
26.   glutSwapBuffers();
27.   }
28.   void reshape(int w,int h)
29.   {
30.   glViewport(0,0,(GLsizei)w,(GLsizei)h);
31.   glMatrixMode(GL_PROJECTION);
32.   glLoadIdentity();
33.   gluPerspective(65.0,(GLfloat)w/(GLfloat)h,1.0,20.0);
34.   glMatrixMode(GL_MODELVIEW);
35.   glLoadIdentity();
```

```
glTranslatef(0.0,0.0,-5.0);
}
void keyboard(unsigned char key,int x,int y)
{
switch (key){
case 's ' :/*s key rotates at shoulder */
shoulder =(shoulder +5)%360;
glutPostRedisplay();
break;
case 'S ':
shoulder =(shoulder -5)%360;
glutPostRedisplay();
break;
case 'e ' :/*e key rotates at elbow */
elbow =(elbow +5)%360;
glutPostRedisplay();
break;
case 'E ':
elbow =(elbow -5)%360;
glutPostRedisplay();
break;
default:
break;
}
}
int main(int argc,char**argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_DOUBLE |GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(100,100);
glutCreateWindow(argv [0]);
init();
glutDisplayFunc(display);
glutReshapeFunc(reshape);
glutKeyboardFunc(keyboard);
glutMainLoop();
return 0;
```

```
74.  }
```

尝试一下

修改示例程序 3-7，在机器人手臂上再增加几段。

修改示例程序 3-7，在相同的位置上增加几段。例如，在机器人的腕部增加几个“手指”，如图 3-26 所示。提示：使用 glPushMatrix()和 glPopMatrix()保存手腕处坐标系统的位置和方向。如果打算绘制手指，需要在设置每个手指的位置之前保存当前矩阵，并在画完每个手指之后恢复当前矩阵。

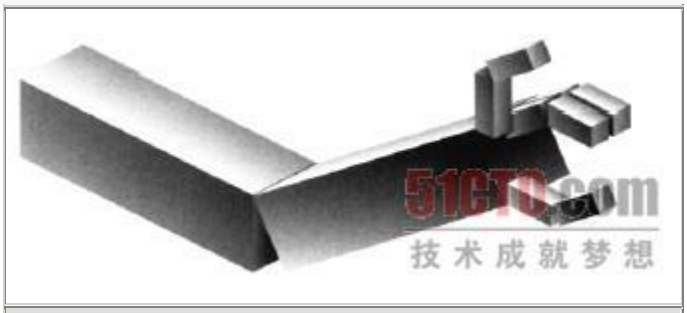


图 3-26 带手指的机器人手臂

Nate Robin 的变换教程

如果已经下载了 Nate Robin 的教学程序包，可以回到“transformation”教程，并再次运行它。可以使用弹出菜单修改 glRotate*()和 glTranslate()的顺序，并注意交换这些函数调用的效果。

3.9 逆变换和模拟变换

几何处理管线擅长使用视图和投影矩阵以及用于裁剪的视口把顶点的世界（物体）坐标变换为窗口（屏幕）坐标。但是，在有些情况下，需要反转这个过程。一种常见的情形就是应用程序的用户利用鼠标选择了三维空间中的一个位置。鼠标只返回一个二维值，也就是鼠标光标的屏幕位置。因此，应用程序必须反转变换过程，确定这个屏幕位置源于三维空间的什么地方。

OpenGL 工具函数库中 gluUnProject()和 gluUnProject4()函数用于执行这种逆变换操作。只要提供一个经过变换的顶点的三维窗口坐标以及所有对它产生影响的变换，gluUnProject()就可以返回这个顶点的源物体坐标。如果深度范围不是默认的[0, 1]，应该使用 gluUnProject4()函数。

```
1.   int gluUnProject(GLdouble winx, GLdouble winy, GLdouble winz,
2.   const GLdouble modelMatrix[16],
3.   const GLdouble projMatrix[16],
4.   const GLint viewport[4],
5.   GLdouble *objx, GLdouble *objy, GLdouble *objz);
```