**积累缓冲区：**

与RGBA模式下的颜色缓冲区一样，累积缓冲区也用于存储RGBA颜色数据（在颜色索引模式下使用累积缓冲区的结果是未定义的）。累积缓冲区通常用于把一系列的图像合成为一幅图像。通过这种方法，可以对图像进行超量采样，然后对样本求平均值，并将结果写入到颜色缓冲区中，从而实现场景的抗锯齿处理。我们不能把数据直接写入到累积缓冲区，累积操作总是以矩形块为单位对数据进行操作，冗长是把数据移入或移出颜色缓冲区

累积缓冲区通过void glAccum(GLenum op, GLfloat value)控制，第一个参数表示对累积缓冲区所进行的操作，第二个参数是浮点数用于指定缩放因子

OpenGL支持的累积缓冲区的操作如下表：

GL\_ACCUM 将颜色缓冲区的颜色值进行放缩后，累加到累积缓冲区

GL\_LOAD 将颜色缓冲区的颜色值进行放缩后，替换掉累积缓冲区的颜色值

GL\_RETURN 将累积缓冲区的颜色值进行放缩后，拷贝回颜色缓冲区

GL\_MULT 将累积缓冲区的颜色值进行放缩后，替换掉原累积缓冲区的颜色值

GL\_ADD 将累积缓冲区的颜色值进行放缩后，累加到累积缓冲区

由于累积缓冲区会带来大内存的开销，因此在实时应用程序中比较少用，但在非实时应用程序中，可以产生实时应用程序无法做到的效果。例如，你可以多次渲染场景，并在每次渲染时进行抖动零点几个像素，这样就可以产生整个场景的反走样效果，比多重采样的效果还要好。还可以模糊前景或背景，然后清洗的渲染一个物体来模拟照相机景深的效果

void drawMotionBlur()

{

GLfloat rot = 35.0f;

GLfloat pass = 10.0f;

for(GLsizei i = 0; i < 10; i++)

{

rot += 0.75f;

drawGemerotry(rot);

// 颜色缓冲区复制到累积缓冲区

if(i == 0)

{

glAccum(GL\_LOAD, 0.5f);

}

else

{

// 颜色缓冲区累加到累积缓冲区

glAccum(GL\_ACCUM, (0.5f \* 1 / pass));

}

}

// 累积缓冲区复制到颜色缓冲区，在之后会调用glutSwapBuffers进行缓冲区的交换

glAccum(GL\_RETURN, 1.0f);

}

void drawGemerotry(GLfloat rot)

{

glPushMatrix();

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

drawGround();

glColor3f(1.0f, 0.0f, 0.0f);

glTranslatef(0.0f, 0.3f, -3.5f);

glRotatef(-rot \* 2.0f, 0.0f, 1.0f, 0.0f);

glTranslatef(1.0f, 0.0f, 0.0f);

glutSolidSphere(0.1f, 17, 13);

}

glPopMatrix();

}

void drawGround()

{

GLsizei colorIndex = 0;

GLfloat range = 20.0f;

GLfloat step = 0.5f;

GLfloat zMin = 0.0f, zMax = 40.0f;

glShadeModel(GL\_SMOOTH);

glEnable(GL\_DEPTH\_TEST);

for(GLfloat x = -range; x <= range; x += step)

{

colorIndex = 0;

glBegin(GL\_LINE\_STRIP);

for(GLfloat z = zMin; z <= zMax; z += step)

{

if(colorIndex % 2 == 0)

{

glColor4f(1.0f, 1.0f, 1.0f, 1.0f);

}

else

{

glColor4f(0.0f, 0.0f, 0.0f, 1.0f);

}

glVertex3f(x, 0, -z);

colorIndex++;

}

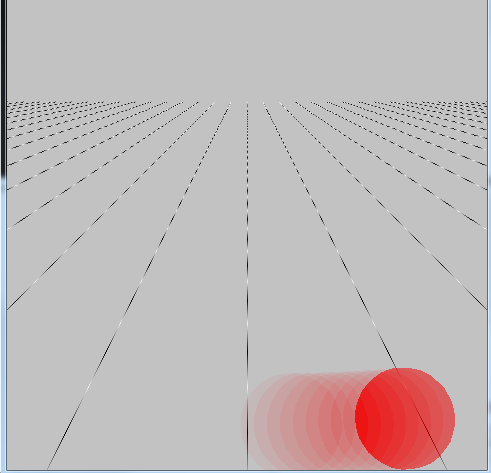
glEnd();

}

}

注意：一定要开启累积缓冲区

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_DEPTH | GLUT\_ACCUM | GLUT\_RGBA);



除了颜色、模板和深度缓冲区外，opengl还支持累积缓冲区。这个缓冲区允许在渲染到颜色缓冲区之后不是把结果显示到窗口，而是把颜色缓冲区的内容复制到累积缓冲区。Opengl所支持的一些复杂操作允许我们按照不同的方法将颜色缓冲区的内容与累积缓冲区的内容进行反复混合（这也是累积缓冲区名字的由来）。完成一幅图像的累积之后，就可以把累积缓冲区的内容复制回颜色缓冲区，并执行缓冲区交换以显示结果。

累积缓冲区是为合成多幅图像而设计的，它不是简单的用引入像素片元来代替像素值，而是将片元进行缩放，然后加到已有的像素值上。

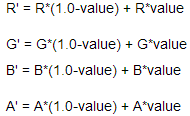
我们可以像其他缓存一样清空累积缓存，可以用glClearAccum()来设置红、绿和蓝色分量的清空值，按位顺序清空累积或以GL\_ACCUM\_BUFFER\_BIT调用，你不能直接渲染进累积缓存，而是应该渲染到一个选定的缓存，然后用glAccum来将在那缓存中的当前图像累积进累积缓存。glAccum用当前选择的读取缓存

累积缓存主要是为了辅助累积计算。例如：运动模糊和全局反走样

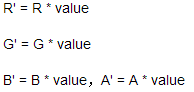
Void glAccum(GLenumm op, GLfloat value)

第一个参数指定了希望使用哪种累积操作，第二个参数是个浮点值，用于对操作进行缩放

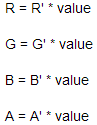
GL\_ACCUM：根据参数值对颜色缓冲区的值进行缩放，并将它们添加到累积缓冲区的当前 内容中



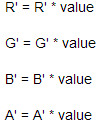
GL\_LOAD：根据参数值对颜色缓冲区的值进行缩放，并将它们替换累积缓冲区的当前内容



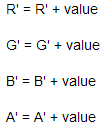
GL\_RETURN：根据累积缓冲区的值对颜色进行缩放，然后把结果复制到颜色缓冲区



GL\_MULT：根据累积缓冲区的值对颜色进行缩放，然后把结果存储到累积缓冲区



GL\_ADD：根据累积缓冲区的值对颜色值进行缩放，然后把结果添加到累积缓冲区的当前内容上



由于累积缓冲区操作必须复制大量的内存并进行大量的处理，所以实时应用极少使用这个功能。对于非实时的渲染，opengl可以产生一些令人惊异的效果。

