OpenGL

glut 函数详解

云南大学信息学院

袁国武 整理

2008 年 5 月 10 日

OpenGL glut 函数详解

目　录

glut 函数详解(1)--glut 初始化 API

(1) void glutInit(int *argc, char **argv);

这个函数用来初始化 GLUT 库.这个函数从 main 函数获取其两个参数.对应 main 函数的形式应是：int main(int argc,char* argv[]);

(2) void glutInitDisplayMode(unsigned int mode);

设置图形显示模式.参数 mode 的可选值为:

GLUT_RGBA:当未指明 GLUT—RGBA 或 GLUT—INDEX 时,是默认使用的模式.表明欲建立 RGBA 模式的窗口.

GLUT_RGB:与 GLUT-RGBA 作用相同.

GLUT_INDEX:指明为颜色索引模式.

GLUT_SINGLE:只使用单缓存

GLUT_DOUBLE:使用双缓存.以避免把计算机作图的过程都表现出来,或者为了平滑地实现动画.

GLUT_ACCUM:让窗口使用累加的缓存.

GLUT_ALPHA:让颜色缓冲区使用 alpha 组件.

GLUT_DEPTH:使用深度缓存.

GLUT_STENCIL:使用模板缓存.

GLUT_MULTISAMPLE:让窗口支持多例程.

GLUT_STEREO:使窗口支持立体.

GLUT_LUMINACE:luminance 是亮度的意思.但是很遗憾,在多数 OpenGL 平台上,不被支持.

(3) void glutInitWindowPosition(int x, int y);

设置初始窗口的位置(窗口左上角相对于桌面坐标(x,y))

(4 void glutInitWindowSize(int width, int height);

设置初始窗口的大小

(5) void glutMainLoop(void);

让 glut 程序进入事件循环.在一个 glut 程序中最多只能调用一次,且必须调用.一旦调用,会直到程序结束才返回.

(6) void glutInitDisplayString(const char *string);

通过一个字符串初始化 display mode
参数:string:display mode 的描述字符串
这个描述字符串用在创建顶级窗口,子窗口和重叠层时,给将要被创建的窗口或重叠层设置 display mode.
这个字符串是由 0 个或多个功能描述参数组成,每个功能描述参数用空格或 tab 键隔开.(若未给参数限制缓存精度大小,则采用默认值)
例如:
glutInitDisplayString("stencil~2 rgb double depth>=16 samples");
上例将窗口初始化为至少 2 位的模板缓存,RGB模式alpha占用位数为 0,深度缓存至少为 16 位,如果平台支持的话使用mutlisampling技术.
可以使用的符号有:
= 等号
!= 不等号

< 小于号(越小越好)

> 大于号(越大越好)

<= 小于等于(越小越好)

>= 大于等于(尽可能选择大的数值,主要用于颜色缓存或深度缓存等对位数要求高的参数设置)

~ 大于等于(但尽可能选择小的数值,有效利用资源,主要用于模板缓存等)

主要参数:

alpha : alpaha 缓存精度, 默认值>=1,即大于等于 1 位;

acca : red, green, blue, 和 alpha 累积缓存精度, 默认值>=1

acc : red, green, blue 累积缓存精度, 默认值>=1,alpha 累积缓存精度为 0;

blue : blue 颜色缓存精度, 默认值>=1;

buffer: 颜色索引缓存精度, 默认值>=1;

conformant : 布尔值,指示帧缓存配置是否一致,该值基于 GLX 的 EXT_visual_rationg 扩展的支持,若不支持,则默认为一致, 默认值=1;

depth : 深度缓存精度, 默认值>=12;

double: 布尔值,指示颜色缓存是否是双倍缓存. 默认值=1;

green : green 颜色缓存精度,默认值>=1;

index : 布尔值,指示是否为颜色索引,true 表示是颜色索引, 默认值>=1;

num : 专用名词,指示数值表示的第 n 个帧缓存配置与这个描述字符串相符合的地方,当没有指定,则 glutInitDisplayString 也返回初始(最佳符合)配置.

red : red 颜色缓存精度, 默认值>=1;

rgba : rgba 模式,颜色缓存精度默认值>=1;

rgb : rgb 模式,颜色缓存精度默认值>=1,alpha 精度=0;

luminance: 设置 red 颜色缓存精度,默认值>=1,其他颜色缓存精度=0(alpha 没有指定);

stencil: 模板缓存精度

single: 布尔值,指示颜色缓存是否为单缓存, 默认值=1;

stereo :布尔值,标示颜色缓存支持 OpenGL 的三维系统,默认值=1;

samples: 标示 multisamples 的值,这个值是基于 GLX 的 SGIS_multisample 的扩展.

默认值<=4.这个默认值表示如果支持的话可以在 glutInitDisplayString 中添加描述参数让 glut 请求 multipsampling;

slow : 布尔值,标示帧缓存配置是否是 slow 的.对于 X11 对 glut 的执行,slow 信息是基于 GLX 的 EXT_visual_rating 扩展的支持,如果不支持,就认为是 fast;对于 win32 对 glut 的执行,slow 是基于像素格式描述(pixel format Descriptor 即 PFD)被标记为"generic"并且未被标记为"accelerated",这说明 Microsoft 关于 slow 的 OpenGL 执行只用在 PFD 中.这个参数的作用是帮助程序避免对于当前机器的帧缓存配置越来越慢.默认值>=0,表示 slow visuals 优先于 fast visuals,但 fast visuals 仍然被允许.

win32pdf : 只在 win32 系统中识别 glut 的请求,这个参数与 win32 中的像素格式(pixel format)相匹配,它的值是个数字

xvisual: 只在 X Window 系统中识别 glut 的请求,这个参数与 X visual ID 相匹配,它的值是个数字

xstaticgray: 只在 X Window 系统中识别 glut 请求,是个布尔值,标示帧缓存配置的 X visual 是否是 StaticGray. 默认值=1

xgrayscale : 只在 X Window 系统中识别 glut 的请求,是个布尔值,标示帧缓存配置的 X visual 是否是 GrayScale. 默认值=1;

xstaticcolor: 只在 X Window 系统中识别 glut 的请求,是个布尔值,标示帧缓存配置的 X visual 是否是 StaticColor. 默认值=1;

xpseudocolor : 只在 X Window 系统中识别 glut 的请求,是个布尔值,标示帧缓存配置的 X visual 是否是 PsuedoColor. 默认值=1;

xtruecolor : 只在 X Window 系统中识别 glut 的请求,是个布尔值,标示帧缓存配置的 X visual 是否是 TrueColor. 默认值=1;

xdirectcolor : 只在 X Window 系统中识别 glut 的请求,是个布尔值,标示帧缓存配置的 X visual 是否是 DirectColor. 默认值=1;

glut 函数详解(2)--窗口 API

(1) int glutCreateWindow(const char *title);

产生一个顶层的窗口.title 作为窗口的名字,也就是窗口标题栏显示的内容.
返回值是生成窗口的标记符,可用函数 glutGetWindow()加以引用.

(2) int glutCreateSubWindow(int win, int x, int y, int width, int height);

创建一个子窗口.win 是其父窗口的标记符.x,y 是子窗口左上角相对父窗口的位移,以像素表示.width,height 是子窗口的宽和高.

(3) void glutDestroyWindow(int win);

销毁窗口,win 是所要销毁的窗口的标识符.这个函数销毁指定的窗口以及与窗口关联的 OpenGL 上下文,重叠层,子窗口等一切与此窗口相关的内容.

(4) void glutPostRedisplay(void);

标记当前窗口的图像层需要重新绘制,在 glutMainLoop 函数的事件处理循环的下一个循环中,将调用该窗口的显示回调函数重绘该窗口的图像层.

(5) void glutPostWindowRedisplay(int win);

标记指定的窗口需重绘,在 glutMainLoop 函数的事件处理循环的下一个循环中,将调用该窗口的显示回调函数重绘该窗口的图像层.

参数:win:需刷新的窗口标识符

(6) void glutSwapBuffers(void);

当窗口模式为双缓存时,此函数的功能就是把后台缓存的内容交换到前台显示.当然,只有单缓存时,使用它的功能跟用 glFlush()一样.而使用双缓存是为了把完整图画一次性显示在窗口上,或者是为了实现动画.

(7) int glutGetWindow(void);

返回当前窗口的标识符

(8) void glutSetWindow(int win);

设置标记符为 win 的窗口为当前窗口

(9) void glutSetWindowTitle(const char *title);

设置当前窗口（必须是顶层窗口）的标题,窗口一旦建立后,窗口标题就可以由这个函数来改变

(10) void glutSetIconTitle(const char *title);

设置当前窗口（必须是顶层窗口）图标化时的标题.

(11) void glutPositionWindow(int x, int y);

改变当前窗口的位置：当前窗口是顶层窗口时,x,y 是相对于屏幕的的位移;当前窗口若是子窗口时,x,y 是相对其父窗口原点的位移.

(12) void glutReshapeWindow(int width, int height);

改变当前窗口的大小.width,height 是当前窗口新的宽度和高度值,当然只能是正值.

(13) void glutPopWindow(void);

在同一个父窗口的子窗口中,使当前的子窗口与排列在它前一个位置的子窗***换位置

(14) void glutPushWindow(void);

在同一个父窗口的子窗口中,使当前的子窗口与排列在它后一个位置的子窗***换位置
这两个函数对顶层窗口和子窗口都起作用,但函数的结果并不立即发生,直到下一个事件循环.

(15) void glutIconifyWindow(void);

让当前窗口成为一个图标,也即是最小化,使当前窗口图标化显示.

(16) void glutShowWindow(void);

显示当前窗口（这时它还是可能被其它窗口挡住）.

(17) void glutHideWindow(void);

隐藏当前窗口

(18) void glutFullScreen(void);

把当前窗口用全屏显示,当前窗口是顶层窗口时才有效.

(19) void glutSetCursor(int cursor);

设置当前窗口的光标样式.

参数:

cursor:指定鼠标的形状,为下面的一种值光标的形状

/* Basic arrows. */

GLUT_CURSOR_RIGHT_ARROW

GLUT_CURSOR_LEFT_ARROW

/* Symbolic cursor shapes. */

GLUT_CURSOR_INFO

GLUT_CURSOR_DESTROY

GLUT_CURSOR_HELP

GLUT_CURSOR_CYCLE

GLUT_CURSOR_SPRAY

GLUT_CURSOR_WAIT

GLUT_CURSOR_TEXT

GLUT_CURSOR_CROSSHAIR

/* Directional cursors. */

GLUT_CURSOR_UP_DOWN

GLUT_CURSOR_LEFT_RIGHT

/* Sizing cursors. */

GLUT_CURSOR_TOP_SIDE

GLUT_CURSOR_BOTTOM_SIDE

GLUT_CURSOR_LEFT_SIDE

GLUT_CURSOR_RIGHT_SIDE

GLUT_CURSOR_TOP_LEFT_CORNER

GLUT_CURSOR_TOP_RIGHT_CORNER

GLUT_CURSOR_BOTTOM_RIGHT_CORNER

GLUT_CURSOR_BOTTOM_LEFT_CORNER

/* Fullscreen crosshair (if available). */

GLUT_CURSOR_FULL_CROSSHAIR:

(在全屏模式下使用,等同于 GLUT_CURSOR_CROSSHAIR)

/* Blank cursor. */

GLUT_CURSOR_NONE:不显示鼠标

/* Inherit from parent window. */

GLUT_CURSOR_INHERIT:使用父窗口的鼠标

(20) void glutWarpPointer(int x, int y);

这个函数将鼠标指针移动到一个相对于当前窗口左上角的新的位置,以窗口左上角为原点,右为 X 轴正方向,下为 Y 轴正方向,参数 x,y 是在这个坐标系中的点,可以为负值.如果坐标值超出屏幕可见范围,则将该值强行控制在屏幕可见范围以内.一般情况下这个函数最好不要使用,因为鼠标动作还是留给用户控制比较人性化.

glut 函数详解(3)--重叠层 API

(1) void glutEstablishOverlay(void);

创建当前窗口的重叠层,该重叠层的显示模式由初始显示模式决定,应在调用 glutEstablishOverlay 之前调用,glutInitDisplayMode 函数来设置初始的显示模式.实现重叠层需要硬件的支持,并不是所有的系统都提供该支持,如果系统不支持重叠层,那么对 glutEstablishOverlay 函数的调用就会引起运行错误,GLUT 将终止程序的运行.可以调用 glutLayerGet(GLUT_OVERPLAY_POSSIBLE)来判断系统是否支持在当前窗口中当前的显示模式下创建重叠层.

(2) void glutRemoveOverlay(void);

删除当前层的重叠层

(3) void glutUseLayer(GLenum layer);

改变当前窗口的使用层
参数:layer:指定窗口的使用层,为
GLUT_NORMAL:使用正常的颜色层
GLUT_OVERLAY:使用重叠层

(4) void glutPostOverlayRedisplay(void);

标记当前窗口的重叠层需要重绘

(5) void glutPostWindowOverlayRedisplay(int win);

标记指定窗口的重叠层需要重绘
参数:win:需要重绘的窗口的重叠层的标识符

(6) void glutShowOverlay(void);

显示当前窗口的重叠层

(7) void glutHideOverlay(void);

隐藏当前窗口的重叠层

这两条语句即时执行.注意一下,只有窗口可视时,使用 glutShowOverlay 才能使其覆盖图层可视.当窗口被其他窗口遮挡时,其覆盖图层也被遮挡从而不可视.

glut 函数详解(4)--菜单 API

(1) int glutCreateMenu(void (*func)(int value));

创建一个新的弹出示菜单

参数:

func:形如 void func(int a);当点击菜单时,调用这个回调函数

value:传递给回调函数的数值,它由所选择的菜单条目对应的整数值所决定

这个函数创建一个新的弹出式菜单,并返回一个唯一的标识次菜单的整型标识符,并将新建的弹出菜单与 func 函数关联在一起,这样,当选择此菜单中的一个菜单条目时,调用回调函数 func.

(2) void glutDestroyMenu(int menu);

删除指定的菜单

menu:被删除的菜单的标识符

(3) int glutGetMenu(void);

获取当前菜单的标识符

(4) void glutSetMenu(int menu);

设置为当前菜单

menu:被设置为当前菜单的标识符

(5) void glutAddMenuEntry(const char *label, int value);

添加一个菜单条目

参数:

label:菜单条目的名称

value:传递到菜单处理函数的值,即在 glutCreateMenu 中注册的回调函数

如果用户选择了这个菜单条目,对应的菜单回调函数就被调用,并以 value 值作为传递给此回调函数的参数:

(6) void glutAddSubMenu(const char *label, int submenu);

在当前菜单的底部增加一个子菜单的触发条目

参数:

label:子菜单触发条目的名称

submenu:子菜单的标识符

(7) void glutChangeToMenuEntry(int item, const char *label, int value);

更改当前菜单中指定菜单项

参数:

item:更改菜单项的索引(最顶层的菜单项的索引为 1)

label:菜单条目的名称

value:传递到菜单处理函数的值,即在 glutCreateMenu 中注册的回调函数

(8) void glutChangeToSubMenu(int item, const char *label, int submenu);

将指定的当前菜单中菜单项变为子菜单触发条目

参数:

item:更改菜单项的索引(最顶层的菜单项的索引为 1)

label:子菜单触发条目的名称

submenu:子菜单的标识符

(9) void glutRemoveMenuItem(int item);

删除指定的菜单项

参数:

item:当前菜单中要删除的菜单项的索引(最顶层的菜单项的索引为 1)

(10) void glutAttachMenu(int button);

把当前窗口的一个鼠标按键与当前菜单关联起来

(11) void glutDetachMenu(int button);

解除鼠标按键与弹出式菜单的关联关系

参数:

button:指明何种鼠标按键,可用的符号为以下三种:

GLUT_LEFT_BUTTON 鼠标左键

GLUT_RIGHT_BUTTON 鼠标右键

GLUT_MIDDLE_BUTTON 鼠标中键

glut 函数详解(5)--内建模型 API

(1) void glutWireSphere(GLdouble radius, GLint slices, GLint stacks);

绘制线框球体

(2) void glutSolidSphere(GLdouble radius, GLint slices, GLint stacks);

绘制实心球体

参数:

radius:球体的半径

slices:球体围绕 z 轴分割的数目

stacks:球体沿着 z 轴分割的数目

绘制中心在模型坐标原点,半径为 radius 的球体,球体围绕 z 轴分割 slices 次,球体沿着 z 轴分割 stacks 次

(3) void glutWireCone(GLdouble base, GLdouble height, GLint slices, GLint stacks);

绘制线框圆锥体

(4) void glutSolidCone(GLdouble base, GLdouble height, GLint slices, GLint stacks);

绘制实心圆锥体

参数:

radius:圆锥体的半径

height:圆锥体的高

slices:圆锥体围绕 z 轴分割的数目

stacks:圆锥体沿着 z 轴分割的数目

圆锥体的轴为 z 轴方向,它的基底位于 z=0 的平面内,顶点 z=height,圆锥体围绕 z 轴分割 slices 个数目,沿着 z 轴分割 stacks 个数目.

(5) void glutWireCube(GLdouble size);

绘制线框立方体

(6) void glutSolidCube(GLdouble size);

绘制实心立方体

参数:size:立方体的边长

(7) void glutWireTorus(GLdouble innerRadius, GLdouble outerRadius, GLint sides, GLint rings);

绘制线框圆环

(8) void glutSolidTorus(GLdouble innerRadius, GLdouble outerRadius, GLint sides, GLint rings);

绘制实心圆环

参数:

innerRadius:圆环的内半径

outerRadius:圆环的外半径

nsides:圆环腔的分割数

rings:圆环的环线数

(9) void glutWireDodecahedron(void);

绘制线框十二面体

(10) void glutSolidDodecahedron(void);

绘制实心十二面体

(11) void glutWireTeapot(GLdouble size);

绘制线框茶壶

(12) void glutSolidTeapot(GLdouble size);

绘制实心茶壶

参数:size:茶壶的相对大小

(13) void glutWireOctahedron(void);

绘制线框八面体

(14) void glutSolidOctahedron(void);

绘制实心八面体

(15) void glutWireTetrahedron(void);

绘制线框四面体

(16) void glutSolidTetrahedron(void);

绘制实心四面体

(17) void glutWireIcosahedron(void);

绘制线框二十面体

(18) void glutSolidIcosahedron(void);

绘制实心二十面体

glut 函数详解(6)--颜色索引表管理 API

(1) void glutSetColor(int cell, GLfloat red, GLfloat green, GLfloat blue);

设置当前窗口当前层一个颜色表单元的颜色

参数:

cell:索引值,标记在颜色表中的位置,其值从 0 开始,并且必须小于颜色表的最大单元数.

当前颜色表的大小可通过调用 glutGet(GLUT_WINDOW_COLORMAP_SIZE)获得.

red:红色成分

green:绿色成分

blue:蓝色成分

(2) GLfloat glutGetColor(int cell, int component);

获取指定的颜色索引的颜色值.

参数:

cell:索引值,标记在颜色表中的位置,其值从 0 开始,并且必须小于颜色表的最大单元数.

当前颜色表的大小可通过调用 glutGet(GLUT_WINDOW_COLORMAP_SIZE)获得.

component:选择下列的值:

GLUT_RED:让函数返回该索引的红色成分.

GLUT_GREEN:让函数返回该索引的绿色成分.

GLUT_BLUE:让函数返回该索引的蓝色成分.

(3) void glutCopyColormap(int win);

将逻辑颜色表从指定的窗口拷贝到当前窗口

参数:win:窗口标识符,逻辑颜色表从该窗口拷贝到当前的窗口.

glut 函数详解(7)--字体处理 API

(1) void glutBitmapCharacter(void *font, int character);

绘制一个图像字符

(2) int glutBitmapWidth(void *font, int character);

返回一个图像字符的宽度

参数:

font:要使用的图像字体,如下表所示:

GLUT_BITMAP_8_BY_13:一种固定宽度字体,每个字符都放在一个 8x13 像素的矩形框内

GLUT_BITMAP_9_BY_15:一种固定宽度字体,每个字符都放在一个 9x15 像素的矩形框内

GLUT_BITMAP_TIMES_ROMAN_10:一种 10 点均匀间距的 Times Roman 字体

GLUT_BITMAP_TIMES_ROMAN_24:一种 24 点均匀间距的 Times Roman 字体

GLUT_BITMAP_HELVETICA_10:一种 10 点均匀间距的 Helvetica 字体

GLUT_BITMAP_HELVETICA_12:一种 12 点均匀间距的 Helvetica 字体

GLUT_BITMAP_HELVETICA_18: 一种 18 点均匀间距的 Helvetica 字体 character:绘制的字符的 ASCII 码.

(3) int glutBitmapLength(void *font, const unsigned char *string);

获取一个图像字符串的宽度.

参数:

font:同上.

string:想要测量宽度的字符串.

(4) void glutStrokeCharacter(void *font, int character);

绘制一个图形字符

(5) int glutStrokeWidth(void *font, int character);

返回一个图形字体的宽度

参数:

font:要使用的图形字体,如下表所示:

GLUT_STROKE_ROMAN 一种等间距的 Roman Simplex 字体,仅包括 32 到 127 的 ASCII 字符

GLUT_STROKE_MONO_ROMAN 一种单一间距的 Roman Simplex 字体,仅包括 32 到 127 的 ASCII 字符

Character:绘制的字符的 ASCII 码.

(6) int glutStrokeLength(void *font, const unsigned char *string);

获取一个图形字符串的宽度

参数:

font:同上.

string:想要测量宽度的字符串.

glut 函数详解(8)--调试 API

(1) void glutReportErrors(void);

这个函数打印出 OpenGL 的运行时错误,它应该只被用在 debug 中,因为它的调用会降低 OpenGL 的速度,它所做的仅仅是不断调用 glGetError()直到没有错误产生.每一个错误都以 GLUT warning 形式报告,并由 gluErrorString()产生相应的错误信息.

glut 函数详解(9)--回调 API

(1) void glutDisplayFunc(void (*func)(void));

注册当前窗口的显示回调函数

参数:

func:形为 void func()的函数,完成具体的绘制操作

这个函数告诉 GLUT 当窗口内容必须被绘制时,那个函数将被调用.当窗口改变大小或者从被覆盖的状态中恢复,或者由于调用 glutPostRedisplay()函数要求 GLUT 更新时,执行 func 参数指定的函数.

(2) void glutReshapeFunc(void (*func)(int width, int height));

指定当窗口的大小改变时调用的函数

参数:

func:形如 void func(int width, int height)

处理窗口大小改变的函数.

width,height:为窗口改变后长宽.

这个函数确定一个回调函数,每当窗口的大小或形状改变时(包括窗口第一次创建),GLUT 将会调用这个函数,这个回调函数接受这个窗口新的长宽作为输入参数.

(3) void glutKeyboardFunc(void (*func)(unsigned char key, int x, int y));

注册当前窗口的键盘回调函数

参数:

func:形如 void func(unsigned char key, int x, int y)

key:按键的 ASCII 码

x,y:当按下键时鼠标的坐标,相对于窗口左上角,以像素为单位

当敲击键盘按键(除了特殊按键,即 glutSpecialFunc()中处理的按键,详见 glutSpecialFunc())时调用.

(4) void glutMouseFunc(void (*func)(int button, int state, int x, int y));

注册当前窗口的鼠标回调函数

参数:

func:形如 void func(int button, int state, int x, int y);

button:鼠标的按键,为以下定义的常量

GLUT_LEFT_BUTTON:鼠标左键

GLUT_MIDDLE_BUTTON:鼠标中键

GLUT_RIGHT_BUTTON:鼠标右键

state:鼠标按键的动作,为以下定义的常量

GLUT_UP:鼠标释放

GLUT_DOWN:鼠标按下

x,y:鼠标按下式,光标相对于窗口左上角的位置

当点击鼠标时调用.

(5) void glutMotionFunc(void (*func)(int x, int y));

当鼠标在窗口中按下并移动时调用 glutMotionFunc 注册的回调函数

(6) void glutPassiveMotionFunc(void (*func)(int x, int y));

当鼠标在窗口中移动时调用 glutPassiveMotionFunc 注册的回调函数

参数:

func:形如 void func(int x, int y);

x,y:鼠标按下式,光标相对于窗口左上角的位置,以像素为单位

(7) void glutEntryFunc(void (*func)(int state));

设置鼠标的进出窗口的回调函数

参数:

func:形如 void func(int state);注册的鼠标进出回调函数

state:鼠标的进出状态,为以下常量之一

GLUT_LEFT 鼠标离开窗口

GLUT_RIGHT 鼠标进入窗口

当窗口取得焦点或失去焦点时调用这个函数,当鼠标进入窗口区域并点击时,state 为 GLUT_RIGHT,当鼠标离开窗口区域点击其他窗口时,state 为 GLUT_LEFT.

(8) void glutVisibilityFunc(void (*func)(int state));

设置当前窗口的可视回调函数

参数:

func:形如 void func(int state);指定的可视回调函数

state:窗口的可视性,为以下常量

GLUT_NOT_VISIBLE 窗口完全不可见

GLUT_VISIBLE 窗口可见或部分可见

这个函数设置当前窗口的可视回调函数,当窗口的可视性改变时,该窗口的可视回调函数被调用.只要窗口中的任何一个像素是可见的,或者他的任意一个子窗口中任意一个像素是可见的,GLUT 则认为窗口是可见的.

(9) void glutIdleFunc(void (*func)(void));

设置空闲回调函数

参数:

func:形如 void func(void);

当系统空闲时调用.

(10) void glutTimerFunc(unsigned int millis, void (*func)(int value), int value);

注册一个回调函数,当指定时间值到达后,由 GLUT 调用注册的函数一次

参数:

millis:等待的时间,以毫秒为单位

unc:形如 void func(int value)

value:指定的一个数值,用来传递到回调函数 func 中

这个函数注册了一个回调函数,当指定的毫秒数到达后,这个函数就调用注册的函数,value 参数用来向这个注册的函数中传递参数. 但只能触发一次,若要连续触发,则需在 func 中重新设置计时函数 glutTimerFunc();

(11) void glutMenuStateFunc(void (*func)(int state));

注册菜单状态回调函数

参数:

func:形如 void func(int state);

state:

GLUT_MENU_IN_USE:菜单被使用.

GLUT_MENU_NOT_IN_USE:菜单不再被使用,即菜单被关闭.

如果 state 代入 GLUT_MENU_IN_USE,则当菜单被使用时调用该函数;

如果 state 代入 GLUT_MENU_NOT_IN_USE,则当菜单被关闭时调用该函数.

(12) void glutMenuStatusFunc(void (*func)(int status, int x, int y));

设置菜单状态回调函数

参数:

func:形如void func(int status, int x, int y);

status:当前是否使用菜单,为以下定义的常量

GLUT_MENU_IN_USE:菜单正在使用

GLUT_MENU_NOT_IN_USE:菜单未被使用

x,y:鼠标按下时,光标相对于窗口左上角的位置

这个函数调用时glut程序判定是否正在使用菜单,当弹出菜单时,调用注册的菜单状态回调函数,同时status设置为常量GLUT_MENU_IN_USE,当菜单使用完毕时,也调用菜单状态回调函数,此时status变量变为GLUT_MENU_NOT_IN_USE.从已弹出的菜单中再弹出的菜单不产生菜单状态回调过程.每个glut程序只有一个菜单状态回调函数. 此函数与上面一个函数相比,只是多传了一个光标位置,其他相同.

(13) void glutSpecialFunc(void (*func)(int key, int x, int y));

设置当前窗口的特定键的回调函数

参数:

Func:形如 void func(int key, int x, int y);

key:按下的特定键,为以下定义的常量

GLUT_KEY_F1:F1 功能键

GLUT_KEY_F2:F2 功能键

GLUT_KEY_F3:F3 功能键

GLUT_KEY_F4:F4 功能键

GLUT_KEY_F5:F5 功能键

GLUT_KEY_F6:F6 功能键

GLUT_KEY_F7:F7 功能键

GLUT_KEY_F8:F8 功能键

GLUT_KEY_F9:F9 功能键

GLUT_KEY_F10:F10 功能键

GLUT_KEY_F11:F11 功能键

GLUT_KEY_F12:F12 功能键

GLUT_KEY_LEFT:左方向键

GLUT_KEY_UP:上方向键

GLUT_KEY_RIGHT:右方向键

GLUT_KEY_DOWN:下方向键

GLUT_KEY_PAGE_UP:PageUp 键

GLUT_KEY_PAGE_DOWN:PageDown 键

GLUT_KEY_HOME:Home 键

GLUT_KEY_END:End 键

GLUT_KEY_INSERT:Insert 键

x,y:当按下键时鼠标的坐标,相对于窗口左上角,以像素为单位

注意:ESC,回车和 delete 键由 ASCII 码产生,即可以用 glutKeyboardFunc()处理. 当在键盘上敲击上述按键时调用该函数.注意与 glutKeyboardFunc()的区别.

(14) void glutSpaceballMotionFunc(void (*func)(int x, int y, int z));

注册一个当前窗口的 spaceball 平移的回调函数

参数:

func:形如 void func(int x, int y, int z);

x,y,z:spaceball 的三维空间坐标.

paceball 即一种特殊的带 3D 滚轮的鼠标,不仅可以前后转动,更可以在三维空间里滚动,具体图片,可以在百度里搜索.

当 spaceball 在当前注册的窗口内平移时,调用该函数.

(15) void glutSpaceballRotateFunc(void (*func)(int x, int y, int z));

注册一个当前窗口的 spaceball 转动的回调函数

参数:

func:形如 void func(int x, int y, int z);

当 spaceball 在当前注册的窗口内滚动时调用.

(16) void glutSpaceballButtonFunc(void (*func)(int button, int state));

注册当前窗口的 spaceball 的按键回调函数.

参数:

func:形如 void func(int button, int state);

button: 按键编号,从 1 开始,可用的按键编号可以通过 glutDeviceGet(GLUT_NUM_SPACEBALL_BUTTONS)查询.

state:按键状态

GLUT_UP:按键释放

GLUT_DOWN:按键按下

当 spaceball 在当前窗口中敲击相应的按键时调用.

(17) void glutButtonBoxFunc(void (*func)(int button, int state));

注册当前窗口的拨号按键盒按键回调函数

参数:

func:形如 void func(int button, int state);

button: 按键编号,从 1 开始,可用的按键号可通过 glutDeviceGet(GLUT_NUM_BUTTON_BOX_BUTTONS)查询

state:按键状态

GLUT_UP:按键释放

GLUT_DOWN:按键按下

当拨号按键盒按键被按下时调用.

(18) void glutDialsFunc(void (*func)(int dial, int value));

注册当前窗口的拨号按键盒拨号回调函数.

参数:

func:形如 void func(int dial, value);

dial:dial 的编号,从 1 开始,可通过 glutDeviceGet(GLUT_NUM_DIALS)查询可用编号.

value:dial 所拨的值,value 是每次所拨的值的累加,直到溢出.

当拨号按键盒拨号时被调用.

(19) void glutTabletMotionFunc(void (*func)(int x, int y));

注册图形板移动回调函数

参数:

func:形如 void func(int x, int y);

x,y:图形板移动的坐标.

当图形板移动时调用.

(20) void glutTabletButtonFunc(void (*func)(int button, int state, int x, int y));

注册当前窗口的图形板按键回调函数

参数:

func:形如 void func(int button, int state, int x, int y);

button:按键号,通过 glutDeviceGet(GLUT_NUM_TABLET_BUTTONS)查询可用键号.

state:按键状态.

GLUT_UP:按键被按下

GLUT_DOWN:按键被释放

x,y:当按键状态改变时,相对于窗口的坐标.

(21) void glutOverlayDisplayFunc(void (*func)(void));

注册当前窗口的重叠层的显示回调函数

参数:

func:形如 void func(void);指向重叠层的显示回调函数.

这个函数告诉 GLUT 当窗口内容必须被绘制时,那个函数将被调用.当窗口改变大小或者从被覆盖的状态中恢复,或者由于调用 glutPostOverlayRedisplay()函数要求 GLUT 更新时,执行 func 参数指定的函数.

(22) void glutWindowStatusFunc(void (*func)(int state));

注册当前窗口状态的回调函数.

参数:

func:形如 void func(int state);

state:窗口状态.

GLUT_HIDDEN:窗口不可见

GLUT_FULLY_RETAINED:窗口完全未被遮挡

GLUT_PARTIALLY_RETAINED:窗口部分遮挡

GLUT_FULLY_COVERED:窗口被全部遮挡

当窗口状态发生相应改变时调用.

(23) void glutKeyboardUpFunc(void (*func)(unsigned char key, int x, int y));

注册释放普通按键的回调函数

参数:

func:形如 void func(unsigned char key, int x, int y);

key:按键的 ASCII 码.

x,y:释放按键时鼠标相对于窗口的位置,以像素为单位.

当普通按键被释放时调用.

(24) void glutSpecialUpFunc(void (*func)(int key, int x, int y));

注册释放特殊按键的回调函数

参数:

func:形如 void func(int key, int x, int y);

key:特殊按键的标识

GLUT_KEY_F1:F1 功能键

GLUT_KEY_F2:F2 功能键

GLUT_KEY_F3:F3 功能键

GLUT_KEY_F4:F4 功能键

GLUT_KEY_F5:F5 功能键

GLUT_KEY_F6:F6 功能键

GLUT_KEY_F7:F7 功能键

GLUT_KEY_F8:F8 功能键

GLUT_KEY_F9:F9 功能键

GLUT_KEY_F10:F10 功能键

GLUT_KEY_F11:F11 功能键

GLUT_KEY_F12:F12 功能键

GLUT_KEY_LEFT:左方向键

GLUT_KEY_UP:上方向键

GLUT_KEY_RIGHT:右方向键

GLUT_KEY_DOWN:下方向键

GLUT_KEY_PAGE_UP:PageUp 键

GLUT_KEY_PAGE_DOWN:PageDown 键

GLUT_KEY_HOME:Home 键

GLUT_KEY_END:End 键

GLUT_KEY_INSERT:Insert 键

x,y:释放特殊按键时鼠标相对于窗口的位置,以像素为单位.

当特殊按键被释放时调用.

(25) void glutJoystickFunc(void (*func)(unsigned int buttonMask, int x, int y, int z), int pollInterval);

注册操纵杆的回调函数

参数:

buttonMask:操纵杆按键

GLUT_JOYSTICK_BUTTON_A

GLUT_JOYSTICK_BUTTON_B

GLUT_JOYSTICK_BUTTON_C

GLUT_JOYSTICK_BUTTON_D

x,y,z:操纵杆在三维空间内移动的位移量

pollInterval:确定检测操纵杆的间隔时间,其单位为毫秒.

该函数在两种情况下被调用:

1.在 pollInterval 所规定的时间间隔内调用.

2.在调用 glutForceJoystickFunc()函数时调用一次 glutJoystickFunc();

glut 函数详解(10)--设备控制 API

(1) void glutIgnoreKeyRepeat(int ignore);

确认是否忽略自动的连续击键(即当一个键被长时间按下不松开时,判断其为一次击键或是多次击键).只对当前窗口有效.对 glutKeyboardFunc()和 glutSpecialFunc()两个回调函数起作用.

参数:

ignore:(相当于布尔值)

0 :不忽略,即认为是连续击键

非 0:忽略,即认为是一次击键

(2) void glutSetKeyRepeat(int repeatMode);

设置自动连续击键模式的状态

参数:repeatMode:

GLUT_KEY_REPEAT_OFF :关闭自动连续击键

GLUT_KEY_REPEAT_ON :打开自动连续击键

GLUT_KEY_REPEAT_DEFAULT:将自动连续击键模式重置为默认状态

注意:这个函数工作在全局范围内,即它会影响所有窗口而不仅仅是当前窗口,所以当关闭了自动连续击键模式后,确保在关闭当前程序销毁前,将自动连续击键模式重置为默认状态.

这个函数的安全性不好,最好使用安全性高的 glutIgnoreKeyRepeat(),尽量将操作限制在当前窗口,而不要轻易使用工作在全局状态下的函数.

(3) void glutForceJoystickFunc(void);

强制调用当前窗口注册的操纵杆回调函数glutJoystickFunc()一次.

glut 函数详解(11)--状态查询 API

(1) int glutGet(GLenum state);

检索指定的 GLUT 状态

参数:

state:指定要检索的状态类型,为以下常量:

GLUT_WINDOW_X:当前窗口的 x 坐标,以像素为单位.

GLUT_WINDOW_Y:当前窗口的 y 坐标,以像素为单位.

GLUT_WINDOW_WIDTH:当其窗口的宽度,以像素为单位.

GLUT_WINDOW_HEIGHT:当前窗口的高度,以像素为单位.

GLUT_WINDOW_BUFFER_SIZE:当前窗口中,颜色分量占用的位数,即用多少 bit 表示颜色分量.

GLUT_WINDOW_STENCIL_SIZE:当前窗口中,蒙板分量占用的位数,即用多少 bit 表示蒙板分量.

GLUT_WINDOW_DEPTH_SIZE:当前窗口中,深度分量占用的位数,即用多少 bit 表示深度分量.

GLUT_WINDOW_RED_SIZE:当前窗口中,红色分量占用的位数,即用多少 bit 表示红色分量.

GLUT_WINDOW_GREEN_SIZE:当前窗口中,绿色分量占用的位数,即用多少 bit 表示绿色分量.

GLUT_WINDOW_BLUE_SIZE:当前窗口中,蓝色分量占用的位数,即用多少 bit 表示蓝色分量.

GLUT_WINDOW_ALPHA_SIZE:当前窗口中,alpha 色分量占用的位数,即用多少 bit 表示 alpha 色分量.

GLUT_WINDOW_ACCUM_RED_SIZE:当前窗口累积缓存中,红色分量占用的位数,即用多少 bit 表示红色分量.

GLUT_WINDOW_ACCUM_GREEN_SIZE:当前窗口累积缓存中,绿色分量占用的位数,即用多少 bit 表示绿色分量.

GLUT_WINDOW_ACCUM_BLUE_SIZE:当前窗口累积缓存中,蓝色分量占用的位数,即用多少 bit 表示蓝色分量.

GLUT_WINDOW_ACCUM_ALPHA_SIZE:当前窗口累积缓存中,alpha 色分量占用的位数,即用多少 bit 表示 alpha 色分量.

GLUT_WINDOW_DOUBLEBUFFER:如果窗口式双缓存模式,返回 1,否则返回 0.

GLUT_WINDOW_RGBA:如果窗口是 RGBA 模式,返回 1,否则返回 0.

GLUT_WINDOW_PARENT:查询当前窗口的父窗口个数,如果为顶层窗口返回 0.

GLUT_WINDOW_NUM_CHILDREN:查询当前窗口的子窗口个数.

GLUT_WINDOW_NUM_SAMPLES:查询多重采样的采样点个数.

GLUT_WINDOW_STEREO:查询是否使用立体模式,是则返回 1,否则返回 0.

GLUT_WINDOW_CURSOR:返回光标的整数标示.

GLUT_SCREEN_HEIGHT:屏幕的高度,以像素为单位.

GLUT_SCREEN_WIDTH:屏幕的宽度,以像素为单位.

GLUT_SCREEN_WIDTH_MM:屏幕的宽度,以毫米为单位.

GLUT_SCREEN_HEIGHT_MM:屏幕的高度,以毫米为单位.

GLUT_MENU_NUM_ITEMS:查询当前菜单包含的菜单项的个数.

GLUT_DISPLAY_MODE_POSSIBLE:查询窗口系统是否支持当前的显示模式,1 表示支持,0 表示不支持.

GLUT_INIT_DISPLAY_MODE:初始窗口的显示模式.

GLUT_INIT_WINDOW_X:初始窗口的 x 坐标.

GLUT_INIT_WINDOW_Y:初始窗口的 y 坐标.

GLUT_INIT_WINDOW_WIDTH:初始窗口的宽度.

GLUT_INIT_WINDOW_HEIGHT:初始窗口的高度.

GLUT_ELAPSED_TIME:返回两次调用 glutGet(GLUT_ELAPSED_TIME)的时间间隔,单位为毫秒,返回值根据查询的内容返回相应的值,无效的状态名返回-1.

GLUT_WINDOW_COLORMAP_SIZE:返回颜色索引表的大小.

(2) int glutDeviceGet(GLenum type);

检索设备信息

参数:

type:要检索的设备信息的名字,为以下常量:

GLUT_HAS_KEYBOARD 如果键盘可用,返回非 0 值,否则,返回 0.

GLUT_HAS_MOUSE 如果鼠标可用,返回非 0 值,否则,返回 0.

GLUT_NUM_MOUSE_BUTTONS 返回鼠标支持的按键数,如果鼠标不可用,返回 0,返回值 0,表示检索的设备不存在,非 0 表示设备可用.

GLUT_HAS_SPACEBALL:如果 spaceball 可用,返回非 0 值,否则,返回 0.

GLUT_HAS_DIAL_AND_BUTTON_BOX:如果拨号按键盒可用,返回非 0 值,否则,返回 0.

GLUT_HAS_TABLET:如果图形板可用,返回非 0 值,否则,返回 0.

GLUT_NUM_SPACEBALL_BUTTONS:返回 spaceball 支持的按键数,如果 spaceball 不存在,返回 0.

GLUT_NUM_BUTTON_BOX_BUTTONS:返回拨号按键盒支持的按键数,如果拨号按键盒不存在,返回 0.

GLUT_NUM_DIALS:返回拨号按键盒支持的拨号数,如果拨号按键盒不存在,返回 0.

GLUT_NUM_TABLET_BUTTONS:返回图形板支持的按键数,如果图形板不存在,返回 0.

GLUT_DEVICE_IGNORE_KEY_REPEAT:如果当前窗口被设置成关闭自动重复按键,则返回非 0 值.

GLUT_DEVICE_KEY_REPEAT:返回自动重复按键的设置状态.(说明文件上说是返回连续击键的速率,其实此说法有误.)

GLUT_HAS_JOYSTICK:如果操纵杆可用,返回非 0 值,否则,返回 0.

GLUT_OWNS_JOYSTICK:如果 glut 认为已经成功获得操纵杆的使用权,则返回非 0 值.否则,返回 0.

GLUT_JOYSTICK_BUTTONS:返回操纵杆支持的按键数,如果操纵杆不存在,返回 0.

GLUT_JOYSTICK_AXES:返回操纵杆支持的操纵轴数,如果操纵杆不存在,返回0.

GLUT_JOYSTICK_POLL_RATE:返回当前窗口操纵杆被拉动的速率.

(注:对操纵杆的查询限制在操纵杆的数量为 1;鼠标被默认为存在,并且其按键数默认是 3.)

(3) int glutExtensionSupported(const char *name);

判定是否支持特定的 OpenGL 扩展

参数:

extension:指定要测试的 OpenGL 扩展的名称,如果给定扩展获得支持,函数返回非0,否则返回 0.

必须要有一个有效的当前窗口来调用 glutExtensionSupported().它只返回 OpenGL 扩展信息,这意味着窗口系统依赖的扩展不被 glutExtensionSupported()反映.

例子:

```
if(!glutExtensionSupported("GL_EXT_texture"))
{
 fprintf(stderr, "Missing the texture extension!");
 exit(1);
}
```

(4) int glutGetModifiers(void);

返回组合功能键的状态

返回值为以下定义的常量

GLUT_ACTIVE_SHIFT:当按下 shift 键时

GLUT_ACTIVE_CTRL:当按下 ctrl 键时

GLUT_ACTIVE_ALT:当按下 alt 键时

(5) int glutLayerGet(GLenum type);

查询属于当前窗口的重叠层的状态

参数:

type:查询的重叠层状态常量:

GLUT_OVERLAY_POSSIBLE:在给定的初始显示模式下,能否为当前窗口创建重叠层.如果能,返回 1;如果不能,返回 0.

GLUT_LAYER_IN_USE:返回当前的使用层,为 GLUT_NORMAL(使用正常的颜色层)或 GLUT_OVERLAY(使用重叠层).

GLUT_HAS_OVERLAY:判断当前窗口是否创建了重叠层.

GLUT_NORMAL_DAMAGED:如果当前窗口的图像层在上一次显示回调函数调用后已经破坏,则返回 TRUE.

GLUT_OVERLAY_DAMAGED:如果当前窗口的重叠层在上一次显示回调函数调用后已经破坏,则返回 TRUE.

GLUT_TRANSPARENT_INDEX:返回当前窗口覆盖层的透明颜色索引值,如果没有覆盖层则返回-1.

glut 函数详解(12)--游戏模式 API

所谓游戏模式其实就是一种全屏模式,这里可以对该模式下的屏幕显示方式进行简单设置.详见下面具体解释.

(1) void glutGameModeString(const char *string);

通过一个字符串对game mode(游戏模式,也即全屏模式)进行设置,即对屏幕进行设置.

参数:

string:一个指向字符串的指针,字符串的内容即是对屏幕的设置.字符串的格式如下所示:

"W*H"

"W*H:Bpp"

"W*H@Rr"

"W*H:Bpp@Rr"

"@Rr"

":Bpp"

"Bpp:@Rr"

(注:W:屏幕宽度,以像素单位;H:屏幕高度,以像素为单位;Bpp:每个像素的内存大小(位数);Rr:屏幕的刷新率.)

例子:

1.如果我们只关心屏幕大小(800*600)而不关心每个像素的内存占用和刷新频率,可以写成:

```
glutGameModeString("800*600");
```

2.如果只想把每个像素的内存占用设置成 32 位,可以写成:

```
glutGameModeString(":32");
```

3.如果只想把刷新率设置成 75 赫兹,可以写成:

```
glutGameModeString("@75");
```

4.如果前三种情况都考虑,可以写成:

glutGameModeString("800*600:32@75");

其他情况按照上面给出的字符串格式写出即可.

注:

1.这个函数只是对硬件的请求,如果设置不合法,则将被忽略.

2.这个函数并不返回错误值,如果要获得错误信息,则要用 glutGameModeGet()函数.

(2) int glutEnterGameMode(void);

进入相应的 game mode,即让 glutGameModeString()的设置生效.

(3) void glutLeaveGameMode(void);

离开 glutGameModeString()设置的 game mode.

(4) int glutGameModeGet(GLenum mode);

检测设置的模式是否有效

参数:

mode:

GLUT_GAME_MODE_ACTIVE:如果程序运行在 game mode,则返回非 0 值,如果运行在窗口模式,则返回 0.

GLUT_GAME_MODE_POSSIBLE:判断 glutAameModeString()的设置是否有效,如果有效则返回非 0 值,否则返回 0.但是 glut 手册中有一个警告,即使这个设置是有效的,也不能保证屏幕设置可以一定成功生效.

GLUT_GAME_MODE_WIDTH:返回屏幕的宽度.

GLUT_GAME_MODE_HEIGHT:返回屏幕的高度.

GLUT_GAME_MODE_PIXEL_DEPTH:返回当前模式下每个像素所占用的内

存空间(位数).

GLUT_GAME_MODE_REFRESH_RATE:返回实际的刷新率(单位赫兹).

GLUT_GAME_MODE_DISPLAY_CHANGED:正如前面所说,不能保证屏幕显示模式一定根据设置发生改变,这个常量可以用来测试是否真的进入了 game mode(先前是窗口模式下的情况下),如果先前已经是 game mode,则可以用来测试设置是否发生改变.返回非 0 值表示进入了 game mode 或设置已经发生改变,否则返回 0.

glut 函数详解(13)--视频大小调整 API

(1) int glutVideoResizeGet(GLenum param);

返回glut视频大小调整的信息.

参数:

param:

GLUT_VIDEO_RESIZE_POSSIBLE:如果底层支持视频大小调整,则返回非 0 值,否则返回 0.如果返回 0,则其他视频大小调整函数的调用将不起作用.

GLUT_VIDEO_RESIZE_IN_USE

GLUT_VIDEO_RESIZE_X_DELTA

GLUT_VIDEO_RESIZE_Y_DELTA

GLUT_VIDEO_RESIZE_WIDTH_DELTA

GLUT_VIDEO_RESIZE_HEIGHT_DELTA

GLUT_VIDEO_RESIZE_X

GLUT_VIDEO_RESIZE_Y

GLUT_VIDEO_RESIZE_WIDTH

GLUT_VIDEO_RESIZE_HEIGHT

后面几个常量值在网上没有搜到解释.

//**********

(2) void glutSetupVideoResizing(void);

(3) void glutStopVideoResizing(void);

(4) void glutVideoResize(int x, int y, int width, int height);

(5) void glutVideoPan(int x, int y, int width, int height);

我把 glut 的头文件中的所有函数都整理了一遍,只可惜 video resize sub-API 中的几个函数网上都没有具体的介绍,本菜鸟也无能为力,幸好这些函数都不是常用的,我想一般用不着.除了和视频有关的 API,前面几个帖子的内容已经基本把 glut 的所有函数都过了一遍,本专题也就算是完成了,以后如果找到相关资料,会把这里的补上,也希望达人能够指点一二.

glut 中去掉控制台

glut 是 OpenGL 应用工具包(OpenGL Utility Toolkit),它为 OpenGL 提供了一个简易的窗口平台,使程序员不用过多的关心与平台相关的窗口系统.

建立工程时,我们一般都建立控制台程序(在 vc6 中即 console win32 application)来使用 glut 库.但是在控制台程序下,运行时都会跳出控制台窗口,很不美观,我们可以用下面的方法将控制台窗口去掉:

```
#pragma comment( linker,"/subsystem:\"windows\" /entry:\"mainCRTStartup\"" )
```

在主函数所在的.cpp 文件的顶部加上上面的宏,就可以轻松将控制台窗口去掉,这样就只剩下 glut 窗口了.

在 vc 中设置 glui

用glut来管理OpenGL窗口确实十分方便,但glut的功能也十分有限,除了弹出菜单以外就没有什么控件了!而基于glut和c++的glui正可以弥补这个缺陷,它其实是GL窗口组件库.目前最新版本是 2.35.

下载地址http://glui.sourceforge.net/#download

下面说说怎么在 vc 中设置 glui:

1.将下载的 glui-2.35.zip 解压

2.打开 msvc 文件夹,找到 glui.dsw 文件,用 vc 打开,链接编译后,会在 msvc 文件夹中生成 Debug 文件夹.

3.在 Debug 文件夹中,找到 glui32.lib,拷贝到 C:\Program Files\Microsoft Visual Studio\VC98\Lib(vc 的默认安装路径)

注:该文件可保留,下次重装系统时可以直接使用而不用重新编译.

4.解压后的 glui-2.35 文件夹中,找到 include 文件夹里的 glui.h,将其拷贝到 C:\Program Files\Microsoft Visual Studio\VC98\Include\GL.至此,glui 设置完成.(不过如果要使用还要设置工程,具体如下)

在解压后的 glui-2.35 文件夹中还有个 example 文件夹,里面是例子,如果直接打开编译就会报错,因为还需要设置工程.选择 Project->settings,在弹出对话框中选择 General 标签,在 Microsoft Foundation Classes 的下拉菜单中选择 Use MFC in a Shared DLL,点击 OK.工程设置完毕,重新编译,一切 OK!!!!!

具体的glui介绍请参看http://www.cs.unc.edu/~rademach/glui/

GLUI 简介(1)--窗体初始化

GLUI 包含三个主要的类:

GLUI_Master_Object

GLUI

GLUI_Control

其中有且只有一个全局的 GLUI_Master_Object 对象 GLUI_Master.

所有的 GLUI 窗口的创建都必须通过这个对象.这可以让 GLUI 通过一个全局对象来追踪所有的窗口,方便窗口的管理.

这里简单介绍一下如何使用 GLUI 创建和控制窗口.这里介绍的函数都属于 GLUI_Master_Object 和 GLUI 类.必须注意,任何 GLUI_Master_Object 类的成员函数都必须通过全局对象 GLUI_Master 来调用,而任何 GLUI 类的成员函数都必须通过 GLUI 指针来调用,并在使用 GLUI 指针之前,GLUI 指针必须获取 GLUI_Master.create_glui()的返回值.

如:

```
float   version = GLUI_Master.get_version();
GLUI   *glui_window = GLUI_Master.create_glui("GLUI");
glui_window->add_StaticText("Hello World!");
```

~~~~~~

初始化:首先介绍使用 GLUI 创建和设置窗口的一下函数.

get_version:

返回当前 GLUI 的版本.

用法:

```
float GLUI_Master_Object::get_version(void)
```

返回值:
~~~~~~

GLUI 版本号

create_glui:

创建一个新的窗口.

用法:

GLUI *GLUI_Master_Object::create_glui(char *name, int flags = 0, int x = -1, int y = -1)

参数:

name:GLUI 窗口的名字.

flags:初始化标记,如果没有给出此参数则默认值为 0,被定义为在当前版本中.

x,y:初始化窗口的坐标.此参数可以不给出,因为 GLUI 可以自动调整窗口大小以适应所有的控件.

返回值:

新的 GLUI 窗口的指针

create_glui_subwindow:

在已经存在的 GLUT 窗口中创建一个新的子窗口

用法:

GLUI *GLUI_Master_Object::create_glui_subwindow(int window, int position)

参数:

window:新建 GLUI 窗口的父窗口(一个已经存在的 GLUT 窗口)的 ID 号.

position:子窗口相对于父窗口的位置,可以为以下的值:

GLUI_SUBWINDOW_RIGHT

GLUI_SUBWINDOW_LEFT

GLUI_SUBWINDOW_TOP

GLUI_SUBWINDOW_BOTTOM

(注:可以在同一个位置创建任意个数的子窗口,多个相同位置的子窗口会简单的相互叠加,如:两个子窗口都使用了 GLUI_SUBWINDOW_TOP 参数,这两

个子窗口都会定位在父窗口之上,同时,第一个子窗口也会覆盖在第二个子窗口之上.)

返回值:

新建的子窗口的指针.

set_glutIdleFunc:

为 GLUI 注册一个标准的 GLUT 空闲回调函数.当 GLUI 处于空闲时,就会调用该注册的函数.GLUI 会截获空闲事件用于自身过程处理,然后才把该事件送给 GLUT 应用程序.需要注意的是,在注册的空闲回调函数中,当前窗口并没有被定义,所以,如果要在空闲回调函数中向 GLUT 窗口获取或发送 redisplay 事件,则必须明确的在回调函数中指定当前窗口.

如:

```
int main_window;
void myGlutIdle(void)//被注册的空闲回调函数
{
  if(glutGetWindow() != main_window)
  {
    glutSetWindow(main_window);
  }
  glutPostRedisplay();
}
```

用法:

void GLUI_Master_Object::set_glutIdleFunc(void (*f)(void))

参数:

f(void):被注册的空闲回调函数.

set_glutReshapeFunc

set_glutKeyboardFunc

set_glutMouseFunc

set_glutSpecialFunc

用法:

void GLUT_Master_Object::set_glutReshapeFunc(void (*f)(int width, int height));

void GLUT_Master_Object::set_glutKeyboardFunc(void (*f)(unsigned char key, int x, int y));

void GLUT_Master_Object::set_glutMouseFunc(void (*f)(int button, int state, int x,int y));

void GLUI_Master_Object::set_glutSpecialFunc(void (*f)(int key, int x, int y));

参数:

详见 glut 函数详解(9)--回调 API

(这里的回调函数与 GLUT 中对应的回调函数用法相似)

set_main_gfx_window:

将一个 GLUT 窗口与一个 GLUI 窗口捆绑,当这个 GLUI 窗口中的一个控件的值发生改变,则该 GLUT 窗口将会被重绘.

用法:

void GLUI::set_main_gfx_window(int window_id);

参数:

window_id:被绑定的 GLUT 窗口 ID,此 ID 号可在 GLUT 窗口被创建时获得(即 glutCreateWindow()的返回值),或通过 glutGetWindow()的返回值获得.

GLUI 简介(2)--窗体视口管理

视口管理:这里介绍使用 GLUI 结合 OpenGL 时,如何管理视口.

~~~~~~

get_viewport_area:

确定当前窗口可绘区域的位置和尺寸.这个函数一般在使用到 GLUI 子窗口时使用,因为子窗口必然会占据父窗口的一小块区域,而绘制在父窗口上的图形并不希望被子窗口覆盖,所以可以通过此函数调整视口的大小.此函数应该在 GLUT 的 reshape callback function 中调用.

用法:

  void GLUI_Master_Object::get_viewport_area(int *x, int *y, int *w, int *h);

参数:

  x,y,w,h:该函数被调用后,就可获得当前窗口可绘区域的左上角坐标及其宽度和高度,然后可以根据这些数据通过调用 glViewport()设置视口区域.

auto_set_viewport:

自动为当前窗口设置尺寸合适的视口

用法:

  void GLUI_Master_Object::auto_set_viewport(void);

例子:

```
int x, y, w, h;
GLUI_Master.get_viewport_area(&x, &y, &w, &h);
glViewport(x, y, w, h);
```

以上三句的功能与下面一句等价.

```
GLUI_Master.auto_set_viewport();
```
~~~~~~

GLUI 简介(3)--窗体管理

窗口管理:当窗体创建之后可以通过下列函数对窗体进行管理

~~~~~~

get_glut_window_id:

返回一个 GLUI 窗口的窗口 ID

用法:

  int GLUI::get_glut_window_id(void);

返回值:

  GLUI 窗口的 ID 号

enable,disable:

使 GLUI 窗口可用或不可用,当一个 GLUI 窗口不可用时,其上的所有控件都不可用.

用法:

  void GLUI::enable(void);

  void GLUI::disable(void);

hide:

使 GLUI 窗口或子窗口隐藏.一个被隐藏的窗口或子窗口不能接受任何用户输入.

用法:

  void GLUI::hide(void);

show:

使一个被隐藏的窗口或子窗口可见.

用法:

  void GLUI::show(void);
~~~~~~

close:

销毁一个 GLUI 窗口或子窗口.

用法:

```
void GLUI::close(void);
```

close_all:

销毁所有的 GLUI 窗口和子窗口.此函数应该被全局对象所调用如:GLUI_Master.close_all();

用法:

```
void GLUI_Master_Object::close_all(void);
```

sync_live:

变量可以与控件相关联,该函数可以使一个 GLUI 窗口上的所有控件和与这些控件相关联的变量保持同步,也即:读取变量的值,然后根据该值设置与其相关联的控件,使该值在控件上反映出来

用法:

```
void GLUI::sync_live(void);
```

sync_live_all:

使所有 GLUI 窗口上的所有控件与与其相关联的变量保持同步,这个函数必须通过全局对象调用,如:GLUI_Master.sync_live_all();

用法:

```
void GLUI_Master_Object::sync_live_all(void);
```

GLUI 简介(4)--控件

GLUI 中,所有的控件都是源于 GLUI_Control 类,所以,他们的操作都非常相似.我们有两种方法创建控件:一种是使用 add_control()直接将控件放在窗口之上,另一种是使用 add_control_to_panel()将控件置于 panel 之内. panel 是一个可以内置其他控件的容器,panel 也可以置于另一个 panel 之内.

这里介绍的函数可以被许多控件调用,用来改变其属性,因此这里介绍的函数可以称为公共函数.

set_name:

为 button,checkbox 等控件设置名字.

用法:

```
void GLUI_Control::set_name(char *name);
```

参数:

name:控件的名字(即:在控件上或控件旁显示的文字)

set_w, set_h:

设置控件的最小宽度或高度

用法:

```
void GLUI_Control::set_w(int new_size);
void GLUI_Control::set_h(int new_size);
```

参数:

new_size:控件的最小宽度或高度.

get, set:

获取或设置控件的当前值.

用法:

```
int GLUI_Control::get_int_val(void);
float GLUI_Control::get_float_val(void);
void GLUI_Control::get_float_array_val(float *float_array_ptr);
char *GLUI_Control::get_text(void);
void GLUI_Control::set_int_val(int int_val);
void GLUI_Control::set_float_val(float float_val);
void GLUI_Control::set_float_array_val(float *float_array_val);
void GLUI_Control::set_text(char *text);
```

(根据控件对输入输出数据值类型的要求,选取相应的函数)

disable, enable:

使控件可用或不可用,radio group 不可用时,其中的 button 也不可用,panel 不可用时,其中的所有控件都不可用.

用法:

```
void GLUI_Control::enable(void);
void GLUI_Control::disable(void);
```

set_alignment:

设置控件的对齐方式(居左,居中,居右)

用法:

```
void GLUI_Control::set_alignment(int align);
```

参数:

align:对齐方式.可选下面之一:

GLUI_ALIGN_CENTER

GLUI_ALIGN_RIGHT

GLUI_ALIGN_LEFT

GLUI 简介(5)--控件

Panels:一个容器,可以内置其他控件,也可以内置另一个 panel.

add_panel:

在 GLUI 窗口上新建一个 panel 控件.

用法:

GLUI_Panel *GLUI::add_panel(char *name, int tyep = GLUI_PANEL_EMBOSSED);

add_panel_to_panel:

在另一个 panel 之内新建一个 panel 控件.

用法:

GLUI_Panel *GLUI::add_panel_to_panel(GLUI_Panel *panel, char *name, int type = GLUI_PANEL_EMBOSSED);

参数:

name:panel 控件的名字(可以为空,如若指定了名字,会在 panel 的左上角显示).

type:panel 的样式.

GLUI_PANEL_EMBOSSED:用内嵌的线条画一个矩形框(默认值).

GLUI_PANEL_RAISED:用外凸的线条画一个矩形框,不显示名字.

GLUI_PANEL_NONE:不绘制矩形框,只用来将控件组织成一个控件组.

panel:指向另一个 panel 控件的指针.新建的 panel 控件将会置于该 panel 之中.

返回值:

新建的 panel 控件的指针.

~~~~~~
~~~~~~

Rollouts:类似于 panel 也是一个容器,功能上可以与 panel 互相替代,不同之处在于该控件可以被折叠起来,此时其内置的控件不可见,只有当其展开后,内置控件才可见.

add_rollout:
在 GLUI 窗口中新建 rollout 控件.
用法:
  GLUI_Rollout *GLUI::add_rollout(char *name, int open = true);

add_rollout_to_panel:
在另一个已经存在的 rollout 或 panel 中新建一个 rollout 控件.
用法:
  GLUI_Rollout *GLUI::add_rollout_to_panel(GLUI_Panel *panel, char *name, int open = true);

参数:
  name:控件的名字.
  open:如果为 true,则 rollout 初始设置为打开;如果为 false,则初始设置为闭合.
  panel:指向另一个 panel 或 rollout 控件的指针.新建的 rollout 控件将会置于该 panel 或 rollout 之中.

返回值:
  新建 rollout 控件的指针.

~~~~~~

Columns:控件在 GLUI 窗口中的布局是按照控件定义的顺序自上而下放置的,在竖直方向上形成一个控件列,而 column 则会开辟一个新的控件列(即在旧的控件列的右侧新建一个新的控件列),其后所定义的控件将置于该新建的控件列中(即
~~~~~~

在新的控件列中自上而下布局),直至新的控件列被创建.

add_column:

在 GLUI 窗口上新建 column.

用法:

```
void GLUI::add_column(int draw_bar = true);
```

add_column_to_panel:

在 panel 中新建 column.

用法:

```
void GLUI::add_column_to_panel(GLUI_Panel *panel, int draw_bar = true);
```

参数:

draw_bar:如果为 true,则在新建控件列时,会绘制一条竖线将其与原先的控件列区分开;如为 false,则只创建控件列,不绘制竖线.

panel:指向一个 panel 控件的指针.新建的 column 控件将会置于该 panel 之中.

~~~~~~

Buttons:按钮

add_button:

在 GLUI 窗口上直接新建按钮.

用法:

```
GLUI_Button *GLUI::add_button(char *name, int id = -1, GLUI_Update_CB callback = NULL);
```

add_button_to_panel:在一个已经存在的 panel 中创建按钮.

用法:

```
GLUI_Button *GLUI::add_button_to_panel(GLUI_Panel *panel,char *name,int id = -1,GLUI_Update_CB callback = NULL);
```
~~~~~~

参数:

name:按钮的名字,即在按钮上显示的文字.

id:按钮的 ID 值.如果 callback 被定义了,则当 callback 被调用时,id 值会作为参数传递给 callback.

callback:接受一个整形参数的 callback 函数.当按钮被触发时,它会被调用.

panel:指向一个 panel 控件的指针.新建的 button 控件将会置于该 panel 之中.

返回值:

新建的按钮控件的指针.

~~~~~~

Checkboxes:复选框

add_checkbox:

在 GLUI 窗口上直接创建新的 checkbox.

用法:

GLUI_Checkbox *GLUI::add_checkbox(char *name, int *live_var = NULL, int id = -1,GLUI_Update_CB callback = NULL);

add_checkbox_to_panel:

在已经存在的 panel 中创建新的 checkbox.

用法:

GLUI_Checkbox *GLUI::add_check_to_panel(GLUI_Panel *panel, char *name, int *live_var = NULL, int id = -1, GLUI_Update_CB callback = NULL);

参数:

name:checkbox 的名字.
~~~~~~

live_var:与控件 checkbox 相关联的整形指针,当 checkbox 控件状态发生变化时,该整形值会自动更新.

id:复选框的 ID 值.如果 callback 被定义了,则当 callback 被调用时,id 值会作为参数传递给 callback.

callback:接受一个整形参数的 callback 函数.当按复选框触发时,它会被调用.

panel:指向一个 panel 控件的指针.新建的 checkbox 控件将会置于该 panel 之中.

返回值:

新建的 checkbox 控件的指针.