```
;/* Engine.c - Execute me to compile me with SAS/C 5.10a
LC -cfistq -v -y -j73 Engine.c
quit ;*/

#include <exec/types.h>
#include <exec/memory.h>
#include <dos/dostags.h>
#include <dos/dos.h>
#include <diskfont/diskfonttag.h>
#include <diskfont/diskfont.h>
#include <diskfont/glyph.h>
#include <diskfont/oterrors.h>
#include <utility/tagitem.h>
#include <string.h>

#include <clib/dos_protos.h>
#include <clib/exec_protos.h>
#include <clib/utility_protos.h>
#include <clib/bullet_protos.h>

#define OTAG_ID 0x0f03   /* this really belongs in <diskfont/diskfont.h>, */
                         /* but it's not there, yet.                      */

extern UBYTE   *librarystring; /* ".library", defined in BulletMain.c. */

struct TagItem *AllocOtag(STRPTR);
void            FreeOtag(void *);
struct Library *OpenScalingLibrary(struct TagItem *);
void            CloseScalingLibrary(struct Library *);
struct GlyphEngine *GetGlyphEngine(struct TagItem *, STRPTR);
void            ReleaseGlyphEngine(struct GlyphEngine *);

#define BUFSIZE     256

extern struct Library *BulletBase, *UtilityBase;

/*****************************************************************************/
/* open the otag file, allocate a buffer, read the file into the buffer, verify that */
/* the file is OK, relocate all of the address relocation tags, close the otag file. */
/*****************************************************************************/
struct TagItem *
AllocOtag(STRPTR otagname)
{
  BPTR            otfile;
  struct TagItem *ti, *tip, *returnti;
  struct FileInfoBlock *fib;

  ti = NULL;

  if (fib = AllocDosObject(DOS_FIB, NULL))    /* The FileInfoBlock of the OTAG file */
  {                                           /* contains the file's size.         */
    if (otfile = Open(otagname, MODE_OLDFILE))
    {
      if (ExamineFH(otfile, fib))
      {
        if (returnti = (struct TagItem *) AllocVec(fib->fib_Size, MEMF_CLEAR))
        {
          if (Read(otfile, (UBYTE *) returnti, fib->fib_Size))
          {
            if ((returnti->ti_Tag == OT_FileIdent)              /* Test to see if */
                && (returnti->ti_Data == (ULONG) fib->fib_Size)) /* the OTAG file  */
            {                                                   /* is valid.      */
              tip = returnti;
              while (ti = NextTagItem(&tip))     /* Step through and relocate tags */
              {
                if (ti->ti_Tag & OT_Indirect)
                {
                  ti->ti_Data = (ULONG) returnti + ti->ti_Data;
                }
              }
            }
          }
        }
      }
      Close(otfile);
    }
    FreeDosObject(DOS_FIB, fib);
  }
  return (returnti);
}

/*****************************************************************************/
/********** Deallocates resources allocated by AllocOtag(). **************/
/*****************************************************************************/
void
FreeOtag(void *vector)
{
  FreeVec(vector);
}


/*****************************************************************************/
/******* Scans through a TagList looking for an scaling engine name. *********/
/*******           If it finds one, it opens that library.           *********/
/*****************************************************************************/
struct Library *
OpenScalingLibrary(struct TagItem * ti)
{
  STRPTR          enginename;
  UBYTE           libnamebuffer[BUFSIZE];

  if (enginename = (STRPTR) GetTagData(OT_Engine, NULL, ti))
  {
    strcpy(libnamebuffer, enginename);
    strcat(libnamebuffer, librarystring);

    return (OpenLibrary(libnamebuffer, 0L));
  }
  return(NULL);  /*   <----- BUG!:--  This line was missing in the original */
}                /*                   publication of this code module. */

/*****************************************************************************/
/******* Deallocates resources allocated by OpenScalingLibrary(). *********/
/*****************************************************************************/
void
CloseScalingLibrary(struct Library * bbase)
{
  CloseLibrary(bbase);
}


/*****************************************************************************/
/* Open the glyph engine, give it the tags from the otag file, and set up */
/* a default device dpi so it doesn't crash if someone forgets to set it. */
/*****************************************************************************/
struct GlyphEngine *
GetGlyphEngine(struct TagItem * ti, STRPTR otagpath)
{
  struct GlyphEngine *ge = NULL;
  BOOL            ok = TRUE;

  if (ge = OpenEngine())
  {
    ok = FALSE;
    if (SetInfo(ge,
                OT_OTagList, ti,
                OT_OTagPath, otagpath,
                TAG_END) == OTERR_Success)
    {
      if (SetInfo(ge,
                  OT_DeviceDPI, ((ULONG) 77) << 16 | 75,
                  TAG_END) == OTERR_Success)
      {
        ok = TRUE;
      }
    }
  }

  if (!ok)
  {
```

```
    CloseEngine(ge);
    ge = NULL;
  }
  return (ge);
}

/*****************************************************************************/
/********* Deallocates resources allocated by GetGlyphEngine(). **********/
/*****************************************************************************/
void
ReleaseGlyphEngine(struct GlyphEngine * ge)
{
  CloseEngine(ge);
}
```

```
;/* BulletMain.c - Execute me to compile me with SAS/C 5.10a
LC -cfistq -v -y -j73 BulletMain.c
quit ;*/

#include <exec/types.h>
#include <exec/memory.h>
#include <dos/rdargs.h>
#include <dos/dos.h>
#include <dos/var.h>
#include <diskfont/diskfonttag.h>
#include <diskfont/diskfont.h>
#include <diskfont/glyph.h>
#include <diskfont/oterrors.h>
#include <utility/tagitem.h>
#include <string.h>
#include <graphics/displayinfo.h>
#include <intuition/intuition.h>
#include <intuition/screens.h>

#include <clib/dos_protos.h>
#include <clib/graphics_protos.h>
#include <clib/exec_protos.h>
#include <clib/utility_protos.h>
#include <clib/bullet_protos.h>
#include <clib/intuition_protos.h>

#define OTAG_ID 0x0f03

#ifdef LATTICE
int CXBRK(void) { return (0);} /* Disable Lattice CTRL/C handling */
int chkabort(void) { return (0);}
#endif

UBYTE   *readargsstring = "FontName,Size/N,XDPI/N,YDPI/N,CharCode/N,CharCode2/N\n";
UBYTE   *librarystring = ".library";
UBYTE   *fontstring     = "fonts:cgtimes.font";
UBYTE   *dpivarname = "XYDPI";  /* Name of an X/Y DPI environment variable. */
                                /* If this ENV: variable exists, this code  */
                                /* will use the X and Y DPI stored there.   */
                                /* This code will also save the X and Y DPI */
                                /* in XYDPI if the user supplies a DPI.     */
                                /* XYDPI encodes the DPI just like the      */
                                /* OT_DeviceDPI tag.                        */
extern struct TagItem *AllocOtag(STRPTR);
extern void     FreeOtag(void *);
extern struct Library *OpenScalingLibrary(struct TagItem *);
extern void     CloseScalingLibrary(struct Library *);
extern struct GlyphEngine *GetGlyphEngine(struct TagItem *, STRPTR);
extern void     ReleaseGlyphEngine(struct GlyphEngine *);
extern void
BulletExample(struct GlyphEngine *,
              struct Window *,
              struct RastPort *, ULONG, ULONG, ULONG, ULONG, ULONG);

#define BUFSIZE     256

#define NUM_ARGS    6    /* Arguments for ReadArgs(). */
#define FONT_NAME   0
#define SIZE        1
#define XDPI        2
#define YDPI        3
#define CODE        4
#define CODE2       5
LONG            args[NUM_ARGS];
struct RDargs  *myrda;

struct Library *BulletBase, *UtilityBase, *GfxBase, *IntuitionBase;

UBYTE           buf[BUFSIZE];
BPTR            fontfile, dpifile;
UBYTE          *otagname;
UWORD           fchid;

struct GlyphEngine *ge;
```

```
struct DrawInfo *drawinfo;
struct RastPort rp;

void            main(int argc, char **argv)
{
  struct TagItem *ti;
  struct GlyphEngine *ge;
  struct Window  *w;

  UBYTE           xydpi[5];

  ULONG           defpointheight = 36;   /* Default values for ReadArgs() */
  ULONG           defxdpi = 68;          /* variables.                    */
  ULONG           defydpi = 27;
  ULONG           defcode = (ULONG) 'A';
  ULONG           defcode2 = 0;

  if (GfxBase = OpenLibrary("graphics.library", 37L))
  {
    if (IntuitionBase = OpenLibrary("intuition.library", 37L))
    {
      if (myrda = ReadArgs(readargsstring, args, NULL))
      {
        if (args[XDPI] && args[YDPI]) /* If the user sets the DPI from the command  */
        {           /* line, make sure the environment variable also gets changed. */
          *(ULONG *)xydpi = ( (*(LONG *) args[XDPI]) << 16 | (*(ULONG *) args[YDPI]) );
          SetVar(dpivarname, xydpi, 5,
              GVF_GLOBAL_ONLY | GVF_BINARY_VAR | GVF_DONT_NULL_TERM);
        }
        else                           /* If the user did NOT set the X OR Y DPI...*/
        {
          args[XDPI] = (LONG) &defxdpi; /* ...set to default values and look for    */
          args[YDPI] = (LONG) &defydpi; /* an environment variable called "XYDPI".  */
                                        /* Read the environment variable, */
          if ((GetVar(dpivarname, xydpi, 5,        /* XYDPI, if it exists.          */
              GVF_GLOBAL_ONLY | GVF_BINARY_VAR | GVF_DONT_NULL_TERM)) != -1)

     /* BUG!: In the original publication of this code, the line above erroneously  */
     /* tested for the wrong return value.  It caused unexpected results when using */
     /* the default X and Y DPI values.  This bug was also present in BulletMain.c. */

          {
            if ( (*(ULONG *)xydpi & 0xFFFF0000) && (*(ULONG *)xydpi & 0x0000FFFF) )
            {     /* Make sure the environment variable is OK to use by making sure */
                  /* that neither X or YDPI is zero. If XYDPI is OK, use it as the  */
              defxdpi = ((*(ULONG *)xydpi) & 0xFFFF0000) >> 16;        /* default.  */
              defydpi = (*(ULONG *)xydpi) & 0x0000FFFF;
            }
          }
        }
        if (!(args[SIZE]))
          args[SIZE] = (LONG) &defpointheight;
        if (!(args[CODE]))
          args[CODE] = (LONG) &defcode;
        if (!(args[CODE2]))
          args[CODE2] = (LONG) &defcode2;
        if (!(args[FONT_NAME]))
          args[FONT_NAME] = (LONG) fontstring;
                                       /* Open the ".font" file which contains  */
                                       /* the FontContentsHeader for this font. */
        if (fontfile = Open((STRPTR) args[FONT_NAME], MODE_OLDFILE))
        {
          if (Read(fontfile, &fchid, sizeof(UWORD)))
          {
            if (fchid == OTAG_ID)             /* Does this font have an .otag file? */
            {
              strcpy(buf, (STRPTR) args[FONT_NAME]); /* Put together the otag file  */
              if (otagname = &(buf[strlen(buf) - 4]))/* name from the .font file.   */
              {
                strcpy(otagname, "otag");
                if (UtilityBase = OpenLibrary("utility.library", 37L))
                {
                  if (ti = AllocOtag(buf))      /* open the otag file and copy its  */
                  {                             /* tags into memory.                */
                    if (BulletBase = OpenScalingLibrary(ti)) /* Pass the function   */
                    {                               /* the OTAG tag list which it*/
                      if (ge = GetGlyphEngine(ti, buf))/* library.  Open the        */
                      {                             /* library's scaling engine. */
                        if (w = OpenWindowTags(NULL,
                                               WA_Width,         640,
                                               WA_Height,        200,
                                               WA_SmartRefresh,  TRUE,
                                               WA_SizeGadget,    FALSE,
                                               WA_CloseGadget,   TRUE,
                                               WA_IDCMP,         NULL,
                                               WA_DragBar,       TRUE,
                                               WA_DepthGadget,   TRUE,
                                               WA_Title,         (ULONG)argv[0],
                                               TAG_END))
                        {
                          rp = *(w->RPort); /* Clone window's RastPort.  The second */
                                            /* Rastport is for rendering with the   */
                                            /* background color.                    */

                          if (drawinfo = GetScreenDrawInfo(w->WScreen)) /* Get the  */
                          {                 /* Screen's DrawInfo to get its pen colors. */
                            SetAPen(w->RPort, drawinfo->dri_Pens[TEXTPEN]);
                            SetAPen(&rp, drawinfo->dri_Pens[BACKGROUNDPEN]);
                            FreeScreenDrawInfo(w->WScreen, drawinfo);

                            BulletExample(ge, w, &rp, *(ULONG *) args[SIZE],
                                          *(ULONG *) args[XDPI],
                                          *(ULONG *) args[YDPI],
                                          *(ULONG *) args[CODE],
                                          *(ULONG *) args[CODE2]);
                          }
                          CloseWindow(w);
                        }
                        ReleaseGlyphEngine(ge);
                      }
                      CloseScalingLibrary(BulletBase);
                    }
                    FreeOtag(ti);
                  }
                  CloseLibrary(UtilityBase);
                }
              }
            }
            Close(fontfile);
          }
          FreeArgs(myrda);
        }
        CloseLibrary(IntuitionBase);
      }
      CloseLibrary(GfxBase);
    }
}
```

```
;/* Rotate.c - Execute me to compile me with SAS/C 5.10a
LC -cfistq -v -y -j73 Rotate.c
Blink FROM LIB:c.o,BulletMain.o,engine.o,Rotate.o TO Rotate LIBRARY
LIB:LC.lib,LIB:Amiga.lib
quit ;*/

#include <exec/types.h>
#include <diskfont/diskfonttag.h>
#include <diskfont/diskfont.h>
#include <diskfont/glyph.h>
#include <diskfont/oterrors.h>
#include <graphics/gfx.h>
#include <graphics/regions.h>
#include <utility/tagitem.h>
#include <intuition/intuition.h>
#include <devices/timer.h>

#include <clib/alib_stdio_protos.h>
#include <clib/alib_protos.h>
#include <clib/bullet_protos.h>
#include <clib/exec_protos.h>
#include <clib/layers_protos.h>
#include <clib/graphics_protos.h>
#include <clib/intuition_protos.h>

extern struct Library *BulletBase, *UtilityBase, *GfxBase, *IntuitionBase;
struct Library *LayersBase;
void            BulletExample(struct GlyphEngine *,
                              struct Window *,
                              struct RastPort *,
                              ULONG, ULONG, ULONG, ULONG, ULONG);

UBYTE          *vers = "\e0$VER: Rotate 38.9";

#define TABLE_ENTRIES 24
#define SINE_INDEX    0
#define COSINE_INDEX  1

/* precalculated sine and cosine */
LONG            table[TABLE_ENTRIES][2] =
{
  {0x0, 0x10000},           /* 0   degrees */ /* Notice that the sine and cosine  */
  {0x424e, 0xf747},         /* 15  degrees */ /* values have to correspond to the */
  {0x8000, 0xddb4},         /* 30  degrees */ /* same angle.  The IntelliFont     */
  {0xb505, 0xb505},         /* 45  degrees */ /* engine will have severe mental   */
  {0xddb4, 0x8000},         /* 60  degrees */ /* problems if the values aren't    */
  {0xf747, 0x424e},         /* 75  degrees */ /* close to representing the same   */
  {0x10000, 0x0},           /* 90  degrees */ /* angle.                           */
  {0xf747, 0xffffbdbe},     /* 105 degrees */
  {0xddb4, 0xffff8000},     /* 120 degrees */
  {0xb505, 0xffff4afb},     /* 135 degrees */
  {0x8000, 0xffff224c},     /* 150 degrees */
  {0x424e, 0xffff08b9},     /* 165 degrees */
  {0x0, 0xffff0000},        /* 180 degrees */
  {0xffffbdbe, 0xffff08b9},/* 195 degrees */
  {0xffff8000, 0xffff224c},/* 210 degrees */
  {0xffff4afb, 0xffff4afb},/* 225 degrees */
  {0xffff224c, 0xffff8000},/* 240 degrees */
  {0xffff08b9, 0xffffbdbe},/* 255 degrees */
  {0xffff0000, 0x0},        /* 270 degrees */
  {0xffff08b9, 0x424e},     /* 285 degrees */
  {0xffff224c, 0x8000},     /* 300 degrees */
  {0xffff4afb, 0xb505},     /* 315 degrees */
  {0xffff8000, 0xddb4},     /* 330 degrees */
  {0xffffbdbe, 0xf747}      /* 345 degrees */
};

struct Rectangle rectangle;
struct Region    *region;

void
BulletExample(struct GlyphEngine * ge,
              struct Window * w, struct RastPort * rp,
     ULONG pointheight, ULONG xdpi, ULONG ydpi, ULONG unicode, ULONG unicode2)
{
  struct GlyphMap *gm;
  PLANEPTR        tempbitmap;
  ULONG           centerx, centery, x, y, dx, dy, sin, cos, oldx, oldy, olddx,
                  olddy, emheight, emwidth;
  ULONG           i = 0;

  struct IntuiMessage *mymsg;
  BOOL            done = FALSE;

  if (pointheight > 180) pointheight = 180;

  if (SetInfo(ge,
              OT_DeviceDPI, ((ULONG) xdpi) << 16 | ydpi,
              TAG_END) != OTERR_Success)
        return;

  emheight = (pointheight * ydpi) / 72; /* Calculate the pixel dimensions */
  emwidth  = (pointheight * xdpi) / 72; /* of the EM square.              */
  centerx  = w->BorderLeft + emheight;
  centery  = w->BorderTop  + emwidth;

  dx = (2 * emwidth) + w->BorderLeft + w->BorderRight;  /* Calculate window size to */
  dy = (2 * emheight) + w->BorderTop + w->BorderBottom; /* fit around glyph com-    */
  dx = (dx > 640) ? 640 : dx;                           /* fortablely.              */
  dy = (dy > 200) ? 200 : dy;
  dx = (dx < 80) ? 80: dx;
  dy = (dy < 50) ? 50: dy;

  if (ModifyIDCMP(w, IDCMP_CHANGEWINDOW))
  {
    ChangeWindowBox(w, w->LeftEdge, w->TopEdge, dx, dy);        /* Set window size  */
    WaitPort(w->UserPort);                                      /* and wait for the */
    while (mymsg = (struct IntuiMessage *) GetMsg(w->UserPort)) /* dimension change */
      ReplyMsg((struct Message *) mymsg);                       /* to take place.   */
    if (!(ModifyIDCMP(w, NULL))) return;  /* Quit if there is a problem with IDCMP. */
  }

  x = centerx; /* calculate original rendering position. */
  y = centery;
  dx = 1;   /* Since dx and dy are no longer necessary for figuring out the window  */
  dy = 1;   /* dimensions, I use them to measure the full width and height of the   */
            /* glyph bitmap supplied by bullet.  I need this to erase the glyph.    */

  if (LayersBase = OpenLibrary("layers.library", 37L)) /* These lines are       */
  {                                                    /* here to install       */
    rectangle.MinX = w->BorderLeft;                    /* a clipping            */
    rectangle.MinY = w->BorderTop;                     /* region to the         */
    rectangle.MaxX = w->Width  - w->BorderRight - 1;   /* window to keep        */
    rectangle.MaxY = w->Height - w->BorderBottom - 1;  /* the glyph within      */
                                                       /* window bounds.        */
    if (region = NewRegion())                          /* For more information, */
    {                                                  /* see the "Layers"      */
      if (OrRectRegion(region, &rectangle))            /* chapter of the        */
      {                                                /* RKRM: Libraries       */
        InstallClipRegion(w->WLayer, region);          /* Manual.               */

        if (SetInfo(ge,
                    OT_GlyphCode, unicode,                      /* Set the glyph to */
                    OT_PointHeight, (ULONG) pointheight << 16,  /* rotate and its   */
                    TAG_END) == OTERR_Success)                  /* pointsize.       */
        {
          SetDrMd(w->RPort, JAM1);
          if (tempbitmap = AllocRaster(640, 200))
          {
            if (ModifyIDCMP(w, IDCMP_CLOSEWINDOW)) /* Turn on close window reports  */
            {                                      /* so the example can quit.      */
              for (i = 0; done == FALSE; i++)
              {
                if (i == TABLE_ENTRIES)
                  i = 0;

                sin = table[i][SINE_INDEX];   /* Step through the sine/cosine array */
                cos = table[i][COSINE_INDEX]; /* 360 degrees @ 15 degree increments */

                if (SetInfo(ge,                 /* Set the current rotation angle. */
                            OT_RotateSin, sin,
                            OT_RotateCos, cos,
                            TAG_END) == OTERR_Success)
                {
```