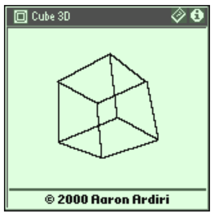
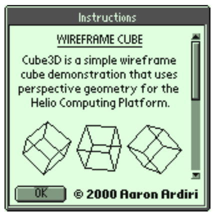
Introduction

Helio Computing Platform

gfx is a replacement graphics application programmer interface (API) for the Helio Computing Platform. The **gfx** library provides developers with optimized graphics routines; low-level window management (off-screen windows, copy routines, clipping) and font management (string management, custom font support).

`Cube3D`, a perspective cube rotation demonstrates how a developer can use the **gfx** library to perform advanced animations and provide a very professional look and feel to their Helio applications. It has been included in the **gfx** distribution (with sources), and is also available at the following website:




Instructions

WIREFRAME CUBE

Cube3D is a simple wireframe cube demonstration that uses perspective geometry for the Helio Computing Platform.

OK © 2000 Aaron Ardiri

Instructions

The source code is available for educational purpose only and shall not be redistributed or modified without the consent of the original author.

Interested in developing for the Helio Computing Platform?

http://www.ardiri.com/

Happy Hacking!

OK © 2000 Aaron Ardiri

http://www.ardiri.com/index.cfm?redir=helio&subcat=cube3D

Using the **gfx** library is simple; include the `gfx.h` header file in your project and link to the `gfx.a` library. The graphics library is compatible with all versions of the vt-os, just write your code, and link it to the `gfx.a` library – and everything should be sweet (see `Cube3D` demo for an example).

Application Programmer Interface

The **gfx** library, although simple, requires a bit of an understanding of some fundamental low-level graphics education in order to use. The following is taken from the `gfx.h` header file, and explained in a bit more detail with examples.

gfx Version

```
#define gfx_beta     0x0000
#define gfx_version1 0x0100

SHORT      GfxGetVersion();
```

In order to maintain compatibility with all third party applications, and since this library is to be improved over time, it is important to keep the version of the graphics library currently available. Future revisions of the **gfx** library will define new version constants that will increase in value. If a routine requires a particular version of the library, this value can determine if it is available or not.

Initialization

```
void       GfxInitialize();
```

The **gfx** library must be initialized before it is used – it is not part of the vt-os, and requires initialization before any graphics routines can actually be used. Failure to initialize the **gfx** library may result in device crashes and instability.

This routine can be placed in the `InitApplication()` function code block.

Windows

```
typedef struct
{
  SHORT width;
  SHORT height;
  WORD  memSize;
  void  *memory;
} GfxWindow;
```

All graphical operations are performed to an off-screen (or LCD) window buffer. The definition is simple, provide the width, height, size of the memory chunk and a reference to the 4bpp bitmap memory chunk, and a window is created.

A window can be manually created as follows:

```
        GfxWindow *window = (GfxWindow *)pmalloc(sizeof(GfxWindow));
        window->width     = bitmap00Width;
        window->height    = bitmap00Height;
        window->memSize   = bitmap00Size;
        window->memory    = (void *)bitmap00;
```

And disposed of as follows:

```
        pfree(window);
```

Once a window is created, various operations can be performed on it. Off-screen windows can be initialized and cleaned up using the following routines. Creating off-screen windows allow for double buffering (animations) and other graphical effects.

```
GfxWindow *GfxCreateWindow(SHORT width,
                           SHORT height);
void       GfxDisposeWindow(GfxWindow *window);
```

These routines perform all the necessary memory allocation and de-allocation. All windows created using the `GfxCreateWindow()` routine should be removed using the `GfxDisposeWindow()` routine.

```
void       GfxSetDrawWindow(GfxWindow *window);
GfxWindow *GfxGetDrawWindow();
GfxWindow *GfxGetDisplayWindow();
```

The **gfx** library determines which window it should perform its operations on by keeping track of the active "draw window". On initialization, the default draw window is the LCD screen itself. This window is been created within the **gfx** library, and can be obtained with the `GfxGetDisplayWindow()` routine.

Window Routines

```
typedef enum
{
  gfxPaint = 0,      // x = y
  gfxMask,           // x = x & ~y
  gfxInvert,         // x = x ^ y
  gfxOverlay         // x = x | y
} GfxDrawOperation;
```

```
typedef struct
{
  SHORT x;
  SHORT y;
} GfxPosition;

typedef struct
{
  GfxPosition topLeft;
  GfxPosition extent;
} GfxRegion;

typedef enum
{
  gfx_white = 0,     // 0000b
  gfx_lgray = 5,     // 0101b
  gfx_dgray = 10,    // 1010b
  gfx_black = 15     // 1111b
} GfxColor;

void       GfxClearWindow(GfxWindow *window);
void       GfxFillRegion(GfxWindow *window, GfxRegion *region, GfxColor color);
```

These routines provide the developer with the ability to fill or clear the contents of a specific graphics window. A region is a rectangle definition, comprising of a top left co-ordinate, and an extent (width, height). Although the **gfx** library supports 16 shades of gray, only four are defined.

```
void       GfxCopyRegion(GfxWindow *srcWin, GfxWindow *dstWin,
                         GfxRegion *region, SHORT x, SHORT y, GfxDrawOperation mode);
```

Inter-window copying is vital for animations and the **gfx** library provides support for a number of copying modes. Data can be overlayed, masked, inverted or simply copied from the source window (area defined by region) to the destination window at the co-ordinate specified.

```
void       GfxResetClip();
void       GfxSetClip(GfxRegion *region);
void       GfxGetClip(GfxRegion *region);
```

Window clipping is supported, and can be obtained, set or reset using the above functions. The clipping is implemented in the **gfx** library at a very low level; so all routines comply with the clipping boundaries.

```
void       GfxSetPixel(SHORT x, SHORT y, GfxColor color);
GfxColor   GfxGetPixel(SHORT x, SHORT y);
void       GfxDrawLine(SHORT x1, SHORT y1, SHORT x2, SHORT y2, GfxColor color);
```

Elementary ☺

```
void       GfxDrawString(BYTE *string, SHORT len,
                         SHORT x, SHORT y, GfxDrawOperation mode);
```

The above routine provides the developer with the ability to draw a simple text message to the current display window. The string is drawn using the currently active font (see below) and is performed using the window copying mechanism. Obtaining non black and white text can be done with a combination of window operations.

Fonts Routines

```
#define gfx_maxFonts      128      // 128 fonts, MAX!
#define gfx_fontCharCount 0x80
#define gfx_fontCharMask  0x7F

typedef enum
{
  gfx_helioSmallFont       = 0,    // vt-os
  gfx_palmosNormalFont     = 16,   // palmos
  gfx_palmosBoldFont,
  gfx_firstUserDefinedFont = 32    // user defined?
} GfxFont;

void      GfxDefineFont(GfxFont font, SHORT fontSize, BYTE *fontData,
                        SHORT fontWinWidth, SHORT fontWinHeight, SHORT *fontWidths);
```

The **gfx** library provides three basic fonts in its current state. The **gfx** library currently only supports the ASCII character set (128 characters). It is possible to define your own fonts, as is done with the definition of window. Providing support for larger font sets can be done using extra font definitions.

```
void      GfxSetFont(GfxFont font);
GfxFont   GfxGetFont();
```

The `GfxDrawString()` routine uses the active font that has been defined. It can be adjusted.

```
SHORT     GfxGetWordWrap(BYTE *string, SHORT maxPixels);
SHORT     GfxGetFontHeight();
SHORT     GfxGetCharsWidth(BYTE *str, SHORT len);
SHORT     GfxGetCharWidth(BYTE chr);
```

To managing the presentation of text, various font management routines have been provided. `GfxGetWordWrap()` determines how many characters can be displayed in a specific pixel width, clipping the string on word boundaries. The other routines can be used to center text and adjust high offsets.

Termination

```
void      GfxTerminate();
```

The **gfx** library must be terminated on exit of the application to de-allocate any resources it may be using during the execution of the application. Failure to terminate the **gfx** library may result in device crashes and instability.

This routine can be placed in the `QuitApplication()` function code block.

Licensing

The **gfx** library is the property of Aaron Ardiri.

Non-commercial usage:
The **gfx** library can be freely used within applications that are non-commercial. An acknowledgement or thanks must be provided in the README documentation for the use of this product, providing a link to the www.ardiri.com website.

Commercial usage / Source Code Licensing:
A lot of time and effort was placed in the development of the **gfx** library. In the event you wish to use the **gfx** library in a commercial application or obtain access to the source code of the library you must contact Aaron Ardiri directly for the negotiation of licensing by sending an email to aaron@ardiri.com

Cheers!

// az
aaron@ardiri.com
http://www.ardiri.com/