

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

/* This program demonstrates optimal window refreshing using a scrolling text
 * display as a sample.
 *
 * Copyright © 1992 Martin Taillefer
 * All Rights Reserved
 */

#include <exec/types.h>
#include <exec/libraries.h>
#include <exec/memory.h>
#include <utility/hooks.h>
#include <utility/tagitem.h>
#include <graphics/gfxmacros.h>
#include <intuition/intuition.h>
#include <intuition/screens.h>
#include <intuition/gadgetclass.h>
#include <dos.h>

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

/*****

/* There is one Line structure for every line of text in our fictional
 * document.
 */
struct Line
{
    struct MinNode ln_Link;    /* to link the lines together */
    STRPTR ln_Text;          /* pointer to the text for this line */
    ULONG ln_TextLen;         /* the length of the text for this line */
};

/*****

/* system libraries */
struct Library *IntuitionBase;
struct Library *GfxBase;
struct Library *LayersBase;

/* global display handles */
struct Screen *screen;
struct Window *window;
struct Gadget *scroller;
struct Hook refreshHook;

struct RastPort render;
struct RastPort clear;

/* our document along with associated view information */
struct MinList document;
ULONG numLines;
ULONG topLine;
ULONG oldTopLine;
ULONG linesVisible;
ULONG columnsVisible;
ULONG fontHeight;
ULONG fontWidth;
ULONG viewHeight;
ULONG viewWidth;
ULONG usefulWidth;
ULONG usefulHeight;

/* a state flag indicating whether the main application is busy */
BOOL taskBusy;
*****/
```

```
VOID InitDocument(VOID);
VOID FreeDocument(VOID);
VOID EventLoop(VOID);
VOID __asm BackFillHook(register __a2 struct RastPort *rp,
                        register __a1 struct BackFillMsg *bfm);

/*****

/* This is where it all begins.
 */
ULONG main(void)
{
    /* open the system libraries we need.
    */
    IntuitionBase = OpenLibrary("intuition.library",37);
    GfxBase = OpenLibrary("graphics.library",37);
    LayersBase = OpenLibrary("layers.library",37);

    if (IntuitionBase && GfxBase && LayersBase)
    {
        /* get a pointer to the default public screen */
        if (screen = LockPubScreen(NULL))
        {
            /* allocate and initialize a scroller as a BOOPSI object */
            if (scroller = NewObject(NULL,"propgclass",
                GA_RelRight, -13,
                GA_Top, 1+screen->WBorTop+screen->Font->ta_YSize+1,
                GA_Width, 10,
                GA_RelHeight, -12-(screen->WBorTop+screen->Font->ta_YSize+1),
                GA_RelVerify, TRUE,
                GA_Immediate, TRUE,
                GA_FollowMouse, TRUE,
                GA_RightBorder, TRUE,
                PGA_Borderless, TRUE,
                PGA_Freedom, FREEVERT,
                PGA_Total, 1,
                PGA_Visible, 1,
                PGA_Top, 0,
                PGA_NewLook, TRUE,
                TAG_DONE))
            {
                /* initialize data used by the backfill hook */
                refreshHook.h_Entry = ( ULONG (*)() )BackFillHook; /* point the */
                taskBusy = TRUE; /* hook to our routine. */

                /* open the window */
                if (window = OpenWindowTags(NULL,
                    WA_Left, 0,
                    WA_Top, 0,
                    WA_PubScreen, screen,
                    WA_AutoAdjust, TRUE,
                    WA_CloseGadget, TRUE,
                    WA_DepthGadget, TRUE,
                    WA_DragBar, TRUE,
                    WA_SizeGadget, TRUE,
                    WA_SizeBRight, TRUE,
                    WA_Title, "Optimized Refresh Sample",
                    WA_SimpleRefresh, TRUE,
                    WA_Activate, TRUE,
                    WA_Gadgets, scroller,
                    WA_MinWidth, 32,
                    WA_MinHeight, 10+12+(screen->Font->ta_YSize+1),
                    WA_MaxWidth, -1,
                    WA_MaxHeight, -1,
                    WA_IDCMP, IDCMP_CLOSEWINDOW | IDCMP_NEWSIZE
                        | IDCMP_REFRESHWINDOW | IDCMP_GADGETUP
                        | IDCMP_GADGETDOWN | IDCMP_MOUSEMOVE
                        | IDCMP_VANILLAKEY,
                    WA_BackFill, &refreshHook,
                    TAG_DONE))
                {
                    /* initialize our document structure */
                    InitDocument();

                    /* process user events in the window */
                    EventLoop();
                }
            }
        }
    }
}
```

```
/* free our document structure */
FreeDocument();

/* close up shop */
CloseWindow(window);

/* free the scroller BOOPSI object */
DisposeObject(scroller);

}

/* unlock the default public screen */
UnlockPubScreen(NULL,screen);

}

/* close the libraries we opened */
CloseLibrary(LayersBase);
CloseLibrary(GfxBase);
CloseLibrary(IntuitionBase);

/* tell the shell everything is all right */
return(0);
}

/*****

/* This function initializes our document. That means allocating 100
* Line structures and linking them together in an Exec list. The lines
* are filled with a pattern of text so we have something to display
* in our window
*/
VOID InitDocument(VOID)
{
struct Line *line;
UWORD i,j;

NewList((struct List *)&document);
numLines = 0;
i = 100;
while (i--)
{
if (line = AllocVec(sizeof(struct Line)+91,MEMF_CLEAR|MEMF_PUBLIC))
{
line->ln_Text = (STRPTR)((ULONG)line + sizeof(struct Line));
line->ln_TextLen = 40;
AddTail((struct List *)&document,(struct Node *)line);
numLines++;

j = 0;
while (j < 90)
{
line->ln_Text[j] = (numLines % 96) + 32;
j++;
}
}
}

/*****

/* This function frees all the memory allocated by InitDocument() */
VOID FreeDocument(VOID)
{
struct Line *line;

while (line = (struct Line *)RemHead((struct List *)&document))
FreeVec(line);
}

/*****
* This is the message packet passed by layers.library to a backfill hook.
* It contains a pointer to the layer that has been damaged, a Rectangle
* structure that defines the bounds of the damage. No rendering can occur
```

```
* outside of these coordinates.
*
* The backfill hook is also passed a RastPort in which the rendering
* should be performed.
*/
struct BackFillMsg
{
struct Layer *bf_Layer;
struct Rectangle bf_Bounds;
LONG bf_OffsetX;
LONG bf_OffsetY;
};

VOID __asm BackFillHook(register __a2 struct RastPort *rp,
register __a1 struct BackFillMsg *bfm)
{
struct RastPort crp;

crp = *rp; /* copy the rastport */
crp.Layer = NULL; /* eliminate bogus clipping from our copy */

if (taskBusy)
{
SetWrMsk(&crp,0xff); /* if the main task is busy, clear all planes */
}
else
{
SetWrMsk(&crp,0xfe); /* otherwise, clear all planes except plane 0 */
}

SetAPen(&crp,0); /* set the pen to color 0 */
SetDrMd(&crp,JAM2); /* set the rendering mode we need */
RectFill(&crp,bfm->bf_Bounds.MinX, /* clear the whole area */
bfm->bf_Bounds.MinY,
bfm->bf_Bounds.MaxX,
bfm->bf_Bounds.MaxY);
}

/*****

/* Adjust the scroller object to reflect the current window size and
* scroll offset within our document
*/
VOID SetScroller(struct Window *window, struct Gadget *scroller,
ULONG linesVisible, ULONG numLines, ULONG topLines)
{
SetGadgetAttrs(scroller>window, NULL, PGA_Visible, linesVisible,
PGA_Total, numLines,
PGA_Top, topLine,
TAG_DONE);
}

/*****

/* Render a single line of text at a given position */
VOID RenderLine(UWORD x, UWORD y, UWORD w, STRPTR text, ULONG len)
{
Move(&render,x,y); /* move the cursor to the position */

if (len > columnsVisible) /* is line is longer than allowed? */
len = columnsVisible; /* yes, so reduce its length */

Text(&render,text,len); /* write to the window */

if (len < columnsVisible)
RectFill(&clear,render.cp_x,y-render.TxBaseline,
x+w-1,y-render.TxBaseline+fontHeight-1);
}

/*****

/* This function performs most of the rendering work needed by our sample.
* It first locks the window's layer to insure it doesn't get sized during
```

```
* the rendering process. It then looks at the current window size and
* adjusts its rendering variables in consequence. If the damage parameter
* is set to TRUE, the routine then proceeds to explicitly erase any area
* of the display to which we will not be rendering in the rendering loop.
* This erases any left over characters that could be left if the user sizes
* the window smaller. Finally, the routine determines which lines of the
* display need to be updated and goes on to do it.
*/
VOID RefreshView(BOOL damage)
{
ULONG i;
struct Line *line;
UWORD x,y;

/* lock the window's layer so its size will not change */
LockLayer(NULL>window->WLayer);

/* determine various values based on the current size of the window */
viewWidth = window->Width - window->BorderLeft - window->BorderRight;
fontWidth = window->RPort->Font->tf_XSize;
columnsVisible = viewWidth / fontWidth;

viewHeight = window->Height - window->BorderTop - window->BorderBottom;
fontHeight = window->RPort->Font->tf_YSize;
linesVisible = viewHeight / fontHeight;

usefulWidth = columnsVisible * fontWidth;

if (linesVisible > numLines)
{
usefulHeight = numLines * fontHeight;
topLine = 0;
}
else if (topLine + linesVisible > numLines)
{
topLine = (numLines - linesVisible);
usefulHeight = (numLines - topLine) * fontHeight;
}
else
{
usefulHeight = linesVisible * fontHeight;
}

/* if we were called because of damage, we must erase any left over
* garbage
*/
if (damage)
{
/* erase anything left over on the right side of the window */
if ((window->BorderLeft + usefulWidth < window->Width - window->BorderRight)
&& usefulHeight)
{
RectFill(&clear>window->BorderLeft + usefulWidth,
window->BorderTop,
window->Width - window->BorderRight - 1,
window->BorderTop + usefulHeight - 1);
}

/* erase anything left over on the bottom of the window */
if ((window->BorderLeft < window->Width - window->BorderRight)
&& (window->BorderTop + usefulHeight < window->Height - window->BorderBottom))
{
RectFill(&clear>window->BorderLeft,
window->BorderTop + usefulHeight,
window->Width - window->BorderRight - 1,
window->Height - window->BorderBottom - 1);
}
}

/* if we have at least one line and one column to render... */
if (usefulHeight && usefulWidth)
{
/* get a pointer to the first line currently visible */
line = (struct Line *)document.mln_Head;
i = topLine;
while (line->ln_Link.mln_Succ && i--)
line = (struct Line *)line->ln_Link.mln_Succ;
```

```
if (damage
|| (topLine >= oldTopLine + linesVisible - 1)
|| ((oldTopLine > linesVisible)
&& (topLine <= oldTopLine - linesVisible + 1)))
{
/* the whole display must be redrawn */
x = window->BorderLeft;
y = window->BorderTop + window->RPort->Font->tf_Baseline;
i = linesVisible;
}
else if (topLine < oldTopLine)
{
/* we just need to scroll the text */
ScrollRaster(&render,0,-(LONG)((oldTopLine - topLine) * fontHeight),
window->BorderLeft,
window->BorderTop,
window->BorderLeft+usefulWidth-1,
window->BorderTop+usefulHeight-1);

/* indicates what section needs to be redrawn */
x = window->BorderLeft;
y = window->BorderTop + window->RPort->Font->tf_Baseline;
i = oldTopLine - topLine;
}
else if (topLine > oldTopLine)
{
/* we just need to scroll the text */
ScrollRaster(&render,0,(topLine - oldTopLine) * fontHeight,
window->BorderLeft,
window->BorderTop,
window->BorderLeft+usefulWidth-1,
window->BorderTop+usefulHeight-1);

/* indicates what section needs to be redrawn */
i = linesVisible - (topLine - oldTopLine);
while (line->ln_Link.mln_Succ && i--)
line = (struct Line *)line->ln_Link.mln_Succ;

x = window->BorderLeft;
y = window->BorderTop + window->RPort->Font->tf_Baseline
+ (fontHeight * (linesVisible - (topLine - OldTopLine)));
i = topLine - oldTopLine;
}
else
{
/* we don't need to render anything */
i = 0;
}

/* render all the lines we need */
while (i-- && line->ln_Link.mln_Succ)
{
RenderLine(x,y,usefulWidth,line->ln_Text,line->ln_TextLen);
y += fontHeight;
line = (struct Line *)line->ln_Link.mln_Succ;
}

/* unlock the layer so normal operations can resume */
UnlockLayer(window->WLayer);

/* keep track of what the current top line is. That way, when we
* come back in this routine later, and "topLine" has changed, we
* can tell how many lines we need to scroll in order to sync up the
* display
*/
oldTopLine = topLine;
}

/*****

/* Whenever the application is busy, this function is called. It will
* change the behavior of the backfill hook in order to improve the
* appearance of the display until the application completes its lengthy
* task.
*
* You could also set a busy pointer in the document window in this routine
```

```

/* to tell the user you are not listening to him for awhile.
*/
VOID BusyState(BOOL makeBusy)
{
    taskBusy = makeBusy;

    if (LAYERREFRESH & window->WLayer->Flags)
    {
        BeginRefresh(window);
        RefreshView(TRUE);
        EndRefresh(window,TRUE);
    }
}

/*****

/* This routine is a typical event processor. It looks and acts on all events
* arriving at the window's port.
*/
VOID EventLoop(VOID)
{
    struct IntuiMessage *intuiMsg;
    ULONG                class;

    topLine    = 0;
    oldTopLine = 0;

    /* initialize rendering attributes we are going to use */
    render = *window->RPort;
    SetDrMd(&render,JAM2);
    SetWrMsk(&render,1);          /* we only want to render in the first plane */
    SetAPen(&render,1);

    /* initialize clearing attributes we are going to use */
    clear = *window->RPort;
    SetDrMd(&clear,JAM2);
    SetWrMsk(&clear,1);          /* we only want to clear the first plane */
    SetAPen(&clear,0);

    /* render the initial display */
    RefreshView(TRUE);

    /* set the initial scroller position and size */
    SetScroller(window,scroller,linesVisible,numLines,topLine);

    /* we aren't busy, so register that fact */
    BusyState(FALSE);

    while (TRUE)
    {
        /* if the LAYERREFRESH flag is set in the window's layer, it
        * means the layer has some damage we should repair.
        */
        if (LAYERREFRESH & window->WLayer->Flags)
        {
            /* enter optimized repair state */
            BeginRefresh(window);

            /* redraw the whole display through the optimized repair
            * region
            */
            RefreshView(TRUE);

            /* tell the system we are done repairing the window
            */
            EndRefresh(window,TRUE);
        }

        /* nothing left to do but wait for user input */
        WaitPort(window->UserPort);
        intuiMsg = (struct IntuiMessage *)GetMsg(window->UserPort);
        class = intuiMsg->Class;
        ReplyMsg(intuiMsg);

        /* we got a message, so act on it */

```

```

switch (class)
{
    /* user clicked on the close gadget, exit the program */
    case IDCMP_CLOSEWINDOW : return;

    /* user sized the window. We need to redraw the whole
    * display in order to eliminate any garbage. Start by
    * calling BeginRefresh() and EndRefresh() to eliminate
    * the window's damage regions then completely redraw
    * the window contents.
    */
    case IDCMP_NEWSIZE      : BeginRefresh(window);
                            EndRefresh(window,TRUE);
                            RefreshView(TRUE);
                            SetScroller(window,
                                        scroller,
                                        linesVisible,
                                        numLines,
                                        topLine);
                            break;

    /* Intuition is telling us damage occurred to our layer.
    * Don't bother doing anything, the check at the top of the
    * loop will catch this fact and refresh the display
    */
    /* Even though we don't do anything with these events, we
    * still need them to be sent to us so we will wake up and
    * look at the LAYERREFRESH bit.
    */
    case IDCMP_REFRESHWINDOW: break;

    /* user is playing with the scroller. Get the scroller's current
    * top line and synchronize the display to match it
    */
    case IDCMP_GADGETUP      :
    case IDCMP_GADGETDOWN    :
    case IDCMP_MOUSEMOVE     : GetAttr(PGA_Top,scroller,&topLine);
                            RefreshView(FALSE);
                            break;

    /* whenever a key is hit, we fake becoming busy for 4
    * seconds. During that time, try to size and depth arrange
    * the window to see what happens to its contents
    */
    case IDCMP_VANILLAKEY    : BusyState(TRUE);
                            Delay(200);
                            BusyState(FALSE);
                            break;
}
}
}

```