Introduction to Super Hi-Res in cc65

Chapter Five – More Super Hi-Res Pixel Graphics in cc65 - SHRFan

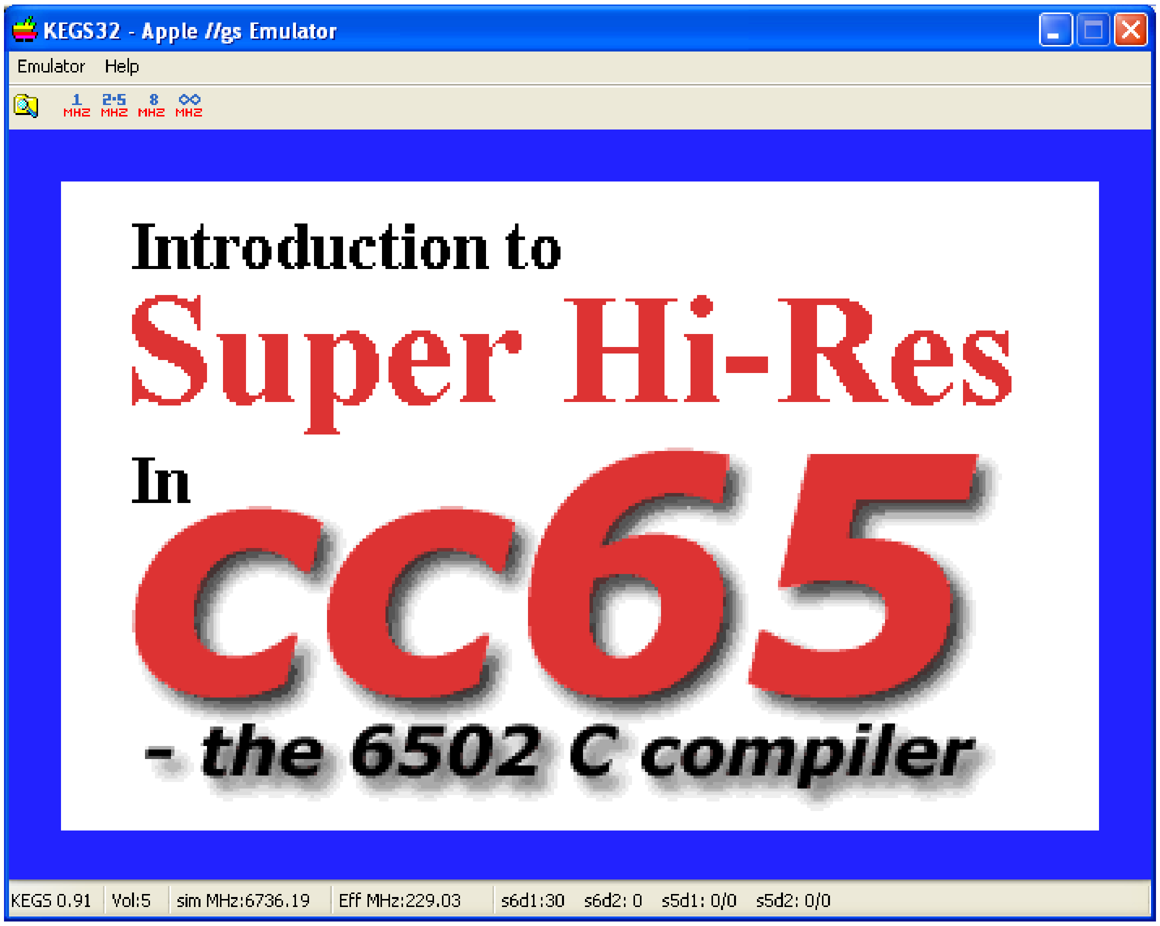
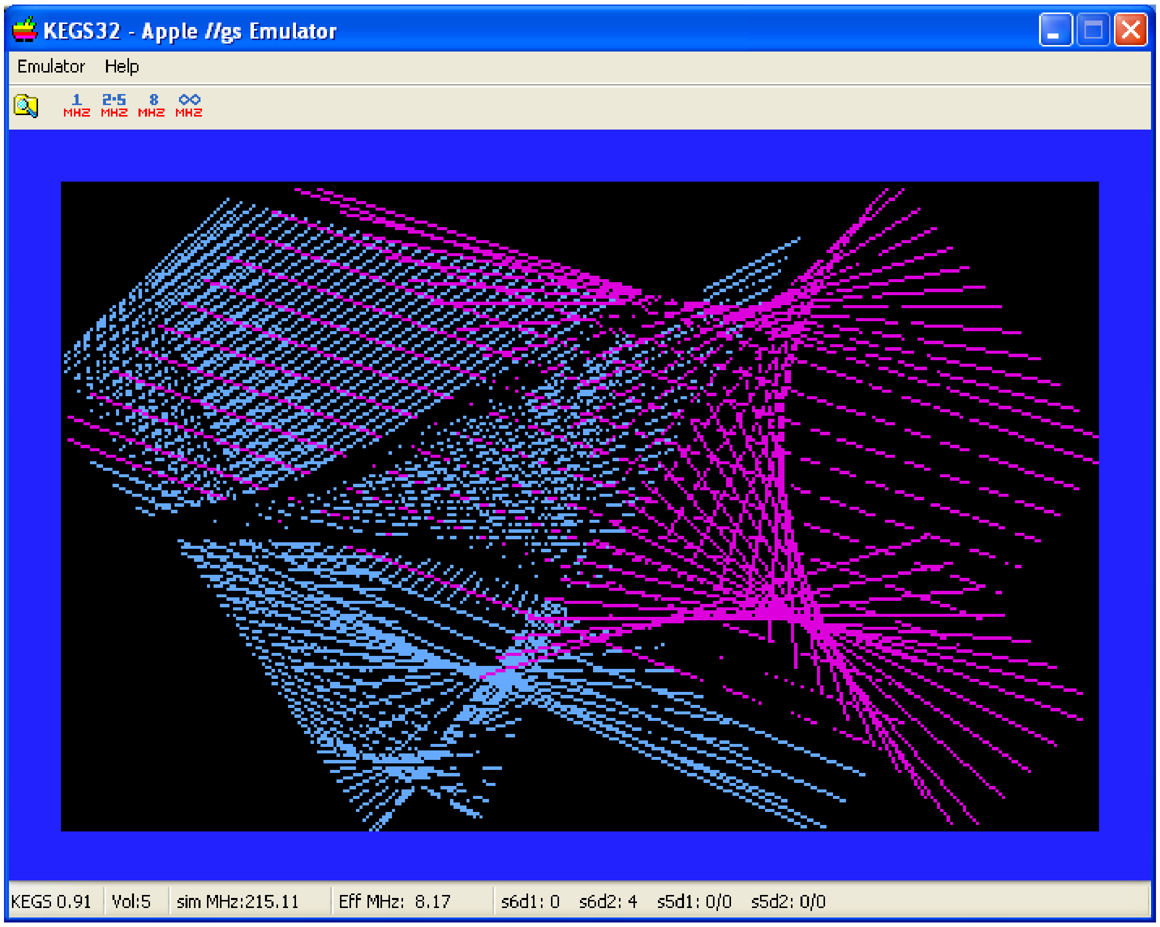
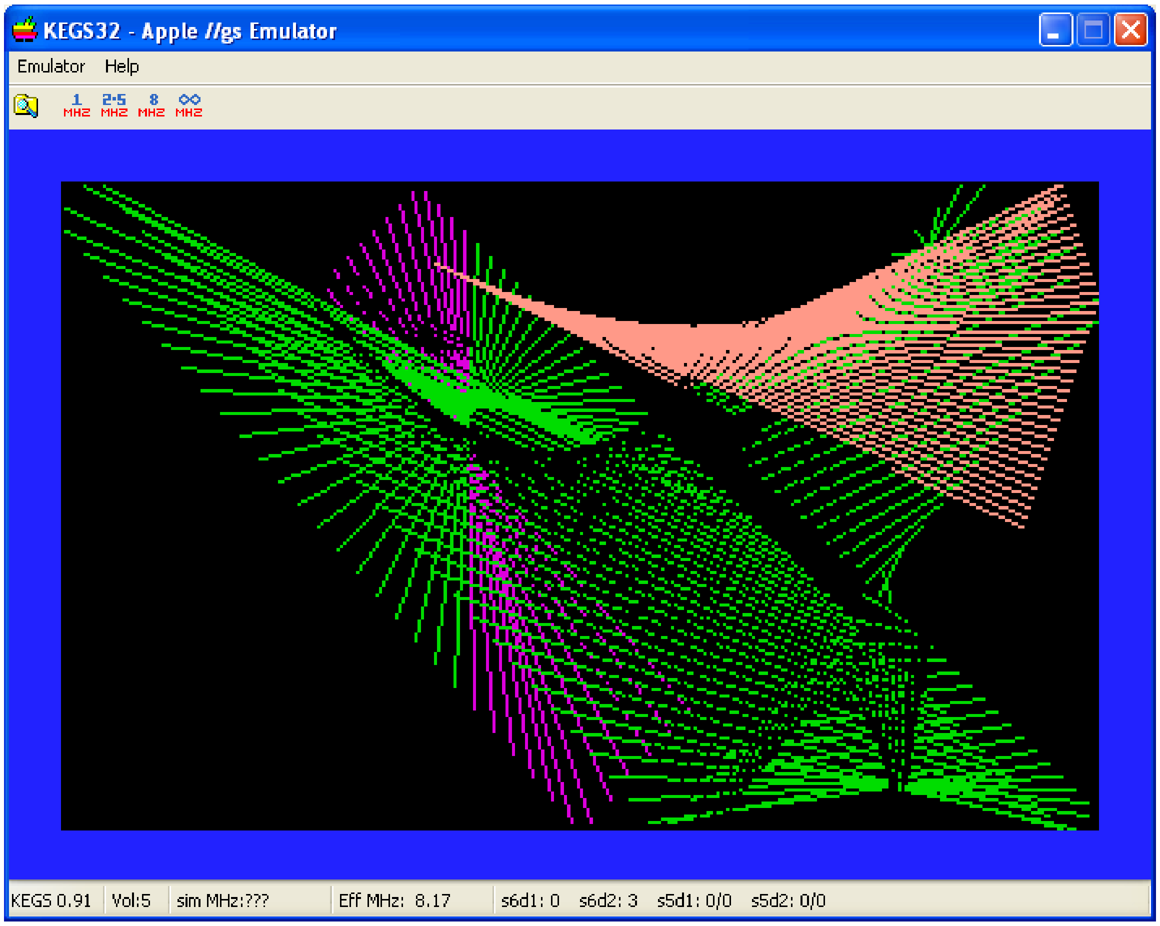

KEGS32 - Apple //gs Emulator
Emulator Help
1 MHz 2.5 MHz 8 MHz ∞ MHz
Introduction to
Super Hi-Res
In
cc65
- the 6502 C compiler
KEGS 0.91 Vol:5 sim MHz:6736.19 Eff MHz:229.03 s6d1:30 s6d2: 0 s5d1: 0/0 s5d2: 0/0

Table of Contents


KEGS32 - Apple //gs Emulator
Emulator
Help
1 MHz
2.5 MHz
8 MHz
∞ MHz
KEGS 0.91
Vol:5
sim MHz:215.11
Eff MHz: 8.17
s6d1: 0
s6d2: 4
s5d1: 0/0
s5d2: 0/0

Forward


KEGS32 - Apple //gs Emulator
Emulator Help
1 MHz 2.5 MHz 8 MHz ∞ MHz
KEGS 0.91 Vol:5 sim MHz:??? Eff MHz: 8.17 s6d1: 0 s6d2: 3 s5d1: 0/0 s5d2: 0/0

And Now For Something Completely Different

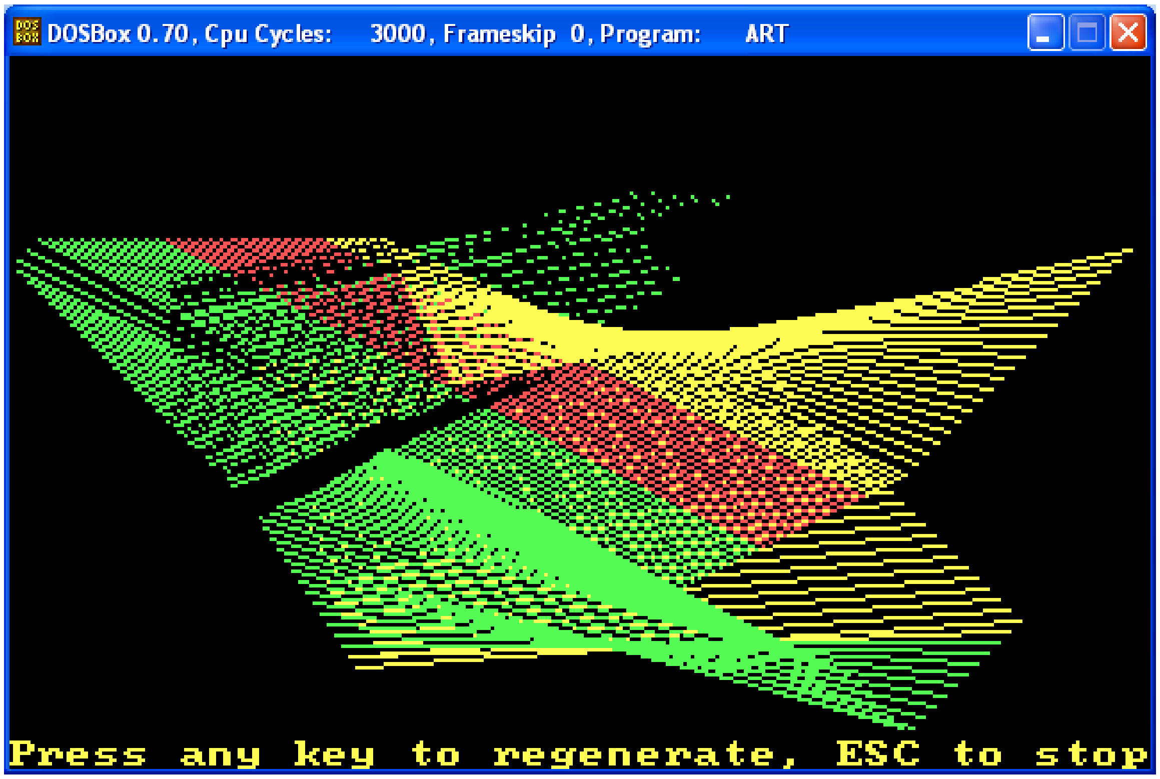
Now that we have learned something about Super Hi-Res (SHR) pixel graphics on the Apple IIgs and have slapped together an SHR routine or two we probably know enough about how pixel graphics work in SHR to do a little more than we did in our cc65 shrworld demo program in **Chapter Three**.

Shrworld was a lame demo. In this Chapter, to explore the potential of cc65's SHR pixel graphics capabilities further, we will be re-writing a somewhat less-lame demo called SHRFan that has been around in one way or another for many years.

Among other things, we will add a routine to our demo that saves up to 100 SHR Screen Shots to disk, and we'll provide 2 resolutions; 320 x 200 with normal pixels or 160 x 200 with double-wide pixels. We'll also add a routine to slide the colors in a palette from side to side to instantly change the colors of lines already on the screen. And we'll provide some other user options to make SHRFan a little more interesting.

History

The Early Years - 1985



```
Press any key to regenerate, ESC to stop
```

SHRFan is a new program written in cc65 for the Apple IIgs, but its origins predate the Apple IIgs and its Super-Hi Res (SHR) display. The IIgs was announced on September 15, 1986 but wasn't available until 1987. By 1986 **Turbo Pascal** (TP), based on the Blue Label Pascal Compiler written in 1981 by Anders Hejlsberg, was already on its 3rd release. Version 1 of TP in 1983 included Apple II's equipped with a CP/M 80 Microsoft SoftCard. After TP's success Bill Gates couldn't understand why Microsoft's stuff was so slow and he would yell at Greg Whitten, Microsoft's director of Programming Languages, for half an hour at a time. The release of TP version 3 for the IBM-PC included a sample graphics program called ART.PAS Version 1.00A written March 3, 1985, which ran in the 320 x 200 x 4 color CGA graphics mode. The author of ART.PAS is Unknown. No credits or Copyright notices appear in the source code which simply described itself as an "ART DEMONSTRATION PROGRAM". Maybe it was written by **Frank Borland** himself.

SHRFan is based on ART.PAS. The source code for ART.PAS is listed below:

ART.PAS Source Listing

```
Program ArtExample;
{
          ART DEMONSTRATION PROGRAM  Version 1.00A

     This program demonstrates the use of color graphics
     using TURBO PASCAL on the IBM PC and true compatibles
     with a color graphics adapter.

     INSTRUCTIONS
     1.   Compile and run this program using the TURBO.COM
          compiler.
     2.   Type <ESC> to exit the program, any other key to
          regenerate the screen.
}

const
  MemorySize = 150;
var
  X1, X2, Y1, Y2,
  CurrentLine,
  ColorCount,
  IncrementCount,
  DeltaX1, DeltaY1, DeltaX2, DeltaY2,
  I, Color: integer;
  Ch: char;
  Line: array [1..MemorySize] of record
                                   LX1, LY1: integer;
                                   LX2, LY2: integer;
                                   LColor:  integer;
end;
procedure Check;
var
  ch: char;
begin
  writeln('This program will only work if you have the color graphics
adapter installed');
  write('Continue Y/N ');
  repeat
    read (Kbd,Ch)
  until Upcase(Ch) in ['Y','N', #27];
  if Upcase(Ch) in ['N', #27] then
    Halt;
end;
procedure Init;
begin
  for I := 1 to MemorySize do
  with Line[I] do
  begin
    LX1 := 0;
    LX2 := 0;
    LY1 := 0;
    LY2 := 0;
  end;
  X1 := 0;
  Y1 := 0;
  X2 := 0;
  Y2 := 0;
  CurrentLine := 1;
  ColorCount := 0;
  IncrementCount := 0;
  Ch := ' ';
  GraphColorMode;
  Palette(2);
  Color := 2;
  gotoxy(1,25);
  write('Press any key to regenerate, ESC to stop');
end;
procedure AdjustX(var X,DeltaX: integer);
var
  TestX: integer;
begin
  TestX := X+DeltaX;
  if (TestX<1) or (TestX>320) then
  begin
    TestX := X;
    DeltaX := -DeltaX;
  end;
  X := TestX;
end;
procedure AdjustY(var Y,DeltaY: integer);
var
  TestY: integer;
begin
  TestY := Y+DeltaY;
  if (TestY<1) or (TestY>190) then
  begin
    TestY := Y;
    DeltaY := -DeltaY;
  end;
  Y := TestY;
end;
procedure SelectNewColor;
begin
  Color := Random(3)+1;
  ColorCount := 5*(1+Random(10));
end;
procedure SelectNewDeltaValues;
begin
  DeltaX1 := Random(7)-3;
  DeltaX2 := Random(7)-3;
  DeltaY1 := Random(7)-3;
  DeltaY2 := Random(7)-3;
  IncrementCount := 4*(1+Random(9));
end;
procedure SaveCurrentLine;
begin
  with Line[CurrentLine] do
  begin
    LX1 := X1;
    LY1 := Y1;
    LX2 := X2;
    LY2 := Y2;
    LColor := Color;
  end;
end;
procedure Regenerate;
var
  I: integer;
begin
  NoSound;
  GraphColorMode;
  Palette(2);
  for I := 1 to MemorySize do
    with Line[I] do
      Draw(LX1,LY1,LX2,LY2,LColor);
  gotoxy(1,25);
  write('Press any key to continue, ESC to stop');
  read(Kbd,Ch);
end;
procedure WanderingLines;
begin
  repeat
    repeat
      with Line[CurrentLine] do
        Draw(LX1,LY1,LX2,LY2,0);
      if ColorCount=0 then SelectNewColor;
      if IncrementCount=0 then SelectNewDeltaValues;
      AdjustX(X1,DeltaX1);
      AdjustY(Y1,DeltaY1);
      AdjustX(X2,DeltaX2);
      AdjustY(Y2,DeltaY2);
      Draw(X1,Y1,X2,Y2,Color);
      SaveCurrentLine;
      CurrentLine := Succ(CurrentLine);
      if CurrentLine>MemorySize then CurrentLine := 1;
      ColorCount := Pred(ColorCount);
      IncrementCount := Pred(IncrementCount);
    until KeyPressed;
    read(Kbd,Ch);
    if Ch <> #27 then
    begin
      Regenerate;
      gotoxy(1,25);
      write('Press any key to regenerate, ESC to stop');
    end;
  until Ch = #27;
end;
begin
  ClrScr;
  Check;
  Init;
  WanderingLines;
  TextMode;
end.
```

The 90's

I had great fun with my graphics programming in the 1990's while ART.PAS sat forgotten on a floppy in the corner of my computer lab for a while. When I first got into writing Screen Savers for Windows 3.1, I had **"Berkley After Dark"** installed on both my Mac IIci and on my 486 Multi-Media Windows machine, networked with my BBS in the other corner of my computer lab.

At some point in those years I crossed the room and resurrected ART.PAS from its floppy, and created an After Dark Screensaver for Windows 3.1 based on ART.PAS which I called VGAFan. One thing led to another and because I didn't like the fact that I was using some third-party run-time, I got busy and rewrote VGAFan in Microsoft C.

Around about 1996 I had the bright idea of writing a promotional "six pack" of screen savers in my off-hours but the company that I was working for at the time wasn't interested in using them so in 1997, wishing to be rid of the burden of this "six pack", I created a free screensaver download page on one of my websites and within a month I had almost 20,000 "hits". I called the "six pack" VGAFan after my first screensaver.

These were really "polished-up" too and VGAFan and all the rest were quite good for Windows 3.1 and only ran in 16 colors. But they were free and came with source.

Y2K

I didn't think about VGAFan and ART.PAS for a while until after the turn of the century rolled around. Then one day, I saw my VGAFan Screensaver running on a computer while visiting somebody, and they told me that it was part of a distributed processing experiment of some kind. Futher investigation revealed that in the year 2000, a fellow by the name of Elmar Krieger with the Center for Molecular and Biomolecular Informatics (CMBI) at the Katholic University in Nijmegen, Netherlands had modified VGAFan as the Windows component for the **"Models at Home"** experiment, which was later published in BioInformatics 18 by Oxford Press. ART.PAS certainly gets around!

Not to be outdone, but knowing that Windows 3.1 was long dead, after a while I wrote yet another version of VGAFan, this time as a Win32 screensaver for Windows XP, again with many additional features and no longer limited to 16 colors. That version of VGAFan aka ART.PAS is still available for download from my **ClipShop Website** with source of course. And through the death of Windows XP I have never thought much about VGAFan or ART.PAS again until just the other week when I was rummaging around for pixel graphics code to use in cc65 programs and remembered VGAFan.

Present Day – 2014

SHRFan was written by merging the code from the latest Win32 version of VGAFan together with the cc65 SHR routines that I have been working-on lately. Naturally the ART.PAS program has been heavily modified through its various generations as

VGAFan. Now due to the limitations of the SHR display it has devolved somewhat from the Win32 version of VGAFan back to 16 colors again, and due to the SHR screen resolution of 320 x 200 it also bears more resemblance to ART.PAS in screen metrics and other respects than VGAFan ever did. The Timer CallBacks and the Event Handlers are gone; it is no longer a GUI Application, and it launches in text mode just like ART.PAS but it has also retained the settings and the “flavour” of VGAFan. The untrained eye might even think that VGAFan descended from SHRFan and not the other way around.

However SHRFan also has some new settings targeted at the SHR display. It is not merely a port of ART.PAS but still retains the same purpose as ART.PAS; SHRFan is totally useless for any real practical purpose as ART.PAS always was.

Like ART.PAS, SHRFan is an “ART DEMONSTRATION PROGRAM” with the additional similarity of demonstrating the use of a compiler to write a graphics program for a vintage computer of the 1980’s. Whether that is useful or practical in 2014 is unlikely but it is fun. It is not productivity software but little is these days.

So now you know some of the history behind the SHRFan demo program, and how it came to be. I can hardly claim ownership of ART.PAS and its descendents, but they’ve certainly lived with me a long time and we have grown old together, and with a little luck and if the Apple II is truly Forever, we may even outlive Frank Borland although he still seems to be alive and kicking at Embarcadero.

Bill Buckels
bbuckels@mts.net
July 2014

Program Description

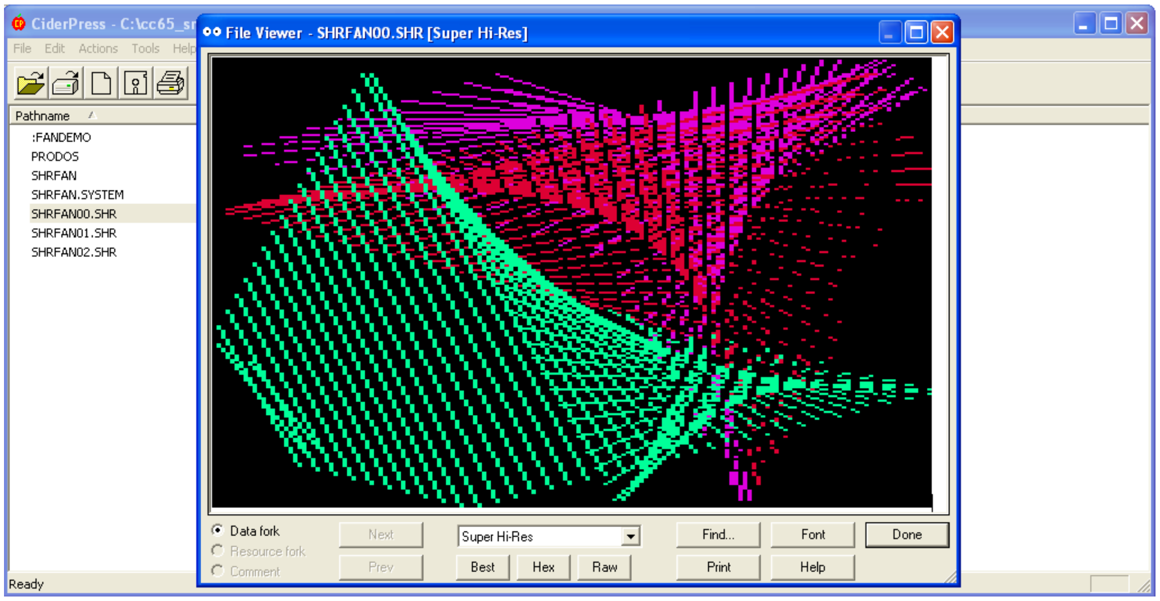
This program creates a Color Serpent Fan on a black background which bounces-off the sides of the viewport, and changes color and shape randomly.

The color fan can contain up to 200 lines of "Serpent Length". (This is a user-definable setting.)

Before a new line is drawn, the oldest line in the fan is redrawn in black, which effectively erases the "tail" of the serpent fan and "cross-hatches" over-top of any line segments that share the same area on the screen creating a 3d effect.

By erasing the oldest line in the fan, we keep the screen from becoming too cluttered and eventually filling-up with broken lines.

Adding Useless Features to a Useless Program



In today's world, the most useless of things seem widely coveted; Beer for example. There was a time when a man could drink copious amounts of beer and stagger down the street (with other large hairy men without tatoos). The value of Beer was measured by Volume. Today's men seem to value the useless sipping of small amounts of beer brewed to exact recipes created with apps that run on their telephones, with a taste that is hardly swillable at best. Staggering down the street is generally frowned upon today, but there was a time, especially on payday when men staggering through town at closing time or supper time raised nary an eyebrow even from the staunchest abstainer, and when programming of computers was measured in Volume and Usefulness.

Today's programs are to yesterday's programmer, like a twitter tweet is to a technical writer. So a program like SHRFan is probably far ahead of its time considering that it was conceived decades ago when useful things like swillable beer still had value and when programs like SHRFan were as useless as 8" floppies are today.

One of the most useless features that can be added to a graphics demo is the ability to save screen shots. SHRFan can save up to a hundred of them, automatically naming with no two the same either in name or appearance, and no way to tell which is which. Since these are 32K in size and are saved on the modest disk space of the IIgs, I stopped short of adding a feature for making movies of each frame. Saving pictures of lines when one can run a demo to see the same lines would seem to have much value to today's users of computers who save copious amounts of large pictures but drink very little beer.

The next thing I expect to see is somebody using this program to save a bunch of screens and winning a retro-art contest somewhere with them. This seems the same as using

PhotoShop and a Phone Camera to win a photo contest. Beer on the other hand was never considered ART. Today's amazing talent seems to swill commodities of much less useful but more highly coveted consumables. Like Apple II Programming which is arguably as useless as small amounts of beer and large amounts of pictures of colored lines (or pictures of your last vacation), this feature probably should have been added to SHRFan's early ancestor ART.PAS, but back then useless things had fewer working parts.

Saving a Screen Shot to a PIC File

Let's take a brief look at how we save the Super Hi-Res Screen Memory to a $C1 $0000 PIC file (this is simply the reverse of loading a PIC file):

```
/* ----------------------------------------------------- */
/* save the SHR screen to disk when the S key is pressed. */
/* ----------------------------------------------------- */
/* a simple $C1 $0000 SHR image save routine */
int picsave(char *name)
{
    FILE *fp;
    int i, c, status = ERR_WRITEPIXELS;
    unsigned src = 0x2000;  /* set src to scanline 0 */
    extern unsigned char _filetype;
    extern unsigned _auxtype;

    /* remove file if it already exists */
    fp = fopen(name,"rb");
    if (fp != NULL) {
        fclose(fp);
        remove(name);
    }
    /* set file type to Screen Image */
    _filetype = 0xC1;
    _auxtype = 0x0000;
    fp = fopen(name,"wb");
    if (fp == NULL) return ERR_OPEN;
    for (;;) {
        /* write image data - 32000 bytes */
        for (i=0;i<4;i++) {
           /* move to main memory */
           auxtomain(src,src+7999,0x2000);
           c = fwrite((char *)0x2000,1,8000,fp);
           if (c != 8000)break;
           src += 8000;
        }
        if (c!=8000) break;
        /* write scbs and palette - 768 bytes */
        status = ERR_WRITEPALETTE;
        auxtomain(src,src+767,0x2000);
        c = fwrite((char *)0x2000,1,768,fp);
        if (c!=768) break;
        status = SUCCESS;
        break;
    }
    /* close file and remove if a write error occurred */
    /* likely out of disk space */
    fclose(fp);
    if (status != SUCCESS) remove(name);
    return status;
}
```

The code above is very straight-forward and also demonstrates how we set the ProDOS File Type and Auxiliary Type when a file is created in cc65. We did the same thing when we created our PICLIST in our **picshow** SHR demo Slide Show program.

Automatically Naming Screen Shot Files

"Hot Keys" during a Save

There is no point in typing file names anymore. Or really in typing much of anything if one can't even see text on a screen without a magnifying glass. So for the modern user SHRFan offers a single key press, the 'S' key, to save a Screen Shot. Think of it as the 'SS' key if SHRFan is not paused, because if SHRFan is not paused first by using the Space Bar, the first 'S' that is typed will pause the screen just like the Space Bar, and the user can then change the colors by using the Arrow Keys or abandon the file altogether.

To abandon a save in progress, the user can either press the Space Bar and the demo will resume where it left off, or the user can press the RETURN key and the demo will regenerate with a different "Fan Serpent".

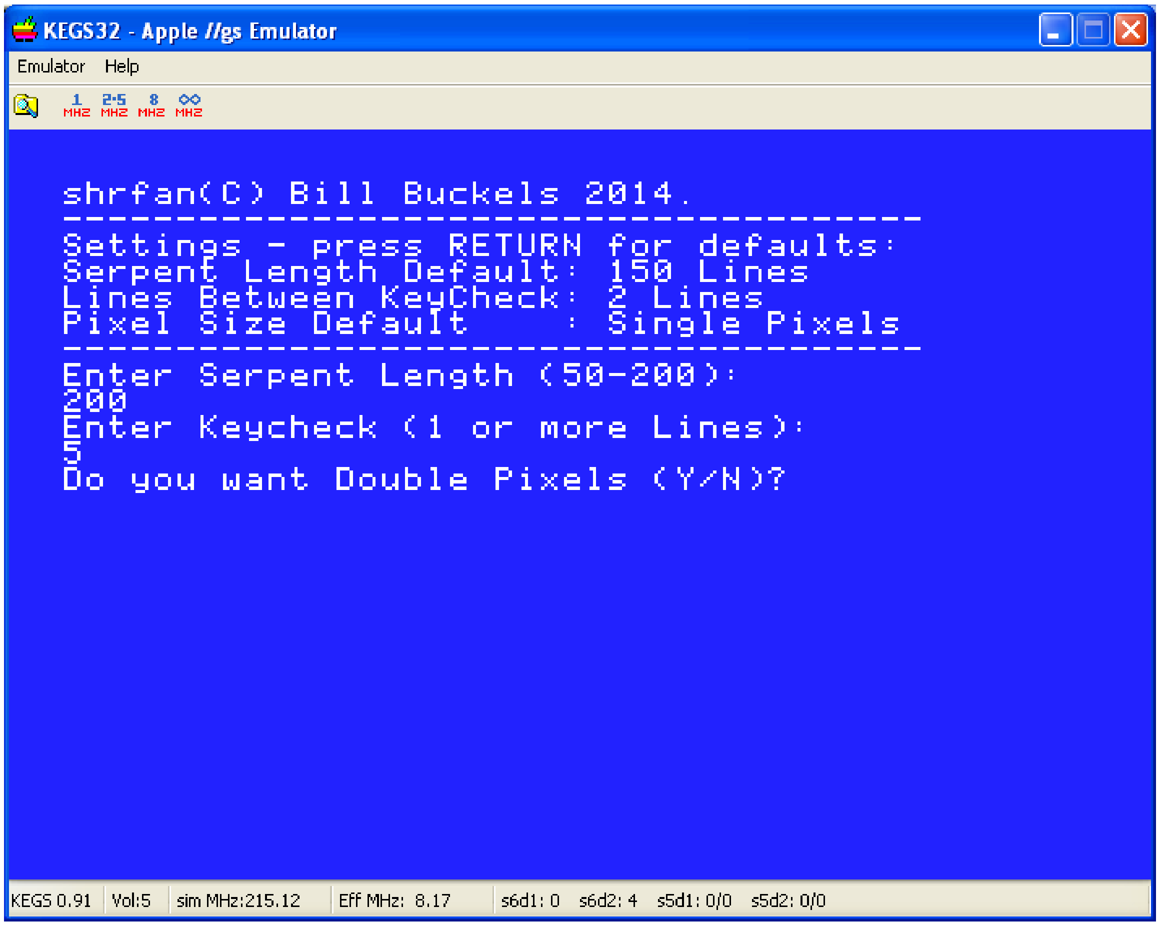
If the user is satisfied with what he sees, then pressing 'S' during a pause will save the Screen Shot to a PIC file provided there is no problem saving. If there is a problem saving, SHRFan never stops and only lets the user know that there is a problem at the end of the program. This is how things like Windows Updates work today. Many years ago standards like the CUI (Common User Interface) defined the behaviour of programs, but with the advent of systems that eventually looked like telephones, desktop ergomics has been replaced by thumbernomics and pinching of telephone screens. But with mnemonic key presses and automatic naming of useless files, SHRFan is certain to please even the most modern of user. ☺

Getting An Automatic Name

```
/* check for existing saved screen snapshot files
   we are supporting 100 using automatic naming
   this is a first-fit so if there are gaps in the
   existing file names the gaps will be filled */
int GetNextPicName(char *name)
{
    FILE *fp;
    int i;
    /* SHRFAN00.SHR to SHRFAN99.SHR */
    for (i = 0; i < 100; i++) {
        sprintf(name,"SHRFAN%02d.SHR",i);
        fp = fopen(name,"rb");
        if (fp == NULL) return SUCCESS;
        fclose(fp);
    }
    /* if we already have 100 files that's enough */
    name[0] = (char)0;
    return ERR_EXCESSIVE;
}
```

The code above is called from the key press loop in the **main()** program to create an automatic name during a screen shot save. If successful it leaves the name that it created in the buffer, and returns 0. Otherwise it zeroes the filename buffer and returns an error:

SHRFan main() Program Organization

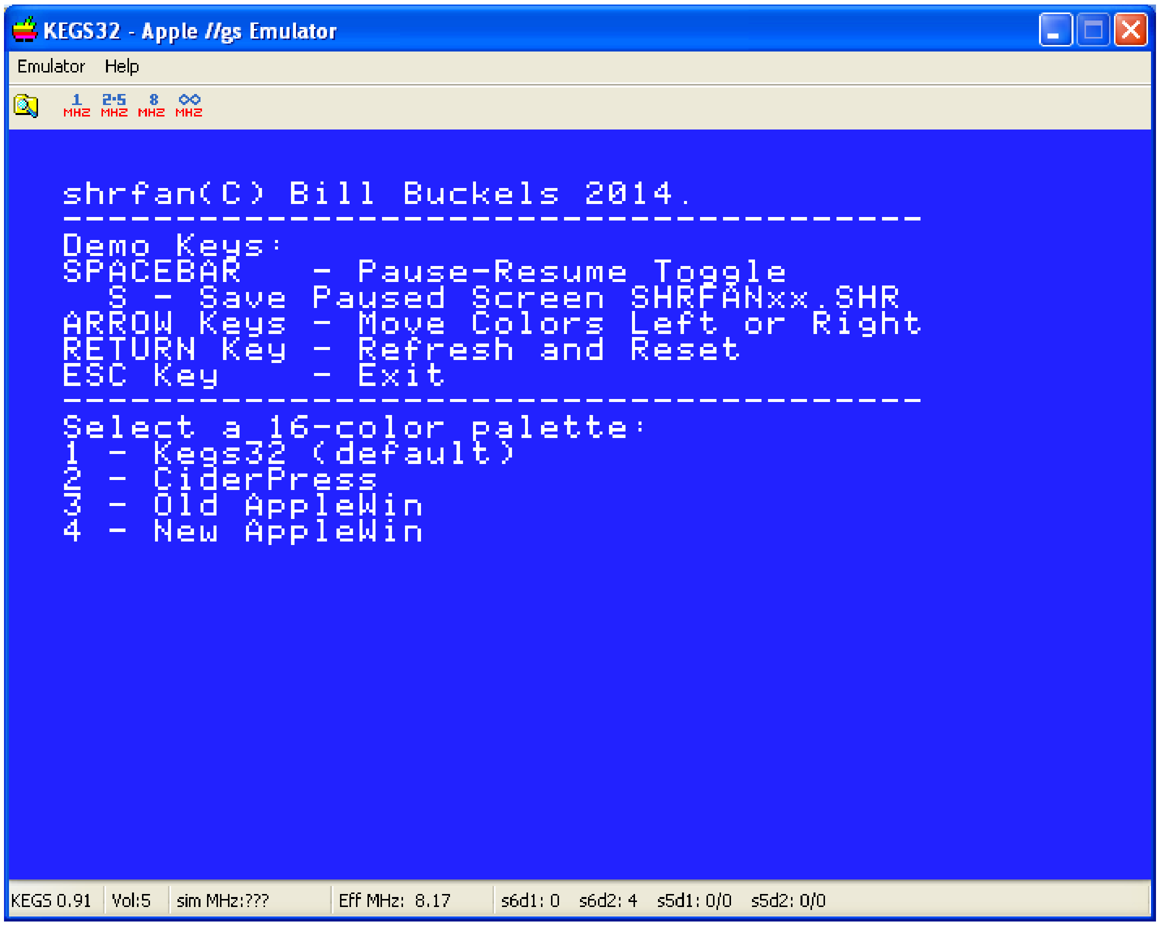

KEGS32 - Apple //gs Emulator
Emulator Help
1 MHz 2.5 MHz 8 MHz ∞ MHz
KEGS 0.91 Vol:5 sim MHz:215.12 Eff MHz: 8.17 s6d1: 0 s6d2: 4 s5d1: 0/0 s5d2: 0/0

```
shrfan(C) Bill Buckels 2014.
--------------------------------------
Settings - press RETURN for defaults:
Serpent Length Default: 150 Lines
Lines Between KeyCheck: 2 Lines
Pixel Size Default    : Single Pixels
--------------------------------------
Enter Serpent Length (50-200):
200
Enter Keycheck (1 or more Lines):
5
Do you want Double Pixels (Y/N)?
```

The **main()** program is organized like a Windows Screensaver, except that it is procedural and not event driven. In today's world it is like a VB.NET program compared to a C++ program. Very modern indeed, because the world seems to not code much in C++ anymore. Fortunately for the cc65 programmer, modern ISO C is much nicer than either language to write small programs like this in. So is Ansi C (MinGW etc) but unfortunately a programmer can't get so close to his video hardware on a modern computer so easily these days which is one of the reasons the Apple II is forever.

Settings

For the modern user, we have set-up a number of defaults, and in fact every setting has a default, so the RETURN key can be used by every modern user at every entry on both settings screens, and hopefully these modern users will pause long enough to read what the key presses can be used for. No problem if they don't though except for one thing. The only way out of this thing is by pressing the ESC key after SHRFan starts the demo. Other than that they can hit keys at random like most modern users.

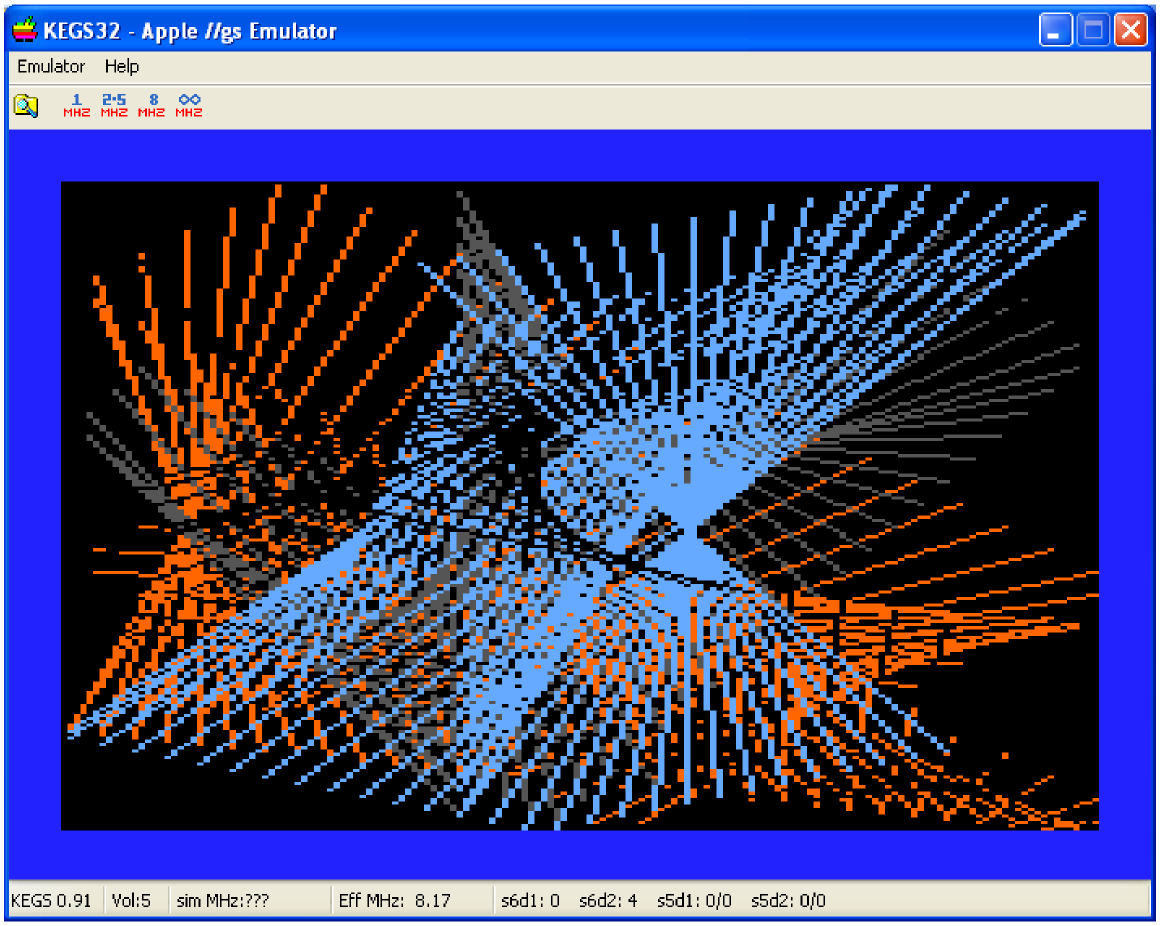


```
shrfan(C) Bill Buckels 2014.
-------------------------------------
Demo Keys:
SPACEBAR    - Pause-Resume Toggle
   S  - Save Paused Screen SHRFANxx.SHR
ARROW Keys  - Move Colors Left or Right
RETURN Key  - Refresh and Reset
ESC Key     - Exit
-------------------------------------
Select a 16-color palette:
1 - Kegs32 (default)
2 - CiderPress
3 - Old AppleWin
4 - New AppleWin
```

```
KEGS 0.91 | Vol:5 | sim MHz:??? | Eff MHz: 8.17 | s6d1: 0  s6d2: 4  s5d1: 0/0  s5d2: 0/0
```

For the serious user there are options. And for the others including the modern user, the display is in large print so it is readable on a tiny phone or tablet screen; a true marvel of modern design. In fact these settings screens are much more readable than the settings screens on the screen savers from which they descended… much more "in-synch" with the users of today's useless apps on small screened telephones.

Double Pixels


KEGS32 - Apple //gs Emulator
Emulator Help
1 MHZ 2.5 MHZ 8 MHZ ∞ MHZ
KEGS 0.91 Vol:5 sim MHz:??? Eff MHz: 8.17 s6d1: 0 s6d2: 4 s5d1: 0/0 s5d2: 0/0

The logical resolution above is coarser than the SHR's 320 x 200 display resolution… it is 160 x 200 pixels. On the first settings screen we responded "Y" when we asked if we wanted double pixels. When we draw a line using single pixels you will recall from previous Chapter Three that we must double-buffer the packed byte containing 2 pixels, and do some additional work to remove the previous pixel and then replace it with the new pixel while we leave the adjacent pixel alone.

But when we go to double pixels we just replace the pixel pair that is already on the SHR screen with the pixel pair of double pixels in the new color. We don't need to worry about preserving adjacent pixels. This is a quicker process; much fewer operations.

If you run SHRFan on a quicker IIgs like an 8 MHZ machine, it's almost fast enough for single pixels, but a slower machine may want double pixels.

Key Check

The Key Check setting is a yield setting.

The setting for number of lines is used to gain some additional performance by drawing lines in groups before checking for a key press.

It's a trade-off like trying to see the telephone screens of today instead of the decent monitors of days gone by. If you are a modern user and wait for more than 5 lines it will seem to take forever. But if you want to run this thing a Serpent at a time just make the settings for Serpent Length and Key Check the same. If you want to save screens or change colors set it to 1 or 2. The default is 2 so just press RETURN and you should be good to go.

Serpent Length

The serpent length setting allows from 50 – 200 lines in the fan. By default this setting is 150 lines.

Until now I have not explained what a Serpent Fan is. To begin with it's the state of an aggregate line object of a fixed number of lines that draws itself twice. When the line is at the head of the Color Serpent it draws in color and when it reaches the end of the tail it draws in black, which overwrites the colored copy of the line and a new line takes its place and becomes the head of the Color Serpent as the serpent continues to wind its fanned segments around the screen.

So when we selected a Serpent Length at the settings screen and asked for 200 lines instead of the default of 150 we got a longer and nicer Color Serpent than a modern user who just pressed RETURN in a hurry and accepted the defaults.

SHRFan main() Program Listing

```
int main(void)
{
    unsigned char ch, buf[40];
    int status = SUCCESS;

    /* initialize text mode. stay in 40 column mode. */
    texton();
    clrscr();
    /* initialize empty palette and scb's */
    initbuffers();
    /* clear keyboard buffer */
    while (kbhit() != 0) {
        cgetc();
        random();
    }
    /* settings screen */
    puts("shrfan(C) Bill Buckels 2014.");
    puts("--------------------------------------");
    puts("Settings - press RETURN for defaults:");
    puts("Serpent Length Default: 150 Lines");
    puts("Lines Between KeyCheck: 2 Lines");
    puts("Pixel Size Default    : Single Pixels");
    puts("--------------------------------------");
    puts("Enter Serpent Length (50-200):");
    gets(buf);
    wSerpentLength = (WORD) atoi(buf);
    if (wSerpentLength < MEMORYMIN)
        wSerpentLength  = MEMORYMEDIUM;
    else if(wSerpentLength > MEMORYSIZE)
        wSerpentLength  = MEMORYMEDIUM;
    puts("Enter Keycheck (1 or more Lines):");
    gets(buf);
    wNumberOfLines = (WORD) atoi(buf);
    if (wNumberOfLines <  MINLINES)
        wNumberOfLines = DEFLINES;
    else if (wNumberOfLines >  wSerpentLength)
        wNumberOfLines = wSerpentLength;
    puts("Do you want Double Pixels (Y/N)?");
    while (kbhit() == 0) random();
    ch = toupper(cgetc());
    if (ch == 'Y') wXBound = XPAIR;
    /* clear keyboard buffer */
    while (kbhit() != 0) {
        cgetc(); random();
    }
    /* KeyPress Info and Palette Selection Screen */
    clrscr();
    puts("shrfan(C) Bill Buckels 2014.");
    puts("--------------------------------------");
    puts("Demo Keys:");
    puts("SPACEBAR   - Pause-Resume Toggle");
    puts("  S - Save Paused Screen SHRFANxx.SHR");
    puts("ARROW Keys - Move Colors Left or Right");
    puts("RETURN Key - Refresh and Reset");
    puts("ESC Key    - Exit");
    puts("--------------------------------------");
    puts("Select a 16-color palette:");
    puts("1 - Kegs32 (default)");
    puts("2 - CiderPress");
    puts("3 - Old AppleWin");
    puts("4 - New AppleWin");
    while (kbhit() == 0) random();
    ch = cgetc();
    /* turn shr on */
    shgron();
    /* now that the shr display is in auxiliary memory */
    /* clear the palette and point all the scbs to the
       first palette */
    clearpalette();
    clearscbs();
    /* clear the shr screen to black */
    clear320(LOBLACK);
    /* set the palette that the user has selected */
    switch(ch) {
        case '2' : paletteptr = (unsigned char *)&rgbciderpress[0];
                   break;
        case '3' : paletteptr = (unsigned char *)&rgbawinold[0];break;
        case '4' : paletteptr = (unsigned char *)&rgbawinnew[0];break;
        default  : paletteptr = (unsigned char *)&rgbkegs32[0];break;
    }
```

```
setpalette(paletteptr,0);
/* save a copy of the user's palette selection
   for resetting if needed */
memcpy(&rgbsaved[0],&paletteptr[0],48);
/* clear keyboard buffer */
while (kbhit() != 0) {
    cgetc(); random();
}
for (;;) {
    WanderingLines();
    if (kbhit() != 0) {
        ch = toupper(cgetc());
        /* color change if arrow key is pressed */
        ShufflePalette(ch);
        while (kbhit() != 0) {
            cgetc(); random();
        }
        if (ch == ESCAPE) break;
        else if (ch == RETURNKEY)ReGenerate();
        else if (ch == SPACEBAR || ch == 'S') {
            /* if they pressed the S key before pausing they will
               need to press it again to save */
            for (;;) {
                while (kbhit() == 0) random();
                ch = toupper(cgetc());
            /* the spacebar and the return key have the same effect
               when paused both resume but pressing the return key
               clears the screen and regenerates the demo */
                if (ch == SPACEBAR) break;
                else if (ch == RETURNKEY) {
                    ReGenerate(); break;
                }
                else if (ch == 'S') {
              /* not reporting errors here */
              /* if we had an error saving before just skip this */
                    if (status == SUCCESS)
                        status = GetNextPicName((char *)&buf[0]);
                    if (status == SUCCESS) status = picsave(buf);
                }
                while (kbhit() != 0) {
                    cgetc(); random();
                }
                if (ch == 'S') break;
                /* when paused also allow color changes */
                ShufflePalette(ch);
            }
        }
    }
}
/* clear the palette and scbs then
   turn shroff and re-initialize text mode */
clearpalette();
clearscbs();
shgroff();
texton();
clrscr();
```

```
    /* report the error if a problem saving the screenshot */
    if (status != SUCCESS) {
        switch(status) {
            case ERR_OPEN:
                 printf("Error Opening %s\n",buf); break;
            case ERR_WRITEPIXELS:
            case ERR_WRITEPALETTE:
                 printf("Error Writing %s\n",buf); break;
            case ERR_EXCESSIVE:
                 puts("Error: 100 ScreenShot Limit Exceeded!"); break;
            default: printf("Unknown Error: %d!\n",status);
        }
        cgetc();
    }
    /* and exit */
    return SUCCESS;
}
```

Well that's certainly not as simple a main() program as the other main() programs in the previous chapters! But when we back-read the code above from the forever(;;) loop and WanderingLines() to the point where we clear the SHR palette and turn SHR off, and keep in mind that the rest of the code is just setting-up then cleaning-up afterwards, we can see that this whole program is just an event loop that checks for key presses and draws some lines then checks for key presses etc.

Not so very different from a Screen Saver. Very modern indeed! ☺

Let's have a look at some of the Helper Functions that are called in our event loop.

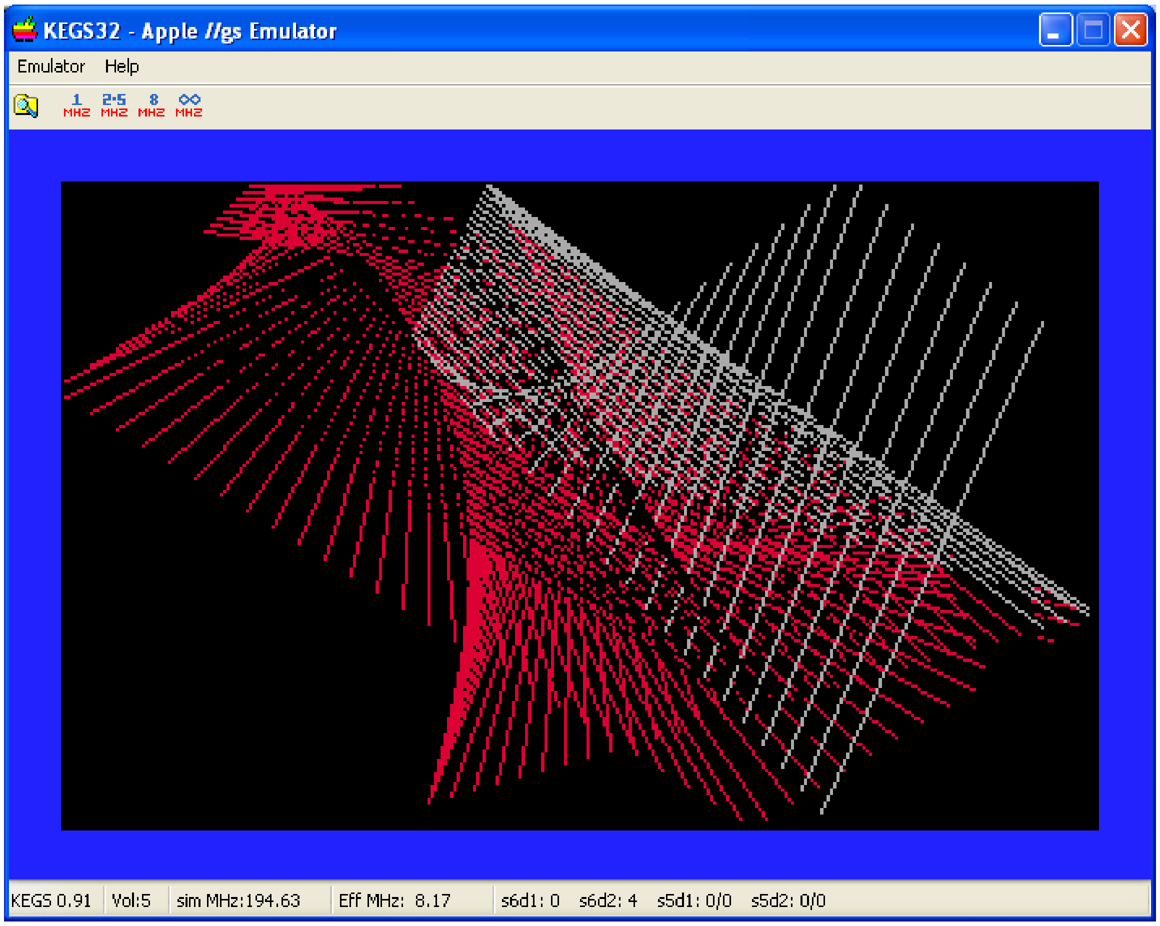
The Random Functions and the Functions That Use Them

The SHRFan program is entirely dependent on using random values. After realizing the random number generator in cc65 wasn't working for us and kept returning a one, we grabbed another one off the web to do this demo.

```
/* -------------------------------------------------------------- */
/* local random number generator to overcome the lack of a usable */
/* random number generator in cc65...                             */
/* -------------------------------------------------------------- */
/* http://stackoverflow.com/questions/7602919/how-do-i-generate-random-
numbers-without-rand-function */
unsigned RandomSeed = (unsigned)0xACE1;
unsigned random()
{
    unsigned bit = ((RandomSeed >> 0) ^ (RandomSeed >> 2) ^
                    (RandomSeed >> 3) ^ (RandomSeed >> 5) ) & 1;

    RandomSeed =    (RandomSeed >> 1) | (bit << 15);

    return RandomSeed;
}
```

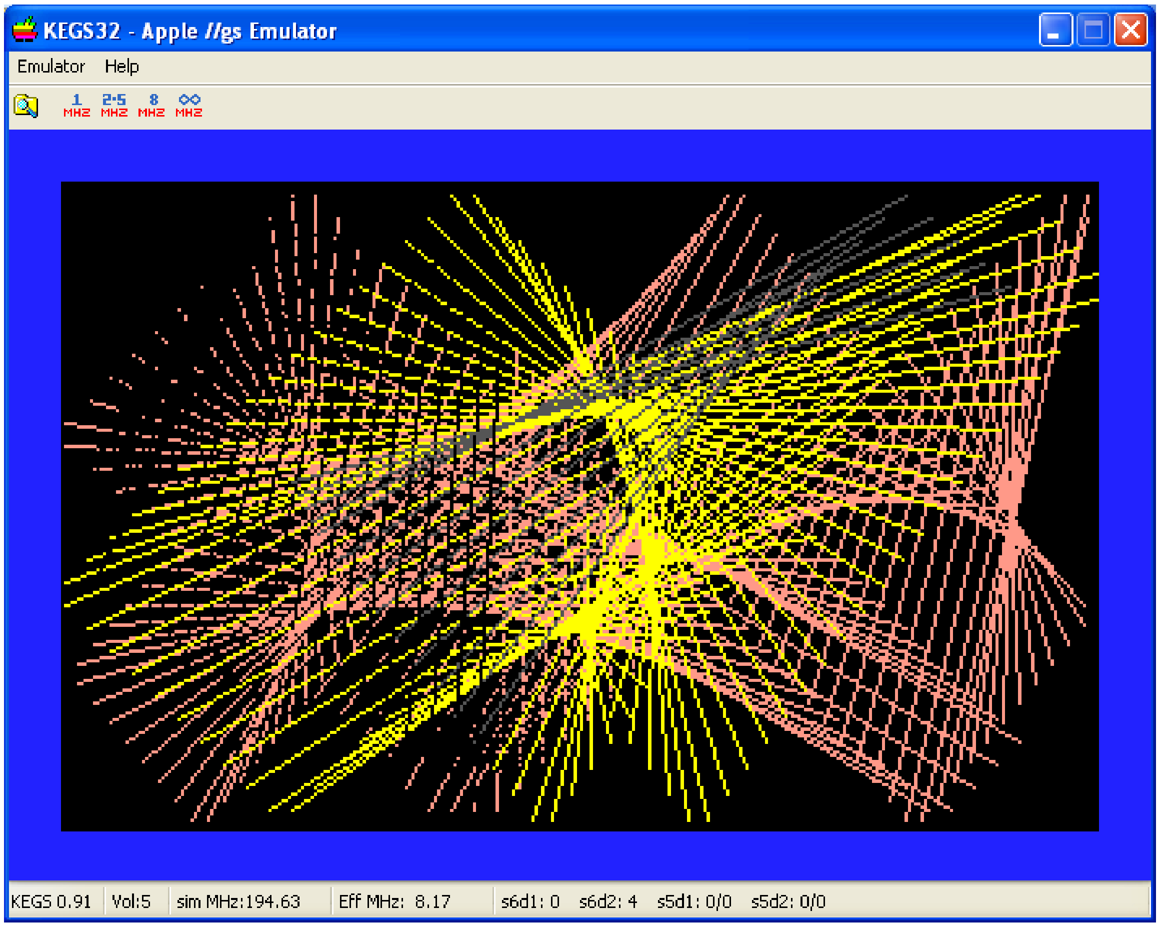
```
/* ------------------------------------------------------------- */
/* RandomRange                                                   */
/*   returns a random number in a usuable range  (1 to MaxValue) */
/* args                                                          */
/*   MaxValue - the highest number we want                       */
/* ------------------------------------------------------------- */
int RandomRange(int iMaxValue)
{
  int iRetVal;

  do {
    /* get random number */
    iRetVal = random();
    /* get a positive value */
    if (iRetVal < 0) iRetVal *= -1;

  } while(iRetVal < 1);
  /* use modulus of MaxValue if not in range */
  if (iRetVal > iMaxValue)
    iRetVal = (iRetVal%iMaxValue)+1;
  return iRetVal; /* return a value in range */
}

/* -------------------------------------------------------------- */
/* select new delta values - for geometric fan effect             */
/*   delta values are added during plotting to increment or       */
/*   decrement the end points on the lines to create a fan effect */
/* -------------------------------------------------------------- */
void SelectNewDeltaValues()
{
  iDeltaX1 = RandomRange(DELTAPLUS)-DELTAMINUS;
  iDeltaY1 = RandomRange(DELTAPLUS)-DELTAMINUS;
  iDeltaY2 = RandomRange(DELTAPLUS)-DELTAMINUS;
  iDeltaX2 = RandomRange(DELTAPLUS)-DELTAMINUS;

  /* the increment count controls how far each fan will travel */
  /* before it is replaced with a different geometry           */
  iIncrementCount = (DELTAMULTIPLIER)*(1+RandomRange(DELTASEED));

}

/* ------------------------------------------------------------- */
/* select new color                                              */
/*   called at intervals during plotting to select another color */
/* ------------------------------------------------------------- */
void SelectNewColor()
{

  DrawColor = (unsigned char) RandomRange(HIGHESTCOLOR);

  /* determine how long we will run this color before reselecting */
  iColorCount = (DELTAMULTIPLIER+1)*(1+RandomRange(DELTASEED+1));

}
```

Wandering Lines


KEGS32 - Apple //gs Emulator
Emulator Help
1 MHZ 2.5 MHZ 8 MHZ ∞ MHZ
KEGS 0.91 Vol:5 sim MHz:194.63 Eff MHz: 8.17 s6d1: 0 s6d2: 4 s5d1: 0/0 s5d2: 0/0

The function below is pretty standard in a graphics bounce algorithm that reverses direction when it hits the side of the screen's view port. We do something similar in Chapter 4 in our shrbounce demo. This function is called from **WanderingLines()** which is really the heart of SHRFan and this program.

```
/* ------------------------------------------------------- */
/* adjust - bounds processing                              */
/*   called to test and reverse x or y term during plotting */
/* ------------------------------------------------------- */
void Adjust(int *iPosition, int *iDelta, int iBoundMax)
{
  int iTest;

  iTest = *iPosition + *iDelta;
  if ((iTest < 1) || (iTest > iBoundMax)) {
    iTest = *iPosition;
    *iDelta = -*iDelta;
  }
  *iPosition = iTest;
}
```



The **WanderingLines()** code below draws the Serpent Fan. All those Random Number Functions previously listed are called from here. When we set the Key Check value at the start of SHRFan it sets the **wNumberOfLines** variable used below, and as previously noted only 2 lines are drawn by default before we check our keyboard.

```
/* ----------------- */
/* wander the lines  */
/* ----------------- */
void WanderingLines(void)
{

  WORD wCnt;

  for (wCnt = 0; wCnt < wNumberOfLines; wCnt++) {
    /* erase an old line (if any) before losing values     */
    /* the line array is a ring buffer using FIFO to erase */
    /*   the oldest line before drawing a new one          */
    if (wXBound == XBOUND)
        line320(WanderLine[iCurrentLine].iLX1,
                WanderLine[iCurrentLine].iLY1,
                WanderLine[iCurrentLine].iLX2,
                WanderLine[iCurrentLine].iLY2,0);
    else
        doublewide320(WanderLine[iCurrentLine].iLX1,
                      WanderLine[iCurrentLine].iLY1,
                      WanderLine[iCurrentLine].iLX2,
                      WanderLine[iCurrentLine].iLY2,0);

    /* end of color term so select a new color */
    if (iColorCount < 1) SelectNewColor();

    /* end of shape so get new values for our geometry */
    if (iIncrementCount < 1) SelectNewDeltaValues();

    /* get values for the current line */
    /* if we have collided with the sides */
    /*   then change direction */

    Adjust(&iLX1, &iDeltaX1, wXBound);
    Adjust(&iLY1, &iDeltaY1, YBOUND);
    Adjust(&iLX2, &iDeltaX2, wXBound);
    Adjust(&iLY2, &iDeltaY2, YBOUND);

    /* -------------------------------------------------- */
    /* save the current line into the line structure      */
    /*   restored in black when required again            */
    /* -------------------------------------------------- */
    WanderLine[iCurrentLine].iLX1 = iLX1;
    WanderLine[iCurrentLine].iLY1 = iLY1;
    WanderLine[iCurrentLine].iLX2 = iLX2;
    WanderLine[iCurrentLine].iLY2 = iLY2;
    WanderLine[iCurrentLine].color = DrawColor;

    if (wXBound == XBOUND)
        line320(WanderLine[iCurrentLine].iLX1,
                WanderLine[iCurrentLine].iLY1,
                WanderLine[iCurrentLine].iLX2,
                WanderLine[iCurrentLine].iLY2,
                WanderLine[iCurrentLine].color);
    else
        doublewide320(WanderLine[iCurrentLine].iLX1,
                      WanderLine[iCurrentLine].iLY1,
                      WanderLine[iCurrentLine].iLX2,
                      WanderLine[iCurrentLine].iLY2,
                      WanderLine[iCurrentLine].color);

    /* get ready for the next pass... and increment and decrement
         animation counters as required */
    iCurrentLine++;
    if (iCurrentLine > (int)wSerpentLength)
      iCurrentLine = 0;
    iColorCount--;
    iIncrementCount--;
  }
  return;
}
```

There that wasn't so bad. Admittedly, ART.PAS is somewhat more straight forward, but we really do get better ART from SHRFan. The code in either needs some study to wrap around ones head. So we keep in mind when we study the code that we are bouncing a line around like the bitmapped graphic we bounced in shrbounce.

Note in the code above we are calling one of two line drawing functions; **line320()** and **doublewide320()**. We already know about line320() from Chapter Three. When we selected Double Pixels at the start of SHRFan, logical horizontal resolution was changed to 160 double pixels so doublewide320() is used to draw lines instead of line320().

Drawing Double Lines

The doublewide320() function below uses the same pixel pair plotting function that we did in Chapter Three: **putpairs320()**

Using pairs of pixels to draw coarser lines gives us the advantage of not needing to read from SHR screen memory, and isolate a pixel in the high or low nibble of a byte, then update the byte, before writing. We therefore gain some speed on writing the byte, but since we still have the code overhead of the rest of this our efficiency gain is limited, and the coarser resolution may not be worth the sacrifice.

```
/* Bresenham Algorithm doublewide line drawing routine for SHR mode320
   resolution is 160 x 200 rather than 320 x 200 */
void doublewide320(int x1, int y1, int x2, int y2, unsigned char color)
{
    int dx, dy, sx, sy, err, err2;

    /* single pixel */
    if (x1 == x2 && y1 == y2) {
        putpairs320((unsigned)x1*2,(unsigned)y1,1,color);
        return;
    }
    /* vertical line */
    if (x1 == x2) {
        /* swap y co-ordinates if out of order */
        if (y1 > y2) {
            sy = y1;
            y1 = y2;
            y2 = sy;
        }
        x1 *= 2;
        y2++;
        while (y1 < y2) {
            putpairs320((unsigned)x1,(unsigned)y1,1,color);
            y1++;
        }
        return;
    }
    /* horizontal line */
    if (y1 == y2) {
        if (x1 > x2) {
            sx = x1;
            x1 = x2;
            x2 = sx;
        }
        sx = ((x2 + 1) - x1);
        putpairs320((unsigned)x1*2,(unsigned)y1,sx,color);
        return;
    }
    /* vector */
    if(x1 < x2) {
        dx = x2 - x1;
        sx = 1;
    }
    else {
        sx = -1;
        dx = x1 - x2;
    }

    if(y1 < y2) {
        sy = 1;
        dy = y2 - y1;
    }
    else {
        sy = -1;
        dy = y1 - y2;
    }
    err = dx-dy;
    for (;;) {
        putpairs320((unsigned)x1*2,(unsigned)y1,1,color);
        if(x1 == x2 && y1 == y2)break;
        err2 = err*2;
        if(err2 > (0-dy)) {
             err = err - dy;
             x1 = x1 + sx;
        }
        if(err2 <  dx) {
             err = err + dx;
             y1 = y1 + sy;
        }
    }
}
```

Regenerating The Demo

When the RETURN key is pressed in main(), ReGenerate() is called to clear the screen and to reset the Serpent Fan.

```
/* -------------------------------------------- */
/* regenerate the demo when RETURN is pressed. */
/* -------------------------------------------- */
void ReGenerate()
{
    int idx;

    /* start somewhere new each time */
    iLX1 = iLY1 = iLX2 = iLY2 = RandomRange(YBOUND);
    /* reset other values */
    iColorCount = iIncrementCount = iCurrentLine = NIL;
    /* zero-out previous line coordinates */
    for (idx = 0; idx < MEMORYMAX; idx++) {
        WanderLine[idx].color = 0;
        WanderLine[idx].iLX1 =
        WanderLine[idx].iLY1 =
        WanderLine[idx].iLX2 =
        WanderLine[idx].iLY2 = NIL;
    }
    random(); /* shake-it-up */

    /* clear screen and restore initial palette */
    clearpalette();
    clear320(LOBLACK);
    memcpy(&paletteptr[0],&rgbsaved[0],48);
    setpalette(paletteptr,0);
    /* good to go now! */
}
```

Move Colors Left or Right

When the arrow keys are pressed during the demo, the line colors 1-15 slide in the direction of the arrow key. This causes our lines to instantly change to different colors; some better, some worse. This is especially useless when we are paused and slide the colors around before saving yet another screen of colored lines, quickly filling-up our disk media. Modern users may enjoy this feature, because with hardly any effort, the illusion that some activity of some value has occurred can be achieved. A pleasant day spent using SHRFan in this manner may even result in a sense of satisfaction and achievement, especially among the less artistic. ☺

```
/* ----------------------------------------------------------- */
/* slide colors 1-15 sideways when the ARROW keys are pressed. */
/* ----------------------------------------------------------- */
void ShufflePalette(char direction)
{
    random(); /* shake-it-up */
    switch(direction) {
        case RTARROW:
        case DNARROW:
        case '+': /* forward - leave black alone */
            memcpy(&rgbwork[0],&paletteptr[3],45);
            memcpy(&paletteptr[6],&rgbwork[0],42);
            memcpy(&paletteptr[3],&rgbwork[42],3); break;
        case LTARROW:
        case UPARROW:
        case '-': /* reverse - leave black alone */
            memcpy(&rgbwork[0],&paletteptr[3],45);
            memcpy(&paletteptr[3],&rgbwork[3],42);
            memcpy(&paletteptr[45],&rgbwork[0],3); break;
        default:
            return;
    }
    setpalette(paletteptr,0);
    /* good to go now! */
}
```

Additional Notes and Downloads

The SHRFan program comes complete with source code and a working disk image. This document is only intended to provide an overview. In Chapter Three we covered the basics of SHR Pixel Graphics in cc65 and looked at the SHR core routines.

I would recommend that you backread the previous chapters for info on those. This document provides the additional notes that we need to write SHRFan using those same core routines. If there is anything else to be learned from SHRFan I would be shocked and amazed. But then I am not a modern user. ☺

SHR	Pixel Graphics – Chapter 5 - SHRFan
Demo Doc	http://www.appleoldies.ca/cc65/programs/shr/shrfan.zip http://www.appleoldies.ca/cc65/docs/shr/shrfan.pdf

SHR	Bitmapped Graphics – Chapter 4
B2Sprite Doc	http://www.appleoldies.ca/cc65/docs/shr/b2sprite.pdf
B2Sprite Utility	http://www.appleoldies.ca/cc65/programs/shr/b2sprite.zip
Bounce Demo	http://www.appleoldies.ca/cc65/programs/shr/shrbounce.zip
Fraglode Demo	http://www.appleoldies.ca/cc65/programs/shr/fraglode.zip

SHR	Bit-Mapped Graphics – Chapter 1 and Chapter 2
Demo	Image Loader: http://www.appleoldies.ca/cc65/programs/shr/piclode.zip
Demo	SlideShow: http://www.appleoldies.ca/cc65/programs/shr/picshow.zip
SHR	Pixel Graphics – Chapter 3
Demo Doc	http://www.appleoldies.ca/cc65/programs/shr/shrworld.zip http://www.appleoldies.ca/cc65/docs/shr/shrworld.pdf
DLGR	Bit-Mapped Graphics
Demo	http://www.appleoldies.ca/cc65/programs/dlgr/dloshow.zip
Doc	http://www.appleoldies.ca/cc65/docs/dlgr/dloshow.pdf
DLGR	Pixel Graphics
Demo	http://www.appleoldies.ca/cc65/programs/dlgr/dlodemo.zip
Doc	http://www.appleoldies.ca/cc65/docs/dlgr/dlodemo.pdf
DHGR	Bit-Mapped Graphics
Demo	http://www.appleoldies.ca/cc65/programs/dhgr/dhishow.zip
Doc	http://www.appleoldies.ca/cc65/docs/dhgr/dhishow.pdf
DHGR	Pixel Graphics
Demo	http://www.appleoldies.ca/cc65/programs/dhgr/dhiworld.zip
Doc	http://www.appleoldies.ca/cc65/docs/dhgr/dhiworld.pdf
Other	No-Slot Clock (not Graphics)
Demo	http://www.appleoldies.ca/cc65/programs/REALTIME.zip