Announcing **Pixel Blaster** - a high speed, versatile graphics and animation library for OS-9. Written in 100% efficient assembly-language, **Pixel Blaster** delivers lightning-fast action through incredible functions, designed for fast graphics manipulation. And, with the included BASIC09 and C interfaces, you can now incorporate sophisticated animation techniques into your own programs easily!

Licensed & distributed exclusively by CoCoPRO!, a division of DNM Enterprises, Inc.

Contents

PIXEL BLASTER For OS-9 Level II Version 1.0

Indy Heckenbach, Rt 4 Box 2122, Abbeville, La 70510

Documentation layout & design by Dave Myers.

Pixel Blaster : An Introduction

Have you ever dreamed of programming your own game? Maybe an arcade game? How about a 3D adventure? Or do you just dream of faster graphics? Do you need more graphics power? It's no secret: to program decent CoCo animation, you must use Disk Basic Assembly-Language. Until Today!

Announcing Pixel Blaster - a high speed, versatile graphics library for OS-9. Written in 100% efficient assembly-language, Pixel Blaster delivers lightning-fast action through incredible functions, designed for fast graphics manipulation.

Animation is a fundamental property of any game today. Without animation, a man can't walk, a ball can't bounce, and a sun can't set. Therefore, without efficient animation tools, there can be no game(notice how few animated games are written under OS-9).

The Get/Put functions are the heart of Pixel Blaster. The new GET/PUT functions are FAST! Under the typical windowing system, 100 sequential PUT's takes well over a minute. Pixel Blaster does the equivelant in about 4 seconds!!! This is a speed increase by nearly 1200% (at least 12 times faster)!

The special HIDE function really makes Pixel Blaster shine! This unprecedented ability opens the door to 3D animation. Under certain circumstances, HIDE will cause part of an object to disappear. By using simple slot techniques, this function will cause an object to appear to move behind its scenery. Imagine a man walking behind a table, a ball bouncing behind a tree, or a ship orbiting a planet. HIDE must be seen to fully appreciate it.

How do you do true animation? Easy - Pixel Blaster can easily flip through a series of buffers on the screen. About all you do is specify speed! This technique is similar to the way genuine cartoons are created. Other sophisticated functions such as flips, rotates, zoom, shrink, and animate can really bring your game to life. Sounds great, but is it hard to use? NO! Using Pixel Blaster is as easy as using other OS-9 graphics function! It's like programming with Gfx2, or the C graphics library. Since it is a system module, simply pass a control code to it, and away it goes! What's more, anybody can use it - whether they program in Basic09, C, or Assembly! In fact, a short C library and a Gfx4 module is included to make interfacing easier.

Precautions and Setup

The programs which you have purchased are contained on a 5-1/4" diskette in standard OS-9 format (single-sided, 35 track). I strongly suggest that you make a backup copy of the program disk.

Although I don't mind backups, I strongly discourage pirating and program exchange. This program was developed to help others program graphics animation. It took a great deal of effort and time to complete. I request that you respect my work and rights. Support me and I will fully support you.

Remember that all new programs may contain bugs. **Pixel Blaster** has been thoroughly tested to eliminate all problems. However, I do suggest that you be careful. Always take precautions, such as saving important work.

The simplest way to upload/copy the program is by using the COPY utility. Copy *'Pix'* from the /dd/CMDS directory. The other files such as demos and interfacing modules may also be copied.

Pix contains the entire Pixel Blaster program. *Pix* acts as an extension to the windowing system. A patch must be installed into Windint or Grfint before using Pixel Blaster. This patch links *Pix* to the windowing system. MODPATCH the appropriate module. Afterwards, you should COBBLER your boot disk. Otherwise, you must repatch every time at startup. In order to avoid the hassle of using OS9GEN to install a new bootfile, the patch replaces previous code. A data section of the windowing interface was removed. Hence, the computer boots with the wrong colors. This can be corrected by a SS.DFPal. The program *Color* does this. I suggest merging *Color* with the Shell and running it from the startup file. This way, the problem is virtually unnoticed. If the colors are reset after another window is initialized, the other window remains with incorrect color - so run *color* as early as possible. Do NOT put *Pix* in the bootfile! Even though it is a system module, *Pix* must be loaded off of disk due to technical reasons, similar to Grfdrv. I also suggest doing this in the startup file. Error 221 will indicate that *Pix* has not been loaded. Examine the sample startup files and installation procedure in Appendix A. If *Pix* constantly crashes on a boot disk, you may have encountered *Pix*'s requirement: in order to efficiently access a graphic screen, the bootfile must be over 24k. If it is not, you must lengthen the bootfile or load and activate more system modules.

Program Operations

The best way to start programming is to experiment a little. Examine the demos and read this manual thoroughly. You might find that certain things about the program are unusual. Remember though, **Pixel Blaster** was designed for optimum efficiency in speed. This section tells specifics on *Pix*.

Pixel Blaster commands are invoked by sending appropriate codes via I$Write system call. Basic09 Gfx4 and C Library pix.l does this for you. Refer to appendix B for examples in assembly, C, and Basic09.

Pixel Blaster operates on a 32k window - either on a 640*192 type 7 or 320*192 type 8 window. Most of the functions were designed for the 16 color window because it is the practical choice for any graphics programming. Using commands on the type 7 window may render interesting, yet unsupported results.

Pixel Blaster operates on a grid of 160*200. Note that standard modes give only 192 pixels vertical. The horizontal range is 0-159. On a 320*192 screen, you can manipulate two pixels minimum. On a 640*192 screen, four pixels is your minimum. This lower grid should cause no problems. Standard commands use a relative grid of 640. Translate horizontal coordinated by dividing by four.

PixBlast uses a type of virtual screen, one with no horizontal limit. Coordinates are plotted the same as before. However, if your coordinate goes beyond 160, the buffer will physically wrap around to the other side of the screen. This can be extremely useful in games. If a buffer is plotted below the visible 192 row, part of the buffer will be scrolled off the screen. A resolution of 200 vertical is allowed for compatibility with existing patches.

The buffers used by **Pixel Blaster** are not exactly identical to OS-9 buffers; you can NOT interchange them. Buffers can be defined as they are under OS-9. Groups may be in the range of 1-255, buffers between 1-255. **Pixel Blaster** can use buffers of any size, yet it can only get/put data under 8k. Commands may be redirected to another window or the standard output path(1).

For a quick test of *Blaster* functions, DISPLAY command may be used. Although **Pixel Blaster** operations are extremely fast, there may be times when every cycle counts. The first and foremost rule is to keep the buffer size as small as possible.

Pixblast can transfer horizontal bytes faster because they are consecutive. Also, the horizontal transfer is the routines' nested loop. Hence, it is better for the buffer to have a larger width than height. When you are trying to choose an operation based on speed, remember the speed precedence:

NOP -> CBAR-> PUT -> GET -> MIX -> HID -> HIX -> others

Obviously, NOP is quickest. Although CBAR is slightly faster, GET and PUT are the most useful and operate almost as quickly. The rest are much more complex and therefore take a longer time.

Blaster Library

The following is a complete listing of the commands offered by **Pixel Blaster**. This library is arranged in order of the control codes. This arrangement also happens to be from simple-complex. I suggest experimenting with each function thoroughly. Parameters passed are GRouP, BuFfer Number, Speed, X coordinate, Y coordinate, Width, Height, and nuLL.

NOP–No OPeration

FUNCTION :	Enters Pix and exits without performing an operation
CODE :	1B 70
PARAMETERS :	LL LL LL LL LL LL
NOTES :	This can be used to determine whether or not Pix is in memory

GET–GET buffer

FUNCTION :	Get specified area of the screen into a buffer
CODE :	1B 71
PARAMETERS :	GRP BFN X Y W H

PUT–PUT buffer

FUNCTION :	Put specified buffer on to the screen
CODE :	1B 72
PARAMETERS :	GRP BFN X Y

MIX–MIX buffer

FUNCTION :	Put buffer and mix it with the back ground
CODE :	1B 73
PARAMETERS :	GRP BFN X Y
NOTES :	Mix is accomplished by putting only non-zero pixels. Pixels of zero(slot 0 - generally white) are ignored

HIDE–HIDE buffer

FUNCTION :	Put a specific buffer onto the screen but hide part of it under background
CODE :	1B 74
PARAMETERS :	GRP BFN X Y
NOTES : -	Hide is accomplished by skipping back ground pixels with a slot of 12, 13, 14, or 15 - For example: a table could be drawn with slots 12-15. The rest of the scenery would be of other slots. A man figure is placed in a buffer. When a hide is acti vated over the table, the man would seem to partially disappear. As the co ordinates are changed, the man will appear to be moving behind the table from a 3D perspective

HIX–HIde and miX buffer

FUNCTION :	Put a specific buffer onto the screen with the qualities of MIX and HID
CODE :	1B 75
PARAMETERS :	GRP BFN X Y
NOTES :	See Hide and Mix for details

ANIP–ANImate by Putting buffers

FUNCTION :	Animate by rapidly putting a series of buffers
CODE :	1B 76
PARAMETERS :	GRP S X Y
NOTES : -	This function is similar to a genuine cartoon animator. Genuine animation is done by rapidly swapping a series of pictures, each slightly different. Thiscan give an illusion of movement -Animation flips between every buffer in this group sequentially. Any num ber of buffers may be used. Any buffer may be missing. If buffer 255 is not found, it will return an error upon completion - The speed specified is multiplied and used for a timing delay loop. A speed of zero causes no delay and may be too fast. Large numbers allows one to easily view the animation at slower speeds

ANIM–ANImate by Mixing buffer

FUNCTION :	Animate by rapidly mixing a series of buffers
CODE :	1B 77
PARAMETERS :	GRP S X Y
NOTES :	See also AMP and Mix for a complete description

ANIH–ANImate by Hiding buffer

FUNCTION :	Animate by rapidly hiding a series of buffers
CODE :	1B 78
PARAMETERS :	GRP S X Y
NOTES :	See also AMP and HID

ANIX–ANImate by hiXing buffer

FUNCTION :	Animate by rapidly hixing a series of buffers
CODE :	1B 79
PARAMETERS :	GRP S X Y
NOTES :	See also AMP, MIX, HID, and HIX

CBAR–Clear BAR

FUNCTION :	Draw a blank bar at specified coordinates
CODE :	1B 7A
PARAMETERS :	X Y W H
NOTES : -	CBar is always drawn with slot 0 - This is designed for a quick screen clear or buffer erase

HFLIP–Horizontal FLIP

FUNCTION :	Flip specified area horizontally
CODE :	1B 7B
PARAMETERS :	X Y W H
NOTES :	HFLIP does not work through buffers for faster on-screen operations and larger flips (the entire screen may be easily flipped)

VFLIP–Vertical FLIP

FUNCTION : Flip specified area vertically
CODE : 1B 7C
PARAMETERS : X Y W H
NOTES : see HFLIP

SDW–SiDeWays Flip

FUNCTION : Rotate a buffer on its side and put
CODE : 1B 7D
PARAMETERS : GRP BFN X Y
NOTES : - SDW will turn a buffer on its side. Horizontal pixels are converted to vertical pixels. For this reason a buffer may appear slightly awkward when rotated - SDW also differs from VFP and HFP in that it goes through a buffer. This is necessary due to technical reasons. - objects can be rotated again by reGETting the on-screen product and SDW

BLOW–BLOW up

FUNCTION : Magnify a buffer
CODE : 1B 7E
PARAMETERS : GRP BFN X Y
NOTES : - BLOW doubles each pixel in the buffer both horizontally and vertically to the screen - The screen is used so a buffer may be blown up beyond the buffer's limit - BLOW is designed specifically to magnify an image for use in a 3D scenario. A figure may be blown or shrunken to simulate spacial difference

SHRINK–SHRINK down

FUNCTION : Shrinks a buffer
CODE : 1B 7F
PARAMETER : GRP BFN X Y
NOTES : - SHRINK is to be used in conjunction with BLOW- Shrunken figures will be distorted - The screen is used for quick response and to provide consistency with BLW

C Compiler Interface Module

In order to increase **Pixel Blaster's** efficiency, a small library has been written for C users. The library is naturally written in C. It contains a short name for each function. Call the function by simply using the function name and passing the appropriate parameters. Parameters should be defined as CHARs in your program. In order to use the functions, you must link this C library, Pix.l, as follows:

CC1 prog.c -l=/dd/lib/pix.l

Here are the names of the functions and the appropriate parameters to pass:

nop(path,null,null,null,null,null,null);
get(path,grp,bfn,x,y,w,h);
put(path,grp,bfn,x,y);
mix(path,grp,bfn,x,y);
hide(path,grp,bfn,x,y);
hix(path,grp,bfn,x,y);
anip(path,grp,speed,x,y);
anim(path,grp,speed,x,y);
anih(path,grp,speed,x,y);
anix(path,grp,speed,x,y);
cbar(path,x,y,w,h);
hflip(path,x,y,w,h);
vflip(path,x,y,w,h);
sdw(path,grp,bfn,x,y);
blow(path,grp,bfn,x,y);
shrink(path,grp,bfn,x,y);

For those without the C graphics standard library, a few extra functions are included in *Pix.l*. These extra functions that relate heavily to the use of *Pix*. Define parameters as CHARs, with the exception of size - define it as INTEGER.

defbuf(path,grp,bfn,size);

kilbuf(path,grp,bfn);

defcol(path);

fcolor(path,prn);

bcolor(path,prn);

border(path,prn);

palette(path,prn,col);

Basic09 Interface Module

Basic09 is such an excellent language, it would be unwise not to write an interfacing module for it. Hence, *Gfx4* calls **Pixel Blaster** for a Basic09 user. *Gfx3* is often used to call upon point and click windowing commands, so the name *Gfx4* is used. Simply do a RUN to call *Gfx4* and pass necessary parameters. Define all parameters as INTEGER, not BYTE or REAL. Unlike the C library, each function must be invoked by its control code, not its name. Following is a list of functions in order of control code.

```
RUN Gfx4(pth,fun,grp,bfn,x,y,w,h)
  $70    NOP $71      GET

RUN Gfx4(pth,fun,grp,bfn,x,y)
  $72    PUT $73      MIX $74      HIDE $75
  HIX

RUN Gfx4(pth,fun,grp,spd,x,y)
  $76    ANIP $77     ANIM $78     ANIH $79
  ANIX

RUN Gfx4(pth,fun,x,y,w,h)
  $7A    CBAR $7B     HFLIP $7C    VFLIP

RUN Gfx4(pth,fun,grp,bfn,x,y)
  $7D    SDW $7E      BLOW $7F     SHRINK
```

A Look Behind the Scenes

This section is about technical matters of **Pixel Blaster** and its environment. Although you do not need to read this section, it provides important details which may help you understand the program.

Here is a summary of the graphics windows. A graphics window is simply a large area of memory(nearly 32k) that is used to hold graphics data. The graphics window is bit-mapped (raster graphics), meaning that each bit or group of bits are the actual pixels the monitor screen displays. The beginning of the bit-mapped graphics memory is the upper, left corner of the screen. The graphics memory increases as you move across horizontally. When you get to the right hand side of the screen, the next row is the following memory. This progresses until you get to the end of the screen, the lower, right corner. I mentioned before that each bit or group of bits are the actual pixels the screen displays. The size used by pixels varies according to the resolution and mode. The more information a pixel keeps track of, the larger it is. A single bit can contain a zero or a one. When a pixel only uses one bit, it can contain two values. These values represent the first two slots. When a pixel can be any of sixteen colors, it requires four bits. The number each nibble(four bits or half of a byte) contains represents the sixteen slots. These slots are not actual colors; they are just slots. To set a color you simply place a shade number(any of the 64) into a GIME chip slot registers. I advise against this; OS-9 will do it for you, easier. See the Appendix C for a quick reference to these commands. Although any slots may be used for background/foreground colors, the first slot is commonly used for background, the second or third for foreground. **Pixel Blaster** MIX uses 0 for standard background. **Pixel Blaster** HIDE uses slots 12 through 15, so as to conflict with as few pictures as possible.

One of the reasons **Pixel Blaster** is so fast and powerful is that it does not do coordinate bit-banging. Bit-banging is a phrase generally used to describe bit rotation in order to obtain certain bits. This is most often used in serial communication. However, the phrase can fit graphics also. Bit-banging is used to manipulate each specific pixel on the screen. Doing so is extremely slow. *PixBlast* overcomes this dilemma by operating on entire bytes. This is why you can only access 0-159 horizontal

coordinates. The other reason behind **Pixel Blaster**'s power and speed is that it is very low-level. Low-level software is the fastest possible. **Pixel Blaster** directly manipulates the screen and directly accesses the window/screen tables. **Pixel Blaster** does not contain a single OS-9 system call. Everything is done as directly as possible. My program masks IRQ and FIRQ(ORCC #$50) to temporarily stop multitasking. This is done to map in the screen directly, suppress any form of interruptions, and generally improve program speed. The multitasking is resumed an instant later. The delay is so short, you won't even notice it. Setting up its workspace is much faster than telling the kernal to set one up for it. Also, *Pix* must temporarily handle the controller NMI, the only interrupt that can not be masked. NMI's are intercepted and appropriate actions taken.

The patch that must be installed to the window/graphics interface does the following:

- move WRITE ptr up a few bytes to compensate for altered code
- change the control code test to allow calling of Pix graphics
- optimize short part of interface so link patch may be inserted
- replace standard colors with Pix link:

Here is the actual source code for the patch. *Grfint* is slightly different from *Windint*. The source code is the same for both though, the assembled offsets are just different.

```
LEAX   name, PC      Get name of program
STU    $FE           Store dev mem ptr for future
LBSR   link          Run local link routine
BCS    error         Exit if error(avoid crash)
JSR    ,y            IF OK, jump to module
LBRA   unlink        Run local unlink routine name
FCC    'PIX'
```

The linkage is fairly simple. The U register points to the device (current) window's memory. This memory area contains an indirect pointer to window table entry, which is vital to Pix. Although this patch uses a system constant, multiprogramming need not be concerned. Again, Pix masks interrupts, so anything stored is very temporary. Hence, it can not affect another user. After linkage, Pix requests more parameters(the system has only passed two at this time). Control is returned and

CC3IO fetches more parameters. CC3IO then recalls Pix, which proceeds with mapping in the screen, buffer(s), etc.

The Future of PixBlast

I plan to use my program for various games. I have many exciting ideas which I am working on right now. I originally created the program for my personal use. The Blaster tools are the answer to my animation prayers. Finally, I can have fun programming good animation!

Whether or not I do offer any games in the future, it is quite obvious that the Blaster tools are fully capable of incredible animation. I developed this program to be used, not stored away in a closet with other protected programs. I would like to see my program aid other people. Therefore, if you have used Pixel Blaster in your own game or program and would like to market them, please contact the author directly. CoCoPRO! would like to market your program. They will take care of sales, distribution, and cost of licencing my program. Otherwise, it has been agreed that an additional fee of $25 will cover a licence for use in your own program. That, and $1 per-game-sold royalty will compensate full rights to market your program and offer *Pix* as a run time module.

If at any time you experience a crash, program bug, or problem of any sort, simply contact me at the address below. Also, I would enjoy hearing from anyone, whether it be questions, comments, or anything else. I strongly believe in user support. My address is listed below:

Indy Heckenbach Rt 4 Box 2122 Abbeville, La 70510

Appendix B : Utilities and Sample Programs

The disk contains two utilities(merged together as pix_util) in the commands directory. The first is *Show*. *Show* is a very simple graphics display program. It can handle only VEF uncompressed file. It was written simply for the purpose of demonstrations. It requires over 64k to run. *Pos* is the other utility. *Pos* is quite useful - it can ease the tedious task of locating coordinates of an object. Enter *Pos* and a graphics pointer will appear, controlled by the mouse. Position the cursor in the upper left corner of an item that you would like to GET. Click on it and a little overlay window pops up with the X,Y coordinates in hexadecimal. Now, find the lower right coordinates of the image and click again. Subtract the first pair of coordinates from the second to obtain width and height. The first starting coordinates and the width, height can then be passed to *Pix*.

Now for the most awaited part - demos. What would be more effective - to give a few short ones or give long, complex ones? I think most would prefer the former. Once someone grasps how to send simple control codes to *Pix*, they should be able to go as deep as they like. However, more complex demos are included in original source code on disk.First in assembly, a quick get followed by a sideways routine. The buffer should of been previously defined and the window should be of type 8.

```
            Psect   Demo,$11,$81,1,100,entry
code        fdb     $1b71   GET control
            fdb     $0501   GRP=5 BFN=1
            fdb     $0000   X=0 Y=0
            fdb     $3030   W=$30 H=$30
            fdb     $1b7d   SDW control
            fdb     $0501   GRP=5 BFN=1
            fdb     $0000   X=0 Y=0
entry       leax    code,pc Get codes start
            ldy     #14     Send 14 bytes
            lda     #1      To standard output(current window)
            os9     I$Write Write it
            bcs     error   if problem, gen err
            clrb
error       os9     I$Exit
            endsect
```

Secondly, a very short example in Basic09. A horizontal screen flip followed by a vertical flip renders interesting results. This procedure also shows how to pass parameters direct.

```
Procedure : Demo
dim fun:integer
fun=$7B
run gfx4(1,fun,0,0,159,191)
fun=$7C
run gfx4(1,fun,0,0,159,191)
end
```

To conclude with, a quick loop in C. This program scrolls a buffer across the screen and erases its remains from behind. The buffer is defined by the program.

```
main()
{
   char path=1,grp=5,bfn=1,x=0,y=0,w=30,h=30; int size=4000;

   defbuf(path,grp,bfn,size);
   get(path,grp,bfn,x,y,w,h);
   for (x=0 ; x != 100 ; x++)
            {
                     mix(path,grp,bfn,x,y);
                     cbar(path,x-2,y,w+2,h);
            }
   kilbuf(path,grp,bfn);
}
```

Appendix C : Related Commands, Errors and Conversion

The OS-9 windowing system features many commands that can be used in conjunction to Pix commands. The most important ones are listed below for quick lookup from a single reference.

Function	Control	Parameters	Description
DfnGPBuf	1B 29	GRP BFN HBL LBL	Define buffer
KilBuf	1B 2A	GRP BFN	Kill buffer
DefColr	1B 30		Selects standard colors
Palette	1B 31	PRN CTN	Select color for slot
FColor	1B 32	PRN	Select Foreground Slot
BColor	1B 33	PRN	Select Background slot
Border	1B 34	PRN	Select Border slot

The OS-9 error codes are used to describe an error that has occurred during *Pix*'s operation. The following table lists the relevant codes and how the error relates to *Pix*.

Error	Definition	Relation to Pix
183	ILLEGAL WINDOW TYPE	Pix requires 32k windows
189	ILLEGAL COORDINATES	Vertical coordinate error or buffer SDW/BLOW will go off screen bottom
191	BUFFER SIZE	Requires a larger buffer
194	UNDEFINED BUFFER	Pix can not find a buffer
207	MEMORY FULL	Pix can not operate from where it is located
221	MODULE NOT FOUND	Patch can not link to Pix