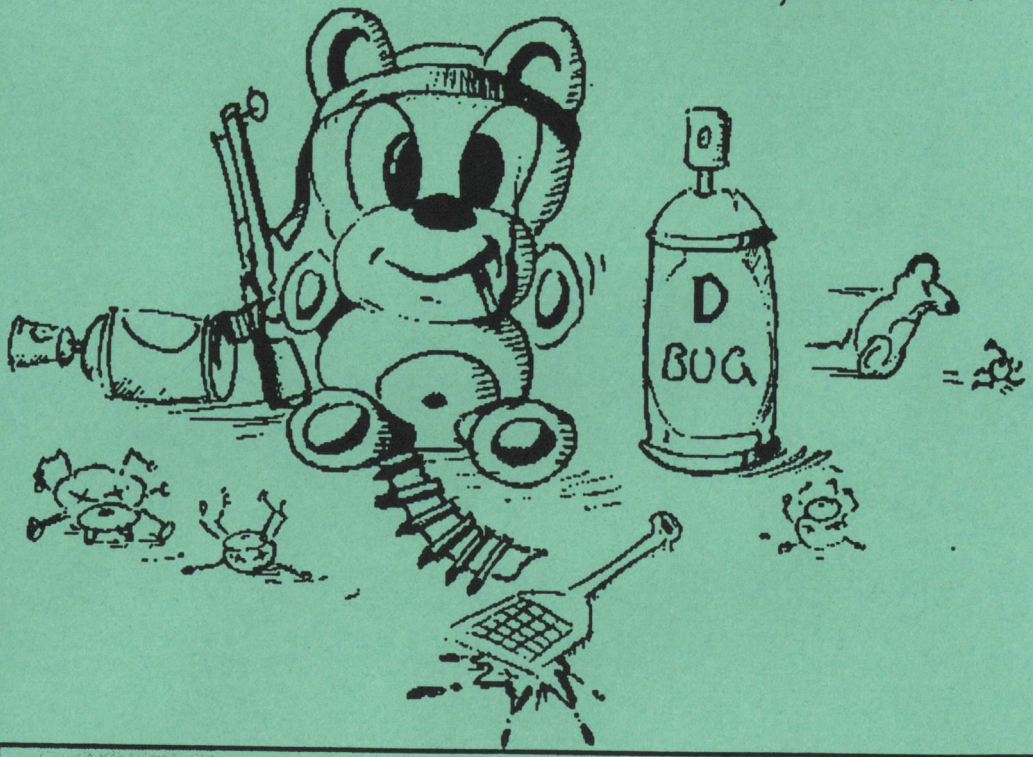
INFORMATION AND SUPPORT FOR BLITZ BASIC 2 USERS WORLD WIDE

BLITZ USER

ISSUE 3, March 1993



WO! Major upgrade time! Ted now has new sex appeal including overscan as well as 3.0 compatibility. And debugger me! yup, it's the long awaited runtime debugger for Blitz2, completely and utterly of use to everyone, it knows, it remembers, it obeys and it's pretty darn slick. Sorry about the wait subscribers but we're sure you will agree the improvements make Blitz Users the power programmers of the new age. Just a pity about the lack of contributions :) *Simon*

BACK IN THE OFFICE...

More gossip from the St Kevins Arcade office of Acid Software & Vision Software.

The new **Vision Software** game Woody's World is finally complete with the following specs:

DEVELOPEMENT TIME: Seven months
SCREENS: 4000
LEVELS: 60
GRAPHIC STYLES: 22
SOUNDTRACKS: 26
SCREEN UPDATE: 50 frames p/sec (100% Assembler of course)
COLOURS ON SCREEN: 32

Obviously its a big game, but it's also good fun and will be a welcome addition to any sophisticated game player's software library, so go out and buy it!

Mark Sibly has released his latest defender reincarnation into the Amiga PDscene called OBLIVION. Needless to say it's fast, furious and very very busy.

Mark's also had time to write Insectoids 2 an excellent Galaga style game in Blitz2 which will be released soon on the upcoming Blitz PD Disk 3. Accompanying Insectoids2 will be a new version of BuzzBar and a new car racing game SkidMarks written by office junior **Andrew Blackbourne.**

CD of the month goes to Sugar's Copper Blue, recomended to all Husker Du fans out there, probably an acquired taiste for anyone else.

As for life in New Zealand, summer is finally ending and hopefully hot, humid and sticky days are a thing of the past, replaced with nice windy autumn days perfect for windsurfing and the like....

No. 3

CONTENTS

BLITZ USER

Editorial 4
Hmmm, so we're a little late but it's quality not quantity yes?

Letters 5
Messages from all over the world and beyond...

Back in the office... 2
Gossip from the ever messy hive of activity in St Kevins Arcade.

The Blitz2 Debugger 6
Docs on using the brand new debugger added in the included upgrade disk.

Using The Blitz2 Debugger 8
Roger Lockerbie contributes a hands on approach to using the new debugger.

Optimizing Code 10
Lifted from the new Blitz2 User Guide, hints on speeding up your Blitz2 routines.

Also enclosed in this issue is a questionaire that you should return to us, it will help us in our quest to develop Blitz2 into the ultimate programming language for Amiga programmers.

Blitz User is a publication of Acid Software.

Editor

Simon Armstrong

Art Director

Rod Smith

Forward all contributions, advertising and correspondence to:

ACID SOFTWARE
10 StKevins
Arcade
Karangahape Rd
Auckland
New Zealand

fax:64-9-358-1658

EDITORIAL

Happy New Year to all, goodness it's March already!

Alright, so this issue is a little late but we haven't been slacking, it's just that it's so easy to under estimate the time it takes to get things done (even with fast machines and great programming languages).

I'll try and catch up with another issue next month hopefully with some user contributions:)

So whats new?

A debugger, a spunky ted, bug fixes and more. The main workload for me has been the new User Guide which is going to print next week, then it's time to get the sales figures cooking.

Included with this issue is a questionaire which I hope you will take the time to fill out and send back to us. As Blitz2 is continually under development we need as much feedback from users as possible.

If you've got a modem then see if you can locate a BBS that supports Amiga Net, a sort of fido network for Amiga users.

There is an echo coming out of Sydney Australia called Blitz_Ami, needless to say you can meet other Blitz Users, download and upload program listings, get support, have a moan an generally participate in one of the more happening echoes around.

Well the new Amiga 1200 finally arrived in New Zealand which has given Mark new inspiration.

He's been hacking pretty hard on the new wonder machine. Commodore has stated that "hitting the hardware" is definately not on with AGA and no hardware manual will be made available.

Wo, first time I heard that I thought they meant that crashing the machine and then smashing down on Ctrl/Amiga/Amiga with both fists was no longer acceptable behaviour!

No, hitting the hardware is where the Amiga programmer accepts the ROM contains all the routines needed to write fast smooth scrolling dualplayfield 8 bitmap games. Ha ha bloody ha!

The truth of the matter, after hacking the system copper lists apart seems to be that the new AGA design requires bulk cludges and a reference manual would have to admit to the fact and confuse any would be hardware hitter.

Thats not to say there's no hope, the whole chip set can go 4 times as fast in 'quad mode', 8 bitplanes with 100+ blits per frame looks feasible.

The only problem is that bitplane DMA is really weird and the way the operating system does smooth scrolling is very scary (certain horizontal positions will slow the system down by 4 times, ouch!).

Anyway, AGA Blitz mode is on it's way, it's just going to take a little while.

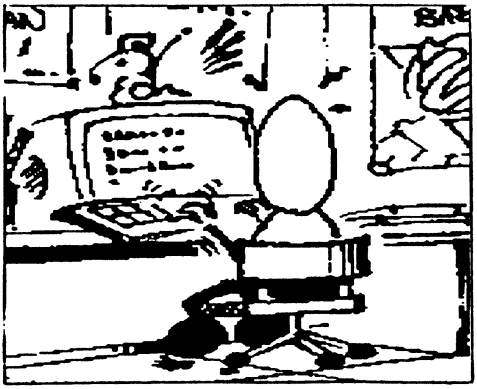
As for extended operating system support, it's right up the top of the list of things to do, next issue, promise!

More gadgets, better intuitools and better text handling are all being addressed.

As for our German users, our humble apologies for taking so long with the translations. They are going to print this week.

Anyway, fill out the questionaire and let us know how you are getting on with Blitz2, WE CARE!

Simon



LETTERS

Dear BlitzMan,

How can I switch Ted between interlace and non-interlace? The only way I can work out is if I delete l:BlitzEditor.opts which is a real pain.

Reply:

The new Ted on this issue's cover disk will now configure itself to the same resolution as the Workbench Screen. Both interlace and overscan will be the same settings as the current workbench, l:BlitzEditor.opts no longer saves this information.

Dear BlitzMan,

I was very angry to read the terse review of Blitz2 in the latest issue of Amiga World. It's really funny, the guy complains about the editor scrolling too fast and having to use quotes to terminate string definitions, pretty stupid really. Then he completely ignores all the workbench support you guys have added, I mean what was he on?

Reply:

Hmmm yes it was a very strange review. We have just finished a new User Guide which should help point people like him in the right direction. But we would appreciate if you could write to AmigaWorld as a BlitzUser pointing out the innaccuracies.

Dear BlitzMan,

So what's the low down, I have been waiting for a German translation of the manual for ages, where is it? Working out how to use Blitz2 with English instructions is a real pain.

Reply:

I'm sorry it's taken so long, it's off to the printer this week I promise and will be available no later than the third week of March.

Dear BlitzMan,

I'm having big problems using the FD converter and implementing libraries such as ASL into Blitz2. Can you help?

Reply:

A special Blitz2 developers kit is being put together that will include advanced documentation and better tools for converting Amiga libraries and developing custom commands in Blitz2. It is scheduled for release end of April 1993.

Dear BlitzMan,

I am having great difficulties learning Blitz2. The User Guide does not seem adequate in its descriptions of NewTypes, pointers and so forth.

Reply:

A new User Guide will be available soon that will be addressing these problems and more. It will be made available to existing Blitz Users at cost price, more details will be in the mail soon to all registered Blitz Users.

Please all, send BlitzMan your thoughts, we dedicated this months letters column to complaints.

However a bit of humour, world news, politics, sex, recipies, music wouldn't go amiss so send us a letter.

The Blitz2 Debugger

If a runtime error occurs when a program is run from the editor the Blitz2 debugger will be activated. It of course must be enabled in the compiler options requester.

It will not be activated if there is an error-handler already enabled in the program using the **SetErr** command.

The debugger can also be activated py using the CTRL/ALT C keyboard combination. The **STOP** command can also be used in your program to cause a break and invoke the debugger.

The debugger is a powerful tool in finding out causes of errors and locating bugs. The ability to step back through code executed prior to the break gives the programmer an excellent understanding of how an error has occurred.

All debugger commands are initiated by holding down CTRL/LEFT ALT and pressing the appropriate command key. There are 2 types of commands available. One type is selectable at any time while a program is running, the other type is selectable only when the program is stopped.

Note that when a runtime error occurs, keylock will be automatically toggled **ON**. See **K** (keylock) command below. This means that the Ctrl/Alt keys do not have to be pressed in combination with the debugger keyboard command.

These commands are available at all times:

C - stop program.

H - hide/show debugger window.

V - View/hide graphics behind debugger window.

M - select hires/lores mode of graphics when **V** command enabled.

These commands are only available when a program is stopped:

Q - Quit program, ESC key also. (same as executing **End**)

R - Run program.

T - Trace program.

S - Single step program.

I - Ignore current command and skip to next one.

L - toggle level trace mode for S & T commands. This enables you to pass through Gosubs, procedure calls and For...Next loops.

B - Back-up to previous instruction executed.

F - go forward to next instruction executed.

E - Evaluate an expression. The result of the evaluation will be printed out in the debugger window.

X - Execute a command.

K - toggle keylock mode. If keylock is enabled, you don't have to press CTRL/LEFT ALT to initiate debugger commands - simply type the command letter.

Debugger Features

Display Options

The debugger's display will overlay itself on any screen or if in Blitz Mode ontop of any slice. The H, V and M commands control the display options.

Hide will toggle the display on and off, View will toggle the debugger between being transparent (so you can see the background display) and opaque.

When in transparent mode, Mode will toggle the background display between hi-res and lo-res. Because only 2 bitplanes are displayed behind the debugger in transparent mode, the display will not always be perfect but should give the programmer an idea of what is happening.

Tracing program execution

The debugger allows the user to single step through or trace program execution, displaying in it's window which command is currently being executed.

Step is used to single step through your program, each time you press S the debugger will execute the command pointed to by the arrow and stop.

Trace steps continuously through the code displaying each command as it goes. To stop the Trace use the **C** command.

Level is used to change the trace level, if Level is toggled on, the debugger will not trace or single step through the inside of **For..Next** loops but execute normally until the loop exits.

It will also not trace the execution of any procedures or subroutines called, this is most useful for watching the program's main loop while not having to sit through the trace of each subroutine when called.

Resuming Normal Execution

Program execution can return normally after the debugger is activated using the **Run** command

If the debugger was activated using the **STOP** command the arrow will be pointing to the **STOP**, before continuing the command must be skipped over using the Igonore command. This is true for any command that has caused a RunTime error and invoked the debugger. To return to the editor from the debugger either the **Quit** command can be used or alternatively the ESC key.

Viewing command history

The debugger keeps a record of the commands executed before the program is stopped in a large buffer.

The Back-up command will step backwards from where the program halted, allowing the programmer to view the previous commands executed by the computer. A hollow arrow marks the current position in the history buffer.

The Forward command is used to step forwards through the history buffer, attempting to step past where the program was stopped will produce a *AT END OF BUFFER* error.

These features are invaluable to following through program execution up to where the program was halted. If a program halted in the middle of a subroutine or procedure you can step backwards to find where the routine was called from.

Direct Mode

While the debugger is activated the programmer has two tools available to examine the internal state of the program.

To find out the value of any variables the **Evaluate** command can be used. A prompt will appear, after typing the name of the variable and hitting return the value will be printed on the debugger display.

The **eXecute** command is used to run a Blitz2 command. A prompt will appear and the programmer can then type in any Blitz2 command such as CLS or n=20.

Debugger Errors

The following errors may occur when using the direct mode commands **Evaluate** and **eXecute**:

Can't Create in Direct Mode: Occurs if you try and Evaluate a variable that does not exist (hasn't been created) in the program.

Library Not Available in Direct Mode: Occurs when a Blitz2 command is eXecuted and is from a command library not used by the program. If the program does not use strings for instance, the string command library will not be part of the object code and so any string type commands will not be able to be eXecuted.

Not Enough Room in Direct Mode Buffer: This error should never occur, if it does the object buffer size in the Compiler Options requester should be increased.

AT END OF BUFFER: Occurs if the programmer tries to view Forward of where the program stopped (see viewing command history).

Using The Debugger

Your new Source Level Debugger:

What is it?
How do I use it?
Why should I?

Distributed with this issue of Blitz User is what, in my opinion, the single most important add-on ever for Blitz Basic: a Source Level Debugger (SLD).

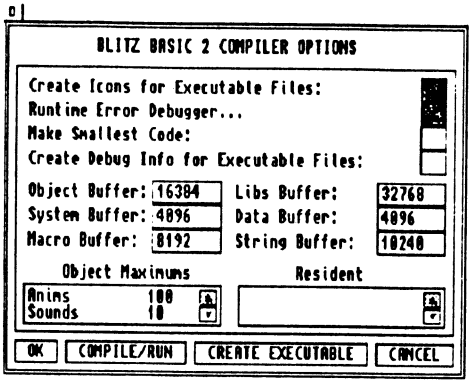
So what's this Debugger thing anyway?

As you'll be aware, errors in programmes that either mean the programme does not function properly or crashes are called BUGS. Well, a debugger aids you in your quest to remove these bugs from your programme to make it bug free.

Basically it's like having a can of flyspray to kill flies with instead of hitting and missing with a rolled up newspaper. The SLD distributed with Blitz2 is more than a debugger, its an error handler as well so without further adeu lets get into it...

What's Changed?

The first thing you will notice about the new version of Blitz2 is the compiler options requester. The following is a picture of the new options requester:



As you can see the check boxes at the top have been changed. They now consist of:

- Create Icons for Executables
- Runtime Error Debugger
- Make Smallest Code
- Create Debug Info

Make Smallest is now always set on when creating executables.

One important thing to note is the absence of the optimiser. This is intentional as it was found that it tended to be a bit shakey in the stability of the code it produced so its gone. A new improved optimiser is planned in the future.

The bit that really interests us is the Runtime Error Debugger. It's this which decides if you want the debugger active or not if this button is turned off your code will run at full speed, no error checking and will be as small as possible, great for final executables but not so hot for the development stage. If you click it off then on again you will find some sub options which we'll go through now:

Auto Run

With this on your programme will run as normal and the debugger will only cut in if

1. Theres a stop command somewhere in your programme flow

2. you press CNTRL-Alt-C (which now works reliably in both Amiga mode and Blitz mode!!!)

3. the programme halts on a runtime error or GURUS.

If this option is off however you will jump straight into the debugger and be able to step through your programme (more on this stepping bizzo later) from the first command.

Interupt Checking

Enabled, this option will be able to locate occurences of errors in any interupts. Unlike errors in the main program, code in interupts cannot be traced or single stepped, however execution can be continued after an interupt error (the offending interupt is disabled).

Overflow Checking

Kind of a mixed blessing this one, as a quick rundown if you have overflow checking on if a .b variable >127 or a .w variable >32768 it will perform an overflow error

Without trying to get technical thats NOT always what you want as those variables unsigned can actually hold up to 255 and 65535 respectively (unsigned means they can be positive numbers only, thus the last bit of the byte or word can be used to increase the capacity by the power of 2 instead of being used as a flag for a positive or negative number. The upshot of all this is if you:

a) Plan to use .b or .w variables as UNSIGNED and thus use larger values then turn overflow errors OFF

b) If you dont know what the hell I'm going on about then turn them OFF

c) If you understand what I'm on about you'll know when you can and can't use them anyway. (Ed: Jeeze wayne!)

OK lets DEBUG...

Here is an example programme for you to type in that we are going to debug:

```
For i = 1 to 10
    Nprint i
Next i
```

OK now compile and run it.... Hmmm no bugs, thats right none whatsoever, however we are going to use this example to demonstrate how you can SINGLE STEP through your code using the debugger, and check the status of variables and change them mid stream.

OK so go to your options requester and click on Runtime Error Debugger.

Enable the AUTO RUN option (this turns on the debugger straight away at the start of the programme) and compile and run the programme again.

WOW! with any luck a green box should have appeared at the bottom of your screen showing a couple of lines of your code with an arrow pointing at the current line. If not retrace your steps back and check which step you have missed out. If the debugger options are not there at all then you may not have installed the new software correctly.

If you're still with us then you've survived the hardest part. Before we start stepping through the programme heres a few things about the debugger.

All the debuggers commands are activated by pressing CNTRL and ALT and certain letter keys simultaneously.

Also note that the debugger window (unlike normal windows) is always activated so you do not need to click in it for it to receive input.

OK, STEP Time...

if you press CNTRL ALT S the arrow pointer in the debugger code should move to the next line. This means that the previous line of code (the For i = 1 to 10 statement) has been executed and the nprint line is now the current piece of code.

Right, so if you press CNTRL ALT S (hereafter refered to as STEP) then the number 1 will be printed to the current output window, so as you can see you are quite literally SINGLE STEPPING through your programme. Now hit the STEP key again and the 'next I' statement is executed and woosh... up the arrow goes back to the 'For I...' statement. So as you can see the arrow follows the programme flow exactly, groovy huh?

Direct Mode

Right, now to show you a couple more useful commands I want you to STEP though the whole for..next loop 3 more times so that the value of 'i' should now be 4.

You can check this by going CNTRL ALT E (hereafter called EVALUATE) you will get a small '>' prompt in the debugger window. Type 'i' and press return and it should give you the number 4 if you have done everything right.

EVALUATE enables you to view the current state of any variable, string, pointer etc.

Now the real groovy bit, hit CNTRL ALT X (hereafer called EXECUTE) and you get the '>' prompt again. Now type 'i=8' and hit the return key.

STEP another time through the loop and you will find that 8 is printed to the screen and i is incremented to 9, in other words you can interactivly execute functions or statements in the middle of a programme, there is a small limitation to this which I will discuss later.

STEP through the programme to the end (only one more iteration) and the debugger will return you to Ted, if you wish to terminate a programme or exit the debugger prematurely CNTRL ALT Q (no prizes for guessing that this means QUIT) will exit the debugger and return control to Blitz.

Once you are back in Blitz re-run the programme, only this time when the debugger window comes up press CNTRL ALT T (this one will be refered to as TRACE because it traces through the programme).

PHEW! the debugger window text would be zooming as if on steroids and the code will be executing at a slightly slower speed). This is handy to watch programme flow up to a point where you would CNTRL ALT C (pause programme and enter debugger called PAUSE hereafter) and either do some STEP, EVALUATE, or EXECUTE.

Now as I just said using trace you can zoom through lots of code at a reasonable speed and stop roughly in the right area of code but this has a couple of problems.

a) If you had a For Next loop with 10000 iterations even tracing through it would take an eternity

b) Stopping in roughly the right place is not such a great thing to have to do. It would be nice to be able to stop in exactly the right place.

Luckily you can, using the standard Blitz2 command **Stop, the** best way to explain is with example code so lets do it!

```
mymem.l = AllocMem_(1024,0)

If mymem     ;if the allocation worked
    For i = 0 To 1023
        Poke.b mymem+i,0
    Next i
    Stop     ;envoke debugger here
    NPrint "Memory clear succesfull"
Else
    NPrint "Allocation failed"
    End
EndIf
```

Ok so what's happened here is (by the way run this with your auto run OFF so that the debugger does not kick in at first) your programme runs at full speed, does the memory allocation, does the 1023 iteration mem clear loop and then Halts execution and enters the debugger right at your stop or break point.

You can then single step or whatever exactly from this point onwards, you can set as many stops as you like in your code. The next important command to tell you about is the RUN command CTL ALT-R.

If you are in the debugger and you've stepped through the bit your interested in, done some evaluation or whatever, and just want the programme to run as normal, using the RUN command will continue execution as normal.

One last command for assembly language programmers (or the curious) CTL-ALT D (will DISPLAY the contents of all the 68000 registers, the Status bits etc etc so if you are in assembly language code, or curious, you can display at any time using the debugger, at any rate it looks real impressive so try it once :))

As with programming no-one can teach you HOW to debug, everyone develops their own style however it basically boils down to a process of elimination, having modular code helps here, test a section of code or validity, if it passes eliminate it from your enquiries (rather like being a 'code policeman' really...)

Lastly a couple of things to remember..

a) If you're writing games ONCE their debugged TURN THE DEBUGGER OFF!

b) If you want ANY code to be as tight and fast as possible, then TURN OFF the debugger but makesure there's a

SetErr..*handler*..End Seterr

command in there somewhere so if something has been overlooked then you can at least exit cleanly without guruing someone's machine and taking other tasks with you.

THIS IS IMPORTANT!! sort of a programming ethic.

Seeya,

Roger.

Optimizing Code

Introduction

It is always important to have a firm grasp on how much time is being taken by certain routines to do certain things. The following are a few things to keep in mind when trying to get the best performance from your Blitz2 programs.

Performance is most important with arcade type games where a sluggish program will invariably destroy the playability of the game. However, it is also important in applications and other types of software to keep things as efficient as possible. Anything that makes the user wait will detract from the productivity of the package in general.

Algorithms

The most important key to optimising different routines is the overall approach taken to implementing them in the first place. There will always be half a dozen ways of approaching a problem giving half a dozen possible solutions. In programming, it is usually best to pick the solution that will produce the result in the quickest time.

Loops

When looking for ways to optimise a routine the best place to start is to examine the loops (for..next, while..wend etc.). The time it takes to perform the code inside a loop is multiplied by the number of times it loops. This may seem rather logical but often programmers will equate the number of lines of code in a routine to the time taken to execute it:

```
For i=1 to 100
    Nprint "hello"
Next
```

Will take exactly the same amount of time as typing:

```
For i=1 to 1
    Nprint "hello"
Next
```

the same as 300 lines of code!

Once one can visualise loops expanded out, the notion that if anything can be removed from inside a loop to before or after the loop then DO IT becomes rather obvious!

Lookup tables

Replacing numeric functions with look up tables is an effective way of gaining excellent speed increases. A look up table or LUT for short, is an array that contains all the possible solutions that the numeric function would be expected to provide.

The most common example of using LUPs for healthy speed increases is when using trig functions such as Sine or Cosine. Instead of calling the Sin function, an array containing a sine wave is created, the size of the array depends on the accuracy of the angle parameter in your program.

If a was an integer variable containing an angle between 0 and 360 we could replace any Sin functions such as x=Sin(a*180/pi) with x=sinlup(a) which will of course be more than 10 times as quick.

The array would be setup in the program initialisation as follows:

```
Dim sinlup(360)
For i=0 To 360
    sinlup(i)=Sin(i*180/pi)
Next
```

Using Pointers

When doing many operations on a particular subfield in a NewType a temporary pointer variable of the same subfield type can be created and that used instead of the larger (and slower) path name:

```
UsePath a(i)\alien\pos
```

replaced by:

```
UsePath *a
*a.pos=a(i)\alien
```

Testing Performance

Often it is important to test two different routines to see which offers the faster solution. The easiest way is to call each of them 5000 times or so and time which is quicker by hand.

When writing arcade games that will be performing a main loop each frame, it is useful to poke the background colour register before and after a specific routine to see how much of the frame it is using.

The following will show how much of a frame it takes to clear a bitmap:

```
While JoyB(0)=0
    VWait
    CLS
    ;poke background colour red
    MOVE #$f00,$dff180
Wend
```

Different colours can be used for different parts of the main loop. Remember that at the top of each slice the background colour will be reset.

Optimising Games

A quality arcade game should always run to a 50th, meaning the main loop always takes less than a frame to execute and so animation etc. are changed every frame giving the game that smooth professional feel.

This time frame means the programmer will often have to sacrifice certain elements in the game and maybe reduce colours and size of shapes to get the main loop fast enough.

The following are several methods for optimising main loop code in games:

• Disable Runtime Errors in the compiler options when testing speed of code as the error checker slows code dramatically.

• Poke the background colour register with different values between main routines to work out which ones are taking too long:

```
MainLoop:
    VWait
    Gosub movealiens:move.w $f00,$dff180  ;red
    Gosub drawaliens:move.w $0f0,$dff180  ;green
```

• Use QBlits if possible as they are the fastest way of implementing animated graphics in Blitz2.

• If aliens change direction using complex routines, split up the aliens into groups and every frame select a different group to have their directions changed, the others can move in the same direction until it is their turn to change. This method applies to any routines that do not have to happen every frame but can be spread across several frames in tidy chunks.

• Decrease the size of the display. During a frame, the display slows down the processor and blitter. A smaller display increases the amount of time given to the processor and blitter.

Conclusion

There is an infinite number of ways to increase the speed of Blitz2 code. Those developing games on machines with fast mem and faster processors should remember that most people do not have either! It is a good idea to disable fastmem when testing the speed of your code.
Again, always remember to disable the runitme error checker (debugger) when speed testing your program.

Simon