Thread: Wire GPU Tutorial: by Drunkie

11-28-2009 #1

Drunkie
Euphoria
WM HELPER
GLOBAL MOD
REGISTERED
Drunkie's Avatar
Join Date: Feb 2009
Location: Canada
Posts: 4,188
Blog Entries: 1

Wire GPU Tutorial: by Drunkie

Introduction

This tutorial is intended to guide you, not make you advanced with the GPU. Its a starting point for beginner and intermediate users. The best way to get better at the GPU is to learn through trial and error, and experience. The more you try to code with the GPU, the more you will get familiar with it and the faster you will learn. Try not to give up right away, I know it can be hard at times but you'll eventually understand the syntax and how the GPU works.

There's alot of useful opcodes you can use for math, comparisons, and subroutine calls/jumps. Make sure you read the following CPU tutorials because most of that content can be used in the GPU because they use the same language.

The "Foxy" CPU.
The "Foxy" CPU - Part 2

Also here's some raw documentation on the GPU as well as some example programs.

GPU Documentation
GPU Example Programs

Table of Contents

The Basics

Loading programs

When you first make a program you need to click "New File". When your done with it you need to save it. To load the program you type the file name you saved it as into the text box or click on the text file in the menu and click "Quick Load". Click the update button in the tool editor to make sure it's there.

GPU Model

To use a custom screen find the name of the prop you like, preferably square (PHX plates work well) and type this in your console. "wire_gpu_model <model name here>". For example, if I were to use the 2x2 phx plate I would type:

Code:
```
wire_gpu_model models/props_phx/construct/metal_plate2x2.mdl
```

Variable Types

When using variables, you need to specify which type it is. Here is a list of the most common data types you can use.

Code:
```
color
string
vec2f
vec3f
alloc
float
define
matrix
```

color will allow you to make a variable that holds a Red,Green,Blue,Alpha value
string will hold a piece of text
vec2f will make a vector with X and Y components
vec3f will make a vector with X, Y and Z components
alloc will declare a number variable
float will declare a number variable
define will make a constant number that cannot be changed.
matrix will create a matrix for 3D Projections

Comments

You can make a comment any time in your code by typing two "//" slashes

Code:
```
// I am a comment, I will not be executed!!
```

Ending statements

You may notice the ";" symbol in my code. It isn't necessary to use it if you put everything on seperate lines. You use it to tell the GPU that you are done with that statement. I personally use it because its good practice, and languages like C# require it after every statement. If you decide to use it, then you will be able to put multiple things on one line.

Important Things

Exiting the Frame

This is VERY important and always needs to be used or else we will be doing an infinite loop which lags like crazy. It will stop rendering the screen when you call it. Always put dexit at the bottom, and declare variables **UNDER** it. All of the code that tells the GPU what to do will be **ABOVE** dexit;

Code:
```
 dexit;
```

Hardware Clearing

Automatically after every frame in the GPU, it wil clear the screen. To make it so anything you draw stays on screen and is not cleared put this at the top of your code.

Code:
```
mov #regHWClear,0;
```

Otherwise, if you just want to clear the screen (Make it black) then use:

Code:
```
dclr;
```

Draw Color

The GPU will always need color. We call dcolor <name> to set the current color to draw. Always call dcolor before drawing and if you need another color then call it again with a different color variable.

Code:
```
 dcolor white;
```

Then below dcolor we put dexit and declare the variable "white".

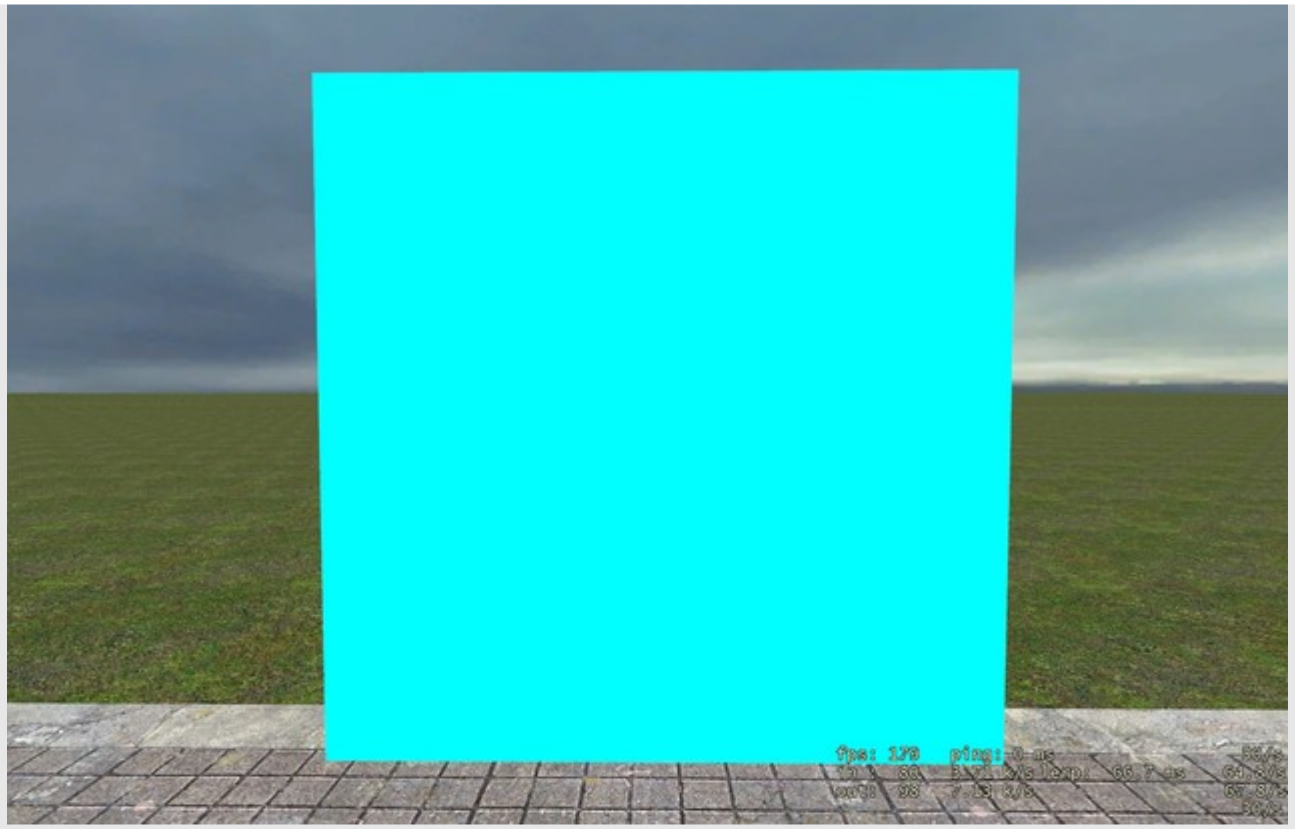
Code:
```
dexit;

color white,255,255,255;
```

4 parameters here, color name,R,G,B;
You can choose any variable name, I just use the color name white, because it is easy to identify it in the code. You can also add an additional parameter at the end for Alpha.

Background Color

Here we set the background color of the GPU

Code:
```
dclrscr blue;

dexit;

color blue,0,255,255;
```



Screen resolution

It is important to know that the GPU's screen resolution is 512x512. This will come in handy when determining positions to draw at on the screen. When you develop more advanced things it is also important to know that you can change the GPU to use different coordinates.

The opcode "**dcvxpipe <number>**" will change the screens coordinates, here are the ones you can use:
This is optional, if you dont use this then the coordinates will be defaulted to 0..512

Code:
```
0    Direct mapping
    X = X
    Y = Y

1    Screen resolution mapping
    X = 512 * (X / RSCREEN_WIDTH)
    Y = 512 * (Y / RSCREEN_HEIGHT)

2    Coordinates in 0..1 range
    X = 512 * X
    Y = 512 * Y

3    Coordinates in -1..1 range
    X = 256 + X * 256
    Y = 256 + Y * 256

4    Coordinates in -256..256 range
    X = X + 256
    Y = Y + 256
```

Primitive Shapes

Rectangles

First we set a color. Lets use blue.

Code:
```
dcolor blue;
```

Next we use one of two opcodes, either "**drect**" or "**drectwh**". **drect** will accept two positions and draw a rectangle from the top left to the bottom right or you can use **drectwh** which draws a rectangle from one position to a specified width and height.

Both opcodes will use 2 parameters that are vectors such as "**direct pos1,pos2;**" or "**drectwh pos,size;**" it's up to you to decide which one to use, but for now I will use drectwh.

Code:
```
drectwh pos,size;
```

We will now tell the GPU to exit by declaring **dexit**.

Code:
```
dexit;
```

Now, at the very bottom under dexit; we need to state the variables we will be using. Remember, always declare variables **UNDER** dexit.

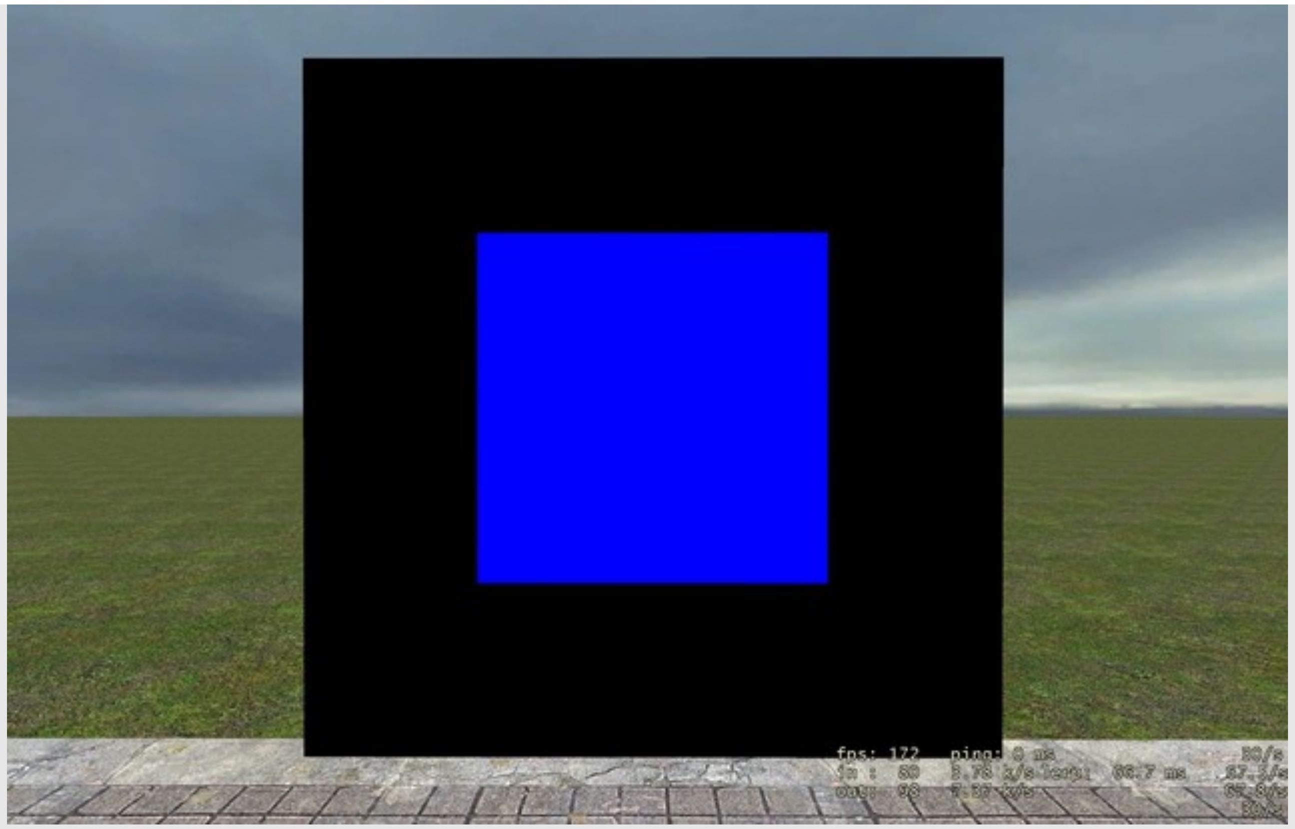
Code:
```
color blue,0,0,255;
vec2f pos,0,0;
vec2f size,128,128;
```

We use "color" to declare a color with 4 parameters "name,R,G,B"
We use "vec2f" to declare a vector with 3 parameters "name,X,Y"

Our final code will now be:

Code:
```
dcolor blue;
drectwh pos,size;

dexit;

color blue,0,0,255;
vec2f pos,128,128;
vec2f size,256,256;
```

This will now draw a rectangle (Well more of a square, but with similar results. Try stretching out the size and play around with it)



Circles

Note: Lots of circles create FPS lag. To reduce the lag you can change the number of points on a circle by changing the appropriate register. The code below will make the circle have 16 points, this is optional.

Code:
```
mov #regCircleQuality,16;
```

First we set a draw color, lets try green this time.

Code:
```
 dcolor green;
```

Now we use the **dcircle** command. It takes 2 parameters "**dcircle pos,radius**". Pos is a vector and radius is a number.

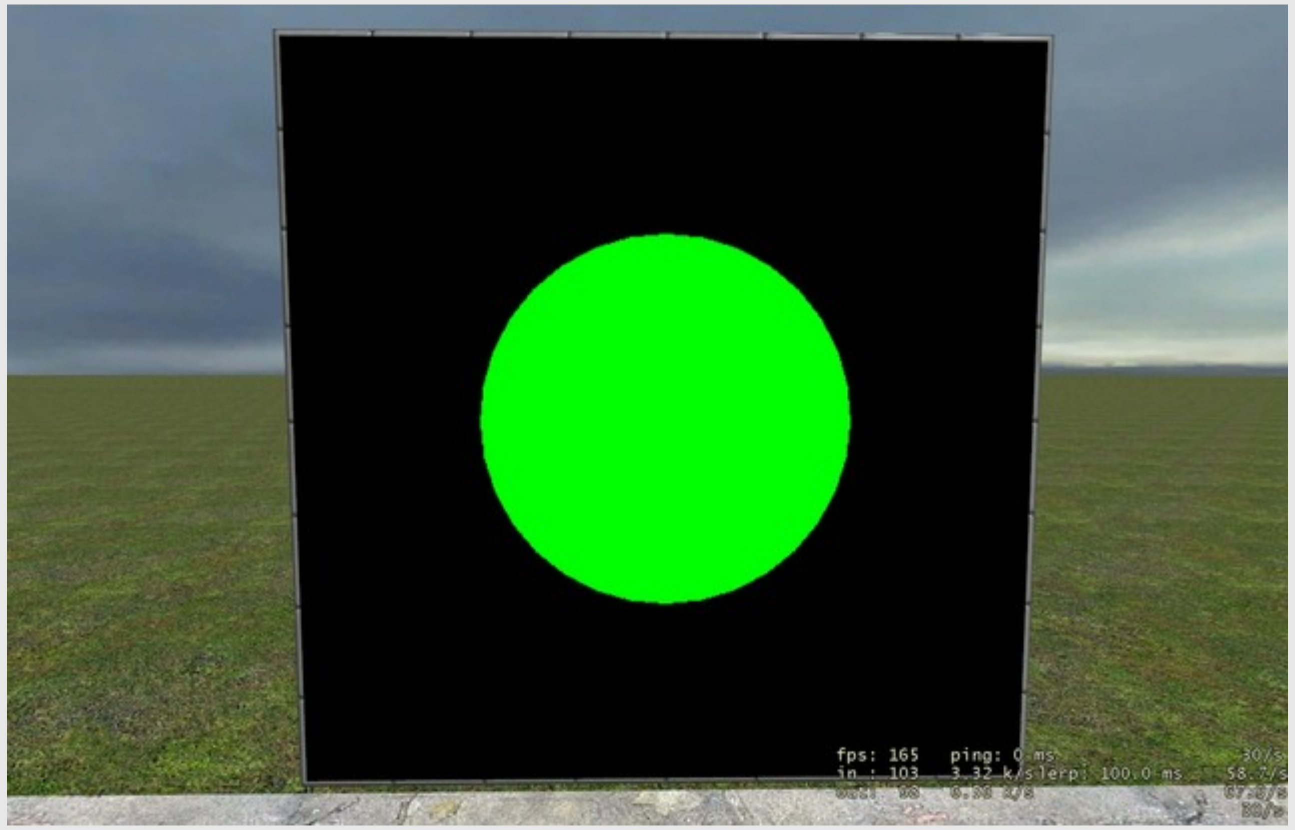
Code:
```
 dcircle pos,128;
```

The circle will have a size of 128. This is a hardcoded value, you can also refer to a variable which holds a value.

Okay so now we are done drawing, so we need to call dexit and declare our variables under it. The final code will be:

Code:

```
dcolor green;
dcircle pos,128;

dexit;

color green,0,255,0;
vec2f pos,256,256; // Center of the screen
```



Lines

Now lets draw a line
First we set our color, lets try red this time.

Code:
```
dcolor red;
```

Now, there is a command we can use to set the thickness of the line which is "**dsetwidth**". Lets set our line thicker before we draw it.

Code:
```
dsetwidth 32;
```

Our line will now have a thickness of 32 units.
Now to draw the line we use the "**dline**" command. It will accept 2 parameters which are positions. It draws a line from the first position to the second.

Code:

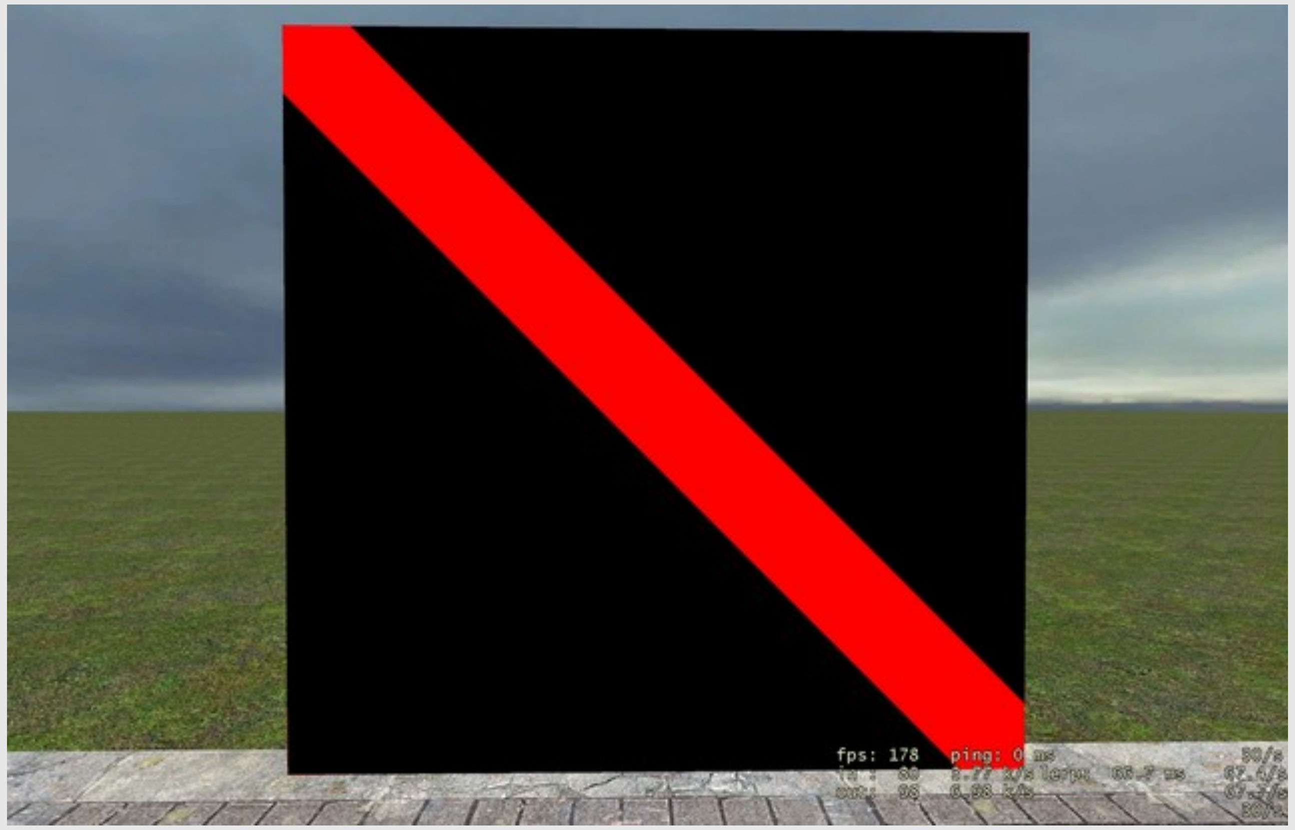
```
dline pos1,pos2;
```

Now we are done drawing, we call **dexit** and declare the variables we used under it. The final code is now:

Code:

```
dcolor red;
dsetwidth 32;
dline pos1,pos2;

dexit;

color red,255,0,0;
vec2f pos1,0,0; // Top left corner
vec2f pos2,512,512; // Bottom right corner
```

If done correctly this will draw a big red line going down the screen.



Custom Shapes (Polygons)

Note: You can only create convex shapes in order for this to work correctly.

Here we will draw a triangle. First we set the color.

Code:

```
dcolor green;
```

Now we call the opcode "**dvxdata_2f**". It accepts two parameters which are Label name, and amount of points to draw.

Code:
```
dvxdata_2f Triangle,3;
```

Now we declare dexit and our variables.

Code:
```
dexit;

color green,0,255,0;
Triangle: db 256,128; db 64,384; db 448,384;
```

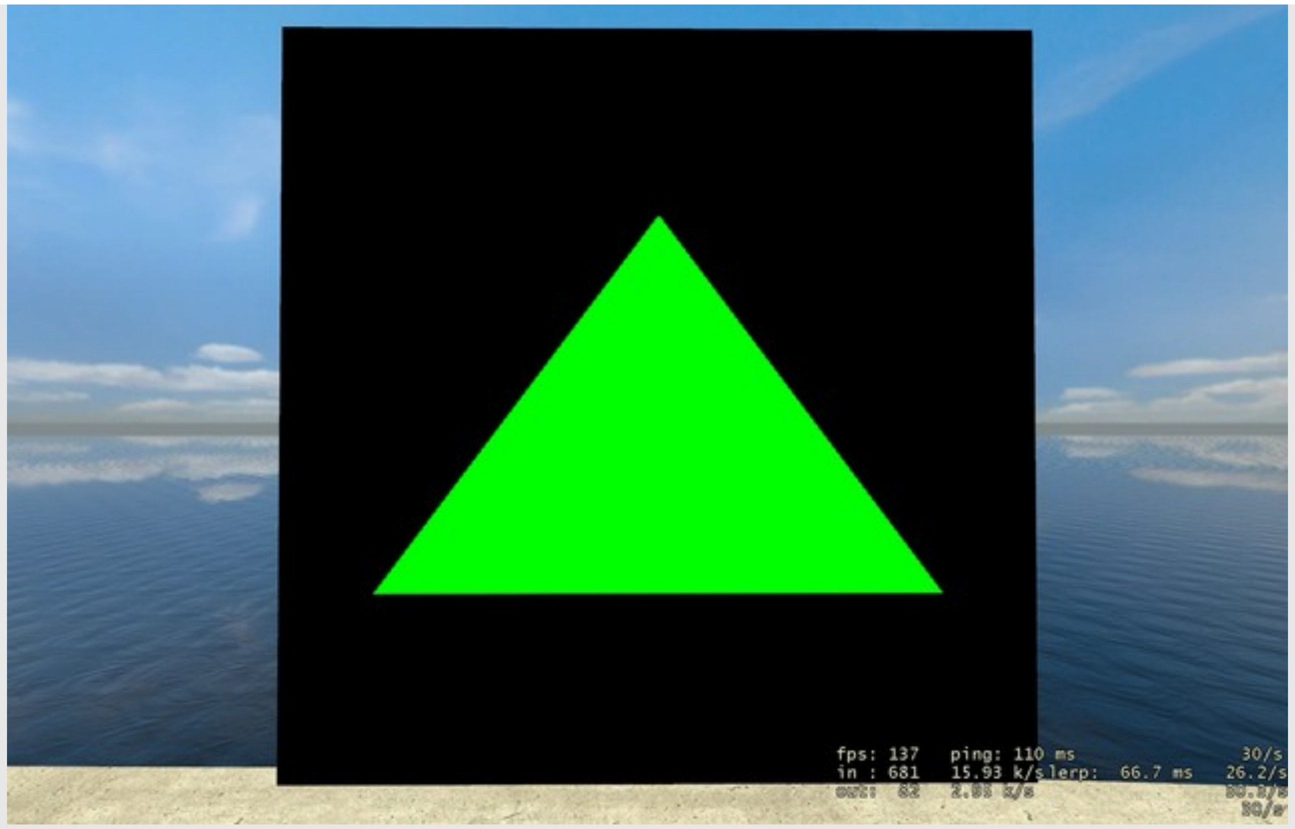
Triangle will be the name of the label that holds our points. We use something called "**db**" in the label to give it vector points to draw from.
Its kind of like "Connect-The-Dots", you give it vector points and it will connect them to draw a shape.

Earlier in the code when we said "**dvxdata_2f Triangle,3**", we are giving it the name and the number 3 represents how many "**db**" points we made. Since we made 3 db's, dvxdata_2f needs to have 3 as the second parameter.

Full Code:

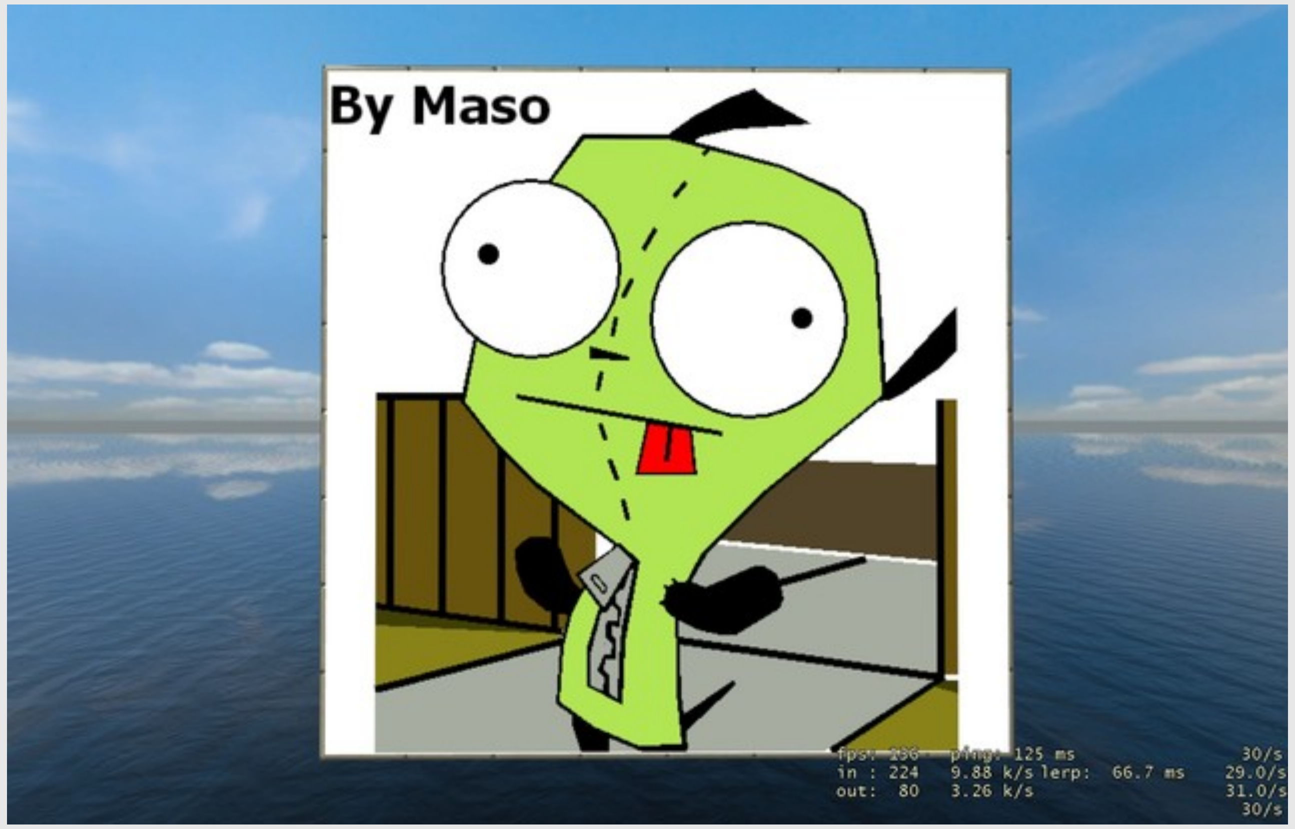
Code:
```
dcolor green;
dvxdata_2f Triangle,3;

dexit;
color green,0,255,0;
Triangle: db 256,128; db 64,384; db 448,384;
```

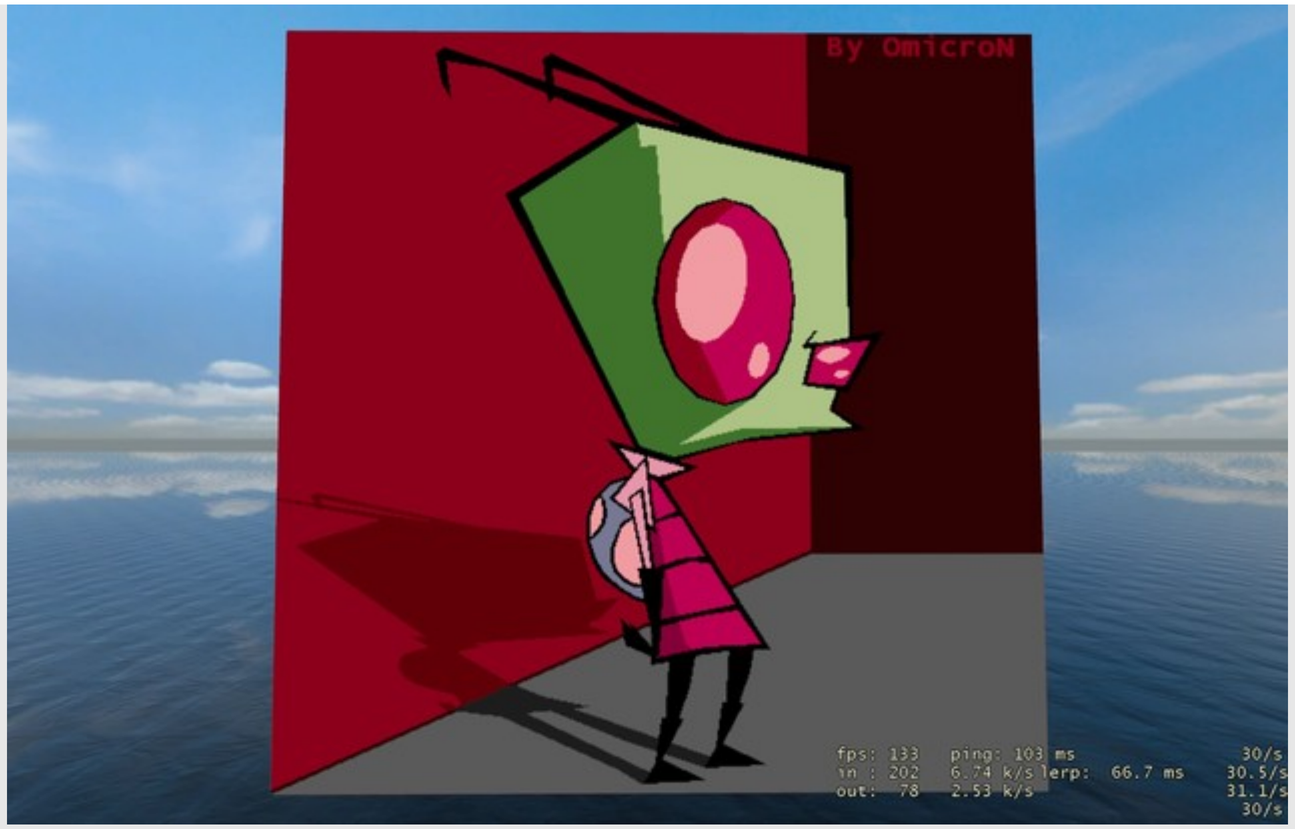
You might be thinking, wow... It's just a triangle, big deal.



But here are some examples of the amazing things you can do with polygons.

Credit goes to my buddies **Maso** and **OmicroN** for making these two.





Text Objects

Writing Strings

First we set a color as usual.

Code:
```
dcolor white;
```

Then we set the size of the text using "**dsetsize**"

Code:
```
dsetsize 64;
```

We can also change the font using "**dsetfont**"

These fonts are available to use

0 - Lucida Console
1 - Courier New
2 - Trebuchet
3 - Arial
4 - Times New Roman
5 - Coolvetica
6 - Akbar
7 - csd

It will default to Lucida Console by default if you dont set a font. Lets use the Trebuchet font. It's number association is 2 so we will write

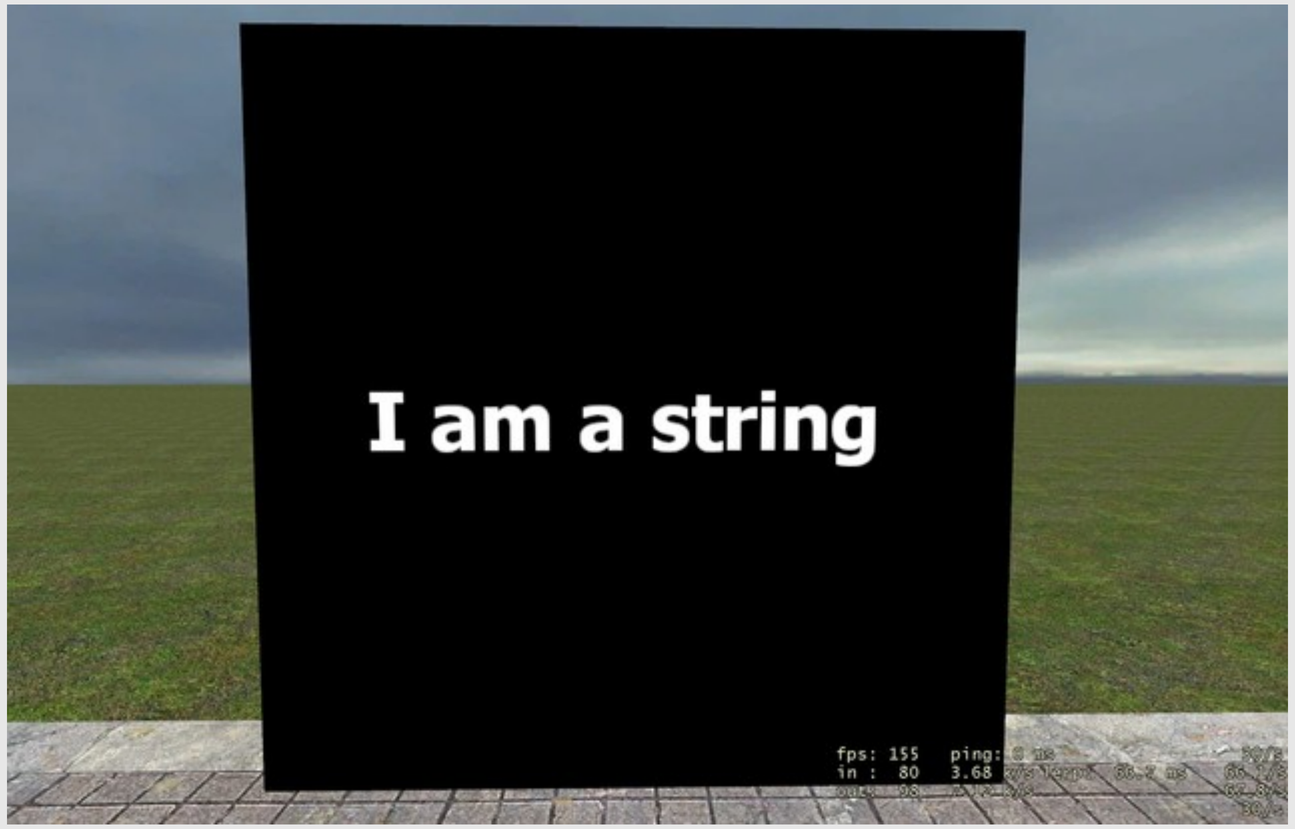
Code:
```
dsetfont 2;
```

Then we call "**dwrite**". It accepts two parameters which are the position and the string.

Code:
```
dwrite pos,text;
```

Then we do our usual dexit and declare the variables we used under it. The final code will be:

Code:
```
dcolor white;
dsetsize 64;
dsetfont 2;
dwrite pos,text;

dexit;

color white,255,255,255;
vec2f pos,76,224;
string text,'I am a string';
```



Writing Numbers

Code:
```
dcolor white;
```

Then we set the size of the text using "**dsetsize**"

Code:
```
dsetsize 64;
```

Then we call "**dwritei**". It accepts two parameters which are the position and the number.

Code:
```
dwritei pos,1337;
```

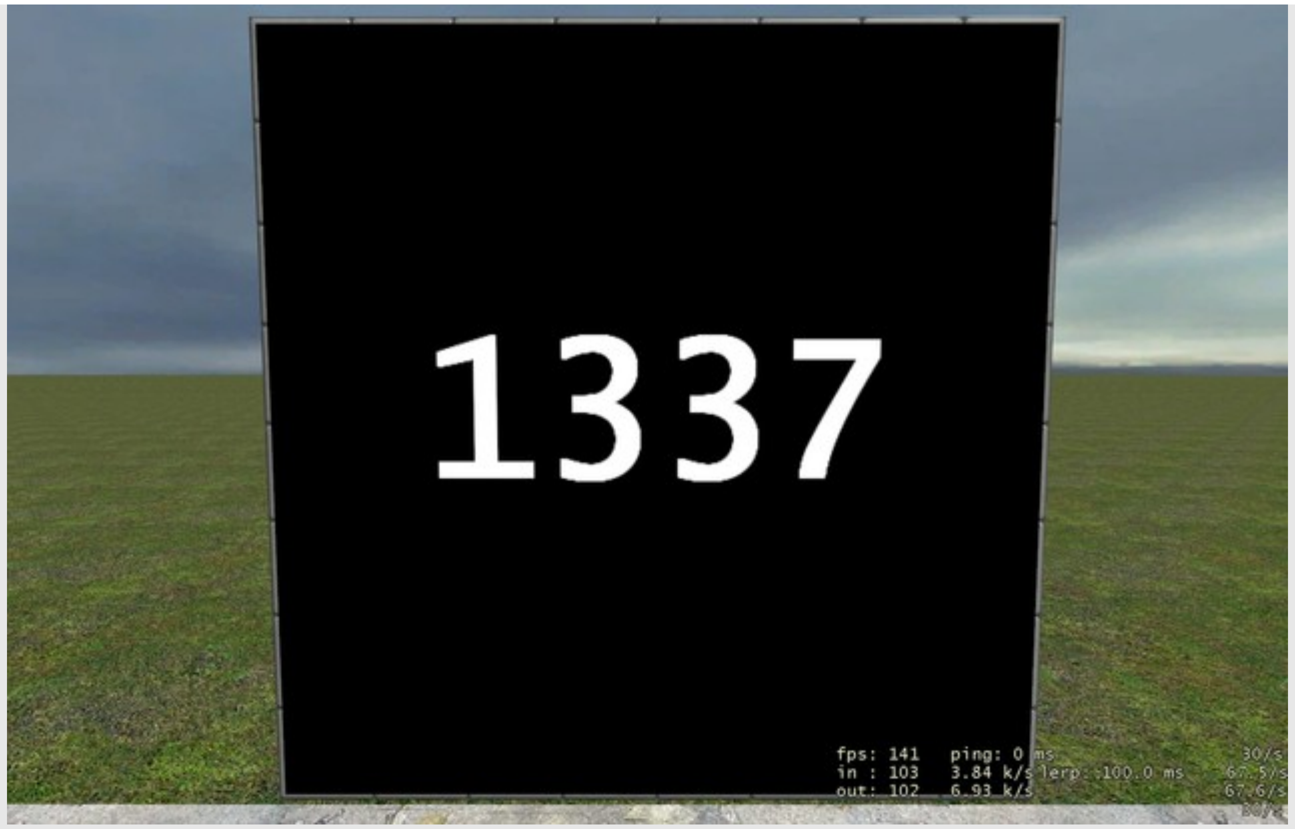
Then we do our usual dexit and declare the variables we used under it. The final code will be:

Code:
```
dcolor white;
dsetsize 128;
dwritei pos,1337;

dexit;

color white,255,255,255;
vec2f pos,100,192;
```

If we want to write the number held in a variable the code would be:

Code:
```
dcolor white;
dsetsize 128;
dwritei pos,#var;

dexit;

color white,255,255,255;
vec2f pos,100,192;
alloc var,1337;
```

The "#" symbol means we are referring to the data held in the variable.



Writing Strings with DB

This is useful if you dont want to declare a bunch of strings, and just want to display a long block of text.
Here's a code example of how to do this.

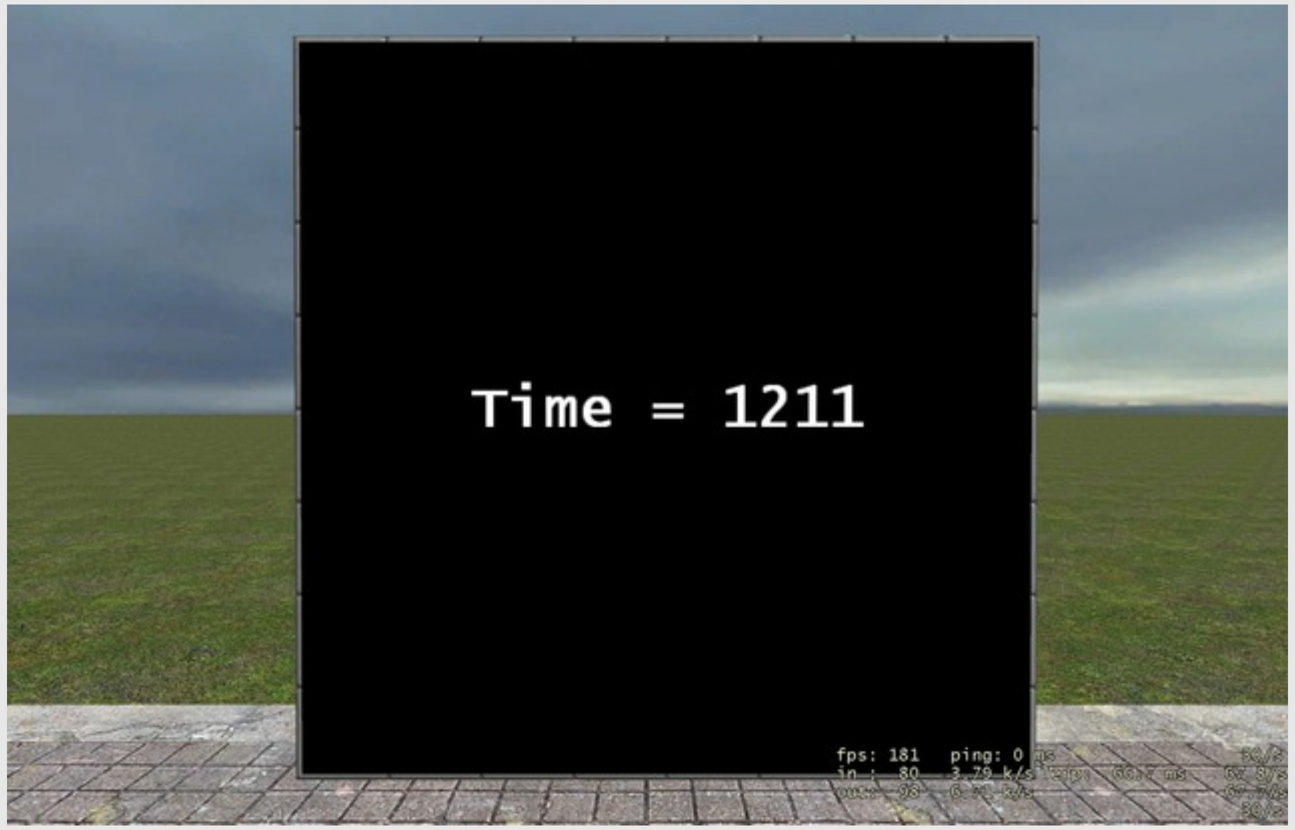
Code:
```
dcolor white;
dsetsize 24;
dwrite pos,text;

dexit;

vec2f pos;
color white,255,255,255;

// Text is a label containing strings
text:
db 'But theres no sense crying',10;
db 'over every mistake.',10;
db 'You just keep on trying',10;
db 'till you run out of cake.',10;
db 'And the science gets done.',10;
db 'And you make a neat gun',10;
db 'for the people who are',10;
db 'still alive.',0;
```

Your probably wondering "What are those 10's for?". Well in ASCII code the 10 represents the newline character. It makes the strings go to the next line. You can look at an ASCII table for the keys and insert any character you want there. This is basically one long concatenated string. You **NEED** to end the db code with a 0, to terminate the string and stop writing.

Writing Formatted Strings

Here an example of how to write a concatenated/formatted string with a number. REMEMBER to use **dwritefmt**, not **dwrite**.

Code:
```
dcolor white;
dsetsize 40;
timer #time; // Assign the variable time to server curtime
dwritefmt pos,text;

dexit;

vec2f pos,120,236;
color white,255,255,255;

string text,'Time = %i'; // the %i means it will take an integer/number
alloc time;
```



Word Wrap

This feature makes text goto a new line when a line is full.

dsettextwrap enables word wrap
dsettextbox sets the boundaries to wrap the words (Accepts a vec2f variable)
dtextwidth gets the width of text
dtextheight gets the height of text

Here's some example code of how to use it:

Code:
```
dsetsize 40;
```

```
dsetfont 2;
dsettextwrap 1;
dsettextbox textbox;
dtextwidth #size.x,text;
dtextheight #size.y,text;
dcolor bg_color;
drectwh pos,size;
dcolor white;
dwrite pos,text;

dexit;

color white,255,255,255;
color bg_color,0,150,0;

vec2f pos;
vec2f size;
vec2f textbox,384,384;

string text,'Hello there, \n\n My name is word-wrap! \n\n I make words goto the next line if they are too long!';
```

E2 Wirelink

Probably the part you are all wanting to know is how to interact with a GPU using E2 and wirelink. GPU ports reside in the address range **63488** to **64511** which means we have **1024** indexes for data storage. There's actually a TON of other addresses you can use for storage, but this address range is empty and has a sufficient amount of space. If you need more memory, then you can optionally use the address range **32768** to **63487**.

Note: If you are going to input values into the GPU whether it be X, Y, Z, coordinates or R, G, B colors, you have to input each seperate element of the variable.

GPU only accepts numbers into its registers.

Storing Numbers

In the **E2**, the code for storing number data looks like this:

Code:
```
@inputs GPU:wirelink

interval(1000)

GPU[63488] = curtime()

# Address 63488 now holds the curtime() timer value.
# Now we need the GPU to update

GPU[0] = GPU[0]

# This means we read cell 0, and write to cell 0, which updates the GPU.
```

In the **GPU**, we can now use this data, lets try writing the number to the screen using this code.

Code:
```
dcolor white;
dsetsize 128;
dwritei pos,#63488;

dexit;

color white,255,255,255;
```

```
vec2f pos,0,0;
```

Remember, we need to use the "#" symbol to tell it that we want the data held in that address number. If done correctly, the number on the screen will gradually increment each second.

Storing Strings

The **E2** code to store strings is:

Code:
```
@inputs GPU:wirelink

interval(1000)

GPU:writeString(63488,"LoL")

# Address 63488 now refers to the string "LoL"

GPU[0] = GPU[0]
```

The **GPU** will store the string like this, so be careful not to overwrite any of the addresses which store the data of the string.

63488 = L
63489 = o
63490 = L

It actually stores them as bytes, but I am just showing them as char's in that example.

The **GPU** code for using the string will be:

Code:
```
dcolor white;
dsetsize 128;
dwrite pos,63488;

dexit;

color white,255,255,255;
vec2f pos,140,192
```

We dont use the "#" symbol here because we dont want the data in 63488, we want 63488 as a pointer to a null-terminated string.

Loops

Loops can be done in GPU quite easily. It is very important to understand though, the GPU updates constantly; you may create a bad loop and lag everyone on a server badly. When dealing with loops in the GPU you need to make sure they work perfectly, otherwise your program will have errors.

Example

The following code is example of a working loop; comments included.

Code:
```
// Loops in the GPU

dcolor white; // Set our draw color
dsetsize 16; // Set text size to 16
mov #pos.y,0; // Reset Position Y
mov #index,0; // Reset our incrementer
jmp Render; // Jump to the Render label

Render:
    cmp #index,32; // Compare index to 32
    jge Exit; // Jump to Exit if index >= 32
    add #index,1; // Increment the looping variable
    dwritei pos,#index; // Write the index number to screen
    add #pos.y,16; // Increment the position for the text
jmp Render; // Keep looping back to the same label

Exit: dexit;

// Variables
//======================
color white,255,255,255;
alloc index; // Our incrementing variable
vec2f pos;
```

Registers

Editing a Register

If you want to change the registers value you use the "**mov**" opcode. For example, if I wanted to turn off the CLK in the GPU I would do:

Code:
```
mov #65535,0;
```

We use the "#" symbol to tell it that we want to change the data inside of that register.

List of Registers

Thanks to Black Phoenix for this list.

[65535] - CLK (can be used to turn on and turn off the GPU)
[65534] - RESET (will reset GPU RAM state to original values, just as if you reloaded GPU program)
[65533] - HARDWARE CLEAR (usually set to 1. If you set it to 0, it will stop clearing background, allowing you to add some rendering to PREVIOUS frame.
[65531] - HALT (will freeze GPU: it will stop executing code, and image on screen will be frozen, it will remain same until you unhalt the GPU)
[65530] - RAM_RESET (resets GPU RAM contents to zero, clears all data and code permanently)

[65525] - Horizontal image scale, **[65524]** - Vertical image scale
These two will modify image size on screen - you can use this to add letterboxing, or cropping

[65523] - Hardware scale
A tricky register. Positive values (> 0) will zoom in the image (without breaking letterboxing). Zero will make image remain

same scale. Negative values should zoom out - if the image becomes too small, it will be tiled (you can have 9 copies of your GPU screen this way, or even mores).

[65522] - Rotation (0 - 0*, 1 - 90*, 2 - 180*, 3 - 270*)
Just rotate the screen.

[65518] - Raster quality
Quality of output graphics. This doesn't work for fonts, only for vector and 3d graphics. It doesn't affect speed, but it's a very neat effect. Values are
0..512, try it.

[65515] - Image width (800), **[65514]** - Image height (600)
Framebuffer size (sort of, it actually stays 512x512, but it remaps all graphics like that. You need to enable one of the coordinate pipes)

[65513] - Real screen ratio
Read-only register, gives you screen ratio (4:3, etc)

[65512] - Parameter list address (for dwritefmt)
You can override location from where dwritefmt reads parameter list (usually it's located after string, like this: string "%f is how much money I have"; float money;
But you can have just the string "I like %f";, and parameters will be read from this location instead.

[65505] - Cursor X (0..1), **[65504]** - Cursor Y (0..1)
GPU cursor position. Currently clientside only, will be serverside eventually

[65503] - Cursor visible
Disable/enable some simple GPU hardware cursor rendering.

[65495] - Brightness W, **[65494]** - Brightness R, **[65493]** - Brightness G, **[65492]** - Brightness B
[65491] - Contrast W, **[65490]** - Contrast R, **[65489]** - Contrast G, **[65488]** - Contrast B
Allows you to adjust brightness and contrast for separate R G B channels, and general contrast and brightness (W).

[65485] - Circle quality (3..128)
Number of points in circle

[65484] - Offset Point X, **[65483]** - Offset Point Y
dmove sets these registers from a vector

[65482] - Rotation (rad), **[65481]** - Scale
drotatescale sets these registers from a vector

[65480] - Center point X, **[65479]** - Center point Y
Centerpoint of rotation. Usually 0;0, can be something else (but I advise you against using this).

[65478] - Circle start (rad), **[65477]** - Circle end (rad)
Was explained in tutorial, circle size.

[65476] - Line width (1)
Width of line (dsetwidth sets this register)

[65475] - Scale X, **[65474]** - Scale Y
Scaling (scales around centerpoint of rotation), separate on X,Y axes.

[65473] - Font align
Font align. 0 is left align (default), 1 is centered text, 2 is aligned from right, 3 is aligned from top, and 4 is aligned from bottom.

[65472] - ZOffset
This will add some offset to Z coordinate of currently drawn stuff. In some vertex pipes this allows you to make cheap 3D

effects by drawing only 2d polygons.

Last edited by Drunkie; 10-06-2010 at 10:33 PM.

Reply With Quote

11-28-2009 #2

Drunkie

Euphoria

WM HELPER

GLOBAL MOD

REGISTERED

Drunkie's Avatar

Join Date:	Feb 2009
Location:	Canada
Posts:	4,188
Blog Entries:	1

re: Wire GPU Tutorial: by Drunkie

Examples

Rotating RGB Circle

Here's what we will be making.

First we need to change the coordinate system by doing:

Code:
```
dcvxpipe 3;
```

This will change the coordinates to -1 to 1, instead of 0 to 512

Then we need to do some math, I will be using the registers eax and ebx. These registers are just places used to store data so we dont need to make a variable.

Code:
```
timer eax; mul eax,20; mod eax,360;
timer ebx; mul ebx,0.5; mod ebx,1.5;
```

I will explain this a bit more. First we set **eax** to a timer, then we multiply it by 20, then we use modulus on it. Modulus of 360 will make it reset to 0 once it reaches the value 360. **Eax** will be used for the rotation of the screen.

Now we set **ebx** to a timer, multiply it by 0.5, and do a modulus of 1.5 on it. When the value reaches 1.5 it will reset to 0. **Ebx** will be used for the scaling of the screen.

Now we rotate and scale the screen using the eax and ebx values. We use "**drotatescale rotation,scale**"

Code:
```
drotatescale eax,ebx;
```

Now we need to cut the circle so it makes 1/3 of a normal circle.

Code:
```
mov #regCircleStart,0; mov #regCircleEnd,2.09;
```

The "**mov**" opcode just means that we are making a value equal something. The variables **regCircleStart** and **regCircleEnd** exist so we dont have to remember the cell number that deals with trimming the circles.

Now we set a color, and draw a circle.

Code:
```
dcolor red;
dcircle pos,1;
```

We repeat the above steps 2 more times to create the other circle parts.

Code:
```
mov #regCircleStart,2.09; mov #regCircleEnd,4.18;
dcolor blue;
dcircle pos,1;

mov #regCircleStart,4.18; mov #regCircleEnd,6.28;
dcolor green;
dcircle pos,1;
```

Now we declare the variables we used under dexit;

The full code:

Code:
```
dcvxpipe 3;

timer eax; mul eax,20; mod eax,360;
timer ebx; mul ebx,0.5; mod ebx,1.5;
drotatescale eax,ebx;

mov #regCircleStart,0; mov #regCircleEnd,2.09;
dcolor red;
dcircle pos,1;

mov #regCircleStart,2.09; mov #regCircleEnd,4.18;
dcolor blue;
dcircle pos,1;

mov #regCircleStart,4.18; mov #regCircleEnd,6.28;
dcolor green;
dcircle pos,1;

dexit;

color red,255,0,0;
color green,0,255,0;
color blue,0,0,255;
vec2f pos,0,0;
```

Cursor

Fairly simple setup here. We will be dealing with registers, so feel free to look at the list of registers if you need to clarify anything.

First we will enable the cursor, by setting this particular register to 1.

Code:
```
mov #regCursor,1;
```

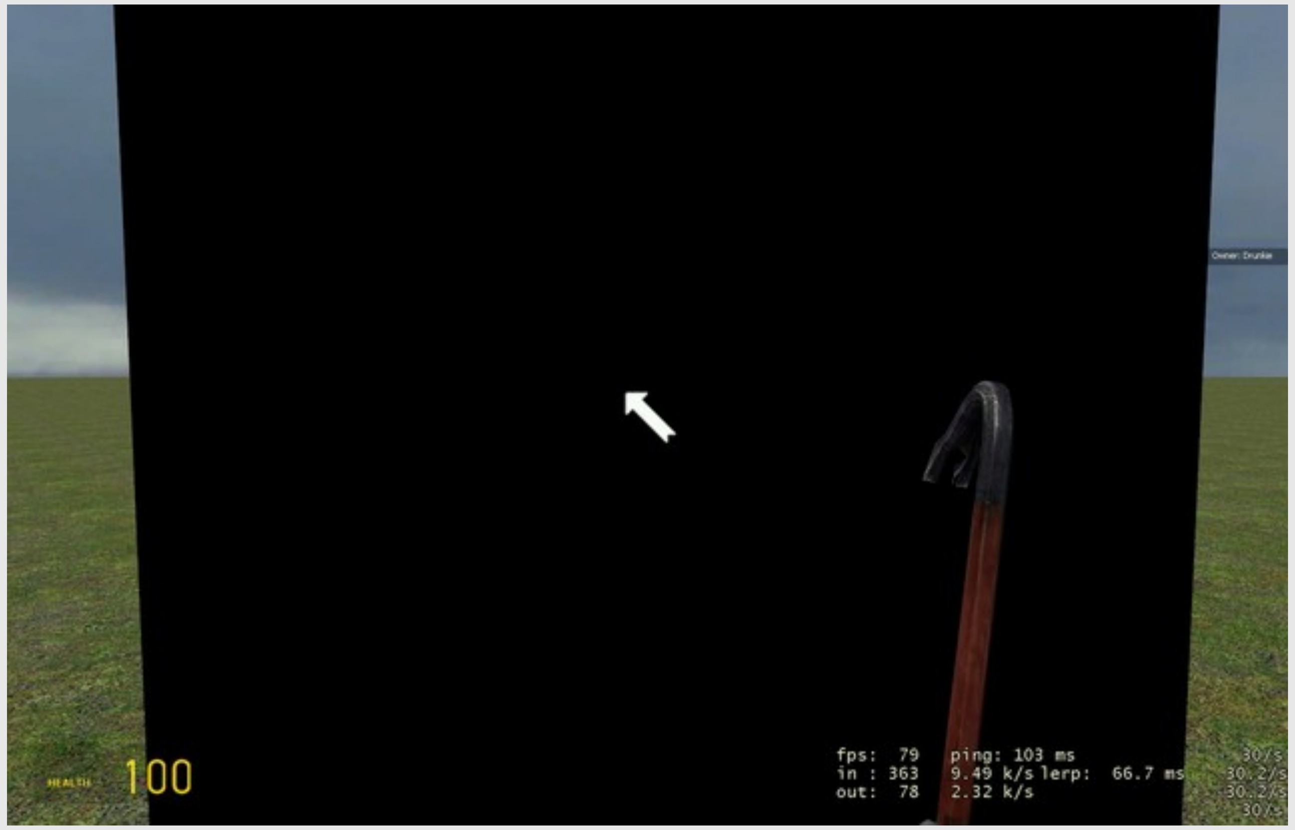
Then we will set the X and Y components of one of our variables to the registers that return the X and Y coordinates of your aim position.

Code:
```
mov #pos.x,#regCursorX;
mov #pos.y,#regCursorY;
```

Full Code:

Code:
```
mov #regCursor,1;
mov #pos.x,#regCursorX;
mov #pos.y,#regCursorY;

dexit;

vec2f pos;
```

This isn't anything extraordinary, it will just make a cursor on the screen. Whatever you use it for is up to you.



Rainbow Swirl

Here's what we will be making:

First off, we need to make some initial changes and alter the coordinate pipe:

Code:
```
mov #regHWClear,0;
mov #regCircleQuality,4;
dcvxpipe 3;
```

We set Hardware Clear to 0 (Let the PREVIOUS frame render on the GPU)
Change the circle quality to 4 (Four points on a circle)

Change to coordinate pipe 3 (-1..1 Coordinates)

We have now set the appropriate registers pertaining to the program.
We will now do some basic math.

Code:
```
timer eax; mul eax,3; // Create a timer and multiply it by 3.
fcos #col.r,eax; mul #col.r,127; add #col.r,128; add eax,1.57; // Red
fcos #col.g,eax; mul #col.g,127; add #col.g,128; add eax,1.57; // Green
fcos #col.b,eax; mul #col.b,127; add #col.b,128; add eax,1.57; // Blue
dcolor col; // Set rainbow color to our variable
```

I will explain this code a bit more. First we assign a timer to a blank register named 'eax';
Then we perform a simple cosine function on it. This will make the color slowly change colors.
Basically all this code does is make a color transform from red -> green -> blue.

Now we have to perform rotations and size of the circle (Note: It's still a circle, we just changed the number of points to 4, which now makes it a square)

Code:
```
timer eax; mul eax,0.6; fsin eax,eax; // Rotation value
timer ebx; mul ebx,0.5; fsin ebx,ebx; // Size value
drotatescale eax,1;
dcircle pos,ebx;
```

So first, we assign the register 'eax' to a timer. Then we multiply it by 0.6 to make it go slower. Then we use sine on it.
Now we assign the register 'ebx' to a timer and multiply it's value by 0.5. Now we use sine on that value as well.
Then we call the opcode 'drotatescale', giving it the parameters of eax and 1.
Essentially this means we rotate the screen based on register 'eax' value, and scale it to 1 (No scaling at all)
Finallly we render the circle at the defined position, with the value held in the 'ebx' register.

Now we define the variables we used under the opcode 'dexit'.

Code:
```
dexit;

color col;
vec2f pos;
```

Full Code:

Code:
```
mov #regHWClear,0;
mov #regCircleQuality,4;

dcvxpipe 3;

// Color
timer eax; mul eax,3;
fcos #col.r,eax; mul #col.r,127; add #col.r,128; add eax,1.57;
fcos #col.g,eax; mul #col.g,127; add #col.g,128; add eax,1.57;
fcos #col.b,eax; mul #col.b,127; add #col.b,128; add eax,1.57;
dcolor col;

// Rotate and Scale
timer eax; mul eax,0.6; fsin eax,eax;
timer ebx; mul ebx,0.5; fsin ebx,ebx;
drotatescale eax,1;
dcircle pos,ebx;
```

```
dexit;

color col;
vec2f pos;
```

Last edited by Drunkie; 08-24-2010 at 11:31 PM.

Reply With Quote

11-28-2009 #3

ryland

Wire Sofaking
wm REGISTERED
ryland's Avatar
Join Date: Oct 2009
Location: Card bord box next to wal-mart.
Posts: 594

re: Wire GPU Tutorial: by Drunkie

THANK YOU DRUNKE!!!

I have not read it yet(im on my iphone)
BUT thank you, ive been waiting for this for a long time.

"I like pie"-Jat Goodwin

<Azrael-> ryland: LOL is such a noobish thing to say.
<ryland> LOL
<Fox682> LOLLOL
<Fox682> LOL
<ryland> LOL

Reply With Quote

11-28-2009 #4

McLovin

◕_◕
wm WM DEVELOPER
wm GLOBAL MOD
wm REGISTERED
McLovin's Avatar
Join Date: Sep 2008
Location: Batman, Turkey
Posts: 1,716
Blog Entries: 3

re: Wire GPU Tutorial: by Drunkie

Nice tutorial. :3

Filipe *- you never know what ends up in your love hole*
Poonage *- It's not a disorder, it can be solved*
Drunkie *- vista was on the trail of complete beauty and perfection*
Omicron *- cable management is for losers, and people who know what they're doing*
JatGoodwin *- I don't normally suck dick, but when I do they call me immibis*
Drunkie *- THERE IS SIMPLY NOT ENOUGH DICKS*
Danking *- also, I am pretty*

__JatGoodwin__ - Also put a magnet under your pillow. It increases beta brainwave activity and can cause miracles to happen.
__Drunkie__ - I feel manlier when I use my ban hammer
__Foss__ - i am commander sheppet and this is my favurt
__Encyclopedia Dramatica__ - Java is like Alzheimers; it starts slow and eventually, it takes away all of your memory.

Reply With Quote

11-28-2009 #5

Matte

Wirezard
WM DEVELOPER
GLOBAL MOD
REGISTERED
Matte's Avatar
Join Date: Jan 2009
Location: Norway
Posts: 3,113

re: Wire GPU Tutorial: by Drunkie

Looks like a really well written tutorial.

Great job!

Really nitpicking (You asked for it):

> *This part will get intense, I only suggest trying it if you think **your** up to the challenge.*

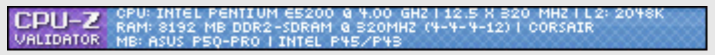
Dont should be don't several places.

> *This means we can store a hell of a lot more data than a dataport which has only 8 ports.*

The GPU can actually have 1024 dataports.

These are tiny errors, but don't blame me for nitpicking.

"If anybody says he can think about quantum physics without getting giddy, that only shows he has not understood the first thing about them."
-- Niels Bohr


CPU-Z VALIDATOR
CPU: INTEL PENTIUM E5200 @ 4.00 GHZ | 12.5 X 320 MHZ | L2: 2048K
RAM: 8192 MB DDR2-SDRAM @ 320MHZ (4-4-4-12) | CORSAIR
MB: ASUS P5Q-PRO | INTEL P45/P43

Ask me anything.

Wire FPGA

Reply With Quote

11-28-2009 #6

Drunkie

Euphoria
WM HELPER
GLOBAL MOD
REGISTERED
Drunkie's Avatar
Join Date: Feb 2009
Location: Canada
Posts: 4,188
Blog Entries: 1

re: Wire GPU Tutorial: by Drunkie

Thanks for the positive feedback so far.

@Matte

I will correct those, thanks for spotting them out!
When I said 8 ports, I actually ment to say there is only 8 **inputs** on the data port gate.

Last edited by Drunkie; 11-28-2009 at 05:20 PM.

Reply With Quote

11-28-2009 #7

N00bDud3

Wire Sofaking
wm REGISTERED
N00bDud3's Avatar
Join Date: Jul 2009
Location: Error: Unknown Location
Posts: 1,079

re: Wire GPU Tutorial: by Drunkie

That seems really useful.
Is there any way to just rotate certain shapes? Or do you have to rotate the whole screen?

Reply With Quote

11-28-2009 #8

jesse1412

Wirererer
wm REGISTERED
jesse1412's Avatar
Join Date: Sep 2009
Location: in your basement
Posts: 315

re: Wire GPU Tutorial: by Drunkie

i love u drunkie 😊

Reply With Quote

11-28-2009 #9

Drunkie

Euphoria
wm WM HELPER
wm GLOBAL MOD
wm REGISTERED
Drunkie's Avatar
Join Date: Feb 2009
Location: Canada
Posts: 4,188
Blog Entries: 1

re: Wire GPU Tutorial: by Drunkie

> Originally Posted by **fireeater67**
>
> *That seems really useful.*
> *Is there any way to just rotate certain shapes? Or do you have to rotate the whole screen?*

I am about 80% sure it rotates the whole screen, unless you can first rotate it, draw a shape, then rotate it back.
Also, you cannot rotate text.

Reply With Quote

11-28-2009 #10

ryland

Wire Sofaking

ryland's Avatar

Join Date: Oct 2009
Location: Card bord box next to wal-mart.
Posts: 594

re: Wire GPU Tutorial: by Drunkie

With your tutorial I made a Music Player with the GPU, thank you.

Here is the E2.

Code:
```
@name Cake's GPU Music Player
@inputs GPU:wirelink On KeyIn
@outputs
@persist Array:array String:string Clear Key
@trigger

#Sets up the keyboard
Key = KeyIn

if(Clear) {String = ""}

if(KeyIn == 127) {String = String:sub(1,String:length()-1)}

if(KeyIn != 127 & KeyIn != 154) {String = String + toChar(KeyIn)}

if(Key == 13 & $Key) {Enter = 1} else {Enter = 0}

if(On == 1)
{
    Array = String:explode(" ") #sets up the array
    GPU:writeString(63488,String) #sends the string to the GPU


    if(Enter == 1)
    {
        #Commands here
        if(Array[1,string] == "/play")
        {
            soundPlay(1000,1000,Array[2,string])
            soundVolume(1000,1000)
            GPU:writeString(63530,Array[2,string])
```

And here is the GPU code.

Code:
```
dclrscr Surrounding; //clears the screen

dcolor blue; //For everythin that is the color Blue.
dsetsize 20; //sets the text size
dwrite pos,63488; //where the string will be
dwrite Song,Sname; //shows the "Now Playing:" message
dwrite P,pitch; //shows what the pitch is

dcolor white; //For everything white
dwrite top,title; //For the message "Cakes Ipod V2"
dwrite Playing,63530; //Shows the song playing
dwrite P1,63590; //Shows where the pitch value is put
dwrite c1,Credits; //For the message "Made by the cake is a lie."
```

```
dsetsize 14; //Sets the text size to 14
dwrite help,63620; //For if you type /help

dcolor bg; //Where everything that is red is placed
drect l1,l2;
drect l3,l4;
drect l5,l6;
drect l7,l8;
drect l9,l10;
drect l11,l12;

dexit;

vec2f pos,40,450; //Pos of the string you type in.

vec2f top,170,35; //Pos for "Cakes Ipod V2"

vec2f Playing,50,345; //Pos of the song that is playing
```

Thank you sooo much, ive been wanting to use GPU for a while everyone is welcome to try out my music player.

"I like pie"-Jat Goodwin

<Azrael-> ryland: LOL is such a noobish thing to say.
<ryland> LOL
<Fox682> LOLLOL
<Fox682> LOL
<ryland> LOL

Reply With Quote

+ Reply to Thread

Page 1 of 27 1 2 3 11 ... Last

« Previous Thread | Next Thread »

LinkBacks (?)

NMD Forums Refback This thread	03-02-2010, 01:10 PM
YouTube - GPU Example #2 Refback This thread	02-14-2010, 05:42 AM
YouTube - GPU Example #1 Refback This thread	02-14-2010, 03:41 AM

Similar Threads

Slot Machine [GPU + E2] By Drunkie in forum Finished contraptions	Replies: 69 Last Post: 11-13-2010, 03:39 PM
Drunkie's Arcade Game - Skee Ball By Drunkie in forum Finished contraptions	Replies: 21 Last Post: 11-06-2010, 06:06 PM
Drunkie's Pub [Updated Wire] [FastDL] By Drunkie in forum Servers	Replies: 340 Last Post: 06-07-2010, 11:06 PM
DrunKie's iPod custom? By Fear57 in forum Wiremod General Chat	Replies: 19 Last Post: 12-01-2009, 02:56 PM

Tags for this Thread

drunkie gpu tutorial zasm
View Tag Cloud

Bookmarks

- Digg
- del.icio.us
- StumbleUpon
- Google

Posting Permissions

You may not post new threads	**BB code** is On
You may not post replies	**Smilies** are On
You may not post attachments	**[IMG]** code is On
You may not edit your posts	HTML code is Off
	Trackbacks are On
Pingbacks are On	
Refbacks are On	

Forum Rules