Complete WolfcamQL tutorial

Written by earth_quake in November 2013. Update for WolfcamQL 10.3

Please distribute, but credit earth_quake. I hope this is useful!

Post problems to "WolfcamQL :D" on QL forums, or visit #WolfcamQL in IRC

Basics

Important Terms

Cvar: console variable

Var: variable

Cvars are given in the following manner ***/cg_autoaction <0|1|2|3>***

"/cg_autoaction" is the cvar. You then leave a space and enter a value; anything inside "<>" is what you can enter. The "|" shows you the different possible values. Sometimes this will be a range, such as "1-360"

Config*:* configuration file

Fx*:* Short for "effects". The .fx files in "*WolfcamQL 10.3/scripts*" controls all ingame effects

***Script**:* A line of code

***Skin**:* A .jpeg or .tga image which covers a model (under the control of a script)

Recording Demos

While you play QL, your computer is constantly receiving data from the server about the position and status of every map object, known as an entity, and player. Your QL client then reconstructs this into a 3D game situation. It might be tempting to think that you are simply receiving an interactive video of sorts, but this is far from the case. You can record the game data into a 'demo' file (.dm_73), which can later be played back. A demo file is typically a few hundred or thousand kb large, depending on number of players, how much they spam and match time.

To record a demo, you can type in the QL console:

/record <demo name>

Alternatively, you can tell QL to automatically record a demo every time a game starts by using:

/cg_autoaction <0|1|2|3> (0 - do nothing; 1 – record demo; 2 – record end game screenshot; 3 – record screenshot and demo)

Once you have recorded some demos, you can locate them in the QL homepath. For those who don't know:

Vista: %appdata%\id Software\quakelive\home\baseq3\demos

Windows XP: %userprofile%\id Software\quakelive\home\baseq3\demos

GNU/Linux: ~/.quakelive/quakelive/home/baseq3/demos

Windows 7: C: \Users\<username>\AppData\LocalLow\id Software\QuakeLive\baseq3\demos

Mac: Users\Library\Application Support\QuakeLive\QuakeLive\baseq3\demos

WolfcamQL

Installing WolfcamQL

First off, download the latest version of WolfcamQL (at time of writing 10.3) from:

http://wolfcamql.fr/en/download/category/wolfcam or;

http://www.esreality.com/post/2447913/wolfcam-10-0/

Copy this zipped file into wherever you want it to be and unzip it. First thing to do is add the .pk3 files from QL:

Go to your QL homepath (as mentioned in **Basics**) and copy all the .pk3 files from "*/baseq3*" to your wolfcam homepath: "*/WolfcamQL 10.3/baseq3*"

You are now ready to run Wolfcam. You can do this just be double clicking "*WolfcamQL 10.3/ wolfcamql.exe*" but it's highly recommended that you do this with a .batch file instead, as it allows you to force Wolfcam to save files where you want it to (if you just run the .exe files will save to *"%appdata%/wolfcamQL/*") and start up with some cvars 'forced'.

I would recommend that you now download a text editing software that is better than notepad, so when you're writing configs, fx files, batch files etc paragraphs are properly structured, brackets are coloured and you generally have more options. I've been using the free "notepad++" for a few years.

Open up your software and type the following:

@echo off

start "wolfcamql" "<file structure>\WolfcamQL 10.3\wolfcamql" +set fs_homepath "<file structure>\WolfcamQL 10.3\" +set r_fullscreen "1" +set r_mode "-1"

Which translates roughly to:

Don't display the following text on screen

Start "a file with this name" "which is found here" save all files "in here" make wolfcam fullscreen and allow me to customize the window size

When you're done with that, save the file as "run_wolfcamql" as a .batch and put it somewhere useful. Double click it and a black box will pop up and then disappear quickly (@echo off), and then wolfcam will open. If you put all the correct .pk3 files into "WolfcamQL 10.3/baseq3" you should see the wolfcam screen (similar to Q3); if not a polite message will inform you so.

The screen may be looking pretty weird at the moment, but that's fine as we haven't explicitly told wolfcam how to display yet.

Creating a wolfcam config

This is really one of the largest parts of QL movie making and it's a continuous process. There is no 'perfect' config, and movie makers tend to guard their own very closely. That said, there are a number of sample configs on the internet which will work just fine. I recommend you download the following, as it includes a convenient explanation for many cvars alongside.

http://wolfcamql.fr/tutos/KittenIgnition_0.9.cfg

If you go to "*WolfcamQL 10.3/wolfcam-ql/*", somewhere in that mess of loose files you will see "q3config.cfg". This is where all the fun happens. Drag it into notepad++ and you will see a vertical list of roughly 1000 cvars, many of which make very little sense to you. You should be able to go through the ***/bind X "do this"*** area and make a few alterations to suit yourself.

When looking through cvars, unless there are many different options “1” will always mean “yes” and “0” “no”. You can either edit the cvars in notepad++, or type them into the console; changes will not be permanent until you resave *q3config.cfg.*

Useful cvars

I’m not going to go through everything, but a few useful cvars are below. I strongly recommend you visit http://wolfcamql.fr/en/cvars-list as not only does it list all the cvars, but it mostly provides a correct description of that cvar.

/r_visibleWindowHeight "720" -- Keep the same as ***r_customheight ""***

/r_visibleWindowWidth "1280" -- Keep the same as ***r_customwidth ""***

/r_customwidth "1280" -- The width your wolfcam screen will appear

/r_customheight "720" -- The height wolfcam will appear

/r_mode "-1" -- Allows the use of r_customwidth and r_customheight (we forced this with the .batch anyway)

/r_fullscreen "<0|1>" -- We’ve already forced this to “1” in the .batch file

/cvarsearch <keyword> -- Incredibly useful. If you can remember part of a cvar, but not all of it; search a keyword and the results are shown in the console.

/cg_drawstatus “<0|1>” -- 0=no weapon bar, ammo, timer etc

/cg_draw2d “<0|1|2>” -- 0=display nothing of the HUD, 1=display all HUD, 2=display some HUD

/bind <key> “freecam” -- allows you to switch between freecam and pov mode

/bind <key> “pause” -- press once = pause, press again = unpause

/cg_drawfragmessagetokens <frag message> -- Customise the frag message. You can use some tokens which display the relevant details in game: %v=victim name, %k=killer, %i = weapon icon, %a = weapon accuracy. Full list can be found in “*WolfcamQL 10.3/README-wolfcam.txt*”

/timescale “<0-100>” -- speed the game happens at. 1=normal, 0=paused, 2=2x faster than normal.

/fastforward “<time in seconds>” -- can be a bit buggy sometimes. Stick to 60 or 120 max.

/rewind “<time in seconds>” -- see above

/bind <key> “+scores” -- Press and hold for the scores

/bind <key> “+acc” -- Press and hold for weapon accuracy

/bind <key> record -- Records a new demo without altering the current one

/bind <key> stoprecord -- Stops recording the new demo

/bind <key> video avi -- starts recording an avi video. See next two cvars

/cl_aviframerate -- The framerate of recorded avi video (use between 30 and 60 for normal speed, use something like 200 for ¼ timescale)

/bind <key> stopvideo -- Stops recording the avi video

/fragforward -- Skips through the demo to frags

/cg_killbeep "0" -- Remove that annoying chime when you frag someone

/players -- Displays a list of players and their numbers in the console

/follow < number> -- Allows you to follow other players. Note that if the player moves too far from the pov (demo taker) it will switch back to pov as the demo doesn't record data very far from the pov.

/cg_ fov "" -- the field of view in degrees

/cg_ zoomfov "" -- the field of view in degrees while zoomed

/bind <key> "+zoom"

/cg_gunx "" -- align weapons in the x (left/right), y (up/down)and z (back and forwards) axis

/cg_guny""

/cg_gunz""

/cg_freecamcrosshair "<0|1>" -- Show/hide crosshair while freecamming

/cg_freecam_noclip "2" -- You can pass through walls while in freecam mode

/cg_freecam_speed "" -- Speed you move while in freecam mode

Some cool and often pointless effects are below:

/r_showtris "<0|1>" -- "1" draws the vertices of the map and all sprites in white

/cg_thirdperson "<0|1>" -- "1" follows the player from just behind their head

/cg_shadows "<0|1>" -- 1 = player shadows on, 0 = player shadows off

/cg_drawentnumbers "<0|1>" -- draws the numbers of every entity

/view <entity number> -- Locks the camera onto that entity. /view -1 undoes this.

/r_drawentities "<0|1|2>" -- Show/hide all entities for some reason

Advanced Customisation

Customising Textures and Models

This section is copied near-directly from a tutorial I wrote on esr a while ago. Feel free to refer back to it: http://www.esreality.com/post/2465741/all-ql-textures-compilation-for-wolfcam/

A while ago I compiled all (or pretty damn near) of the files found inside the .pk3 files. The result is an incredibly useful tool for movie makers, or indeed anyone who wants to modify the appearance of maps, players and objects while watching back demos.

How it works:
When wolfcam runs, it constantly calls upon the files that "skin" maps, weapons, ammo etc. These files are mainly located in the pakXX.pk3 files, but some are strewn around in random map files (hence why my compilation may not be quite complete). By de-crypting the pk3 files and copying the relevant contents into "*wolfcamQL 10.3/wolfcam-ql/*", wolfcam uses those files (instead of the ones in "*wolfcamQL 10.3/baseq3*"), meaning you have easy access to the files that decorate QL's interior!

1. Download the .zip file:

http://auri.co.uk/files/quake/skins/misc/eq_a...xtures.zip
http://q3.atdan.net/pub/misc/eq_all-default-quake-textures.zip

(cheers to auri for hosting and uZu for mirroring my file)

2. Unzip. You will notice loose in the initial structure a .jpeg called "howto" (Clickme). Folders with a red dot are the ones that you have just downloaded, and this is where you need to put them! As you can see I'm using wolfwhisperer, but this doesn't change anything. If you already have some of these folders, you can just merge them.

3. Now if you see a texture in wolfcam that you dislike or want to brighten etc you can do something about it! You need to locate the relevant file (try searching keywords, and most map textures are in textures/[base_wall|base_floor|gothic_block]). Worst case senario typing /shaderlist will output a list of all the shaders used in a map. Not fun.

Useful locations:
"*textures/ad_content*" -- Easiest way to override ingame ads
"*icons*" and "*menu/icons*" -- you might want to edit the weapon and item icons
"*menu/medals*" -- can change impressive, excellent etc
"*models/ammo*" -- edit grenade and rocket, for example
"*models/weaphits*" -- get really good and edit the rocket explosion. You'll want to read about fx scripting first
"*models/weapons2*" -- make a skin for weapons

If you want to actually modify weapon models, then you will want to download a software such as blender that allows you to edit skins by projecting the skin onto the model (.md3 file). Blender does not import .md3 by default, but it is possible to modify it to do so.

Custom Player Models and Skins

You can quite easily change models of enemy, team and pov (first cvar given for each), but if you want to use a custom skin it's a bit more tricky.

Download from http://ws.q3df.org/models/?model=&page=0 and place in "*WolfcamQL 10.3/baseq3*"

Use ***cg_forcepovmodel "1"***

Now you have a fair few cvars to change...

Team

/cg_teammodel <relevant model for skin>

/cg_teamtorsoskin <skin name>

/cg_teamlegsskin <skin name>

/cg_teamheadskin <skin name>

Enemy

/cg_enemymodel <relevant model for skin>

/cg_enemyheadskin <skin name>

/cg_enemytorsoskin <skin name>

/cg_enemylegsskin <skin name>

POV player

/model <relevant model for skin/skin name>

Q3 Style Bleeding and Gibs

You can do this the proper way (if you have Q3 or can be bothered to download pak0.pk3), or find it somewhere else (http://wolfcamql.fr/en/download/category/scripts-modifications). Some downloads will include all you need, so you can just put it into "*WolfcamQL 10.3/baseq3*" and run wolfcam, but I'll go through each step.

Copy the following from pak0.pk3 of Q3 to "*WolfcamQL 10.3/wolfcam-ql/gfx/wc*".

models/weaphits/blood201.tga

models/weaphits/blood202.tga

models/weaphits/blood203.tga

models/weaphits/blood204.tga

models/weaphits/blood205.tga

Also, copy these to "*wolfcamql/wolfcam-ql/damageq3*".

gfx/damage/blood_spurt.tga

gfx/damage/blood_stain.tga

Not to forget these to "*WolfcamQL 10.3/wolfcam-ql/models/gibsq3/"*

models/gibs/gibs.jpg

models/gibs/leg.md3

models/gibs/skull.md3

models/gibs/abdomen.md3

models/gibs/chest.md3

models/gibs/intestine.md3

models/gibs/brain.md3

models/gibs/foot.md3

models/gibs/forearm.md3

models/gibs/arm.md3

models/gibs/fist.md3

And finally copy these to "*WolfcamQL 10.3/wolfcam-ql/sound/player/*"

sound/player/gibimp1.wav

sound/player/gibimp2.wav

sound/player/gibimp3.wav

sound/player/gibsplt1.wav

You can now activate Q3 style gibs with ***/cl_useq3gibs "1",*** because the script necessary is already there; "*wolfcamql/wolfcam-ql/shaderoverride/wcq3blood.shader*".

However for Q3 bleeding you need to do a bit more. Paste the following into notepad++ and save it as:

"*wolfcamql/wolfcam-ql/shaderoverride/q3bleed.shader*"; leave "Save As Type" as .txt

```
bloodExplosion
{
    cull disable
    {
```

animmap 5 gfx/wc/blood201.tga gfx/wc/blood202.tga gfx/wc/blood203.tga gfx/wc/blood204.tga gfx/wc/blood205.tga

blendFunc blend

}

}

Then set ***/com_blood "1"***. All of this information can be found in "*WolfcamQL 10.3/README-wolfcam.txt*"

Fx Scripting

This is perhaps the hardest thing to write a tutorial for, and it's easily the most complicated. Few people spend the time necessary to learn how to write their own Fx scripts, and there are many to download. I'll attempt to patch together information I've found and learnt over the past few years on this topic, so I'm not taking much credit here.

FX scripting works by taking an input of certain parameters and processing them in a way described by the scripter to produce an output effect.

The FX system is made up on float, vector and string variables.

Float variables:

Floating point variables can store a single number. <floatX> can be a mathematical expression using any valid arithmetic, Boolean or comparative operator.

Size <floatX> The number of game units diameter an fx object is

Width <floatX> The number of game units wide an object (e.g. spark) is

Angle <floatX> Draw a shader every X degrees around dir (e.g. angle 45 draws 4)

t0 <floatX> You can save floats to these temporary values for complex calculations. Also t1, t2, t3 etc

red <floatX> The first float component of the vector color. Also blue and green.

origin0 A component of a vector. Also v"elocity2", for example. See **Vectors**

rotate <floatX> A sprite or spark is rotated by <floatX> degrees. Often 360*rand

rand A random value between 0 and 1. (read only variable)

crand A random value between -1 and 1. (read only variable)

loop Goes from 0 to 1 during any kind of repeating action in a script (read only variable). See **repeat**

loopcount The same as loop, but goes from 0 to <floatX> in the example of **repeat**. (read only variable)

lerp Moves linearly from 0 to 1 during the life of an emitter (read only variable)

pi A read-only mathematical constant: 3.14

time The time in seconds since the beginning of a map (read only variable)

life Amount of life in seconds that an entity got when it was emitted. (read only variable)

alpha <0-1>: The opacity of a fx object; "alpha 1" is fully visible, "alpha 0" is invisible

Vectors:

3 floats grouped together as an [x,y,z] vector, under one name.

You can also access a specific float from the vector by adding 0, 1, 3 to the vector name, for example **origin0** refers to the x float of the origin vector. Also **red**. **blue** and **green** are the float components of the vector **color**.

origin The position in the map a script event happened at

velocity The velocity in u (game units/second) of a projectile.

dir A normal (magnitude 1) vector in the direction of an event, such as an impact. There is a bug with lg so that dir is in the direction of the beam, but normally dir is perpendicular to the impact surface, regardless of impact angle.

angles Don't really have a clue

parentvelocity velocity of the script's "parent", normally a projectile

parentangles

parentorigin origin of the script's parent

parentdir direction of the script's parent

color The colour of a sprite is technically a vector. Uses RGB e.g. "color 1 0 0" is red.

You can also use **v0**, **v1**, **v2**, **v3** to store temporary vectors, in a similar way to **t0** etc.

String Variables

A string is any argument comprised of characters, for example a word.

model <model name> The .md3 model a block is using

shader <shader name> Loads the named shader from Wolfcam folder for use in a block

pop <variable> No idea. <variable> e.g. color

pushparent <variable> Something like 'use this variable from parent'. Not sure.

Emitter Specific Float Variables

All of these values are floats, but I felt it important to differentiate from other float variables as these can only be used within an emitter <block>.

alphafade <0-1> The fraction of "life" at which alpha starts to fade; 1=fade immediately, 0=never fade**.** If life is 5 seconds and alphafade is 0.2, alpha starts to fade after 4 seconds.

colorfade <0-1>: The fraction of "1- life" at which color is faded; 1 = fade immediately, 0=never fade

movegravity <floatGravity>: The amount of gravitational acceleration a particle feels e.g. "movegravity 98". Note that <floatGravity> can also be negative, making the particle move upwards

movebounce <floatGgravity> <float Coefficient of restitution>: Same as **movegravity**, but also with another parameter. Coefficient of restitution is the speed of rebound divided by speed of impact. Basically how much a particle bounces back from a surface. Can be 0-1

impactdeath <floatGravity> Same as movegravity but the particle dies upon impact

sink <floatDelay> <floatDepth> Make the particle sink into a surface upon impact

Block Statements

All of the following statements are used to define when or how a block will be run. Each block has to be preceded by one of these.

repeat <floatX> <block> <block> is run <floatX> times. The variable loop goes from 0 to 1 in <floatX> steps, i.e. the first time <block> is run, loop = 1/<floatX>, the second time 2/<floatX>etc.

distance <floatX> <block> <block> will be run every time the parent (e.g. rocket projectile) has moved <floatX> game units.

interval <floatX> <block> Since wolfcam 8.5 distance was introduced. Interval is essentially useless; <block> is run every <floatX> seconds, but is timescale and fps dependent.

emitter <floatX > <block> Particles created in <block> live for <floatX> seconds. lerp runs from 0 to 1 during the life of the entity. The variable life contains the age of the entity in seconds. Apparantly entity is an alias for emitter

if <testX> <block> [else <testX> <block>] [elif <testX> <block>] Each <block> is executed only if <testX> is true. <testX> e.g. cg_oldrail 1 or time > 300

once <block> <block> will only ever be run once, regardless of how many times it is passed

shaderlist <floatX> <block> A way to choose from a list of shaders. <floatX> chooses which shader is chosen from the list; 1 refers to the final shader and 0 the first, etc. It is often useful to make <floatX> rand.

colorlist <floatX> <block>A list of colours in <block>, chosen by <floatX>. See **shaderlist**.

modelList <floatX> <block> Same as shaderlist, but for models.

colorblend <floatX> <block> <block> is a list of colors. Unlike colorlist, colorblend interpolates the two colours adjacent to <floatX>

impact <floatSpeed> <block> <floatspeed> is squared, other than that I don't know

death <block> No idea

Float math commands:

Some float variables can be used like an algebraic character. These can even be used in math that includes themselves, as the script is read sequentially. Examples:

size 5*size -> Size is now 5*original size

angle 360*rand + size -> angle is now a random value * 360 + size

rotate 360*rand -> Sprite is rotated randomly within a full circle of 360 degrees

Vector Math Commands:

Most math commands take several inputs and a final storage vector name. They are sometimes also followed by a math value. Src stands for source, dst stands for destination, i.e. the vector you're starting with is <src>, the vector you're saving the answer to is <dst>. Math is any float expression.

scale <src> <dst> <math>

*<dst> = <src> * <math>*

add <src1> <src2> <dst>

<dst> = <src1> + <src2>

addScale <src1> <src2> <dst> <math>

*<dst> = <src1> + <src2> * <math>*

sub <src1> <src2> <dst>

<dst> = <src1> - <src2>

subScale <src1> <src2> <dst> <math>

*<dst> = src1> - <src2> * <math>*

copy <src> <dst>

<dst> = <src>

clear <dst>

<dst> = [0,0,0]

wobble <src> <dst> <math>

Vary <src> within a cone of <math> degrees and save it as <dst>

random <dst>

Create a vector with magnitude (length) 1 in a random direction

normalize <src> [dst]

Normalize a vector (make its magnitude = 1). If [dst] isn't specified, it will be saved to <src>

perpendicular <src> [dst]

Create a perpendicular vector to <src> and save it as [dst]. If [dst] isn't specified, it will be saved to <src>

cross <src1> <src2> <dst>

*Create the cross product of <src1> * <src2> and save it as <dst>*

rotatearound <src1> <src2> <dst> <math>

Rotates the point <src1> around the vector <src1> by <math> degrees and saves as the vector <dst>

Inverse <src> [dst]

Make <src> point in the opposite direction and save it as [dst]. If [dst] isn't specified, it will be saved to <src>

Render Commands:

These will render something on the screen using the current state of the variables it needs. This means that if you render a sprite and haven't defined color, the renderer will use the last value of color in the script.

sprite [options] An image that is always aligned to the screen. Uses origin, rotate, shader, color, size, angle. [options] can be "cullNear", which I've heard draws only the first sprite the screen sees, which prevents unnecessary rendering and alpha accumulating, although I'm not 100% sure

spark [options] An oval spark aligned to the screen. Uses origin, velocity, shader, color, size (long side), width (thin side).

quad A fixed aligned sprite. Uses origin, dir, shader, color, size, angle, rotate

beam A screen-aligned image stretched between 2 points. Uses origin, shader, color, size

decal <floatLife> [options] A flat image projected onto a map surface, centred at origin with dir as its normal vector. Lives for <floatLife> milliseconds, which if not included is default 10,000ms (10 seconds) Uses origin, shader, dir, color, rotate, size. [options] can be "temp"; fade the alpha instead of rgb, "alpha"; fade rgb in beginning followed by an alpha fade, or "energy" which makes the decal last only one frame.

light Adds a sphere of light which shows up on surfaces. Centred at origin, of radius size. Uses origin, size, color

sound <soundname> Plays the sound soundname. Include folder path e.g. "sound sound/weapons/lightning/lg_hit.wav"uses origin

loopSound <soundname> Plays the sound soundname in an endless loop. Uses origin

soundList <block> Plays a sound that is randomly picked from the list of sounds in <block>. <block> contains paths to soundfiles. Uses origin.

vibrate <floatStrength> The amount the screen vibrates with the script is passed. Uses origin.

rings Renders a set of rings every width (or size, not sure) units. Used in weapon/rail/trail sometimes. Uses origin, dir, size, width, shader, color

anglesModel No idea. Uses origin, angles, model, size

axisModel Draws a model aligned to the map axis (I think). Uses origin, axis, model, size

dirModel Draws a model aligned to dir. Uses origin, dir, model, rotate, size

Some other [options] are: [firstPerson] [thirdPerson] [shadow] [cullnear] [cullRadius] [depthHack] [stencil], although I don't know how or when to use these.

Float Math Operators:

+ add

- subtract

/ divide

***** multiply

Math Test Operators:

Special group of operators that return 0 if a test is incorrect or 1 if the test is correct. Useful with the "if" block statement.

<floatX> < <floatY> Tests if <floatX> is less than <floatY>

<floatX> > <floatY> Tests if <floatX> is more than <floatY>

<floatX> ! <floatY> Tests if <floatX> is unequal to <floatY>

<floatX> = <floatY> Tests if <floatX> is equal to <floatY>

& example: (2 & 2) returns 1 as 2 && 2

| example: (2 | 0) returns 1 as 2 || 0

Note: I've never seen the & or | notation before, and know little Boolean. CaNaBiS (Q3mme developer, upon which the WolfcamQL fx system is based) mentions that "The Boolean operations only check for not equal to zero, no Boolean algebra".

Float Math Functions:

These functions can be repeated within themselves, for example sin(sin(270)) is valid. All trigonometric functions use degrees, whereas in Q3mme they used radians.

sqrt(<floatX>) Returns the square root of <floatX>

ceil(<floatX>) Rounds <floatX> up to next integer (whole number)

floor(<floatX>) Rounds <floatX> down to next integer (whole number)

sin(<floatX>) The sine of <floatX>

asin(<floatX>) Short for "arcsin", which is the inverse of the sine function.

cos(<floatX>) The cosine of <floatX>

acos(<floatX>) The inverse cosine of <floatX>

tan(<floatX>) The tangent of <floatX>

Atan(<floatX>) The inverse tangent of <floatX>

wave(<floatX>) Just like a sine wave, but one wavelength is 1, rather than 360.

clip(<floatX>) Returns <floatX> clipped to either 0 or 1. < 0 becomes 0 and > 1 becomes 1

Math Cvars

This is a special group, if the fx math parser can't find the variable name it'll try to see if there's a regular q3 cvar with that name and then it'll use that value, for example:

size r_railWidth*0.5 **or** if cg_oldrail 1<block>

Getting Started with FX Scripting

All of the above information is very complicated and will make little sense until you start to look at scripts, making changes and checking wolfcam for the effect. First off, you can change the fx file that is run by WolfcamQL using ***cg_fxfile "scripts/weapon.fx"***. Call your script file weapon.fx and save it in scripts. I would recommend starting with q3mme.fx, which you will see in the scripts folder. Run wolfcam and already you have some cool effects! By looking through q3mme.fx and checking through the above scripting guide you can try to make sense of what each block means. Below is an example rocket trail script.

Anything 'commented' in // is not read by the renderer.

weapon/rocket/trail { **//This is the script event.**

//orange flame trail

color 1 0.1 0 **//Set colour to 1 part red, 0.1 blue and 0 green (orange)**

alpha 0.85 **// is opacity 85%**

shader flareshader **//Use the shader called "flareshader"**

distance 1 { **//Repeat the following stuff every time the projectile moves 1 game unit.**

normalize dir v0 **//Take the vector dir and make it length 1 (it already is, *duh*), and save it as v0**

inverse v0 **// Make v0 in the opposite direction**

wobble v0 velocity 5 + rand*5 **// Wobble the direction of v0 randomly within a cone of 5+5*rand degrees and save it as velocity.**

emitter 0.03 + rand*0.05 { **//Render the particles, and give them a life of 0.03+rand*0.05 seconds**

size 5.5-lerp*5.5**// Start at 5.5 units big and gradually reduce to 0 after life seconds**

colorFade 0.3 **//Start to fade the RGB after 0.7*life seconds**

Sprite **// Render it as a sprite**

}

}

//smoke trail **//Leave yourself some notes. This is still part of the same fx event**

color 1 1 1 **//Make its colour black**

alpha 0.33 **// make it 33% opaque (66% transparent)**

shaderlist rand { **//Randomly select a shader from these two**

smokepuff

hastesmokepuff **//got a feeling they're both the same though**

}

rotate 360*rand **//Rotate the final thing randomly in a whole circle**

distance 5 { **//Repeat the following stuff every 5 game units**

repeat 3 { **//repeat the following stuff 3 times every time this is passed**

scale parentvelocity velocity 0 **//Give each smoke puff no velocity**

emitter 0.2 { **// Give the particles a life of 0.2 seconds**

alphafade 0.5 **// Start to fade opacity after half of life**

size 5+lerp*25-loop*10 **// Make the size start from 5-loop*10 (as we're repeating 3 times loop should equal 0.3 the first time, 0.6 the second time and 1 the third time) and then increase its size by lerp*25 as time goes by.**

movegravity -50 **// Make it move upwards due to antigravity**

```
            sprite cullnear // Draw it as a sprite
            }
        }
    } //Don't forget to close this last bracket from "weapon/rocket/trail {"
```

You need to make sure that you close as many brackets {} as you open or else the next script won't work. Remember that there is a correct place for each variable, for example size *could* go anywhere but as we're using lerp it has to be inside the emitter (and as we're using loop it has to be inside the repeater). Until you've given the particle life, the variable lerp makes no sense. Also the command, in this case *sprite cullnear*, has to be at the end of the script.

Using/ Making Shaders

Instead of using the same old shaders that quake includes (flareshader, smokepuff, burn_med_mrk etc) for everything, you can make your own with a little photoshop work and a .shader file.

I'll run you through making a custom lightning impact shader that I made.

First, all shaders have to be square unless you want them to get messed up when you use them. Create a square file (I recommend 64x64 px but the larger the better, if your computer can handle it) and get drawing your sprite. Remember that your *color* will only get placed in white areas, and none in black ones. Save it as a .tga file. My completed "lgspark.tga" looks like this and I put it in "*gfx/misc/lgspark.tga*" (see section on **Customising Textures and Models**).

Now you need to write a script to let Wolfcam know what you want it to do when you define *shader lgspark* in the script event *weapon/lightning/impact{*. You can adapt this template for your own use:

```
// lgspark.shader (just to remind me, I'm very forgetful)

Lgspark //here we're telling wolfcam what the name of the shader is.
{
    Nopicmip //we don't want any picmip, thanks
    cull none //draw it no matter where it is, or where it's facing
    {
        map gfx/misc/lgspark.tga //make sure this is the correct location
        blendFunc GL_ONE GL_ONE//some openGL stuff
        rgbGen vertex
```

```
        }
}
```

Save this as lgspark.shader (keep "save as type" .txt) and stick it in shaderoverride/. Now we can write an lg impact script including the shader called "lgspark". Here's mine for example:

```
shader lgspark //use our new shader
alpha 1 //make it full opaque
color 1 0.4 0.1 //make it orange
    repeat 3{ //every impact, repeat the following code 3 times
        scale velocity velocity 0 //make the sprite stay still
        emitter 0.2{ //give the sprite 0.2 seconds of life
            size 5-lerp*4.5 //start with a size of 5 units and decrease to 0.5 after life
            colorfade 1 //start to fade the colour straight away
            movegravity 0 //don't let gravity effect the sprite
            sprite //this is the command line
        }
    } //don't forget to close brackets!
```

Recording

The previous section makes this look easy. I've always used the simplest method for recording, although there are many more complicated (and arguably better quality) methods for recording, for example you can use wolfwhisperer which has a built in ffmpeg engine.

/Cl_aviframerate "" - I recommend using 60 for high quality, or 30 if you just want to put on youtube quickly (youtube is limited to 30fps). If you're using a lower timescale, you can keep the quality the same by increasing the framerate. For example timescale 1 ~ cl_aviframerate 50 is equivalent to timescale 0.25 ~ cl_aviframerate 200.

/bind <key> video avi -- Starts recording an avi video to "*WolfcamQL 10.3/wolfcam-ql/videos*". Size will be very large.

/bind <key> stopvideo -- Stops recording the avi. You'll need to re-render it in another program (e.g. Sony Vegas) before uploading it to youtube.