Quake 3 Arena

Info

Multiplayer Information

- Internet play: yes
- LAN play: yes
- Lobby search: yes
- Direct IP: yes
- Play via GameRanger: no
- Coop: yes
- Singleplayer campaign: yes
- Hotseat: no

The third reincarnation of Quake is one of the most revered first person shooters in gaming history. It was released on December 5th, 1999 by Activision. Its idTech 3 engine is also the basis for a lot of other games, for instance Star Trek - Voyager Eliteforce. It was the main competitor to Epic's Unreal Tournament title from November 1999. In 2005 idTech released the source files of this engine under the GPL, giving rise to even more new games, developed and maintained by the open source community.

Installation

The game still installs well even under Windows 10 (2019-03). The only drawback is the not startable `setup.exe` file, when running a 64 bit version of Windows. In this case simply use the file `Quake3\Setup.exe` from the CD. It works just fine.

In case you still have problems, you may always fall back to simply copying the required files. The only really required files are `quake3.exe` and `baseq3\pak0.pk3`. They can be found in the folder `Quake3`.

Graphics Hack

The game engine does offer a rather limited set of graphics resolutions. Especially the wide screen resolutions are only a few. This however, only goes for the presented resolutions in the game menu. Actually the engine does provide all available resolutions of your graphics card. The article Q3A Graphics Hack describes, how you can use them. These steps also work for most of the other Quake 3 Based Games.

Settings That Affect Gameplay Mechanics

Q3 is known to have »side effects« caused by certain settings. Here are some candidates that might improve your experience, or worsen it.

com_maxfps

This value limits your maximum frame rate, at which you computer updates the graphical representation of the game state (read: what you can see on your monitor). It is an integer value, that theoretically can take any number but in practice can only take values of (1000 / x), where x is an integer number greater than 1. So values such as 500 (x = 2) or 125 (x = 8) are OK. Values that do not hit these spots will be rounded up (e. g. for `com_maxfps` = 120 set the taken value will still be 125). Note: This is the *maximum* frame rate your client will allow itself to produce, even when your hardware might allow for more. On the other hand, if your hardware is not potent enough for this value, it might also produce lower actual frame rates.

Certain values of this cvar may result in higher jumps or lower jumps. The optimal value is 125. This video explains it in more detail: Quake 3's frame rate dependent physics.

rate

This value defines the bytes per second your client will allow to receive at max. It is send to the server and regarded as your upper limit. If the server itself has set a lower limit, it will us its own limit. When nothing is going on on the server (e. g. nobody moves or shoots) this rate must not necessarily be used up entirely. A very common value for this is 25,000. The higher the better for you, unless you reach your downstream capacity. If you set this too high for your downstream, you might experience higher pings, loss and in general a bad experience up to the point where you drop off from the server.

snaps

The server does not send you a continuous stream of changes as it processes them. It sends you snap shots of the current situation. This variable defines how often the server should send you an update on the current game state (kind of a frame rate but for the game state). The higher you set this, the more accurate will the representation of the current game flow be on your end. But at the same time it will consume more bandwidth. So if you reach the upper limit of your connection, you may again experience lagg, loss and drop outs. A very common value would be 40. However, just like with `rate`, the server caps this value, depending on its `sv_fps` value. You will never receive more snaps than the server allows.

cl_maxpackets

This defines how many packages per second you send to the server. Similar to the snaps, this is an update from you (the client) to the server. The higher you set this value, the more smooth your game experience will be. However, if you use a too high value for your upstream bandwidth, you will again

experience loss which might result in situations such as shooting (and from your end actually hitting the target) but not hitting the target anyways as the shot never reaches the server and therefore does not count at all. And of course, lagg, ping spikes and stuff like that will happen more frequently. This is a sensitive value. The max value is 125. For really low bandwidth connections such as dial up, 20-30 might be sensible.

Also note: Setting `cl_maxpackets` to a lower value than your `com_maxfps` value will reduce your effective `cl_maxpackets`. See section Value Compatibility on the matter.

cl_packetdup

This cvar controls how often you re-send *the same* packet (duplication). This is meant as a means against a lossy connections. If one of the duplicated packages is lost, it is covered by another one of the bunch. The downside of this is that of course it increases your bandwidth by the given factor. 0 means it is off. 1 means, a package is send 1 more time than necessary, 2 means two duplicates, and so on. So bandwidth strain is also increased to 2, 3 or even more times. Similar to a too high `cl_maxpackets` values, this may eventually cause higher pings and laggy results when you hit your bandwidth limit. So it is a tradeoff of bandwidth strain and loss occuring. For a healthy connection a value of 1, maybe 2 should suffice, if needed at all. On a connection with potent upstream you might be able to totally cut occuring loss to zero this way. But it might be more advisable to not use your laggy wireless connection or fix that faulty LAN cable. 

cl_timenudge

Any network game has the problem, that the actual game flow is chopped into snapshots or frames and that the rate of this chopping may vary for all involved computers. This quantization also means, that there is kind of a dead time in-between two snaps. But the game continues for everyone locally anyways. Hence, this dead time may be interpolated on the client (so that during those 3 frames on your 125 fps PC you get not stuck playing on a 40 fps server until the next frame updates from the server). This also means, the client has to do predictions on what will be going on in the meantime. (This goes especially for the server, too, as it is kind of the master of what the game states of the clients should be using as a reference.) Only when the next frame update from the server arrives, your game state will be »set straigt«. (You know this from laggy games, where someone moves through the map and all of a sudden does a big jump or stuff like that. That's when the interpolation is dropped and the updated frame takes effect.)

`cl_timenudge` defines in frames relative to the next expected frame from the server, when the client begins to extrapolate. You can set negative values, which will cause the prediction to happen even before the update from the server can kick in. In the example above, you having 125 fps, the server using 40 it means you will have to wait 3-4 client frames until the server will send you another update. So your window of premature interpolation would be -3 to -4 tops. -5 and lower will not make sense, as this will have the same effect as -4. -2 would be somewhere in the middle, meaning half the way your client will start interpolating. The values allowed are calculated as 1000/`sv_fps`. So `sv_fps` = 40 would mean -1 to -3. If a server would have `sv_fps` = 125, the value might be between -1 and -8.

But you can also use positive values, meaning the interpolation starts *after* the expected server update frame would happen. So in essence it would only kick in, if the server's update frame comes in

delayed (or not at all) somehow. When having loss or a high ping, that might make sense, too. It would ensure that under normal circumstances, the client waits at least until the server had a chance to update the client. This will give you the most accurate game state (no interpolation without loss or update delays) as the interpolation only kicks in, when something seems to be going a little bit sideways.

Important: This will by no means fix any ping problems or loss. It just tries to compensate for it.

r_swapinterval

If you set this to 1, the fps value is synced with the actual hardware refresh rate of your monitor. For many players this is 60, which will act exactly as if you set the `com_maxfps` value to that value yourself (with all side effects that may have). If you turn it off, you may be able to set higher values than can actually be effective on your hardware. In the worst case scenario your screen will turn black until you quit Quake 3.

cg_predict

This affects how your client predicts what will go on (see cl_timenudge). This may be 0, 1 or 2. 0 shuts it off, 1 is standard. 2 is an optimized variation, slightly more prone to faulty predictions but a lot faster. This should not be necessary on modern systems but on older machines it might give you an fps boost. Do not set `cg_predictItems` = 1 with `cg_predict` = 2.

cg_predictItems

If set to 1, items being taken by a player are also predicted. Should not be necessary on a good connection.

Value Compatibility

If you use `cl_maxpackets`, make sure to set compatible values. Internally, the `cl_maxpackets` value will be reduced to the next lowest value, according to your `max_fps` value: `cl_maxpackets` = `com_maxfps` / *x* with *x* being a positive integer number.

So for your generic 125 FPS, the allowed `cl_maxpackets` values are: 125 (1), 63 (2), 42 (3), 32 (4), 25 (125),... So if setting `cl_maxpackets` = 100, you are below 125, it takes the next lower value, which is 63. That's quite a big step down.

For a computer that cannot handle 125 but 100 `cl_maxpackets` steps would look like this: 100 (1), 50 (2), 34 (3), 25 (4),.... So when in doubt, set `cl_maxpackets` on the same or higher value as `com_maxfps`. Q3 will lower `cl_maxpackets` automatically anyways.

(Source)

Open Source Adaptions

As the source code of the Quake 3 engine was released by ID Software, picked up by the ioQuake3 team, there are a number of open source projects, that aim at giving the same game experience as the original:

- ioQuake3
- OpenArena
- Quake3e

Also derived from ioQuake3, there is a bunch of projects for games, that were implemented using the original Quake 3 engine or a derivative of it:

- Monkeys Of Doom
- Nexuiz
- q3osc
- Q3Rally
- Reaction
- Smokin Guns
- Star Trek: Voyager Elite Force
- Tremulous
- Turtle Arena
- World of Padman
- ZEQ2-Lite

Back to the games database