Multithreading the update and draw loops

Though about several olutions ... the one im most happy with at the moment is this

We do not mind if the draw loop lags slightly ... the update loop should always be faster than the draw loop ... if not the draw loop will simply redraw the same image (or even better ... do nothing)

Because of this the update loop will always operate on the central objects and the draw loop will operate upon a list of structures generated by the update loop.

These structures are everything necessary for the draw loop to draw. They need to contain a reference to the model. The model itself should be totally independant of the update as several objects may reference the same model. They also need to contain the models position and rotation as vector3 and quaterion as values.

The draw loop will contain a list of all drawable objects and have access to the camera object to draw through. It will perform the logic to determine which objects are in view (if the camera moves than all objects need to be tested again)

This list will get updates when the update loop adds to a queue of changed objects, new objects and objects to be destroyed.

Both of the threads get a reference to the queues passed to them on creation. The only thing that need to be ‘multi-threaded’ here is the queue ... but as one thread will add and the other remove there isn’t a problem here either.

Multithreading inside these threads can be considered – however i am cautious as context switching is a \*bad\* thing. If you exceed an optimum amount (for the program – load will vary with time) the performance plummets. With no way to precalculate this edge things are very difficult.

The main use for it will be File IO / networking etc anything we need asynchronously and is order independant.