

## Re: new here, my lang project...

**Source:** <http://coding.derkeiler.com/Archive/General/comp.object/2005-01/1167.html>

---

**From:** H. S. Lahman ([h.lahman\\_at\\_verizon.net](mailto:h.lahman_at_verizon.net))

**Date:** 01/24/05

Date: Mon, 24 Jan 2005 21:00:19 GMT

Responding to Cr88192...

>>> *ammusing is that the nature of these objects feels pulls even from other  
>>> engines, eg, to match their specific conventions.  
>>> what breaks there have been thus far (eg: calibrating space in meters vs.  
>>> carpentry or standard inches) have resulted in endless annoying troubles.  
>>  
>> The old mix & match trick. One can do the same thing with software  
>> construction paradigms with similar results -- which launched this thread  
>> as I recall. B-)  
>>  
>  
> yes, about.  
>  
> albeit, mix and match is more of a direct hassle when dealing with code, as  
> it is not always clear where and when to convert, and this has led to some  
> annoying kludging.*

It is more of an immediate, short-term hassle. But mix & match at the construction paradigm level can result in very subtle long-term problems. Pay now vs. pay later.

>  
> yes, the float/double issue is also a hassle, but at least it is clear where  
> I have to convert, so it is less so.  
> for some things at least I had considered a convention of going to "SI  
> numbers", or essentially '<number><scale><unit>', eg: I could write '3mm'  
> and have it understood as "3 millimeters" and have it autoconverted.  
> however, this does not fix the problem in general (units are not  
> known/retained in many situations).

Descriptors are a hassle to use, they tend to bloat both storage and executable statements, and they usually have substantial performance penalties. So it is often better to use a consistent units within a subsystem and do any conversions in the subsystem interfaces. Of course, if the code isn't conveniently modularized into subsystems, you are pretty much screwed. B-)

comp.object: Re: new here, my lang project...

>>>however, the monolithic objects aproach is sort of forced by the  
>>>predecessors I have inherited some formats, tools, ... from.  
>>>in some cases, I can do little really wrt fundamental issues.  
>>  
>>Right. That's the classic problem of legacy code. It is generally  
>>regarded as high risk to introduce OO development into a shop with a lot  
>>of legacy code. That's because the legacy code was not constructed with  
>>OO principles so there is no convenient way to partition  
>>responsibilities -- the legacy code has already usurped some of the  
>>responsibilities of the new objects one would like to have. So you either  
>>do a lot of refactoring of the legacy code or you do a kludge.  
>>  
>  
> well, more like legacy data and tools in some ways...  
> I don't control and I can't change the tools, I am stuck with what they  
> generate and understand, and doing something different would be too much  
> work.

What sorts of tools and what are they generating? (I had the impression you were manually coding C.)

>>>oh well, it has been quite a while since I have used (system level)  
>>>threads. on current systems in many cases threads don't have a strong  
>>>justification for the extra work needed to maintain them (eg: avoiding  
>>>race conditions, securing subsystems via locks or such, ...).  
>>  
>>I'm kind of surprised a game like quake didn't have threading. Smooth  
>>graphics usually requires higher priority and I would expect the AI to be  
>>doing something useful while waiting for the next player keystroke.  
>>  
>  
> nope, quake was single-threaded all the way.  
>  
> the main power is a big main loop, that cycles through and calls all the  
> related subsystems to cause them to do whatever, and the loop repeats.  
>  
> the main power of things is keeping the loop cycling fast (>30Hz is good,  
> >50 is better, ...). 10 or 20Hz, however, is not so good, the game lags and  
> things start getting jittery.

That only works for primitive graphics, a dumb AI, and an RTS format where one doesn't wait for the player. That approach basically assumes that if one divides the program up into small functions and executes them serially, everything will be fine because all the processing must be done on each cycle regardless of whether it is time sliced or not. Unfortunately that does not work well for more complicated games for a couple of reasons.

One is that the CPU cycles are not divided equally among activities. Probably 80% of the processor cycles go to graphics, even when downloading operations to a spiffy graphics card's processor. For

Re: new here, my lang project...

comp.object: Re: new here, my lang project...

smooth processing that should not be interrupted very often and then only for short periods. (The graphics card provides a buffer for refresh, but it needs data to eat that must be loaded fairly regularly in synch with the refresh; otherwise images start to overlap.)

The second reason lies in refresh mechanics for the graphics. That is a hard real-time constraint in the 50–80Hz area. If the graphics data for a loop can't be loaded within 2 periods the odds are good that the display will be choppy. Since graphics is eating most of the CPU cycles, that can be a problem if one takes a break to do something for the AI or the player.

If the AI needs to think or the player initiates some action, the resulting computations need to be interwoven with the graphics processing. IOW, the graphics needs the highest priority. Perhaps more important, there will be times when the player is viewing something that is not very active graphically (e.g., a small scale map or some interactive dialogs). Then you need the AI or game mechanics to be making up for lost time by doing a lot of computation while they have a good shot at the CPU. One can manage that manually, but threads are a lot easier

The third problem lies in differing time scales. Player actions and disk I/O are at the millisecond level while everything else is at the nanosecond level. It just doesn't make sense for them to get equal priority with the graphics on each processing loop. Also, you can queue up player events in the UI and poll them from the loop in the game logic, but you have to wait for disk I/O. If everyone is held up waiting milliseconds for I/O in the single-threaded loop that is a huge waste of resources that could be spent on something like a better AI or spiffier graphics. (Don't you just hate the older games where the game just stops for 5 seconds for an autosave while you are banging on a key to get you token out of harm's way?)

> *I don't really get it, however:*  
> *my inheritance model is also outside the app code.*  
>  
> *it is basically just files and directories in the resources directory which*  
> *define "classes", which serve to specify the search path for resources,*  
> *specify scoping for the scripts, ...*  
>  
> *all this is dynamic, eg, no compiled code involved, and even fairly minor*  
> *impact on the scripting. "classes as an abstraction for data" really. the*  
> *masses of game resources are becoming large and difficult to manage or*  
> *navigate, possibly eventually serving as an impediment to modders or such...*

OK, then I am thoroughly confused about what you were doing. B-) I thought you were updating a Quake engine that was written in C.

>>>>*Unfortunately this is getting into rendering issues that are well beyond*  
>>>>*my ken. I haven't paid any attention to graphics techniques for a*

Re: new here, my lang project...

comp.object: Re: new here, my lang project...

>>>>decade, which makes me pretty obsolete in that venue.  
>>>>  
>>>  
>>>yes.  
>>>  
>>>sprites have gone away. statically deformed meshes have risen and gone  
>>>away. static animations are fading, and may be replaced eventually. ...  
>>>  
>>>always new systems with new data, and old tools that can no longer  
>>>generate new data or have arbitrary limitations. it is sort of like a  
>>>treadmill on this front.  
>>>there also seems to be an ever-increasing amount of ties between the  
>>>subsystems, eg, physics, scripts, and rendering appear to be getting  
>>>integrated at ever finer levels in newer games, ...  
>>  
>>That can be good or bad, depending on the context. It is bad if the ties  
>>represent bleeding cohesion and dependencies between the various subject  
>>matters. OTOH, I think I mentioned that good decoupling interfaces  
>>usually have more, finer-grained message traffic.  
>>  
>>IOW, the quality is more about why the messages are sent and how they are  
>>interpreted than it is about the volume of traffic.  
>>  
>  
> dunno. I am not really that sure how to split them up, even with lots of  
> messages.  
>  
> looser parts are easier, just probably need registration-based interfaces or  
> message passing or such.  
>  
> the main problem is a sort of combinatorial problem, namely, it is unclear  
> how any of the stuff can be done in isolation or reasonably broken up into  
> message traffic.

Unfortunately I can't offer much advice about how to split up the subject matters with something more specific. [Especially, when I am beginning to have doubts that I know what you are doing at all. B-)] Could you give an example of a combinatorial problem? (Not a lot of code; just a description of the activities, dependencies, and messages for some operation on a single entity.) For example,...

>  
> I guess ways are found by people to manage this and maintain abstraction,  
> just like physics can be largely simplified to discreet  
> timeslices/integration, the use of a set of handlers for every combination  
> of 2 of each type, and a number of passes for each kind of process:  
> setup (preparing the entity for movement);  
> prediction;

Of what? The next waypoint? The best overall path?

Re: new here, my lang project...

comp.object: Re: new here, my lang project...

> *detection/compensation (possible multiple passes);*

Detection of what? Other entities? Boundaries? Treasure?

Compensation for what? Obstacles? Windage when aiming? Pay raise?

> *verification/adjustment (if a movement can't be worked out by here it is just stopped);*

So verification is just checking if the target location has been reached?

What adjustment?

> *finalization (finishing the move and stabilizing the entity state).*

Sounds reasonable, but it is short on detail. What's involved in 'finishing the move'? What's involved in 'stabilizing'?

>

> *detection is a biggie here. one has to deal with the various pairings of model/solid types.*

What models? What solids? Since they are being compared I assume they represent views of the same thing, but what is it? (This seems to be using a particular technique to detect obstacles, but the details of that technique are no clear, such as the nature of the model.)

> *compensation is a little easier, since one can take shortcuts.*

>

> *this is a hassle not that easily explained, and it requires multiple attempts to get a really usable physics engine.*

What do you mean by 'physics engine'? Can you provide a couple of sentences to describe what you think the subject matter and responsibilities are? (I have my own vision, but that may be the communication problem here.) Some of my questions above are a tad on the facetious side, but the basic push-back isn't. I don't have a good picture of what /you/ think is going on here and that stands in the way of speculating on how to partition things.

As an example of where my preconceptions may be getting in the way, in this of list of process steps the only one that I might regard as physics is the detection of obstacles. Even that I would expect to be owned by a Map or Terrain subsystem rather than by Physics.

> *now, if one looks at it enough, it might also become apparent why things like ik/"ragdoll" are scary-looking. they have both combinatorial and integration related problems. not only that, but the renderer gets involved, the current state of a model/entity (positions, bounds, ...) are needed both by physics and rendering, and don't go well with the typical abstractions (eg: frames, state variables, ...). actually, it could probably be said*

Re: new here, my lang project...

comp.object: Re: new here, my lang project...

> *physics is doing the heavy lifting and rendering is just along for the ride.*

Let the renderer have its own view of the entity. Some aspects of that view, like position, will change as a result of game logic changes while others, like bounds, are likely to be quite stable. Use messages to synchronize things like position data with activities elsewhere.

Let's say the Physics subsystem does detect obstacles to movement. The presence of an obstacle in the path would be important to whoever is managing the entity's movement, but I don't see why Rendering would care. Rendering displays the obstacle but it isn't involved in movement. It just renders the result of moving the entity to a new <probably incremental on a path> position /after/ the path is adjusted around the obstacle.

While I am sure that once the next <incremental> entity move is determined, several subsystems (e.g., Rendering, Game Logic, AI, etc.) might have a need to know about that. So multiple announcements are generated when the move is determined. But whoever is managing the movement probably only needs information from the owner of the map information to select the optimal path and move along it incrementally.

\*\*\*\*\*

There is nothing wrong with me that could not be cured by a capful of Drano.

H. S. Lahman

hsl@pathfindermda.com

Pathfinder Solutions -- Put MDA to Work

<http://www.pathfindermda.com>

blog (under constr): <http://pathfinderpeople.blogs.com/hslahman>

(888)-OOA-PATH