

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

**Source:** <http://coding.derkeiler.com/Archive/General/comp.object/2005-01/1261.html>

---

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

**Date:** 01/29/05

Date: Sat, 29 Jan 2005 20:28:20 GMT

Responding to Cr88192...

>>>>> *What sorts of tools and what are they generating? (I had the  
>>>>> impression you were manually coding C.)*  
>>>>>  
>>>>>  
>>>>> *well, primarily modellers (milkshape, ac3d, ...) and programs like quark  
>>>>> (which do mapping). these tools were generally designed for quake-style  
>>>>> games (quake 1/2/3, half-life, ...).*  
>>>>  
>>>> *Never heard of them. What sort of models? [I assume you aren't doing  
>>>> UML models. B-)] I gather these are sort of home-grown modding tools of  
>>>> some sort?*  
>>>>  
>>>  
>>> *nope, these do 3d models specific to the game...*  
>>> *quark is fairly popular as a multi-game mapper.*  
>>  
>> *Are these pure graphics models (e.g., the equivalent of a GUI builder for  
>> a graphics pane)? Or are they for the physics of resolving movement,  
>> extent, etc.*  
>>  
>  
> *both.*

In the same model? Or are there separate models for display and movement physics?

> *the models consist of entities, which are bags of key/value pairs that can  
> also have geometry attached to them.*  
> *the geometry consists of "brushes", which are defined in terms of  
> intersecting sets of planes (that can also have textures applied to them and  
> such).*

Again, I see where all the pointer indirection comes from. B-)

> *the names, properties, keys, ... of the entity are generally constrained,  
> and variations lead to inconsistencies with the parent games and with the*

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

> editor. I try to keep these inconsistencies low.

So you need a tool like quark to ferret out inconsistencies?

>>>>>>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. <snip threading issues  
>>>>>>>discussion>  
>>>>>>  
>>>>>>>well, but all this is not how the games are done.  
>>>>>>  
>>>>>>>You meant, "... how these games are done", right? I can't imagine the  
>>>>>>>newer action/shooter games being single threaded.  
>>>>>>  
>>>>>>  
>>>>>>>dunno, but somehow I suspect they are.  
>>>>>>  
>>>>>>>really, on current pc's what real incentive is there to be  
>>>>>>>multithreaded?...  
>>>>>>>now, I can't really be certsin about games newer than quake2, but somehow  
>>>>>>>I doubt things have changed thst much since then.  
>>>>>>  
>>>>>>>The problem is that the OS and low level graphics services (e.g., DirectX)  
>>>>>>>are gobbling up an ever increasing fraction of the clock cycles. So when  
>>>>>>>you go from 500 MHz to 1 GHz you can actually get a reduction in  
>>>>>>>performance when the software is upgraded in lockstep. Note that most of  
>>>>>>>today's games would run in slow motion (if they could run at all) on a 200  
>>>>>>>MHz PC, which was state of the art not too long ago.  
>>>>>>  
>>>>>>  
>>>>>>>nope, not slow motion, rather the game will continue in realtime but only  
>>>>>>>redraw once in a great while and be quite difficult to control.

That's putting it mildly. B-)

> one can run quake2 on a 486DX and observe this effect. most newer games are  
> hw-accel only, and thus can't be run on anything without hw accell.

Re: new here, my lang project...

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

>  
> *however, these will not run on a 386, as the 386 lacked a fpu which is  
> required for quake 1/2. a 386+387 could conceivably run quake2, albeit with  
> terrible performance (quake was bad even on early pentiums, one needed at  
> least a 133 or 166 pentium to run quake well, pentium 90's ran it poorly,  
> and a 486DX was worse, but still playable...).*

Even in the Pentium world, the minimum configurations for games has been steadily rising each year. And minimum configurations are usually a joke to sell to a wider audience; even the recommended configurations tend to be on the slow side.

>>*This is a Catch-22 that has plagued the industry for decades. I can  
>>remember IBM forcing their 360 mainframe IT users to upgrade their OS/360.  
>>When they did it would turn out they couldn't get their year-end closing  
>>of the books done in time without also upgrading the mainframe as well. A  
>>few years ago Dijkstra did a piece on software bloat and inefficiency in  
>>an IEEE/ACM rag. The designers of X-Windows said that they had designed  
>>it for a machine that didn't exist yet (and probably still doesn't!). It  
>>goes on and on; code bloat and inefficiency are epidemic in today's  
>>software.*

>>

>

> *yes, true, but the net result is not that bad really, apart for the need for  
> ever faster computers...*

It's bad when everyone is doing it, including the OS, device driver, and graphics services vendors. It is also bad when a modern PC configuration routinely has 20-30 applications running in background.

>><soapbox>

>>*Today we deal with interoperability by converting everything to ASCII,  
>>shipping it, and converting it back to something computable when it gets  
>>there. That is a horrendous performance penalty to pay for  
>>interoperability. Worse, it has crept into everything in the service  
>>infrastructures so one is saddled with the overhead even if one's  
>>application doesn't need interoperability. And it is all because the  
>>hardware vendors can't agree on a standard for endian or where the lsb  
>>goes after seven decades of building hardware!*

>>

>

> *this is just one issue.*

>

> *in general it is easier to come up with general purpose textual formats than  
> binary ones, and people are a little more able to agree on textual ones as  
> well.*

I have to disagree. Messing with ASCII for anything other than external configuration data usually means writing extra code and it is hugely inefficient, especially in a computationally intensive application like a computer game.

Re: new here, my lang project...

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

Worse, it leads to bad practices. Look at the overuse of hash tables. Hashing is a form of distribution sorting where performance depends on /each/ bit in the key being randomly set over all stored values, which is never true for ASCII.

>>>>What I am asking about is how there can be no compiled code. I thought  
>>>>the engine you were working on was in C.

>>>>

>>>>You will also have to put some more words around the notion of an  
>>>>inheritance model outside the app code in terms of files and directories.

>>>>

>>>>

>>>I am not getting what you are asking (possibly due to beers count).

>>>but, anyways, are you familiar with bytecode?...

>>

>>Yes; basically an intermediate code between 3GLs and Assembly often used  
>>by interpreter run time engines. (It was pioneered in R-T/E for things  
>>like DSP, usually burned into ROM.) Also used as a common output for  
>>multiple 3GLs so that one can apply back-end optimization one way for all  
>>the languages. Also used for portability by substituting the run time  
>>engine for local environments, as in Java.

>>

>>But one doesn't write bytecodes directly; they are compiled from the 3GL  
>>code that the developer creates by the compiler's lexical analysis and  
>>preprocessor functions. (The difference between a compiler and  
>>interpreter in the bytecode context is simply whether the bytecodes are  
>>subsequently interpreted or converted directly to machine language at run  
>>time; the 3GL code is still compiled to produce the bytecodes in both  
>>cases.) So I still don't know what you mean by "no compiled code". Are you  
>>saying you produce bytecodes directly?

>>

>

>

> yes.

Yech. That's why god invented 3GLs. B-)

> a good portion of my code is dynamically loaded, bytecode compiled, and is  
> then interpreted at runtime.

>

> the rest is plain c, and often consists a lot of infrastructure and such.

>

>

>>>in the case of the second part, it is a combination of an abstraction,  
>>>and some amount of code to do something along those lines.

>>

>>Alas, not a lot of information content here. B-) I still don't see the  
>>connection between OO inheritance and a file system directory structure,  
>>other than they are both represented by trees.

>>

>

Re: new here, my lang project...

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

- > *well, with the use of files, one can build a class-based inheritance tree on*
- > *top of the filesystem, effectively inheriting files and code from other*
- > *directories into a kind of conceptual "this" directory.*
- >
- > *one can also have links between the code and data as well (data can invoke*
- > *functions, and code can use data).*
- > *similarly, the directory-inheritance system can create objects that match*
- > *the inheritance graph and load code into them based on commands given in*
- > *files and such.*

You can do the same thing with C #include files. But I don't see how that has nothing to do with OO inheritance. What you are doing is creating collections of relevant files (object analogues). OO inheritance is about determining the properties of a single object (file analogue) representing the entire tree. The only thing they have in common is that a tree structure is navigated.

- >>*My basic point here is that, as described, your Physics has to do several*
- >>*distinct things:*
- >>
- >>*(1) Figure out how the bboxes get from the set of locations {A} to the set*
- >>*of locations {B}. That is, predict desired trajectories.*
- >>
- >>*(2) Figure out if bboxes ever occupy the same space when following their*
- >>*trajectories. That is, identify collisions.*
- >>
- >>*(3) Do (2) in real time for multiple entities.*
- >>
- >>*(4) If there is a collision, then figure out a new movement based upon the*
- >>*nature of the collision. That is, revise the trajectories from (1).*
- >>*However, this is a different problem because it involves additional*
- >>*physical properties of the entities and different laws of physics.*
- >>
- >
- > *5. do 3 and 4 until either things stabilize or a timeout is exceeded, in*
- > *which case the involved objects are just stopped.*

There is no iteration in time required. The next time slice has whatever new trajectories one computed in (4). If their incremental position at the end of that time slice ends in a new collision, one does (4) again to create yet a new set of trajectories. But one revisits (4) only when a new collision occurs in a new time slice, so the progression is always forward in time.

[Of course one can fudge special cases in (4). If the new trajectories already occupy the same space, then one might create an explosion graphic to cover rendering for a few time slices until things become stable. One can also split bboxes so that the sword is in between them initially in the display to represent a cut. (I also assume the fancy graphic services today can deal with notions like inserting one image into another.)]

Re: new here, my lang project...

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

>>>>*In that case there are only two possibilities for any given bbox*  
>>>>*incremental move: there is nothing in the way (i.e., no other bbox*  
>>>>*occupies the same end space) or there is something in the way (i.e.,*  
>>>>*another bbox occupies some part of its space). All the Physics subsystem*  
>>>>*needs to do is report when the bboxes collide to whoever is managing*  
>>>>*their skeletal models. IOW, Physics basically does nothing except check*  
>>>>*for collisions at the end of an incremental move.*  
>>>>  
>>>  
>>>*not so simple, or at least this is where "movetype" comes in.*  
>>  
>>*movetype?*  
>>  
>  
> *yeah.*  
>  
> *there is a system of "movetypes" and "solidtypes", which define respectively*  
> *how to go about moving from point to point or dealing with physical*  
> *properties, and how to go about dealing with the different ways of which to*  
> *represent a model that can be collided with.*  
>  
> *this is another one of those design features that was inherited from quake.*  
> *a single view of how to detect collision doesn't allow much variation in*  
> *model representation, and only a single way to represent object behavior*  
> *leads to the need for a lot of special cases and checks.*

That is only true when they are done together (i.e., both views are active at the same time). I suspect that is what is causing all your interaction problems. I have no problem with the concept of movetype vs. solidtype per se. I just want the context for each view to be properly separated so that they don't interact.

I assert that one could do that with the Skeletal Model for both situations. However, the view of the Skeletal Model needed to predict movement trajectories would have some different properties than the view needed for collision responses. In effect one would have a Move Skeletal Model and a Solid Skeletal Model, each relevant to its own subsystem subject matter.

I also assert that if one separates the contexts where the different views are required, then the logic for dealing with each view will become simpler. IOW, there will be no flipping from one view to another; each does its own <limited> thing in a very simple fashion.

I further assert that the geometry of the Skeletal Model is unambiguously convertible to other representations, such as an intersecting plane view of walls for graphics rendering. But one only needs the intersecting plane view because it is more efficient for some game entities when one is down in the Rendering subject matter where one is talking to specialized graphics cards that can optimize planes. (In fact, Rendering is probably a place that needs multiple views to use for

Re: new here, my lang project...

different entities at the same time.)

- >
- > *movetypes basically allow one to specify various sets of physics features an*
- > *object responds to, and how it goes about interacting with other objects.*
- >
- > *eg:*
- > *a character will move on impulses, and fall when being pulled by gravity and*
- > *when falling has little control over movement;*
- > *a door or platform will move in a specified direction a specified distance*
- > *and stop, it will exert force on any characters or objects in the way, but*
- > *otherwise it does not care about gravity or colliding with objects (quite*
- > *frequently doors or such will open or close in such a way that they pass*
- > *through other solid objects);*
- > *a missile will just fly through the air in a straight line and stop when it*
- > *hits something, meanwhile ignoring gravity;*
- > *a grenade will fly through the air while being effected by gravity, and will*
- > *bounce off any objects it collides with eventually coming to rest or such;*
- > ...
- >
- > *each is assigned a name and checks are added for it.*

All of which belong to the general subject matter of predicting unimpeded trajectories. To do those things one only needs to know certain ballistic characteristics about the entities. That should be completely independent from figuring out what to do when there is a collision, which will depend upon a whole flock of other properties.

If one synchronizes data like position in a systematic way, such as the end of a time slice, then the interactions between subject matters should be pretty minimal.

- >>>>*That means that someone else has to manage movement at the level of full*
- >>>>*sword swings. That someone -- let's call it the Move Manager*
- >>>>*subsystem -- understands what bboxes are involved in a sword swing.*
- >>>>*Using that knowledge they are responsible for the prediction portion*
- >>>>*above where they define the set of incremental moves for each relevant*
- >>>>*bbox to get the sword from its present position to the desired position*
- >>>>*after the full swing. Then for each time slice they are triggered to*
- >>>>*send the next set of incremental bbox moves to the Physics subsystem.*
- >>>>
- >>>
- >>>*bboxes are only one of the solid types.*
- >>
- >>*Why? The skeletal model composed of connected bboxes seems like a very*
- >>*general representation. Why wouldn't all game entities be represented*
- >>*that way? Immobile entities would just be simpler. IOW, it seems to me*
- >>*that articulation is just a property of the skeletal model.*
- >>
- >
- > *it is because, insufficient info exists to represent everything with bboxes*

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

- > *effectively.*
- >
- > *bsp trees are a clear issue here, namely you have a mass of faces that*
- > *organize themselves into a tree, and one can work by descending down one*
- > *side or the other of the tree checking for collisions, and ignoring sides*
- > *where a collision is impossible.*

BSP? This sounds like some sort of fractal representation(?). If so, that probably wouldn't be a good representation of boundaries needed for things like collision detection.

- > *other exceptions exist as well (eg: completely nonsolid objects, non-solid*
- > *objects that respond to collisions, ...).*

If they are not amorphous, then they can be represented with Skeletal Models; all they need are appropriate values for relevant characteristics (e.g., isHolographic).

If they are amorphous, then one does, indeed, need a unique representation. Life is rarely as simple as we would like.

>>>>*by a collision. Physics will also report to whoever is managing the Fred*  
>>>>*entity that there is now a sword buried in his head so that hit points,*  
>>>>*etc. can be adjusted.*

>>>>  
>>>>*Note that things like bouncing the sword off a shield or separating*  
>>>>*Fred's head from his body are not relevant to the actions so far. Those*  
>>>>*will trigger the appropriate /next/ full move by Move Manager when it is*  
>>>>*notified of the collision. That is, the bounce or whatever is a move in*  
>>>>*itself triggered by the <premature> completion of the previous move.*  
>>>>*Since the time slice is at the scale of the refresh rate, having both*  
>>>>*bboxes occupying the same display space is not going to be noticed.*

>>>>  
>>>>*[You will also note that Move Manager as described here does things*  
>>>>*(i.e., compute 3D trajectories) that are very close to the sort of stuff*  
>>>>*Rendering may do. But that is not quite so. Move Manager is an excise*  
>>>>*in analytic geometry for individual movements without regard to context.*

>>>

>>>

>>>*ok.*

>>>*again "sword embedded in fred's skull" is a concept for the script layer,*  
>>>*physics is more concerned with the movements and collisions involves just*  
>>>*stopping an action has sohwm itself to be an unreasonable approach.*

>>

>>*I understand that. My point is that Physics is doing too many different*  
>>*sorts of physics things. As I indicated above, I think there at least*  
>>*four different subject matters involved whose concerns need to be*  
>>*separated.*

>>

>

> *ok.*

Re: new here, my lang project...

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

>

- > *it is still an ugly system though, eg: how do I fully seperate movetype from*
- > *solidtype, or the different solidtypes?...*

Use the best abstractions for the relevant subject matter. The key is to abstract a subject matter that has limited responsibilities first. Then worry about what sort of abstractions will work best for its implementation.

I suspect that, with the exception of amorphous entities, the Skeletal Model will Just Work as a basic abstraction for movement trajectories, collision detection, and collision adjustments. The Skeletal Model abstractions won't be exactly the same in those subject matters; they just represent a convenient invariant that is easily mapped between the subsystems. However, the Skeletal Model probably doesn't work well for graphics rendering for certain types of entities (e.g., having planar geometries).

- > *the necessity of nested switch statements and lots of ugly interdependencies*
- > *doesn't seem easy to escape...*

That only happens if cohesion bleeds between subject matters. Isolate and encapsulate the subject matters behind a generic API and that problem should largely go away.

- > *yes, maybe movetype could be replaced by a mass of options and heavier (but*
- > *closer to "complete") physics, but this would somewhat increase*
- > *computational costs or difficulty in working with it (a grenade, for*
- > *example, would go from being "move\_bounce" to "object with low mass, highly*
- > *elastic, with zero acceleration" or such).*

This sounds quite ugly and complicated. I would expect things to be a whole lot simpler once the subject matters focused on narrowly defined problems like movement trajectories vs. collision detection.

- > *likewise, all models would need to have a consistent notion of "collision",*
- > *and a way to seperate it from the model.*

Not at all. Only one subject matter needs to have any notion of collision: the one that detects them and makes adjustments. Within that subject matter implementation one uses the representation of entities that makes those tasks the simplest.

[I would expect adjustments to be computing a force and direction for each bbox and shipping those back to whoever predicts motion trajectories. Whoever that is would not know or care why the forces and directions exist; they just compute a new set of trajectories.]

- >>>>*I am not sure what you mean by 'map' here, but I'll assume you mean*
- >>>>*something like terrain obstacles (trees, walls, etc.).*
- >>>>

Re: new here, my lang project...

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

>>>

>>>*vaguely, yes.*

>>>*it is more a mass collection of polyhedra.*

>>

>>*Relative to my point above about different representations, isn't a*

>>*polyhedron just a special case of a skeletal model that has one bbox with*

>>*a specific regular geometric shape? (Or, if you need to sometimes break*

>>*up the obstacles in an explosion, a single skeletal model with lots of*

>>*identical bboxes that don't articulate?)*

>>

>

> *not really, a bbox is always rectangular, but a polyhedron may not be (it*

> *can be a rough sphere or cylinder for example, or many other shapes).*

Aha! An abstraction disconnect. My OO abstraction bias is showing through; I never even thought of such a restriction. B-) Who says a bbox has to be rectangular? Why can't it be any regular polyhedron or else some reasonably sized set of geometric shapes?

> *similarly, it is not possible to make the same optimizations as polyhedra*

> *with bboxes and vice versa.*

>

> *however, the cost of efficiency is arguably great, but imo dealing with a*

> *system where all object types are unified is worse (a lot slower, more*

> *cumbersome, ...). this is what my previous ones were.*

Only if the representation is ill-suited to the subject matter responsibilities. If one picks the "best" representation for the subject matter one is probably giving highest weight to computational efficiency.

[I happen to like the notion of Skeletal Models w/ bboxes because it seems quite general and I can envision ways to make things like collision detection very efficient using them. But don't get hung up on that. (I am not even convinced it is the best solution for collision detection; I am using it mainly as an example of cohesive subject matter views).

One uses the best representation for the subject matter in hand. Because subject matters are narrowly defined, it is fairly likely that one will need only one representation to express the invariants of a particular subject matter. But that doesn't mean one would never need multiple representations in a subject matter -- that's why OO has subclassing relationships.]

>>>>*The answer here is to provide the map entities as exactly the same sort*

>>>>*of skeletal models with bboxes as the non-map entities within the Physics*

>>>>*subsystem. They just don't move a lot.*

>>>>

>>>>

>>>>*problematic and unreasonable.*

Re: new here, my lang project...

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

>>

>> *Why?*

>>

>

> *it would involve a lot of math that would be unneeded and unreasonable.*

I don't see that at all. Using one representation should be simpler than using multiple representations; one just has a single set of algorithms vs. multiple sets. It also eliminates "glue" code for a combinatorial number of interactions between different representations.

> *it would be difficult to optimize.*

Probably not resolvable here, but I see it exactly the opposite; the structure seems ideally suited for things like predicting trajectories and collision detection, especially if coupled with incremental movement based upon fixed time slices.

> *bboxes can't fully represent polyhedra, or make use of overlaps between them.*

That assumes one restricts the Skeletal Model to employ only rectangular bboxes.

> *however, often bboxes are used at least as a "first pass" in many cases, because a bbox can be checked quicker, and can be used to quickly rule out a lot of impossible collisions. in this sense, all objects do have bboxes, just a loose fit one to rule out impossible collisions, rather than to reliably detect the collisions themselves.*

I don't see that argument in a joy stick game. The player isn't calculating to mm tolerances. The scale is limited by the resolution of the graphics pane and what the player can actually see at the game display scale. At that scale one can be pretty loose with the tolerances of collision.

IOW, I see this as a precision vs. accuracy issue. When I take my temperature I want it to be pretty accurate about whether I am sick, but I don't care about six decimal places; one decimal is more than enough.

Similarly one doesn't want to get into fractal tolerances in computing whether there is a collision or not.

>>>>*In the suggestions above one idea was that Physics was limited to /only/ thinking about position. The state was handled elsewhere. So one didn't need to think about forces because of the time slice granularity. (One probably would have to think about forces in determining the trajectory of the new move, but that would be Move Manager's union.)*

>>>>*[In reality, I think my solution is an oversimplification. I would be inclined to separate out things like impact analysis into a different flavor of Physics as a subsystem. It would be limited to dealing with*

Re: new here, my lang project...

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

>>>>*the "new moves" resulting from collisions. It would understand rebounds, slicing, etc. to create the resulting trajectories for all the bboxes involved in the collision based upon their properties and would ripple outwards through the connected bboxes. In effect it would replace Move Manager in the special situation of collisions for all the involved entities.]*

>>>>

>>>

>>>*dunno.*

>>

>>*Don't know what? That is would be a better organization? That is could*

>>*be done? That the code you already have could be modified to partition*

>>*this way?*

>>

>

> *pretty much.*

>

> *I don't really understand how the movement can really be seperated from the*

> *collision detection in any really reasonable way.*

That is the point of synchronizing around fixed time slices. The collision is detected at the end of the time slice by simply looking for overlap in positions of bboxes. So all collision detection needs is the positions of the bboxes at the end of each time slice. There is no dynamics involved at all.

If the time slice is small enough the player will never see the messy overlap of bboxes at the end of the time slice where the collision is detected even if it is rendering in the display. For the next time slice the collision will be resolved and new movements will have <hopefully> corrected the problem.

> *I have spent a fair amount of time staring at the code to try to figure ways*

> *to clean it up.*

You probably can't. B-) The scheme I suggested is architectural. I would bet it could not be implemented without effectively rewriting most of the code. I think the same would be true of any other clean up alternative. IOW, the design you are working with sounds like it is architecturally flawed because the application is not well modularized.

>>>>>*there is probably at least some reason why a lot of recent game*

>>>>>*developers are outsourcing the game physics to other companies...*

>>>>

>>>>*The algorithms for determining something like how a shadow should look*

>>>>

>>>>*from and arbitrary light source are nontrivial. So are algorithms for*

>>>>

>>>>*determining what's hidden from view, trajectories for multiple bboxes*

>>>>*when swinging a sword, and a bunch of other stuff. Perhaps more*

>>>>*important is that the algorithms are computation intensive and they all*

Re: new here, my lang project...

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

>>>> *need the same CPU, so optimization is very important. IOW, that requires*  
>>>> *a special skill set so it pays to specialize.*  
>>>>  
>>>  
>>> *no, shadows are a rendering concern.*  
>>  
>> *It is just a different suite of algorithms to apply physics of a*  
>> *<somewhat> different sort. Skeletal models, trajectories, collisions,*  
>> *rebounds, ray tracing, whatever -- it's all basically the same science*  
>> *whose flavors need to be partitioned more than just Physics vs. Rendering.*  
>> *(I would argue that the Rendering needs to be partitioned in the same*  
>> *manner as we have been talking about Physics.)*  
>>  
>  
> *yes.*  
>  
> *luckily at least rendering is at least a lot more partitioned normally, I*  
> *just have to try to keep it from being assimilated entirely into physics and*  
> *the code for dealing with fileformats...*

That's because it was initially the biggest problem and because it is very nicely defined in a mathematical sense. Also, one is basically dealing with well organized hardware (nowadays standardized OS graphics services). That sort of well-defined problem tends to get solved first.

The physics issues are also well defined in the algorithmic sense. However, it sits between the game semantics, which are not mathematically defined, and the standardized rendering solutions. That presents the temptation to bleed game space into rendering space over the body of physical laws.

Even if the rendering is highly modularized, one can still be victimized by the API. If the API reflects the internal rendering structures, the the rest of the application is exposed to those internals and must be built around them. That alone can create unnecessary dependencies.

bottom line: the physics stuff will tend to be less well formed than the rendering stuff.

> *sometimes I wonder if and how all these problems have been approached by*  
> *newer commercial games...*

I have no solid information. However, from various snippets from online forums like this one over the years my impression is Not Very Well Yet.

That's hardly surprising because most of the code in the entire software industry is pretty bad. To paraphrase G. B. Shaw on Christianity, the only problem with good software development is that it has never been tried.

\*\*\*\*\*

There is nothing wrong with me that could

Re: new here, my lang project...

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

not be cured by a capful of Drano.

H. S. Lahman

[hsl@pathfindermda.com](mailto:hsl@pathfindermda.com)

Pathfinder Solutions -- Put MDA to Work

<http://www.pathfindermda.com>

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

(888)-OOA-PATH