

## Re: new here, my lang project... PT2

**Source:** <http://coding.derkeiler.com/Archive/General/comp.object/2005-02/0008.html>

---

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

**Date:** 01/31/05

Date: Mon, 31 Jan 2005 20:46:59 GMT

Responding to Cr88192...

>>>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.  
>>  
>  
>  
> usually, however, for many things fairly precise detection is needed.  
> is the player on the ground, above the ground, touching a wall, ... ?  
>  
> to answer these requires a lot of checking, and at least reasonably precise  
> checks.  
>  
> the possibility of multiple simultaneous collisions also, and one can't have  
> due to precision errors the player sliding and managing to fuse into the  
> floor or the object.

I still don't see a need for great precision. I can understand  
Rendering needing to know if objects are actually touching. However,  
whoever is figuring out the movement can simply announce that to  
Rendering when it decides they are close enough to qualify.

here will always be a scale mismatch problem. Say the Ogre and Hero are going at it. The display view may be 10 m wide. That's 10K mm of precision but you only have 1200 pixel widths of resolution. So it really makes no sense to compute collisions with less than 1 cm precision. But that makes anything in the 0–10 mm range be touching.

>>>*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.*

>>

>

> *ok.*

>

>

> *it almost makes sense to drive hard walls between the subsystems.*

Yes. I usually refer to the subsystem interfaces as 'firewalls' because it is so important to decouple the implementations. That's why I keep pushing on data transfer message interfaces where messages simply announce something has happened or is needed by the sender.

> *eg, everything could be split into several parts:*

> *mathematics and interface core (lots of shared math functionality and*

> *similar, basic entity and message passing stuff, ...);*

> *render core (everything related to rendering, and possibly also sound,*

> *input, and the gui);*

> *physics core (physics functionality, little else);*

> *formats core (manages file formats);*

> *glue and misc (a library for gluing together the previous, and misc crap*

> *that doesn't belong in the others).*

I think this is a good start. But for anything as complex as a modern computer game, I would expect some of these could be broken down further. For example, computing movements for acts like swinging a sword seems like one sort of physics (inertia, angular velocity, etc.). But that seems like a very different sort of thing than applying physics (elasticity, angle of incidence, etc.) to computing what happens when entities collide.

>

> *this is would be similar to the division within the more basic libs:*

> *pdbase (mm/fileio/vars/... also includes data compression);*

> *pdnet (networking code);*

> *pdyfs (handles filesystem abstractions/emulation);*

> *pdsript (manages language interpreters);*

comp.object: Re: new here, my lang project... PT2

- > *pdglue (meant to handle gluing the previous together, but is little used).*
- >
- >
- > *just I am not sure how necessary the partitioning is at present.*
- > *I had started partitioning the systems before, but stuff changed around a*
- > *lot. the centralized rendering tended to dissolve, physics rose up and began*
- > *dominating a lot of other things, ...*

At present I am skeptical there is much you can do. Application partitioning is a very fundamental architectural decision that provides the framework for everything that is developed within those partitions.

Unless you are prepared to start over, I don't think partitioning is going to be very viable.

[Caveat. One can use good application partitioning for piecemeal replacement of legacy code when it is infeasible to replace it all at once. It is still painful, but there are ways to reduce the risk. There is yet another category on my blog that deals with Legacy Replacement. So you could do a rewrite as a long-term proposition, one subsystem at a time.]

\*\*\*\*\*

There is nothing wrong with me that could 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.blogs.com/hslahman>

(888)-OOA-PATH