



igCollision

jonathan garrett

1/29/07

introduction

- igCollision - low-level core collision system
- focus on fast asynchronous queries
- game independent
- initial pc port completely re-architected for ps3

topics

- key-features of the system
- overall flow from issue of query to sync
- breakdown of main components
- fast ppu-spu interaction
- issues
- improvements
- questions

key features

- different query types
- all queries against `Object` struct
- static and dynamic objects
- `Result` struct holds single intersection
- can ask for list of `Results`
- can ask for list of prims/tris as result
- spu with virtually no ppu interaction

overall flow

- ppu initiates a query
- spu services it
- ppu stalls on (or polls for) result

overall flow

- spu system split into two phases
- broad-phase – creates likely candidate list
- narrow-phase – intersects candidates against Objects
- two phases run on separate spus

ppu

broad spu

narrow spu



broad-phase

- creates input list for narrow-phase
- considers just aabb of each Object
- narrow wants Object ptr, packet ptr, packet size
- many objects in scene, not enough space on spu
- CollDbObjId struct contains minimum data
- list of these stored on ppu
- way to get to these stored on spu

broad-phase

- voxel-grid accelerates gross-level queries
- 4x4x4 grid of hierarchical cells
- highly compressible (bit per child)
- minimally recursive (depth of 3 or 4) – spu branches expensive
- at each step, current voxel level intersected with query volume (fast – aabb is 30 instructions, no branches)
- end up with list of occupied children to recurse on
- final output list of “descriptors”

broad-phase

- descriptor defines list of CollDbObjIds in a cell (20 bits for ptr/16, 12-bits for count)

descriptors - spu

ptr	size
ptr	size
...	...
...	...

CollDbObjIds - ppu

aabb	flags	Object ptr	packet ptr	packet size
aabb	flags	Object ptr	packet ptr	packet size
aabb	flags	Object ptr	packet ptr	packet size
aabb	flags	Object ptr	packet ptr	packet size
aabb	flags	Object ptr	packet ptr	packet size
aabb	flags	Object ptr	packet ptr	packet size
aabb	flags	Object ptr	packet ptr	packet size
aabb	flags	Object ptr	packet ptr	packet size
...
...

broad-phase

- efficiently process batches of `CollDbObjIds`
- build batches to fill spu input buffer
- double-buffer from ppu to spu – dma in next batch as process current

broad-phase

- aabbs in each batch intersected against query volume
- object flags tested for match against query flags (eg. obj-type, owner-type)
- passing `CollDbObjIds` added to narrow input

broad-phase

- dynamic query adds to end of static list
- upto 1024 objects – queries done against bsphere
- bsphere, flags, packet ptr, packet size stored on spu (updated each frame)
- uses implicit-grids (1024 bits per cell in x, 1024 bits per cell in z)
- `or` of overlapped cells in x `and`-ed with overlapped cells in z to give combined overlap

broad-phase

- optionally sends list of passing Object indices back to ppu
- flags ppu and narrow-spu that broad-phase is complete

narrow-phase

- efficiently process input Object list
- need data from CollDbObjId - dma from broad spu
- then need Object and packet-data – dma from ppu
- staggered double-buffer – process current, dma in next Object and packet, dma in next+1 CollDbObjId

narrow-phase

- process packet – switch on query-type
- handle specific prim/tri-mesh intersection as appropriate

narrow-phase

- prim – union of all types – uniform size
- loop over prims - perform query vs prim, track closest intersection

narrow-phase

- tri-mesh – packed format – preprocess to unpack
- max 512 verts, max span of 64m
- verts: 1.5.10 format – 6 bytes
- tri indices/flags: 9.9.9.5 format – 4 bytes
- loop over tris - perform query vs tri, track closest intersection

narrow-phase

- finalize stage fills out `Result` struct (only partial result calculated during loop)
- sends `Result` back to ppu
- flags ppu that narrow-phase is complete

narrow-phase

- multiple Results or prim/tri list
- builds results-buffer, overlapping dma back to ppu of previous batch with processing of current

deferred

- fully asynchronous
- standard and low-priorities
- standard: sync by end of frame
- low: sync by end of next frame
- standard has priority over low
- immediate has priority over deferred
- better facilitates “free” queries

juggling asynchronosity

- spus run asynchronously to the ppu
- communication incurs large latency
- so overlap spu work with other ppu work
- dma also asynchronous
- so overlap dma in/out with processing – double/triple buffer as needed
- collision also overlaps two spus – narrow starts processing as soon as broad has first candidate

issues

- took two spus
- game had to retrofit asynchronous system
- wanted more deferred queries (mainly a memory tradeoff)
- simplicity of deferred allocator prevented freeing results
- confusing query flags
- difficult to debug – especially deferred

improvements

- combine broad-narrow on single spu (maybe)
- improved primitive support (cylinders – mark)
- issue deferred queries from other spu system
- improve deferred-allocator
- improved scheme for switching out static-objects
- user requested improvements
- optimize – looking at order of magnitude improvement ($> 5x$)

end!

- questions ?

