Software Engineering and VR

The Devil in the Details

Tom Forsyth

Software Architect, Oculus

Jan 27th 2015

Oculus™

Tom Forsyth

- Graphics drivers @ 3Dlabs
- Game graphics @ Muckyfoot
    - Urban Chaos
    - Startopia
    - Blade 2
- Animation middleware: Granny3D @ RAD Game Tools
- Instruction set design on Larrabee / Xeon Phi @ intel
- VR @ Valve
    - VR support on Team Fortress 2 & Half Life 2
- Software Architect @ Oculus
    - All things graphics
    - Distortion & calibration, interacts with lens design
    - Best practices, dev.rel. support, psychology

VR is awesome!

VR is awesome!

VR is awesome!

VR is awesome!

VR is awesome!

VR is awesome!

VR is awesome!

- VR is also incredibly tricky

VR is awesome!

- VR is also incredibly tricky
- “Bleeding-edge realtime 3D graphics” is the easy bit
    - Lots of hard stuff here of course, but needs an entire lecture series

VR is awesome!

- VR is also incredibly tricky
- “Bleeding-edge realtime 3D graphics” is the easy bit
  - Lots of hard stuff here of course, but needs an entire lecture series
- VR adds so many other areas
  - Display and optics
  - Hardware diversity
  - Time and latency
  - Bugs that take a long time to manifest
  - Interactions with user physiology

VR overview

Hardware:

- Screen
- Lenses
- IMU (gyro + accelerometer)
- Camera & LEDs

(and lots of scary details!)

VR overview

Software:

- Tracking
- Prediction
- Rendering
- Timewarp
- Distortion & processing

(all timings are examples, not any specific device)

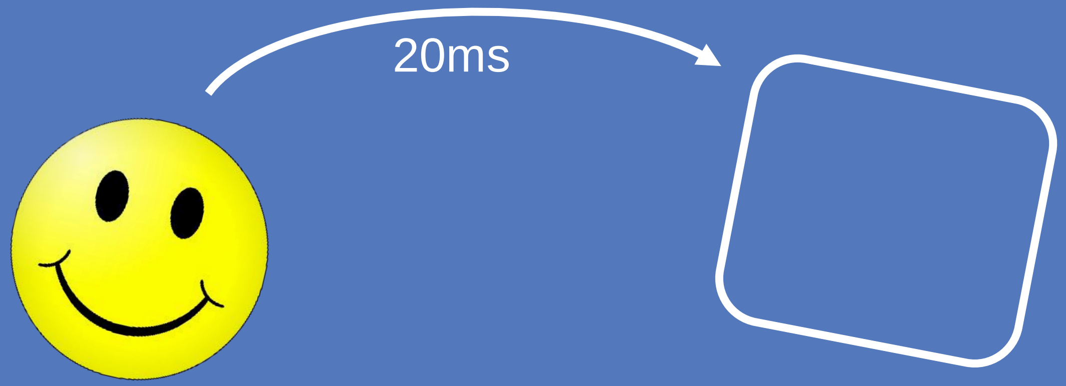
Software in detail

- Read 60Hz camera
  - Find LED dots
  - Find HMD pose from dot positions

Software in detail

- Read 60Hz camera
  - Find LED dots
  - Find HMD pose from dot positions
- Read 1000Hz IMU: gyro + accel
- Sensor fusion
  - Last known-good camera pose
  - Integrate IMU data forwards
    - (this is the concept - in practice it’s a continuous filter)

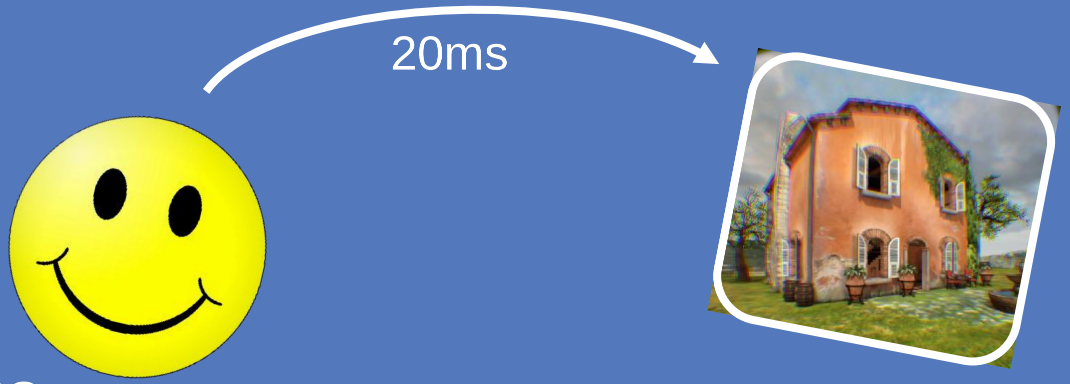
Software in detail



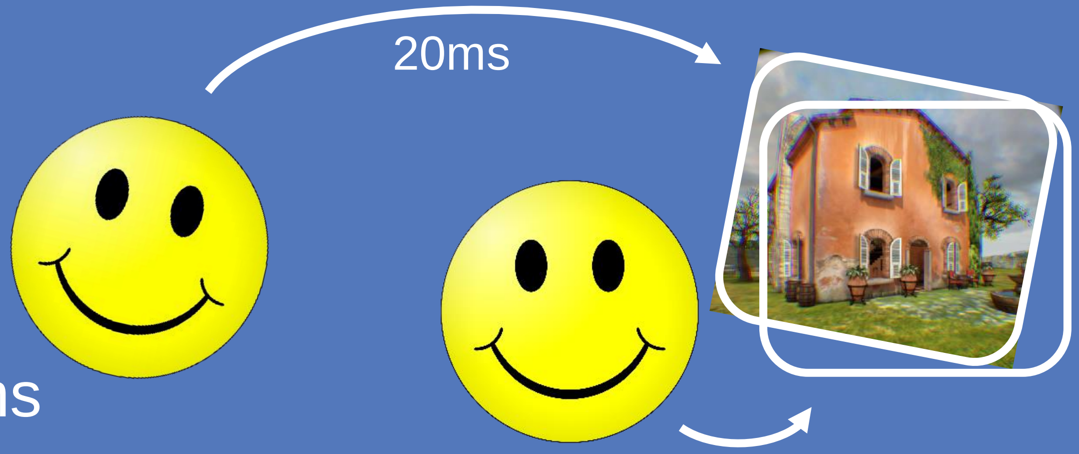
- Read 60Hz camera
  - Find LED dots
  - Find HMD pose from dot positions
- Read 1000Hz IMU: gyro + accel
- Sensor fusion
  - Last known-good camera pose
  - Integrate IMU data forwards
    - (this is the concept - in practice it’s a continuous filter)
- Predict motion forwards to next frame display time
  - Will typically be ~20ms away

Software in detail

- App renders view from each eye
  - Typically takes a frame, i.e. 10-15ms

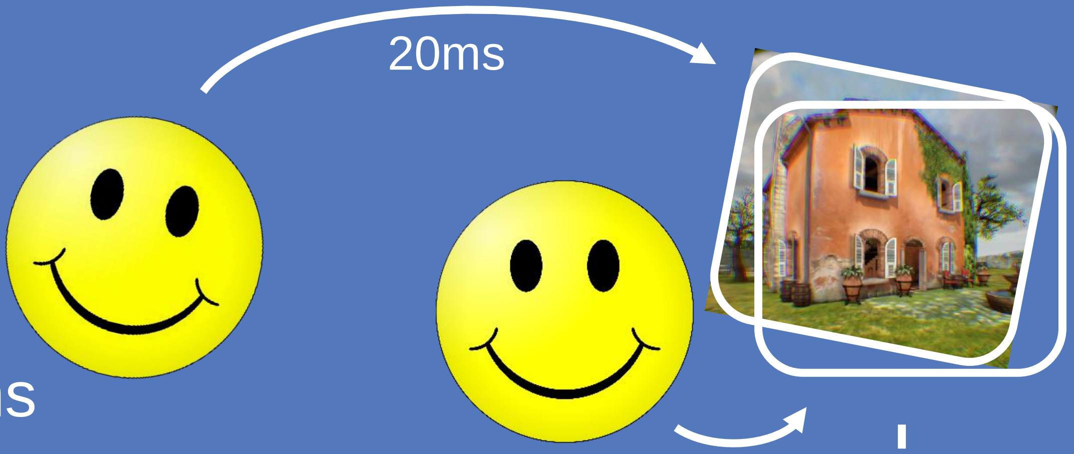


Software in detail



- App renders view from each eye
  - Typically takes a frame, i.e. 10-15ms
- Re-read latest IMU data and re-predict to next frame
  - Prediction is now only ~5ms
  - Timewarp

Software in detail



- App renders view from each eye
  - Typically takes a frame, i.e. 10-15ms
- Re-read latest IMU data and re-predict to next frame
  - Prediction is now only ~5ms
  - Timewarp
- Image processing
  - Distortion
  - Chromatic aberration
  - Compositing
  - Gamma curve
  - Overdrive

Doing the timewarp

- Reprojection of a rendered scene to a different orientation
  - Rotational works well, positional less so
- Three separate effects in one mechanism

Doing the timewarp

- Reprojection of a rendered scene to a different orientation
  - Rotational works well, positional less so
- Three separate effects in one mechanism

1. Late IMU read, re-predict, and fixup
   - Smaller prediction time = much less error

Doing the timewarp

- Reprojection of a rendered scene to a different orientation
  - Rotational works well, positional less so
- Three separate effects in one mechanism

1. Late IMU read, re-predict, and fixup
   - Smaller prediction time = much less error
2. Rolling shutter compensation
   - The display lights up a line at a time, not all at once

Doing the timewarp

- Reprojection of a rendered scene to a different orientation
  - Rotational works well, positional less so
- Three separate effects in one mechanism

1. Late IMU read, re-predict, and fixup
   - Smaller prediction time = much less error
2. Rolling shutter compensation
   - The display lights up a line at a time, not all at once
3. Mitigation for stutters and stalls (“async timewarp”)
   - Makes them less painful, but still look bad
   - This does NOT mean you can render at 30fps!

Sounds simple, what could go wrong?

VR brings some specific problems to the picture

- Timing is critical
  - VR relies on correct prediction
  - Errors in timing are difficult to diagnose
- Hardware configuration & variability
  - GPUs
  - Power management
  - USB devices
- Human variation

Timing, timing, timing

- Rendering uses a lot of predicted HMD poses
  - Requires high-quality data to do good prediction
  - Requires the actual display time to be within ~2ms of predicted
  - Being an entire frame late or early (~12ms) is throw-against-wall bad

Timing, timing, timing

- Rendering uses a lot of predicted HMD poses
    - Requires high-quality data to do good prediction
    - Requires the actual display time to be within ~2ms of predicted
    - Being an entire frame late or early (~12ms) is throw-against-wall bad
- Incorrect timing is very difficult to see
    - Even experienced people can’t reliably see <10ms errors
    - But they will make people nauseous after 15mins
    - “Works on my machine” isn’t even reliable!
    - Most automated testing only does functionality, not timing

Hardware configuration problems - GPUs

- VR requires a fixed, consistent framerate
  - Apps do not have fixed frame content
  - Timewarp can help mitigate the worst stutters

Hardware configuration problems - GPUs

- VR requires a fixed, consistent framerate
  - Apps do not have fixed frame content
  - Timewarp can help mitigate the worst stutters
- Massive range of performance in desktop PCs
  - Multiple axes of perf: drawcalls vs compute vs bandwidth
  - Any can be the limit for different parts of the scene

Hardware configuration problems - GPUs

- VR requires a fixed, consistent framerate
  - Apps do not have fixed frame content
  - Timewarp can help mitigate the worst stutters
- Massive range of performance in desktop PCs
  - Multiple axes of perf: drawcalls vs compute vs bandwidth
  - Any can be the limit for different parts of the scene
- GPU APIs & system optimized for throughput, not low latency
  - Rendering pipeline has very low-quality timing feedback
  - Use big hammers to solve – lots of stalls & syncs
  - Wastes a lot of performance

Hardware configuration problems - Clocks

- CPU and GPU will throttle/overclock according to thermals
- Precise CPU clock measures cycles – changes with frequency!
  - Reliable “chipset” clock has coarse granularity
  - We use one to sync the other - mostly works
  - Finally fixed with Skylake CPUs – reliable AND precise
    - But Skylake isn’t out yet…
- Especially difficult on mobile & laptops
  - Very aggressive throttling
  - Perf will change mid-frame

Hardware configuration problems - USB

- Data collected by CPU polling
- Hubs have unknown buffering granularity
  - Every PC has multiple hubs in it, plus external ones
  - Each adds unpredictable latency
- USB devices can block each other (async isn’t *that* async)
- HMD has a reliable clock, timestamps all events
  - But we still have to sync up HMD and CPU clocks over USB

Bug-fighting tools

- Capture and reply of sensors
- Robots & mo-cap
- Motion-to-pixel tests
- Feedback from HMD

Capture and replay of sensor inputs

- Outputs compared with known-good results
- Good for sensor algorithm development
  - Blob finding
  - Pose estimation
  - Fusion of IMU and vision
- Good for nightly tests
  - Easy to accidentally add noise, off-by-one-sample errors during dev
- Good for performance tests
  - Completely repeatable inputs

Robots & mo-cap

- Robot arm with HMD mounted on it
  - Makes known & somewhat repeatable motions
  - Tests hardware sensor error, noise, temperature, etc
- Cube/room of professional mo-cap cameras
  - Captures actual human movement
  - Data is not as clean as you'd like – lots of filtering neede
- Hard to run nightly or regression tests
  - Slightly different every run
  - Changing HMD or camera requires recalibration
  - But we're starting to try it

Motion-to-pixel tests

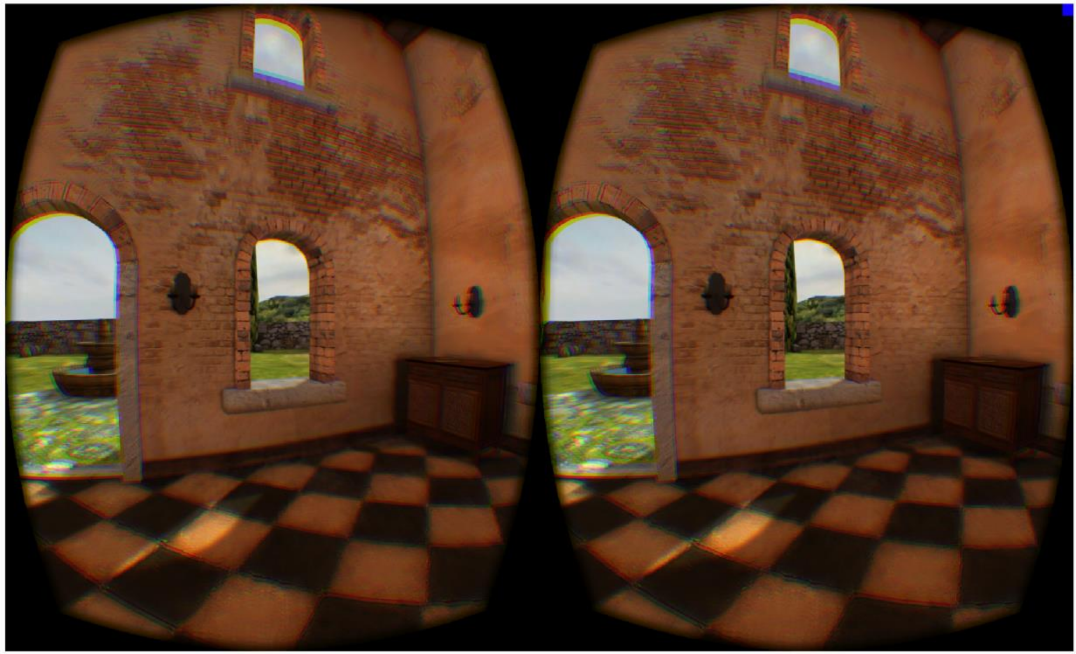
- Feed sensor-fusion outputs to rendering engine
- Optional deterministic time stream
  - Replace real clock with recorded one
- Render & take screen captures

Motion-to-pixel tests

- Feed sensor-fusion outputs to rendering engine
- Optional deterministic time stream
  - Replace real clock with recorded one
- Render & take screen captures
- Comparing images is surprisingly difficult
  - “No change” optimisations can easily cause small pixel/colour changes
  - But are they correct changes or bugs? Requires operator skill
  - Differences between GPUs, even between driver versions
  - Bug fixes & new features require humans to re-approve new images
- Still setting this system up – results TBD

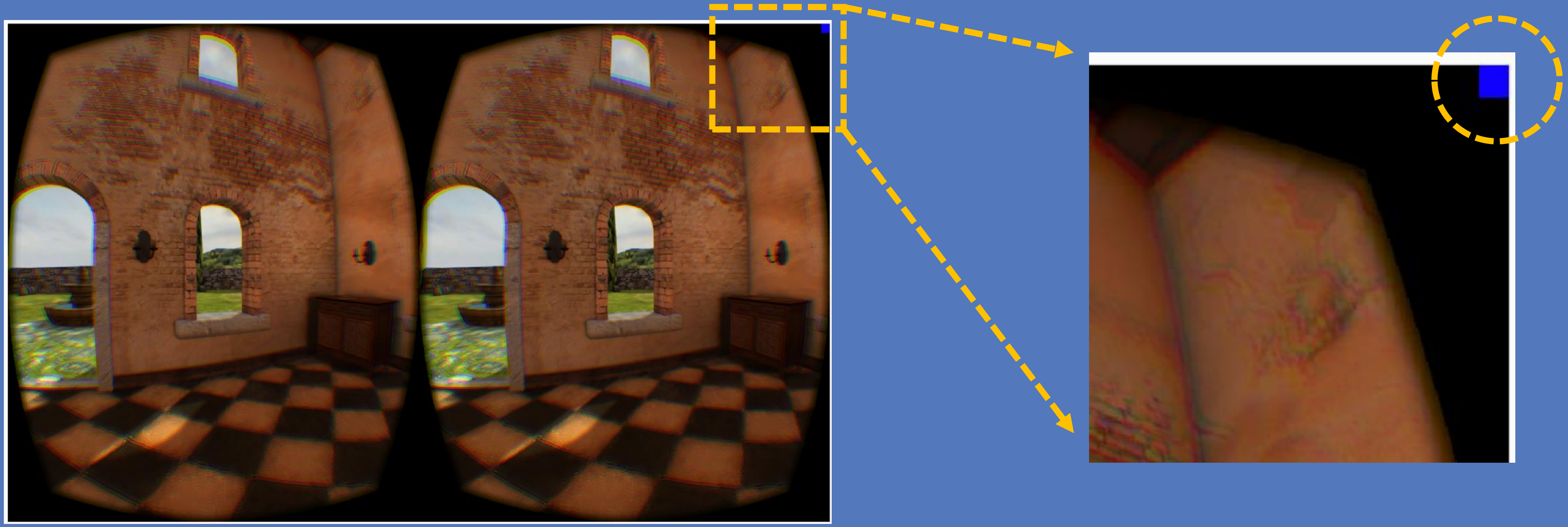
Feedback from HMD

- HMD sends top-left pixel value back over USB, with timestamp
  - Top-left pixel is beyond lens range, so we can use as a frame ID
  - Dynamic end-to-end latency test of graphics system



Feedback from HMD

- HMD sends top-left pixel value back over USB, with timestamp
  - Top-left pixel is beyond lens range, so we can use as a frame ID
  - Dynamic end-to-end latency test of graphics system



Feedback from HMD

- HMD sends top-left pixel value back over USB, with timestamp
  - Top-left pixel is beyond lens range, so we can use as a frame ID
  - Dynamic end-to-end latency test of graphics system
  - Used to fine-tune prediction algos
- Useful for helping a user set up a working system
  - Run simple Oculus test app, are you getting correct latency?
  - Spots dodgy drivers, broken cables, USB problems, etc

Feedback from HMD

- HMD sends top-left pixel value back over USB, with timestamp
  - Top-left pixel is beyond lens range, so we can use as a frame ID
  - Dynamic end-to-end latency test of graphics system
  - Used to fine-tune prediction algos
- Useful for helping a user set up a working system
  - Run simple Oculus test app, are you getting correct latency?
  - Spots dodgy drivers, broken cables, USB problems, etc
- Not as useful to a running app
  - What can an app do with “latency is too high” message?
  - Dynamic scaling of rendering speed has difficult hysteresis & feedback
    - Still a hard research problem

Bug-fighting methodologies

- Naming schemes
- Code standards
- Multiple pairs of eyes
- Unit tests / test-driven development
- Asserts
- Code reviews

Naming schemes

- Not that important, just pick one
  - CamelCaps vs under_scores
  - local/m_Member/g_Global
  - szHungarian

Naming schemes

- Not that important, just pick one
  - CamelCaps vs under_scores
  - local/m_Member/g_Global
  - szHungarian

- We chose:
  - CamelCaps
  - localVar/MemberVar/GlobalVar
  - no

Naming schemes

- Not that important, just pick one
  - CamelCaps vs under_scores
  - local/m_Member/g_Global
  - szHungarian
- We chose:
  - CamelCaps
  - localVar/MemberVar/GlobalVar
  - no
- Spaces and units – important!
  - EyePos -> EyePosInHead
  - Transform -> HeadFromWorld (see blog post for details)
  - Delta -> VsyncDeltaMillisecs
  - Brevity is not a virtue – take advantage of autocomplete!

Code standards

- In the sense of “which features of C++ do we use?”
- Extremely effective

Code standards

- In the sense of “which features of C++ do we use?”
- Extremely effective… at causing arguments
  - Functional purists
  - Template metacoders
  - Inheritance-tree huggers
  - Abstractionistas
  - C99ers

Code standards

- In the sense of “which features of C++ do we use?”
- Extremely effective… at causing arguments
  - Functional purists
  - Template metacoders
  - Inheritance-tree huggers
  - Abstractionistas
  - C99ers
- Conclusions fuzzy, but point towards using fewer language features
  - Sometimes called “C+” (many variants)
  - Fewer surprises, more typing
  - But coders are really good at typing

Multiple pairs of eyes

- Bugs can be invisible to some, really obvious to others
  - Always get others to check your visuals
  - Learn what you can and can’t see

Multiple pairs of eyes

- Bugs can be invisible to some, really obvious to others
  - Always get others to check your visuals
  - Learn what you can and can’t see
- Ten people can’t see a problem and one person can – it’s a problem
  - There is no “authority” in these cases
  - Deal with it the same way as colour-blindness (5%-10% of the pop)
  - Designated guinea-pigs for each artifact

Multiple pairs of eyes

- Bugs can be invisible to some, really obvious to others
  - Always get others to check your visuals
  - Learn what you can and can’t see
- Ten people can’t see a problem and one person can – it’s a problem
  - There is no “authority” in these cases
  - Deal with it the same way as colour-blindness (5%-10% of the pop)
  - Designated guinea-pigs for each artifact
- But sometimes you have to pick your battles
  - 60Hz/75Hz/90Hz low-persistence
  - Fast-moving FPS games
  - Screen door effect vs blur

Unit tests / test-driven development

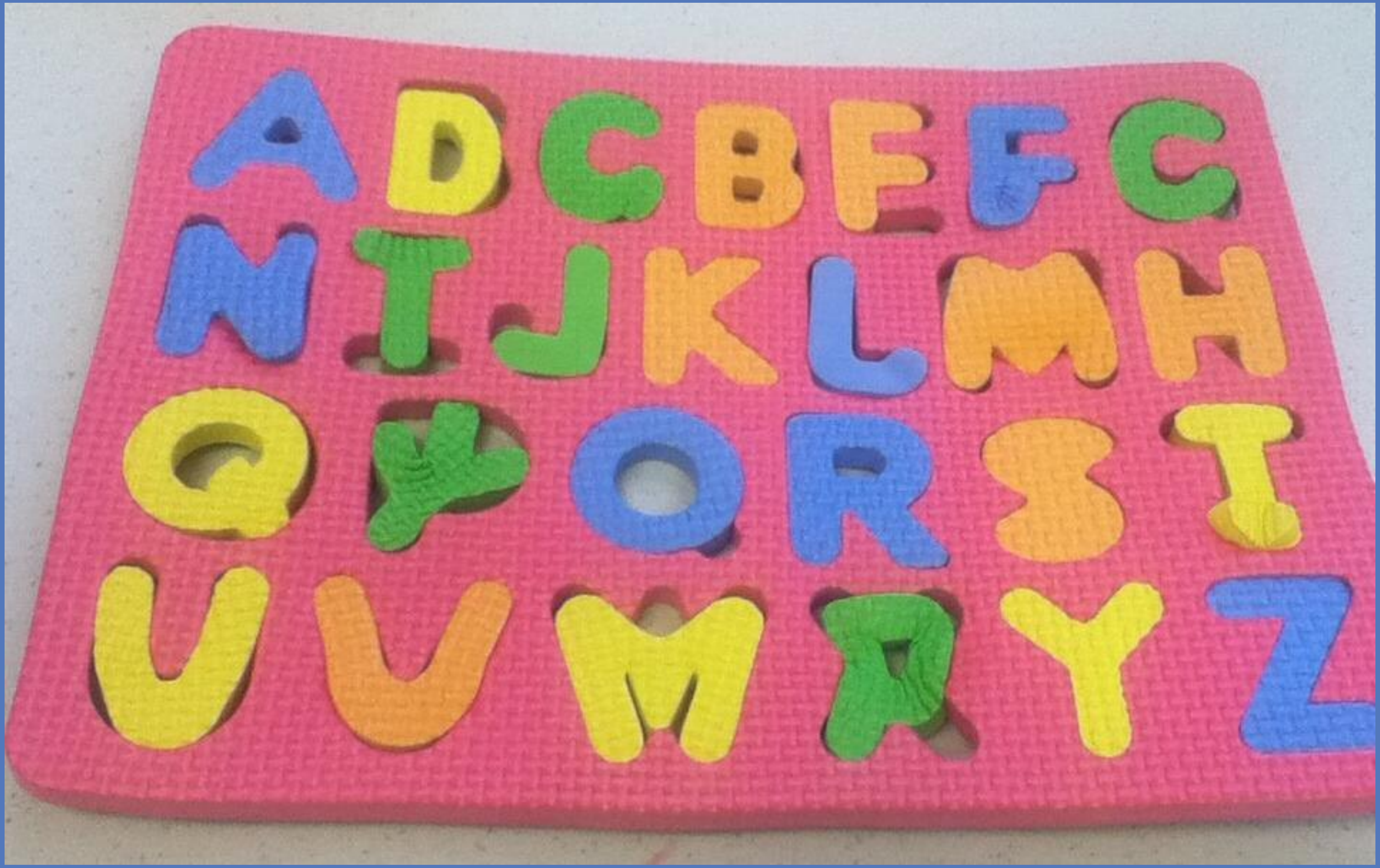
Unit tests / test-driven development

Analysing…

Unit tests / test-driven development

Analysing…

26/26 tests PASSED



Unit tests / test-driven development

- Very little help in subjective algorithm development
    - Only work for refactoring and optimization
    - We just don’t do that very much! VR does not have “big data”
    - Not much use for GPUs
- Actively impede algorithm development

Unit tests / test-driven development

- Very little help in subjective algorithm development
  - Only work for refactoring and optimization
  - We just don’t do that very much! VR does not have “big data”
  - Not much use for GPUs
- Actively impede algorithm development
- I don’t like the bang for the buck
  - Very few interesting bugs would have been caught with them
  - Trivial ones easily caught by other methods
  - Not worth the effort, complexity and maintenance

Asserts

- I love asserts
    - Personal codebase – 20% LOC are asserts!
- Double as documentation
    - Cannot go out of date like comments and docs
    - Especially on function arguments/inputs – ranges, allowed combos, etc
- Can be used for mini unit tests

Asserts

- I love asserts
  - Personal codebase – 20% LOC are asserts!
- Double as documentation
  - Cannot go out of date like comments and docs
  - Especially on function arguments/inputs – ranges, allowed combos, etc
- Can be used for mini unit tests
- Almost nobody dislikes asserts
  - Makes them almost unique amongst language features!
  - Though many aren't very aggressive about using them
- Lots of nasty bug-hunts could have been caught early by asserts

Code reviews

- Cardboard-cutout-dog effect
  - Just explaining it to someone else causes self-analysis
- Increases “bus factor”
  - Ownership = responsibility, not control. Tell people how your code works!
- However, can be a time sink
  - Leads to yes-men review-buddies
  - Making it online, not in-person can help (but reduces CCD effect)
  - Hard-and-fast rules create cheats – make it recommended not mandatory
- Various cultures within Facebook & Oculus
  - Different groups have different code review cultures

Bug-fighting methodologies – score sheet

• Naming schemes	• Pick one, almost any one
• Code standards	• Keep it simple
• Unit tests / TDD	• Nope
• Asserts	• Yes yes yes
• Code reviews	• Maybe, but don’t go nuts

…and that’s just the SDK!

- Developing an app that uses VR has a bunch of other fun
- All the usual complexity of realtime 3D rendering
- Big performance requirements
  - Mono@60º@30fps -> Stereo@100º@75fps (and that was *last* year)
- Input restrictions (can’t see a keyboard)
- Design restrictions
  - Can’t force camera angles – must follow the user’s head
  - Cinematic language reduced – framing, composition, angles, cuts
- For more, see my GDC2014 talk

VR is mean to brains

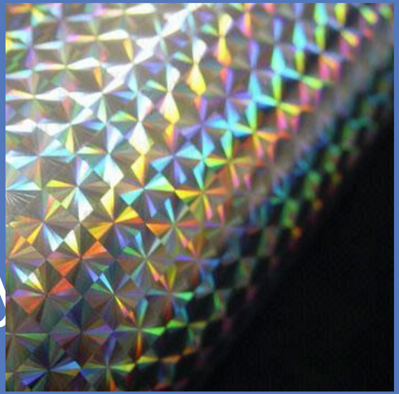
- World scale is due to multiple cues
  - Pupil separation and head motion must match physical user
    - Or be a consistent scaling
  - Height of virtual camera from ground
  - Physical dimensions of in-game avatar
  - Vergence vs focus conflict
  - Floor-dragging effect – your feet overrule your eyes
- Vestibular/optical mismatch – motion sickness
  - Ears say you’re sitting still, eyes say you’re moving
  - HW & SW working perfectly, but induces rollercoaster/travel/seasickness

VR is mean to coders

- Multiple bugs can look identical, e.g. “judder” can be:
  - Framerate dropping below HMD rate
  - Incorrect latency prediction
  - Incorrect physics simulation time
  - Misbehaving USB hubs
  - GPU buffering/syncing (especially with multi-GPU)
  - Misc other rendering bugs
- Errors frequently “invisible”, but still cause nausea
  - Extra frame of latency
  - Off-by-one-pixel rendering
  - Position tracking not working
  - Swapped eyes (yes, really)

VR is really mean to graphics

- Much of our graphics intuition becomes useless
  - Algorithms can be doing exactly what we want, yet feel terrible
- Normalmaps look rubbish with stereo & head motion
  - Looks like “hologram foil”
  - Need high-quality, physically consistent disp.maps
- Billboards and impostors have to be ~10x as far away
  - Grass, leaves, mist, effects
- Most specular methods don’t stereo-fuse properly
  - …and specular that does fuse shows up surface triangle structure ☹
- 2D screenspace labels don’t work – need in-world 3D ones

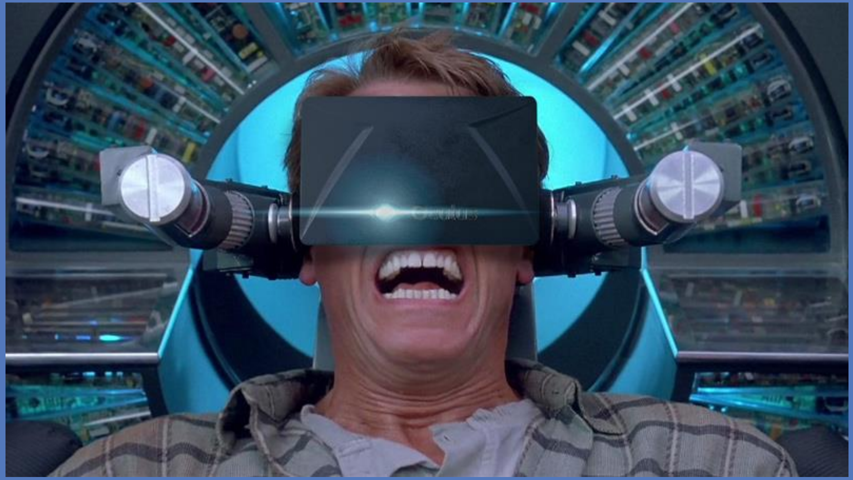


Conclusion

- VR is all the hard things rolled into one
  - Display and optics
  - Wide range of hardware
  - Time and latency
  - Bugs that take a long time to manifest
  - Interactions with user physiology

Conclusion

- VR is all the hard things rolled into one
  - Display and optics
  - Wide range of hardware
  - Time and latency
  - Bugs that take a long time to manifest
  - Interactions with user physiology
- But when it works it's awesome
  - Dumpy: Going Elephants
  - Showdown demo by Epic
  - Elite:Dangerous with a DK2 and HOTAS



---- ELITE ----

Load New Commander (Y/N)?

(C) Acornsoft 1984

Oculus™

Questions?

developer.oculus.com

tom.forsyth@oculus.com

Further reading

Oculus dev area:	Best Practices Guide
GDC / Connect 2014:	Developing VR Experiences with the Oculus Rift
Personal blog:	eelpi.gotdns.org/blog.wiki.html
Relevant entries:	Matrix maths and names Logging, asserts and unit tests