SSG EMEA Software enabling

Practical examples of multi-threading in games

Leigh Davies

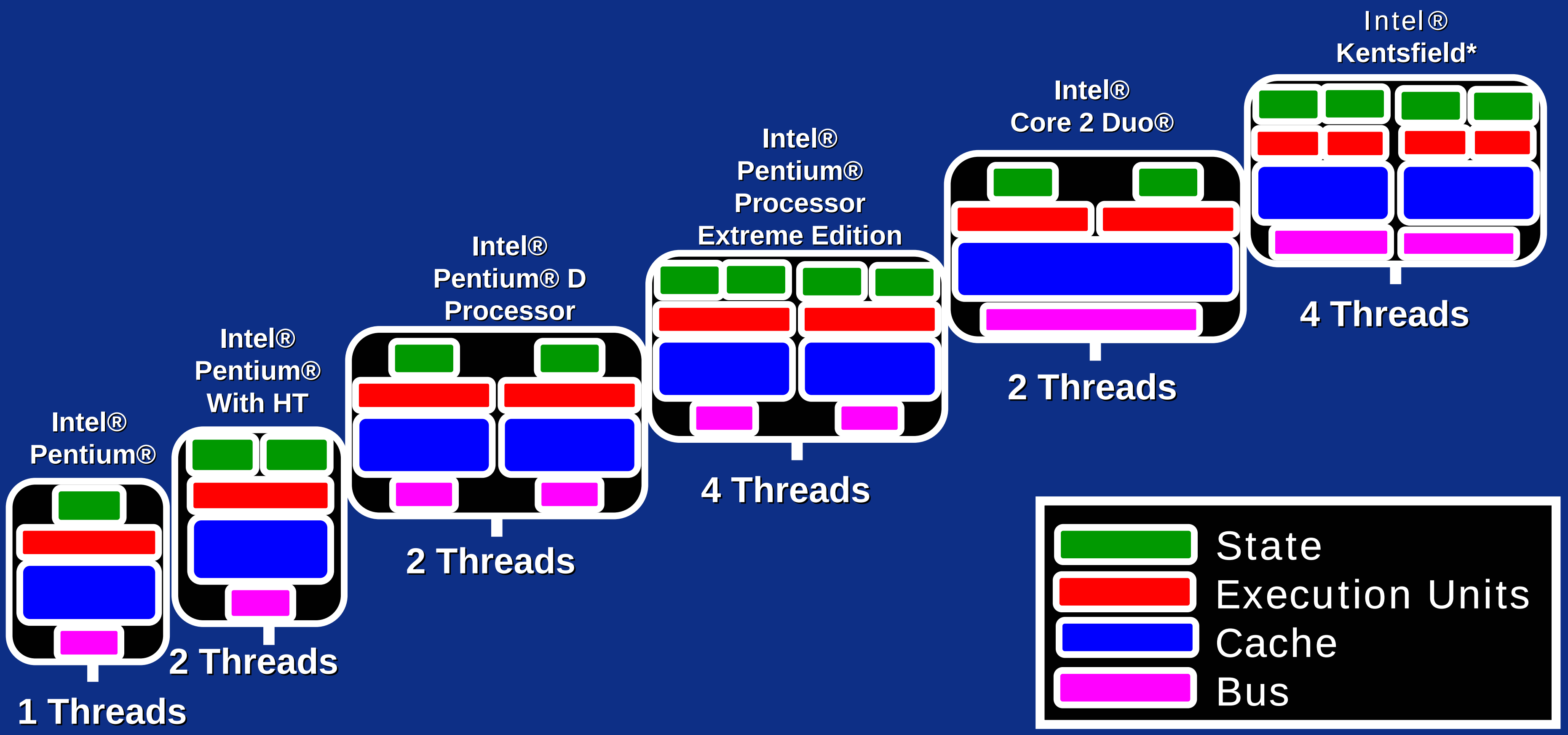
Intel

leighx.davies@intel.com

14th July 2006

PC Architecture

The PC architecture has been rapidly changing over the last 5 years



While single core performance has increased due to clock speed, increased cache and improved ILP the biggest performance increases have come from the thread level parallelism.

What does this mean for gaming?

For PC need to scale between 1-4 Cores.

Consoles need to scale to 6->8 Cores.

Don't assume No. processors == performance.

- See next slide.

Don't assume clock speed == performance.

- Core 2 Duo offers higher performance per clock than previous architectures.

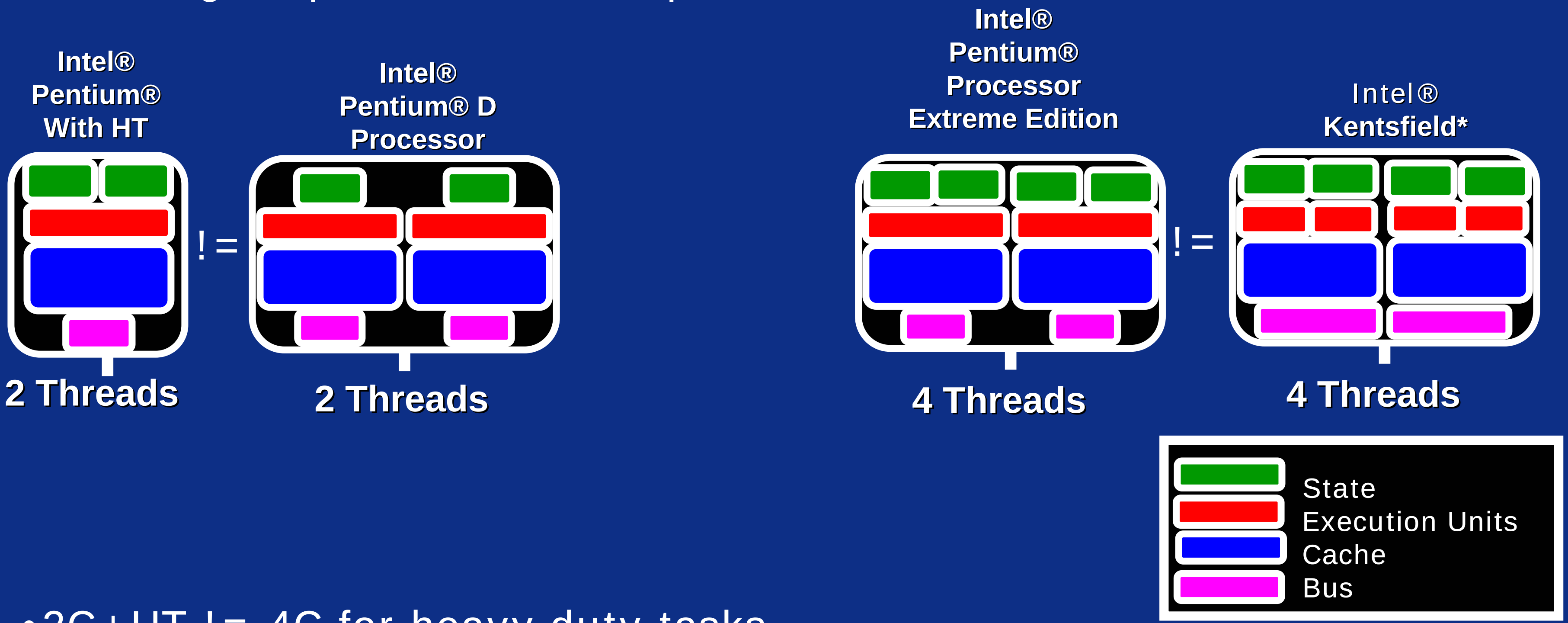
Expect to have to do load balancing on a per-architecture basis.

- Programmer controlled
- Dynamic
- Internal Benchmark

Common practice to target PC in middle of the performance curve and scale down for min spec and up for extras.
Consider Dual Core as middle target for 2007 titles.

What does this mean for gaming? Cont..

Not all logical processors are equal.

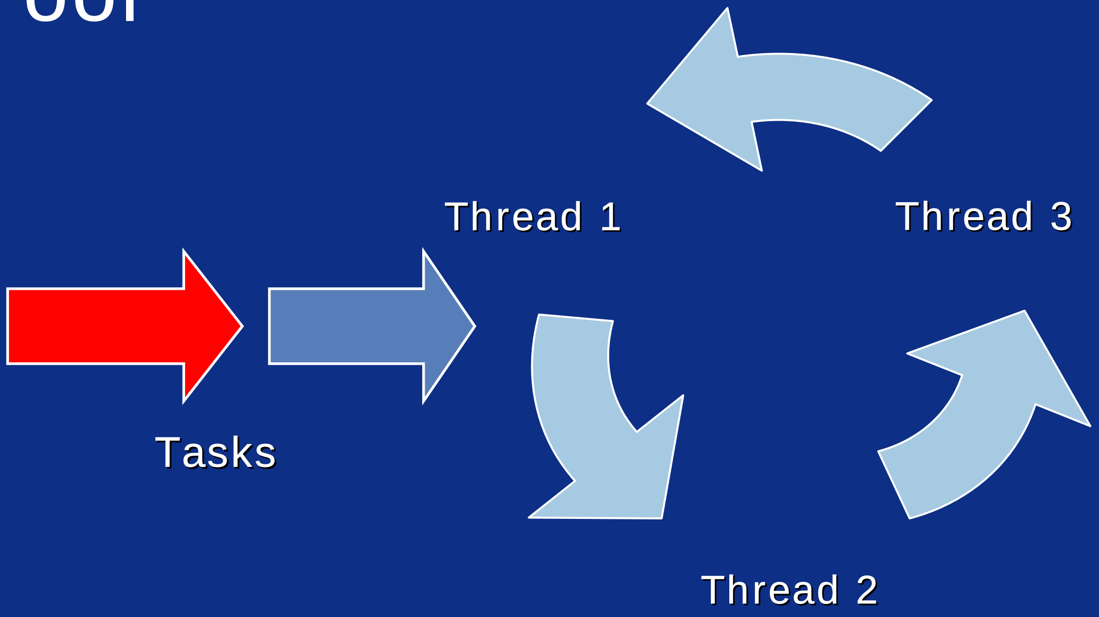


- 2C+HT != 4C for heavy duty tasks.
- Will need to detect number of Cores.
- Might need to detect shared cache.
- HT works best with threads that don’t compete for resources.
- Spin loops on HT adversely affect other thread on core.

SSG EMEA Software enabling

Game Threading Methodologies

Thread Pool



- Also known as a task, job or work queue
- Threads are only created once, put to sleep when not in use
- Try to match the number of CPU intensive threads with the number of hardware threads in the system
- Consider carefully what tasks you thread with each other, playing with different combinations can yield unexpected results

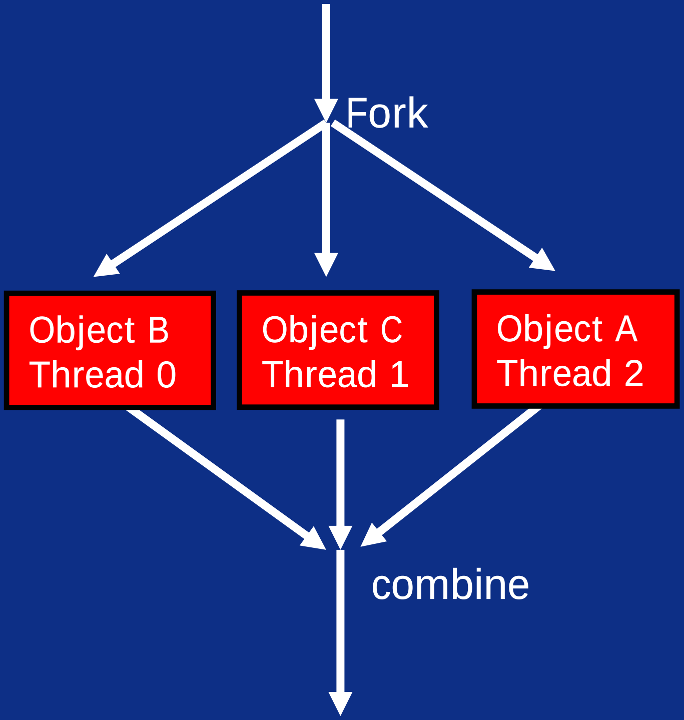
Thread Pool

- **Windows provides a generic QUEUEUSERWORKITEM.**
- **Thread pool concept is used with other threading models.**
- **Provides a central hub that asynchronous tasks can be added to as development progresses.**
- **Large tasks could block other jobs.**

Thread pools have been used in several games including Juiced!* To generate asynchronous effects systems. Or to process data loading requests.

Forking – Data Parallelism

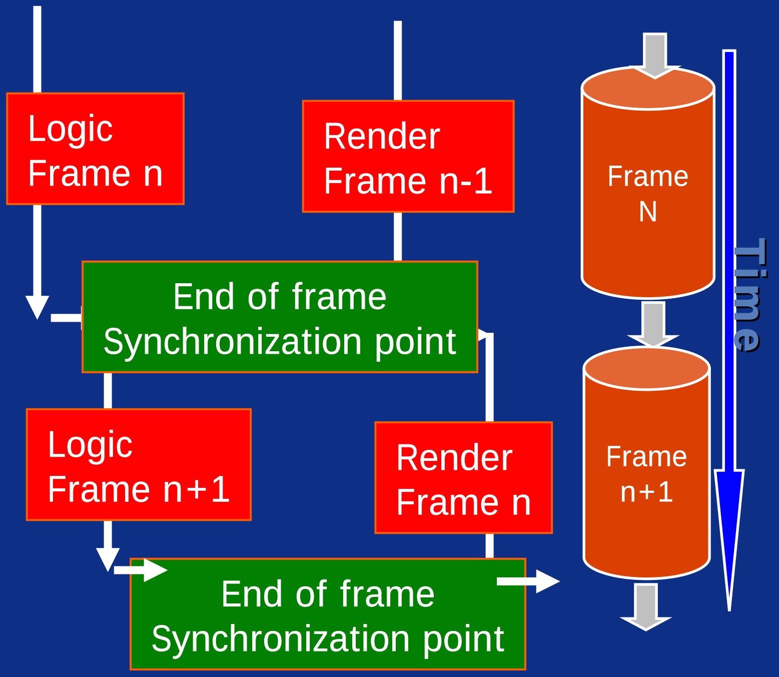
1. Same task on multiple objects in parallel.
2. Thread "forks" into multiple threads across multiple processors
3. Threads repeatedly grab pending objects.
4. When finished, threads combine back into the original thread.
5. Thread safety requirements are limited to the scope of the code within the forked section.
6. Task assignment can often be done using simple interlocked primitives:
   - Int i = InterlockedIncrement(&nextTodo);



- Used in The Movies* for shadow generation.
- Very common in Physics engines.

Pipeline / Cascade – Task Parallelism

- Subdivide problem into discrete tasks
- Solve tasks in parallel, spreading them across multiple processors.
- Inter frame and inter module optimizations
- Require up front planning
- Low synchronization overheads
- Feedback between tasks is difficult



*Other names and brands may be claimed as the property of others

Pipeline/Cascade – Task Parallelism

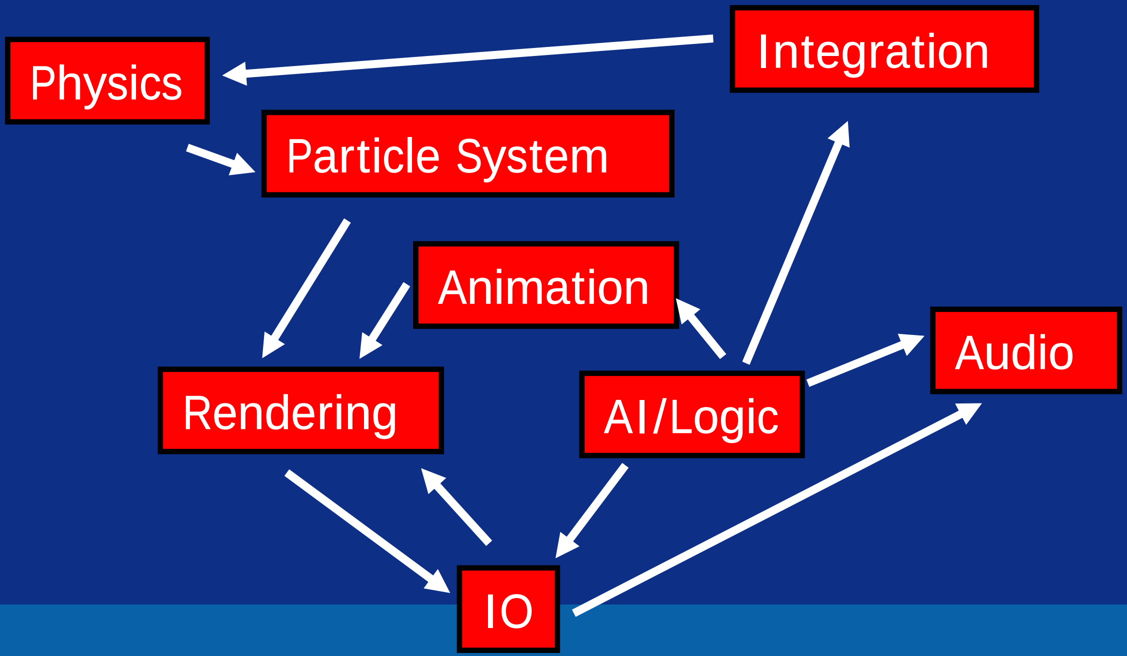
Perimeter* pipelines the graphics and AI. Synchronisation point is once per logical tick, Graphics rendered multiple times

Earthsim* pipelines the creation of new dynamic textures each frame.

Multi-threading option available in Quake 4* Beta 1.1.
Up to 27% improvement on Hyper Threading enabled systems.
Up to 87% improvement on Dual Core Intel® Pentium® Processor Extreme Edition

Work Crew – Task Parallelism

- Similar to pipeline but without explicit ordering.
- Dependencies are handled on a case by case basis.
  - i.e. particles that do not effect game play might not need to be deterministic, so they can run without explicit synchronization.
- Components without interdependencies can run asynchronously, e.g. kinematics and AI.
- Not all threads need to run constantly.



*Other names and brands may be claimed as the property of others

Work Crew – Task Parallelism

The Movies*:- Task crew used to limit frame rate spikes. Static shadow generation and path finding run asynchronously to main logic.

1. Threads only active when triggered by main game thread.
2. Main Game thread polls status of pending requests.

Falcon 4.0* uses a work crew paradigm to split render from the campaign logic.

Light weight threads handle with AI and audio.

Do I need to Synchronisation?

Question:

Imagine a thread creating a landscape based on view frustum, what happens if its not ready by end of frame?

Answer, either

1. Render thread has to wait for landscape, thereby creating load imbalance.
2. Render uses previous landscape and has artifacts around screen edge.

Possible solution.

Build frustum on enlarged FOV, only wait if new camera view falls outside previous expanded FOV.

Do I need to Synchronisation? Cont.

Problem:

- He player has just launched his large Nuke on his latest RTS game and new landscape and building data is needed but takes 3 frames to generate causing slow down.

Potential Solution:

- Calculate data in separate thread and play stock explosion particle effect that hides area of interest until data is ready.

Problem:

- The player decides to lay down 20 particle emitters all that run a complex CPU intensive chunk of code, the game can only process 10 asynchronously before the renderer is ready to draw them.

Potential Solution:

- If some of the emitters are one frame behind the others will the user notice?

Be creative and cheat to limit need to synchronise.

Potential Issues….

Common Issues:

Sleep(0) can cause problems on a HT machine.

Threads with long time outs can block out other important threads.

OS scheduler works on a fairly course gained time slice, will vary between versions of the OS, but expect 10ms+.

Tips:

Favour WaitForSingleObject over Sleep(0) for background tasks.

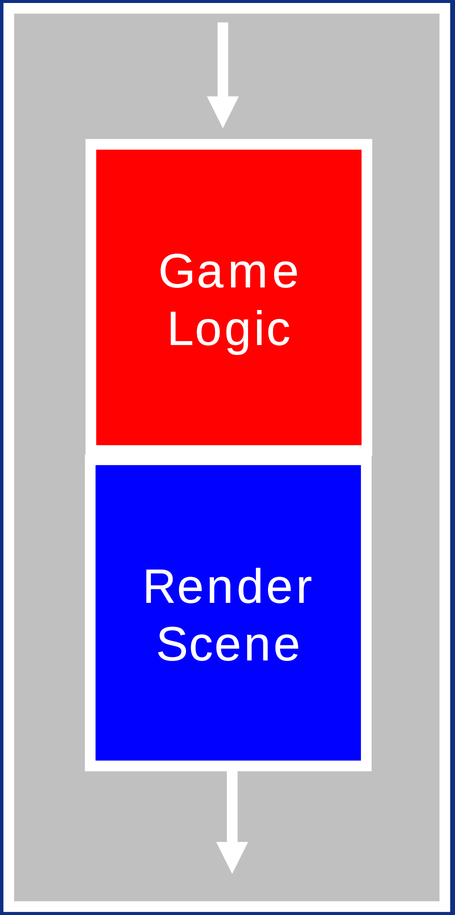
Make sure background tasks yield on regular intervals.

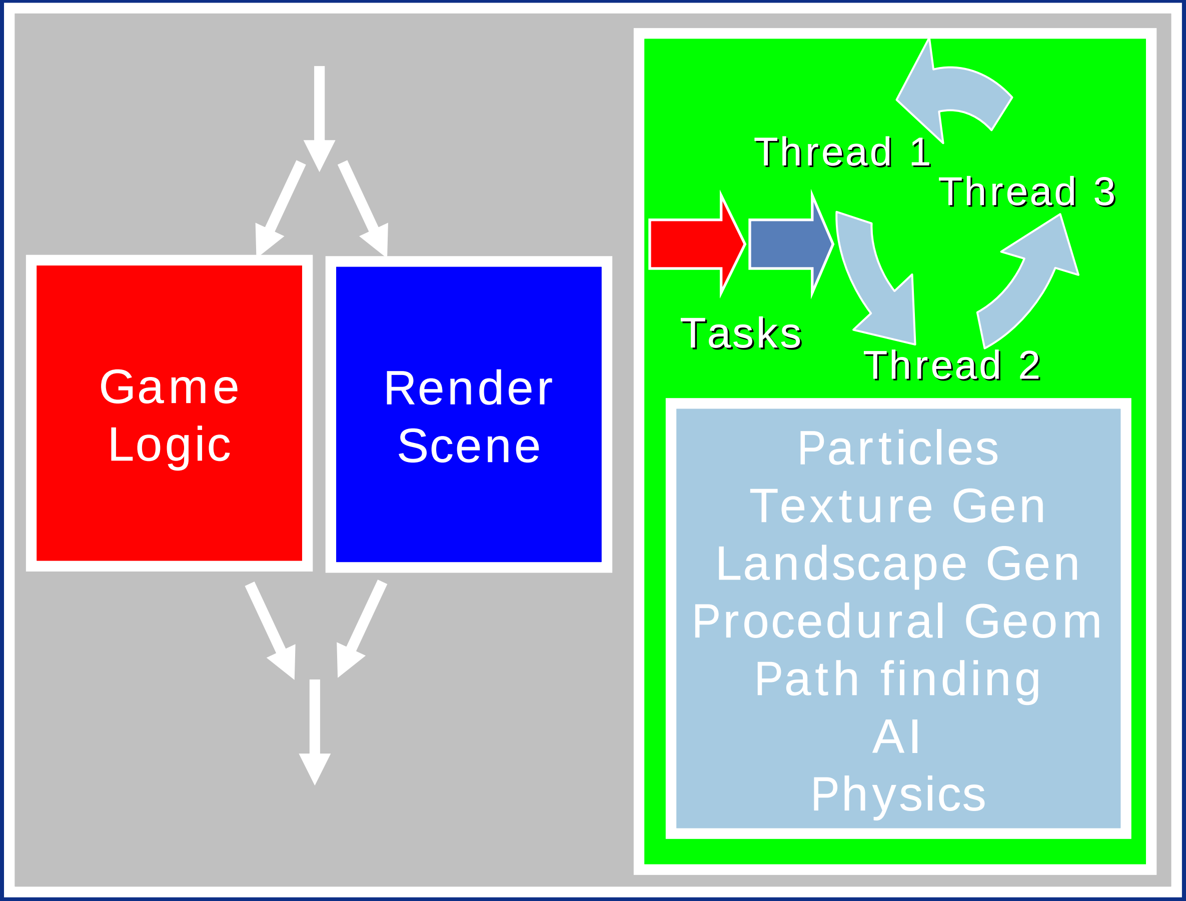
Scale number of active threads to best fit your architecture.

Profile on various PC configurations, 1C, 1C+HT, 2C, 2C+HT and 4C.

Putting it all together…

Writing an engine to scale between 1-4 Cores…





Putting it all together…

Target 2 Cores as main stream.

Easier to scale down than try and scale up from 1 to 4 cores.

On single core call logic and render sequentially to remove need to double buffer data.

Create thread pool for any remaining cores.

Assign tasks either to thread pool or call sequentially from main 2 threads if only 2 main cores.

Scale work load of tasks based on CPU speed and number of spare cores.

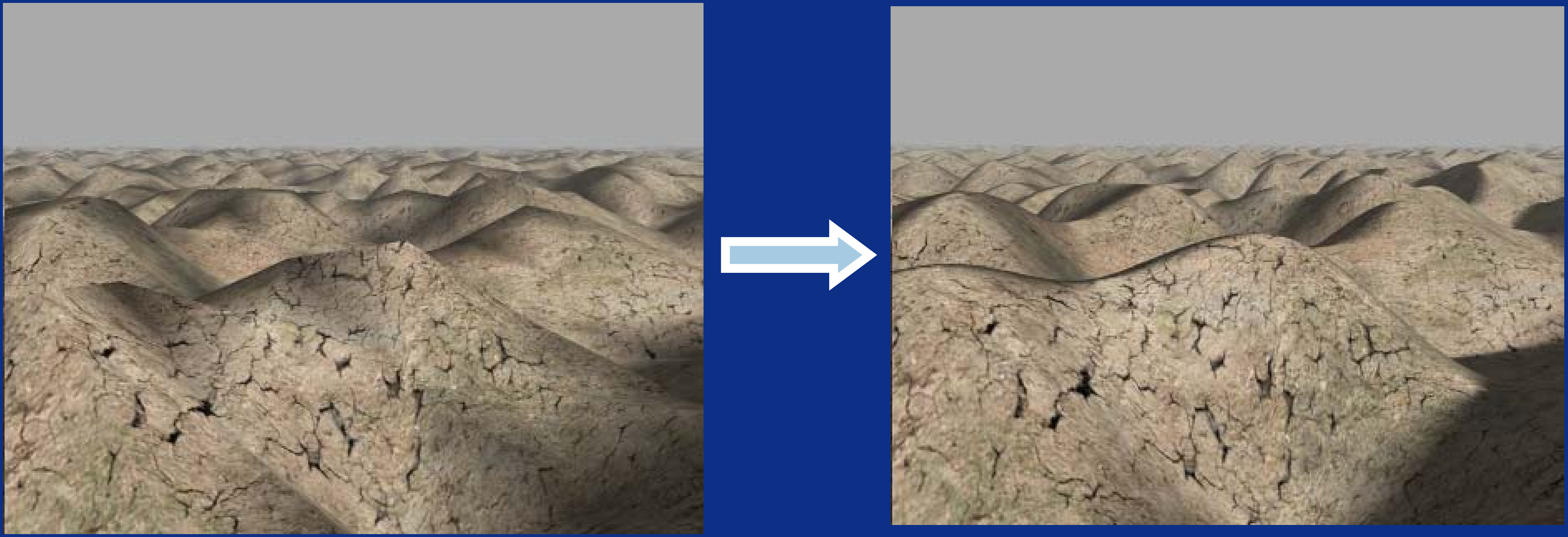
Architecture scales will with next gen consoles, allows scaling on Quad core next year….

Case Study 1

Multi-threaded Terrain Smoothing for Games

The technique in a nutshell

A coarse height field is turned into a smooth curved terrain ...



multi-threading is used to perform the smoothing in a data parallel and scalable way on MC CPUs.

Initial Motivation

- Games are CPU limited at low and medium display resolutions
  - Offload CPU intensive work to free CPU cores for higher FPS
  - Moving everything to the GPU could induce GPU limitation
- Games are GPU limited at high resolutions
  - Try to shift work from GPU to free CPU cores
  - Free GPU for more complex per pixel lighting
- Balance CPU vs GPU load to maximise game performance

Technical details

- Sparse height field defines grid
- Grid cells are replaced by bi-cubic Bezier patches
- Control points chosen for overall C^1 terrain surface
- Surface is evaluated using forward differencing
  - SSE allows to step 4 curves in parallel
  - Approximate directional derivatives
- Every cell tessellated with view dependent LOD
  - More vertices near the viewer
  - Less vertices away from the viewer
- Big triangle strip regenerated every frame
  - Results in very smooth LOD transitioning

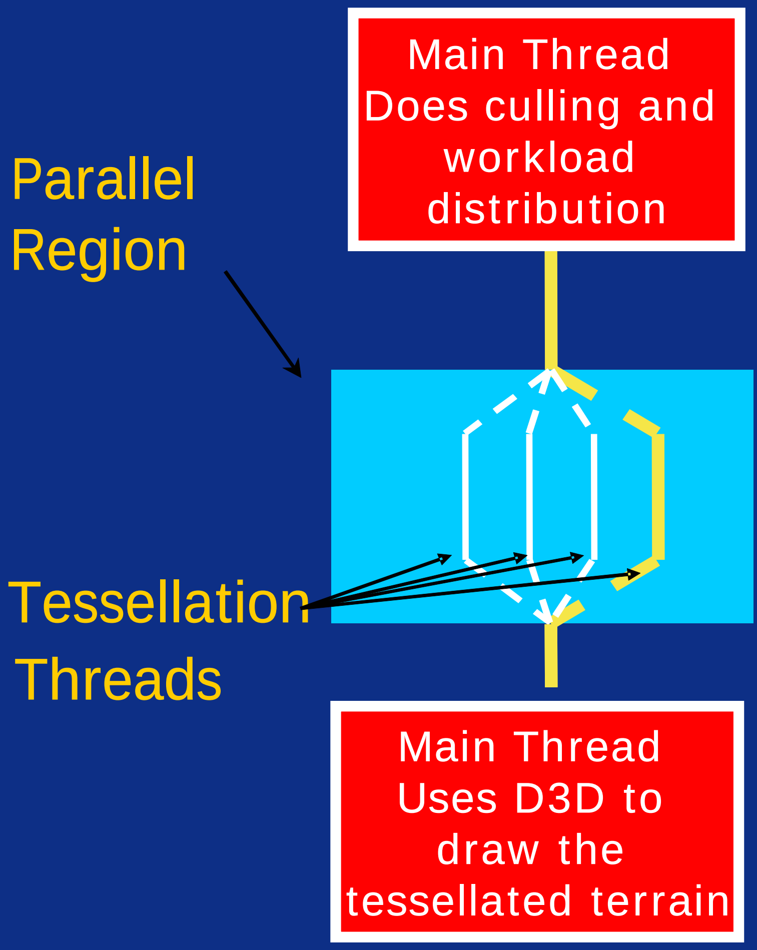
How the multi-threading is done

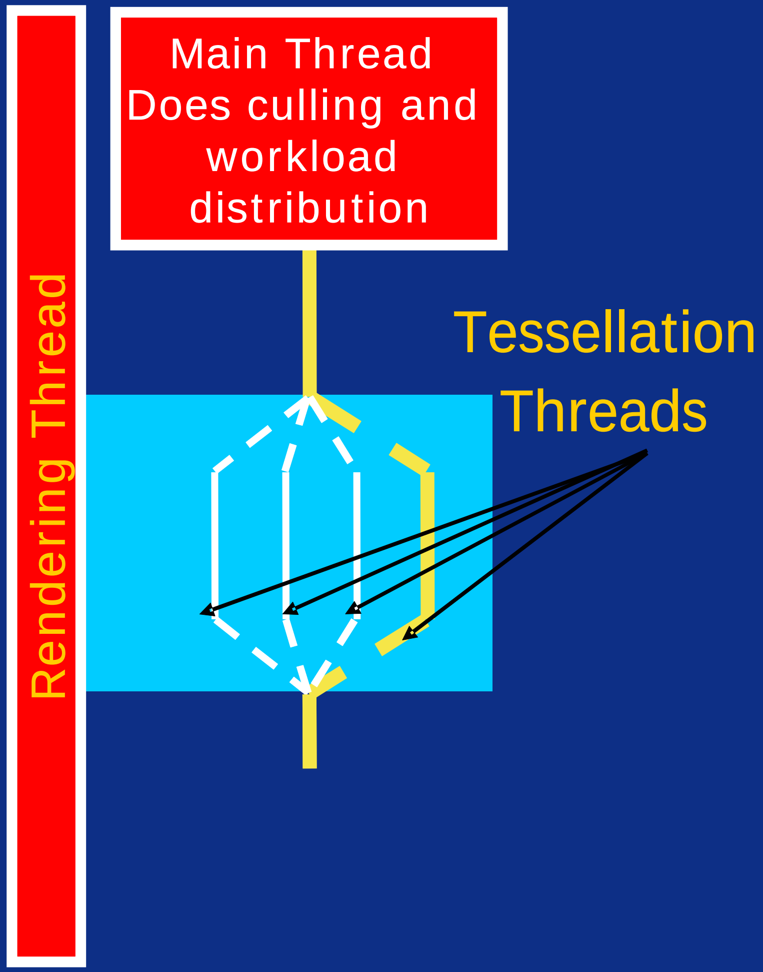
- Dynamic and workload distribution among threads
- Two code paths in the demo
  1. OpenMP™ based multi-threading
     – Speedup usually limited by render time
  2. Windows* Threading API version
     – Decouples tessellation from rendering
     – Tessellation comes for ‘free’
- Tries to not use HT cores but only real cores
  – OpenMP™ allows threads to run on every logical CPU
  – Window* version binds threads to only use cores once

*Names may belong others

OpenMP™ version

Windows* version


Main Thread Does culling and workload distribution
Parallel Region
Tessellation Threads
Main Thread Uses D3D to draw the tessellated terrain



First Results

- OpenMP™ threading
  - roughly 40 million tris/sec
    - Core Duo Laptop 2Ghz, 667 Mhz FSB, ATI X1600
  - Seems limited by GPU transfer from PCIE memory
  - Speedup ~1.5
- Windows* threading
  - Frame rate only limited by graphics card at 90 million triangles/sec
  - Speedup for syncing version ~1.3

*Names may belong others

Case Study 2

Multi-threaded physics engine
for Games

Physics in Games

Current Hot topic............

Lots of middleware solutions,

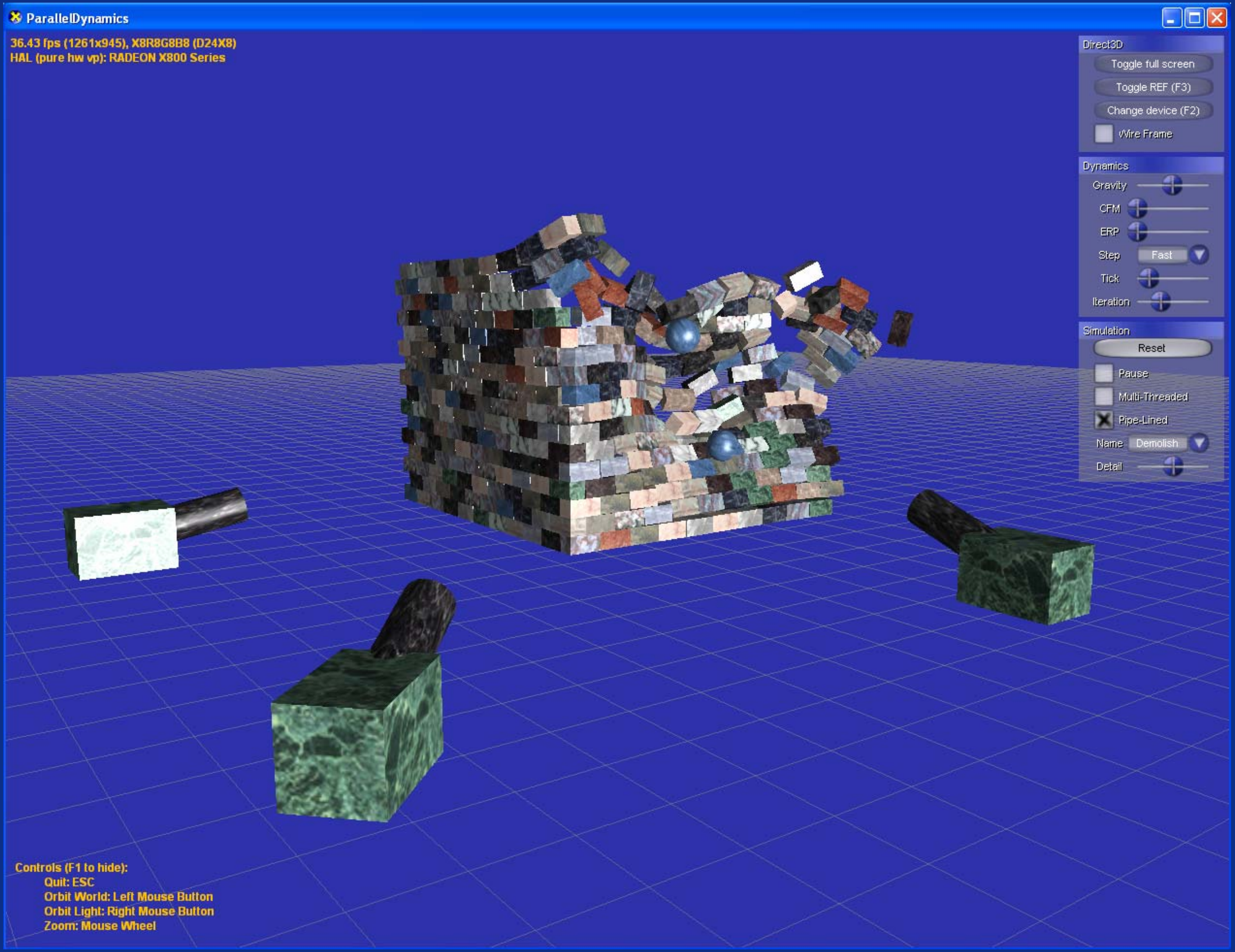
Most middleware can be configured to utilise multiple cores.

Even “Hardware solutions” place extra requirements on the CPU, care needs to be taken so that the system stays balanced.

Using the CPU has the following benefits;

1. Large installed user base, 70%+ of new PC’s are dual core.
2. Keeps system balanced, high physics load normally means high requirements on other parts of the system.
3. Easy to interface results with the main game logic.

Parallel Dynamics Sample

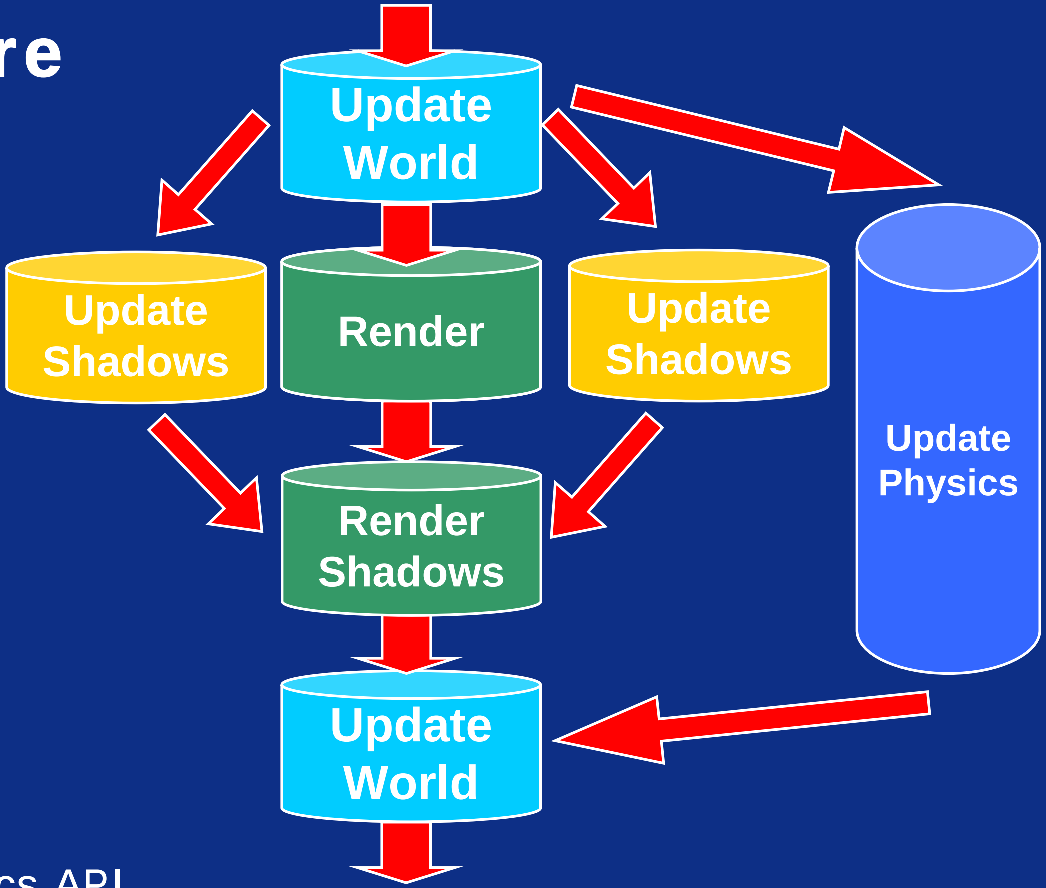


36.43 fps (1261x945), X8R8G8B8 (D24X8)
HAL (pure hw vp): RADEON X800 Series

Controls (F1 to hide):
Quit: ESC
Orbit World: Left Mouse Button
Orbit Light: Right Mouse Button
Zoom: Mouse Wheel

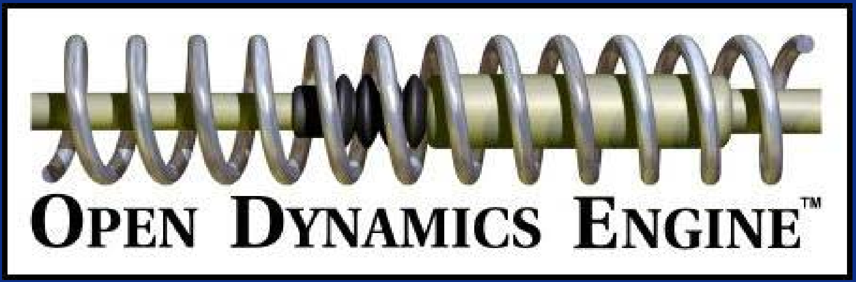
Sample Architecture

- Rendering & physics code is decoupled
- Physics can be updated at different rate to rendering – smoothes frame rate
- Fork Join technique used for Dynamic shadows
- 4 threads active at once



Physics is handled by:

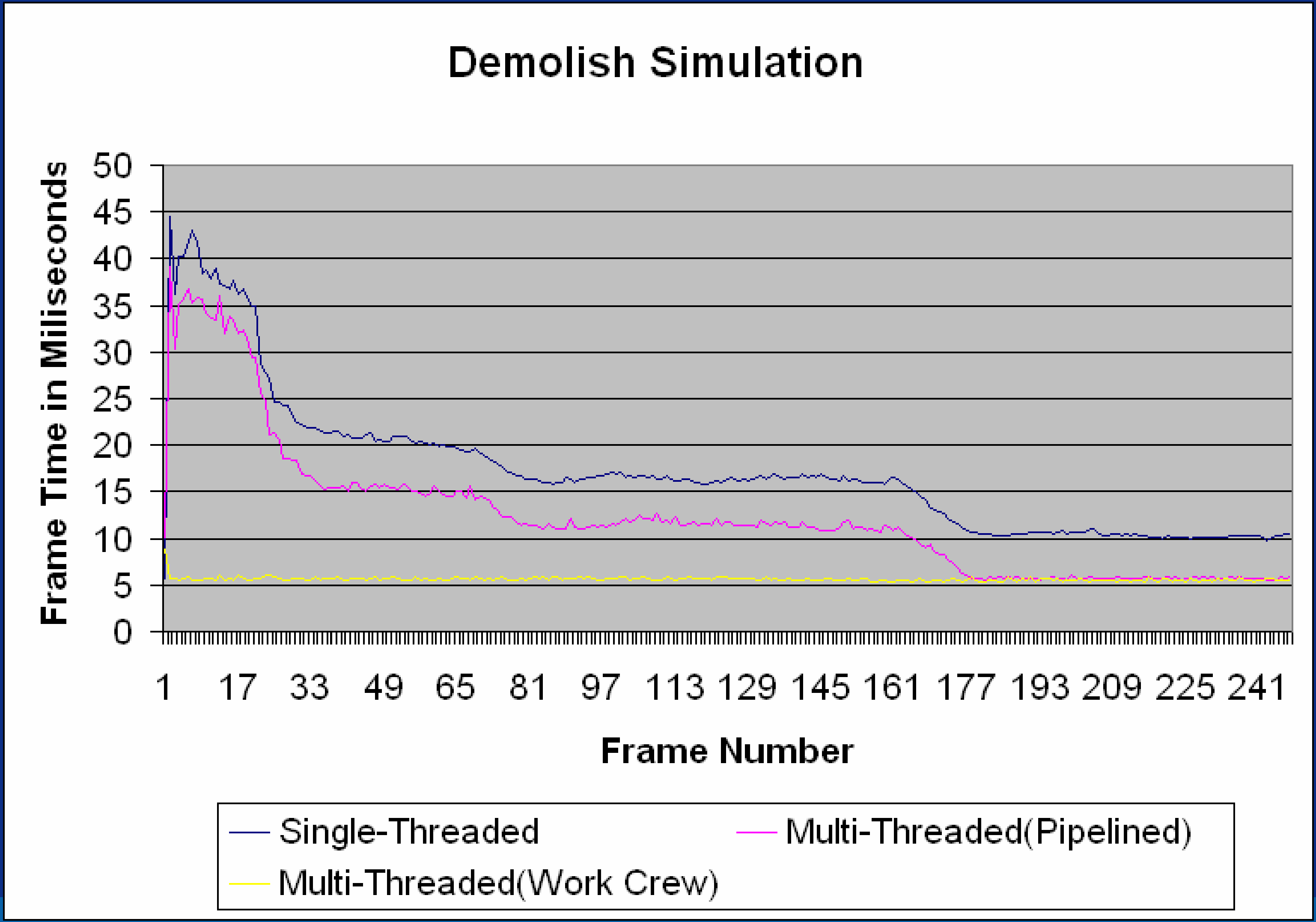
- The ODE is an open source physics API
- It has advanced joint types and collision detection with friction and gravity
- Many applications are using it, more info at http://ode.org/



Synchronisation Techniques

- Synchronise at the end of every frame
  - Also known as “Pipelined” threading paradigm
  - Very simple to implement
  - Low overhead
  - Varying physics workload can lead to peaks and troughs in frame rate
- Synchronise when physics calculation is complete
  - Also known as “Work Crew” threading paradigm
  - Not quite so simple to implement
  - Achieves a smooth frame rate because the physics workload is spread across adjacent frames

Demolish Simulation - Results



Demolish Simulation – Results Cont…

The simulation is limited by the physics workload, which starts off high and decreases over time

The pipelined technique shows a parallel speed up of around 1.3x

The load balance across the CPU intensive threads is critical to achieving parallel speed up

Notice how the work crew technique doesn't impact the frame rate at all

Work Crew Technique Hides Physics Workload

?

intel®