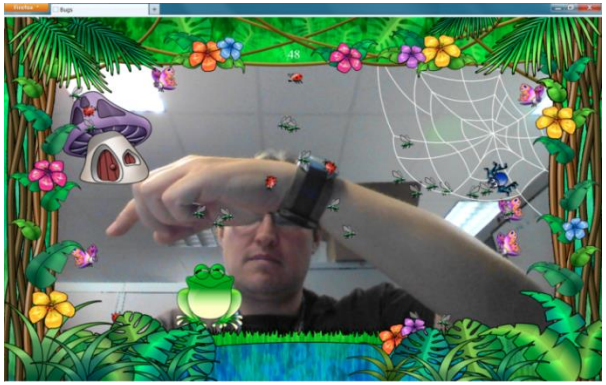
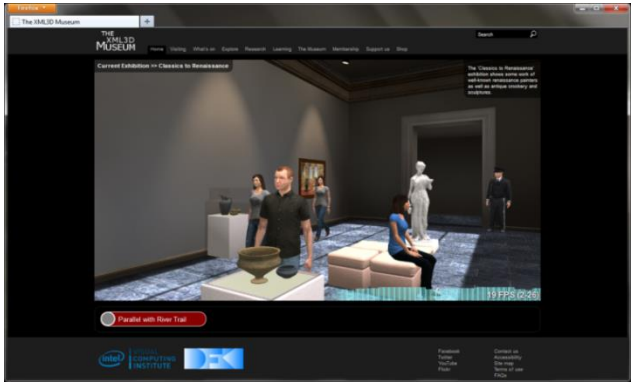
Parallel JavaScript (River Trail) Update

Visual Computing

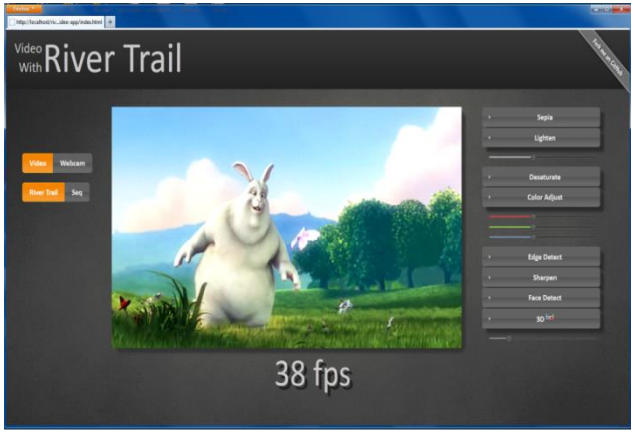


3D Animation

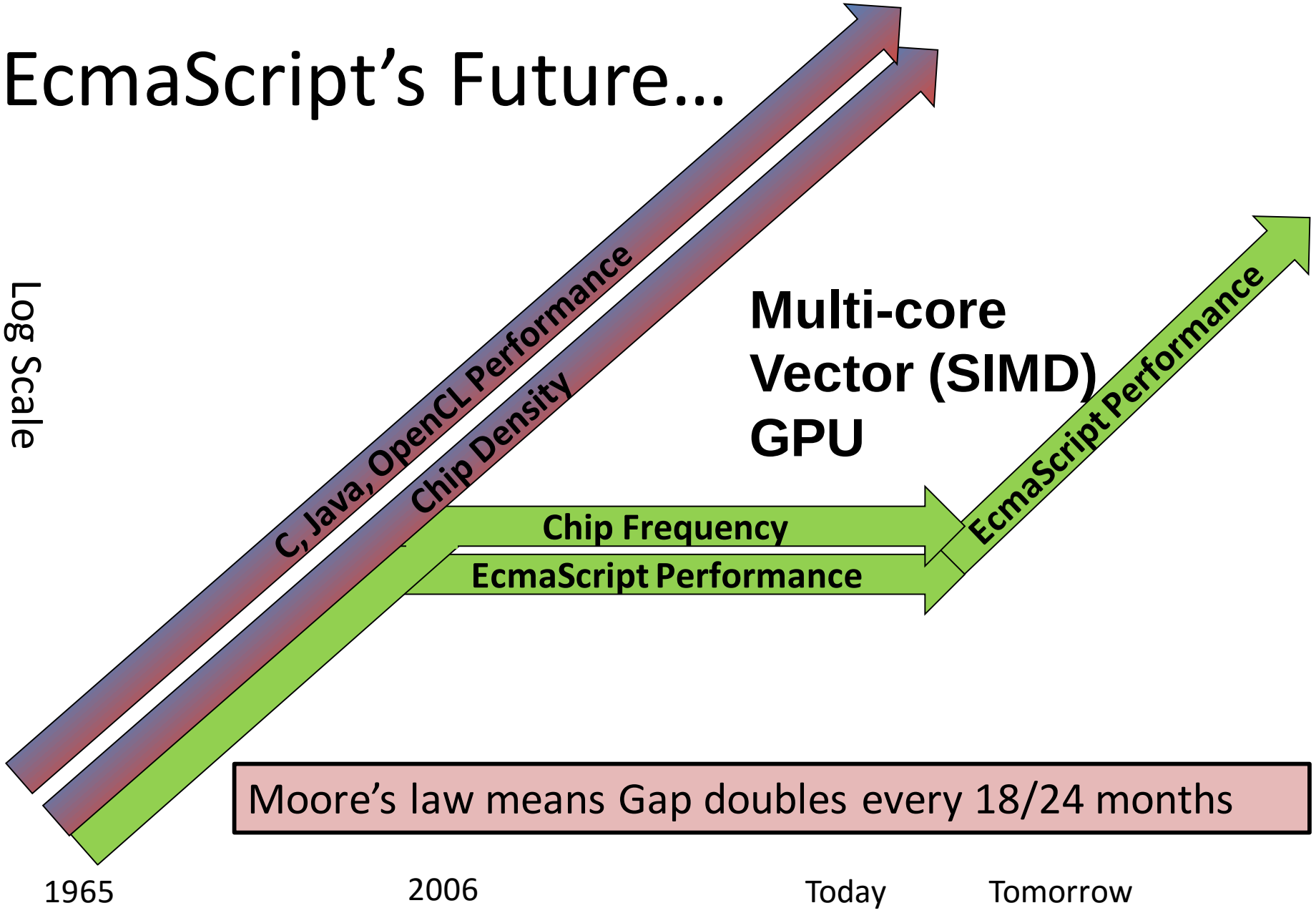


Physics-based Gaming

Video



EcmaScript's Future…



Changes over Past Year

- Added mapPar, reducePar, scanPar, filterPar, and scatterPar to Array and Binary Data aka Structured Types
  - Semantics and signatures as close as possible to Array
- Dropped the ParallelArray type
  - Binary Data functionality a good fit
  - Immutability of ParallelArray data structure dropped
- Introduced optional grainTypeSpec argument
  - Coerces arguments/results accordingly
  - Addresses needs to go from int8 to int8 formats
- Better articulation of temporal immutability

Temporal Immutability Concept

- Shared objects immutable during concurrent execution
  - No communication between concurrent entities
  - Mutable if local to computation
  - [0,1,2,3].mapPar((e, i, c)=>{temporally immutable code };);
- Fundamental to deterministic concurrency
  - Allows read access to shared objects
  - Backwardly compatibility for today's single threaded execution
  - No locks, no memory model, no transactions
- Enables important optimizations
  - Memory management friendly
  - GPU friendly
  - Preemption / scheduler friendly

mapPar

- **Array**
  - myArray.mapPar(elementalFunction)
- **Binary Data**
  - myBD.mapPar(elementalFunction, depth, grainTypeSpec)
- **Arguments**
  - elementalFunction described on next slide
  - depth (optional) the number of dimensions to iterate over (as before), default is 1, the outermost dimension.
  - grainTypeSpec (optional) specification of the underlying storage format of the result at the grain indicated by depth. If not specified the result will be Any. Values returned from the elemental function will be coerced to grainTypeSpec before being placed into the result.

mapPar Elemental Function

- function (element, index, source, outCursor)
  - **element** The element from the source.
  - **index** The index in source where element is located as well as where the result will be placed.
  - **source** The source holding the elements.
  - **outCursor** Output cursor where results can be placed. If a non-undefined value is returned outCursor will be overwritten by the returned value.
    - If outCursor is not specified the result of the function will be placed in mapPar's result at index.
- [0,1,2].mapPar((e,i,c)=>e+1, 1, int8) => [1,2,3]

buildPar

- ArrayType.buildPar(iterationSpace, elementalFunction, grainTypeSpec)
- ArrayType.buildPar(is, efun, [grain1, grain2, …, grainN])
- iterationSpace the shape, either a scalar length or array specifying shape.
- elementalFunction
  - function(index, outCursor) {outCursor.set(something);}
  - If grainTypeSpec is an array then outCursor will be an array of cursors.
  - A returned value will overwrite the primary cursor.
- grainTypeSpec (optional) the primary grainTypeSpec defaults to Any
- [grainTypeSpec1, grainTypeSpec2, …, grainTypeSpecN] (optional) If the grainTypeSpec argument is an array the result of buildPar will be an array holding arrays each of which correspond to a grainTypeSpec.

buildPar Examples

- ArrayType.buildPar(4, (i, out) => out.set(i+1), uint32)
  - // yields [2,3,4,5] grain type is uint32
- ArrayType.buildPar(4, (i, out) => out.set(i+1))
  - // yields [2,3,4,5] grain type is Any
- ArrayType.buildPar(4, (i, [out]) => out.set(i+1), [uint32])
  - // yields [[2,3,4,5]]
- ArrayType.buildPar(4, (i, out) => out[0].set(i+1), [uint32])
  - // yields [[2,3,4,5]]
- ArrayType.buildPar(4, (i, [out]) => out.set(i+1), uint32)
  - // throws
- function foo(i, [out1, out2]) {out1.set(i+i); out2.set(i*i);}
- [a, b] = ArrayType.buildPar(4, foo);
  - a is [1,2,3,4]
  - b is [1,4,9,16]

Conclusion

- Data parallelism strawman is up to date
- Asks
  - Focus on concurrency / scheduling in ES 7
    - Make sure this fits with other concurrency constructs (promises/event queues)
  - Discussion / Acceptance in ES7 process

Backup