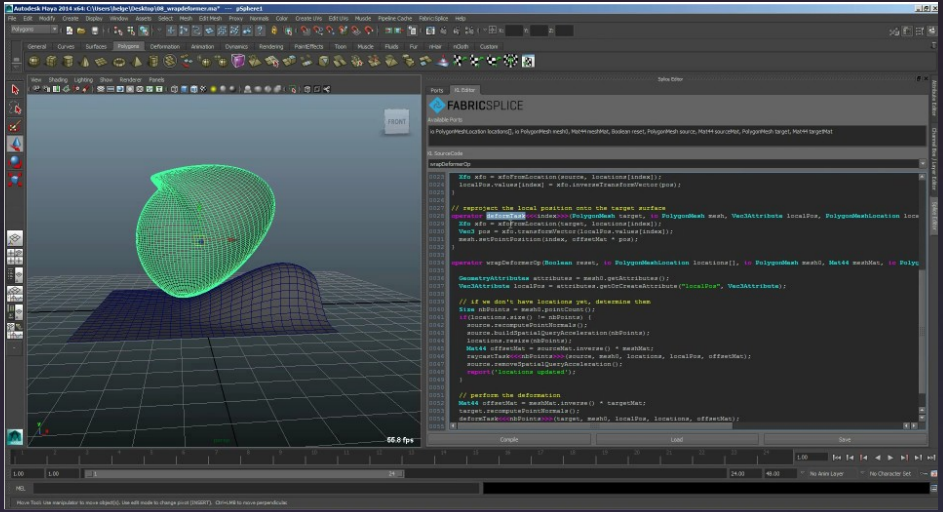
FABRIC ENGINE®

Fabric Engine and KL

LLVM for 3D Digital Content Creation



FABRICSOFTWARE

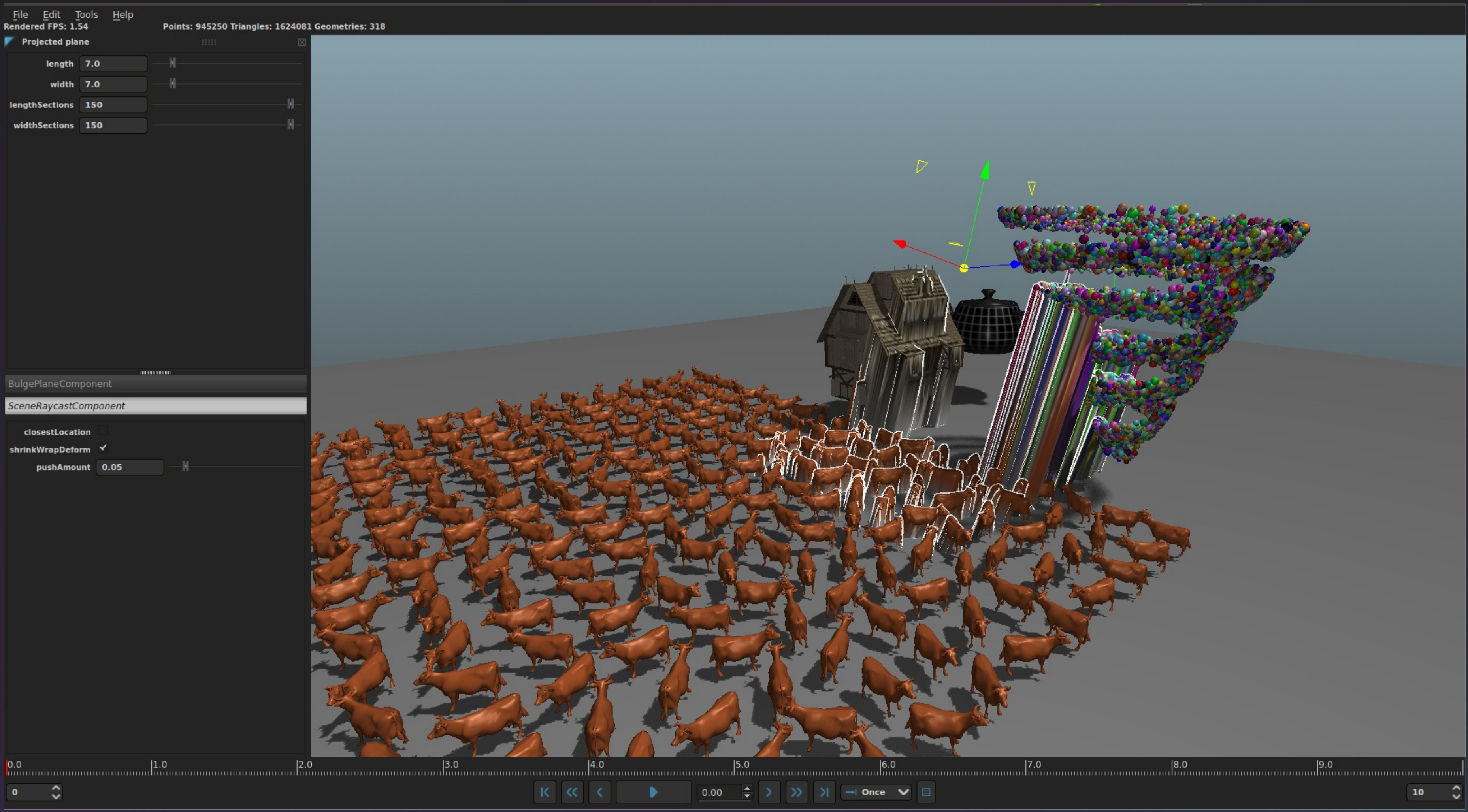
Andrew MacPherson – Core Engineer, Fabric Software Inc.

The Plan

- Overview of Fabric Engine
- Uses of LLVM within KL
- Looking forward

What is Fabric Engine?

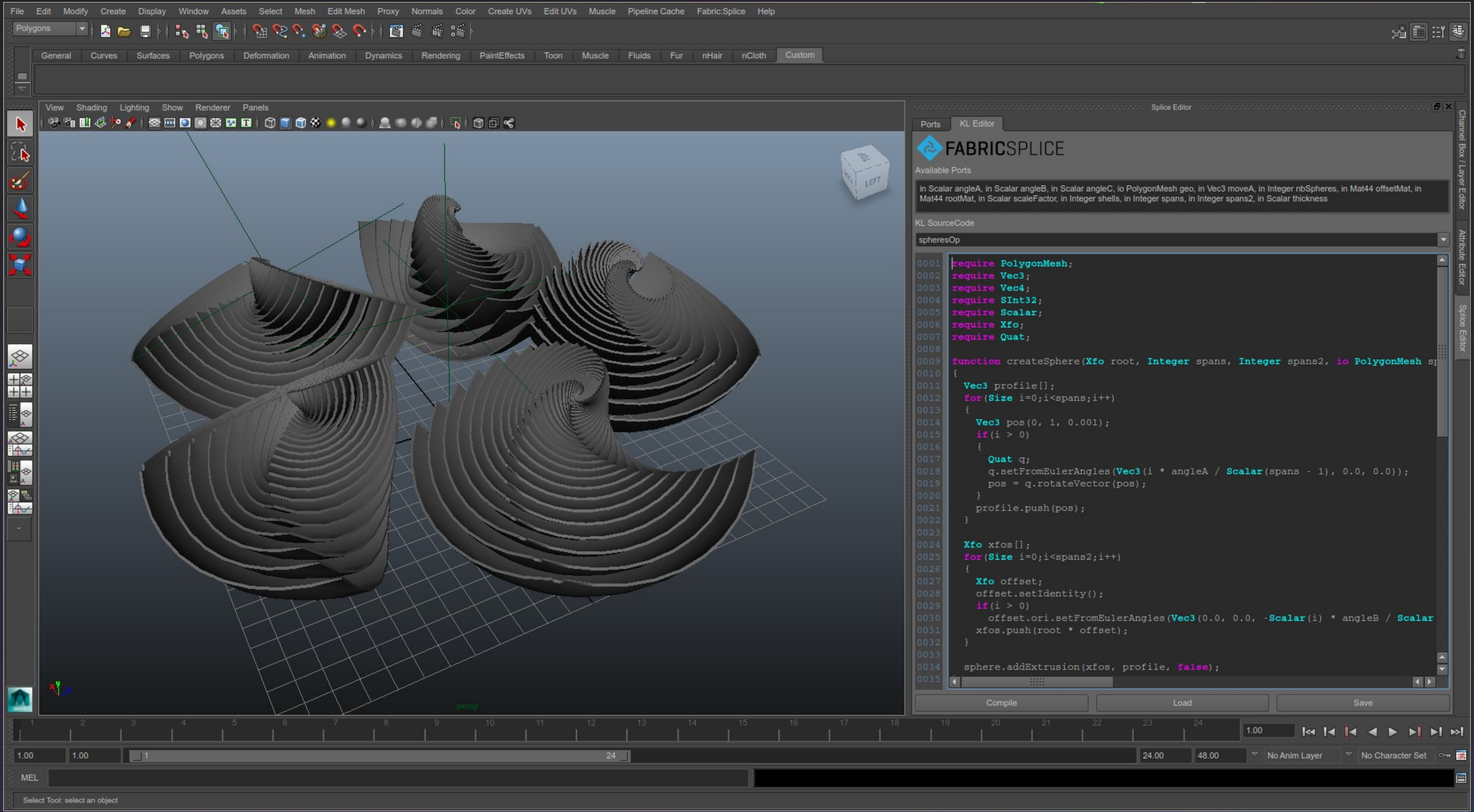
- Digital Content Creation (DCC) framework
- Standalone applications built on PySide



FABRICSOFTWARE

What is Fabric Engine?

- Digital Content Creation (DCC) framework
- Standalone application built on PySide
- Integration with existing DCC tools (Splice)



FABRICSOFTWARE

What is Fabric Engine?

- Digital Content Creation (DCC) framework
- Standalone application built on PySide
- Integration with existing DCC tools (Splice)
- In use by production studios

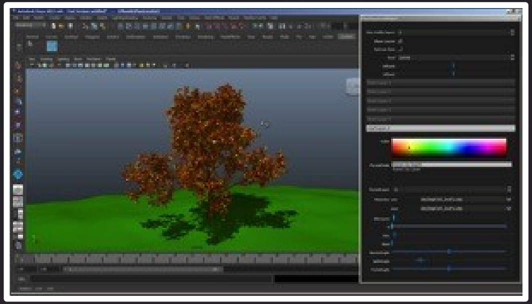
MPC

HYBRIDE
A UBISOFT® Division

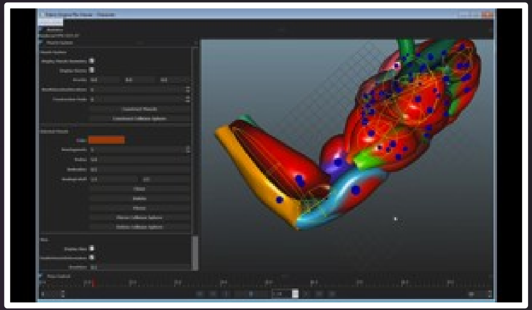
Why does Fabric Engine exist?

- Many DCC tools are old
- Closed and not configurable
- Writing plugins is hard (C++) or slow (Python)
- Development held back by software limitations

FABRICENGINE®



- KL language + dependency graph
- Host language bindings (Python/C++)
- Simple multithreading (MapReduce/PEX)
- Fully cross–platform (Windows/Linux/OSX)
- Code portable among other DCC applications



How does KL help?

- Ease of Python with performance of threaded C++
- Write once, use anywhere
- Crash–free and updateable on the fly
- Supports extensions for integration with existing libraries
- Target selection at runtime (CPU or GPU)

~/llvm-sample.kl (FabricEngine) - Sublime Text 2

llvm-sample.kl

```
require Math;

operator parallel_init<<<index>>>(io Vec3 v[], Float32 m)
{
  v[index] = Vec3(index*m, index, index*2);
}

function initialize(io Vec3 v[])
{
  parallel_init<<<v.size()>>>(v, 3.3);
}

operator entry()
{
  Vec3 v[];
  v.resize(1024);
  initialize(v);
  report(v);
}
```

INSERT MODE, Line 1, Column 14 Tab Size: 2 KL

So how does KL achieve this?

- In short: LLVM!
- MCJIT–backed
- Fabric Core compiler + scheduler
- Let's look at some specifics...

What's important for KL?

- Ease of use
- Fastest possible execution time
- Minimal memory footprint
- No significant startup delay

KL – Compilation Passes

- JIT languages slower to start than interpreted (ex. Python)
- Want maximum performance from LLVM
- Two–pass compilation
  - First unoptimized compilation pass
  - Fully optimized code generated in background

KL – Compilation Passes

- Sample case: CityAtNight.py
- 37k lines KL
- = 1.8M lines IR (pre–opt)

Method	*Startup time*
Upfront optimization	2m56s
Background optimization	0m37s

KL – Caching

- Using MCJIT ObjectCache since its introduction
- Cache both IR and objects
- Key based on hash of KL AST
- Use of IR “stubs” with cached data

~/llvm-sample.kl (FabricEngine) - Sublime Text 2

llvm-sample.kl

```
function initialize(io Vec3 v[])
{
  parallel_init<<<v.size()>>>(v, 3.3);
}
```

llvm-sample.ll

```
define private void @function.initialize.io_AS0.STVec3_VA(%STVec3_VA.Bits** nocapture %v) {
entry:
  %cpuArgsStructPtr = alloca %internal.parallel_init.stub.cpu.args.00
  %gpuArgsStructHostPtr = alloca %internal.parallel_init.stub.gpu.args
  %0 = call i32 @method.size.in_AS0.STVec3_VA(%STVec3_VA.Bits** %v)
  %cpuArgValuePtr.0 = getelementptr inbounds %internal.parallel_init.stub.cpu.args.00* %cpuArgsStructPtr, i32 0,
  store %STVec3_VA.Bits** %v, %STVec3_VA.Bits*** %cpuArgValuePtr.0
  %cpuArgValuePtr.1 = getelementptr inbounds %internal.parallel_init.stub.cpu.args.00* %cpuArgsStructPtr, i32 0,
  store float 0x400A666660000000, float* %cpuArgValuePtr.1
  %cpuArgsStructVoidPtr = bitcast %internal.parallel_init.stub.cpu.args.00* %cpuArgsStructPtr to i8*
  %nonZeroCountCond = icmp ne i32 %0, 0
  br i1 %nonZeroCountCond, label %ep.nonZeroCount, label %ep.done
```

llvm-sample.stub.ll

```
define private void @function.initialize.io_AS0.STVec3_VA(%STVec3_VA.Bits** nocapture %v) {
entry:
  unreachable
}
```

INSERT MODE, Line 1, Column 1   Tab Size: 2   KL

FABRICSOFTWARE

KL – Caching

- Sample case: CityAtNight.py
- 37k lines KL
- = 1.8M lines IR (pre–opt)

Method	*Startup time*
Upfront optimization	2m56s
Background optimization	0m37s
From cache	0m4s

KL – Linking

- Extensions export functions and methods
- Core links swappable function pointer into KL
- Same mechanism used in optimization pass
- Allows updating linked runtime code
- Explicit 'inline' modifier for extension functions

~/Fabric/SceneGraph/Native/Exts/Builtin/Math/Vec3.kl - Sublime Text 2

Vec3.kl

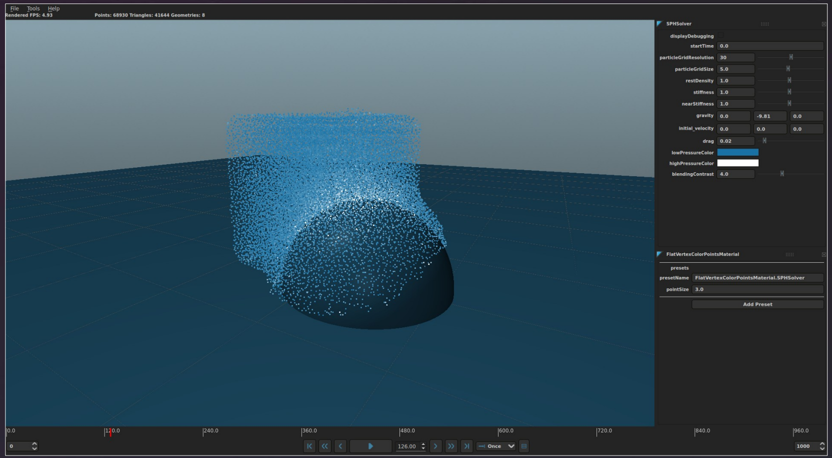
```

// equals operator
inline Boolean == (Vec3 a, Vec3 b) {
  return a.x == b.x && a.y == b.y && a.z == b.z;
}

// not equals operator
inline Boolean != (Vec3 a, Vec3 b) {
  return a.x != b.x || a.y != b.y || a.z != b.z;
}

// adds to vectors
inline Vec3 + (Vec3 a, Vec3 b) {
  return vecAdd(a, b);
}

// adds a vector to this one
inline Vec3. += (Vec3 other) {
  this = this + other;
}

// subtracts two vectors
inline Vec3 - (Vec3 a, Vec3 b) {
  return vecSub(a, b);
}

// subtracts a vector from this one
```

INSERT MODE, Line 1, Column 1 Spaces: 2 KL

KL – Linking

- Sample case: SPHSimulation.py

Method	*Startup time*	*FPS*
Inline everything	2m11s	26
Nothing inlined	0m34s	22
Selective use of 'inline'	0m35s	26



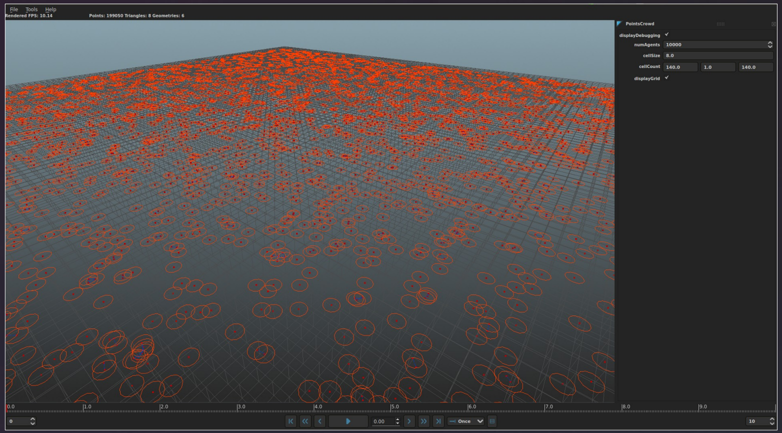
KL – Memory Use

- After compilation want minimal memory use
- LLVM 3.4: delete Module after compile
- Still need multiple ExecutionEngines

KL – Memory Use

- Sample case: Crowd.py
- With ObjectCache

Method	*RSS (MB)*
Full IR + no removeModule	797
Stub IR + no removeModule	428
Full IR + removeModule	367
Stub IR + removeModule	356
Shared ExecutionEngine	296



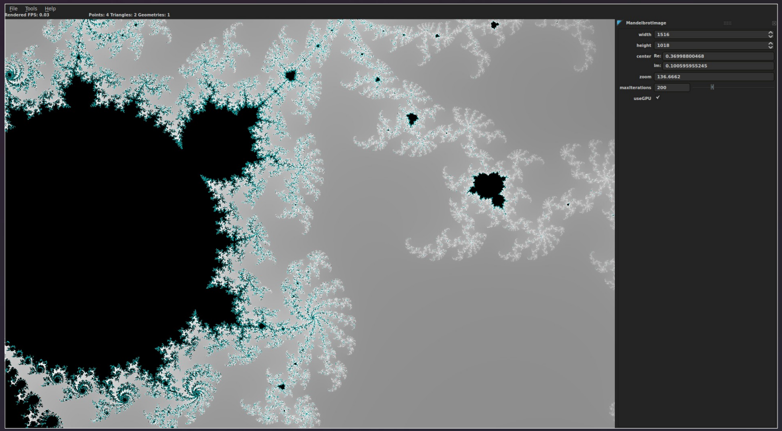
KL – GPU Compute

- KL code run without modification on CPU or GPU
- AMD HSA hardware shared memory
- Nvidia Cuda 6 “shared memory” via driver
- Speedup varies by application and hardware but up to 10x faster
- First release coming in May 2014

KL – GPU Compute

- Sample case: Mandelbrot.py
- Standard desktop hardware

Target	*FPS*
Intel Core i7–3770k @ 3.50GHz	3.7
NVIDIA Quadro K5000	23.5



~/Fabric/working/SceneGraph/Python/Apps/Math/Mandelbrot/Mandelbrot.kl (FabricEngine) - Sublime Te

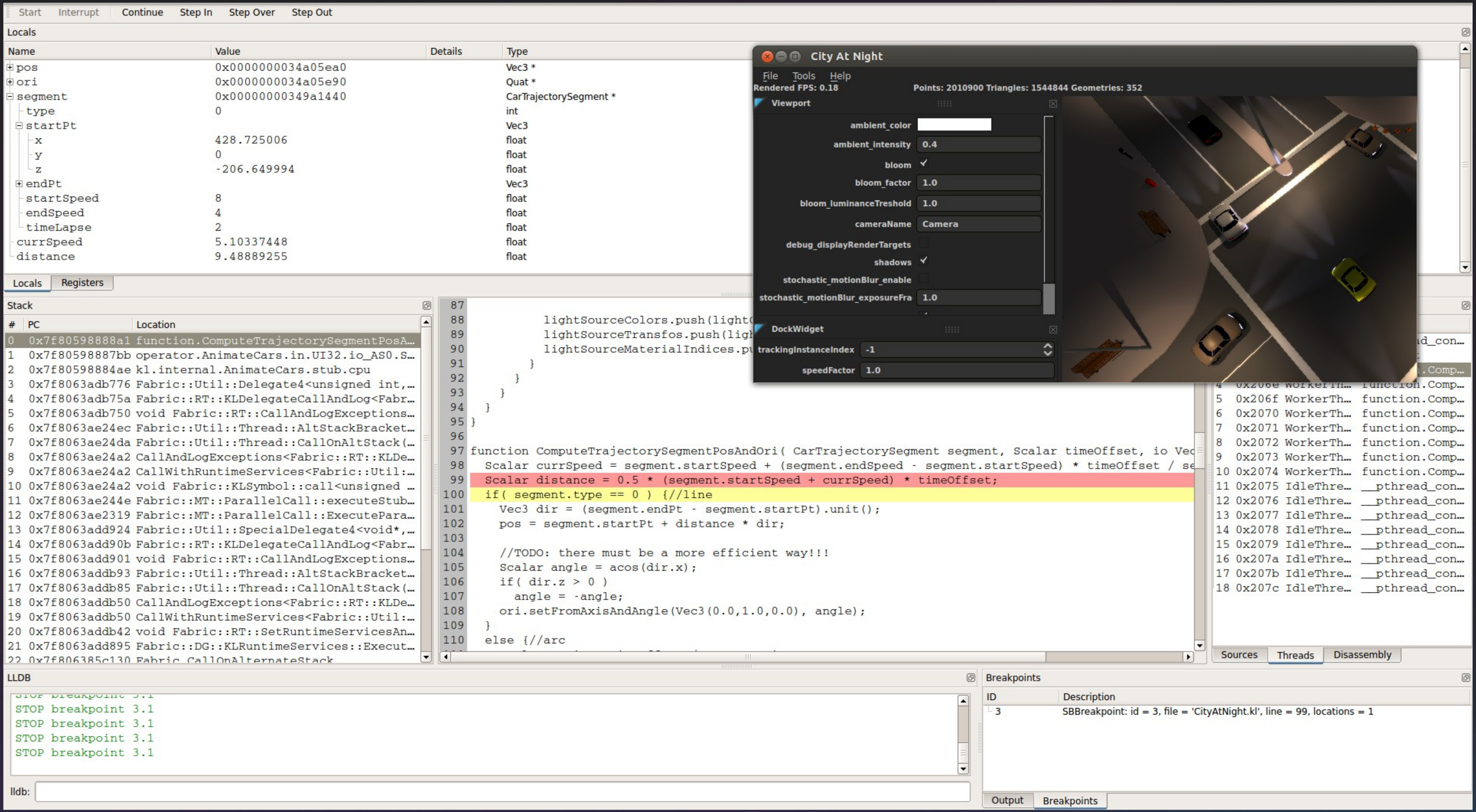
Mandelbrot.kl

```
operator computePixels(
  Size cols,
  Size rows,
  io Image2DRGB image,
  Complex64 center,
  Float64 zoom,
  Size maxIterations,
  Boolean useGPU
){
  image.resize(cols, rows);
  computePixel<<<image.pixels.size()@useGPU>>>(
    cols,
    rows,
    image.pixels,
    center,
    zoom,
    maxIterations
  );
  image.incrementVersion();
}
```

INSERT MODE, Line 1, Column 1 Spaces: 2 KL

KL – Debugging

- Dwarf info via LLVM DIBuilder
- LLDB JIT support
- Breakpoints, threads, variable inspection, etc.
- Python + PySide LLDB front–end



FABRICSOFTWARE

Looking ahead

- Further reducing MCJIT memory footprint
- Better error handling in out–of–memory scenarios
- LLDB on Windows
- Clang on Windows
- GPU debugging?

FABRIC ENGINE®

http://fabricengine.com/

andrew.macpherson@fabricengine.com