Indigo Shader Language and Winter

- By Nicholas Chapman, Managing Director, Glare Technologies Limited
- nick@indigorenderer.com  http://www.indigorenderer.com/
- Presented at European LLVM User Group Meeting, Sep 16\textsuperscript{th} 2011, London, UK
Talk contents

- Background – Indigo Renderer
- Indigo Shader Language
- The Winter programming language
Indigo Renderer

- Photorealistic rendering software
- An unbiased ray tracer
- Main product of Glare Technologies Limited
Indigo Shader Language

- Embedded language used for customising material appearance
- User can write a small program to control e.g. the colour of a material
- Example:

```python
def eval(vec3 pos) vec3 :
    vec3(sin(pos.z * 1000.0))
```
Indigo Shader Language

- Statically typed
- Functional
- Safe (no pointers etc.)
- JIT compiled using LLVM
- Efficient execution using vector types allowing SSE codegen.
- Performance similar to SSE-optimised C++
- Compiled shaders can be run concurrently in different threads
Demo!

- Demo of Indigo Renderer with realtime shader editing
Introducing Winter

- Winter is a more general purpose programmable language in development
- Descendent of Indigo Shader Language
- Primary design goals:
  - High Level
  - Functional
  - Safe (well defined semantics, no pointers etc..)
  - High performance for engineering, scientific computing, and computer graphics
  - Implicit Parallelism
  - Embeddable (in a host C++ program)
- Backends:
  - LLVM JIT for x86, x64
  - GPU: OpenCL or CUDA
Simple Program

- Simple Program:

```python
def main(float x, float y) float :
    x + y
```
def f(float x) float :
  let
    z = 2.0
    y = 3.0
  in
    y + z
def main() float : f(0.0)
Structures and operator overloading

```python
struct s { float x, float y }

def op_add(s a, s b) : s(a.x + b.x, a.y + b.y)

def main() float : x(s(1, 2) + s(3, 4))
```
Higher order functions

def compose(function<float, float> f, function<float, float> g) :
    \( float x) : f(g(x))

def addOne(float x) : x + 1.0
def mulByTwo(float x) : x * 2.0
def main() float :
    let
        z = compose(addOne, mulByTwo)
in
    z(1.0)
Implicit Parallelism

The basic idea here is that map and fold (reduce) are elementary functions that divide the problem into chunks when possible, and distribute across computational units. (CPU cores, GPUs, network?)

WIP!

```python
def f(float x) float : pow(x, 1 / 2.2)

def main(array<T> a) : array<T>
    map(f, a)
```
The future of Winter

- Implement parallelising backend
- Implement GPU backends
- I hope to open source Winter at some point.
Thank you for your time!