GLSL Support in Mesa

Current state
- What parts are good
- What parts are not so good
- What parts are really miserable

Where we want to be
- What is the ideal world

How we're getting there
- Work being done now
- Work being done later
Many shaders work on real hardware
  - We're shipping on i915 and i965 for some time
**Good**

Language extensions allow many language intrinsics to be written in GLSL

- **`__constructor`** allows constructor functions
- **`__operator`** allows operator overloading to decompose complex operators (i.e., matrix multiplications) into simpler operations
  - Dramatically simplifies type checking
- **`__asm`** allows in-line assembly to implement operators
  - Here assembly means ARB_fragment_program
Not So Good

Assumes only hardware type is \texttt{vec4}

- Each integer operation is converted to a floating point operation followed by a rounding operation very early

Language intrinsics are recompiled at run-time

- Tokenized off-line

Missing some GLSL 1.20 language features

- GLSL 1.30 support not even started
Really Bad

One-off parser generator
- Parser generator code isn't in main source tree
Linker can't support multiple compilation units on the same shader target
- Various other bugs related to this missing feature

ARB_fragment_program is the IR
- Really bad fit for lots of real hardware
Infrastructure We Want

Sensible parser infrastructure
- Allow others fix bugs or add language features

Back-end independent IR
- Most real hardware doesn't look much like ARB_fragment_program
- DirectX uses our current approach for HLSL, and drivers have to de-compile and recompile the assembly...fail
Language Features We Want

A compliant linker
  - Multiple compilation units at each shader target has been a required feature since day-1

Real integer support
  - Allow hardware with real integers to use them

Full GLSL 1.20 and 1.30 support
  - Unsigned integers and integer bit twiddling
  - switch-statements
Work Being Done Now

GLSL work being done for the last month or so...around various travel
Parser Infrastructure

Parser re-written using flex and bison

- All 1.20 core language features supported
- Preprocessor still to be implemented
- Generates a tree-based AST that looks a lot like what GCC uses
Back-End Independent IR

AST is converted to middle-level intermediate representation (MIR)

- Work-in-progress
- Encodes a reduced subset of language features
  - "Flattened" expressions, for-loops, if-statements
Back-End Independent IR

MIR is translated to existing IR

- Work-in-progress...mostly hand-waving at this point
- Possible to generate hardware instructions directly from MIR without ever touching existing IR
Back-End Independent IR
Back-End Independent IR

Language intrinsics compiled to MIR off-line

- Fake link-phase combines intrinsic MIR to translation unit being compiled
Compliant Linker

Linker operates on MIR
- Code is generation happens after linking
- Additional in-lining, constant propagation, etc. occurs after linking
Work Not Being Done Now

GLSL infrastructure work will continue on into next year...at least
GLSL 1.30

Some new language features aren't being implemented yet

- New integer operators (e.g. bit twiddling)
- switch-statements
Additional Shader Stages

Geometry shaders not yet supported

- Additional shader stages may be added soon, these aren't supported yet either
Code-Generator Generator

Real multiple-target compilers use code-generator generators

- Don't want to generate machine code from existing low-level IR...it's a bad fit for most hardware
- Hand-writing code generators is tedious
Assembly Shaders to MIR

Support for additional assembly extensions

- Direct implementation is not a good fit on some hardware
- Some apps (e.g. those that use Cg) prefer to use assembly shaders
- `GL_NV_vertex_program3` on R500 hardware for free feels like winning
Assembly Shaders to MIR

GLSL → MIR → ARB_fp-like IR → Hardware

ARB_fp, etc. → MIR
Assembly Shaders to MIR

GLSL

ARB_fp, etc.

MIR

SSA, other low-level IR

Hardware
Fin