C++ variant vs Rust enum

Richard Shepherd

- C++ developer at ESRI R&D Cardiff

Constrained polymorphism

- Traditional OO polymorphism
  - type hierarchy
  - dynamic typing
  - open for expansion
  - group functionality by type
- templates
  - static polymorphism
  - duck typing
  - group functionality by type
- variant
  - known, small set of types
  - closed to expansion
  - group functionality by aspect

Geometry transformations

- Needed to model a small set of families of transformations of curves and polygons
    - No change
    - Simple move
    - Rotation
    - Scale
    - Reshaping
- No others
- Make sure each case is covered
- **std::variant**

Switch

Multiple ways to choose between the types.

- Clarity
- Correctness
- Debugability
- Efficiency

C++ Examples

Adapted the types to a simple, measurable calculation.

Time the different approaches.

C++ conclusions

- difficult to create variant types scoped to the variant
- various library supported syntaxes verbose and intricate
- combining `switch` and `get_if` increases the decision-making work
- good to see 'offical' techniques before well

Rust equivalent

- Much less experience
- Build a simpler program to compare performance
- Build comparable enum example

Simple performance comparison

- non-linear, iterative float calculation
- pretty similar results
- identical floating point calculations

Rust enum conclusions

- easy to create variant types scoped to the variant
    - but more difficult to compose from existing types (need to wrap)
- language supported switch trivial to implement

Next steps

- Constraint based types and programming
- Rust type traits

Any ideas/questions: richard@shepherd.ws