



# An Introduction to the Scout Programming Language

Patrick M<sup>c</sup>Cormick

Advanced Computing Lab

Los Alamos National Laboratory



# Scout Team

---

- **LANL**
  - Pat M<sup>c</sup>Cormick, Jeff Inman, Jim Ahrens
- **UC Davis**
  - Adam Moershell, John Owens
- **Utah**
  - Greg Roth, Chuck Hansen
- **Funding by:**
  - DOE Office of Science
  - LANL Directed Research

# Goals

---

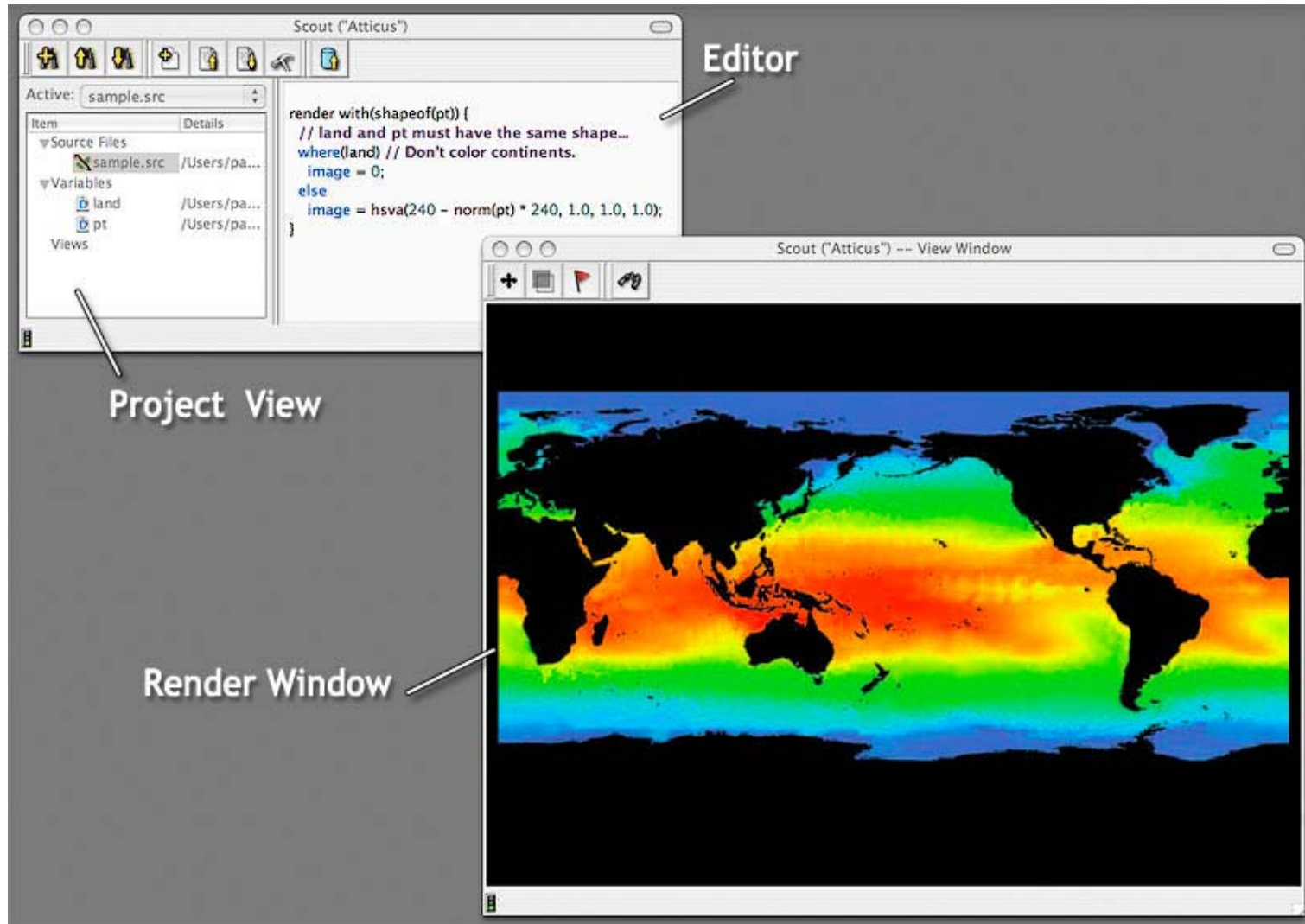
- A GPGPU language to help with both data analysis and visualization
  - Often viewed as two separate tasks... Not good!
- Support for multiple visualization techniques
  - Last year's paper focused on volume rendering...

# Scout Overview

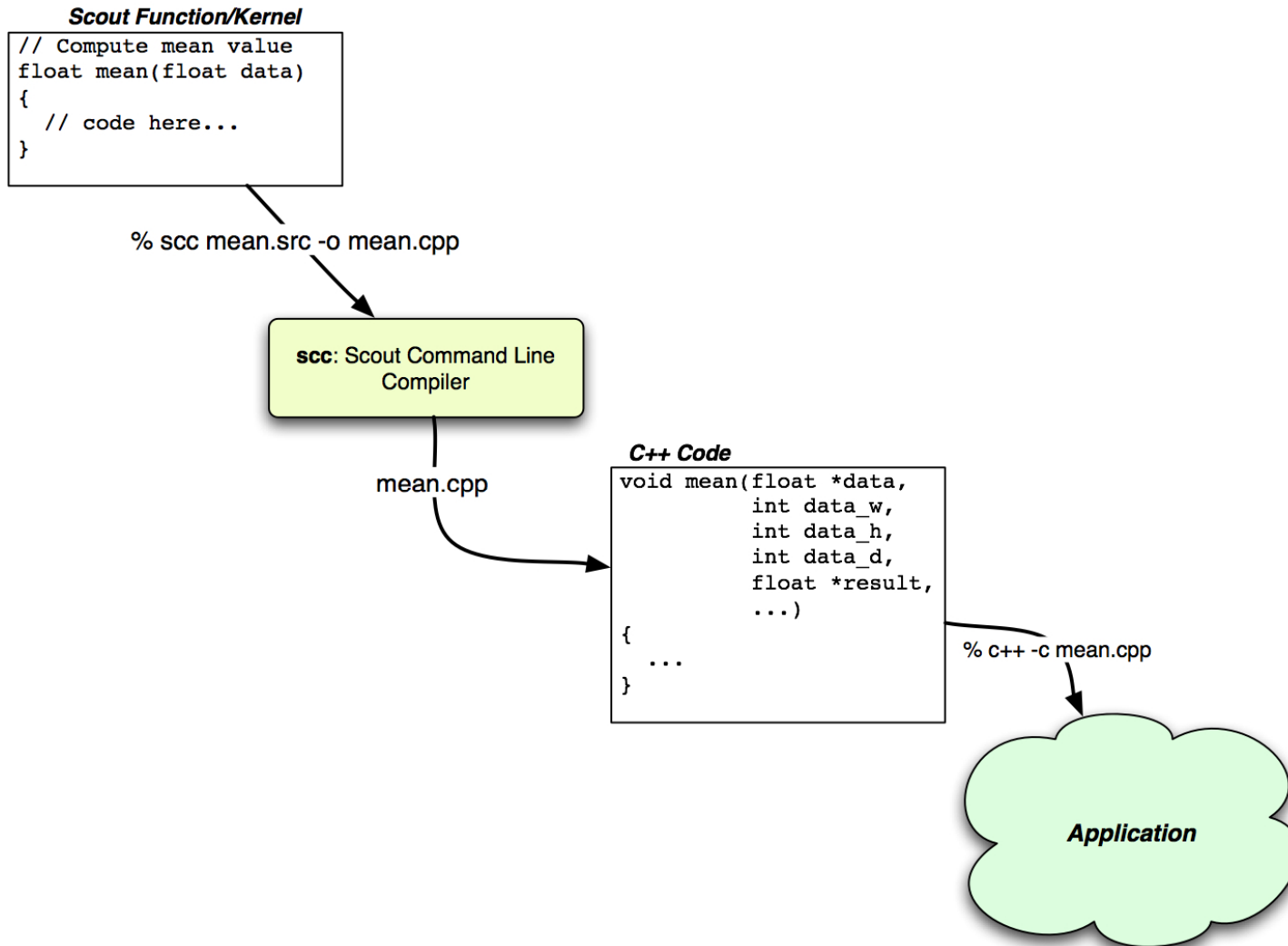
---

- **Data parallel programming model**
  - C\*-like (from Thinking Machines Inc.)
- **Language support for:**
  - Data analysis computations (general purpose)
  - Rendering methods:
    - Volume rendering, point rendering, ray casting,...
- **Cross platform**
  - ATI & NVIDIA cards
  - Linux, Windows, and MacOS X
  - OpenGL
- **Development tools**
  - GUI/IDE - for visualization
  - Command line compiler

# Visualization IDE



# scc - Command line compiler



# Scout Overview

---

- Open source?
  - Hopefully by October 2005
  - Will be available for academic and non-commercial use
  - We'll announce on [gpgpu.org](http://gpgpu.org) when available...

# Language Introduction - Shapes

---

- All variables have a shape
  - A template for parallel data...
  - Defined by:
    - The number of dimensions - the *rank* (current support for 1D, 2D, and 3D)
    - The number of positions along each dimension

```
// Define a two-dimensional grid
shape grid[512][512];
float:grid density;
```

# Language Introduction - **with**

---

- A program can only operate on data from one shape at a time
  - Data read from disk is predefined (no need to define in code)
- With statement designates the current shape
  - **sizeof** operator returns shape of given variable.

```
// Define a two-dimensional grid
shape grid[512][512];
float:grid density;
with(sizeof(density)) {
    // Your code here...
}
```

# Language Introduction - **with**

---

- Scout adds modifiers to C\*'s with statement
  - **compute with**
    - Pure computation (i.e., keep 32-bit precision)
  - **volren with**
    - Volume render - code implements shader for transfer function
  - **raycast with**
    - Raycast - code implements shader for samples

# A Simple Example

---

Compute  
pass

```
// compute the mean
float sum = 0.0;
compute with(shapeof(pt)) {
    sum += pt; // reduction
}
```

```
float mean = sum / positionsof(pt);
```

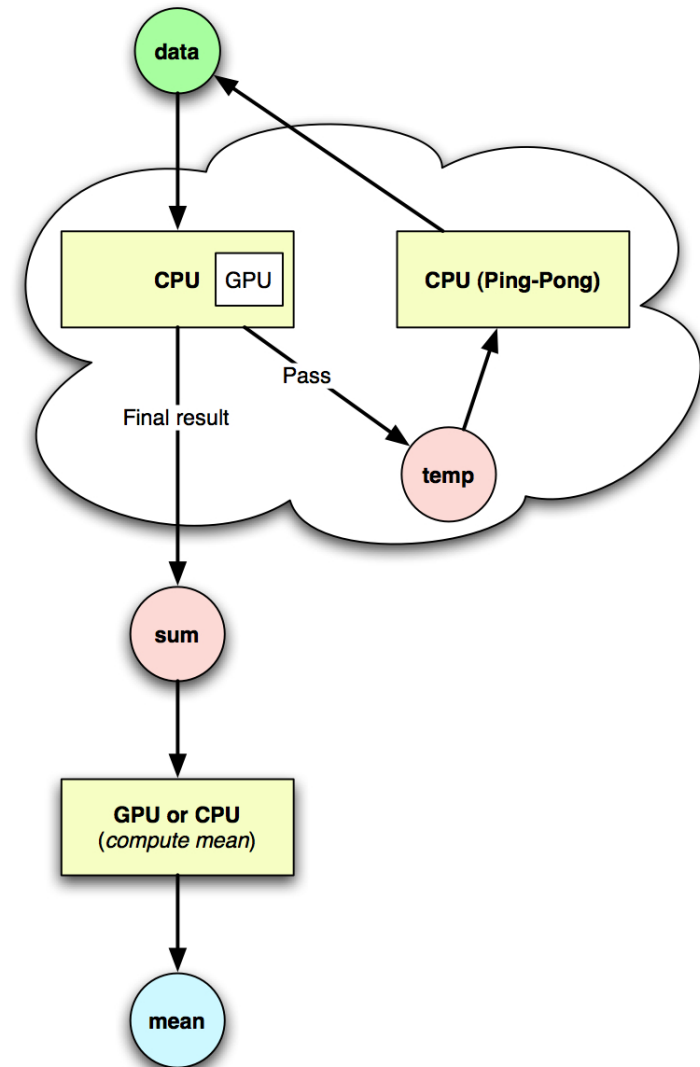
```
// volume render cells only less than the mean.
```

Render  
pass

```
volren with(shapeof(pt)) {
    where(pt < mean)
        image = 0; // black
    else
        image = hsva(240 - norm(pt) * 240, 1, 1, 0.2);
}
```

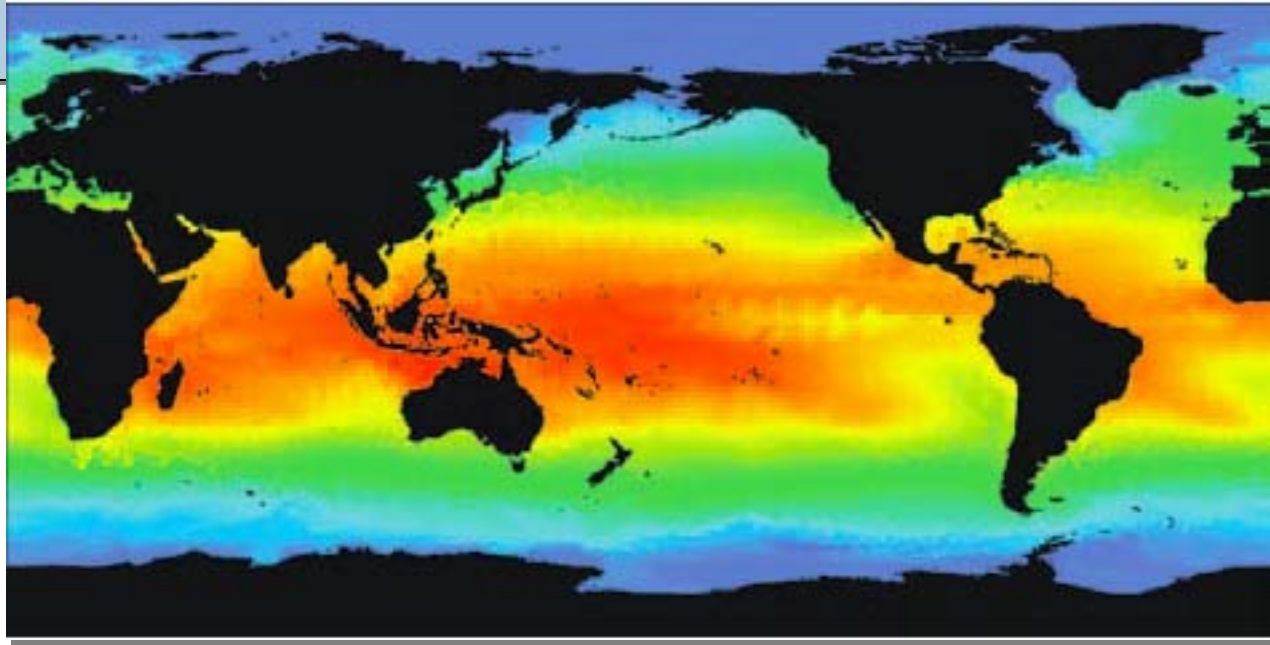
# Under the Hood

```
// Compute mean value
float mean(float data)
{
    float sum;
    compute with(shapeof(data))
        sum += data; // reduction
    return sum / positionsof(data);
}
```



# Example

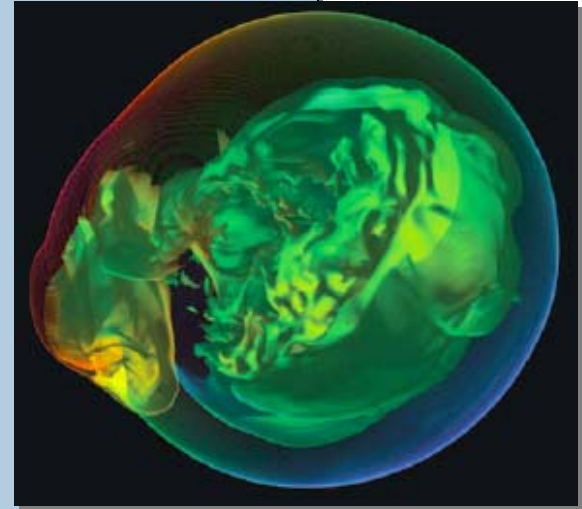
```
// Compute mean value
render with(shapeof(pt)) {
  // land and pt must have the same shape..
  where(land) // Don't color the continents..
    image = 0;
  else
    image = hsva(240 - norm(pt) * 240, 1.0, 1.0, 1.0);
}
```



# Example

```
// compute entropy and velocity magnitude...
float:shapeof(pressure) entropy;
float:shapeof(pressure) vmag; // velocity magnitude
compute with(shapeof(pressure)) {
    entropy = pressure / pow(density, 4.0/3.0);
    vmag     = sqrt(dot3(velocity, velocity));
}

// compute gradient normals for shading here...
volren with(shapeof(entropy)) {
    // select interior region of entropy and clip out along X axis.
    where(i > 115 && entropy > 0.07 && entropy < 0.076) {
        image = hsva(240 - norm(vmag) * 240.0, 1.0, diffuse, 1.0);
    } else where(entropy > 0.01 && entropy < 0.04) {
        // this is the shock wave...
        image = hsva(240 - norm(vmag) * 240.0, 1.0, 1.0, 0.1);
    } else {
        image = 0; // black...
    }
}
```



# More to come...

---

- **Case study later in the day...**
  - More rendering methods
  - Multiple GPUs...
- **Integration with Mio**
  - Riffel, Lefohn, Vidimce, Leone, and Owens, "Mio: Fast Multipass Partitioning via Priority-Based Instruction Scheduling", Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, pp. 35-44, 2004.