

## A Platform for Scalable Micro-Simulations

Johannes Gehrke

(Joint work with Tuan Cao, Al Demers, Oliver Gao, Christoph Koch, Ben Sowell, Marcos Vaz Salles, Guozhang Wang, Xun Wang, Walker White)

Department of Computer Science and  
Department of Civil and Environmental Engineering  
Cornell University

<http://www.cs.cornell.edu/bigreddata/games>

<http://bigreddata.blogspot.com>



---

---

---

---

---

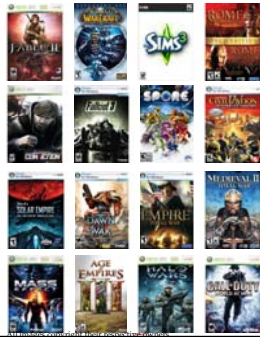
---

---

---

## Computer Games

- Virtual environments
- High degree of interactivity



---

---

---

---

---

---

---

---

## Simulation Games



The Sims 3 © Electronic Arts

- “Doll House” games
  - NPCs have needs and desires.
  - Objects can satisfy needs and desires.
  - Player control via object placement.



---

---

---

---

---

---

---

---

## Simulation Games



Warcraft III © Blizzard Entertainment

- Real-Time Strategy (RTS) games
  - Troops move and fight in real time.
  - Player control via limited number of commands.
  - Player multitasks between large number of units.



---

---

---

---

---

---

---

---

## Data-Driven Game Design



- Game design brings together many disciplines
  - Art, design, storytelling, music, computer science, etc...
- Thus games are designed *data-driven*
  - Game content is separated from game code
  - Examples:
    - Character data is kept in XML
    - Character behavior is specified through *scripting languages*
- Engine is reusable.



---

---

---

---

---

---

---

---

## Simulation Game Design: NPCs



- Non-Player Characters (NPCs): Characters not directly controlled by the player.
  - Main actors in the game
    - *Doll House Games*: Enticing a hungry NPC with some food
  - Enemies controlled by the computer
  - Allies indirectly controlled by the player
    - *RTS Games*: Issuing commands to military units
- **Simulation games**: All characters are NPCs
  - All character actions simulated by computer
  - Player controls everything *indirectly*



---

---

---

---

---

---

---

---

## The Role Of Scripting Languages



NPCs are programmed in a

**scripting language**

- Easy environment for gameplay programmers and designers
  - Sandbox for creation of "fun"
  - Make gameplay development fast and efficient
- User-created content (e.g., Second Life)
- Mods
  - Half-Life → Counter-Strike




---

---

---

---

---

---

---

---

## Why Is Scaling NPCs Hard?



- Example: Morale
  - Battle between  $n$  knights and  $n$  skeletons
  - Assume knights that are afraid of skeletons
  - Morale inverse proportional to number of skeletons in view




---

---

---

---

---

---

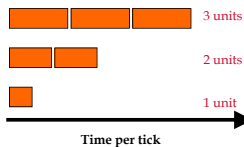
---

---

## Scaling NPCs (Contd.)



- Example: Morale
  - Battle between  $n$  knights and  $n$  skeletons
  - Assume knights that are afraid of skeletons
  - Morale inverse proportional to number of skeletons in view
- Each knight counts the number of skeletons in his view
  - $O(n)$  per unit to count visible skeletons
  - $O(n^2)$  to process all units
- Computation  $\approx$  frame rate




---

---

---

---

---

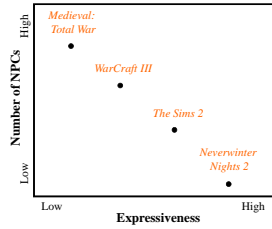
---

---

---

## Expressiveness vs. Performance

- **Expressiveness:** The range of behavior that can be scripted (outside the engine).
- As the number of NPCs increases, expressiveness decreases.
  - *Neverwinter Nights 2*
    - Each NPC fully scriptable
  - *WarCraft III*
    - Script armies, not NPCs
    - Little NPC coordination
  - *Medieval: Total War*
    - No individual scripting at all




---

---

---

---

---

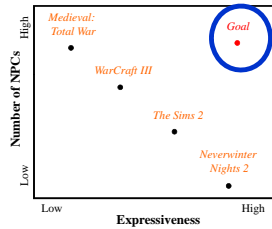
---

---

---

## Expressiveness vs. Performance

- **Expressiveness:** The range of behavior that can be scripted (outside the engine).
- As the number of NPCs increases, expressiveness decreases.
  - *Neverwinter Nights 2*
    - Each NPC fully scriptable
  - *WarCraft III*
    - Script armies, not NPCs
    - Little NPC coordination
  - *Medieval: Total War*
    - No individual scripting at all




---

---

---

---

---

---

---

---

## Disadvantages of Scripting Languages

- Problems:
  - Performance
  - Lack of support for concurrency

→ All of these issues make gameplay development harder, not easier.




---

---

---

---

---

---

---

---

## Abstraction and Performance



Can we have *both* high-level scripting and high performance?

Database systems scale to petabytes. How?

- Queries are specified in a declarative language
- Set-oriented processing: Relational algebra
- Deep query/multi-query optimization
- Automatic parallelization

→ Practical and asymptotic performance benefits

Ideas:

- Declarative language (SQL)
- Set-at-a-time processing

Can we use these ideas to improve scripting for games and simulations?



---

---

---

---

---

---

---

---

## Talk Outline

- Inside a simulation engine
- The simulation language SGL
- From games to simulations



---

---

---

---

---

---

---

---

## Inside the Simulation Engine (1)

- Actions are divided into "ticks", in which each unit
  - Reads the environment and determines its current action
  - Performs that action, creating one or more effects
    - An effect may alter a unit's own state (i.e., movement)
    - An effect may alter the state of others (i.e., damage)



---

---

---

---

---

---

---

---

## Inside the Simulation Engine (2)

- At the end of the tick, effects update the environment.
  - All effects are processed simultaneously.
  - Have rules to combine effects in order-independent ways.
    - **Stackable Effects**: Effects are combined (sum or product)
    - **Nonstackable Effects**: Rule for taking one effect (maximum)



---

---

---

---

---

---

---

---

## Observation

- We can model the state of the simulation through database tables
- At each tick each unit:
  - Reads the environment and determines its current action → **Database queries**
  - Performs that action, creating one or more effects → **Database updates**



---

---

---

---

---

---

---

---

## Impact of this Observation

- Units often perform a lot of similar computation.
  - Example: Units may all follow the same script parameterized by the environment.
  - Optimize with **set-at-time processing**.
    - Determine all effects with a single database query.
    - Apply all effects as single update at end of tick.
- Sometimes computation is shared
  - Example: Units overlapping, not same, line-of-sight.
  - Optimize with **multi-script optimization techniques**.



---

---

---

---

---


---

---

---

## The State-Effect Pattern

- Objects in the game have attributes.
- The attributes are either **states** or **effects**.



States					Effects		
id	player	x	y	health	vx	vy	damage
1	1	12	342	100	-10	3	50
2	1	43	12	100	0	5	0
3	2	123	90	95	9	9	0

Figure from Lineage II © NCSoft



---

---

---

---

---


---

---

---

## The State-Effect Pattern

- **State attributes** are read only variables that are updated only at the end of a tick.
- **Effect attributes** are temporary variables that are used for intermediate computation during a tick.



States					Effects		
id	player	x	y	health	vx	vy	damage
1	1	12	342	100	-10	3	50
2	1	43	12	100	0	5	0
3	2	123	90	95	9	9	0



---

---

---

---

---


---

---

---

## Inside the Simulation Engine

A simulation tick has three phases:



States					Effects		
id	player	x	y	health	vx	vy	damage
1	1	12	342	100	-10	3	50
2	1	43	12	100	0	5	0
3	2	123	90	95	9	9	0



---

---

---

---

---

---


---

---

## Inside the Simulation Engine

A simulation tick has three phases:

- **Query Phase** - Read state variables.



States					Effects		
id	player	x	y	health			
1	1	12	342	100	-10	3	50
2	1	43	12	100	0	5	0
3	2	123	90	95	9	9	0




---

---

---

---

---

---


---

---

## Inside the Simulation Engine

A simulation tick has three phases:

- Query Phase - Read state variables.
- **Effect Phase** - Compute values for effect variables. Multiple assignments to an effect are combined using a commutative and associative **aggregation function**.



States					Effects		
id	player	x	y	health	vx	vy	damage
1	1	12	342	100	-10	3	
2	1	43	12	100	0	5	
3	2	123	90	95	9	9	

12
25
13




---

---

---

---

---

---


---

---

## Inside the Simulation Engine

A simulation tick has three phases:

- Query Phase - Read state variables.
- Effect Phase - Compute values for effect variables. Multiple assignments to an effect are combined using a commutative and associative aggregation function.
- **Update Phase** - Compute new values for state variables from the effects and previous state variables.



States					Effects		
id	player	x	y	health	vx	vy	damage
1	1	2	345	50	0	0	0
2	1	43	17	100	0	0	0
3	2	132	99	95	0	0	0




---

---

---

---

---

---

---

---

## State-Effect Processing Model

### Simulation Tick Summary:

1. **Query Phase:** Units perform actions
  - Produces a collection of effects
  - Output is a table of affected units with their effects
2. **Effect Phase:** Combine effects from actions
  - Multiple assignments to an effect are combined using a commutative and associative aggregation function.
3. **Update Phase:** Update the state
  - Compute new values for state variables from the effects and previous state variables.



---

---

---

---

---

---

---

---

## Scaling Scripts to Many NPCs

- First idea: Use **declarative** language for scripts.
  - Language specifies **what** I want
  - System finds the best way to **compute** it
- Allows us to use declarative processing for scripts
- How about SQL?



---

---

---

---

---

---

---

---

## That's Why Not SQL

```
SELECT U.PLAYER, U.TYPE, U.HEALTH, U.X, U.Y, SUM(E.DAMAGE), MAX(E.HEAL), SUM(E.VX),
SUM(E.VY)
FROM Units U,
( SELECT T.KEY, 0 as DAMAGE, HAMOUNT as HEAL, 0 as VX, 0 as VY
FROM Units T, Units H
WHERE H.TYPE = "Healer" AND H.PLAYER = T.PLAYER AND IN_RANGE(T,H)

UNION

SELECT T.KEY, 0 as DAMAGE, 0 as HEAL, GOAL_X-T.X) / DIST(T.X,GOAL_X) AS VX,
(GOAL_Y-T.Y) / DIST(T.Y, GOAL_Y) AS VY
FROM Units T, Units A
WHERE NOT EXISTS (SELECT * FROM Units T WHERE IN_RANGE(T,A))

UNION

SELECT MIN(T.KEY), DAMOUNT as DAMAGE, 0 as HEAL, 0 as VX, 0 as VY
FROM Units T, Units A
WHERE IN_RANGE(T,A) AND A.TYPE = "Archer" AND A.PLAYER <> T.PLAYER
AND T.HEALTH = ANY(
SELECT MIN(S.HEALTH) FROM Units S
WHERE IN_RANGE(S,A) AND A.PLAYER <> S.PLAYER
)
) as E
WHERE U.KEY = E.KEY
GROUPBY U.KEY
```



---

---

---

---

---

---

---

---

## SGL 2009

- SGL 2009 has an imperative syntax that is familiar to simulation designers.
- To enable efficient processing, we must restrict the language
  - Enforce the state-effect pattern using **class definitions**.
  - Limit iteration with **special looping construct**.



---

---

---

---

---

---

---

---

## Data Definition Language

- Defines state & effects
- Combinators for effects
- Primitive update rules
- Adv. post-processing takes place in engine

```
class Unit {
  state:
    number player = 0;
    number type = 0;
    number x = 0;
    number y = 0;
    number health = 0;
    Unit target = null;
  effects:
    number vx : avg;
    number vy : avg;
    number damage : sum;
    number healing : max;
    Unit acquired : priority;
  update:
    x = x + vx;
    y = y + vy;
    health = health - damage;
    target = (acquired != null
      ? acquired : target);
}
```



---

---

---

---

---

---

---

---

## Example SGL Script

```
main : Unit {
  // Compute # of skeletons and center of crowd
  effect number c : sum, number sx : sum, number sy : sum;
  over Unit u from UNIT {
    if (isEnemySkeleton(u) && dist(me,u) < range) {
      c <- 1; sx <- u.x; sy <- u.y;
    }

    // If too many skeletons
    if (c > morale) {
      const number (norm = (x-sx/c)*(x-sx/c) + (y-sy/c)*(y-sy/c));
      // Run in opposite direction
      vx <- (x-sx/c)/norm; vy <- (y-sy/c)/norm;
    }
  }
}
```



---

---

---

---

---

---

---

---

## Example SGL Script (Contd.)

```
if (c > morale) {  
  ...  
} else if (c > 0 && cooldown == 0) {  
  // Find the nearest skeleton  
  effect Unit target : argmin(Unit t : dist(me,t));  
  over x from UNIT {  
    if (isEnemySkeleton(x)) {  
      target <- x;  
    }  
  }  
  // Attack it if found.  
  if (target != null) {  
    target.damage <- DMG_AMT;  
  }  
}  
} // end main
```



---

---

---

---

---

---

---

---

## SGL 2009 Review

- SGL 2009 is an imperative language that can be processed declaratively.
- Scripter writers program *individual* NPCs.
- Expressive power is limited so that SGL scripts can be compiled to extended relational algebra
  - State-effect pattern
  - Looping

→ Declarative processing



---

---

---

---

---

---

---

---

## Declarative Processing

- Process query phase as one DB query
  - Gather together all NPCs for same script
  - Execute script on all NPCs simultaneously
- Compile each script to a query plan
  - Conditionals are selections ( $\sigma$ )
  - accum-over loops are joins ( $\bowtie$ )
  - consts are table extensions ( $\pi$ )



---

---

---

---

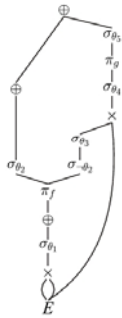
---

---

---

---

## Declarative Processing: An Example



```

main : Unit {
  // Compute # skeletons, group center
  effect number c : sum;
  effect number sx : sum,
    number sy : sum;
  over Unit u from UNIT {
    if (isEnemySkeleton(u) &&
        dist(me,u) < range) {
      c <- 1; sx <- u.x; sy <- u.y;
    }
  }
  // If too many skeletons
  if (c > morale) {
    const norm=(x-sx/c)*(x-sx/c)+
      (y-sy/c)*(y-sy/c));
    // Run in opposite direction
    vx <- (x-sx/c)/norm;
    vy <- (y-sy/c)/norm;
  }
  ...
}

```




---

---

---

---

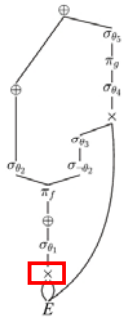
---

---

---

---

## Declarative Processing: An Example



```

main : Unit {
  // Compute # skeletons, group center
  effect number c : sum;
  effect number sx : sum,
    number sy : sum;
  over Unit u from UNIT {
    if (isEnemySkeleton(u) &&
        dist(me,u) < range) {
      c <- 1; sx <- u.x; sy <- u.y;
    }
  }
  // If too many skeletons
  if (c > morale) {
    const norm=(x-sx/c)*(x-sx/c)+
      (y-sy/c)*(y-sy/c));
    // Run in opposite direction
    vx <- (x-sx/c)/norm;
    vy <- (y-sy/c)/norm;
  }
  ...
}

```




---

---

---

---

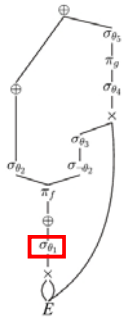
---

---

---

---

## Declarative Processing: An Example



```

main : Unit {
  // Compute # skeletons, group center
  effect number c : sum;
  effect number sx : sum,
    number sy : sum;
  over Unit u from UNIT {
    if (isEnemySkeleton(u) &&
        dist(me,u) < range) {
      c <- 1; sx <- u.x; sy <- u.y;
    }
  }
  // If too many skeletons
  if (c > morale) {
    const norm=(x-sx/c)*(x-sx/c)+
      (y-sy/c)*(y-sy/c));
    // Run in opposite direction
    vx <- (x-sx/c)/norm;
    vy <- (y-sy/c)/norm;
  }
  ...
}

```




---

---

---

---

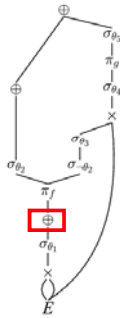
---

---

---

---

## Declarative Processing: An Example



```

main : Unit {
  // Compute # skeletons, group center
  effect number c : sum;
  effect number sx : sum;
  effect number sy : sum;
  over Unit u from UNIT {
    if (isEnemySkeleton(u) &&
        dist(me,u) < range) {
      c <- 1; sx <- u.x; sy <- u.y;
    }
  }
  // If too many skeletons
  if (c > morale) {
    const norm=(x-sx/c)*(x-sx/c)+
              (y-sy/c)*(y-sy/c);
    // Run in opposite direction
    vx <- (x-sx/c)/norm;
    vy <- (y-sy/c)/norm;
  }
  ...
}

```




---

---

---

---

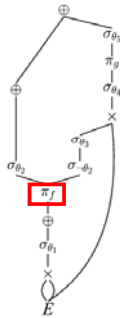
---

---

---

---

## Declarative Processing: An Example



```

main : Unit {
  // Compute # skeletons, group center
  effect number c : sum;
  effect number sx : sum;
  effect number sy : sum;
  over Unit u from UNIT {
    if (isEnemySkeleton(u) &&
        dist(me,u) < range) {
      c <- 1; sx <- u.x; sy <- u.y;
    }
  }
  // If too many skeletons
  if (c > morale) {
    const norm=(x-sx/c)*(x-sx/c)+
              (y-sy/c)*(y-sy/c);
    // Run in opposite direction
    vx <- (x-sx/c)/norm;
    vy <- (y-sy/c)/norm;
  }
  ...
}

```




---

---

---

---

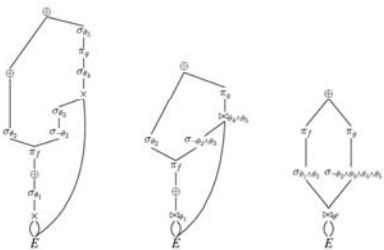
---

---

---

---

## Optimize the Query Plan




---

---

---

---

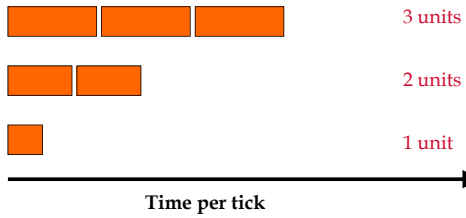
---

---

---

---

## What Have We Achieved?



---

---

---

---

---

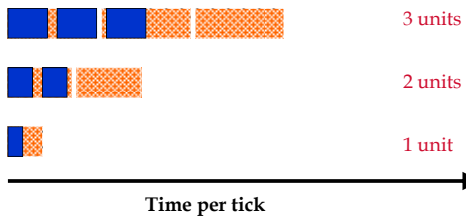
---

---

---

## More Efficient Processing

What about the asymptotically quadratic behavior?



---

---

---

---

---

---

---

---

## Optimizing Aggregates (Contd.)

- The problem is actually a bit harder:
  - Script associated with each NPC
  - Script performs aggregate computation that depends on the unit  $u$ 
    - Example: Count number of skeletons in my neighborhood
    - In our query plans:  $\Pi_{*}^{agg}(\sigma_{u,u}(R))$
  - Needs to be computed for every unit in the environment table  $\rightarrow O(n^2)$  cost
- But: We "understand" the scripts!



---

---

---

---

---

---

---

---

## Optimizing Aggregates: Approaches

- Indexing
  - Construct an index in time  $O(n^2)$
  - Lookup aggregate value through index nested-loops join
- Sweep-line algorithms
  - Compute all aggregate values in time  $O(n^2)$  at the beginning of each tick



---

---

---

---

---

---

---

---

## Experimental Design

- Number of NPCs vs. time for 500 clock ticks.
- Pluggable simulation comparing different query plans
  1. Naive processing of aggregates.
  2. Use of indexing techniques.
    - Queries are two spatial dimensions plus categorical attribute.
    - Categorical attribute is hash table, so constant time.
    - Performance is thus  $O(n \log n)$  for each aggregate
- Hardware parameters:
  - 2Ghz Intel Core Duo running OS X in 1.5 GB RAM.
  - Compiled in C++ using GCC.



---

---

---

---

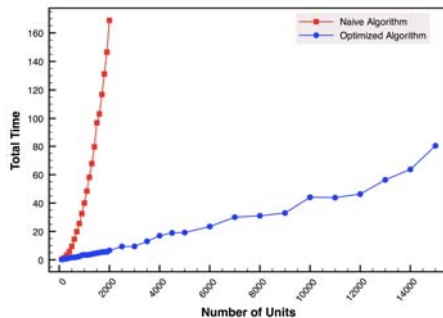
---

---

---

---

## Experimental Results



---

---

---

---

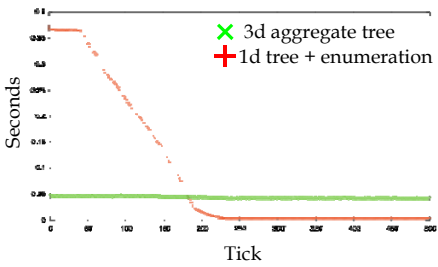
---

---

---

---

## Dynamic Query Optimization



Selective query:  
Iterate over objects normally.

Otherwise:  
Use index.



---

---

---

---

---

---

---

---

## Talk Outline

- Inside a simulation engine
- The simulation language SGL
- From games to simulations of individuals



---

---

---

---

---

---

---

---

## Simulations: Requirements

- High-level programming
  - SGL
- Performance
  - Compiles directly into C++
  - Completely main-memory based simulations
- Scalability
  - Automatic parallelization with MPI
  - Completely transparent to the programmer
- Cost-effectiveness
  - Runs in the cloud
  - Amazon EC2; get 1000 nodes for one hour
    - \$100: 1TB of main memory
    - \$400: 7TB of main memory
- Fault-tolerance



---

---

---

---

---

---

---

---

## Target Applications

- Micro-simulations
- Time-stepped (tick model)
- State-effect pattern



---

---

---

---

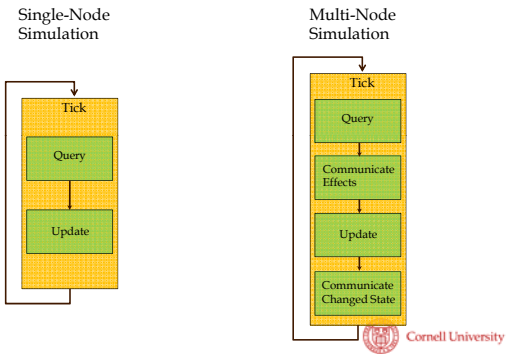
---

---

---

---

## Time-stepped Simulation: Tick Model



---

---

---

---

---

---

---

---

## Many Details

- What is the right communication model?
- How should the data be partitioned?
- How much synchronization is necessary?

(Future research.)



---

---

---

---

---

---

---

---

## Fault Tolerance

- Leverage state-effect pattern to achieve high-performance single-instance checkpointing algorithms
  - Application-level, tick-based
  - Outperform existing work by a factor of three
- Synchronization performed by load balancer can also to coordinate checkpoints
- Recovery is similar to rebalancing:
  - Replace failed node with standby
  - Rollback all nodes to most recent checkpoint




---

---

---

---

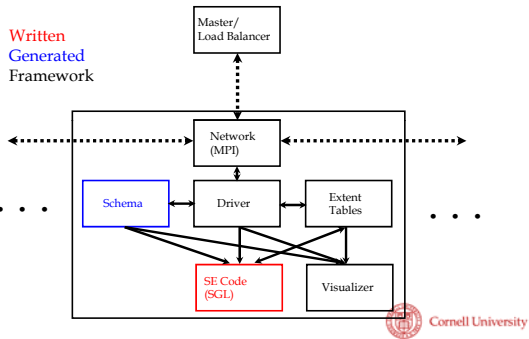
---

---

---

---

## Simulation Architecture




---

---

---

---

---

---

---

---

## One Application: Transportation Micro-Simulations

- Travel-related factors
  - Amount of travel, time of travel, mode choice, engine load
- Driver behavior
  - Variability in speed: Acceleration increases pollution
- Highway network characteristics
  - Traffic conditions
  - Grade
- Vehicle characteristics
  - Interaction of fuel, engine, model, maker, odometer/age

Transportation modeling (demand forecast, network assignment, dynamic simulation)

Slide based on material from Oliver Gao




---

---

---

---

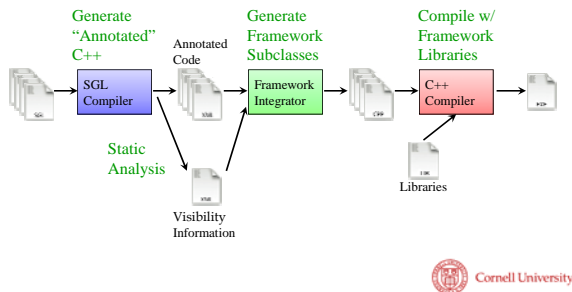
---

---

---

---

## Simulation Development Workflow



---

---

---

---

---

---

---

---

## Vision

Platform for scalable simulations in the cloud

- But *not* "one size fits all"
- Closes the loop jointly with data collection and data mining

Scientist can concentrate on design of simulation instead of performance

System handles (deep) optimizations and generates *scalable* code with performance close to hand-optimized C++

System automatically distributes simulation into the cloud, linear scalability



---

---

---

---

---

---

---

---

## Availability

- Single instance end-to-end system running
- Beta-version in the cloud in progress (beta availability: early November)
- Sample ongoing collaborations
  - Large-scale traffic simulation (joint work with Oliver Gao, Cornell CEE); incorporating MOVES
  - Simulating fish schools (based on models from Iain Couzin)
  - **Your micro-simulation?**



---

---

---

---

---

---

---

---

## Research Challenges

---

- Programming model
- Automatic distribution
- Performance optimizations
- Spatial data structures
- Debugging



---

---

---

---

---

---

---

---

## Questions?

---

<http://www.cs.cornell.edu/bigreddata/games>

<http://bigreddata.blogspot.com>

Thank you: National Science Foundation, Air Force Office of Scientific Research, Microsoft, Yahoo!

---

---

---

---

---

---

---

---