Game Mathematics & Game State

- The Complexity of Games
- Expectations of Players
- Efficiency
- Game Mathematics
  - Collision Detection & Response
  - Object Overlap Testing
  - Minkowski Sum
  - Geometry, Trigonometry
  - Vectors
- Game State
Video Games Are Very Complex Programs

Programming a simple web-hosted Flash game requires a very high level of programming knowledge; may also require expertise in:
• Network Protocols
• Data retrieval and management techniques
• Graphic & Sound creation/management
• Geometry, Trigonometry and Calculus
• Vector mathematics
• 3D mathematics
Player Expectations

In general, games are held to a higher standard than other types programs.

People expect "office applications" to fail, and don't expect 100% up-time from business websites.

What we are willing to tolerate when "working" is wildly different then what we are willing to tolerate when "playing".
Efficiency

Complexity and computability are concepts that are not normally taught on an undergraduate level.

Game programmers need to consider "efficiency" in everything they do.

If a player had to wait more than 30 seconds for levels to load in a game, their "review" of that game was greatly reduced.
"Game Mathematics" refers both to areas of general mathematics (geometry, trigonometry, calculus) as well as specialized areas of mathematics (vectors, matrices).

Graphic libraries, game libraries, 2D and 3D libraries exist for programming languages to help simplify the mathematical problems that you will face. But they can't be relied on to do everything.
Collision Detection

Figuring out if two objects are touching incredibly common problem in a game.

Two basic techniques:
1. Overlap testing
   o Detects whether a collision has already occurred
2. Intersection testing
   o Predicts whether a collision will occur in the future
Overlap Testing

Facts:
• Most common technique used in games
• Exhibits more error than intersection testing

Concept
• For every simulation step, test objects to see if they overlap
• Easy for simple volumes like spheres, harder for polygonal models
Simple Overlap Testing

Simple example is particle interacting with a square. This will still require 4 logical tests in a 2D game.

Depending on the type of game that played, the order of those 4 tests can have a profound effect on efficiency.
Complex Shape OOT

How many tests would be required now?
Bounding Boxes

Bounding Boxes can be used to reduce the complexity of shapes to simplify overlap testing. Note that secondary testing may need to be done if the bounding box is found to overlap. How many test now?
Minkowski Sum

By taking the Minkowski Sum of two complex volumes and creating a new volume, overlap can be found by testing if a single point is within the new volume.

Variations on the Minkowski Sum include calculating the x, y and z distances between the two objects that are being tested.
Minkowski Sum
Collision time calculated by moving object back in time until right before collision.

- Bisection is one effective technique.
- Minkowski values are another.
Collision Response

Having captured the exact moment and position of collision, geometry, and trigonometry can be applied to calculate new trajectories.
Limits of OT

OT is easy, but limited.
- Fails with objects that move too fast
- Unlikely to catch time slice during overlap

Possible solutions
- Design constraint on speed of objects
- Reduce simulation step size
- Use Vectors
Vectors

- You are already know of "vector images", images represented by a mathematical formula.
- We can represent entire objects (and their movement) with formula's as well.
- Vector (an matrice) mathematics can then be applied to reveal information about where objects will be and whether or not the will collide (at any time, past or future).

P1[ x1, y1, x2, y2]; // A particle vector.
Game State

All games consist of a sequence of states. Each state is characterized by a combination of visual, audio and/or animation effects, as well as a set of rules that are being applied.
Objects in the game proceed through their own states as well. These states are defined by the behavior and functionality applied at that time.

- **Initialization**
- **Wandering**
  - if(see_enemy==1)
- **Attacking**
  - if(health<20)
- **Resolution**
- **Fleeing**
  - if(health<0)
Game State in Scratch

At typical game in Scratch might use the following state transition diagram.

**Green Flag State**
- when flag clicked

**Setup**
- when I receive setup
  - level 0 state, level 1 state
  - level 2 state, level 99 state

**Go**
- when I receive go
  - level 0 state, level 1 state
  - level 2 state, level 99 state

**Stop**
- when I receive stop
  - level 0 state, level 1 state
  - level 2 state, level 99 state

**Game Over State**

In every game state each object/sprite will have its own local state that is based on variables like level.

The stop or resolution state is where we will update variables that apply to state (like level.)
Object State in Scratch

In a typical game in Scratch, all of your normal sprites (not stage or any control sprites like buttons) will work very well with only 4 scripts. These scripts (and any associated variables) will control the objects state.

- **when green旗 clicked**
  - hide
  - stop script

- **when I receive setup**
  - point in direction 90°
  - go to x: -120 y: 0
  - set size to 40%
  - switch to costume costume1
  - if level = 0
  - hide
  - stop script

- **when I receive go**
  - stop script

- **when I receive stop**
  - hide
  - stop script
The End