Force Based Anticipatory Collision Avoidance in Crowd Simulation

Stephen J. Guy &
Ioannis Karamouzas
University of Minnesota

AI ARTIFICIAL INTELLIGENCE SUMMIT

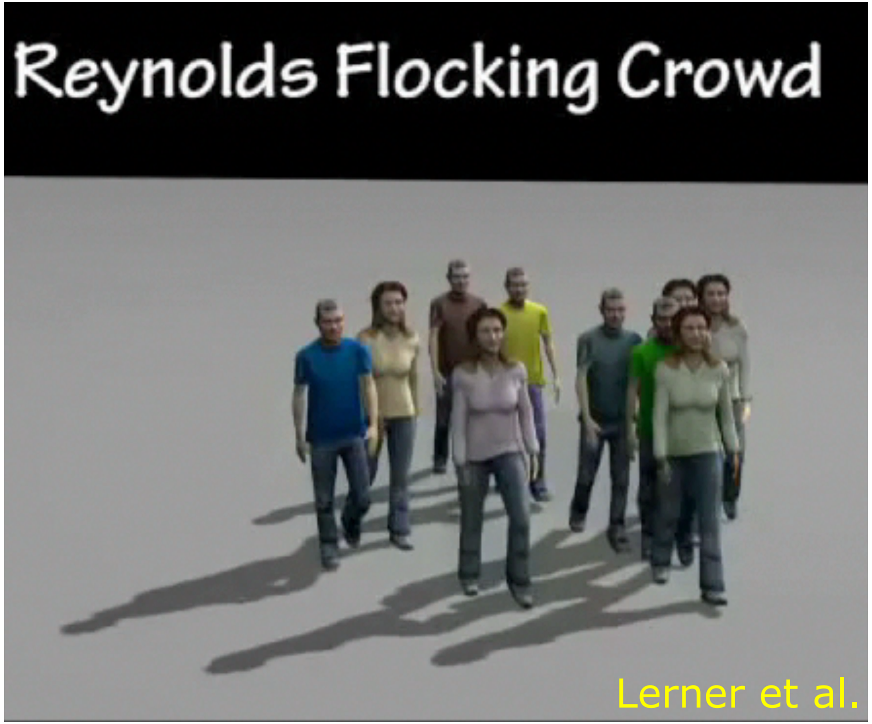
GAME DEVELOPERS CONFERENCE®
MOSCONE CENTER · SAN FRANCISCO, CA
MARCH 2-6, 2015 · EXPO: MARCH 4-6, 2015

Collision avoidance is hard to perfect

No Avoidance:

SimCity V, © 2013 Maxis

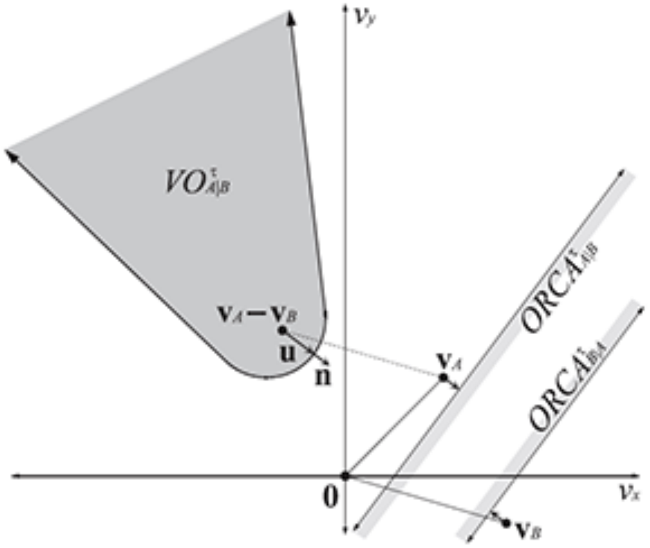
No Anticipation:



Lerner et al.

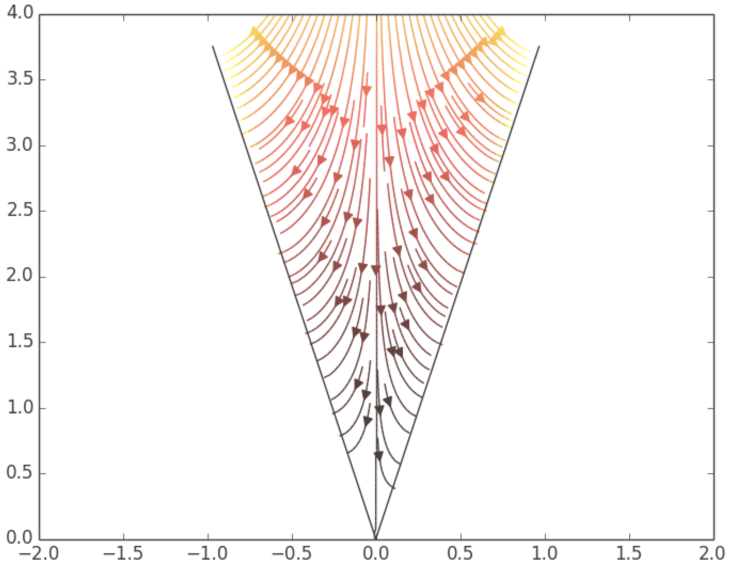
Today's Topics

1)



Geometric Optimization

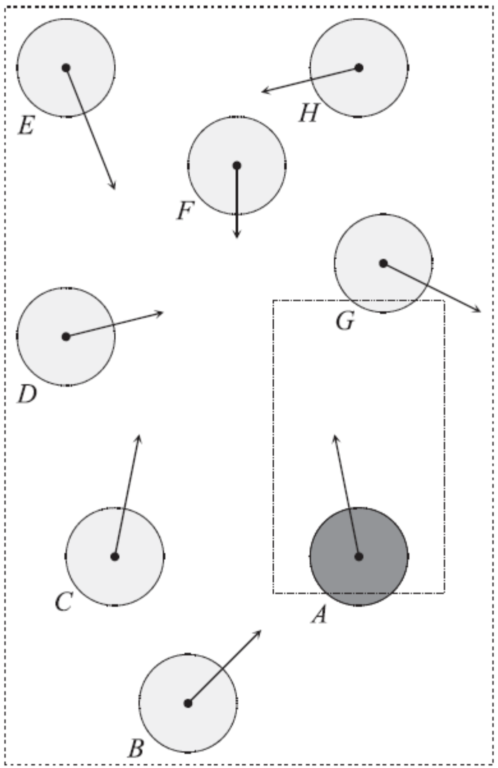
2)



Data-Driven Forces

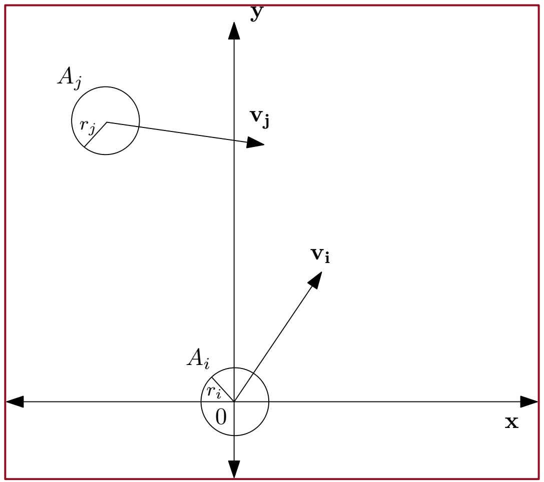
Optimal Reciprocal Collision Avoidance (ORCA)

- Formulate as an optimization problem
- Assume agents reciprocate
- Locally optimal solution can be computed for each agent in parallel!

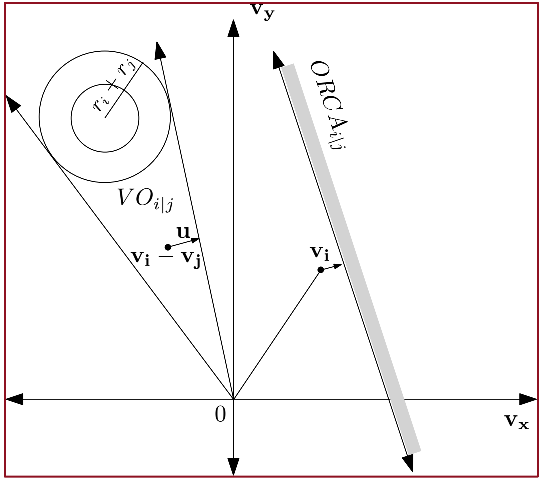


Velocity Space

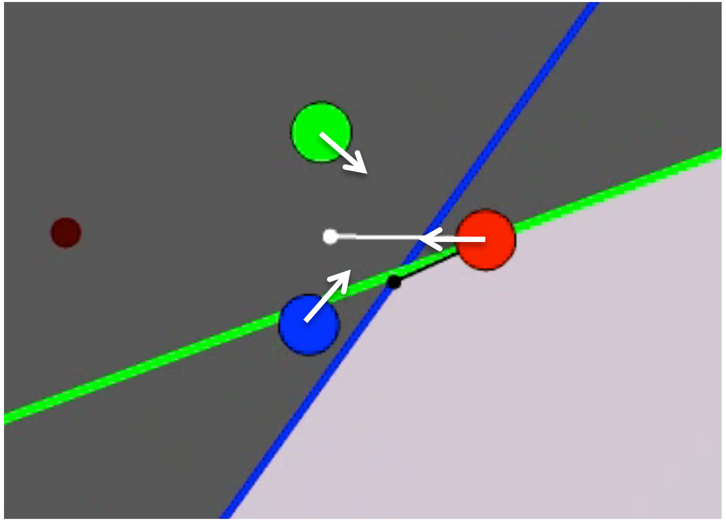
- Each agent plans in velocity space



World-Space



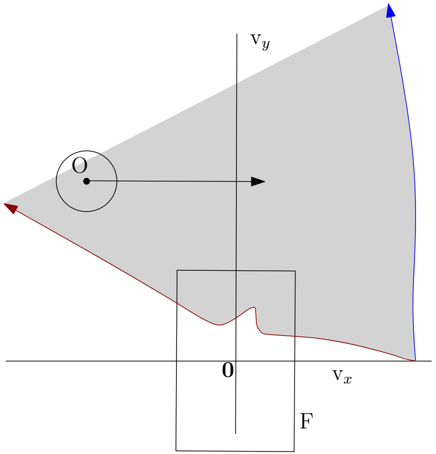
Velocity-Space

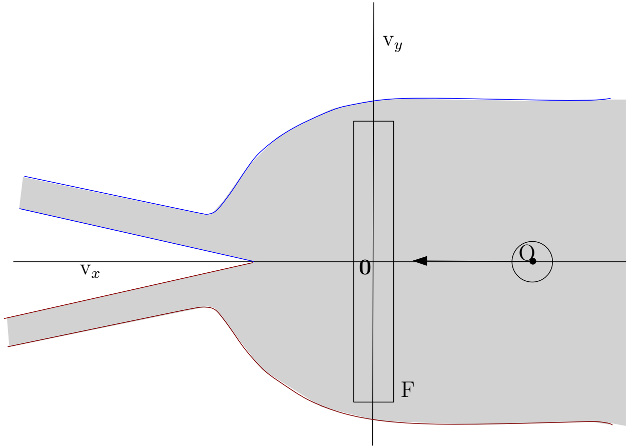


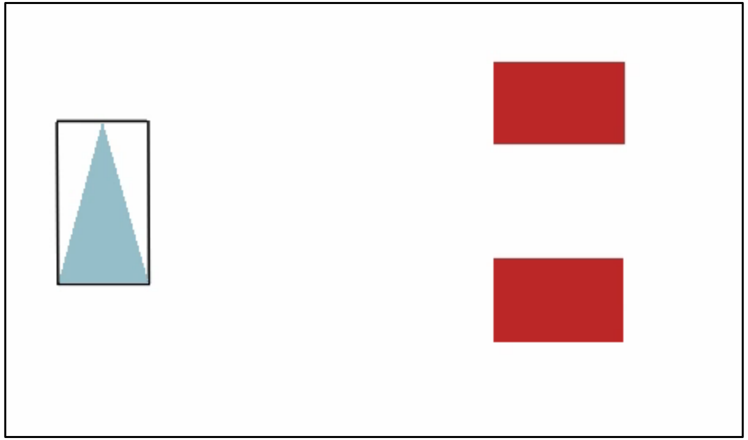
http://gamma.cs.unc.edu/RVO2

Extending ORCA & VOs

- ORCA can be extend to oriented dynamics
  - Useful for formations





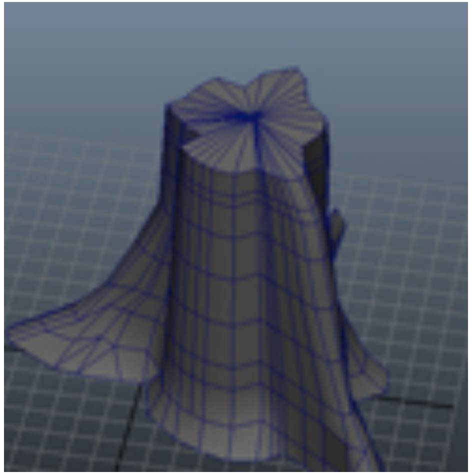


Success with ORCA & Vel. Space

- Developers have incorporated this ideas


Disney/ETH, 2012


© Relic Entertainment, 2012

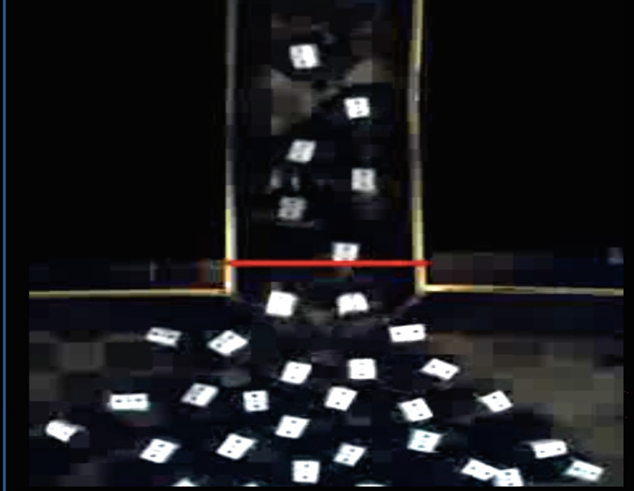

Dan Brewer's
AI Summit Talk

When ORCA goes wrong

- *Locally optimal* is not really optimal
  - Agents can be “afraid”

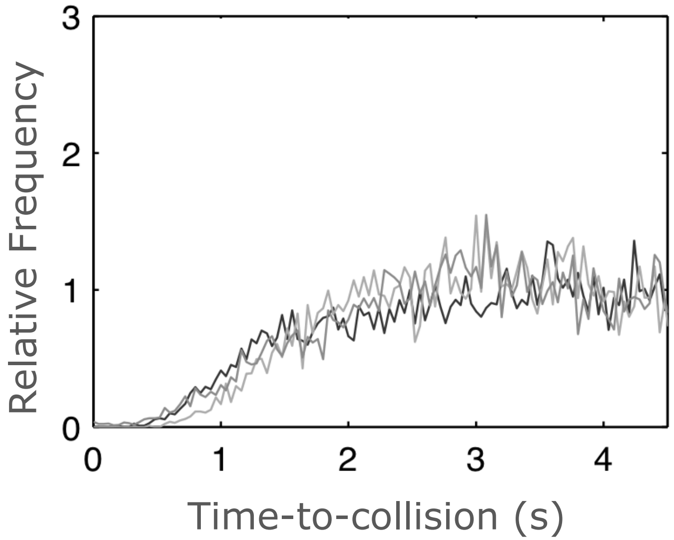
Turning to Humans…

- How do humans do it?
  - Lots of data exist!
- What are the statistical trends in pedestrian trajectories?
- Can we quantify human collision-avoidance interactions?



Crowd Energy

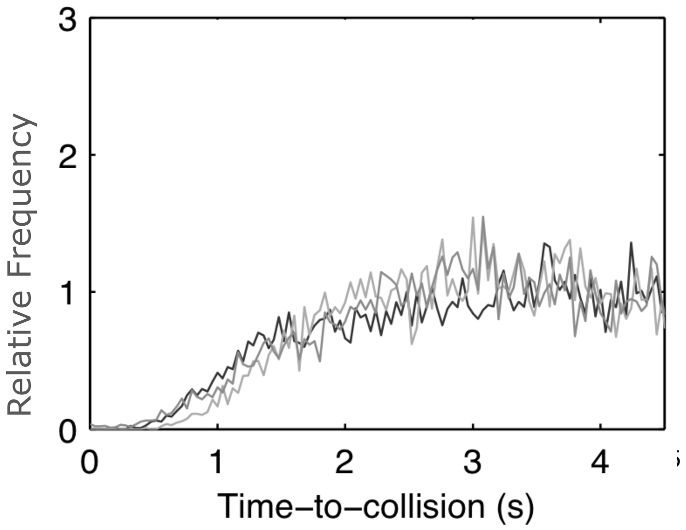
- Navigation discomfort can be quantified
  - Function of time-to-collision (τ)

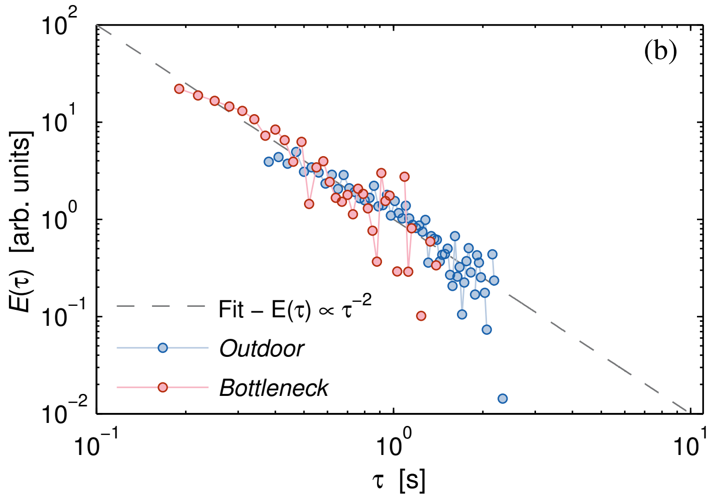


[Karamouzas et al. 2014]

Crowd Energy Law

- $Energy = k * \log(1/frequency)$
- Energy falls off inversely proportional to τ^2

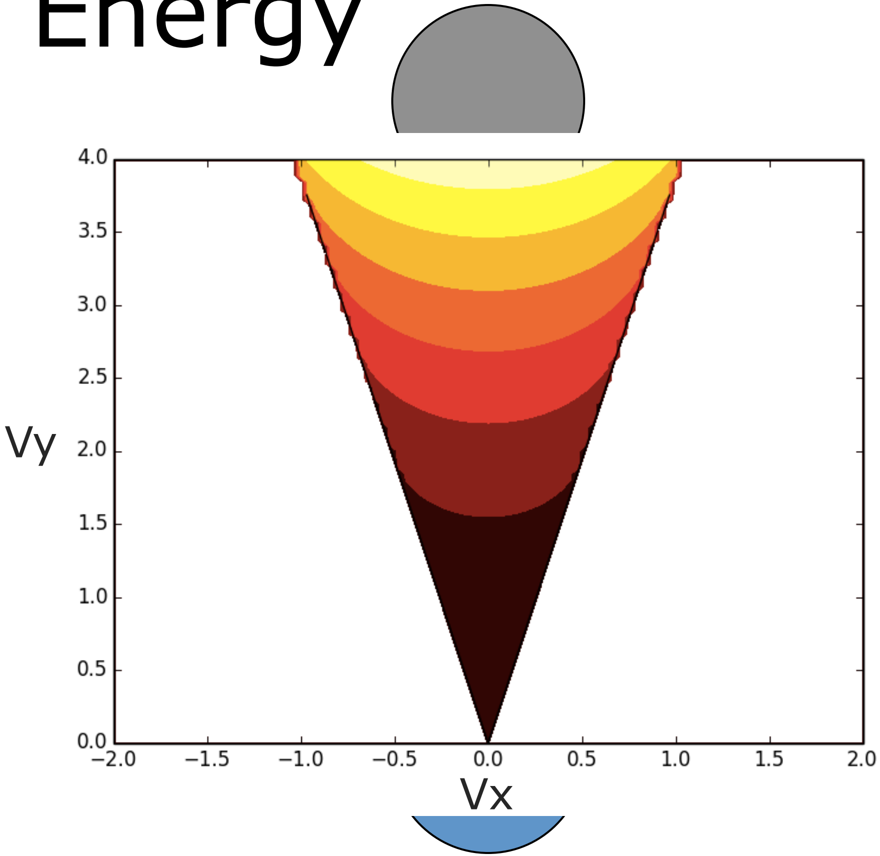




$$E(\tau) = k / \tau^2$$

Visualizing Crowd Energy

- Energy is defined inside the VO
- Agents ignore neighbors not on a collision course
- Energy increases with higher speeds

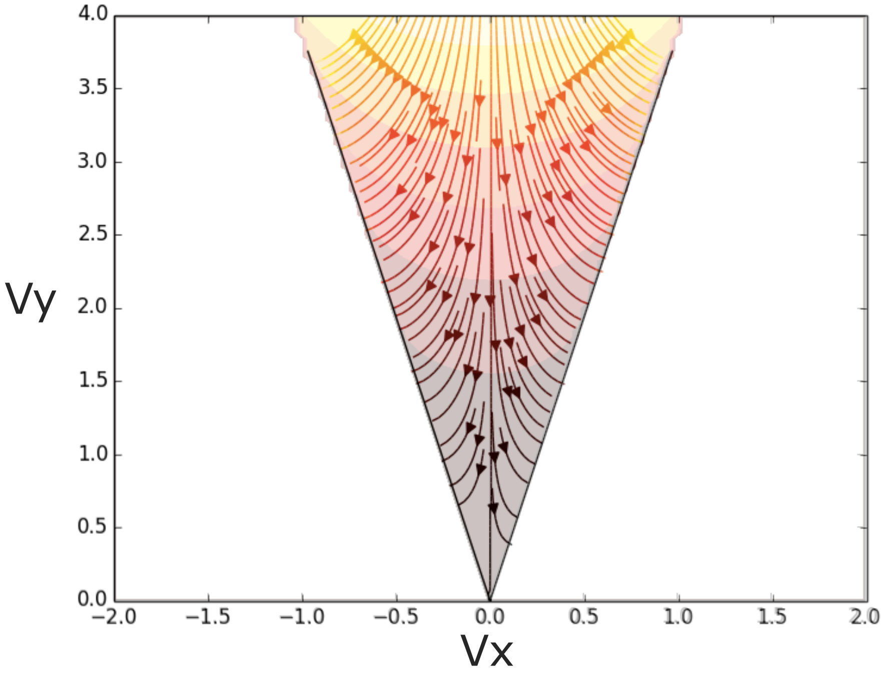


From Energy to Forces

- The spatial derivative of energy gives avoidance forces

$$\vec{F} = -\nabla k / \tau^2$$

The forces can be computed analytically!



Computing Forces

- Analytical solution to forces

Compute TTC (τ)

```
def dE(pa, pb, va, vb, ra, rb):
    w  = pb - pa; v = va - vb; radius = ra+rb
    a = v.dot(v); b = w.dot(v); c = w.dot(w) - radius*radius;
    discr = b*b - a*c;
    if (discr < 0): return np.array([0, 0])
    discr = sqrt(discr);
    t = (b - discr) / a;

    if (t < 0): return np.array([0, 0]) #no collision = no force

    return k*(v - (v*b - w*a)/(discr))/(a*t**2)*(2/t)
    #k is a scaling constants
```

Computing Forces

- Analytical solution to forces

```
def dE(pa, pb, va, vb, ra, rb):
    w  = pb - pa; v = va - vb; radius = ra+rb
    a = v.dot(v); b = w.dot(v); c = w.dot(w) - radius*radius;
    discr = b*b - a*c;
    if (discr < 0): return np.array([0, 0])
    discr = sqrt(discr);
    t = (b - discr) / a;

    if (t < 0): return np.array([0, 0]) #no collision = no force

    return k*(v - (v*b - w*a)/(discr))/(a*t**2)*(2/t)
    #k is a scaling constants
```

Compute Energy

The Full Simulation

- Main forces: Goal force & Avoidance force

The Full Simulation

- Main forces: **Goal force** & Avoidance force

```
for i in range(num):
    F[i] += (gv[i]-v[i])/.5
    F[i] += 1*np.array([rnd.uniform(-1.,1.),rnd.uniform(-1.,1.)])

    for n, j in enumerate(nbr[i]): #j is neighboring agent
        dEdx = dE(p[i],p[j],v[i],v[j],r,r)
        FAvoid = -dEdx
```

The Full Simulation

- Main forces: Goal force & **Avoidance force**

```
for i in range(num):
    F[i] += (gv[i]-v[i])/.5
    F[i] += 1*np.array([rnd.uniform(-1.,1.),rnd.uniform(-1.,1.)])

    for n, j in enumerate(nbr[i]): #j is neighboring agent
        dEdx = dE(p[i],p[j],v[i],v[j],r,r)
        FAvoid = -dEdx
```

The Full Simulation

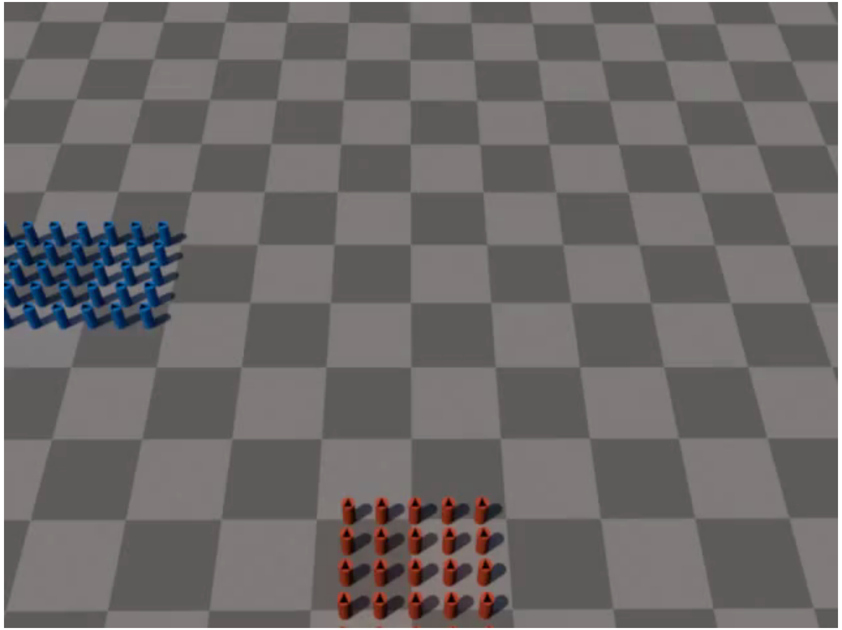
- Multiple forces:
  - Goal directed motion
  - Collision avoidance
  - Follow player forces
  - Grouping forces
  - …

Practicalities

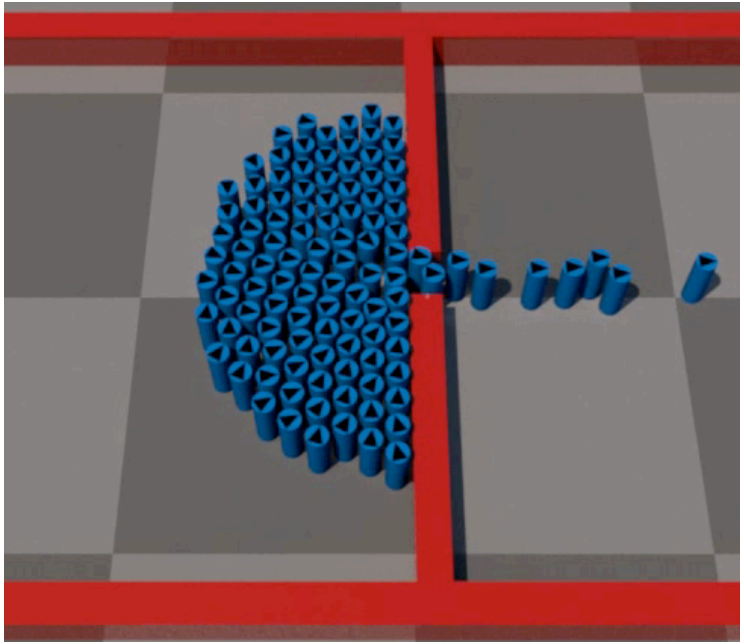
- Ignore far away collisions (> 3s)
- Clamp to a max force
- Choose a small timestep (e.g., 5ms)
  - Use multiple simulation timesteps per rendering frame

http://motion.cs.umn.edu/PowerLaw

Results

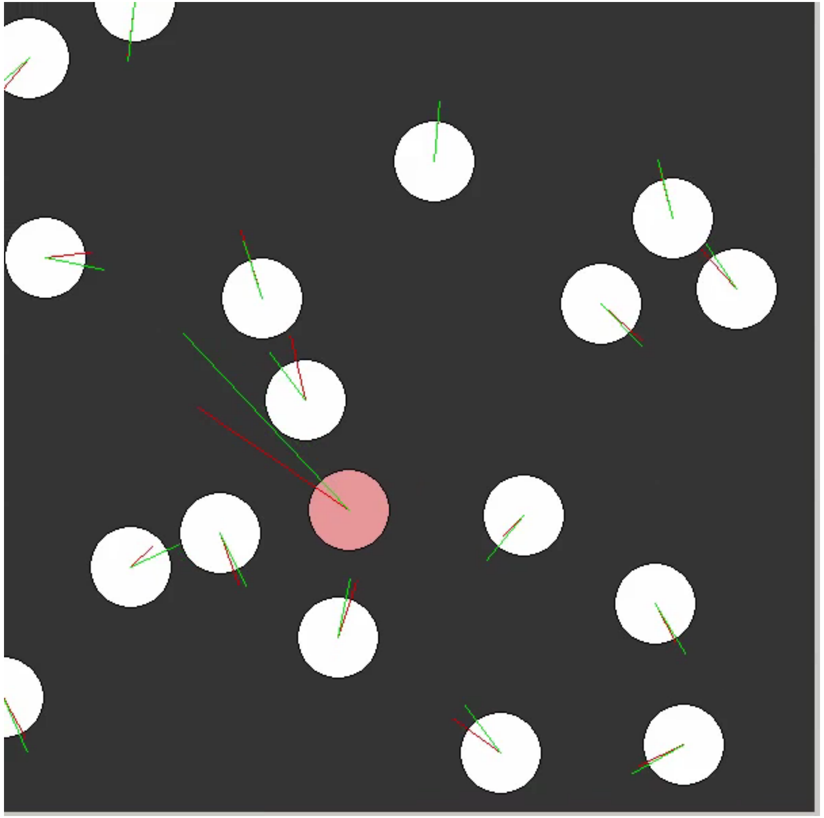


Complex Crossing



Dense Interaction

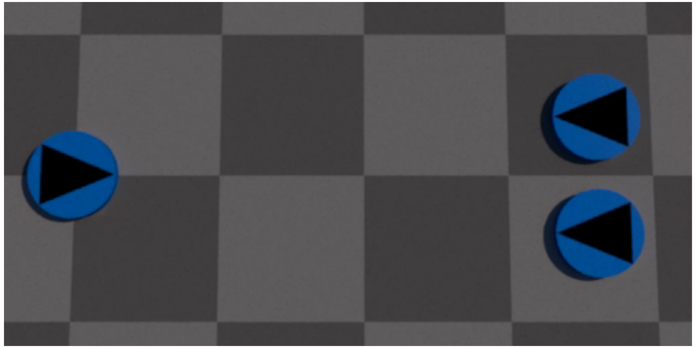
Heterogeneous Speeds:



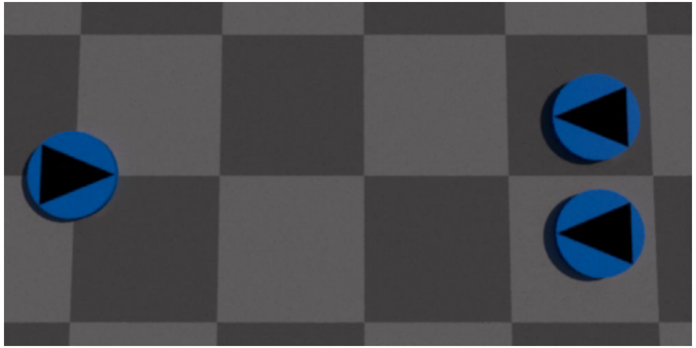
Comparison to ORCA

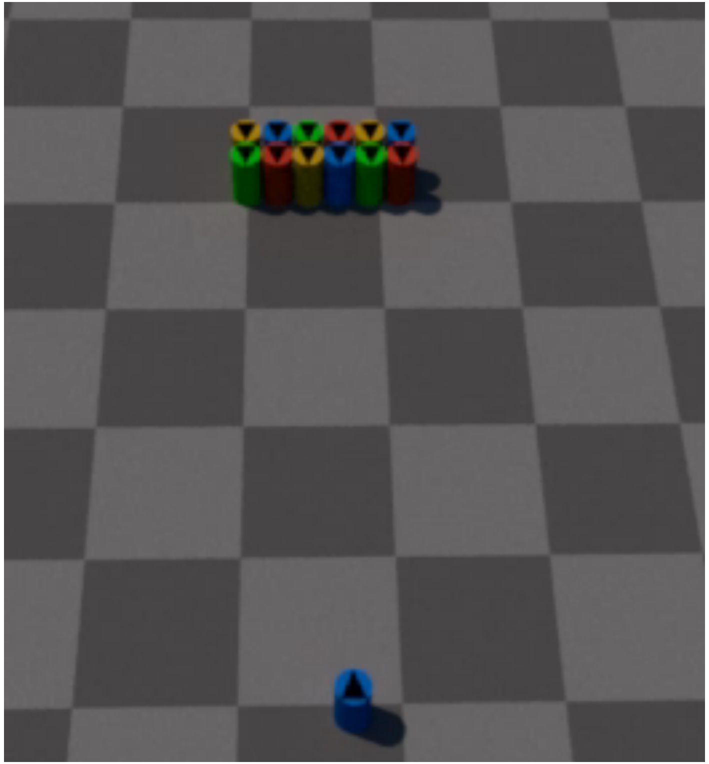
- Less timid agents

TTC Forces:

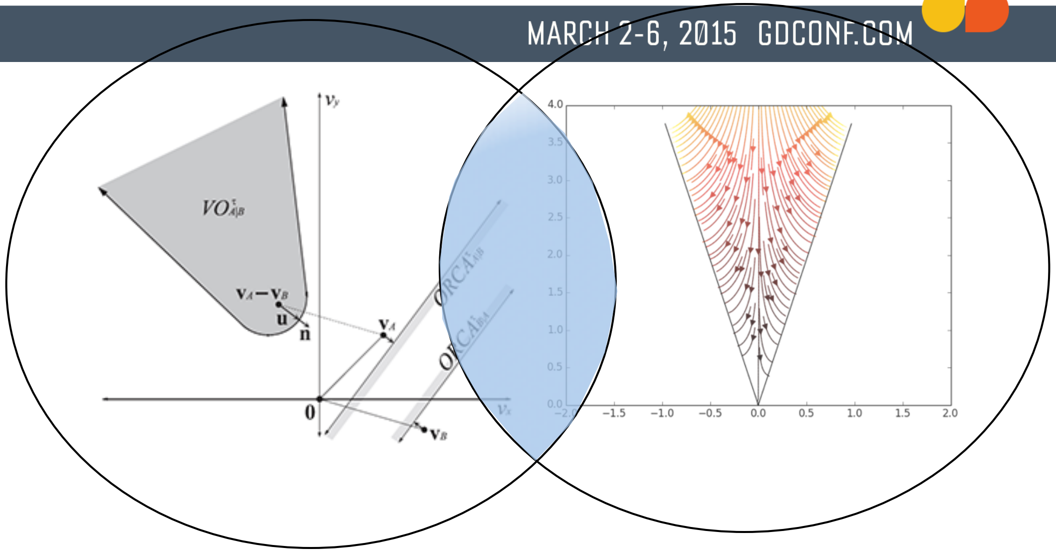


ORCA:





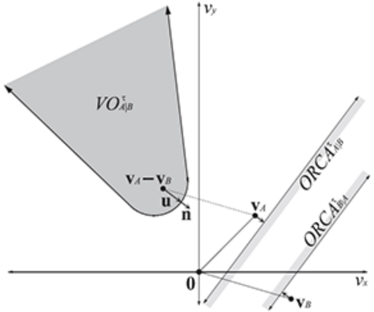
Take-Aways

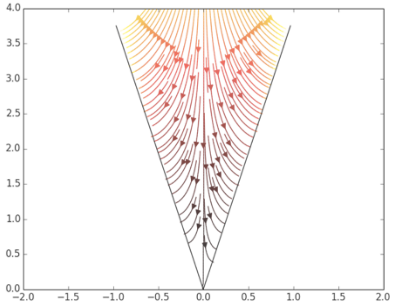


- Two methods, many similarities

- Both methods are:
  - Fast & Scalable to 1000s of agents
  - Collision free
  - Anticipatory

Take-Aways
(Differences)





ORCA:

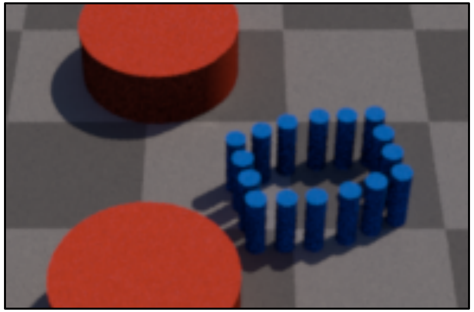
- Strong guarantees
  - Even with large timesteps
- Flexible framework
- Difficult to implement
- Takes too much control over velocity

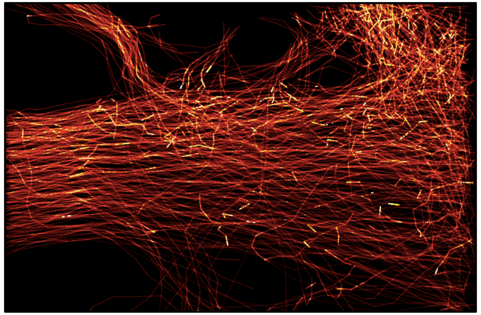
TTC Forces:

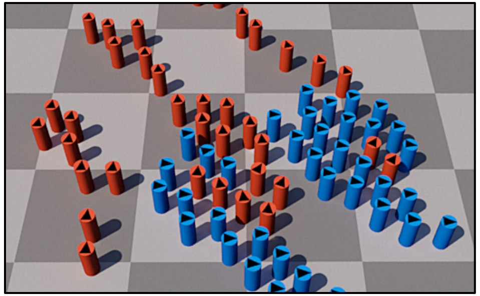
- May choose a colliding vel.
- Collision free
- … but only for small enough timesteps
- Natural to tweak forces

Conclusions

- ORCA provides guarantees is very extendable
- Data-driven TTC forces work well in practice
- TTC forces are simple to implement with good results







Compact Sim.

- Goal force
- Avoidance force
  - $\vec{F}_j = -\vec{\nabla} k / \tau^2$
- vel += F*dt;
- pos += vel*dt

```
#include <cstdlib> //  card > ksg.gif
#include <cstdio>  //By: Stephen J. Guy
#include <cmath>   //sjguy@cs.umn.edu
typedef int i;typedef float f;struct V{f x,y;
V(){x=y=0;}V operator*(f f){return V(x*f,y*f)
;}V(f a,f b){x=a;y=b;}V operator+(f f){return
V(x+f,y+f);}V operator-(V r){return V(x-r.x,y
-r.y);}f operator%(V o){return x*o.x+y*o.y;}
void n(){f l=sqrt(x*x+y*y);x/=l;y/=l;}};f dt=
.01;f r=.25;f s=15;i w=500;i h=500;const i A=
14;V p[A],g[A],v[A];void B(i n){putchar(n);}
void P(i n){printf("%s",&n);}void Z(i n=3){
for(;n--;){B(0);}}f m;i main(){for(i a=0;a<A;
a++){p[a]=V(sin(6.28*a/A),cos(6.28*a/A))*6+s/
2;g[a]=p[a]-(p[a]+-s/2)*2;v[a]=(g[a]-p[a])*(1
+rand()%8)*.02;}printf("%s","GIF89a");P(w);P(
h);B(246);Z(2);for(i n=125;n--;){B(rand()%255
);B(rand()%255);B(rand()%205);}P(0xbbbbbb);P(
0xdd5050);Z();P(0x0BFF21);printf("%s","NETSC\
APE");P(0x302E32);P(259);Z();for(i z=62;z--;)
{P(0x04F921);B(8);P(10);Z();B(',');Z(4);P(w);
P(h);Z(1);B(7);i b=0;f n,q,y,t,d;for (n=h;n--
;){for(m=w;m--;){if(b%50==0){B(51);B(128);}i
a=0;for(;a<A;a++){V x(s*m/w,s*n/h);x=x-p[a];
if(x%x<r){if(x%x>.8*r)a=127;V h=v[a];h.n();x=
x-h*1.6*r;if(x%x<.3*r)a=127;if(a<A/2)a+= A/2;
break;}}(a==A)?B(((i)n/50%2==(i)m/50%2)?126:
125):B(a);b++;}}P(1);P(129);B(0);for(i l=.1/
dt;l--;){for(i a=A;a--;){for(i b=A;b--;){if(a
!=b){V w=p[b]-p[a];V u=v[a]-v[b];q=u%u;y=w%u;
d=y*y-q*(w%w-4*r);if(d*q>0){;d=sqrt(d);t=(y-d
)/q;if(t>-0){V f=(u-(u*y-w*q)*(1/d))*(20/(q*t
*t));if (f%f>961)f=f*(31/sqrt(f%f));v[a]=v[a]
-f*dt;}}}}V o=g[a]-p[a];if(o%o>1){o.n();o=o*3
;}v[a]=v[a]-(v[a]-o)*dt*3;p[a]=p[a]-v[a]*-dt;
}}}B(';');}
```

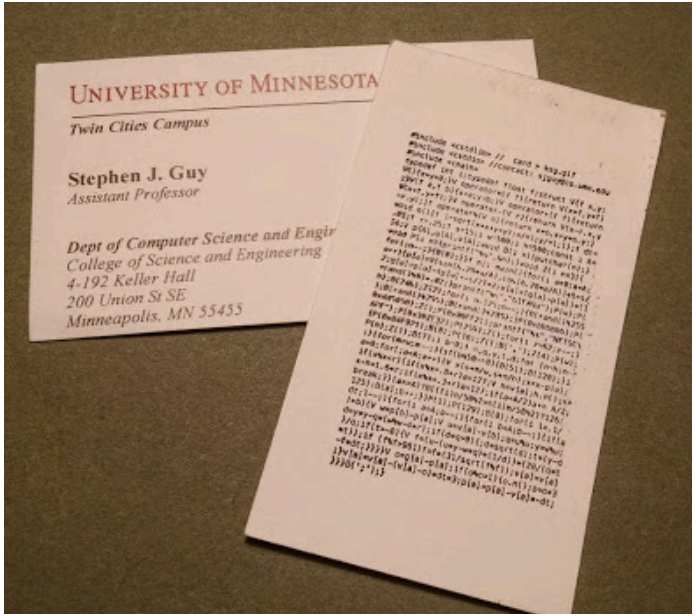
Compact Sim.

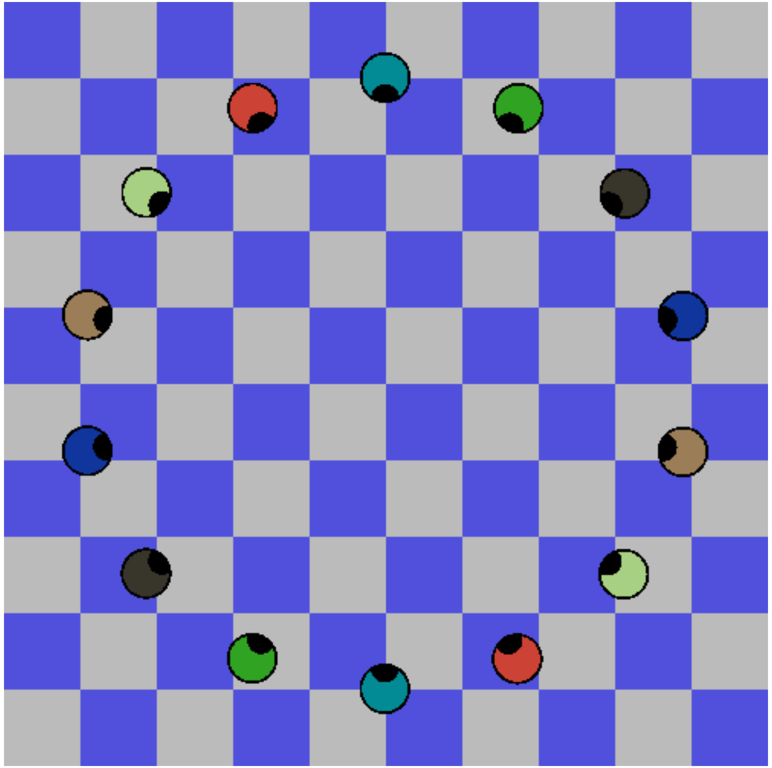
- Vector Library
- Scene Initialization
- Animated Gif Header
- Rasterizer
- The actual forces
- Simple Integration

```
#include <cstdlib> //  card > ksg.gif
#include <cstdio>  //By: Stephen J. Guy
#include <cmath>   //sjguy@cs.umn.edu
typedef int i;typedef float f;struct V{f x,y;
V(){x=y=0;}V operator*(f f){return V(x*f,y*f)
;}V(f a,f b){x=a;y=b;}V operator+(f f){return
V(x+f,y+f);}V operator-(V r){return V(x-r.x,y
-r.y);}f operator%(V o){return x*o.x+y*o.y;}
void n(){f l=sqrt(x*x+y*y);x/=l;y/=l;}};f dt=
.01;f r=.25;f s=15;i w=500;i h=500;const i A=
14;V p[A],g[A],v[A];void B(i n){putchar(n);}
void P(i n){printf("%s",&n);}void Z(i n=3){
for(;n--;){B(0);}}f m;i main(){for(i a=0;a<A;
a++){p[a]=V(sin(6.28*a/A),cos(6.28*a/A))*6+s/
2;g[a]=p[a]-(p[a]+-s/2)*2;v[a]=(g[a]-p[a])*(1
+rand()%8)*.02;}printf("%s","GIF89a");P(w);P(
h);B(246);Z(2);for(i n=125;n--;){B(rand()%255
);B(rand()%255);B(rand()%205);}P(0xbbbbbb);P(
0xdd5050);Z();P(0x0BFF21);printf("%s","NETSC\
APE");P(0x302E32);P(259);Z();for(i z=62;z--;)
{P(0x04F921);B(8);P(10);Z();B(',');Z(4);P(w);
P(h);Z(1);B(7);i b=0;f n,q,y,t,d;for (n=h;n--
;){for(m=w;m--;){if(b%50==0){B(51);B(128);}i
a=0;for(;a<A;a++){V x(s*m/w,s*n/h);x=x-p[a];
if(x%x<r){if(x%x>.8*r)a=127;V h=v[a];h.n();x=
x-h*1.6*r;if(x%x<.3*r)a=127;if(a<A/2)a+= A/2;
break;}}(a==A)?B(((i)n/50%2==(i)m/50%2)?126:
125):B(a);b++;}}P(1);P(129);B(0);for(i l=.1/
dt;l--;){for(i a=A;a--;){for(i b=A;b--;){if(a
!=b){V w=p[b]-p[a];V u=v[a]-v[b];q=u%u;y=w%u;
d=y*y-q*(w%w-4*r);if(d*q>0){;d=sqrt(d);t=(y-d
)/q;if(t>-0){V f=(u-(u*y-w*q)*(1/d))*(20/(q*t
*t));if (f%f>961)f=f*(31/sqrt(f%f));v[a]=v[a]
-f*dt;}}}}V o=g[a]-p[a];if(o%o>1){o.n();o=o*3
;}v[a]=v[a]-(v[a]-o)*dt*3;p[a]=p[a]-v[a]*-dt;
}}}B(';');}
```

My Business Card ☺

- 1,500 char crowd sim





Acknowledgements

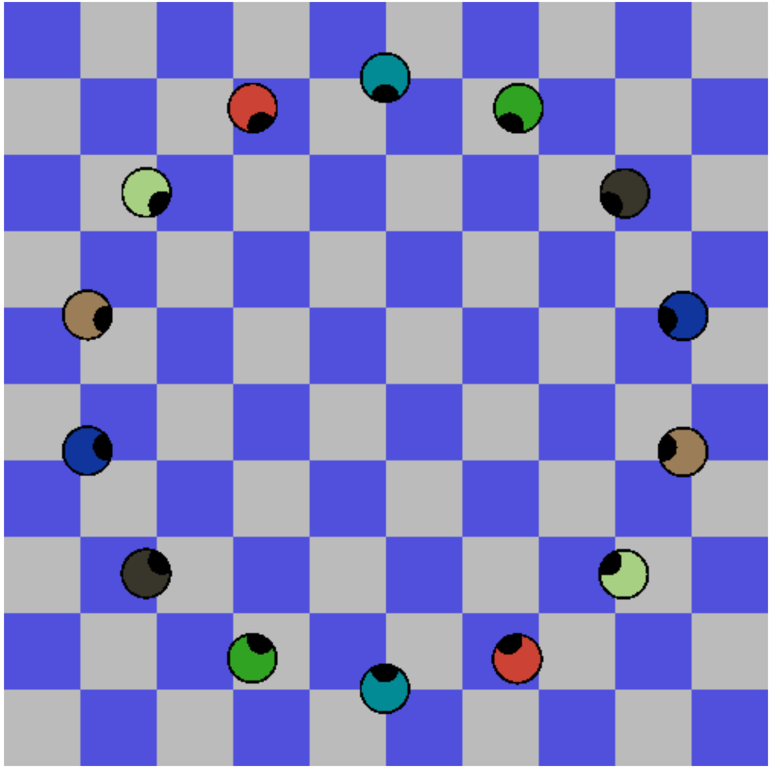
- Lab Members:
  - **Ioannis Karamouzas**
  - John Koenig
  - Bilal Kartal
  - Julio Godoy
  - Ran Hu
  - Bobby Davis
  - Devin Lange

External Collaborators:

- Dinesh Manocha & Ming Lin, UNC
- Jur van den Berg, UNC & UC Berkley (now at Google)

Thanks!

- Stephen J. Guy
  - sjguy@cs.umn.edu
- Ioannis Karamouzas
  - johnoriginal@gmail.com



http://motion.cs.umn.edu/PowerLaw