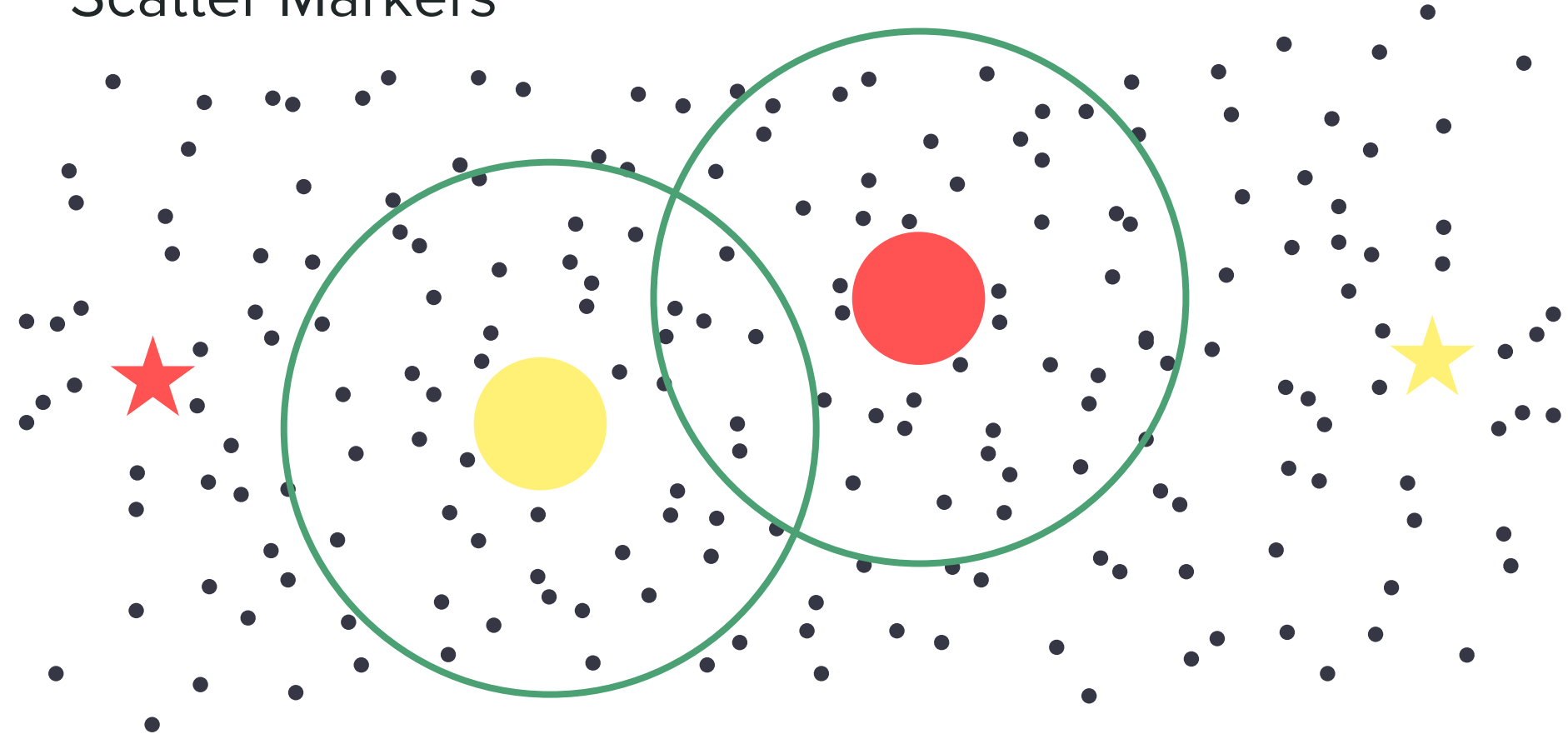
WebGL BioCrowds

Austin Eng

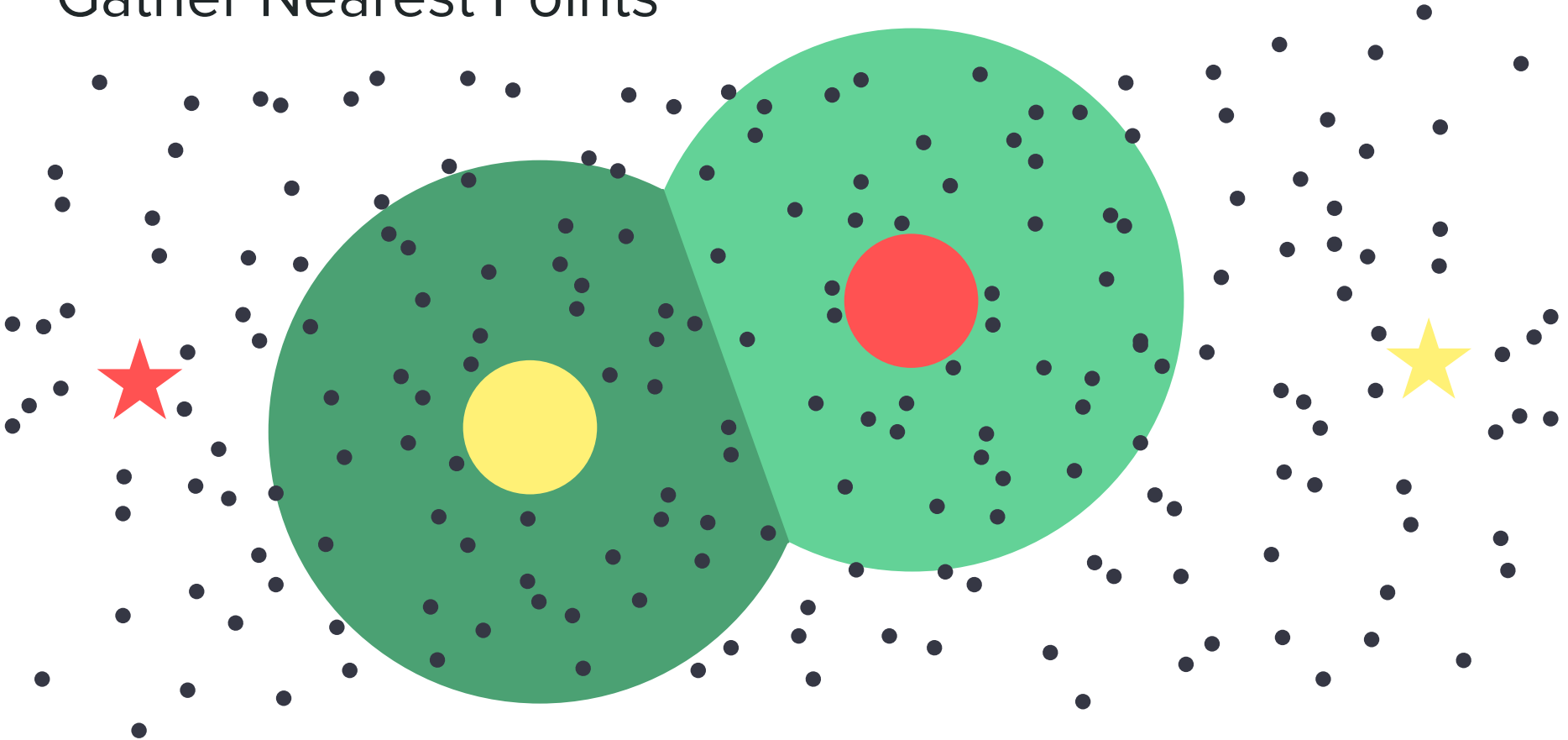
http://www.sciencedirect.com/science/article/pii/S0097849311001713

How it Works

Scatter Markers

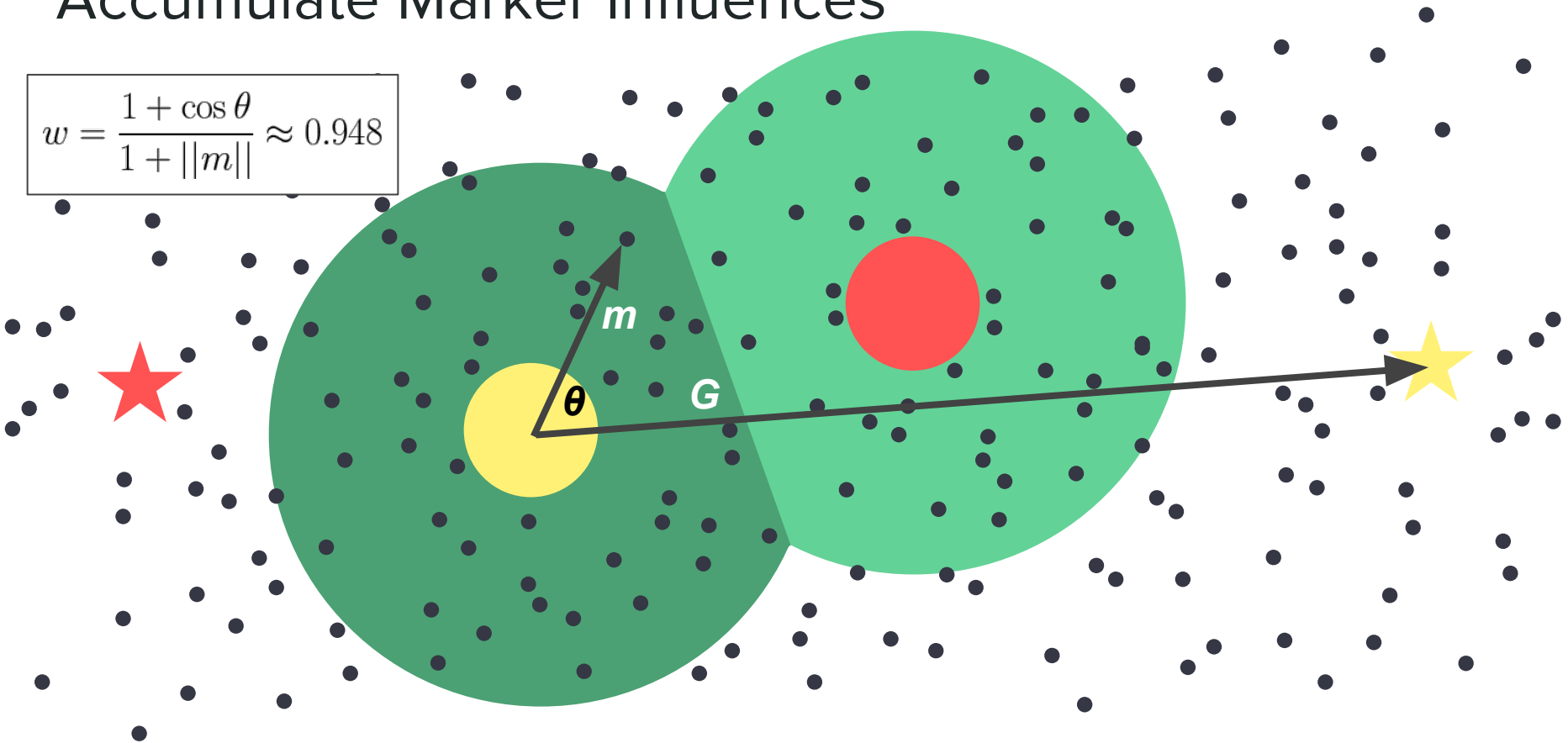


Gather Nearest Points



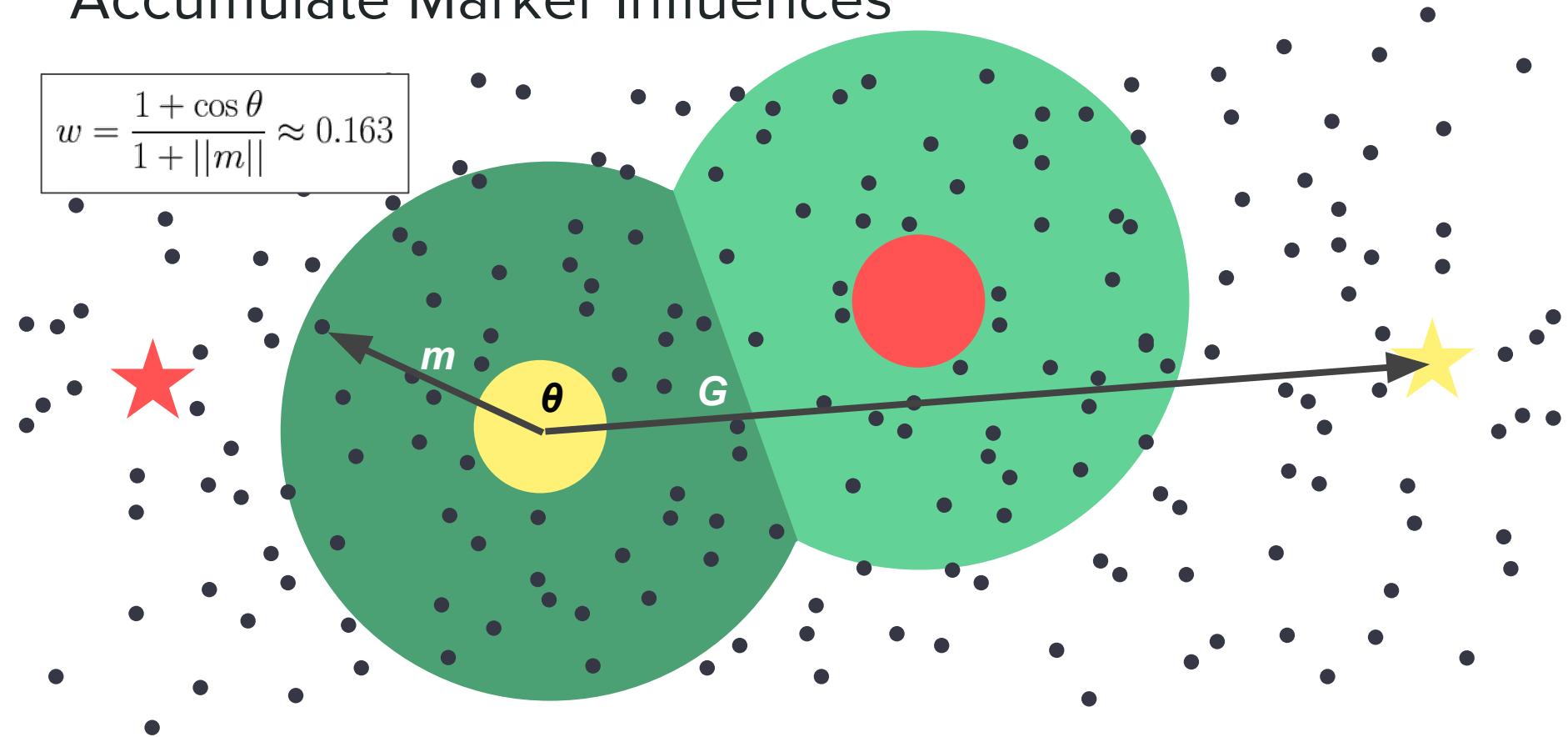
Accumulate Marker Influences

$$w = \frac{1 + \cos\theta}{1 + ||m||} \approx 0.948$$



Accumulate Marker Influences

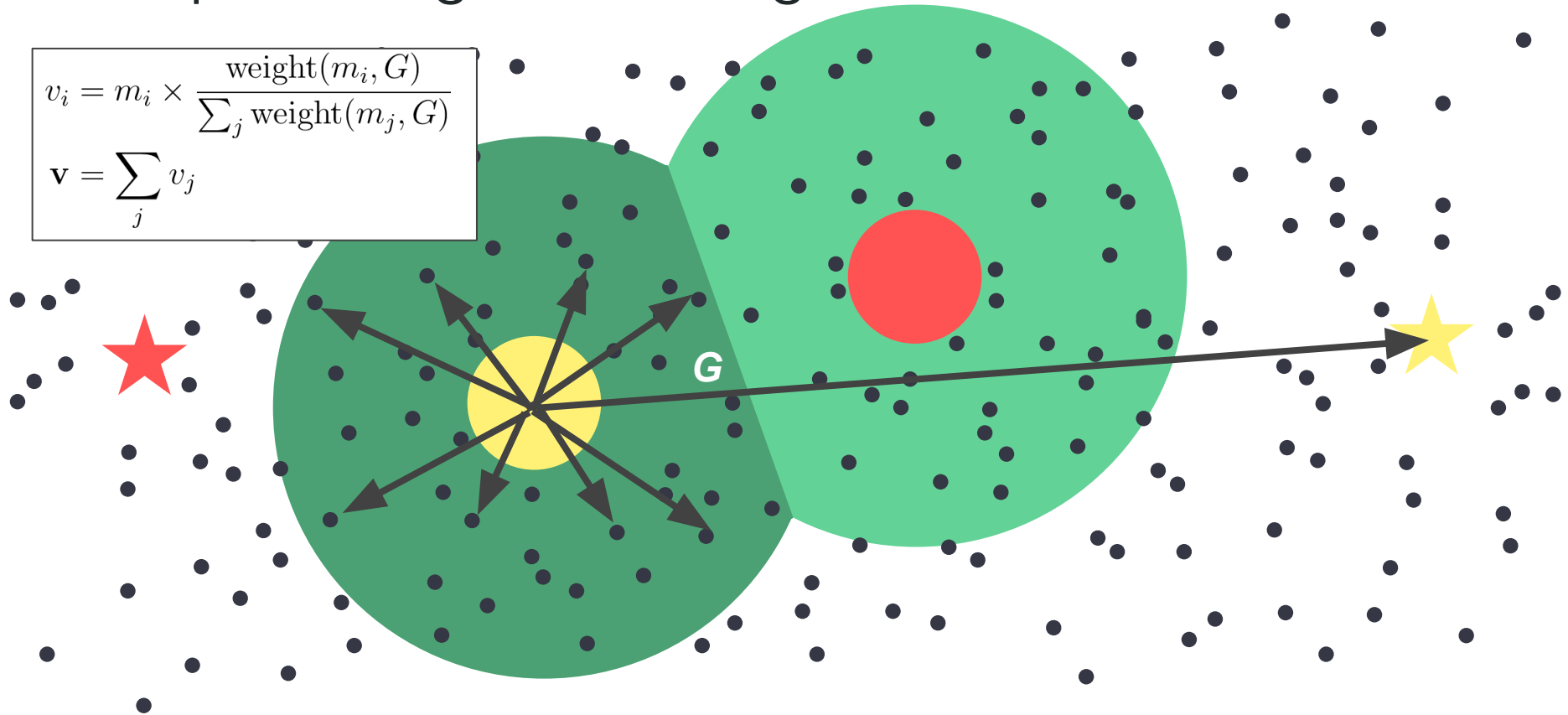
$$w = \frac{1 + \cos\theta}{1 + ||m||} \approx 0.163$$



Compute Weighted Average of Marker Influences.

$$v_i = m_i \times \frac{\text{weight}(m_i, G)}{\sum_j \text{weight}(m_j, G)}$$

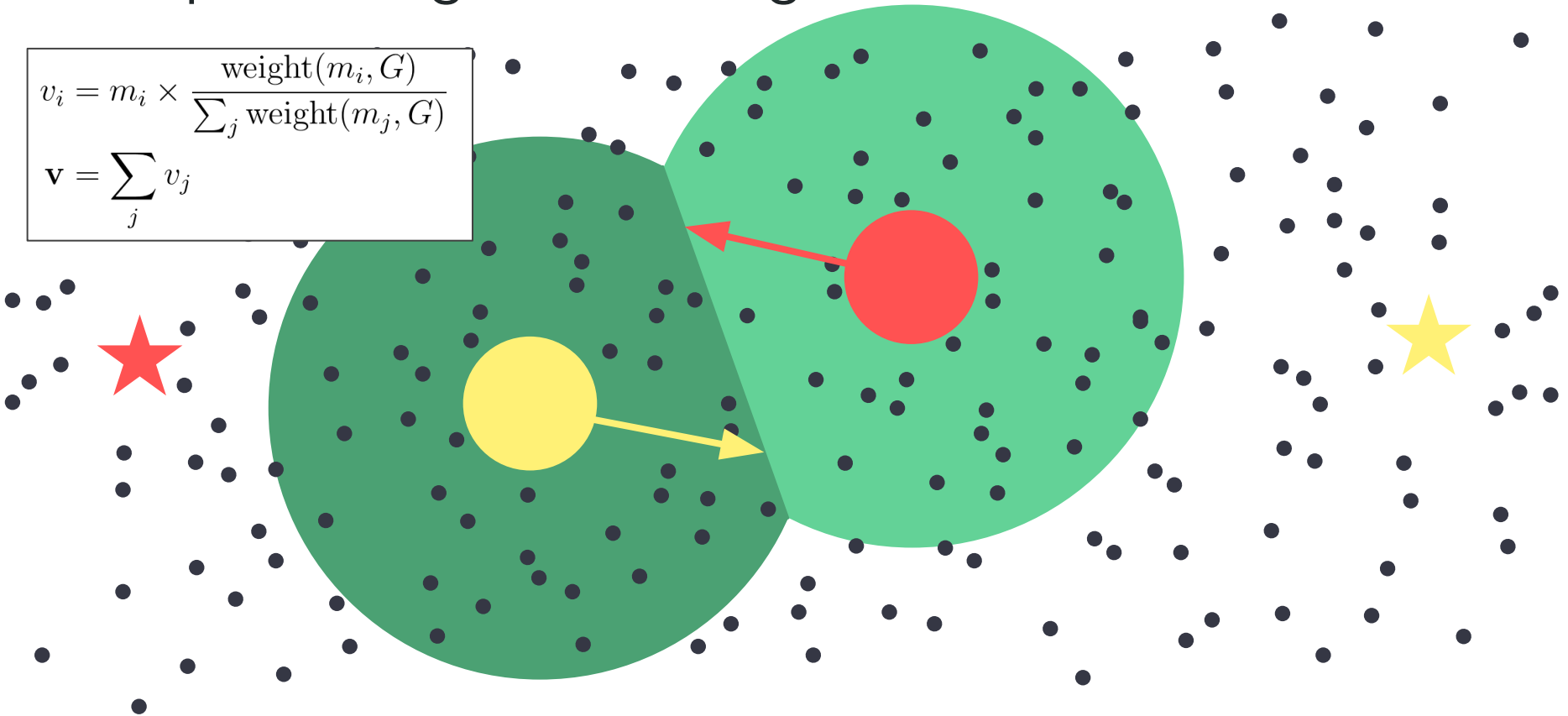
$$\mathbf{v} = \sum_j v_j$$



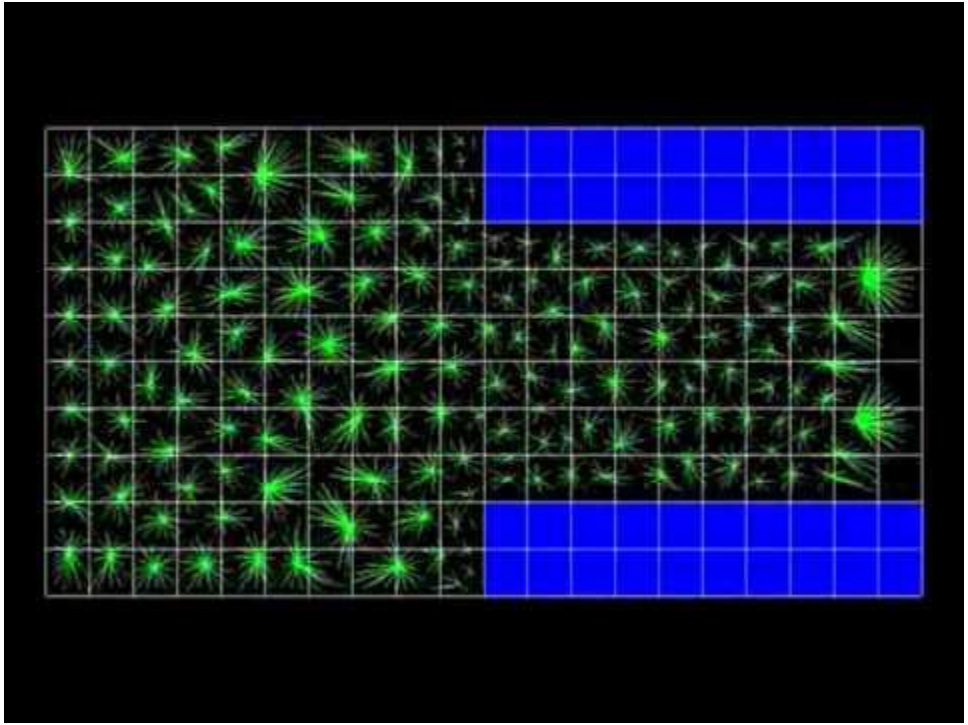
Compute Weighted Average of Marker Influences.

$$v_i = m_i \times \frac{\text{weight}(m_i, G)}{\sum_j \text{weight}(m_j, G)}$$

$$\mathbf{v} = \sum_j v_j$$



Evaluates at < 30fps



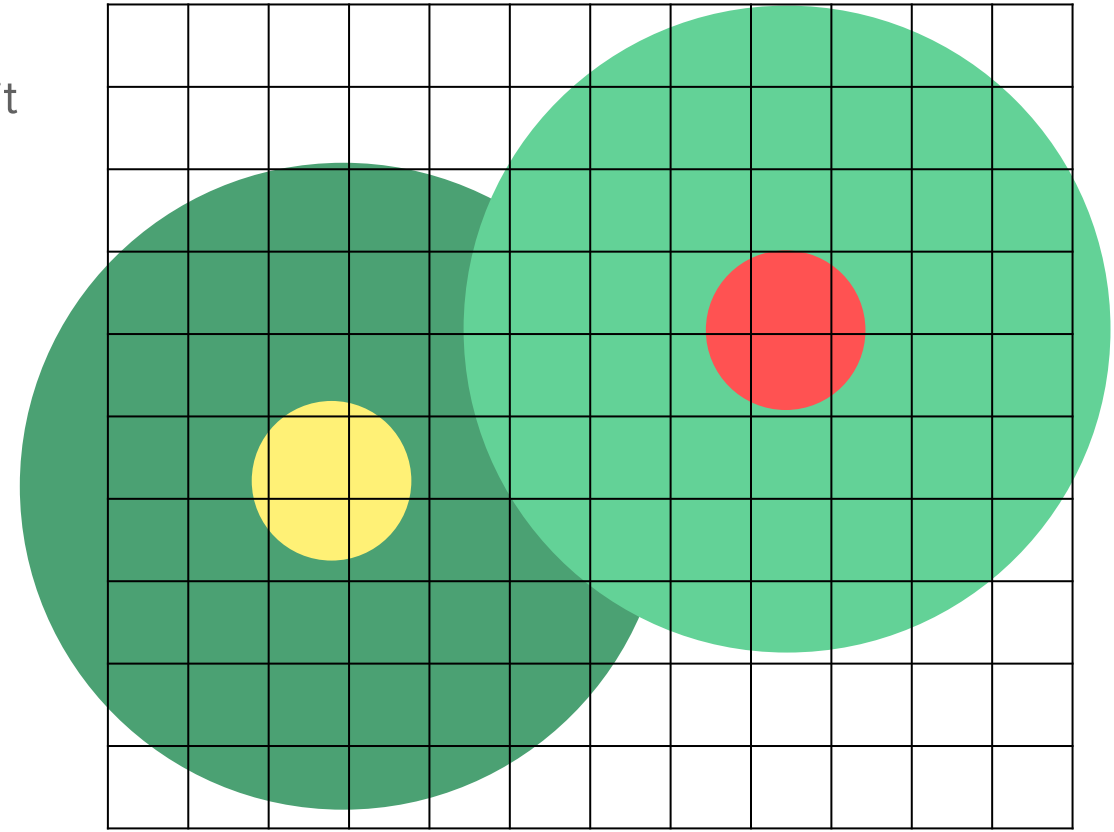
https://www.youtube.com/watch?v=aygTsijGkf0

BioCrowds in WebGL Shaders

- Pack agent data into image textures
  - Position
  - Goal
  - ID
- Eliminate nearest-neighbor search
- Don’t use markers

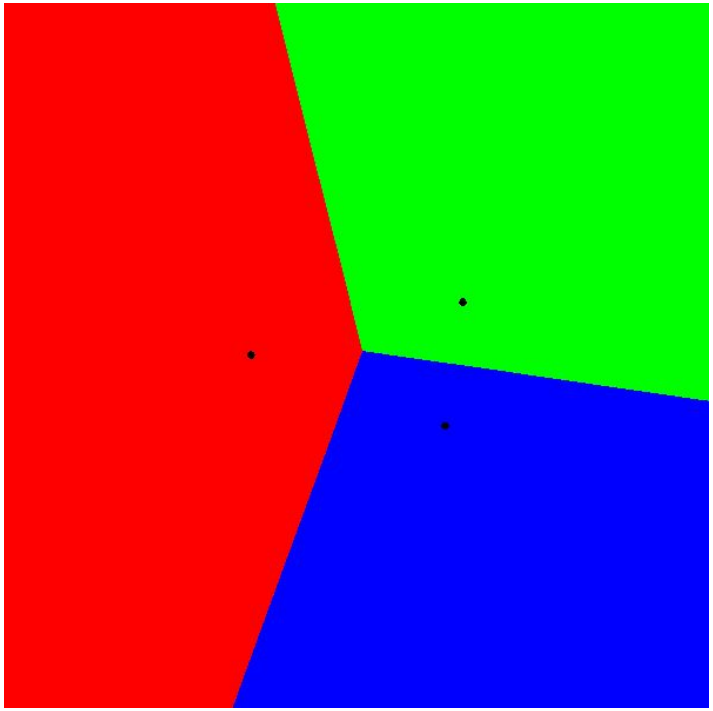
No Markers?

- Treat each pixel as an implicit marker
- Do we lose random behavior?
  - Not really. A noise texture can be easily added to perturb weights



Computing Voronoi in a Shader

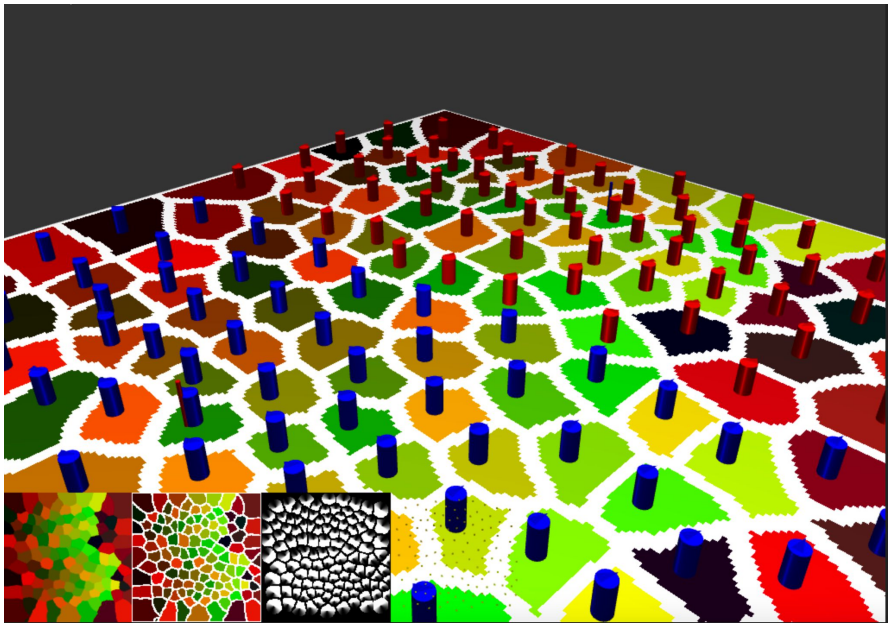
- Instance each agent as a uniquely colored vertical cone
- Other methods:
  - Jump Flooding - constant time regardless of the number of agents



http://nullprogram.com/blog/2014/06/01/

Refined Voronoi Diagram

- Agents have non-zero radius. Shrink voronoi cells to avoid interpenetration
- Run a shader that sets the pixel color to WHITE if there are at least two different colors within radius *r*



Computing Agent Velocity

- Check the color of the current fragment and look up the position and goal of the respective agent
- Write the resulting weight to a texture

- For each agent, accumulate marker influences and write out the computed velocity

Additional Features

- Nearest-agent search for chasing behavior
- Proximity field computation for avoidance behavior
- Procedural noise fields for random motion
- Arbitrary texture-driven “comfort” regions to guide agents away/towards regions

Improvements (now that I've taken GPU)

- Investigate Jump Flooding for Voronoi computation
  - Simultaneously compute distance-to-marker
- Keep all data on the GPU
  - The current implementation looks up agent velocities in a texture, updates positions on the CPU, and then copies position data back to a texture
  - Store positions and velocities only in a texture
  - Read from agent data texture to determine locations to draw agents
- Summed Area Tables for blur / voronoi refine