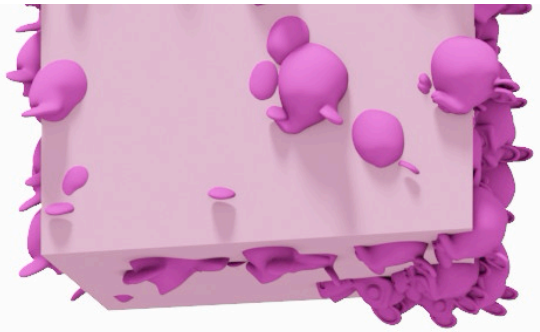


Particles and Flocking

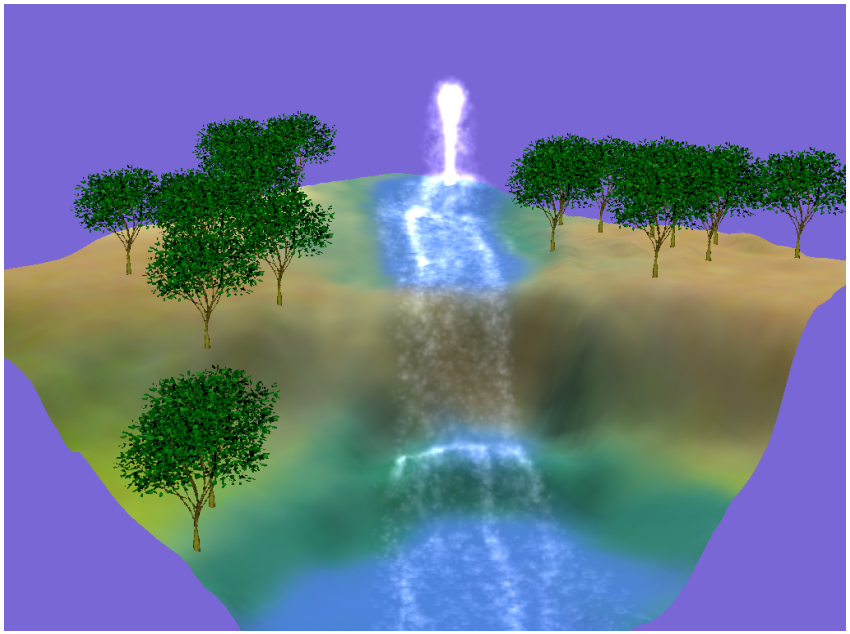
02563 Week 7

Andreas

Overview

- Image based effects:
  - Billboards, sky boxes, panoramas, point based rendering, billboard clouds
- Particle Systems
  - Particle simulation & rendering
  - GPU accelerated particles
  - Stateless particles
- Flocking (boids)

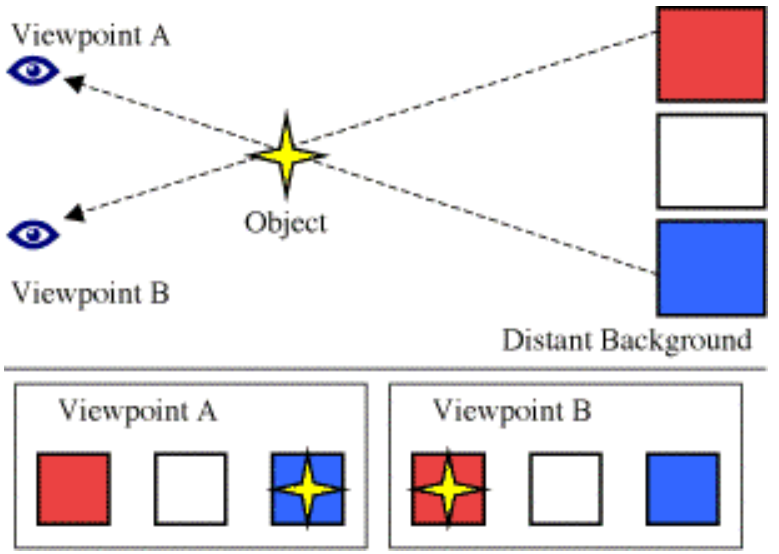
Image Based Effects
[Examples from Gamasutra and elsewhere on the Internet]



Parallax

“the effect whereby the position or direction of an object appears to differ when viewed from different positions”

- Greek:
  - alteration
- Shift in angle between objects due to
  - Motion parallax: moving observer.
  - Binocular parallax: eye separation
- Helps us to perceive depth.
- Annoying issue when using sprites



Rendering Far Away Objects

- Skybox: A simple cube displaying the distant environment.
  - We assume no parallax: Player always at centre of box – box moves with player.
  - Box needs only be large enough to contain scene.

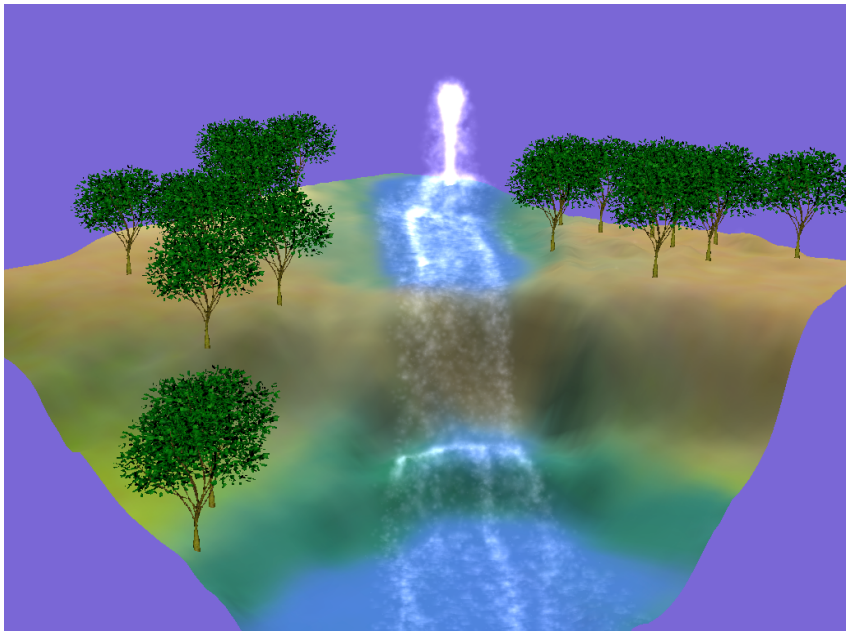
Rendering (relatively) Far Objects

- Quicktime VR is based on cylindrical or cubic projections.
- Again no parallax

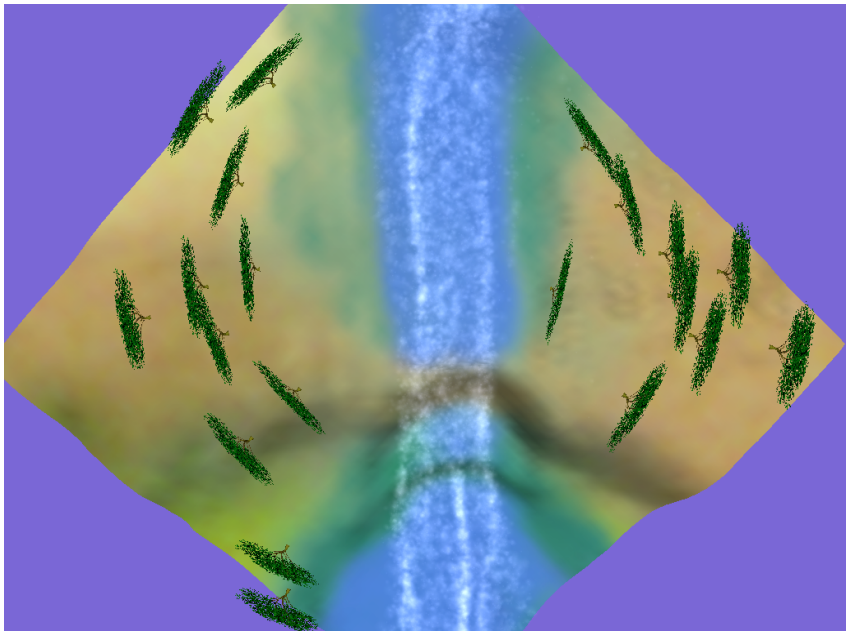
Billboards – a simple trick

Still no parallax (within individual billboard)

Trees seen from the side



Trees seen from above



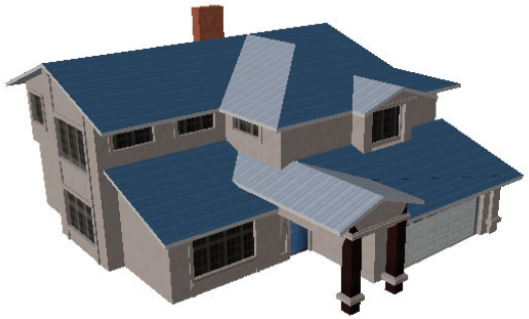
Billboard Clouds

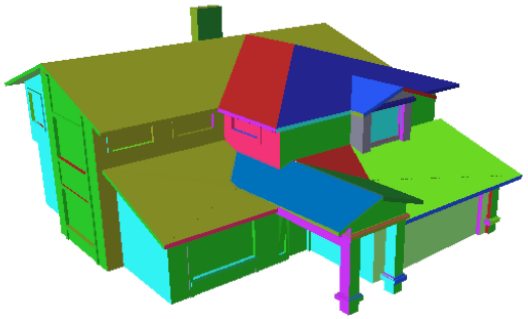
- For complex geometry which is not suitable for mesh decimation:
  - Fit planes to model.
  - Render triangles to best fit planes.
  - Use planes as billboards
  - Arguably we have parallax

Billboard Clouds

Orignal

1960 polygons





Clustered faces

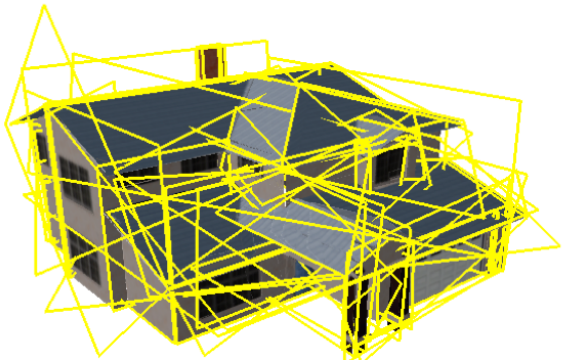
(a)

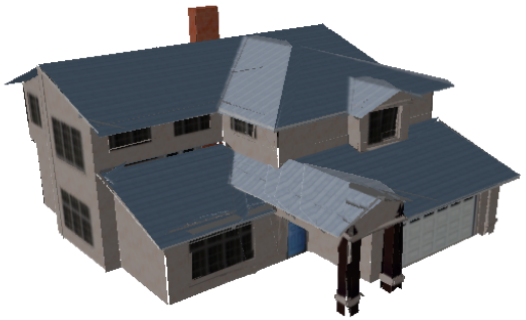
Collector mode set to **TEXTURE+QUAD**

(b)

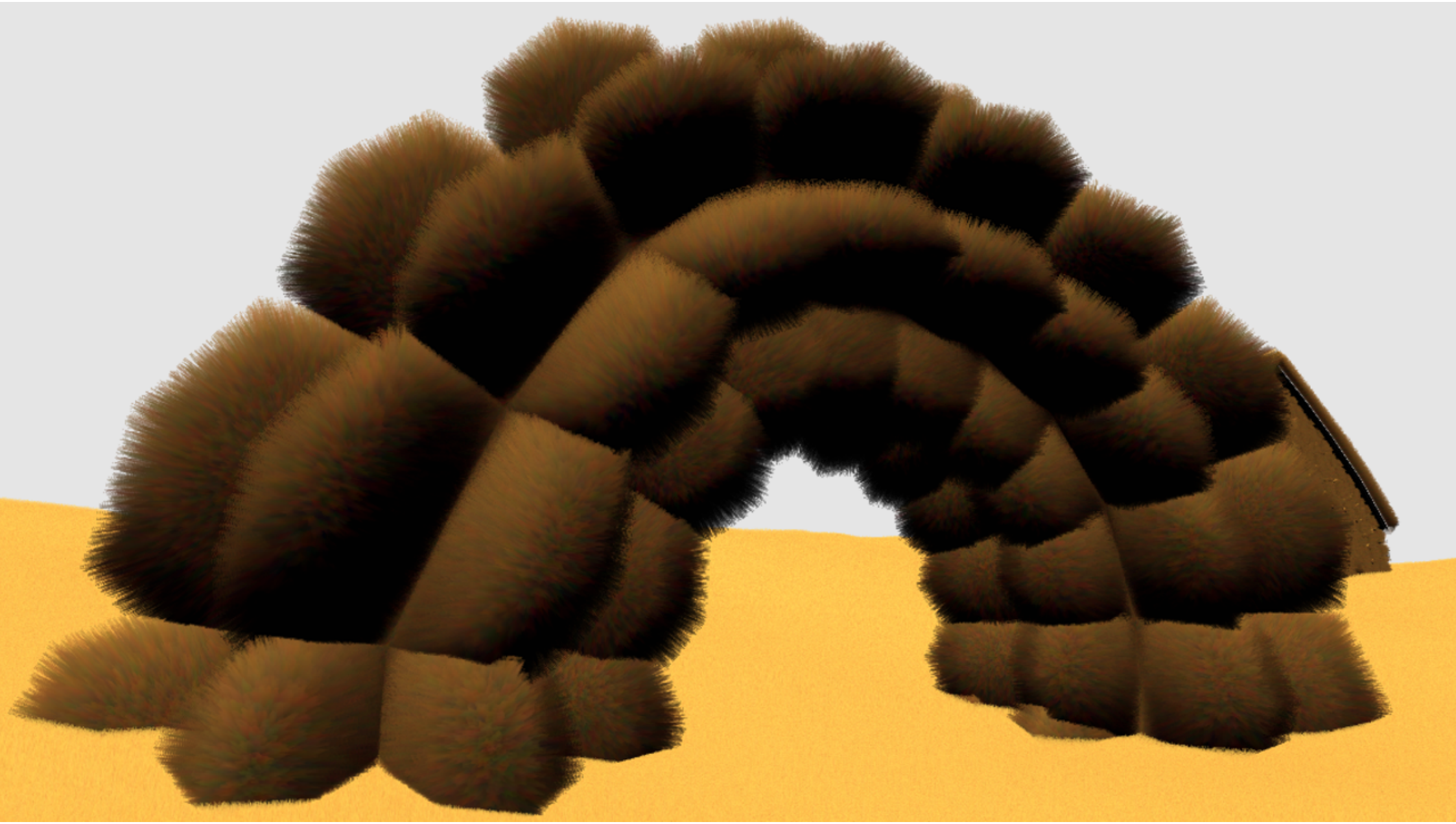
Collector mode set to **TEXTURE**

52 billboards



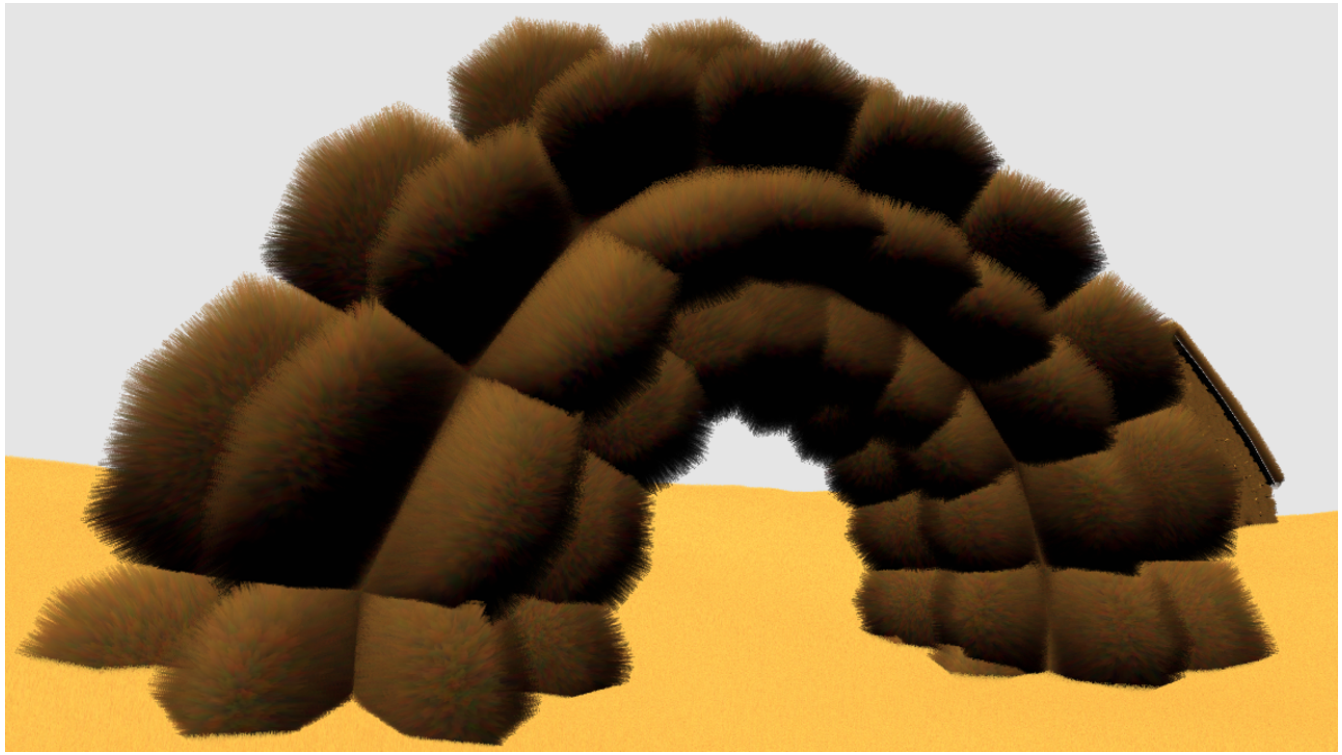


Billboard cloud



Fur

- Often based on shells: Billboards which are layered on top of each other
- This is similar to volume rendering



Point Rendering

Complex organic objects can be represented as point clouds

- Each point is rendered as a screen aligned quad (often using geometry shaders).
- Since the cloud is the object we have parallax.
- Advantage:
  - Entire representation is a single list of points.
  - Render a subset of this list - depending on distance.
  - Thus continuous LoD is obtained.

Point Rendering

First Use of Particles in a Movie

Wrath of Khan (Start Trek)

[Reeves]

Particle Systems
In a bit more detail

- Particles
  - are points that move
  - are rendered using small billboards
  - (usually) model dynamic phenomena
  - are either stateful or stateless
  - used in large numbers
  - possible to accelerate using the GPU
  - Sometimes obey the laws of physics

Particle System Modelling

- Individual Particles
  - Physical variables (position, velocity, ...)
  - Appearance variables.
- Particle System
  - An array of particles
  - Physical Constants (gravity, drag, ...)
  - Constants related to emitter

Particle Water Simulation

How to Simulate Water

Foam Particles really ...

- For each frame
  - Emit new particles
  - Update velocity V and position P (state)
    - Use drag to slow down particles
  - Check for and handle collisions (change velocity)
    - Collision response should take restitution and friction into account
  - Draw particles
    - Particles are blended onto screen

Particle Data

- Particle system contains
  - Vector of particles
  - Gravity acceleration, restitution, friction, drag.
  - Point where particles are emitted, initial velocity, etc.
  - Point size, texture parameters (if you use texture)
  - Terrain pointer
- Each particle contains
  - Position and velocity
  - Flag telling whether it is active

Particle Emission

- Do not emit all particles at once: Emit a few each update.
- Give particles a (slightly) random initial velocity and (possibly) a random position.
- Organic particles: Recycle particles outside some predefined box. Recycle particles that are still.

Particle Physics

Update Position and Velocity

- Based on velocity, $\mathbf{v}_t$, position, $\mathbf{p}_t$, acceleration $\mathbf{a}$, and drag $d \approx 0.01$

$$\mathbf{p}_{t+\Delta t} = \mathbf{p}_t + \Delta t \mathbf{v}_t$$
$$\mathbf{v}_{t+\Delta t} = (1 - d)\mathbf{v}_t + \Delta t \mathbf{a}$$

- Time step should be fixed
- Verlet integration could replace the Euler step,

$$\mathbf{p}_{t+\Delta t} = 2\mathbf{p}_t - \mathbf{p}_{t-\Delta t} + (\Delta t)^2 \mathbf{a}$$

  - No velocity in this formulation. No third order error.

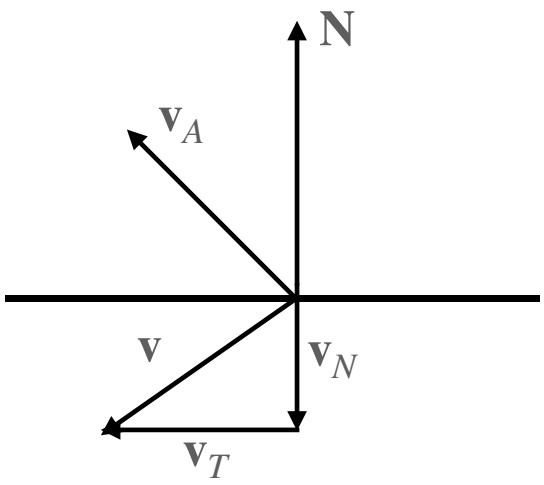
Collisions

Detection and Response

- For terrain: collision detected if $z < h(x, y)$ where $h(x, y)$ is terrain height and particle at $[x, y, z]$
- To compute velocity, $\mathbf{v}_A$, after collision:

$$\begin{aligned} \mathbf{v}_N &= \mathbf{N}(\mathbf{v} \cdot \mathbf{N}) \\ \mathbf{v}_T &= \mathbf{v} - \mathbf{v}_N \\ \mathbf{v}_A &= (1 - f)\mathbf{v}_T - r\mathbf{v}_N \end{aligned}$$

- Advice: Use no friction, f, but a bit of restitution, r.

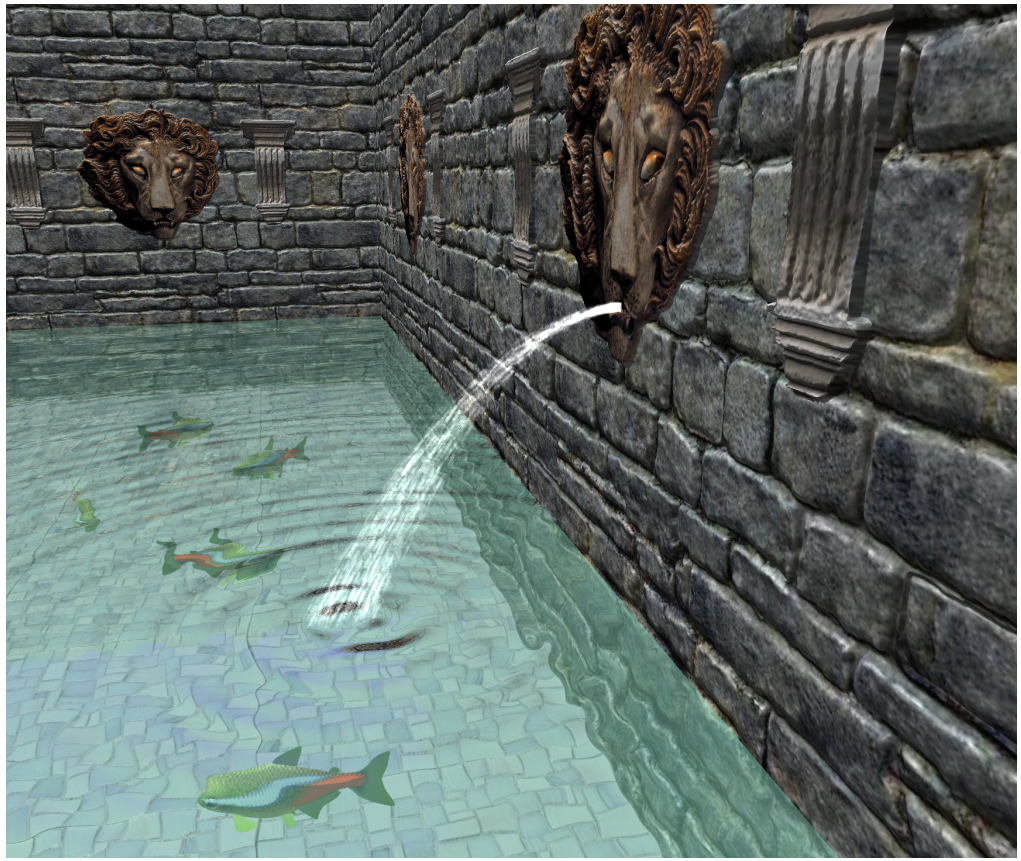


A More Sophisticated System

- ... would include
- Multiple particle systems with different behavior:
  - floating foam, spray, etc.
- Hierarchical particles [Reeves]
- Particles that relate to one another (spring lattice systems [Witkin])
- Dynamics: Buoyancy, wind, attractors, etc.
- Implemented on the GPU

Draw Particles

- Basic particle rendering
  - Disable lighting and Z-buffer update
  - Generate particle geometry in geometry shader or compute shader based on velocity and camera position
  - Enable additive blending and use low alpha value

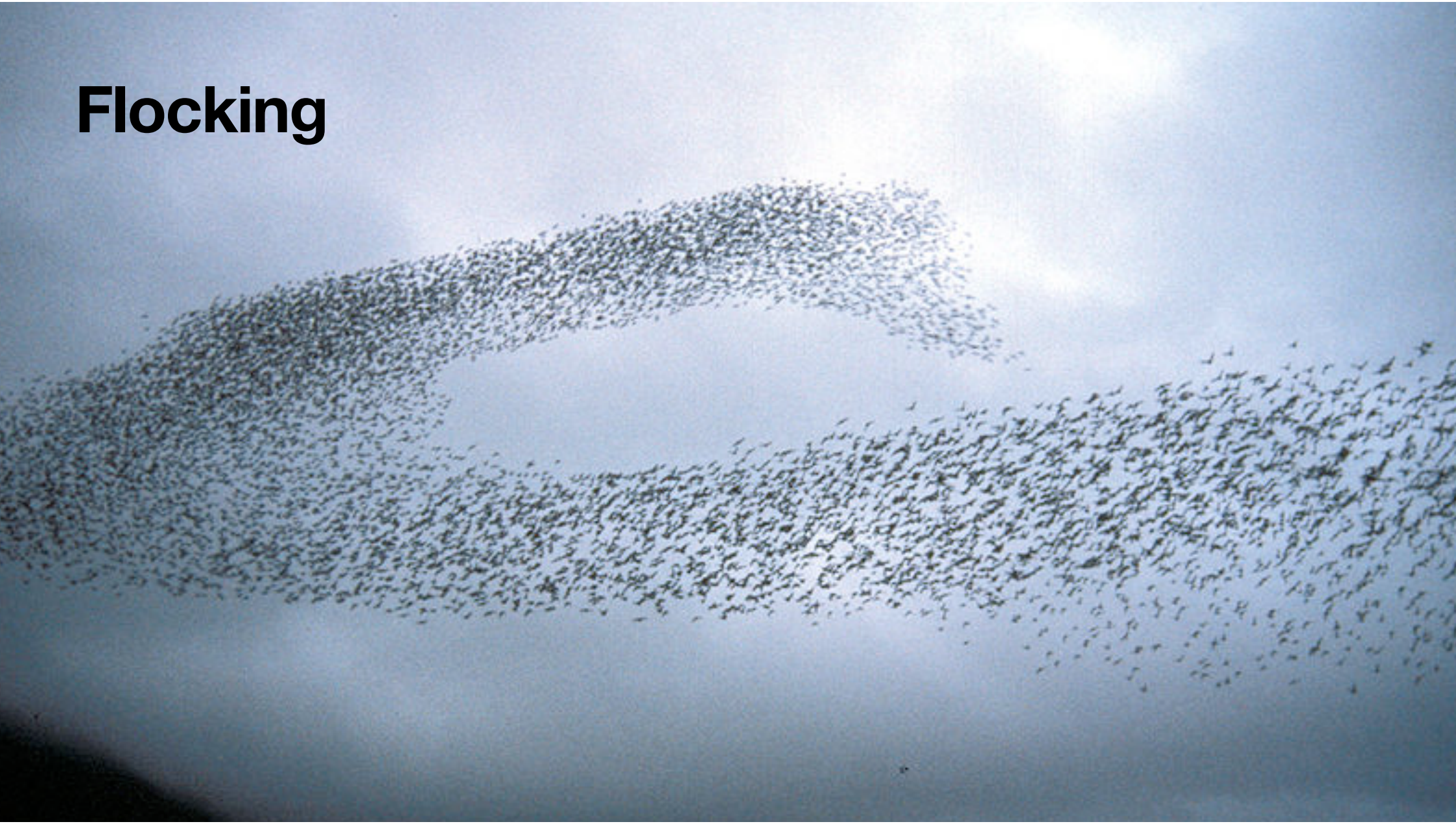


Stateless Particles

If the particles are completely determined from the initial state

- compute particle positions at all points in time just from the initial state.
- Get rid of the need for an additional pass for computing the particle position and velocity.
- Get rid of the buffers storing particle state.
- Caveat: Only trivial interaction with environment is possible!

Flocking



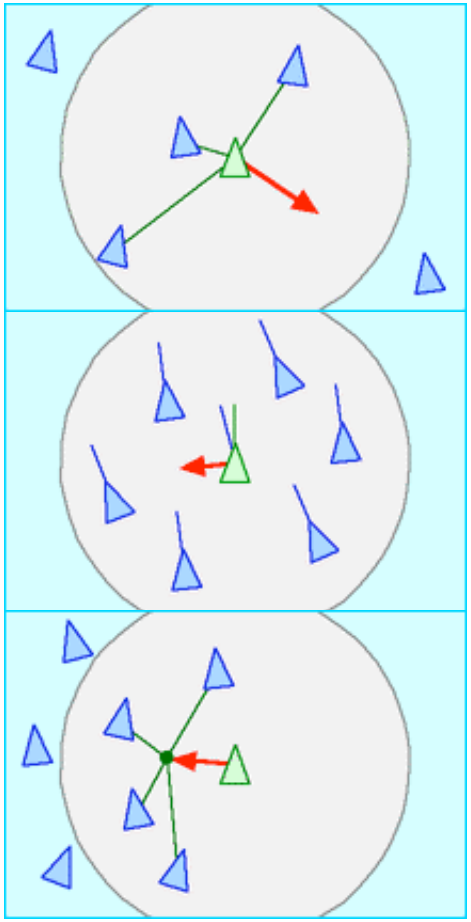
These darn boids seem to have a mind of their own!

- A boid (bird - pronounced with heavy accent) is a member of a flock (or school, or herd)
- We model a boid as a particle
  - which can see its neighbors and
  - has an orientation (coordinate system)
- Because each boid reacts to its neighbor a collective behavior results as an **emergent** phenomenon.

Flocking Principles

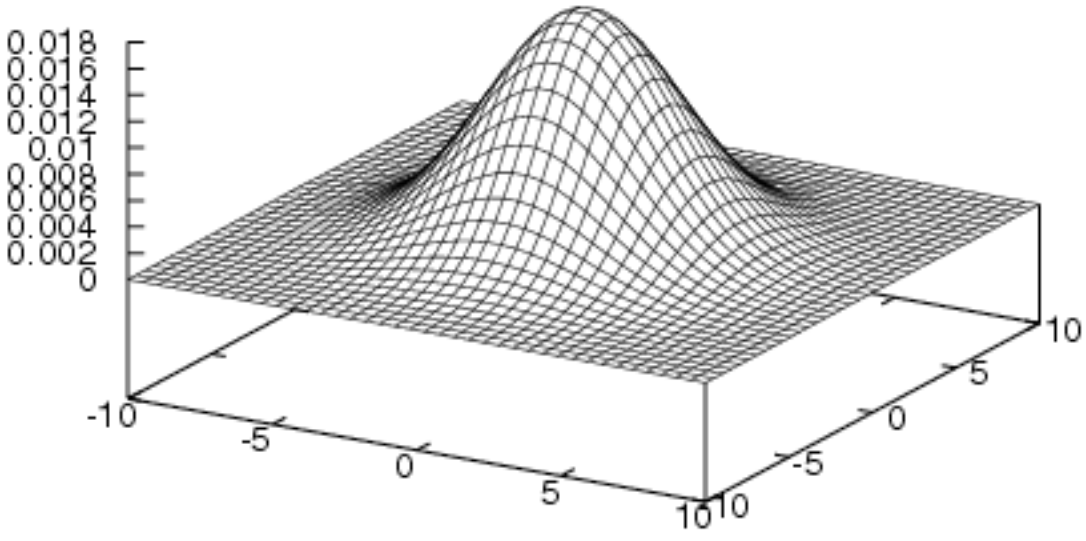
- Collision Avoidance: avoid collisions with nearby flockmates
- Velocity Matching: attempt to match velocity with nearby flockmates
- Flock Centering: attempt to stay close to nearby flockmates

  [Reynolds 1987]



Gaussian

- Can be used as a weighting function
- Gradient of a Gaussian is a vector useful for e.g. repelling.



References

- W. Reeves, Particle Systems — A Technique for Modeling a Class of Fuzzy Objects. https://dl.acm.org/doi/pdf/10.1145/357318.357320

- Witkin, SIGGRAPH ’97 Course notes. https://www.cs.cmu.edu/~baraff/sigcourse/notesc.pdf

- Nikita Lisitsa’s Video about a particle system implementation. https://youtube.com/watch?v=tea0qJx9_hs&feature=shares

Exercise

- Part 1: Particle Fountain
  - Create a particle system in Blender. Your scene should has an object called FountainHead which is intended to be the emitter of the particles.
- Part 2: Flocking
  - Your task is to make a flocking animation such that the fish a) avoid the basin walls, b) avoid swimming into each other and c) slightly align with each other : two nearby fish should influence each other's velocities. A number of constants have been defined for this task. Vary some of the parameters to study how the emergent behaviour changes.
- Hand in a single PDF with screen shots of both parts. Describe the parameters that you needed to set for the particle system and your implementation of the flocking for the fish. Include source code for the script used for flocking.