Lecture 12

Collision detection and handling

Lab progress could be better?

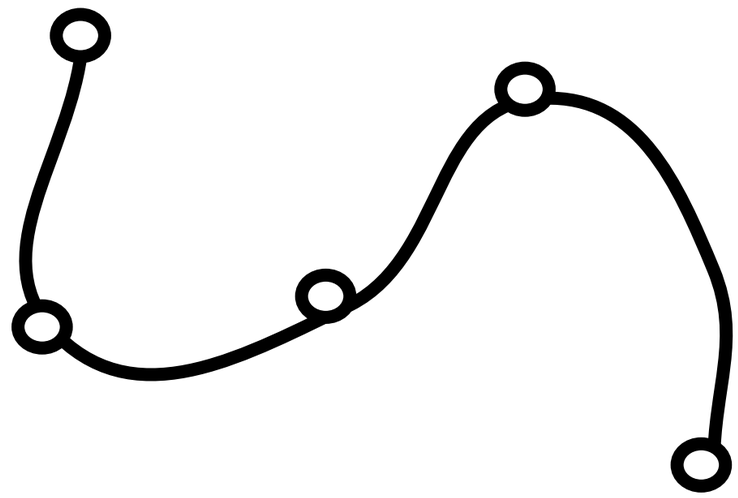
Too many stuck on hard to find bugs

Lab assistants will be instructed to give more hands-on help in such cases

- **A bit more on particle systems**
- **Collision detection**
- **Polyhedra-polyhedra, broad phase-narrow phase, SAT etc**
- **Global phase**
- **Simplified case**
- **Collision handling**

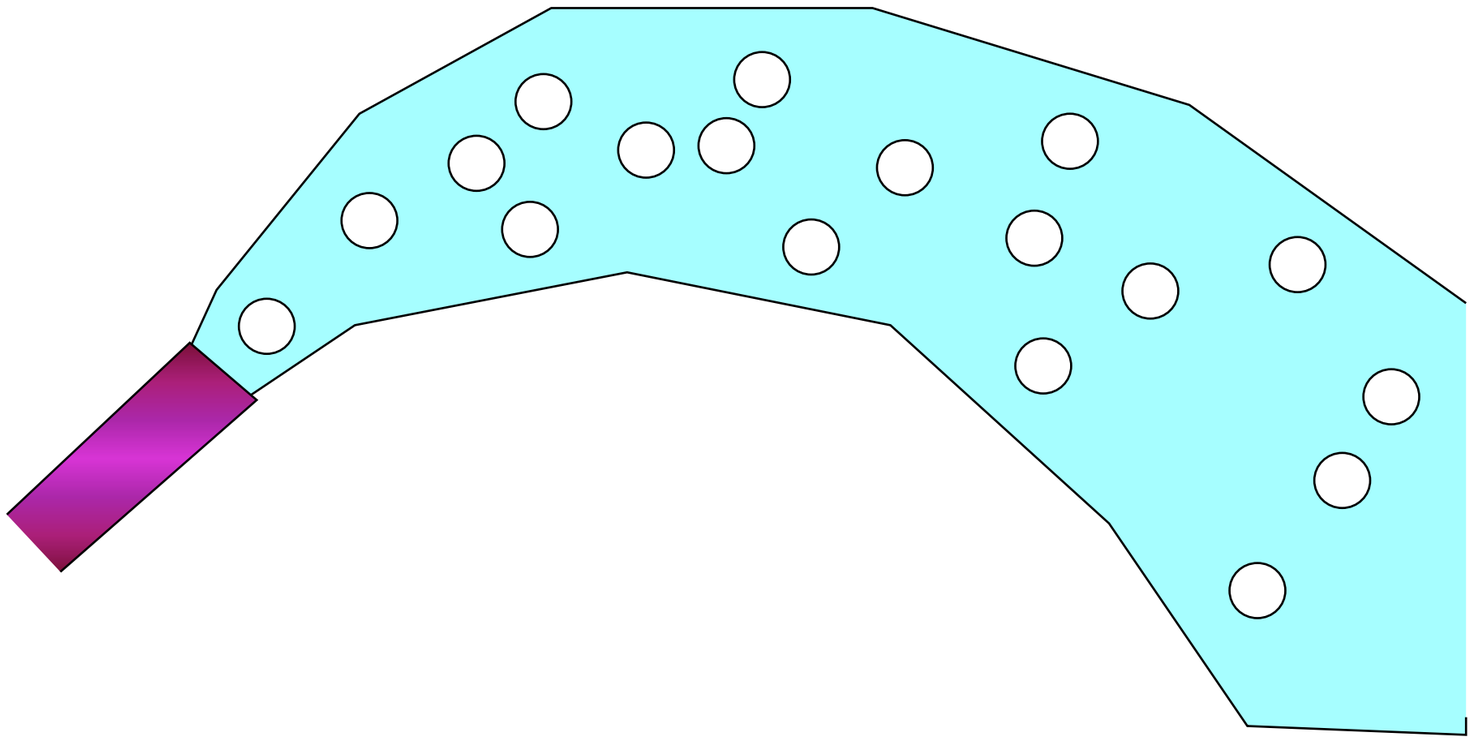
Animation paths

Use Catmull-Rom splines! Predictable, smooth, continuous!



Particle system

Example: Water



Types of particle system

- **Independent particles**
- **Free moving interacting particles**
- **Connected particles**

Types of particle system

- **Independent particles** — Easy!
- **Free moving interacting particles** — Hard!
- **Connected particles** — Moderate!

Particle system

- **Initial position**
- **Initial speed (usually with some randomness))**
- **Movement (usually independent, physically realistic)**
- **Termination rule (e.g. hits ground, fades away after some time...)**

Particle system

Movement according to fundamental physics:

acceleration = gravity + forces/mass
speed = speed + acceleration
position = position + speed

“Euler integration”

Particle system on GPU

CPU-driven particle systems OK up to a certain size

Data transfer (new positions) of all particles can be a bottleneck

Can the whole particle system be computed on the GPU?

Texture based particle systems

Use textures to store x, y, z, dx, dy, dz

Store as color components (r, g, b)

Needs advanced texturing features (render to texture, floating-point buffers)

Particles as billboards. Each polygon must identify its particle data.

Separate compute kernels for particle systems

CUDA, OpenCL, Compute shaders

Free choice of data formats

Less integration with the OpenGL pipeline

Drawing particle systems

Large number of very simple models (billboards)!

Modest demands on GPU, but very large number of function calls!

Solution: Instancing

Instancing

Draw a large number of the same model!

Each instance has an index, the instance number.

glDrawArraysInstanced(GL_TRIANGLES, 0, 3, 10);

draws a triangle 10 times!

gl_InstanceID tells the shader which instance we have. Use for affecting position.

Billboard instancing demo

One single call to glDrawArraysInstanced

Position trivially affected by gl_Instance ID

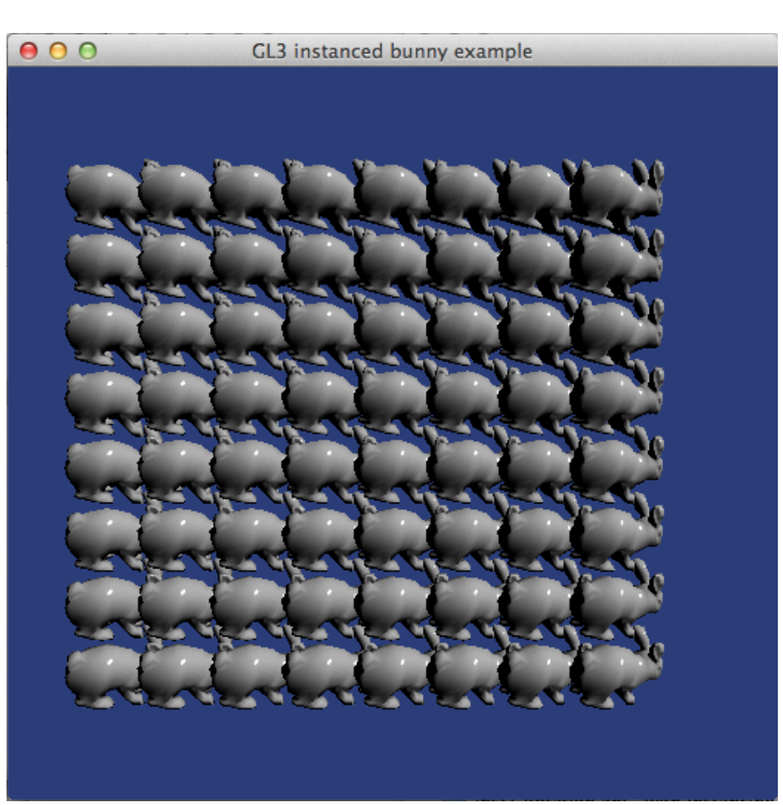
```
#version 150

in  vec3 in_Position;
uniform mat4 myMatrix;
uniform float angle;
uniform float slope;
out vec2 texCoord;

void main(void)
{
 mat4 r;
 float a = angle + gl_InstanceID * 0.5;
 float rr = 1.0 - slope * gl_InstanceID * 0.01;
 r[0] = rr*vec4(cos(a), -sin(a), 0, 0);
 r[1] = rr*vec4(sin(a), cos(a), 0, 0);
 r[2] = vec4(0, 0, 1, 0);
 r[3] = vec4(0, 0, 0, 1);
 texCoord.s = in_Position.x+0.5;
 texCoord.t = in_Position.y+0.5;
 gl_Position = r * myMatrix *  vec4(in_Position,
1.0);
}
```

Instancing complex models

Less significant; A more complex model puts enough load on the system to hide the impact of instancing.



Basic: Start on CPU
Advanced: Go for GPU acceleration

Performance is important, but GPU based particle systems are beyond basic course goals.