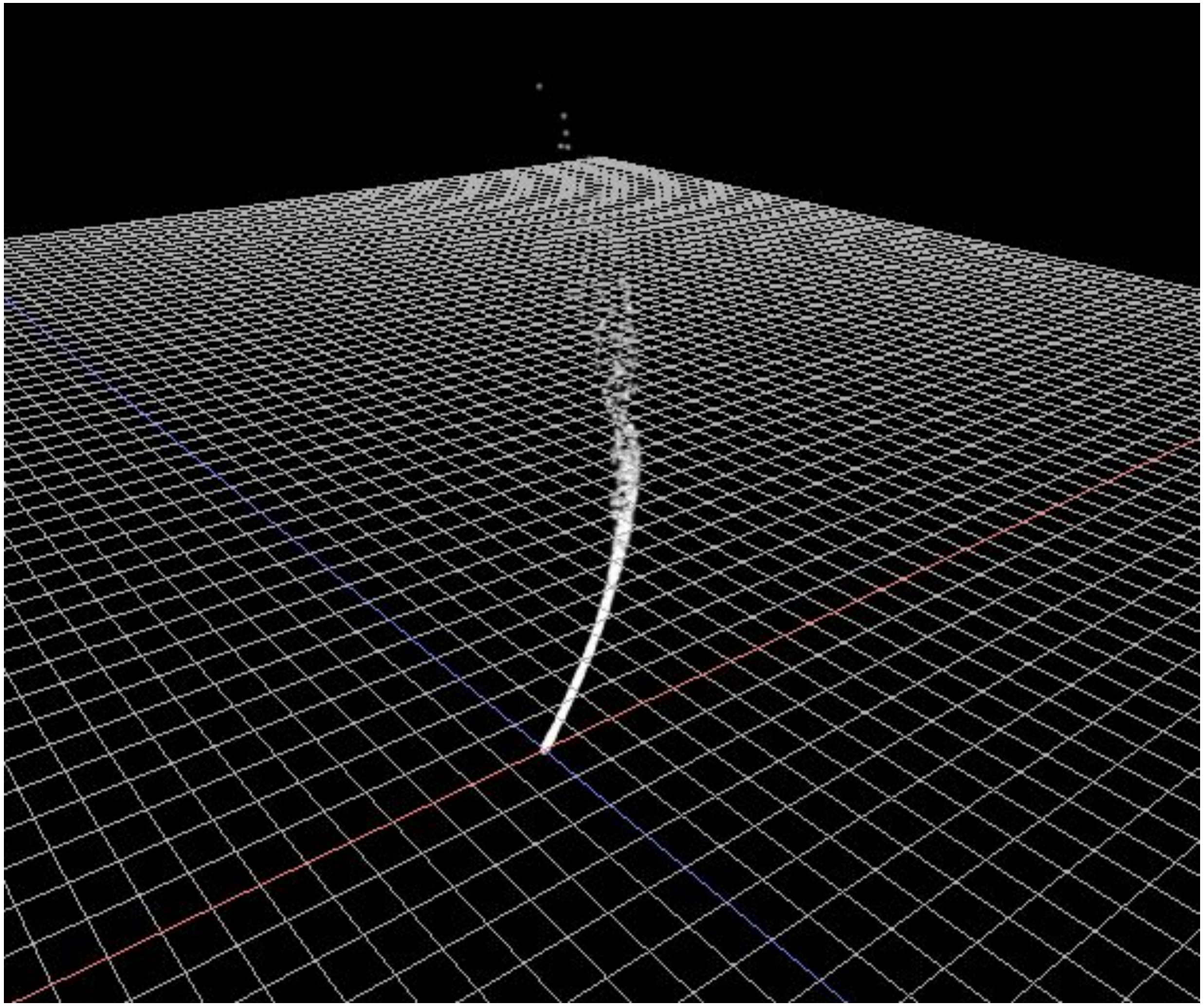
MVD: Advanced Graphics 2

24 - Particles

alunthomas.evans@salle.url.edu

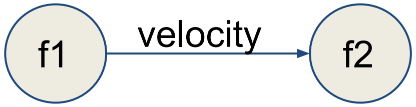
laSalle ENG
Universitat Ramon Llull

Particles



Particle Systems

A particle system is essential a set of rules that govern the change in position of a point ('particle') between frames



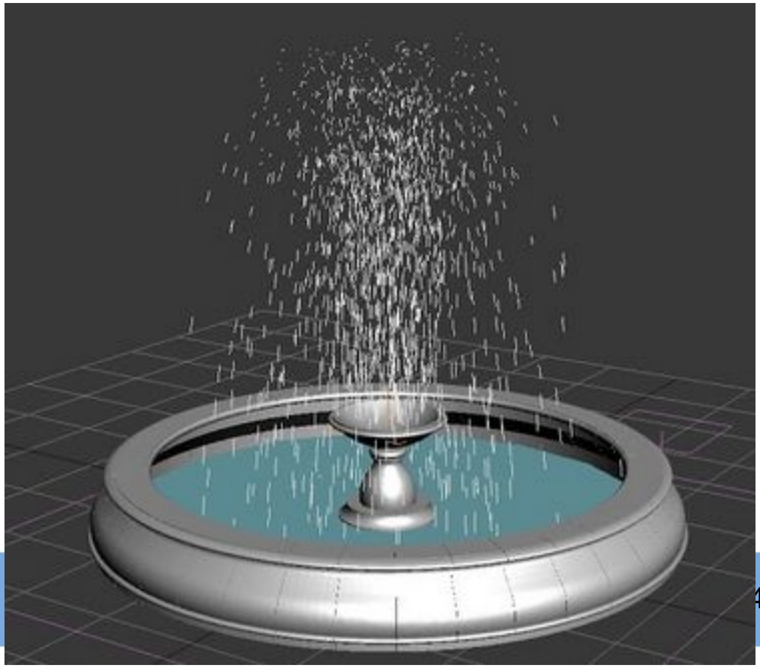
every particle has a velocity, every frame you add that velocity to the particles position, and *hey presto*

Particle Systems

The velocity equation differs depending on the effect required

e.g. **smoke**, slow upwards, slight random left and right

fountain - initial guided velocity, then apply gravity every frame



Particle textures

Can apply textures to particles (with transparency and blending) to create effect

Calculating particles movement - CPU

The ‘classic particle system is CPU based.

Particles are stored as arrays of positions, every frame CPU applies movement equations.

Updated positions are passed to GPU buffers (which are created using GL_DYNAMIC_DRAW or similar)

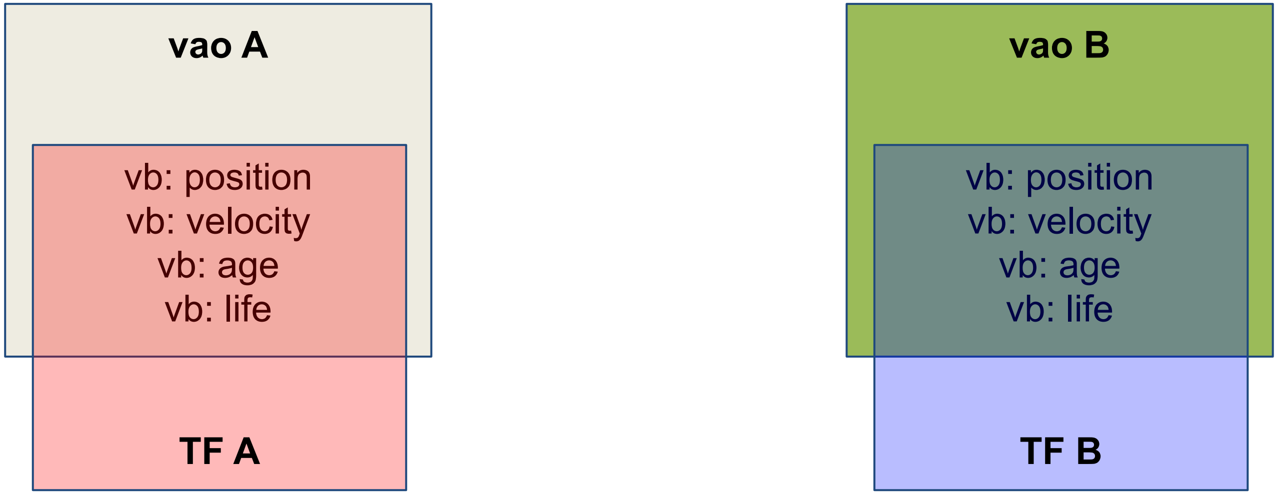
Calculating Particle Movement - GPU

GPU is obviously a better choice due to parallel architecture

But, traditionally, GPUs draw to screen buffer (pixels), and can't update buffers.

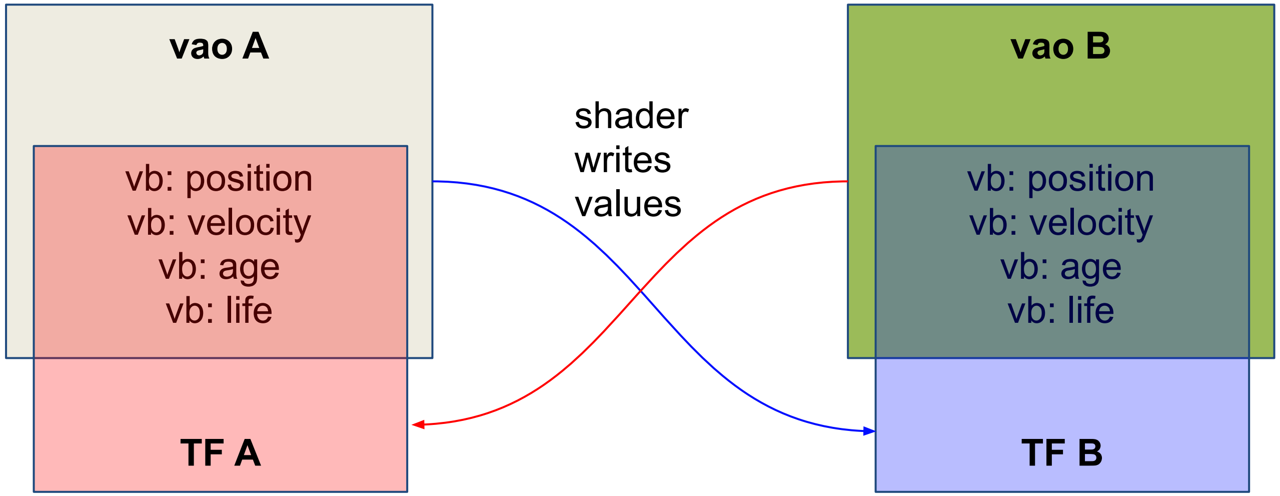
But modern graphics APIs provide a mechanism to do just that, via ***transform feedback***

Transform feedback buffers overview



Note sharing of vertex buffers

Transform feedback rendering



vaoB shares same buffers as tfB

If we read vaoA and write to to tfB, when we read vaoB we read the results of rendering vaoA with the shader

Shader must write variables

Look at particles.vert - see four new ‘out’ variables?

When compile shader, must tell it that we are using transform feedback varyings (see Shader::makeShaderProgram).

Compile like this, and enable setting point size in shader:

```
//names of values we want to save in second buffer
const GLchar* feedback_varyings[] = { "v_vertex", "v_velocity", "v_age", "v_life" };

//create shader
particle_shader_ = new Shader("data/shaders/particles.vert", "data/shaders/particles.frag", 4, feedback_varyings);

//tell openGL we want to be able to set point size in shader
glEnable(GL_PROGRAM_POINT_SIZE);
```

Initializing data

Random lifespan (between 0 - 9 seconds)

Initial age < -9 (will understand why later)

```
std::vector<GLfloat> vertices; vertices.resize(num_points * 3);
std::vector<GLfloat> velocities; velocities.resize(num_points * 3);
std::vector<GLfloat> ages; ages.resize(num_points);
std::vector<GLfloat> lives; lives.resize(num_points);

for (int i = 0; i < num_points; i++) {
    vertices[i] = 0; velocities[i] = 0; // these will be regenerated in the shader so are kind of useless here
    ages[i] = -9.01f; // this is a negative value larger than the lifetime of the particle to force reset in shader
    lives[i] = (float)(rand() % 9000) / 9000.0f; // the only value which is actually set here
}
```

Create vaos and tfs

```
//create vertex arrays - class member variables as we need to access them in update
glGenVertexArrays(1, &vaoA_);
glGenVertexArrays(1, &vaoB_);

//create transform feedback ids - these are just handles that point to existing vertex buffers
//again class member variables, as we need to access them in update
glGenTransformFeedbacks(1, &tfA_);
glGenTransformFeedbacks(1, &tfB_);
```

Create buffers for vaoA + tfA

Bind vaoA

Create four vbos (position, velocity, age, lifespan) and fill with data from arrays using glBufferData

vao A

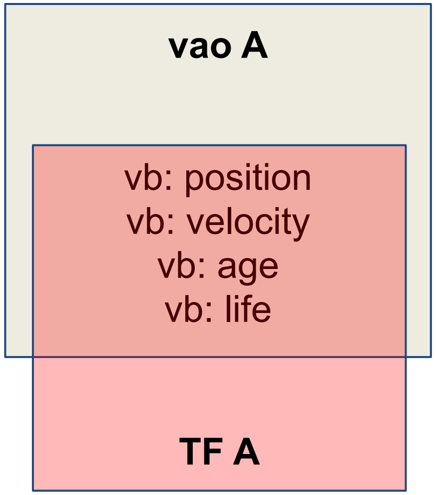
vb: position
vb: velocity
vb: age
vb: life

use GL_STREAM_COPY instead of GL_STATIC_DRAW

see [spec]

Bind tf A to vbos

New code




```
//output positions
glBindTransformFeedback(GL_TRANSFORM_FEEDBACK, tfA_);
//bind transform array to vboA buffers
glBindBufferBase(GL_TRANSFORM_FEEDBACK_BUFFER, 0, vb_A_pos);
glBindBufferBase(GL_TRANSFORM_FEEDBACK_BUFFER, 1, vb_A_vel);
glBindBufferBase(GL_TRANSFORM_FEEDBACK_BUFFER, 2, vb_A_age);
glBindBufferBase(GL_TRANSFORM_FEEDBACK_BUFFER, 3, vb_A_lif);
```

the numbers 0-3 correspond to order that we passed the names when compiling the shader:

```
//names of values we want to save in second buffer
const GLchar* feedback_varyings[] = { "v_vertex", "v_velocity", "v_age", "v_life" };
```

Do the same for vaoB and tfB

```
//now fill array B, same as A
glBindVertexArray(vaoB_);
GLuint vb_B_pos, vb_B_vel, vb_B_age, vb_B_lif;

glGenBuffers(1, &vb_B_pos);
glBindBuffer(GL_ARRAY_BUFFER, vb_B_pos);
glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(float), &(vertices[0]), GL_STREAM_COPY);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);

//velocity buffer
glGenBuffers(1, &vb_B_vel);
glBindBuffer(GL_ARRAY_BUFFER, vb_B_vel);
glBufferData(GL_ARRAY_BUFFER, velocities.size() * sizeof(float), &(velocities[0]), GL_STREAM_COPY);
glEnableVertexAttribArray(1);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 0, 0);
//age buffer
glGenBuffers(1, &vb_B_age);
glBindBuffer(GL_ARRAY_BUFFER, vb_B_age);
glBufferData(GL_ARRAY_BUFFER, ages.size() * sizeof(float), &(ages[0]), GL_STREAM_COPY);
glEnableVertexAttribArray(2);
glVertexAttribPointer(2, 1, GL_FLOAT, GL_FALSE, 0, 0);
//life buffer
glGenBuffers(1, &vb_B_lif);
glBindBuffer(GL_ARRAY_BUFFER, vb_B_lif);
glBufferData(GL_ARRAY_BUFFER, lives.size() * sizeof(float), &(lives[0]), GL_STREAM_COPY);
glEnableVertexAttribArray(3);
glVertexAttribPointer(3, 1, GL_FLOAT, GL_FALSE, 0, 0);


glBindTransformFeedback(GL_TRANSFORM_FEEDBACK, tfB_);
glBindBufferBase(GL_TRANSFORM_FEEDBACK_BUFFER, 0, vb_B_pos);
glBindBufferBase(GL_TRANSFORM_FEEDBACK_BUFFER, 1, vb_B_vel);
glBindBufferBase(GL_TRANSFORM_FEEDBACK_BUFFER, 2, vb_B_age);
glBindBufferBase(GL_TRANSFORM_FEEDBACK_BUFFER, 3, vb_B_lif);
```

Rendering

Need to enable blending and disable depth test

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glDepthMask(GL_FALSE);
```

also set model matrix and ViewProjection Uniforms

Rendering - switching between VAOs

Declare a member variable to store ‘current’ VAO source

Alternate between VAO every frame.

vaoA + tfB

vaoB + tfA

```
if (vaoSource == 0) {
    glBindVertexArray(vaoA_);
    glBindTransformFeedback(GL_TRANSFORM_FEEDBACK, tfB_);
    vaoSource = 1;
}
else {
    glBindVertexArray(vaoB_);
    glBindTransformFeedback(GL_TRANSFORM_FEEDBACK, tfA_);
    vaoSource = 0;
}
```

Rendering with transform feedback

No index buffer, so call draw arrays

```
glBeginTransformFeedback(GL_POINTS);

glDrawArrays(GL_POINTS, 0, 1000);

glEndTransformFeedback();
```

Copying Transform Feedback in Shader

For transform feedback shader to compile, you must write declare ‘out’ variables for the feedback variables, and **also write to them**

```
layout(location = 0) in vec3 a_vertex;
layout(location = 1) in vec3 a_velocity;
layout(location = 2) in float a_age;
layout(location = 3) in float a_life;

out vec3 v_vertex;
out vec3 v_velocity;
out float v_age;
out float v_life;
```

```
v_velocity = a_velocity;
v_vertex = a_vertex + vec3(0,0.001,0);
v_age = a_age;
v_life = a_life;
```

this example moves all vertices upwards

More realistic physics

Add velocity to position

Decrease velocity every frame:

```
//move particle
v_velocity = a_velocity - vec3(0.0, 0.005, 0.0);
v_vertex = a_vertex + v_velocity * 0.005;
v_age = a_age;
v_life = a_life;
```

But initial velocity = 0 so this doesn't work. Must reset particle

Reseting particle through time

Send time to shader:

```
particle_shader_->setUniform(U_TIME, (float)glfwGetTime());
```

returns time (in seconds) since start of application

Alun Evans – alunthomas.evans@salle.url.edu

Resetting particle through time

Calculate age in shader.

```
float age = u_time - a_age;

if (age > a_life) { //it will definitely be for the first iteration
    //reset particle
    //...
}
```

a_age initial value was -9.01. a_life was between 0->9

u_time will be > 0. So first iteration will always enter the ‘if’ block

Reset particle

Reset vertex position to zero

Reset velocity to a start velocity

Reset age to current time

Lifespan is a constant, just pass on

```
v_vertex = vec3(0);
v_velocity = vec3(0,10,0);
v_age = u_time;
v_life = a_life;
```

Update particle

If not reseting,

add gravity

add velocity (with scale)

pass age

pass life

```
} else {
    //move particle
    v_velocity = a_velocity - vec3(0.0, 0.005, 0.0);
    v_vertex = a_vertex + v_velocity * 0.005;
    v_age = a_age;
    v_life = a_life;
}
```

Random directional vector

This is where the actual particle distribution is programmed

e.g. a spinning fountain

```
float r = random(vec2(gl_VertexID,u_time*1000.0));

//this makes a nice spinning fountain with a bit of random
float angle = mod(u_time * 1, 6.283);
vec3 ideal_dir = vec3(2.0 * cos(angle), 3.1415, 2.0 * sin(angle));
vec3 randomized_dir = ideal_dir * vec3(r * 0.5 + 1.5);

v_vertex = vec3(0);
v_velocity = randomized_dir;
v_age = u_time;
v_life = a_life;
```

Drawing a texture on point

Modern graphics APIs allow you to draw a texture on a point

Need to enable a constant in init

```
//tell we want textured sprites
glEnable(GL_POINT_SPRITE);
```

Load a texture and send to fragment shader

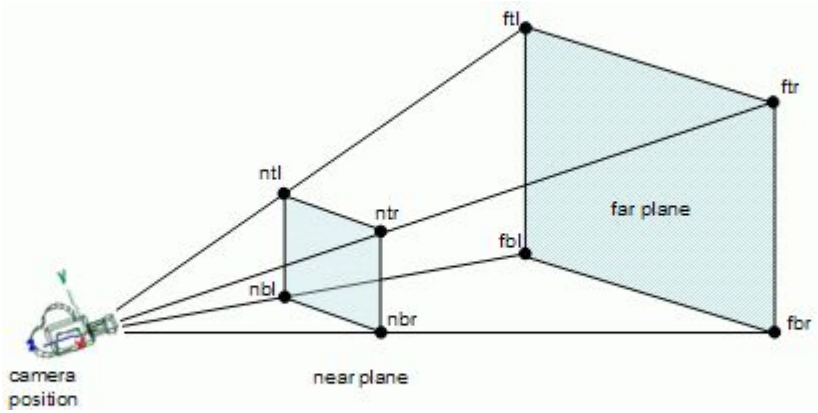
TODO:

- use Parser::parseTexture to load drop texture (in init)
- send that texture to shader uniform (in draw function)
- draw texture in fragment shader (use gl_PointCoord as texture coords)

```
fragColor = texture(u_diffuse_map, gl_PointCoord);
```

Making the point size the same

gl_PointSize = **size of points in pixels (on screen)**



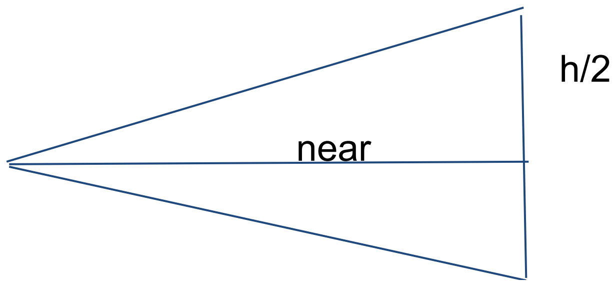
we need to **normalize** the point size:

multiply size by height (or width) of window (to get pixel size)

also divide by w because of homogenous coordinates

getting the window height

Need to ask openGL driver then do some trigonometry




```
//we need to get the height of near plane in order to scale the point size correctly
//as points are drawn in pixel size (so world distance to camera is not good enough)
int viewport[4];
glGetIntegerv(GL_VIEWPORT, viewport);
float height_near_plane = std::abs(viewport[3] - viewport[1]) / (2 * tan(0.5f * cam.fov));
```

then upload to shader and set size (0.2 is size here):

```
gl_PointSize = (0.2 * u_height_near_plane) / gl_Position.w;
```