Particle Systems
with Compute & Geometry Shader

Institute of Computer Graphics and Algorithms
Vienna University of Technology
Motivation

https://www.youtube.com/watch?v=SpwbwTu9Xyk
Overview

• **Particle with properties**
  – Position
  – Velocity
  – Type, ...

• **Compute shader for physics**
  – Apply forces
  – Update properties

• **Geometry shader to create a quad**
  – Create a billboard / sprite

• **Fragment shader to render**
Frame Conditions

- **Upper limit of particles (maximum amount)**
- **One or more particle emitters**
- **Particle count is variable**
  - Each particle can spawn new particles
  - Particles can die
- **Because of this flexibility we need** (*)
  - One array of particles from the previous frame
  - One array of particles from the current frame
  - One atomic counter to keep track of the number

(*) If we would not use two arrays we would need an indication which particles are alive or dead plus memory management
Data Structure

• Think about particle properties
  – vec4: Position \((x,y,z,TTL)\)
  – vec4: Velocity and time-to-live \((vx,vy,vz,\text{<unused>})\)

• C++
  – 2 x 2 SSBOs for position and velocity

• Geometry Shader
  – `layout(std430, binding = 0)` buffer Pos1
    ```cpp
    { vec4 Position_In[]; }
    ```
  – `layout(std430, binding = 1)` buffer Vel1
    ```cpp
    { vec4 Velocity_In[]; }
    ```
  – `layout(std430, binding = 2)` buffer Pos2
    ```cpp
    { vec4 Position_Out[]; }
    ```
  – `layout(std430, binding = 3)` buffer Vel2
    ```cpp
    { vec4 Velocity_Out[]; }
    ```
Setup SSBO

• **We create 2 x 2 SSBOs**
  – Incoming position & velocity buffers
  – Outgoing position & velocity buffers

```cpp
// i = [0,1]
glGenBuffers(1, &ssbo_pos[i]);
glBindBuffer(GL_SHADER_STORAGE_BUFFER, ssbo_pos[i]);
glBufferData(GL_SHADER_STORAGE_BUFFER, MAX_PARTICLES * sizeof(glm::vec4), NULL, GL_DYNAMIC_DRAW);
```
glGenBuffers(1, &atomicCounter);
glBindBuffer(GL_ATOMIC_COUNTER_BUFFER, atomic_counter);
glBufferData(GL_ATOMIC_COUNTER_BUFFER, sizeof(GLuint), NULL, GL_DYNAMIC_DRAW);

GLuint value = 0;
glBufferSubData(GL_ATOMIC_COUNTER_BUFFER, 0, sizeof(GLuint), &value);
glBindBuffer(GL_ATOMIC_COUNTER_BUFFER, 0);

// Because of a performance warning when reading the atomic counter, we
// create a buffer to move-to and read-from instead.
glGenBuffers(1, &temp_buffer);
glBindBuffer(GL_COPY_WRITE_BUFFER, temp_buffer);
glBufferData(GL_COPY_WRITE_BUFFER, sizeof(GLuint), NULL, GL_DYNAMIC_READ);
glBindBuffer(GL_COPY_WRITE_BUFFER, 0);
Setup VAO

• For rendering
  – we connect each of our position SSBOs
  – to a vertex array object

GLuint vaos[2]; // one VAO for each SSBO
const GLuint position_layout = 0;
glGenVertexArrays(2, vaos);
glBindVertexArray(vaos[i]);
glBindBuffer(GL_ARRAY_BUFFER, ssbo_pos[i]);
glEnableVertexAttribArray(position_layout);
glVertexAttribPointer(position_layout, 4, GL_FLOAT, GL_FALSE, 0, 0);

Vertex shader:
layout (location = 0) in vec4 Position;
Setup – Initial Data

- We need initial particle emitters

```cpp
const int TTL = 10000; // time-to-live in seconds
std::vector<glm::vec4> positions;
std::vector<glm::vec4> velocities;

positions.push_back(glm::vec4(0, 0, 0, TTL));
positions.push_back(glm::vec4(2, 0, 1, TTL));
velocities.push_back(glm::vec4(0, 0, 0, 0)); // no velocity
velocities.push_back(glm::vec4(0, 0, 0, 0));

particle_count = positions.size();

// copy the data to the SSBO:
glBindBuffer(GL_SHADER_STORAGE_BUFFER, ssbo_pos[0]);
glBufferSubData(GL_SHADER_STORAGE_BUFFER, 0,
                 particle_count * sizeof(positions[0]), &positions[0]);
// do the same for velocities
```
#version 430

// choose a sufficient work group size:
layout (local_size_x = 16, local_size_y = 16, local_size_z = 1) in;

[...] // our buffers as defined before

// current particle count (atomic counter)
layout (binding = 4, offset = 0) uniform atomic_uint Count;

uniform uint LastCount;
uniform uint MaximumCount;
uniform float DeltaT;

const vec3 GRAVITY = vec3(0, -9.81f, 0);

void main() { [...] }
void main() {
    // unique one-dimensional index:
    uint idx = gl_GlobalInvocationID.x +
        gl_GlobalInvocationID.y * gl_NumWorkGroups.x *
        gl_WorkGroupSize.x;
    if(idx >= LastCount) return;
    vec3 forces = GRAVITY;
    forces += vec3(-5.5f, 0, 0); // add wind or other forces
    vec3 velocity = Velocity_In[idx].xyz + DeltaT * forces;
    vec3 pos = Position_In[idx].xyz + DeltaT * velocity;
    float TTL = Position_In[idx].w - DeltaT;
    if (TTL > 0) { // particle’s still alive
        addParticleToOutputList(pos, velocity, TTL);
    }
}

see next slide
void addParticleToOutputList(vec3 pos, vec3 vel, float TTL) {
    // increment the atomic counter and use its previous value:
    uint nr = atomicCounterIncrement(Count);
    if (nr >= MaximumCount) {
        // we are out of memory
        atomicCounterDecrement(Count);
        return;
    }
    Position_Out[nr] = vec4(pos, TTL);
    Velocity_Out[nr] = vec4(vel, 0);
}
#version 420

layout (location = 0) in vec4 Position;

uniform mat4 ModelViewMatrix;

out vData
{
  float TTL1;
  // additional vertex data goes here
} vertex;

void main()
{
  gl_Position = ModelViewMatrix * vec4(Position.xyz, 1.0f);
  vertex.TTL1 = Position.w;
}
#version 330
layout (points) in; // points in
layout (triangle_strip, max_vertices = 4) out; // quads out
uniform mat4 ProjectionMatrix;
in vData { float TTL1; } vertex[]; // from vertex shader
out vec2 TexCoord0;
flat out float TTL0; // flat: no interpolation between vertices

void main (void) {
    const vec2 size = vec2(5, 5);
    vec4 P = gl_in[0].gl_Position; // position in view-space
    vec2 va = P.xy + vec2(-0.5, -0.5) * size; // bottom-left corner
    gl_Position = ProjectionMatrix * vec4(va, P.zw);
    TexCoord0 = vec2(0.0, 0.0);
    TTL0 = vertex[0].TTL1;
    EmitVertex();
    [...] // repeat accordingly for other corners
    EndPrimitive();
}
in vec2 TexCoord0;
flat in float TTL0;
out vec4 FragColor;

vec3 pow3(vec3 v, float p) { return pow(abs(v), vec3(p)); }
vec3 glowSphere(vec3 color, vec2 p, float radius, float glow) {
  // Inspired by https://www.shadertoy.com/view/4lfXRf
  float term = 1./length(p)/5.;
  return pow3(color * term * 0.5, glow) * radius;
}

void main() {
  vec3 baseColor = vec3(0.588, 1.0, 0.588) * clamp(TTL0, 0.0, 1.0);
  vec3 color = glowSphere(baseColor, TexCoord0 * 2 - 1, 5, 3.25);
  float lum = dot(color, vec3(0.299, 0.587, 0.114));
  FragColor = vec4(color, lum); // use luminance as alpha
}
1. **Run the compute shader**
   - Use one set of buffers (position, velocity) as input
   - Use the other set as output
   - Switch buffers in the next frame (ping/pong)

2. **Run the rendering**
   - Use the computed positions as input
   - Points are converted to quads
   - Use textures or procedural code for display
void calculate(double deltaTime) {
    glUseProgram(computeShader);
    // set all uniforms, like deltaTime, current particle count, ...
    glUniform...( ... );
    // set SSBO and atomic counters...
    glBindBufferBase(GL_SHADER_STORAGE_BUFFER, 0, ssbo_pos[index]);
    index = !index; // ping-pong between buffers

    Compute Shader:
    layout (local_size_x = 16, local_size_y = 16, local_size_y = 1) in;

    // Execute compute shader with a 16 x 16 work group size
    GLuint groups = (particle_count / (16 * 16)) + 1;
    glDispatchCompute(groups, 1, 1);
    [...] // Read back results, like new particle count... (next slide)
}
void calculate(double deltaTime) {
    [...]
    // Read atomic counter through a temporary buffer
    glBindBuffer(GL_ATOMIC_COUNTER_BUFFER, atomic_counter);
    glBindBuffer(GL_COPY_WRITE_BUFFER, temp_buffer);
    // from atomic counter to temp buffer:
    glCopyBufferSubData(GL_ATOMIC_COUNTER_BUFFER, GL_COPY_WRITE_BUFFER, 0, 0,
                        sizeof(GLuint));

    GLuint *counterValue = (GLuint*)glMapBufferRange(GL_COPY_WRITE_BUFFER, 0,
                                                      sizeof(GLuint), GL_MAP_READ_BIT | GL_MAP_WRITE_BIT);
    particle_count = counterValue[0];
    counterValue[0] = 0; // reset atomic counter in temp buffer
    glUnmapBuffer(GL_COPY_WRITE_BUFFER); // stop writing to temp buffer
    // copy temp buffer to atomic counter:
    glCopyBufferSubData(GL_COPY_WRITE_BUFFER, GL_ATOMIC_COUNTER_BUFFER, 0, 0,
                         sizeof(GLuint));

    // memory barrier, to make sure everything from the compute shader is written
    glMemoryBarrier(GL_VERTEX_ATTRIB_ARRAY_BARRIER_BIT);
}
void draw() {
    enableBlendMode();
    glUseProgram(renderProgram); // program with VS -> GS -> FS
    // set uniforms (matrices, ...) as usual

    glBindVertexArray(vaos[index]); // bind VAO

    glDrawArrays(GL_POINTS, 0, particleCount);

    glBindVertexArray(0);
    glUseProgram(0);
    disableBlendMode();
}
Blending

• Many blending possibilities

– http://www.andersriggelsen.dk/glblendfunc.php

```c
void enableBlendMode() {
    glEnable(GL_BLEND);  // activate blending
    glDepthMask(GL_FALSE); // disable writing to depth buffer
    glBlendFunc(GL_SRC_COLOR, GL_SRC_COLOR);
    glBlendEquation(GL_MAX);
}
```

```c
void disableBlendMode() {
    glDisable(GL_BLEND);
    glDepthMask(GL_TRUE);
}
```
Dynamic Spawning of Particles

- **Spawning X particles per second**
  - Problem: FPS are not constant
  - High framerate -> Spawn only $\frac{1}{2}$ or $\frac{1}{4}$ of a particle ??
  - Solution: Accumulate count over multiple frames

```cpp
double particles_to_spawn = 0;
void calculate(double deltaTime) {
    const double spawnRatePerSecond = 2;
    particles_to_spawn += spawnRatePerSecond * deltaTime;
    GLuint spawnCount = 0;
    if (particles_to_spawn > 0) {
        spawnCount += (GLuint) particles_to_spawn;
        particles_to_spawn -= spawnCount;
    }
    // set spawnCount as uniform in compute shader
}
```
Have fun!

- Look at the accompanied source code for more information
- References are at the end of the presentation

Disclaimer:
- This approach was tested on NVidia cards
- It may or may not work on AMD, Intel, etc.
References

• http://www.tomshannon3d.com/2014/07/blueprintfireworks.html
• http://www.gamedev.net/page/resources/_/creative/visual-arts/make-a-particle-explosion-effect-r2701
• http://www.lighthouse3d.com/tutorials/opengl-atomic-counters/