Introduction to Computer Graphics
Project Part 3 – Interaction & Animation

Figure 1: Frames from a camera path through snowy mountains

1 Advanced - Particle System

Particle system have been used extensively to visually simulate various effects such as fire, stars, liquids, clouds, dust. For our landscape, we propose you to use one for snow modeling, e.g. as in the screen cast: `http://lgg.epfl.ch/teaching/icg/Snowing.mov`.

A particle system is usually modeled as a set of points in space together with some generators. The points move due to physical or non-physical laws, depending on what they try to model. In our case we opt for a very simple model with a constant acceleration pulling downwards plus some random offsets to synthesize details: Let p_t^i and v_t^i be the position and velocity of the i-th particle at time t. Given an acceleration a, the position p_{t+1}^i and velocity v_{t+1}^i can be estimated by explicit Euler integration:

$$v_{t+1}^i = v_t^i + \Delta_t(a + r), \tag{1}$$

$$p_{t+1}^i = p_t^i + \Delta_t v_t^i, \tag{2}$$

where Δ_t is the time passed between t and $t+1$ and r is a random vector. In our implementation of snow we reset/delete particles that fall below the water level by setting their position to a random point of a quad above the terrain, and their velocity to zero. In our fragment shader we use a radial gradient to draw the particle using `gl_PointCoord`, and `discard` to reject fragments outside of a circle. We also rescale the particle sizes in the vertex shader by `gl_PointSize` by the inverse of squared distance to the camera, which you have to enable by `glEnable(GL_PROGRAM_POINT_SIZE)`.

Advanced task: Particles on the CPU There are various ways of implementing this in OpenGL: You can keep the positions and velocities on the CPU, perform the integration step there as well. Then send the particles positions to the GPU using `glBufferSubData`, which, in comparison to `glBufferData` only uploads without re-allocating memory. Also consider using `GL_DYNAMIC_DRAW` when allocating your buffer to get better performance. For drawing points use `GL_POINTS`.

Advanced task: Particles on the CPU Integrating the positions and velocities on the CPU and sending them to the GPU each and every time will not scale well with the number of particles. For bigger systems you can represent the particle positions and velocities each in a texture. The integration can then be done on the GPU in the fragment shader while rendering a full-screen quad to a second texture of positions and velocities. Also, after initialization the data never needs to be transferred between GPU and CPU. To get a texture with 3 unbound floats per pixel use `GL_RGB32F` as internal format.

Advanced task: Extending the Particle System We encourage you to change the particles system in any way you like. Here are some inspirations: There are many other phenomena that can be modeled by particles, e.g. rain, clouds, pollen or dust, stars, vulcanos. Also the possibilities of drawing the particles on the screen can range from simple points to billboards and alpha-transparency, see here: `http://www.opengl-tutorial.org/intermediate-tutorials/billboards-particles/`. Particle systems where particles interact with each other can model fluids, soft bodies and flocking (herd behavior of animals such as birds and fish), but are considerably more complex to implement and expensive to run due to neighborhood queries.