CMPS 415/515 Fall 2006 Final Project (Assignment 5), Part I

Due on the last class day

This describes Part I of a two-part assignment.

Part I: Implement a scene with moving particles, an object that attracts them, an object that emits them, and spherical obstacles that they bounce off of.

Part II will require scene elements to be moved along curve functions.

Summary of methods discussed in lecture

Storing particle state
Each particle can be stored as a six-element state vector $\mathbf{s}$ of 3D position $\mathbf{p}$ and 3D velocity $\mathbf{v}$.

Creating particles
Initialize all particles to have some reasonable randomized $\mathbf{p}$, $\mathbf{v}$, and mass m. When a particle has moved very far from the scene or very close to the attractor, reset it at the emitter position with some reasonable randomized ejection velocity.

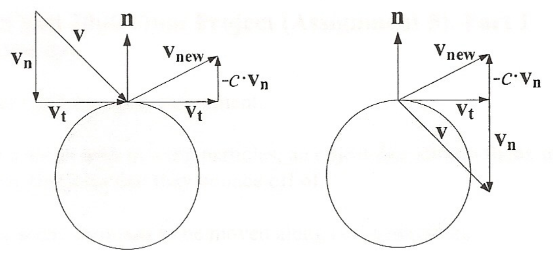
Particle motion
Let Δ be a time step (choose a constant Δ for whatever speed/accuracy you want). To advance an animation forward one step, update each particle with:

$$\mathbf{s} \leftarrow \mathbf{s} + \Delta\cdot\text{deriv}(\mathbf{s}), \quad \text{where deriv}(\mathbf{s}) \text{ is the derivative of } \mathbf{s}.$$

The elements of deriv($\mathbf{s}$) are $\mathbf{v}$ and $\mathbf{a}$, where $\mathbf{v}$ is as above and $\mathbf{a}$ is 3D acceleration computed by Newton's second law, $\mathbf{a} = \mathbf{f}/m$, where m is particle mass and $\mathbf{f}$ is net force on the particle. Net force is the sum of attractor force ($k_1 m\mathbf{d} / r^2$) and viscous drag ($-k_2(\mathbf{v} - \mathbf{v}_{\mathbf{wind}})$), where k_1 and k_2 are constants controlling effect strength, $\mathbf{d}$ is a *normalized* direction vector pointing from $\mathbf{p}$ to the attractor's position, r is the distance between $\mathbf{p}$ and the attractor, and $\mathbf{v}_{\mathbf{wind}}$ is wind velocity.

Collision Detection
After $\mathbf{s}$ is updated as above, detect collisions with obstacles. Compute the distance between the particle and a sphere's center – if it is smaller than the sphere's radius, there is a collision.

Collision Response
When a collision occurs, bounce the particle by changing $\mathbf{v}$. Let the sphere's surface normal at the collision point be $\mathbf{n}$, computed using the difference between $\mathbf{p}$ and sphere center *and normalized.* Then $\mathbf{v}_\mathbf{n} = [(\mathbf{v}\cdot\mathbf{n})\mathbf{n}]$ is the velocity component normal to the surface and $\mathbf{v}_\mathbf{t} = (\mathbf{v} - \mathbf{v}_\mathbf{n})$ is the tangential component. Change velocity to $\mathbf{v}_{\mathbf{new}} = \mathbf{v}_\mathbf{t} - c\cdot\mathbf{v}_\mathbf{n}$, where c is the restitution coefficient (the amount of elastic bounce, usually between 0.0 and 1.0). This ignores friction, since the tangent component is unchanged. Two ways of illustrating the components are shown:



Nonpenetration Constraints
There is a catch – when collision is detected this way, **p** is inside the sphere, not at the boundary. One fix is to move **p** just outside the boundary (e.g., to a point found by moving about one radius away from the sphere center in the **n** direction). Or, allow the particle to be in the sphere but only perform collision response when its velocity is actually inward, i.e., when $\mathbf{n}\cdot\mathbf{v}$ is negative. Otherwise, a particle on its way out can get stuck inside the sphere, especially for low *c*.

Drawing Particles
Draw a motion-blurred particle as a line segment from the particle's position before the state update to its position after the update. You may want to test with points before trying lines.

Animation in OpenGL
Use double buffering, a technique to be described in lecture. This is trivial – use GLUT_DOUBLE in place of GLUT_SINGLE and glutSwapBuffers() in place of glFlush.

One way to animate a scene is to use glutIdleFunc() (set_glutIdleFunc() if you use GLUI) during initialization to specify a function that gets called whenever GLUT has nothing else to do (this "idle function" gets called repeatedly when there are no events being handled). Your idle function updates the simulation by one step and then calls glutPostRedisplay(). The redisplay event is handled after the idle function exits, when GLUT processes the event by calling your display function. After the display function returns, GLUT calls the idle function again, and so on and so on. The result is that the idle function and the display function are called alternately, except that the call to the idle function can be delayed while GLUT processes other events such as keyboard input. This method preserves GLUT's event-handling capabilities.

Input
Allow user control of various parameters, including at least Δ, *c*, $\mathbf{v}_{\mathbf{wind}}$, k_1, k_2, and some sort of control over obstacles.

Graduate students only:

Add a substantial effect or feature. If you have implemented similar work in a past class, you should show more substantial new work instead of resubmitting existing code.

CMPS 415/515 Fall 2006 Final Project (Assignment 5), Part II

Extension to basic particle system requirements

1) Limit each particle's lifetime by assigning a randomized integer at particle creation and decrementing it once per iteration until it reaches 0, at which time the particle should be reinitialized at the emitter. This will ensure there are always more particles to emit. (Note that using modulo when randomizing lifetime is appropriate, unlike for position and velocity, because lifetime is an integer. You will probably want to avoid small lifetimes).

2) Do something interesting with color or some other visual aspect of particles. There are no specific techniques required, to allow some freedom in developing effects, but credit will be deducted if only minimal (or no) effort is made. An example would be random minor color variations, color that varies meaningfully with properties such as age or velocity magnitude, use of blending, etc. Another example would be to use view-oriented billboards, but you should include an option to render particles as line segments.

Animating scene elements using Bézier curves

Animate scene elements other than particles by continuously varying them according to Bézier curves. Pick at least one 3D position, one other major 3D element, and one interesting system parameter to control this way (for example, emitter position, mean ejection velocity, and mean mass for new particles). Curve implementation must match the monomial form given in the course lectures and the construction below. No credit will be given for other implementations.

Suppose you are given at least three points $\mathbf{p}_i$, $0 \le i < n$. For each adjacent pair ($\mathbf{p}_i$, $\mathbf{p}_{(i+1)\%n}$), the midpoint between them is $\mathbf{m}_i = [(\mathbf{p}_i + \mathbf{p}_{(i+1)\%n}) / 2]$. A piecewise-cubic C^1-continuous curve can be passed through all of the midpoints by constructing a Bézier segment for each pair of adjacent midpoints. In the textbook's notation, a segment is $Q(t) = T \cdot M_B \cdot G_B$, $0 \le t \le 1$, where T is the vector $[\, t^3\ t^2\ t\ 1 \,]$, M_B is the 4×4 Bézier basis matrix, and G_B is the 4×3 geometry matrix with rows P_1, P_2, P_3, and P_4. Define the segment from $\mathbf{m}_i$ to $\mathbf{m}_{(i+1)\%n}$ to have a geometry matrix with rows $P_1 = \mathbf{m}_i$, $P_2 = (1/3)\mathbf{m}_i + (2/3)\mathbf{p}_{(i+1)\%n}$, $P_3 = (1/3)\,\mathbf{m}_{(i+1)\%n} + (2/3)\mathbf{p}_{(i+1)\%n}$, and $P_4 = \mathbf{m}_{(i+1)\%n}$. Animate by moving an object along this sequence of curve segments, looping around continuously, and give the user control over the t-increment used when animating.

The technique generalizes to other parameters: just grow the $\mathbf{p}_i$ to contain them ($\mathbf{m}_i$ and G_B grow accordingly). Or, repeat the above, but replace x, y, and z in $\mathbf{p}_i$ with the other parameters.

Read input from an ASCII file giving the n sets of values, one per line. Provide working sample input that represents at least six such sets, with variation in all degrees of freedom.

Draw some reasonable representation of inputs and curve constraints, and provide a mechanism for toggling this on and off. For example, for 3D position, draw small spheres at the input positions and at the four control points per curve segment as generated above. In case you are controlling velocity, inputs and constraints can be shown as vectors. In case of a parameter such as mass, a visual representation is not required, but effects must be clearly visible for credit.

515-level students only:
For Part II, use forward differences as described in section 11.2.9 of the textbook.