COMP SCI 3GC3: Computer Graphics
Assignment 2

Due: Friday, November 7, 2014 at 12:00pm.
Accepted Late until Wednesday, November 12, 2014, 12:00pm, at a 20% per day penalty.
This project is out of 50 marks, and is worth 8% of your final grade.

Particle System

Develop a program using OpenGL and C/C++ to create and display a 3D particle system. Ideally, develop this program in a modular way, and you should be able to re-use it in later applications (e.g., your project!!). The system will consist of a list of particles made up of the following particle structure:

Particle Structure

Start by creating a particle structure/class that contains (at minimum) the following:

- current particle position (*px, py, pz*)
- particle movement direction (*dx, dy, dz*)
- particle movement speed (higher is faster, 0 is stationary)
- rotation angle (*rx, ry, rz*) (for rotating/spinning particles)
- size
- colour (could be constant across the particle, or a different colour at each vertex)
- age (particles have a finite lifespan)

You may add extra fields while developing your program. For example, you may want to add a state variable (e.g., exploding, falling, etc.), shape type (sphere, cube, teapot, etc.), and so on.

Particle List

Create a particle system class containing a list of these structures, as well as a "particle origin" (i.e., point where they come from). You can use the Standard Template Library list to simplify this. See http://www.cppreference.com/wiki/stl/list/start for details and examples of use if you are unfamiliar with the STL. Alternatively, you may develop your own linked list structure (e.g., add a "next" pointer to the particle structure).

Animation of Particles

Particles are created with an initial direction and speed. Their initial position is the particle origin. These values determine how they move through the environment. Note that initial direction and speed depend on how you want to use the particle system. For instance, a particle fountain will have different initial values than a particle explosion. These rules govern the particle update step (see below).

Each frame of animation should conform roughly to the following:

1. Clear frame buffer
2. Render environment (walls, floor, etc.)
3. For each particle in the list:
    a) Update the particle:
        - compute new position (using current position, direction and speed): P' = P+d*S
        - modify direction based on environment variables (gravity, wind, etc.)
        - compute other state variables for particle (kill/delete if too old, etc.)
    b) Render the particle in its new position/orientation (if it still exists)

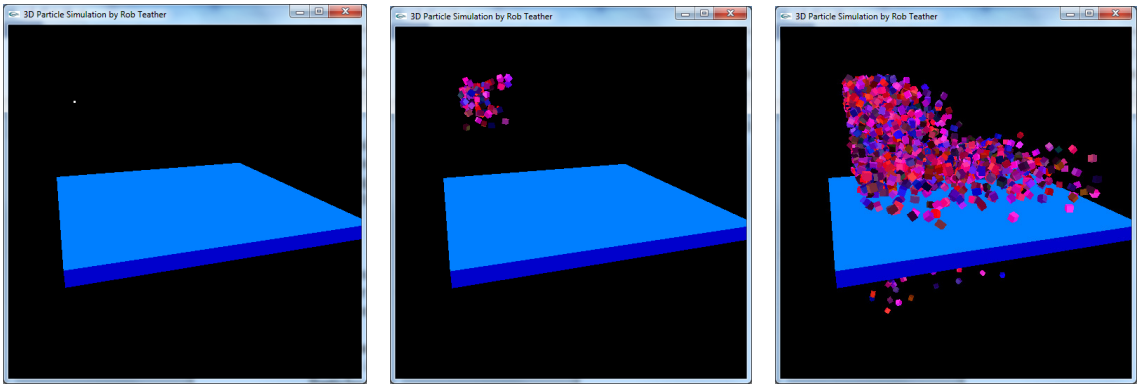
Add a method to the particle system class to perform this update cycle. The update method will be called at regular intervals (e.g., during the GLUT idle or timer function).

Particles can be displayed as small cubes or spheres (e.g., using *glutSolidSphere* with fairly low resolution for efficiency). You will call this in the display function for each particle in the system. Particles can be coloured randomly, or using a specific colour (group). Particle size can be determined randomly upon initialization, but they should be quite small.

Particle Stream

Depending on how you want to be able to use the particle system, you will want to add different ways of initializing them, and different methods for updating direction. For this assignment, create a particle stream which originates at an arbitrary position over top of a large flat quad (displayed in the *xz* plane at *y*=0). Consider, for example, if the particle origin is positioned at (-10, 10, -10) (i.e., to the left, back, and above the ground plane). Particles should be continuously added to the

system, and should shoot outward from the stream origin, falling down to the plane. Using this example, their initial direction should have a positive X and Z components and high speed, with a small random Y component (so they don't all shoot *straight* along the same path). Direction should be modified by gravity, i.e., at each step of the animation, direction = direction + (0, -g, 0), where g is the gravity constant. Examples of a particle system similar to that described above follow:



Note that you can reuse this particle system in your project by modifying the particle initialization and direction update methods. For example, an explosion could be modeled by creating a large number of particles with initial random directions and high speed, with origin centered on the object you want to blow up.

Other Features

1. *Particle lifespan*
   Particles should die and be removed from the list after a pre-determined number of animation steps to avoid slowing down the simulation. You can simply increment the age value with each step of the animation, and when it reaches a pre-determined limit, remove it.
2. *Particle spin*
   Particles should be initialized with a random rotation angle around each axis. This rotation should be applied each frame to cause them to spin as they fly through the scene. Note that angles should be constrained between 0 and 360 degrees in each axis. *Note*: the spinning direction need not make sense with respect to their actual movement direction.
3. *Particle floor collision*
   Particles should bounce off the floor upon striking it. To implement this, check if the *next* movement of the particle will put it beneath the floor (i.e., within the *xz* extents of the floor, and $y < 0$). In this case, invert its movement direction in *y* (i.e., make it positive). Particles that fall off the floor should be deleted after they fall past some pre-determined *y* height (i.e., something less than 0).
4. *Friction mode toggle – "F" key*
   By default, particles should lose some speed (e.g., 10%) each time they bounce off the ground plane to simulate friction. Particles will thus eventually stop bouncing, and become stationary (eventually they will be removed when their age limit is hit). Pressing the "f" key should toggle this friction mode on and off. When off, particles will bounce continuously, never losing and speed – eventually bouncing off the ground plane and falling into the abyss.
5. *Scene rotation - Arrow keys*
   Pressing the arrow keys on the keyboard should rotate the scene about its centre. Left and right should rotate about the *y* axis, and up and down should rotate about the *x* axis.
6. *Start/Stop Simulation – Space bar*
   Pressing the space bar should start and stop the simulation. While the system is on, particles should be continuously added to the list. When the system is off, no new particles should be added but existing particles should continue to animate until they hit their age limit. The system may be either on or off by default upon startup.
7. *Reset – "R" key*
   Pressing the "r" key should reset the simulation, stopping the flow of particles from the stream and clearing the list of particles.
8. *Graphics Features*
   You should use backface culling, double buffering, and perspective projection.

Implement any 2 of the following extra features. You may optionally implement a 3rd extra feature for bonus marks:

9. *Particle Trail*
   Pressing a key turns on particle trails, i.e., a path tracing each particle's motion through the scene. These trails should also disappear when the particles die.

10. *Object Collisions*
    Add some (2 – 3) larger objects (e.g., spheres, boxes) in the scene, resting on the ground plane. These should be substantially larger than individual particles. Implement collision detection with these objects (similar to the collision detection with the floor) so particles bounce off them. You may also consider adding different physical properties to these objects (e.g., particles gain momentum upon striking rubber objects, etc.).
11. *Particle Collisions*
    Implement collision detection between particles. Experiment with how to adjust their direction upon collision to get nice results. Comparing all particle positions to each other is likely to be a very slow operation, so part of the challenge here is figuring out how to make this fast enough to be feasible.
12. *Particle Cam*
    Add a mode (pressing a key) that puts the camera position inside the next particle that enters the system. When the particle dies, the camera reverts to its original position.
13. *Particle explosions*
    When particles die, or hit the floor, they blow up! Add a key-toggled mode that implements this behaviour. This can be implemented by instantiating another (smaller) particle system when the particle explodes, with initial positions set to the position of the exploding particle, and initial direction vectors in random outward directions. Experiment with how many particles can be reasonably added in such an explosion before the system gets too slow. You may want to prevent the explosion particles from recursively exploding to avoid a drastic increase in particles in the system.
14. *Lighting*
    Add materials and normal vectors for all surfaces (particles, floor, any additional objects) and add lighting to the scene.
15. *Snow/Rain*
    Add a mode that changes the behaviour of the system. Rather than originating from a stream, particles instead are initialized at random (constrained) *xz* positions at some high *y* value. They will then fall like snow or rain to the ground plane. This will also require modifying their initial direction vectors, e.g., snow may have a slight "wiggle" as it falls, whereas rain will generally fall straight down. You should also set the colours of these particles to look appropriate (e.g., white for snow).
16. *Particle Cannon*
    Add a cannon object (e.g., a cylinder) from which the particles emit. Pressing keys (e.g., "W", "A", "S", "D") should change the angle of the cannon, which should also affect the initial particle trajectory. In other words, if the cannon is angled at 45˚, then particles should fly out at this angle.
17. *Wind*
    Add variable speed wind, i.e., another force that acts on the particles direction (like gravity) but in a side-to-side fashion, instead of a downward one.
18. *Floor Holes*
    Add a hole (or two?) in the floor. You may optionally toggle this on and off using a key like with some other features. Particles should fall through these holes. This will require enhancing your floor collision detection routine.
19. *Particle Properties*
    Have different types of particles with different properties. Consider things like rubber particles that gain momentum upon hitting the floor, 0-gravity particles that are not affected by gravity, and so on.
20. *Other Feature*
    Implement some cool feature of your own design! These should be of similar complexity to the other features listed above. If you aren't sure about a feature, feel free to contact me to ask about it.

Submission Notes

All programs must initially print out a list of keyboard/mouse/menu commands and how they are used in the program. The marker should not have to look at your source code to figure out how to run your program!! Marks will be deducted otherwise!

You have the option of implementing your assignment on your platform of choice, BUT please make sure your programs can be fully compiled and executed on the departmental machines (e.g., ITB 237). Also be sure to include any additional libraries that you use. You should provide details of:

- The platform used
- The IDE/compiler used
- Any other details relevant to getting your program to run

If the (highly experienced) TA cannot get your code to run, ***your assignment will not be marked!***

Zip your code files and all instructions on how to operate your program. Submit this file to the link provided in the Avenue to Learn website. Submit your code, any project files, and an executable. Include instructions on which IDE you used (if any) as well as compiler instructions.

Familiarize yourself with the department's policy on plagiarism and the university regulations on plagiarism and academic misconduct. Plagiarism will not be tolerated, and will be dealt with harshly.