

## CS334: Assignment #3 – Collision Detection and Animation

**Out:** February 20, 2008

**Back/Due:** March 19, 2008

### Objective:

This focus of this assignment is to obtain a basic understanding of performing collision detection and animation calculations within a computer graphics program. You can recycle as much as you like from previous assignments. You will write a program that creates a basic scene model from the GUI and bounces the objects of the walls of the environment.

### Specifics:

- (0) Object Creation. The program must create and use at least the following objects: box, tetrahedra, and cone. While the `glutSolidXXX` functions from the previous assignment enable drawing the corresponding objects, they do not provide you with the geometry. Thus, represent all objects as a simple list of triangles. You will need to write three generator functions that given simple parameters create a box, tetrahedra, and cone. Then, the common representation of a list of triangles is used for rest of the computations.
  - a. Box: parameterized by (width, height, depth). Using these three numbers you can create a box. Each face of the box consists two triangles (forming a rectangle). Hence, a box object has  $6 \times 2 = 12$  triangles.
  - b. Tetrahedra: parameterized by (height): Using this number you can create a tetrahedra. A tetrahedra consists of 4 triangles linked together. The size parameter determines the height of the tetrahedral (one triangle is the “bottom” and three triangles are the “sides” meeting at the apex).
  - c. Cone: parameterized by (radius, height, and tessellation-factor). Using these numbers you can create a cone. The radius defines the size of the circle at the bottom of the cone. The height defines the distance from the middle of the bottom circle to the apex. The tessellation factor defines how many “steps” and the bottom circle you should have. Each step generates a triangle using the apex vertex. For instance, a cone with a tessellation factor of 10 will end up with 10 triangles.
- (1) Animation. Each object can have a linear velocity direction, speed, and linear acceleration component. Implement these so that the user can choose, via the GUI, one of the following two:
  - a. Velocity: enable specifying for each object an initial linear velocity vector  $v$  and a constant speed (magnitude of  $v$ ); thus given an initial position  $x_0$ , the next position  $x$  is computed using  $x = x_0 + vt$  for some time  $t$ .
  - b. Acceleration: enable specifying an acceleration value in addition to the above; thus have an initial position  $x_0$ , an initial velocity  $v_0$ , and an acceleration  $a$ ; at each time step, the position and the velocity is updated using  $x = x_0 + vt + 0.5at^2$  and  $v = v_0 + at$ .
- (2) Collision Detection. To detect if two objects A and B have collided or interpenetrated, you should perform  $O(N^2)$  tests between the features of each object. The features of the object are its points, edges, and faces (triangles in this case). For

the minimum requirements of this assignment, you only need to implement a subset of the full set of possible tests. In particular, in this assignment object B can be assumed to always be one of the walls (i.e. one of the six surrounding walls composed to two triangles each). Thus, since the objects are convex and the environment is a box (also convex), it suffices to use point-triangle test. In other words, if any of the vertices of object A hits or falls outside the wall (object B), this is a collision or penetration.

- (3) Collision Response. The response has two components.
  - a. The object should be “reflected” off the wall. Thus if the incoming velocity is  $V$ , the reflection  $R$  should be the vector reflected about the normal of the wall (recall raytracing/illumination reflected vector?). The object should *\*never\** be seen outside the environment – that is it should not penetrate the wall. Moreover, it should advance to the same displacement per frame time step despite the collision, e.g., if velocity is 1, acceleration is 0, and distance to the wall at the beginning of the timestep is 0.5, then after the collision, the object should be along the reflected vector at a distance 0.5 from the wall.
- (4) Rendering/GUI. The GUI should support:
  - a. A checkbox for having the box object
  - b. A checkbox for having the tetrahedral object
  - c. A checkbox for having the cone object
  - d. Linear velocity direction, speed, and acceleration controls for each object.
  - e. Initial object translation and rotation controls.
  - f. Parameters for defining each object
  - g. Parameters for defining the size of the world box
  - h. A time step value
  - i. A “start”, “stop”, and “reset” value for the collision/animation. Reset implies all values being set to a reasonable default setting, so that a subsequent “start” produces something reasonable.
  - j. Camera viewpoint translation and rotation.

The rendering loop should use a pleasant Gouraud shading and lighting setup for the objects. A perspective field of view of 60 degrees should be used.

**Extra Credit (0 to 20%):**

- a. Implement a sphere object as well.
  - b. Implement object-to-object collisions using the more general methods explained in class (i.e. object B can be any of the three object types)
- If you do extra-credit, please document and make it easy to use/test.

**Grading:**

We will use the interface to alter the aforementioned parameters and expect to see the correct behavior. *If the interface does not allow a certain parameter to be changed, it will be considered not implemented.* If the program does not compile, zero points will be given.

**This program is a significant step from the previous assignments and thus please start early on the assignment. If you have questions, please see the instructor or TA.**