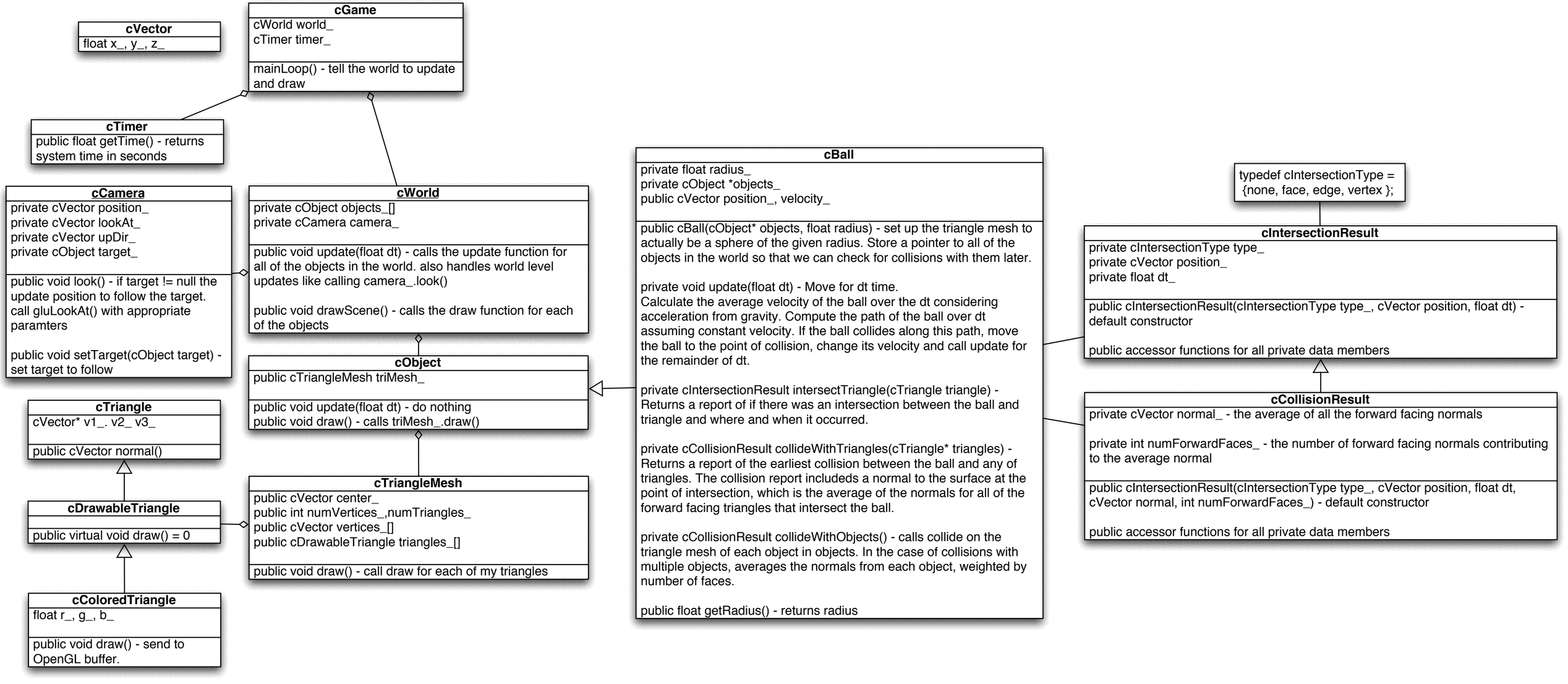
cVector
float x_, y_, z_
cGame
cWorld world_
cTimer timer_
mainLoop() - tell the world to update and draw
cTimer
public float getTime() - returns system time in seconds
cCamera
private cVector position_
private cVector lookAt_
private cVector upDir_
private cObject target_
public void look() - if target != null the update position to follow the target. call gluLookAt() with appropriate paramters
public void setTarget(cObject target) - set target to follow
cWorld
private cObject objects_[]
private cCamera camera_
public void update(float dt) - calls the update function for all of the objects in the world. also handles world level updates like calling camera_.look()
public void drawScene() - calls the draw function for each of the objects
cObject
public cTriangleMesh triMesh_
public void update(float dt) - do nothing
public void draw() - calls triMesh_.draw()
cTriangle
cVector* v1_. v2_ v3_
public cVector normal()
cDrawableTriangle
public virtual void draw() = 0
cColoredTriangle
float r_, g_, b_
public void draw() - send to OpenGL buffer.
cTriangleMesh
public cVector center_
public int numVertices_,numTriangles_
public cVector vertices_[]
public cDrawableTriangle triangles_[]
public void draw() - call draw for each of my triangles
cBall
private float radius_
private cObject *objects_
public cVector position_, velocity_
public cBall(cObject* objects, float radius) - set up the triangle mesh to actually be a sphere of the given radius. Store a pointer to all of the objects in the world so that we can check for collisions with them later.
private void update(float dt) - Move for dt time.
Calculate the average velocity of the ball over the dt considering acceleration from gravity. Compute the path of the ball over dt assuming constant velocity. If the ball collides along this path, move the ball to the point of collision, change its velocity and call update for the remainder of dt.
private cIntersectionResult intersectTriangle(cTriangle triangle) - Returns a report of if there was an intersection between the ball and triangle and where and when it occurred.
private cCollisionResult collideWithTriangles(cTriangle* triangles) - Returns a report of the earliest collision between the ball and any of triangles. The collision report includeds a normal to the surface at the point of intersection, which is the average of the normals for all of the forward facing triangles that intersect the ball.
private cCollisionResult collideWithObjects() - calls collide on the triangle mesh of each object in objects. In the case of collisions with multiple objects, averages the normals from each object, weighted by number of faces.
public float getRadius() - returns radius
typedef cIntersectionType = {none, face, edge, vertex };
cIntersectionResult
private cIntersectionType type_
private cVector position_
private float dt_
public cIntersectionResult(cIntersectionType type_, cVector position, float dt) - default constructor
public accessor functions for all private data members
cCollisionResult
private cVector normal_ - the average of all the forward facing normals
private int numForwardFaces_ - the number of forward facing normals contributing to the average normal
public cIntersectionResult(cIntersectionType type_, cVector position, float dt, cVector normal, int numForwardFaces_) - default constructor
public accessor functions for all private data members