**Quake 3 expanded Vectors and Quaternions**

Quaternions are 4 dimensional imaginary numbers representing rotation without an axis.

Individual values within the quaternion have no relationship to 3 real axis and should not be directly modified.

**Additions to default library**

* VectorFill – Faster way to initialize vector values
* VectorMult - Multiplies 2 vectors component by component
  + A,b,c mult d,e,f = ad, be, cf
  + (V1, V2, Output)
* QuatCopy – Copies q1 into q2
  + (q1, q2)
* QuatMult – Combines 2 quaternion rotations
  + Associative but **Not** communitive, equivalent to applying q1 then q2 to a vector
  + (Q1, Q2, Output)
* QuatApp – Rotates a vector by a quaternion
  + (V1, Q1, Output)
* QuatID – Generates a quaternion with zero rotation
  + (Output)
    - This is not the default value of a quaternion when declared
* VectorDiv – Generates a quaternion from 2 vectors that rotates in between from v2 to v1
  + V1 / v2 = q1 and q1 \* v2 = v1
  + (V1, V2, output)
    - Converting a vector into a quaternion is dividing by (0,0,1)
* QuatAngleAxis – Generates a quaternion from an angle to rotate around a vector
  + Right hand rule, thumb is the axis fingers are direction of rotation
  + (radians, axis, output)
* QuatRand – Generates a quaternion in a random direction uniformly distributed
  + (output)
* QuatSlerp – Generates a quaternion moving between 2 quaternions at a uniform circular rate
  + (Initial quaternion, Final quaternion, time variable (0 – 1), output)
    - Using a time > 1 will continue rotating beyond the target quaternion, and < 0 will rotate backwards
    - Interesting effect by applying recursively to initial quaternion with a constant time of about 0.1 – 0.15, approaches target at exponentially decaying rate.
* VectorProj – Projects u onto v, only keeps component of u in direction of v, returns projection factor (if neg opposite direction of v)
  + (u, v, output) return projection factor
* VectorProjPlane – Projects u onto the plane with normal vector v, only keeps component of u not in direction of v, returns projection factor (if neg opposite direction of v)
  + (u, v, output) return projection factor
* QuatInv – Makes the quaternion rotate in the opposite direction
  + (q1, output)
* FibSpi – Wraps the finocchi spiral around a sphere using unit vectors, constant time approximation of sphere packing
  + (n, I, output)
  + N is total points output is the Ith point on the unit sphere

**Tutorials**

**Plasma Rand** – QuatRand QuatApp

Plasma gun shoots in random directions

1. Projects used game, Classes Used: g\_weapon.c
2. Search game for void Weapon\_Plasmagun\_Fire
   1. Add vec3\_t temp; and quaternion rot; to variables at top
   2. Replace forward with temp on fire\_plasma
   3. Add the following code right below the variable definitions
      1. QuatRand(rot);
      2. QuatApp(forward, rot, temp);

Try applying QuatRand to other vectors such as ps->velocity

**Plasma Loop** – QuatAngleAxis QuatApp

Plasma gun shoots a loop of bolts centered around

1. Projects used game, Classes Used: g\_weapon.c, g\_missile.c, bg\_pmove.c
2. Search game for void Weapon\_Plasmagun\_Fire
   1. Add int I; int n; and vec3\_t temp; and quaternion rot; to variables at top
   2. Initialize n to be the number of bolts in the loop
   3. Add the following code right below the variable definitions. This initializes the first fire vector and the rotation in between subsequent fire vectors.
      1. QuatAngleAxis(M\_PI/6, right, rot);
      2. QuatApp(forward, rot, temp)
      3. QuatAngleAxis(2\*M\_PI / n, forward, rot)
   4. Make a for loop surrounding the last 3 lines of code iterate I to n
   5. Replace forward with temp on fire\_plasma
   6. Add the following code inside the for loop after the 3 lines. This rotates to the next fire vector
      1. QuatApp(temp, rot, temp);
3. Search Game for gentity\_t \*fire\_plasma
   1. Change 10000 in bolt->nextthink = level.time + 10000; to a lower number
      1. This is the lifetime in milliseconds of the projectile
4. Search Game for static void PM\_Weapon and scroll down to case WP\_PLASMAGUN:
   1. Change addTime = 100; to a higher number
      1. This is the reload in the Plasma Gun in milliseconds
   2. Recompile cgame this is a shared file

ENTITYNUM\_MAX\_NORMAL will be quickly exceeded without steps 3 or 4

**Plasma Sphere** – FibSpi

Plasma gun shoots a sphere of bolts in all directions

1. Projects used game, Classes Used: g\_weapon.c, g\_missile.c, bg\_pmove.c
2. Search game for void Weapon\_Plasmagun\_Fire
   1. Add int I; int n; and vec3\_t temp; to variables at top
   2. Initialize n to be the number of bolts in the sphere
   3. Replace forward with temp on fire\_plasma
   4. Make a for loop surrounding the last 3 lines of original code iterate I to n
   5. Before the first line inside the loop add FibSpi(I, n, temp);
3. Search Game for gentity\_t \*fire\_plasma
   1. Change 10000 in bolt->nextthink = level.time + 10000; to a lower number
      1. This is the lifetime in milliseconds of the projectile
4. Search Game for static void PM\_Weapon and scroll down to case WP\_PLASMAGUN:
   1. Change addTime = 100; to a higher number
      1. This is the reload in the Plasma Gun in milliseconds
   2. Recompile cgame this is a shared file

ENTITYNUM\_MAX\_NORMAL will be quickly exceeded without steps 3 or 4

**Swing Grapple** – VectorProjPlane

Spiderman style swinging

1. Projects used game, Classes used bg\_pmove.c
2. Complete the tutorial to enable the grapple
3. Search Game for static void PM\_GrappleMove
4. Comment out the method body and add the following code. This prevents the player from moving away from the grapple.
   1. float factor;
   2. vec3\_t temp;
   3. vec3\_t dir;
   4. VectorSubtract(pm->ps->origin, pm->ps->grapplePoint, dir);
   5. factor = VectorProjPlane(pm->ps->velocity, dir, temp);
   6. if(factor>0) VectorCopy(temp, pm->ps->velocity);
      1. if velocity moves towards grapple don’t remove it
5. Search Game for void PmoveSingle
6. Scroll to the if statement containing PM\_GrappleMove();
7. Move the entire if statement to the top of the if else tree
   * 1. This is now separate from the tree allowing normal movement while grappling.
8. Recompile cgame this is a shared file

**Plasma Wave** – QuatID QuatAngleAxis QuatApp QuatSlerp

Oscillates between shooting high and low to make a wave

1. Projects used game, Classes Used: g\_weapon.c, g\_missile.c
2. Search game for void Weapon\_Plasmagun\_Fire
   1. Add vec3\_t temp; quaternion id; and quaternion rot; to variables at top
   2. Replace forward with temp on fire\_plasma
   3. Add the following code right below the variable definitions.
      1. QuatID(id);
      2. QuatAngleAxis(M\_PI/16, right, rot);
         1. Swap right with up to make wave horizontal
      3. QuatSlrp(id, rot, sin(level.time / 1000.0f \* 2 \* M\_PI / 1), rot);
         1. The 1 at the end of sin() is how many seconds per wave cycle.
         2. Try experimenting with rational and irrational ratios to the reload time
      4. QuatApp(forward, rot, temp);
3. Search Game for gentity\_t \*fire\_plasma
   1. VectorScale( dir, 2000, bolt->s.pos.trDelta );
      1. Change 2000 to a lower number.
         1. This is the velocity of the bolt.

Try plasma loop and plasma wave at the same time.

**Heavy Plasma** – VectorDiv QuatApp QuatSlerp

Plasma beam lags behind where you are looking

1. Projects used game, Classes Used: bg\_public.h, g\_weapon.c, g\_missile.c
2. Search game for } persEnum\_t;
   1. Add PERS\_FIREX, PERS\_FIREY, and PERS\_FIREZ to the end of the enum.
3. Recompile cgame this is a shared file
4. Search game for void Weapon\_Plasmagun\_Fire
   1. Add vec3\_t temp; variables at top
   2. Add the following line of code right below the variable definitions
      1. VectorFill(ent->client->ps.persistant[PERS\_FIREX]/((1 << 15) - 1.0f), ent->client->ps.persistant[PERS\_FIREY]/((1 << 15) - 1.0f), ent->client->ps.persistant[PERS\_FIREZ]/((1 << 15) - 1.0f), temp);
5. Search Game for void ClientThink\_real(
   1. Add quaternion rot; quaternion id; vec3\_t temp; and vec3\_t forward to the variable definitions
   2. Add the following code right below client = ent->client;
      1. AngleVectors(client->ps.viewangles, forward, NULL, NULL);
      2. QuatID(id);
      3. VectorFill(client->ps.persistant[PERS\_FIREX]/((1 << 15) - 1.0f), client->ps.persistant[PERS\_FIREY]/((1 << 15) - 1.0f), client->ps.persistant[PERS\_FIREZ]/((1 << 15) - 1.0f), temp);
      4. if (abs(1 - VectorLength(temp)) > 0.1f){VectorCopy(forward, temp);}
         1. Initializes the vector, should only happen once
      5. VectorDiv(forward, temp, rot);
      6. QuatSlrp(id, rot, 0.02f, rot);
         1. Rotates 2% of the way to the forward vector every frame
      7. QuatApp(temp, rot, temp);
      8. client->ps.persistant[PERS\_FIREX] = temp[0] \* ((1 << 15) - 1);
      9. client->ps.persistant[PERS\_FIREY] = temp[1] \* ((1 << 15) - 1);
      10. client->ps.persistant[PERS\_FIREZ] = temp[2] \* ((1<< 15) - 1);
6. Search Game for gentity\_t \*fire\_plasma
   1. VectorScale( dir, 2000, bolt->s.pos.trDelta );
      1. Change 2000 to a lower number.
         1. This is the velocity of the bolt.