

-MAKE A BACKUP OF TORQUE BEFORE TRYING THIS

Hey there! The long awaited and much delayed locational damage code is finally finished, and ready for your use! ☺

Some words before we get started:

-There are 60 pages of code and comments here. It's nicely spaced out, but that's still a lot of code

-All the code I have created that is somewhat relevant to locational damage is lumped into here. So you might not need all of it, if, say for example, you don't have any close combat weapons (like swords) in your game. However, separating it all into individual tutorials (one for projectiles, one for close combat, etc) would be too much work.

-This work is not exclusively mine. The projectile portion of it was graciously donated by Luigi Rosso. The rest however, is my work

-Any improvements to this code or tutorial would be welcome. If you would like to try your hand at separating it into smaller pieces, or adding some comments, I would be happy to work with you and update this resource.

-Please do NOT copy this tutorial, or host it anywhere else. This is just so that we never get old versions floating around (I plan on updating it with any problems or submissions from people)

-I take no responsibility for any damage of any form that may result from any interaction with this code snippet. That includes falling out of your chair while reading this statement. ☺

-If you are having problems, or have any questions at all, come onto the IRC channel and talk to me, AIM me (screen name: JoshSAlbrecht), or send me an email at raistlin@rochester.rr.com I'd love to get some feedback.

-MAKE A BACKUP OF TORQUE BEFORE TRYING THIS

The format is as follows:

-Words in all caps are the names of the file you are going to be adding code to.

-Obviously, you shouldnot put ... into a compiler. The ... symbolizes that I have left out a chunk of code that isn't relevant

-The new code, or code lines that are changed are highlighted in blue. The other code (in black) is there so you can find the correct place to put the code in. Red comments are special instructions about where to put the code.

-Use the find command to help you find the places in the file that I am talking about. For example, in this first piece of code, doing a find for "if (obj->mProcessTick)" would probably find you the location that I am talking about so you can add the new code

-If you plan on really using this in your game, I HIGHLY suggest that you read through all the coments and try your best to undertand it. I know its not perfect, but it should be clear enough to at least get a decent understanding.

Now, I am going to provide a quick summary of what this code does, and how it does it. If you don't care, and just want the code, just skip this next section. (I don't suggest that)

-IF YOU HAVENT ALREADY, MAKE A BACKUP OF TORQUE BEFORE TRYING THIS

Summary:

Basically, this code creates a new way of applying damage in Torque. Prior to my “improvements”, Torque simply used the bounding box of whatever is being attacked or shot at for ALL collision. This is acceptable in a game like Tribes 2, where the action is fast-paced, the players are moving at high speeds, and the weapons almost all do area damage. However, in a slower game, or a game where close combat is involved, the inaccuracy that results from this method is unacceptable. So, I added a system much like that in the Half-Life engine: Boxes are created around each limb, and they are just big enough to enclose the limb. When the limbs move, the boxes stay around the limbs, as if the person were wearing a whole bunch of boxes on their limbs (← that’s a candidate for the worst simile ever 😊) Then, collision is done against these boxes, to see if any of them are hit, and damage can be applied based on exactly where the player is hit.

So that’s the concept behind this madness. In terms of code, I just created a ‘hitbox’ (the name that Half Life uses for the boxes around the limbs) structure. It has the dimensions of the box, and the name of the bone it is centered on. Basically, I transform the box and whatever line is trying to collide against it into ‘bone space’ (the coordinate system that is aligned with the bone). Then, the box can be treated as ‘axis-aligned’ (ie, its side are perpendicular and parallel to the axis). This makes it MUCH easier on the computer to do collision. The same basic thing is done for melee weapons and projectiles; it’s simply a change in what line we collide against the boxes.

In addition to this code, you will find methods for initializing and removing melee weapons, code for shields and armor, damage handling for the different parts of the body, an example of how to initialize the hitboxes for a player, code to make arrows (projectiles) stick into players when they hit, code for removing these projectiles, and more. Its pretty much everything you need to use the locational damage code, which is why this tutorial wasn’t released months ago: there would have been too much overlapped effort, as everyone would have had to add the same basic functionality.

Enjoy. 😊

-I WASN’T JOKING. MAKE A BACKUP OF TORQUE BEFORE TRYING THIS. 😊

Add this file to the project. Put the file in /torque/engine/game

MELEE.H

```
// Torque Game Engine
// Copyright (c) 2002 Epsilon Entertainment
//-----

#ifndef _MELEE_H_
#define _MELEE_H_

//the structure used to represent the boxes that enclose the limbs.
struct Hitbox
{
    //this is used to store the min and max for the box (the size and location)
    Box3F hitbox;

    //the name of the node that the box is attached to
    StringTableEntry nodeName;

    //the location of the box, in world cords, during the last frame
    Box3F lastLocation;
};

//the structure used to store information about melee weapons
struct MeleeWeapon
{
    //the nodes for the start and end of the weapon.
    //Yes, it could have been done with a single node and a vector, and that's how
    //it is essentially done, but this was done in case I ever wanted to make a
    //weapon that spins, like a flail
    StringTableEntry node1;        //typically the base of the weapon
    StringTableEntry node2;        //typically the tip of the weapon

    //this is the slot that the weapon is mounted on
    S32 slot;

    //the real endpoint of the weapon. It is calculated based off of the position of the
    //'end' node defined above. Is in the spatial coordinate system of the first node
    Point3F end;

    //this is the hitbox used for collision with the weapon.
    //you can make a shield by making a very large hitbox
    Hitbox col;

    Point3F lastStart;            //last starting point (in world-space)
    Point3F lastEnd;              //last ending point (in world-space)
};
#endif
```

```
};
```

```
#endif
```

Add the following code to shapebase.h

SHAPEBASE.H

//at the top of the file

```
...
#ifndef _MELEE_H_
#include "game/melee.h" //need to include the definition of hitbox and meleeweapon
#endif
//declare the projectile and projectile data classes so we can interact with them
class Projectile;
class ProjectileData;
```

...

//NOTE: The following code occasionally appears to have redundant public modifiers. However, I have them there to show that it makes more sense to have certain things in certain public section. You will notice that there are multiple public sections in shapebase for example. Some have primarily variables, others functions, etc. If you want, you can keep it simply by putting all the code under one public modifier. Otherwise, you can distribute it in keeping with the current organization. Just make sure it's public. 😊

```
struct ShapeBaseData : public GameBaseData {
...
public:
    enum Constants {
...
        AIRRepairNode = 31, //←Add that comma
        Max_Hitboxes = 20, //maximum number of hitboxes allowed per player
    };
    enum {
        HITBOX_DETACH_BITS = 5
    };
...
public:
...
//the following are defined for shapedata so that we can set the size, number,
//location size, and the node they are attached to for each hitbox.
    F32 numBoxes; //number of boxes for this player
    Point3F HitMin[Max_Hitboxes]; //min point for the box
    Point3F HitMax[Max_Hitboxes]; //max point for the hitbox
    StringTableEntry NodeName[Max_Hitboxes]; //string that represents the name of //the
node that our box is attached to
```

...

```

class ShapeBase : public GameBase
{
...
public:
    enum PublicConstants {
        ...
        CollisionTimeoutValue = 250,    // Timeout in Ms.
        Max_Hitboxes = 20 //same as before
        //unfortunately, I believe that it must be identically set both here, and in the
//datablock class. Sorry. ☹
    };
...
    ShapeBaseData* mDataBlock;
private:
    //Raistlin: moved datablock pointer from private to public
    //ShapeBaseData* mDataBlock;
...
public:
...
//defines an array of meleeweapons for each arm. The reason for this is that you
//may need more than one line to define a weapon. Might as well be flexible. And if
//you don't like it, you can remove it, or just ignore it, it's not hurting performance. ☺
    MeleeWeapon mLMeleeWeapons[2];
    MeleeWeapon mRMeleeWeapons[2];

//defines the array of hitboxes
    Hitbox mHitboxes[Max_Hitboxes];

//Functions that we'll need for the collision testing
//main function for testing collision
    void Melee();
//updates the positions of each melee weapon at the end of each tick
    void updateMelee();
//check one weapon against one player for any hits
    void checkForHits(ShapeBase *p, Point3F start, Point3F end, MeleeWeapon
&mMeleeWeapon);
//finds the player that the weapon could potentially hit
    ShapeBase* findVictim(Point3F start, Point3F end);
//checks the weapon against a hitbox
    bool CheckSwing(int Interpolations, Hitbox& Hitbox, MeleeWeapon& MeleeWeapon,
    MatrixF mat, Point3F Start, Point3F End);
...
public:
...
    // Network state masks
    enum ShapeBaseMasks {
        ...

```

```

    SoundMaskN    = Parent::NextFreeMask << 7,    // Extends + MaxSoundThreads
bits
    detachMask    = Parent::NextFreeMask << 8,
//NOTE: The above number (8) may need to be different if you have already made
changes to this section of the code. Just make sure it's one more than the one before it,
and one less than the one after (if there is one)
...
};
...
public:

//two functions for graphically displaying the weapons and hitboxes
//useful for debugging
static void MyLine(const Point3F& start, const Point3F& end);
    void DrawCollideHitbox(Box3F& Box, MatrixF& mat);

//functions for adding and removing the meleeweapons.
    bool registerMelee(char* node1, char* node2, char* wepID, char* arm, char*
min, char* max);
    bool unregisterMelee(char* wepID);

// this data block stores members of our
// linked attached projectile list
struct attachedProjectile
{
    Projectile *p;
    attachedProjectile *next;
    int aHBID;
    attachedProjectile()
    {
        p = NULL;
        next = NULL;
    }
};
// the head and tail of the list
attachedProjectile *attachedProjectiles;
attachedProjectile *attachedProjectilesTail;

// member for rendering the attached projectiles
void renderAttachedProjectiles(SceneState* state);
// member for adding projectiles to the list
void attachProjectile(Projectile *proj, int hbid);

// members for detaching projectiles
void detachAllProjectiles();
void detachAllProjectilesGhost();

void detachProjectiles(int limb);

```

```
void detachProjectilesGhost(int limb);

int detachValue;
bool dProjectile, dHitbox;

// this member will be invoked by sticky projectiles
// to check for hitbox collision
bool checkProjectileCollisionHB(Point3F start, Point3F end, Point3F vel, Point3F
rayPoint, MatrixF &xform, int &collided);

//needed to get weapon transforms
MatrixF getNodeTransform2( StringTableEntry nodeName, S32 slot );
```

Make the following modifications to shapebase.cc Note that the bulk of the code is n this file, and this doc is 28 pages long. ☺

SHAPEBASE.CC

```
...
//at the top of the file obviously...
#include "game/projectile.h"
...

ShapeBaseData::ShapeBaseData()
{
...
//assign some default values to the hitboxes in case they aren't set manually
numBoxes = 0;
for(int i=0; i< Max_Hitboxes; i++)
{
    HitMin[i].set(0,0,0);
    HitMax[i].set(0,0,0);
    NodeName[i] = NULL;
}
...

void ShapeBaseData::initPersistFields()
{
...
//add these fields to the script interpreter so people can set them in datablocks
addField("HitMin", TypePoint3F, Offset(HitMin, ShapeBaseData), Max_Hitboxes);
addField("HitMax", TypePoint3F, Offset(HitMax, ShapeBaseData), Max_Hitboxes);
addField("NodeName", TypeString, Offset(NodeName, ShapeBaseData),
Max_Hitboxes);
addField("numBoxes", TypeF32, Offset(numBoxes, ShapeBaseData));
...
}
...

//MAKE SURE THESE ARE TOWARDS THE TOP OF THE FILE
//this function calls the function that registers weapons
static bool cregisterMelee(SimObject *ptr, S32 argc, const char **argv)
{

    ShapeBase *obj = static_cast<ShapeBase*>(ptr);

    return obj-
>registerMelee((char*)argv[2],(char*)argv[3],(char*)argv[4],(char*)argv[5],
(char*)argv[6], (char*)argv[7]);
}
```

```

//this function calls the function that unregisters weapons
static bool cunregisterMelee(SimObject *ptr, S32 argc, const char **argv) {
    ShapeBase *obj = static_cast<ShapeBase*>(ptr);
    return obj->unregisterMelee((char*)argv[2]);
}

//this function registers meleeweapons
bool ShapeBase::registerMelee(char* node1, char* node2, char* wepID, char* arm,
char* min, char* max)
{
    //Still need support for flails etc.

    MeleeWeapon wep;

    Point3F TEMPmin(0,0,0);
    Point3F TEMPmax(0,0,0);
    wep.col.lastLocation.min = TEMPmin;
    wep.col.lastLocation.max = TEMPmax;

    //get the values for the min and max from the strings that we're passed
    dSscanf(min,"%f %f %f",&TEMPmin.x,&TEMPmin.y,&TEMPmin.z);
    dSscanf(max,"%f %f %f",&TEMPmax.x,&TEMPmax.y,&TEMPmax.z);

    //get an int for the arm, 1 is left, 0 is right
    S32 Arm = dAtoi(arm);

    ShapeBaseImageData *weapon;

    //find the weapon and weapon slot, if their isn't one, return false (that's bad)
    if (Sim::findObject(wepID,weapon))
    {
        wep.slot = getMountSlot(weapon);
    }
    else
    {
        wep.slot = -1;
        return false;
    }

    S32 ref = 0;

    //this is more than a little lame
    //we need to find the start and end nodes, though we aren't sure if they are on the
    //player or on the shape (weapon) itself. So we check everything for both the start
    // and end positions.
    ref = weapon->shape->findName(node1);
    if(ref!=-1)
    wep.node1 = weapon->shape->getName(ref);

```

```

ref = weapon->shape->findName(node2);
if(ref!=-1)
wep.node2 = weapon->shape->getName(ref);

ref = mShapeInstance->getShape()->findName(node1);
if(ref!=-1)
wep.node1 = mShapeInstance->getShape()->getName(ref);

ref = mShapeInstance->getShape()->findName(node2);
if(ref!=-1)
wep.node2 = mShapeInstance->getShape()->getName(ref);

wep.col.hitbox.min = TEMPmin;
wep.col.hitbox.max = TEMPmax;
wep.col.nodeName = wep.node1;

MatrixF mat;
Point3F temp1;
temp1.set(0,0,0);

wep.lastStart = temp1;

//get the transformation from the weapon's coordinates into the world coordinates,
// and apply it to the end node so we can see where it is in world space. We set lastEnd
// to this so that it has a logical value.
MountedImage& image = mMountedImageList[wep.slot];
if (image.dataBlock) {
    ShapeBaseImageData& data = *image.dataBlock;

    MatrixF nmat;
        MatrixF nmat2;
        getRenderMountTransform(data.mountPoint,&nmat); //mount to world
        nmat2.mul(nmat, getNodeTransform2(node2, wep.slot));
            mat.mul(nmat2,data.mountTransform);
            mat.mulP(temp1);
    }

wep.lastEnd = temp1;
//obj to world, node to obj
//get the transformation from the node to the world, then apply it to the start position
//to set the "last" start position to something that makes sense
mat.mul(getRenderTransform(), getNodeTransform2(node1, wep.slot));
    mat.mulP(wep.lastStart);
//invert it, and bring the last end position from worldspace into the nodespace of the
//start node. This gives us the offset of the end node in the start node's coordinates.
//This should always be constant if spacetime isn't warping and the weapon is a solid,
//so it saves us from having to get transformation matrices for each node every time

```

```

//we want to calculate the new position. Instead, we just get the transformation for
//the start position and apply it to both start and end
    mat.inverse();
    mat.mulP(temp1);
    wep.end = temp1;

//assign this weapon to the correct meleeWeapon
    if(Arm==0)    //right
    {
        if(mRMeleeWeapons[0].slot==-1)
        {
            mRMeleeWeapons[0] = wep;
        }
        else
            mRMeleeWeapons[1] = wep;
    }

    if(Arm==1)    //left
    {
        if(mLMeleeWeapons[0].slot==-1)
        {
            mLMeleeWeapons[0] = wep;
        }
        else
            mLMeleeWeapons[1] = wep;
    }

return false;
}

bool ShapeBase::unregisterMelee(char* WepID)
{
    //find the weapon that corresponds to the ID, and set the slot to -1
    //slot is always checked to see if the weapon is initialized; if it's -1, we assume there
    //is no weapon
    S32 wepID = dAtoi(WepID);
    if(mLMeleeWeapons[0].slot==wepID)
        mLMeleeWeapons[0].slot=-1;
    if(mLMeleeWeapons[1].slot==wepID)
        mLMeleeWeapons[1].slot=-1;
    if(mRMeleeWeapons[0].slot==wepID)
        mRMeleeWeapons[0].slot=-1;
    if(mRMeleeWeapons[1].slot==wepID)
        mRMeleeWeapons[1].slot=-1;
    else
        return false;
    return true;
}

```

...

```
void ShapeBaseData::packData(BitStream* stream)
```

```
{
```

...

```
//write the data for each hitbox so the datablock containing the info can be sent to  
//the clients.
```

```
    stream->write(numBoxes);
```

```
    for (U32 i = 0; i < numBoxes; i++)
```

```
    {
```

```
        stream->write(HitMin[i].x);
```

```
        stream->write(HitMin[i].y);
```

```
        stream->write(HitMin[i].z);
```

```
        stream->write(HitMax[i].x);
```

```
        stream->write(HitMax[i].y);
```

```
        stream->write(HitMax[i].z);
```

```
    }
```

```
    for(int i=0; i<numBoxes; i++ )
```

```
    {
```

```
        if( stream->writeFlag( NodeName[i] != NULL ) )
```

```
        {
```

```
            stream->writeString( NodeName[i] );
```

```
        }
```

```
    }
```

...

```
}
```

...

```
void ShapeBaseData::unpackData(BitStream* stream)
```

```
{
```

...

```
//read the hitbox info in the datablocks coming from the server
```

```
    stream->read(&numBoxes);
```

```
    for (U32 i = 0; i < numBoxes; i++)
```

```
    {
```

```
        stream->read(&HitMin[i].x);
```

```
        stream->read(&HitMin[i].y);
```

```
        stream->read(&HitMin[i].z);
```

```
        stream->read(&HitMax[i].x);
```

```
        stream->read(&HitMax[i].y);
```

```
        stream->read(&HitMax[i].z);
```

```
    }
```

```

for(int i=0; i<numBoxes; i++)
{
    if( stream->readFlag() )
    {
        NodeName[i] = stream->readSTString();
    }
}
...
}

```

...

```

ShapeBase::ShapeBase()
{
...
//safe default values for our variables
mRMeleeWeapons[0].slot = mRMeleeWeapons[1].slot = -1;
mLMeleeWeapons[0].slot = mLMeleeWeapons[1].slot = -1;

attachedProjectiles = NULL;
attachedProjectilesTail = NULL;
dProjectile = false;
dHitbox = false;

```

...

```

bool ShapeBase::onNewDataBlock(GameBaseData* dptr)
{
...
//cycle through each of the hitboxes and transfer the data from the datblock
MatrixF mat;
for(S32 i = 0; i<mDataBlock->numBoxes; i++)
{
    Box3F temp;
    temp.min = mDataBlock->HitMin[i];
    temp.max = mDataBlock->HitMax[i];

    mHitboxes[i].hitbox=temp;

    mHitboxes[i].nodeName = mDataBlock->NodeName[i];

    mHitboxes[i].lastLocation.min.set(0,0,0);
    mHitboxes[i].lastLocation.max.set(0,0,0);
}

```

```

mLMeleeWeapons[0].slot=mLMeleeWeapons[1].slot=-1;
mRMeleeWeapons[0].slot=mRMeleeWeapons[1].slot=-1;
    return true;
}

...

void ShapeBase::processTick(const Move* move)
{
...
    Con::executef(mDataBlock,3,"onDamage",scriptThis(),delta);
}
}
}
//during the update, if we are a player on the server (ie not a ghost), calculate collision
    if(!isGhost())
        Melee();

...

//Room for optimization in the following function:
//Too many unnecessary matrix multiplications.
//This function updates the position of the hitboxes and meleeweapons at the end of
//each tick
void ShapeBase::updateMelee()
{
    MatrixF mat;
    Point3F start;
    Point3F end;
    //cycle through each weapon
    for(int i=0; i<2; i++)
    {
        if(mLMeleeWeapons[i].slot!=-1)
        {
            start.set(0,0,0);
            end = mLMeleeWeapons[i].end;
            //get the node->world transformation, and apply to the weapon and its hitbox
            //to properly set the current world location (which will be the last location next tick)
            mat.mul(getRenderTransform(),
getNodeTransform(mLMeleeWeapons[i].node1));
            mat.mulP(start);
            mat.mulP(end);

            mLMeleeWeapons[i].lastStart = start;
            mLMeleeWeapons[i].lastEnd = end;

            start = mLMeleeWeapons[i].col.hitbox.min;

```

```

        end = mLMeleeWeapons[i].col.hitbox.max;

        mat.mulP(start);
        mat.mulP(end);

        mLMeleeWeapons[i].col.lastLocation.min = start;
        mLMeleeWeapons[i].col.lastLocation.max = end;
    }
}
//exact same thing, but on the right side weapon
for(int i=0; i<2; i++)
{
    if(mRMeleeWeapons[i].slot!=-1)
    {
        start.set(0,0,0);
        end = mRMeleeWeapons[i].end;

        mat.mul(getRenderTransform(),
getNodeTransform(mRMeleeWeapons[i].node1));
        mat.mulP(start);
        mat.mulP(end);

        mRMeleeWeapons[i].lastStart = start;
        mRMeleeWeapons[i].lastEnd = end;

        start = mRMeleeWeapons[i].col.hitbox.min;
        end = mRMeleeWeapons[i].col.hitbox.max;

        mat.mulP(start);
        mat.mulP(end);

        mRMeleeWeapons[i].col.lastLocation.min = start;
        mRMeleeWeapons[i].col.lastLocation.max = end;
    }
}

//cycle through each hitbox, setting the current worldspace location of each
for(int i = 0; i<mDataBlock->numBoxes; i++)
{
    mat.mul(getRenderTransform(),
getNodeTransform(mHitboxes[i].nodeName));

    start = mHitboxes[i].hitbox.min;
    end = mHitboxes[i].hitbox.max;

    mat.mulP(start);
    mat.mulP(end);
}

```

```

        mHitboxes[i].lastLocation.min = start;
        mHitboxes[i].lastLocation.max = end;
    }
}

```

```

void ShapeBase::Melee()
{
    if(!mShapeInstance)
        return;
}

```

//NOTE: The following code creates a gigantic weapon sticking out of your face, so you can test whether collisions really work or not. It **OBVIOUSLY** shouldn't be left here. It just forcibly creates the weapon so we have something to work with.

```

mLMeleeWeapons[0].slot=1;
mLMeleeWeapons[0].node1="Bip01 Head";
Point3F temp;
temp.set(0,5,0); //head
mLMeleeWeapons[0].end=temp;
mLMeleeWeapons[0].col.lastLocation.min.set(0,0,0);
mLMeleeWeapons[0].col.lastLocation.max.set(0,0,0);
mLMeleeWeapons[0].col.hitbox.min.set(-0.2,0,-0.2);
mLMeleeWeapons[0].col.hitbox.max.set(0.2,5,0.2);
mLMeleeWeapons[0].col.nodeName = "Bip01 Head";

```

```

    MatrixF mat;
    Point3F start;
    Point3F end;
//cycle through each weapon
    for(int i=0; i<2; i++)
    {
        if(mLMeleeWeapons[i].slot!=-1)
        {
//get the node->world matrix, then apply it to find the location of the weapon
            start.set(0,0,0);
            end = mLMeleeWeapons[i].end;

            mat.mul(getRenderTransform(),
getNodeTransform(mLMeleeWeapons[i].node1));
            mat.mulP(start);
            mat.mulP(end);
//see if the weapon could possibly hit anyone
            ShapeBase* p = findVictim(start, end);
            if(p)

```

```

        {
//if so, check against their hitboxes
                checkForHits(p, start, end, mLMeleeWeapons[i]);
        }
    }
}
//same exact thing, but now with the right side weapons
for(int i=0; i<2; i++)
{
    if(mRMeleeWeapons[i].slot!=-1)
    {
        start.set(0,0,0);
        end = mRMeleeWeapons[i].end;

        mat.mul(getRenderTransform(),
getNodeTransform(mRMeleeWeapons[i].node1));
        mat.mulP(start);
        mat.mulP(end);

        ShapeBase* p = findVictim(start, end);

        if(p)
        {
//move this into findvictim in the dist based approach
                checkForHits(p, start, end, mRMeleeWeapons[i]);
        }
    }
}

//find the player that could be hit by the weapon
ShapeBase* ShapeBase::findVictim(Point3F start, Point3F end)
{
    RayInfo rinfo;

    disableCollision();
//do a castRay to determine if a player could be hit by the weapon
    if (gServerContainer.castRay(start, end, ShapeBaseObjectType, &rinfo) == true)
    {
//if we find someone, check if we collide
        enableCollision();
        ShapeBase *p = dynamic_cast<ShapeBase *>( rinfo.object );
        return p;
    }

    enableCollision();

//cycle through each client, and see if they are close enough to possibly collide with

```

```

//There is significant room for optimization here. The fewer times we check
//for collisions, the better.
    SimGroup *clG = Sim::getClientGroup();
    for( SimGroup::iterator i = clG->begin(); i != clG->end(); i++ )
    {
        GameConnection *t = dynamic_cast<GameConnection *>( *i );
        if( t )
        {
            ShapeBase *p = dynamic_cast<ShapeBase *>( t-
>getControlObject() );
            if( p )
            {
                if (!(p == this)&&!p->isGhost())
                {
                    Point3F us = getPosition();
                    Point3F them = p->getPosition();

                    F32 xDist = (us.x-them.x);
                    F32 yDist = (us.y-them.y);
                    F32 zDist = (us.z-them.z);
                    F32 distSqr = xDist*xDist + yDist*yDist +
zDist*zDist;
                    //90 is a completely arbitrary number. I did some quick testing, and it seemed like it gave
                    //decent results
                    //IDEA: We could do a if, else if, else if, else with different distances, then check
                    //against just weapons, then weapons and arms, then full body.
                    //not pretty, but it would work
                    if(mAbs(distSqr)<=90)
                    {
                        return p;
                    }
                }
            }
        }
    }

    return NULL;
}

//LastHitbox should be unnecessary

```

```

void ShapeBase::checkForHits(ShapeBase *p, Point3F start, Point3F end, MeleeWeapon
&mMeleeWeapon)
{
    MatrixF mat;
    Box3F Temp;
    int HitLocation = -1;
    int WepCol = -1;
    bool done = false;

    //first check our weapons against the other player's weapons.
    for(int i=0; i<2; i++)
    {
        if(p->mLMeleeWeapons[i].slot!=-1)
        {
            if(done!=true)
            {
                mat.mul(p->getRenderTransform(), getNodeTransform(p-
>mLMeleeWeapons[i].node1));
                if(p->CheckSwing(10, p->mLMeleeWeapons[i].col,
mMeleeWeapon, mat, start, end))
                {
                    //if we do collide, note what weapon we collide with, and we don't need to do any
                    //more collisions
                    WepCol = mLMeleeWeapons[i].slot;
                    done = true;
                }
            }
        }
    }

    for(int i=0; i<2; i++)
    {
        //same thing, but check for the right side weapons on the other player this time
        if(p->mRMeleeWeapons[i].slot!=-1)
        {
            if(done!=true)
            {
                mat.mul(p->getRenderTransform(), getNodeTransform(p-
>mRMeleeWeapons[i].node1));
                if(p->CheckSwing(10, p->mRMeleeWeapons[i].col,
mMeleeWeapon, mat, start, end))
                {
                    WepCol = mRMeleeWeapons[i].slot;
                    done = true;
                }
            }
        }
    }
}

```

```

//if the other player didn't block our swing, check to see if we actually hit him
//cycle through the hitboxes and check each
    for(int i = 0; i<p->mDataBlock->numBoxes; i++)
    {
        if(done!=true)
        {
            mat.mul(p->getRenderTransform(), p->getNodeTransform(p-
>mHitboxes[i].nodeName));
            if(p->CheckSwing(10, p->mHitboxes[i], mMeleeWeapon, mat,
start, end))
            {
                HitLocation = i;
                done = true;
            }
        }
    }

    if(HitLocation!=-1)
    {
        //execute this console function. The function is passed three parameters.
        //first: who we hit
        //second: who hit him
        //third: where he was hit
        //the following commented out lines show how you can see the type
//of weapon that hit the player, in case that is important in your game.
        //ShapeBaseImageData* weapA =
getMountedImage(mMeleeWeapon.slot);
        //if(weapA)
        //Con::evaluatef( "HitBoxes(%d, %d, %d, %d);", p->getId(), getId(),
HitLocation, weapA->getId() );

        Con::evaluatef( "HitBoxes(%d, %d, %d, %d);", p->getId(), getId(),
HitLocation, 0 );
    }

//if we collided with a weapon:
    if(WepCol!=-1)
    {
        //execute this console function. The function is passed three parameters.
        //first: who we hit
        //second: who hit him
        //third: left = 1, right = 2
        //WepCol = slot of defender
        //mMeleeWeapon.slot = slot of attacker
//same thing as before, but this time you can tell the type of both weapons.

```

```

        //ShapeBaseImageData* weapA =
getMountedImage(mMeleeWeapon.slot);
        //ShapeBaseImageData* weapD = getMountedImage(WepCol);
        //if(weapA && weapD)
        //Con::evaluatef( "WeaponHit(%d, %d, %d, %d);", p->getId(), getId(),
weapA->getId(), weapD->getId() );
        Con::evaluatef( "WeaponHit(%d, %d, %d, %d);", p->getId(), getId(), 0, 0 );

    }

}

```

//OVERVIEW: (read this) This function interpolates the position of the weapon and the hitbox, based on the previous position and the current position. The more Interpolations, the more often it checks, thus the more accurate it is. But it will also take more resources with more interpolations.

```

bool ShapeBase::CheckSwing(int Interpolations, Hitbox& Hitbox, MeleeWeapon&
MeleeWeapon, MatrixF mat, Point3F Start, Point3F End)
{

```

```

    //Parameters:
    //Interpolations = Number of interpolations between the last frame and this frame.
    //NOTE: The limb CAN pass by the sword in this code. But usually limbs don't
move that fast, and they have wide enough hitboxes.
    //Hitbox = the box we are checking against
    //MeleeWeapon = the weapon that could be colliding with the.
    //mat = THIS IS IMPORTANT. mat is actually the bone->world transform. To
check our world points against the box, we first need to get them to bone space. So, we
take the inverse of mat, thus giving us a world->bone transform. I hope that makes sense
:)

```

```

    //take the inverse to get the world->bone transformation
    mat.inverse();
    //These points are going to be reset after this function finishes anyway, so we can
afford to change the points. They need to be converted to bone space
    mat.mulP(Hitbox.lastLocation.max);
    mat.mulP(Hitbox.lastLocation.min);
    //We cant however, change the hitbox itself, so we just make a temporary one.
    Box3F TEMP;
    //Some temporary start and end points for teh interpolation loop
    Point3F tempSTART;
    Point3F tempEND;
    //loop through the number of interpolations.
    for(int q = 1; q<=Interpolations; q++)
    {
        //get the dimensions of the interpolated box. Hopefully you understand how we
are interpolating. All I did was find the difference between this and last frame, then

```

divide that by the number of interpolations. That difference is multiplied by the number of the loop, then added to the beginning point, thus giving us the interpolated position.

```
    TEMP.max = (((Hitbox.lastLocation.max - Hitbox.hitbox.max)/Interpolations)*q)
+ Hitbox.hitbox.max;
    TEMP.min = (((Hitbox.lastLocation.min - Hitbox.hitbox.min)/Interpolations)*q)
+ Hitbox.hitbox.min;
    //interpolate the melee weapon positions.
    tempSTART = MeleeWeapon.lastStart+(((Start -
MeleeWeapon.lastStart)/Interpolations)*q);
    tempEND = MeleeWeapon.lastEnd+(((End -
MeleeWeapon.lastEnd)/Interpolations)*q);
    //transform the melee weapon coords into bone space.
    mat.mulP(tempSTART);
    mat.mulP(tempEND);
    //and finally, collide our interpolated line and box. If its true, we return. Other wise,
we go onto the next interpolation.
    if(TEMP.collideLine(tempSTART, tempEND))
    {
        return true;
    }

}
//if we still haven't hit, we give up.
return false;

}
```

```
bool ShapeBase::checkProjectileCollisionHB(Point3F start, Point3F end, Point3F
vel, Point3F rayPoint, MatrixF &xform, int &collided)
```

```
{
    // if there are no hitboxes the xform
    // is only relative to the bounds object

    if(mHitboxes == NULL)
    {
        MatrixF mat = getRenderTransform();
        mat.inverse();
        xform = MathUtils::createOrientFromDir(vel);
        xform.setPosition(rayPoint);
        mat.mul(xform);
        xform = mat;
        collided = mDataBlock->numBoxes+1;
        return true;
    }
}
```

```

Point3F tpoint, n;
F32 t, shortest_distance = 10000, distance = 0;

MatrixF mat, renderTransform = getRenderTransform();
collided = -1;

if(vel.isZero())
    vel.set(0,0,1);
else
    vel.normalize();

// here we compare the line (described by start and end)
// to the hitboxes.
//
// we compare it with all hitboxes in order to get the
// closest point to the start of the line, which would
// be the first point of intersection.
//
// you could also sort the boxes...

//for(int i=0; i < mDataBlock->mNumHitboxes; i++)
//Raistlin: Changed to:
for(int i=0; i < mDataBlock->numBoxes; i++)
{
    Point3F tvel = vel;
    // bone->world

    mat.mul(renderTransform,
getNodeTransform(mHitboxes[i].nodeName));

    Box3F temp;
    Point3F tstart = start, tend =end;

    temp.max = mHitboxes[i].hitbox.max;
    temp.min = mHitboxes[i].hitbox.min;

    // put the temp box in world transform
    mat.mulP(temp.max);
    mat.mulP(temp.min);
    mat.inverse();
    // put the temp box and line in bone transform
    mat.mulP(temp.max);
    mat.mulP(temp.min);
    mat.mulP(tstart);
    mat.mulP(tend);

    if(temp.collideLine(tstart, tend, &t, &n))

```

```

        {
            tpoint.interpolate(tstart, tend, t);
            // find the closest box which is intersected with, this would
be hit first
            distance = sqrt( (tpoint.x - tstart.x)*(tpoint.x - tstart.x) +
(tpoint.y - tstart.y)*(tpoint.y - tstart.y) + (tpoint.z - tstart.z)*(tpoint.z - tstart.z) );
            if(distance < shortest_distance)
            {
                collided = i;
                shortest_distance = distance;

                MatrixF temp = mat;
                xform = MathUtils::createOrientFromDir(vel);
                temp.mul(xform);
                //xform.mul(mat);
                temp.setPosition(tpoint);
                xform = temp;
            }
        }
    }
    if(collided == -1)
        return false;
    else
        return true;
}

```

...

```
void ShapeBase::renderObject(SceneState* state, SceneRenderImage* image)
{

```

...

```

    else
    {
        renderImage(state, image);
    }
    renderAttachedProjectiles(state);

```

...

//BE SURE THESE LINES ARE COMMENTED OUT: Yes, I'm aware that it comments out the bracket. Just follow along and make the changes I indicate, and everything will work out pretty. Be sure to comment out the last bracket before the next section that we add in

```

    // Debugging Bounding Box
    //if (!mShapeInstance || gShowBoundingBox) {

```

...

```

    for (U32 i = 0; i < MaxMountedImages; i++) {
        MountedImage& image = mMountedImageList[i];
        if (image.dataBlock && image.shapeInstance) {

```

```

...
    wireCube(Point3F(0.05,0.05,0.05),Point3F(0,0,0));
    glPopMatrix();
}
//} //comment out this bracket
//The final matrix to be applied to the points.
MatrixF mat;
//The transformation from player space to world space
MatrixF renderTransform;

        renderTransform = getRenderTransform();
//Loop through the hitboxes (for this one client)
for(int i = 0; i < mDataBlock->numBoxes; i++)
{
        //Multiply the player->world transform by the bone->player
matrix to get, obviously, the bone->world... :) (stores the matrix in mat)
        //I like the color red. :)
    glColor3f(1, 0, 0);
        //call our previous function, and pass it the dimensions of our hitbox. Those
will be transformed to world coords that are centered on the limb, and drawn. Just what
we want. :)
        mat.mul(renderTransform,
getNodeTransform(mHitboxes[i].nodeName));
        //call our previous function, and pass it the dimensions of our hitbox. Those
will be transformed to world coords that are centered on the limb, and drawn. Just what
we want. :)
    DrawCollideHitbox (mHitboxes[i].hitbox, mat);

}

    glColor3f(1, 1, 0);
//Tell the OpenGL wrapper that we are starting to draw lines

//glBegin(GL_LINES);

//Draw the line....
//MyLine(mMeleeWeapon.Start, mMeleeWeapon.End);
Point3F start;
Point3F end;
for(int i = 0; i<2; i++)
{
//if there are any weap[ons, transform the points to world cords and display
    if(mLMeleeWeapons[i].slot!=-1)
    {
        start.set(0,0,0);
        end = mLMeleeWeapons[i].end;
    }
}

```

```

        mat.mul(getRenderTransform(),
getNodeTransform(mLMeleeWeapons[i].node1));
        mat.mulP(start);
        mat.mulP(end);
        glColor3f(0, 0, 1);
        glBegin(GL_LINES);
        MyLine(start, end);
        glEnd();
        glColor3f(0, 1, 1);
        DrawCollideHitbox (mLMeleeWeapons[i].col.hitbox, mat);
    }
}
glColor3f(0, 1, 0);
for(int i = 0; i<2; i++)
{
    if(mRMeleeWeapons[i].slot!=-1)
    {
        start.set(0,0,0);
        end = mRMeleeWeapons[i].end;
        mat.mul(getRenderTransform(),
getNodeTransform(mRMeleeWeapons[i].node1));
        mat.mulP(start);
        mat.mulP(end);
        glColor3f(1, 0, 1);
        glBegin(GL_LINES);
        MyLine(start, end);
        glEnd();
        glColor3f(0, 1, 1);
        DrawCollideHitbox (mRMeleeWeapons[i].col.hitbox,
mat);
    }
}

```

```

//Stop drawing lines
//glEnd();

```

```

//get the server player....
//do the exact same thing for the server player
ShapeBase *p=NULL;
p=dynamic_cast<ShapeBase* > ((NetObject *)mServerObject);
if(p)
{

```

```

        renderTransform = p->getRenderTransform();
//Loop through the hitboxes (for this one client)

```

```

    for(int i = 0; i<p->mDataBlock->numBoxes; i++)
    {
        //Multiply the player->world transform by the bone->player matrix to get,
        obviously, the bone->world... :) (stores the matrix in mat)
        mat.mul(renderTransform, p->getNodeTransform(p-
>mHitboxes[i].nodeName));
        //I like the color black. :)
        glColor3f(0, 0, 0);
        //call our previous function, and pass it the dimensions of our hitbox. Those
        will be transformed to world coords that are centered on the limb, and drawn. Just what
        we want. :)
        DrawCollideHitbox (p->mHitboxes[i].hitbox, mat);
    }

    //Tell the OpenGL wrapper that we are starting to draw lines
    //glBegin(GL_LINES);

    for(int i = 0; i<2; i++)
    {
        if(p->mLMeleeWeapons[i].slot!=-1)
        {
            start.set(0,0,0);
            end = p->mLMeleeWeapons[i].end;
            mat.mul(p->getRenderTransform(), p-
>getNodeTransform(p->mLMeleeWeapons[i].node1));
            mat.mulP(start);
            mat.mulP(end);
            glColor3f(1, 1, 1);
            glBegin(GL_LINES);
            MyLine(start, end);
            glEnd();
            glColor3f(1, 1, 0);
            DrawCollideHitbox (p->mLMeleeWeapons[i].col.hitbox,
mat);
        }
    }
    glColor3f(1, 0, 1);
    for(int i = 0; i<2; i++)
    {
        if(p->mRMeleeWeapons[i].slot!=-1)
        {
            start.set(0,0,0);
            end = p->mRMeleeWeapons[i].end;
            mat.mul(p->getRenderTransform(), p-
>getNodeTransform(p->mRMeleeWeapons[i].node1));
            mat.mulP(start);
            mat.mulP(end);

```

```

        glColor3f(0, 1, 0);
        glBegin(GL_LINES);
        MyLine(start, end);
        glEnd();
        glColor3f(1, 1, 0);
        DrawCollideHitbox (p->mRMeleeWeapons[i].col.hitbox,
mat);
    }
}

//MyLine(p->mMeleeWeapon.Start, p->mMeleeWeapon.End);
//Stop drawing lines
//glEnd();

}

glEnable(GL_DEPTH_TEST);

dglSetCanonicalState();
TextureManager::setSmallTexturesActive(false);

AssertFatal(dglIsInCanonicalState(), "Error, GL not in canonical state on exit");
PROFILE_END();
}

...

// goes through each attached projectile and invokes
// its attached rendering method
void ShapeBase::renderAttachedProjectiles(SceneState* state)
{
    SceneRenderImage* image;
    for(attachedProjectile *curr = attachedProjectiles; curr != NULL; curr =
curr->next)
    {
        curr->p->attachedRender(state);
    }
}

// attaches a projectile to our projectile list, the hit
// box id is stored as well for when we detach them
void ShapeBase::attachProjectile(Projectile *proj, int hbid)
{
    // if we have no head, make one
    if(attachedProjectiles == NULL)
    {
        attachedProjectiles = new attachedProjectile;
        attachedProjectiles->p = proj;
    }
}

```

```

    attachedProjectiles->aHBID = hbid;
    attachedProjectilesTail = attachedProjectiles;
}
else
{
    // add to the tail
    attachedProjectilesTail->next = new attachedProjectile;
    attachedProjectilesTail->next->p = proj;
    attachedProjectilesTail->next->aHBID = hbid;
    attachedProjectilesTail = attachedProjectilesTail->next;
}
}
// the ghost method for detaching the entire
// projectile list, doesn't need to
// delete the actual projectile
// as that's handled by the simdelete
void ShapeBase::detachAllProjectilesGhost()
{
    attachedProjectile *last = NULL;
    attachedProjectile* curr=attachedProjectiles;
    while(curr != NULL)
    {
        last = curr;
        curr=curr->next;
        delete last;
    }
    attachedProjectiles = NULL;
    attachedProjectilesTail = NULL;
}
// the detach all projectiles method
// called on the server player
void ShapeBase::detachAllProjectiles()
{
    attachedProjectile *last = NULL;
    attachedProjectile* curr=attachedProjectiles;
    while(curr != NULL)
    {
        last = curr;
        curr=curr->next;
        last->p->deleteObject();
        delete last;
    }
    attachedProjectiles = NULL;
    attachedProjectilesTail = NULL;

    dProjectile = false;
    dHitbox = false;
}

```

```

        setMaskBits(detachMask);
    }
    // same issue as detach all projectiles
    // but these methods only detach them
    // by the limb, so you can 'heal' only
    // one arm for example
    void ShapeBase::detachProjectilesGhost(int limb)
    {
        attachedProjectile *last = NULL;
        attachedProjectile *del = NULL;

        for(attachedProjectile* curr=attachedProjectiles; curr!=NULL;
curr=curr->next)
        {
            delete del;
            del = NULL;
            if(curr->aHBID == limb)
            {
                if(last == NULL)
                    attachedProjectiles = curr->next;
                else
                    last->next = curr->next;

                if(curr == attachedProjectilesTail)
                    attachedProjectilesTail = last;

                del = curr;
            }
            else
                last = curr;
        }
    }
    void ShapeBase::detachProjectiles(int limb)
    {
        attachedProjectile *last = NULL;
        attachedProjectile *del = NULL;

        for(attachedProjectile* curr=attachedProjectiles; curr!=NULL;
curr=curr->next)
        {
            delete del;
            del = NULL;

            if(curr->aHBID == limb)
            {
                if(last == NULL)
                    attachedProjectiles = curr->next;
                else

```

```

        last->next = curr->next;

        if(curr == attachedProjectilesTail)
            attachedProjectilesTail = last;

        del = curr;
        curr->p->deleteObject();
    }
    else
        last = curr;
}
dProjectile = false;
dHitbox = true;
detachValue = limb;
setMaskBits(detachMask);
}

```

```

void ShapeBase:: DrawCollideHitbox (Box3F& Box, MatrixF& mat)
{

```

//This code is old. I used to use 8 points for the collision, and I didnt feel like changing the drawing code. Dont worry that it is overly complex.

```

//Make 8 points for the cube
Point3F FUL; // FRONT UPPER LEFT
Point3F FUR; // FRONT UPPER RIGHT
Point3F BUL; // BACK UPPER LEFT
Point3F BUR;
Point3F FLL; //FRONT LOWER LEFT
Point3F FLR;
Point3F BLL;
Point3F BLR;
Point3F START;
Point3F END;
//initialise them based on the box dimensions
FUL.set(Box.min.x,Box.max.y,Box.max.z);
FUR.set(Box.max.x,Box.max.y,Box.max.z);
BUL.set(Box.min.x,Box.max.y,Box.min.z);
BUR.set(Box.max.x,Box.max.y,Box.min.z);
FLL.set(Box.min.x,Box.min.y,Box.max.z);
FLR.set(Box.max.x,Box.min.y,Box.max.z);
BLL.set(Box.min.x,Box.min.y,Box.min.z);
BLR.set(Box.max.x,Box.min.y,Box.min.z);

```

//multiply the points by the matrix we are passed. I will explain this when we use the function.

```

    mat.mulP(FUL);

```

```

mat.mulP(FUR);
mat.mulP(BUL);
mat.mulP(BUR);
mat.mulP(FLL);
mat.mulP(FLR);
mat.mulP(BLL);
mat.mulP(BLR);
//Begin lines. This is a GL command
    glBegin(GL_LINES);
//We have to define this function(MyLine). I couldnt find a simple line drawing function,
so I made my own.
    MyLine(FUL, FUR);
    MyLine(FUR, FLR);
    MyLine(FLR, FLL);
    MyLine(FLL, FUL);

    MyLine(BUL, BUR);
    MyLine(BUR, BLR);
    MyLine(BLR, BLL);
    MyLine(BLL, BUL);

    MyLine(FUL, BUL);
    MyLine(FLL, BLL);
    MyLine(FUR, BUR);
    MyLine(FLR, BLR);

    glEnd();
//Stop drawing lines.... :)

}

//Define the MyLine function
//Yeah, there is probably already a function for this, but I didnt feel like finding it.
void ShapeBase::MyLine(const Point3F& start, const Point3F& end)
{
    //create two vertices. This will tell openGL to make a line. Very simple. :)
    glVertex3f(start.x, start.y, start.z);
    glVertex3f(end.x, end.y, end.z);
}

...

U32 ShapeBase::packUpdate(NetConnection *con, U32 mask, BitStream *stream)
{
    ...
    if (stream->writeFlag(mask & detachMask)) {

```

```

        if(stream->writeFlag(dHitbox))
            stream->writeInt(detachValue,
ShapeBaseData::HITBOX_DETACH_BITS);
    }
    return retMask;
}

void ShapeBase::unpackUpdate(NetConnection *con, BitStream *stream)
{
...
    if (stream->readFlag()) // detachMask
    {
        if(stream->readFlag())
        {
            detachValue = stream-
>readInt(ShapeBaseData::HITBOX_DETACH_BITS);
            detachProjectilesGhost(detachValue);
        }
        else
            detachAllProjectilesGhost();
    }
}
...

```

//It makes sense to put this after getNodeTransform obviously

```

// Get a node transform for a shape that is mounted to us (in the slot that we're passed)
MatrixF ShapeBase::getNodeTransform2( StringTableEntry nodeName, S32 slot )
{
    TSShape *shape = mShapeInstance->getShape();
    if (!shape || slot<0)
        return MatrixF(true);

    S32 node = shape->findNode( nodeName );
    if (node < 0)
    {
        MountedImage *obj = getImageStruct(slot);
        if(obj)
        {
            shape = obj->shapeInstance->getShape();
            node = shape->findNode(nodeName);
            if(node>=0)
            {
                return obj->shapeInstance->mNodeTransforms[node];
            }
            else
                return MatrixF(true);
        }
    }
}

```

```

        }
        else
            return MatrixF(true);
    }

    return mShapeInstance->mNodeTransforms[node];
}

...

static bool cdetachProjectiles(SimObject *ptr, S32, const char **argv)
{
    ShapeBase* obj = static_cast<ShapeBase*>(ptr);
    int id = dAtoi(argv[2]);
    obj->detachProjectiles(id);
    return true;
}
static bool cdetachAllProjectiles(SimObject *ptr, S32, const char **argv)
{
    ShapeBase* obj = static_cast<ShapeBase*>(ptr);
    obj->detachAllProjectiles();
    return true;
}

...

//make the following functions available from the scripts
void ShapeBase::consoleInit()
{
    ...
    Con::addCommand("ShapeBase", "detachProjectiles", cdetachProjectiles,
"obj.detachProjectiles(\"hitboxID\")", 3, 3);
    Con::addCommand("ShapeBase", "detachAllProjectiles",
cdetachAllProjectiles, "obj.detachAllProjectiles()", 2, 2);

    ...
    Con::addCommand("ShapeBase", "registerMelee", cregisterMelee,
"obj.registerMelee(nodeBegin, nodeEnd, wepID, arm, min, max)", 8, 8);
    Con::addCommand("ShapeBase", "unregisterMelee", cunregisterMelee,
"obj.unregisterMelee(wepID)", 3, 3);
}

```

Add the following code to gameprocess.cc

GAMEPROCESS.CC

```
...
void ProcessList::advanceObjects()
{
...
    }
    if (obj->mProcessTick)
        obj->processTick(0);
    }

    GameBase plist;
    plist.plLinkBefore(head.mProcessLink.next);
    head.plUnlink();

    while ((obj = plist.mProcessLink.next) != &plist)
    {
        obj->plUnlink();
        obj->plLinkBefore(&head);

        if (obj->mTypeMask & ShapeBaseObjectType)
        {
            ShapeBase* s = static_cast<ShapeBase*>(obj);
            s->updateMelee();
        }
    }

    PROFILE_END();
}
...
```

Make the following changes in player.cc This melee collision code should have theoretically worked with any shapebase object, but NOT if it animates, because the only things we have animating on the server are the players.

PLAYER.CC

```
Player::Player()
{
...
mActionAnimation.animateOnServer = true;
}

...

void Player::setActionThread(U32 action,bool forward,bool hold,bool wait,bool fsp, bool
forceSet)
{
...
    mActionAnimation.firstPerson = fsp;
    mActionAnimation.holdAtEnd = hold;
    mActionAnimation.waitForEnd = hold? true: wait;
    mActionAnimation.animateOnServer = true;
    mActionAnimation.atEnd = false;
    mActionAnimation.delayTicks = (S32)sNewAnimationTickTime;
    mActionAnimation.atEnd = false;

    if (sUseAnimationTransitions && (isGhost() ||
mActionAnimation.animateOnServer)) {
        // The transition code needs the timeScale to be set in the
        // right direction to know which way to go.
...

void Player::updateAnimation(F32 dt)
{
    if (isGhost() || mActionAnimation.animateOnServer) //you may need to change
//things here, you may not. Just make sure it looks like that
        mShapeInstance->advanceTime(dt,mActionAnimation.thread);
    if (mRecoilThread)
        mShapeInstance->advanceTime(dt,mRecoilThread);

    // If we are the client's player on this machine, then we need
    // to make sure the transforms are up to date as they are used
    // to setup the camera.
    if (isGhost() || mActionAnimation.animateOnServer) {
        if (getControllingClient()) {
            updateAnimationTree(isFirstPerson());
```


Make the following changes to projectile.h

PROJECTILE.H

```
class ProjectileData : public GameBaseData
{
...
public:
...
bool isSticky;
...
}

...

class Projectile : public GameBase
{
...
public:
...
enum UpdateMasks {
    BounceMask    = Parent::NextFreeMask,
    ExplosionMask = Parent::NextFreeMask << 1,
    //be sure this next line is commented out, then add the next two
    // NextFreeMask = Parent::NextFreeMask << 2,
    AttachedMask = Parent::NextFreeMask << 2,
    NextFreeMask = Parent::NextFreeMask << 3
...
public:
...
    void attachedRender(SceneState* state);
    ShapeBase *attachedObject;
    int aHBID;
    MatrixF attachedTransform;
...
}
```

Luigi “Cirrus” Rosso wrote most of this code. There are fewer comments because I haven’t taken the time to read every line of his code and understand exactly what he did, though I have a pretty good idea because we worked pretty closely getting it to work. I’d be happy to answer any questions on it, because that way it will force me to look at it and understand it. ☺ I am sure that it works though, because I’ve tested it pretty thoroughly.

Make the following changes in projectile.cc

PROJECTILE.CC

```
ProjectileData::ProjectileData()
{
...
//Raistlin: Set this to true or false depending on whether you want
//projectiles to stick in people or not by default (true = they stick in by default)
//is set in the datablocks obviously. I have it set to true so you can be sure it works
    isSticky = true;
...
}

void ProjectileData::initPersistFields()
{
...
    addNamedField(isSticky, TypeBool, ProjectileData);
...
}

void ProjectileData::packData(BitStream* stream)
{
...
    stream->writeFlag(isSticky);
...
}

void ProjectileData::unpackData(BitStream* stream)
{
...
    isSticky = stream->readFlag();
...
}

Projectile::Projectile()
{
```

```

...
    attachedObject = NULL;
    aHBID = -1;
...
}

void Projectile::processTick(const Move* move)
{
    Parent::processTick(move);

    if(attachedObject != NULL)
        return;
...

    if (getContainer()->castRay(oldPosition, newPosition,
        csmDynamicCollisionMask | csmStaticCollisionMask,
        &rInfo) == true)
    {
        if(isServerObject() && (rInfo.object->getType() &
            csmStaticCollisionMask) == 0 && !mDataBlock->isSticky)
            setMaskBits(BounceMask);

        if(isServerObject() && (rInfo.object->getType() &
            ShapeBaseObjectType) !=
            0 && mDataBlock->isSticky)
        {
            ShapeBase *obj=NULL;
            obj=dynamic_cast<ShapeBase* > ((NetObject *)rInfo.object);
            Point3F attachedPoint;

            if(obj->checkProjectileCollisionHB(oldPosition, newPosition,
                mCurrVelocity, rInfo.point, attachedTransform, aHBID))
            {
                attachedObject = obj;
                attachedObject->attachProjectile(this, aHBID);
                //calls onAttach for the particular projectile.
                //is passed:
                //the projectile that hit
                //what it hit
                //where it hit

                Con::executef(mDataBlock, 4, "onAttach",
                    scriptThis(),
                    Con::getIntArg(attachedObject->getId()),
                    Con::getIntArg(aHBID));
                setMaskBits(AttachedMask);
                break;
            }
        }
    }
}

```

```

    }
}
...

void Projectile::interpolateTick(F32 delta)
{
    Parent::interpolateTick(delta);

    if(attachedObject != NULL)
        return;
}
...

U32 Projectile::packUpdate(NetConnection* con, U32 mask, BitStream* stream)
{
    ...
    else if (stream->writeFlag(mask & BounceMask))
    {
        // Bounce against dynamic object
        mathWrite(*stream, mCurrPosition);
        mathWrite(*stream, mCurrVelocity);
    }

    else if(stream->writeFlag(mask & AttachedMask))
    {
        // projectile is attached to a player
        S32 ghostIndex = con->getGhostIndex(attachedObject);
        if (stream->writeFlag(ghostIndex != -1))
        {
            stream->writeRangedU32(U32(ghostIndex), 0,
NetConnection::MaxGhostCount);
            ShapeBaseData *db =
static_cast<ShapeBaseData*>(attachedObject->getDataBlock());
            //stream->writeRangedU32(U32(aHBID), 0, db-
>mNumHitboxes+1);
            //Raistlin: Changed to:
            stream->writeRangedU32(U32(aHBID), 0, db->numBoxes+1);
            stream->writeAffineTransform(attachedTransform);
        }
    }

    return retMask;
}
...

```



```

}
else
{
    // check if we have an attached object
    if(attachedObject != NULL)
    {
        MatrixF limbWorld;

        // if the hitbox id is equivalent to the num of hitboxes + 1 then
        // transform relative to object's bounding box (or pivot)

        if(aHBID != attachedObject->mDataBlock->numBoxes+1)
            limbWorld.mul(attachedObject->getRenderTransform(),
attachedObject->getNodeTransform(attachedObject->mHitboxes[aHBID].nodeName));
        else
            limbWorld = attachedObject->getRenderTransform();

        limbWorld.mul(attachedTransform);
        setTransform( limbWorld );
        setRenderTransform( limbWorld );
        dglMultMatrix( &limbWorld );
    }
    else
        dglMultMatrix( &getRenderTransform() );
}
}

```

```

void Projectile::attachedRender(SceneState* state)
{
    RectI viewport;
    glMatrixMode(GL_PROJECTION);
    glPushMatrix();
    dglGetViewport(&viewport);

    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();

    prepModelView( state );

    glScalef( mDataBlock->scale.x, mDataBlock->scale.y, mDataBlock->scale.z
);

    if(mProjectileShape)
    {
        AssertFatal(mProjectileShape != NULL, "Projectile::renderObject: Error, projectile
shape should always be present in renderObject");
    }
}

```

```

mProjectileShape->selectCurrentDetail();
mProjectileShape->animate();

Point3F cameraOffset;
mObjToWorld.getColumn(3,&cameraOffset);
cameraOffset -= state->getCameraPosition();
F32 fogAmount = state->getHazeAndFog(cameraOffset.len(),cameraOffset.z);

if (mFadeValue == 1.0) {
    mProjectileShape->setupFog(fogAmount, state->getFogColor());
} else {
    mProjectileShape->setupFog(0.0, state->getFogColor());
    mProjectileShape->setAlphaAlways(mFadeValue * (1.0 - fogAmount));
}
mProjectileShape->render();
}

glDisable(GL_BLEND);
glDisable(GL_TEXTURE_2D);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);

glMatrixMode(GL_MODELVIEW);
glPopMatrix();

glMatrixMode(GL_PROJECTION);
glPopMatrix();
glMatrixMode(GL_MODELVIEW);
dglSetViewport(viewport);

}

bool Projectile::prepRenderImage(SceneState* state, const U32 stateKey,
                                const U32 /*startZone*/, const bool /*modifyBaseState*/)
{
    if (isLastState(state, stateKey))
        return false;
    setLastState(state, stateKey);

    if(attachedObject != NULL)
        return false;

    ...

```

Add this to game.cs (/torque/example/fps/server/scripts):

GAME.CS

```
exec("./meleeWeapon.cs");  
exec("./sword.cs");
```

Add this file in ... \torque\example\fps\server\scripts\

MELEEWEAPON.CS

```
//-----  
// Copyright (c) 2001 Epsilon Entertainment  
//-----  
  
// This file contains Weapon and Ammo Class "namespace" helper methods  
// as well as hooks into the inventory system. These functions are not  
// attached to a specific C++ class or datablock, but define a set of  
// methods which are part of dynamic namespaces "class". The Items  
// include these namespaces into their scope using the ItemData and  
//ItemImageData "className" variable.  
  
// All ShapeBase images are mounted into one of 8 slots on a shape.  
// This weapon system assumes all primary weapons are mounted into  
// this specified slot:  
  
//Obviously, this code was modified by Raistlin. It was done to provide some nice  
support for meleeweapons.  
$WeaponSlot = 0;  
  
//-----  
// Weapon Class  
//-----  
  
function MeleeWeapon::onUse(%data,%obj)  
{  
    // Default behavoir for all weapons is to mount it into the  
    // this object's weapon slot, which is currently assumed  
    // to be slot 0  
  
    if (%obj.getMountedImage(%data.slot) != %data.image.getId()) {  
        %obj.mountImage(%data.image, %data.slot);  
  
        if (%obj.client)  
            messageClient(%obj.client, 'MsgWeaponUsed', "\c0Weapon selected");  
    }  
}  
  
//Raistlin: This needs work.  
function MeleeWeapon::onPickup(%this, %obj, %shape, %amount)  
{  
    // The parent Item method performs the actual pickup.  
    // For player's we automatically use the weapon if the  
    // player does not already have one in hand.
```

```

if (Parent::onPickup(%this, %obj, %shape, %amount)) {
  if (%shape.getClassName() $= "Player" &&
      %shape.getMountedImage(%this.slot) == 0)
  {
    //if (%shape.getClassName() $= "Player")
    //{
      %shape.use(%this);
    }
  }
}
}

function MeleeWeapon::onInventory(%this,%obj,%amount)
{
  echo("MeleeWeapon::onInventory");
  // Weapon inventory has changed, make sure there are no weapons
  // of this type mounted if there are none left in inventory.
  if (!%amount && (%slot = %obj.getMountSlot(%this.image)) != -1)
    %obj.unmountImage(%slot);
}

```

```

//-----
// Weapon Image Class
//-----

```

```

function MeleeWeaponImage::onMount(%this,%obj,%slot)
{
  if(%this.arm != 3)
  {
    %obj.registerMelee(%this.Node1, %this.Node2,
    %this, %this.Arm, %this.Min, %this.Max);
  }
  //Raistlin: this shouldnt be necessary, but let's play nice anyway and pretend we have
  ammo.
  %obj.setImageAmmo(%slot,true);
}

```

```

function MeleeWeaponImage::onUnmount(%this,%obj,%slot)
{
  %obj.unregisterMelee(%slot);
  //I wonder if this works... isn't that an encouraging thing to see? ☺
  echo("Unmounting?");
}

```

Add this file in ... \torque\example\fps\server\scripts\

SWORD.CS

```
//-----  
//Sword. By Josh Albrecht (Raistlin)  
//-----  
  
//-----  
// Weapon Item. This is the item that exists in the world, i.e. when it's  
// been dropped, thrown or is acting as re-spawnable item. When the weapon  
// is mounted onto a shape, the RifleImage is used.  
  
datablock ItemData(Sword)  
{  
    // Mission editor category  
    category = "Weapon";  
  
    // Hook into Item Weapon class hierarchy. The weapon namespace  
    // provides common weapon handling functions in addition to hooks  
    // into the inventory system.  
    className = "MeleeWeapon";  
    //className = "Weapon";  
  
    // Basic Item properties  
    shapeFile = "~/data/shapes/player/weapon_gladius_03_01.dts";  
    mass = 1;  
    elasticity = 0.2;  
    friction = 0.6;  
    emap = true;  
  
    // Dynamic properties defined by the scripts  
    pickUpName = "a sword";  
    image = SwordImage;  
  
    cost = 8;  
    slot = 0;  
};  
  
//-----  
// Sword image which does all the work. Images do not normally exist in  
// the world, they can only be mounted on ShapeBase objects.  
  
datablock ShapeBaseImageData(SwordImage)  
{  
    // Basic Item properties  
    //shapeFile = "~/data/shapes/player/weapon_gladius_03_01.dts";
```

```

shapeFile = "~/data/shapes/player/testsword1.dts";
emap = true;

//names of the start and end nodes for the melee weapon, right or left arm, then the size
// of the box used for collision with other weapons
Node1 = "mount0";
Node2 = "tip";
Arm = 1;
Min = "-0.1 -0.1 -0.2";
Max = "0.1 0.1 1.2";

// Specify mount point & offset for 3rd person, and eye offset
// for first person rendering.
//if you want a shield, make a weapon with a large bounding box that does no damage
//if you want something on the other arm, or in some other place, change the
//next line according to the chart below
mountPoint = 0;
//0 = right
//1 = left
//2 = blank?
//3 = head
//4 = torso
//5 = R wrist
//6 = L Wrist
//7 = R foot
//8 = L Foot
offset = "0 0 0";
eyeOffset = "0.1 0.2 -0.55";

// Add the WeaponImage namespace as a parent, WeaponImage namespace
// provides some hooks into the inventory system.
className = "MeleeWeaponImage";

// Projectile & Ammo.
item = Sword;

// Images have a state system which controls how the animations
// are run, which sounds are played, script callbacks, etc. This
// state system is downloaded to the client so that clients can
// predict state changes and animate accordingly. The following
// system supports basic ready->fire->reload transitions as
// well as a no-ammo->dryfire idle state.

// Initial start up state
stateName[0] = "Preactivate";
stateTransitionOnLoaded[0] = "Activate";
stateTransitionOnNoAmmo[0] = "NoAmmo";

```

```

// Activating the gun. Called when the weapon is first
// mounted and there is ammo.
stateName[1]          = "Activate";
stateTransitionOnTimeout[1] = "Ready";
stateTimeoutValue[1]   = 0.5;
stateSequence[1]       = "Activate";

// Ready to fire, just waiting for the trigger
stateName[2]          = "Ready";
stateTransitionOnNoAmmo[2] = "NoAmmo";
stateTransitionOnTriggerDown[2] = "Fire";

// Fire the weapon. Calls the fire script which does
// the actual work.
stateName[3]          = "Fire";
stateTransitionOnTimeout[3] = "Reload";
stateTimeoutValue[3]   = 0.1;
stateFire[3]          = true;
stateRecoil[3]         = LightRecoil;
stateAllowImageChange[3] = false;
stateSequence[3]       = "Fire";
stateScript[3]         = "onFire";
stateEmitter[3]        = RifleFireEmitter;
stateEmitterTime[3]    = 0.3;

// Play the reload animation, and transition into
stateName[4]          = "Reload";
stateTransitionOnNoAmmo[4] = "NoAmmo";
stateTransitionOnTimeout[4] = "Ready";
stateTimeoutValue[4]   = 0.1;
stateAllowImageChange[4] = false;
stateSequence[4]       = "Reload";
stateEjectShell[4]     = true;

// No ammo in the weapon, just idle until something
// shows up. Play the dry fire sound if the trigger is
// pulled.
stateName[5]          = "NoAmmo";
stateTransitionOnAmmo[5] = "Reload";
stateSequence[5]       = "NoAmmo";
stateTransitionOnTriggerDown[5] = "DryFire";

// No ammo dry fire
stateName[6]          = "DryFire";
stateTimeoutValue[6] = 1.0;
stateTransitionOnTimeout[6] = "NoAmmo";
};

```


Add the following functions into server/scripts/commands.cs

COMMANDS.CS

```
//this function can do whatever you want really. I just have it echo the hit location to the
console. You can use:
//%target.getDataBlock().damageObject(%target, %killer,
VectorAdd(%target.getPosition(),%vec), 50, $DamageType::Ground);
//to kill a player. The 50 is the damage. The player health is about 1, so this is massive
overkill... :)
//damage is out of 1 I think
function HitBoxes(%target, %killer, %loc, %weap)
{
    echo(getSimTime());
    //add var
    %killer.lastHit++;
    echo(%killer.lastHit);

    if((%killer.lastHit+500)<getSimTime())
    {
        %helmet = %target.getMountedImage(3);
        echo("Armor Stuff");
        echo(%helmet);
        echo(%helmet.item.armorValue);
        if(%helmet.item.armorValue)
        %armor = %helmet.item.armorValue;
        else
        %armor = 1;

        echo("You were hit!");

        %data = %target.getDatablock();
        %damageMod = 1;
        //a crude switch statement based off of the names of the nodes you could have hit
        if(%data.NodeName[%loc]$="Bip01 Head2")
        {
            echo("Head");
            %damageMod = 3;
        }
        if(%data.NodeName[%loc]$="Bip01 Spine2")
        {
            echo("Spine");
            %damageMod = 2;
        }
        if(%data.NodeName[%loc]$="Bip01 L UpperArm2")
        {
```

```

echo("L UpperArm");
%damageMod = 0;
//%damageMod = 1;
}
if(%data.NodeName[%loc]$="Bip01 R UpperArm2")
{
echo("R UpperArm");
%damageMod = 1;
}
if(%data.NodeName[%loc]$="Bip01 L Forearm2")
{
echo("L Forearm");
%damageMod = 0;
//%damageMod = 0.5;
}
if(%data.NodeName[%loc]$="Bip01 R Forearm2")
{
echo("R Forearm");
%damageMod = 0.5;
}
if(%data.NodeName[%loc]$="Bip01 L Thigh2")
{
echo("L Thigh");
%damageMod = 1;
%target.L_LegWound = true;
}
if(%data.NodeName[%loc]$="Bip01 R Thigh2")
{
echo("R Thigh");
%damageMod = 1;
%target.R_LegWound = true;
}
if(%data.NodeName[%loc]$="Bip01 L Calf2")
{
echo("L Calf");
%damageMod = 0.5;
%target.L_LegWound = true;
}
if(%data.NodeName[%loc]$="Bip01 R Calf2")
{
echo("R Calf");
%damageMod = 0.5;
%target.R_LegWound = true;
}

%damage = (%damagemod * 15)/%armor;

//ServerPlay3D(HitArmorSound, %target.getTransform());

```

```
//ServerPlay3D(OuchSound, %target.getTransform());

%target.damage(%killer, %target.getPosition(), %damage, "Normal");
//%killer.damage(0, %killer.getPosition(), %damage, "Normal");

%killer.lastHit = getSimTime();

}

}

function WeaponHit(%target, %killer, %weapA, %weapD)
{

if((%killer.lastHit+500)<getSimTime())
{
//ServerPlay3D(SwordClashSound, %target.getTransform());
echo("hit weapon!");
%killer.lastHit = getSimTime();
}

}

}
```

PLAYER.CS

In the playerdatablock for lightmalehumandata. Put it right before:

```
footstepSplashHeight = 0.35;
```

```
numBoxes = 10;
```

```
HitMin[0] = "-0.1 -0.22 -0.22";  
HitMax[0] = "0.6 0.22 0.22";  
NodeName[0] = "Bip01 Spine";  
HitMin[1] = "-0.1 -0.15 -0.15";  
HitMax[1] = "0.6 0.15 0.15";  
NodeName[1] = "Bip01 L Forearm";  
HitMin[2] = "-0.1 -0.15 -0.15";  
HitMax[2] = "0.6 0.15 0.15";  
NodeName[2] = "Bip01 R Forearm";  
HitMin[3] = "-0.1 -0.15 -0.15";  
HitMax[3] = "0.6 0.15 0.15";  
NodeName[3] = "Bip01 L Upperarm";  
HitMin[4] = "-0.1 -0.15 -0.15";  
HitMax[4] = "0.6 0.15 0.15";  
NodeName[4] = "Bip01 R Upperarm";  
HitMin[5] = "-0.1 -0.15 -0.15";  
HitMax[5] = "0.6 0.15 0.15";  
NodeName[5] = "Bip01 L Thigh";  
HitMin[6] = "-0.1 -0.15 -0.15";  
HitMax[6] = "0.6 0.15 0.15";  
NodeName[6] = "Bip01 R Thigh";  
HitMin[7] = "-0.1 -0.15 -0.15";  
HitMax[7] = "0.6 0.15 0.15";  
NodeName[7] = "Bip01 L Calf";  
HitMin[8] = "-0.1 -0.15 -0.15";  
HitMax[8] = "0.6 0.15 0.15";  
NodeName[8] = "Bip01 R Calf";  
HitMin[9] = "-0.1 -0.15 -0.15";  
HitMax[9] = "0.3 0.15 0.15";  
NodeName[9] = "Bip01 Head";
```

```
maxInv[SwordAmmo] = 1;
```

```
maxInv[Sword] = 1;
```