

HED - Hitbox Editor

by Hendrik Felix Pohl aka Superku

1 Introduction

1.1 Contents

This hitbox implementation consists of three parts:

1. An editor to create hitbox configurations for complex models, called HED.
2. Two code files, *hitbox.h* and *hitbox.c*.
3. A small example.

1.2 Why HED?

When you are creating a shooter or any kind of game that includes collision detection on animated models and bounding box collision is not precise enough, then feel free to use this implementation.

With HED you can create hitbox configurations for different model files. Suppose that you have 100 enemies with 10 hitboxes each, that would be 1000 additional entities in total which need to be placed and rotated, too. Of course that is an unacceptable solution, that's why this implementation only uses (in this case) 10 hitbox entities in total which are only placed once when you shoot an enemy.

You can set the damage individually per hitbox, so exemplarily a head shot will kill the enemy instantly. The entity will receive the hitbox-ID that was hit by the shoot function, thus you can play different hit animations dependent on that ID.

You can use any model as a hitbox, the collision detection is **polygonal** on those. As a result, you can even split up your models in (low) poly parts without texture and use them as hitboxes.

2 How-to

2.1 How it works

All Entities with a hitbox configuration are surrounded by a (mostly) large cube that includes all vertex positions of all frames (`c_setminmax()`). Shoot with a special function that replaces your `c_trace` command (`hitbox_shoot(...)`). If you hit the said cube, the hitboxes will be placed accordingly to the configuration of the model file. A second short range `c_trace` checks if any hitbox was hit, otherwise the `c_trace` will continue from that place and ignore the already checked entity (that's important when enemies stand in a row or are close to another).

2.2 Using HED

Start *HED.exe*. Load your model, f.i. „*tD4-2.mdl*“, and load the hitbox configuration (click the button below „Remove“).

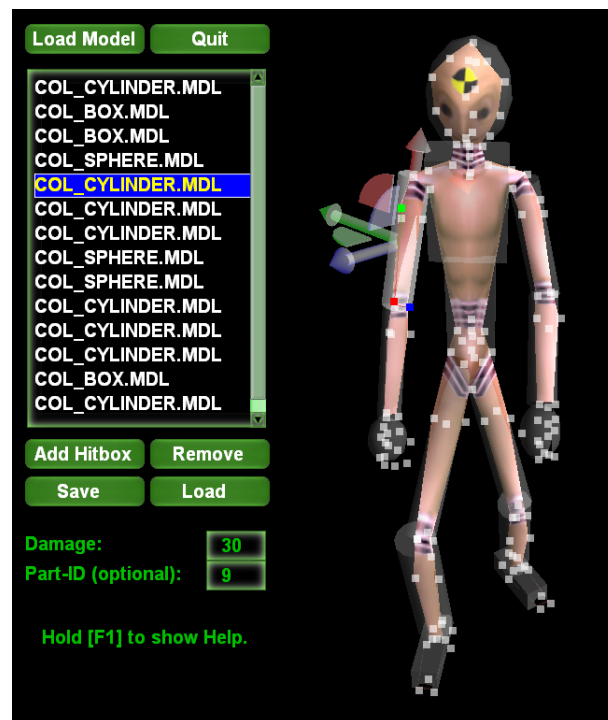


Abbildung 1: HED Interface

Hold [F1] to show HED's Help. When you add a new hitbox, select it, then click two vertices on your model to attach the hitbox to those. You will need to select a third vertex to overcome the **Gimbal Lock** that appears with Euler angles. You can detach vertices with [Delete].

You can move and rotate the hitbox with the gizmo. Hold [Ctrl] and click an arrow/ a corner of the gizmo to scale the hitbox in that direction.

The rest of the editor is pretty self-explanatory, just remember where to find the Help file ([F1]).

2.3 Using the code

Copy *hitbox.h* and *hitbox.c* into your project folder. Include *hitbox.h* (note: the file that ends with „h“) in your game. You will probably need to edit some values in this file, it is well-commented, so don't be afraid!

The available functions are described in this header-file, too. Every time you load a level, you will need to call `hitbox_init()` afterwards.

Here's an example how to set up an enemy action:

```
action act_enemy() {
    my.health = 100;
    my.group = type_enemy;
    wait(1);
    c_setminmax(my);
    hitbox_collisionbox_create(my); // call after (!) c_setminmax
    // now you can change min/max-xyz as you wish
    while(my.health > 0) {
        ...
        c_ignore(type_collisionbox, type_hitbox, 0); // this call is important !
        c_move(me, ...);
        hitbox_collisionbox_update(my);
        wait(1);
    }
    hitbox_collisionbox_delete(my);
    ptr_remove(me);
}
```

As you see, you have to handle the creation, placement and deletion of the collision cube mentioned in **2.3**. It's important that you set the entity's group to `type_enemy`.

Maybe you do not want to create a hitbox configuration for every model file, that's okay. Just set the group to `type_simple_enemy` and do **not** use the collisionbox-commands:

```
action act_simple_enemy() {
    my.health = 100;
    my.group = type_simple_enemy;
    wait(1);
    c_setminmax(my);
    while(my.health > 0) {
        ...
        c_move(me, ...);
        wait(1);
    }
    ptr_remove(me);
}
```

The code uses 4 skills (including health-skill), you can redefine them (and the group numbers) in „hitbox.h“ if they are already in use by your code:

```
#define health skill1
#define hitbox_colbox skill98
#define hitbox_hit_anim skill99 // can be used to play different hit animation
#define _type skill100
```

The last thing is a simple shooting script:

Use the third argument to realize different weapon powers, the damage will be multiplied by this factor. You can use hitbox_hit to retrieve the ID if some hitbox was hit, it's zero otherwise.

```
vec_set(shoot_target, vector(3500,0,0));
vec_rotate(shoot_target, camera.pan);
vec_add(shoot_target, camera.x);
hitbox_shoot(camera.x, shoot_target, 1.0); // 1.0 = damage factor
```

```
void main() {
    level_load(...);
    hitbox_init();
    on_f1 = hitbox_toggle_show;
}
```

If you do not know where to start, have a look at the example that comes together with the rest.

Feel free to ask questions on the gamestudio forum.

2.4 Credits

You can use it in any way you like to do so, commercially or freeware.

If you are a kind person, please give credit to:

Superku aka Hendrik Felix Pohl
www.superku.de

If you are a mean person, don't give any credit.

Special thanks to Lukas for his LBGUI-solution and to Testdummy for his test-dummy.