Atelier sur les effets spéciaux

rev:1.31

Avant propos

Bienvenue à mon atelier sur les effets spéciaux!

Avec cet atelier vous allez apprendre à créer de nombreux effets et lorsque vous le quitterez vous aurez la connaissance des effets WDL et des nouvelles particularités du moteur A5.1 pour créer vos propres effets étonnants!

Cet atelier est fait pour le moteur A5. Une version A4 peut être trouvée sur la section de téléchargement d'Interactive.
(http://acknexinteractive.fws1.com/download.htm)

En écrivant cet atelier je vous envoie un petit mot d'encouragement pour que vous alliez le plus loin possible et que vous puissiez créer de grands nouveaux effets sans ennuis.

Salutations,
Nizzy

Table des matières

Utilisation de time

Avant de commencer,je voudrais expliquer l'utilisation de ***time***. Un codage sans cette est entièrement dépendant de votre taux de rafraîchissement. Comme:

```
while(1)
{
        my.alpha -= 5;
        wait(1);
}
```

Pour contrer ceci, nous allons utiliser la variable système ***time*** Elle représente le temps entre le changement de frame. La plupart du temps > 1 sauf sur les machines lentes. Donc, en multipliant le nombre par ***time*** vous aurez des effets plus lisses indépendants du taux d'affichage.

```
while(1)
{
        my.alpha -= 5 * time;
        wait(1);
}
```

Le rougeoiment alpha

Maintenant, nous allons inclure une gemme rayonnante qui nous redonnera des points de But we don't want it to emit light, instead we will make it ***transparent*** and make the "clearness" increase and decrease to get a glowing effect. To do this we will be using the ***transparent*** flag and the ***alpha*** value. The ***transparent*** flag sets the entity to be semi-opaque and the ***alpha*** value tell you just how opaque it should be, 100 being opaque and 0 being invisible. Lets start out with this:

```
action gem
{
        if (my.skill5 == 0) {my.skill = 25; }
        my.event = medi_pickup;
        item_pickup();
}
```

This is the same code used for the medi packs. Now we can add our "glowing" code like so.

```
action gem
{
        if (my.skill5 == 0) {my.skill5 = 25; } // default +25 health
        if (my.skill3 == 0) {my.skill3 = 75; } // default 75 max opaque
        if (my.skill2 == 0) {my.skill2 = 25; } // default 25 max transparency
        my.event = medi_pickup;
        my.transparent;       make it sensitive to the alpha value
        item_pickup();
        while (1)
        {
                // We want it to keep glowing, not just once.
                while (my.alpha <= my.skill3)
                {
                        my.alpha += time;    // make it more opaque
                        wait(1);
                }
                while (my.alpha >= my.skill2)
                {
                        my.alpha -= time;    // make it more transparent
                        wait(1);
                }
        }
}
```

Now doesn't that look nice! Lets try making a flare effect now.

Well, lets go over the way a flare works first. A flare is attached to lights, and becomes larger the further you are away from it and smaller the closer you are. For this flare we will use a simple circular flare using the ***flare*** transparency flag.

```
var flare_var = 0;

action light_flare
{
        my.flare = on;
        my.red = your.red;
        my.green = your.green;
        my.blue = your.blue;
        my.light = on;
        my.lightrange = 0;
        my.passable = on;
        my.facing = on;
        my.near = on;
        while (1)
        {
                flare_var = vec_dist(player.x,my.x);
                IF (flare_var > 2000)
                {
                        flare_var = 2000;
                }
                my.scale_x = (flare_var / 900 + 0.5);
                my.scale_y = (flare_var / 900 + 0.5);
                if (flare_var < 30)
                {
                        my.invisible =on;
                }
                else
                {
                        my.invisible = off;
                }
                wait(1);
        }
}
```

Okay, this looks great! It is clipped if it is partially blocked by another surface. Lets add a ***my.near*** to it now. Here is the final code:

```
var flare_var = 0;

action light_flare
{
        my.flare = on;
        my.red = your.red;
        my.green = your.green;
        my.blue = your.blue;
        my.light = on;
        my.lightrange = 0;
        my.passable = on;
        my.oriented = on;  // will now be facing according to its pan, tilt and roll
to get a better facing effect using me_on_player()
        my.near = on;  // reduce clipping
        while (1)
        {
                me_on_player();
                flare_var = vec_dist(player.x, my.x);
                flare_var = min(flare_var,2000);                //optimized, don't
        allow < 2000
                my.scale_x = (flare_var / 900 + 0.5);
                my.scale_y = my.scale_x;  // small optimization (scale_x = scale_y
always)
                if (flare_var < 30)
                {
                        my.invisible = on;
                }
                else
                {
                        my.invisible = off;
                }
                wait(1);
        }
}

function me_on_player() {
        // keep me facing the player
        vec_set(temp,player.x);
        vec_sub(temp,my.x);
        vec_to_angle(my.pan,temp);
}
```

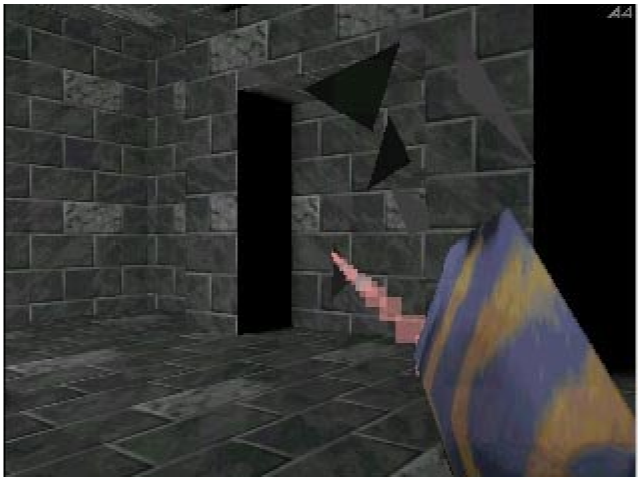
Now it is just the way we want it for our level! Lets move on to some glass effects!
To close this section of the workshop we will create a breaking window using ***alpha*** transparency.
Our breaking window will act like real glass; it shouldn't break on the first hit all the time. To do this we will use the ***_health*** skill used for enemies in war.wdl. Lets look at our ***action*** first:

```
action window
{
    my.transparent = on;
    my._health = 50;
    my.event = glass_hit;
    my.enable_shoot = on;    //sensible to bullets
    my.enable_scan = on;       //sensible to explosions
    my.push = 10;                            //pass through anything with <10 push
    while (1)
    {
        my.alpha = 100 - my._health;    //It will get more opaque the more damage, and
look cloudier (under stress).
    }
}
```

This will set the health of the window to 50 and it will have 50% transparency to start with and get more opaque the more damage it takes. Now lets look at the event function and particle function which will be placed above the action:

```
function glass_hit();
{
        if (event_type == event_scan){ if (indicator != _explode) && (indicator !=
_gunfire) {end;}}  // exit if not a gunshot of explosion
        my._health -= damage;       //subtract damage caused be explosion or bullet.
        if (my._health <= 0)
        {
                glass_gib(10);
                remove(me);
        }
}

function glass_gib(numberOfParts)
{
        temp = 0;
        while(temp < numberOfParts)
        {
                create(<glass.mdl>, my.pos, _gib_action); // use gib_action in war.wdl
        to animate the glass gibs,  makes them bounce and spin
        temp += 1;
    }
}
```

Now the window will shatter into 10 glass gibs that fade out after a while when destroyed.



Here is the final code:

```
function glass_hit();
{
        if (event_type == event_scan){ if (indicator != _explode) && (indicator !=
_gunfire) {end;}}  // exit if not a gunshot of explosion
        my._health -= damage;       //subtract damage caused be explosion or bullet.
        if (my._health <= 0)
        {
                glass_gib(10);
                remove(me);
        }
}

function glass_gib(numberOfParts)
{
        temp = 0;
        while(temp < numberOfParts)
        {
                create(<glass.mdl>, my.pos, _gib_action); // use gib_action in war.wdl
        to animate the glass gibs,  makes them bounce and spin
        temp += 1;
    }
}

action window
{
    my.transparent = on;
    my._health = 50;
    my.event = glass_hit;
    my.enable_shoot = on;    //sensible to bullets
    my.enable_scan = on;       //sensible to explosions
    my.push = 10;                            //pass through anything with <10 push
    while (1)
    {
        my.alpha = 100 - my._health;    //It will get more opaque the more damage, and
look cloudier (under stress).
    }
}
```

Lights

The first thing in lighting is figuring what you want the light to do. In this workshop, we will do fire lighting and a broken light that flickers. We will start with the fire. Okay, we will have the lighting use its ***skill4*** for the light distance and add a random ***skill5*** for the intensity. The light color will be set by ***skill1-3*** and it will execute forever so we will be using the ***while*** loop. Lets start out simple:

```
action fire
{
        my.red = my.skill1;
        my.green = my.skill2;
        my.blue = my.skill3;

        while (1)
        {
                my.lightrange = my.skill4 + random(skill5);
                wait(1);
        }
}
```

The ***red, green, blue*** are set to ***skill1-3*** like we planned on. And the ***lightrange*** sets the distance of the light emitting from the entity, so we set it to a minimum of ***skill4*** and add up to ***skill5*** to create a random flicker of light distance. Then we ***wait*** one frame and repeat.

Now, I've created a new level with only a large hollow block textured with #default and added a torch.mdl (included in this zip) and attached the action fire to it and it does NOTHING! Why is this?! This was because I didn't enter any values in for ***skill1-5***! To prevent this, we should add some default values, but how? Lets try this.

```
action fire
{
        if (my.skill1 == 0) { my.skill1 = 155; }
        if (my.skill2 == 0) { my.skill2 = 55; }
        if (my.skill3 == 0) { my.skill3 = 0; } //orange light
        if (my.skill4 == 0) { my.skill4 = 300; }
        if (my.skill5 == 0) { my.skill5 = 100; }

        my.red = my.skill1;
        my.green = my.skill2;
        my.blue = my.skill3;

        while (1)
        {
                my.lightrange = my.skill4 + random(skill5);
                wait(1);
        }
}
```

Now, before doing anything it check if any of the skills are set to 0, if they are, it will set them to the default value! Now after simply running the level again, I get a fire-like flickering red light just like real life! But it lacks one thing. Variation. Lets try changing the color a little each time, darker the shorter the ***lightrange***, the brighter the longer the ***lightrange***.

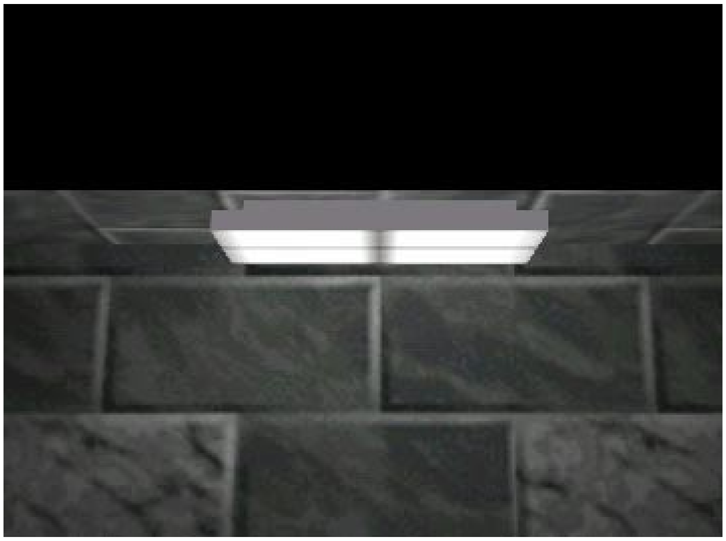
```
action fire
{
        if (my.skill1 == 0) { my.skill1 = 225; }
        if (my.skill2 == 0) { my.skill2 = 140; }
        if (my.skill3 == 0) { my.skill3 = 55; } //red light
        if (my.skill4 == 0) { my.skill4 = 70; }
        if (my.skill5 == 0) { my.skill5 = 100; }
        my.red = my.skill1;
        my.green = my.skill2;
        my.blue = my.skill3;

        while (1)
        {
                my.lightrange = my.skill4 + random(my.skill5);
                my.red += random(my.skill6)-(my.skill6 / 2);
                my.green += random(my.skill7)-(my.skill7 / 2);
                my.blue += random(my.skill8)-(my.skill8 / 2);
                wait(1);
        }
}
```

THERE! It looks *perfect*. Now it adds or subtracts up to half of ***skill6-8*** from the RGB value.

Now for the busted flickering light. We will have it off for a random amount of time and then flicker on for a smaller random time.

```
var light_var;

action light_flicker
{
        if (my.skill1 == 0) { my.skill1 = 155; }
        if (my.skill2 == 0) { my.skill2 = 55; }
        if (my.skill3 == 0) { my.skill3 = 55; } //red light
        while (1)
        {
                // light on
                my.red = my.skill1;
                my.blue = my.skill2;
                my.green = my.skill3;
                my.ambient = 100;
                my.lightrange = 150;
                light_var = random(8);
                wait(light_var);

                // light off
                my.ambient = 0;
                my.lightrange = 0;
                light_var = random(16);
                wait(light_var);

                wait(1);
        }
        }
```

light_var is set to a random value used for the ***wait*** instruction up to16 for the off time and up to 8 for the on time. I think that looks pretty good!



Particles

Before starting this tutorial section I would like to thank Gaehh. His tutorial helped me learn the things that make these particles look so great, I didn't know about the *bright* flag or about the my_pos_x etc. I thought it only had my_speed_x etc. the my_pos_x etc. will be used in future effects and tutorials. ***THANKS GAEHH***.

Okay, lets start simple: bleeding.
Our particle function will be almost the same as particle scatter, and will use many great new features of the engines particle system. In this tutorial I will also explain everything I can about each flag to help you create better effects!

***my.bright*-**
The particles will be brighter in areas that have overlapping particles to create a glow or fire effect.

***my.transparent*-**
*The particles will assume alpha transparency and use the **my.alpha** value to set the % of transparency, if null, the particles will be 50% transparent.*

***my.lifespan*-**
*The particles will live until **my.lifespan** = 0, **lifespan** will decrease over the particles life automatically. Set to 0 to kill the particle..*

***my.flare*-**
*The particles will be transparent like with **my_transparent** but the darker areas will be more transparent while the lighter areas of the particle more opaque.*

***my.alpha*-**
Setting for % of transparency of particles, 100% is opaque and 0% is clear.

***my.function*-**
*Set to null to continue the particle life until **my.lifespan** = 0;.*

All right, lets start the scripting!
Lets start with the ***action***:

```
var firevar;

action fire1
{
        if (my.skill1 == 0) { my.skill1 = 8; }
        while (1)
        {
                firevar = my.skill1;                            // sets the # of
        particles to be released.
                effect(particle_fire,firevar * time,my.x,nullvector);    // emit the
        particles from my.x using the new effect command (A5)
                wait(1);
        }
}
```

This *action* is just like any other particle *action* except for the fact it implements *skills* that interact with the *effect* part. To do this, you have to copy the *skill* to a *var*. and use the var in the *effect* command.

Now we will create the function:

```
        bmap fire_map1 = <light.bmp>;
        var firevar2,1;

        function vec_fire(&vec)
        {
                vec[0] = random(30) - 15;
                vec[1] = random(30) - 15;
                vec[2] = random(10);
        }

        function particle_fire()
        {
                vec_fire(temp);
                vec_set(my.vel_x,temp);
                my.lifespan = 15;
                my.flare = on;  //flare transparency
                my.bright = on; //glowing effect
                my.move = on;   //move according to vel_x, vel_y, and vel_z
                my.streak = on; //new streak effect
                my.bmap = fire_map1;
                my.alpha = 20;
                my.size = 500;
                my.function = null;     //no mid-life mods yet!
        }
```

Wow, doesn't that look BAD? However, this is only the framework for a very advanced particle script with a great outlook on real fire "physics" Lets make it less "out". How about a nice curve at the bottom making it go straight up eventually.

```
        function fire_func()
        {
                my.vel_x == my.vel_x * 0.5;
                my.vel_y == my.vel_y * 0.5;
        }
//also change the "my.function = null;" in particle_fire to "my.function = fire_func;"
```

Now we are cutting the x and y speed in half each frame, this will result in never reaching 0 speed for either, but it won't change the performance (it will get to 1 then .5,.25,.125,.0625 and so on).

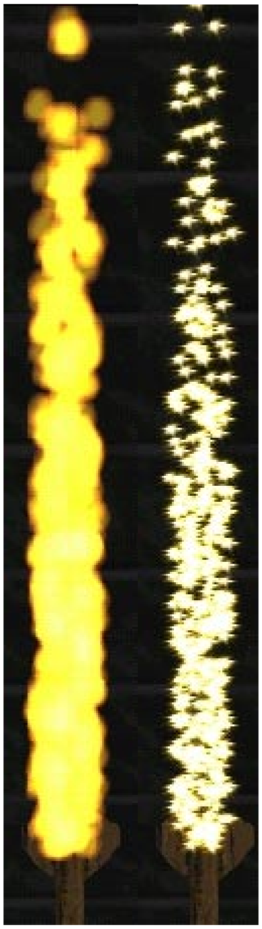
Now the top is even, we want it pointed. This is where the strange outlook on fire "physics" comes in. Rather than making all sorts of formulas that cause the tip to be pointed, we will use a heat idea. If a particle is too far from the center of the fire it will die because it is colder. It will also move slower because it has less heat to rise. SO, that means we will only have to modify *my.vel_z*, as you will see. The speed will be set lower, the further the x and y speeds are from 0 (the center). Then the particle will die before it reaches the tip causing the center particles to be highest forming a tip! This is easier than you think!

```
        function vec_fire(&vec)
        {
                vec[0] = random(30) - 15;
                vec[1] = random(30) - 15;
                vec[2] = random(10) - (abs(my.vel_x) + abs(my.vel_y)) * 0.3 + random(5);
        }
```

Lastly we want to test what graphics look best.

```
function fire_func()
{
        my.vel_x = my.vel_x * 0.5;
        my.vel_y = my.vel_y * 0.5;
        my.alpha -= 2 * time;          //fade out gradually
        if(my.alpha < 0) { my.lifespan = 0; }  // kill when alpha < 0
        if (firevar2 == 1) { my.bmap = firemap1; }
        if (firevar2 == 2) { my.bmap = firemap2; }
        if (firevar2 == 3) { my.bmap = firemap3; }
        if (firevar2 == 4) { my.bmap = firemap4; }
}

function firechange()
{
        firevar2 += 1;
        if (firevar2 == 5) { firevar2 -= 4; }
        wait(1);
}

on_t firechange;
```

Now press T to change the graphics!



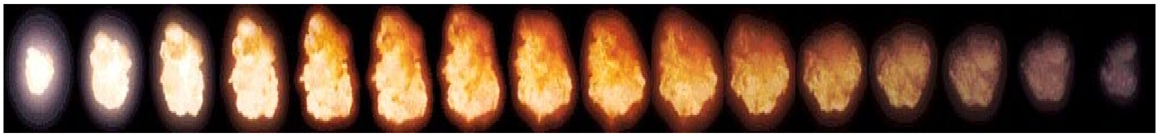
Lastly, we will make a particle fountain like in the Adeptus demo, starting with the function first:

```
bmap fountain_map = <sparkle.bmp>;

function vec_fountain(&vec)
{
        vec[0] = random(6) - 3;
        vec[1] = random(6) - 3;
        vec[2] = random(10) + 20;
}

function particle_alpha_fade()
{
        my.alpha -= time * 2;  // stay with framerate
        if(my.alpha < 0) { my.lifespan = 0;}
}

function particle_fountain()
{
        vec_fountain(temp);
        vec_set(my.vel_x,temp); //set the velocity
        my.lifespan = 50;
        my.flare = on;  //flare transparency
        my.bright = on; //glowing effect
        my.move = on;   //move according to vel_x, vel_y, and vel_z
        my.streak = on; //new streak effect
        my.bmap = fountain_map;
        my.gravity = 3;
        my.alpha = 50;
        my.size = 10;
        my.function = particle_alpha_fade;     //fade out near end of life
}
```

This is almost the same function (only adapted to A5 particles!!), but look at what's changed. I removed all the old SET instructions and replaced them with simple variable definitions and the alpha-fade function. Also, the particles will fade out, rather than just disappearing. Now we will see how it all works. The ***my.vel_x*** and ***my.vel_y*** make it go a maximum of 10 in either direction. The ***my.vel_z*** makes it move up at least 20, but less than 30. ***my.size*** defines the size of the particles. ***my.alpha*** is set to opaque (100) and fades out as its life goes on. ***my.flare*** makes the particle less opaque in darker areas. Lastly, ***my.lifespan = 50*** kills the particle if it is older than 50.

Explosions

Explosions are common and everyone's look different, but I'll give you a start on it. Our explosions will be mostly the animation sprite, but we'll add some other effects. We will use the explosion sprite from MISSION2 as well.



Now lets start with the sprite animation:

```
action explode1
{
	my.facing = on; // face the camera
	my.near = on;
	my.flare = on;
	my.passable = on; // don't push the player through walls
	my.frame = 1;
	my._dieframes = 16;
	wait(1);

	play_entsound(my,explo_wham,1000);
	my.red = 210;
	my.green = 100;
	my.blue = 100;
	my.ambient = 100;
	my.lightrange = 110;

	// use the new sprite animation
	while(my.frame < my._dieframes)
	{
		wait(1);
		my.lightrange += 15;
		my.red += 20 * time; // fade to red  // stay with framerate
		my.blue -= 20 * time;  // stay with framerate
		my.frame += time;  // stay with framerate
	}
	wait(1);
	ent_remove(my);
}
```

Lets try some particles:

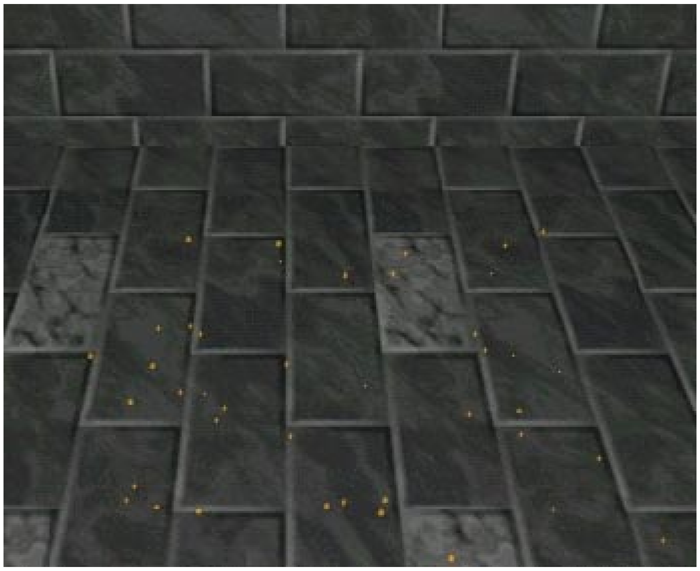
```
bmap explo_map = <expl.bmp>;

function vec_explode(&vec)
{
	vec[0] = random(6) - 3;
	vec[1] = random(6) - 3;
	vec[2] = random(10) + 20;
}


function explosion_func()
{
	if(my.lifespan > 30) { my.alpha += 3 * time; }  // stay with framerate
	if(my.lifespan <= 30) { my.alpha -= time; }
}
```

```
function particle_explo();
{
	vec_explode(temp);
	vec_set(my.vec_x,temp);
	my.flare = on;
	my.bright = on;
	my.move = on;
	my.streak = on;
	my.bmap = explo_map;
	my.gravity = 6;
	my.alpha = 30;
	my.size = random(50) + 50;		// 50-100
	my.lifespan = 50;
	my.function = explosion_func;
}
```

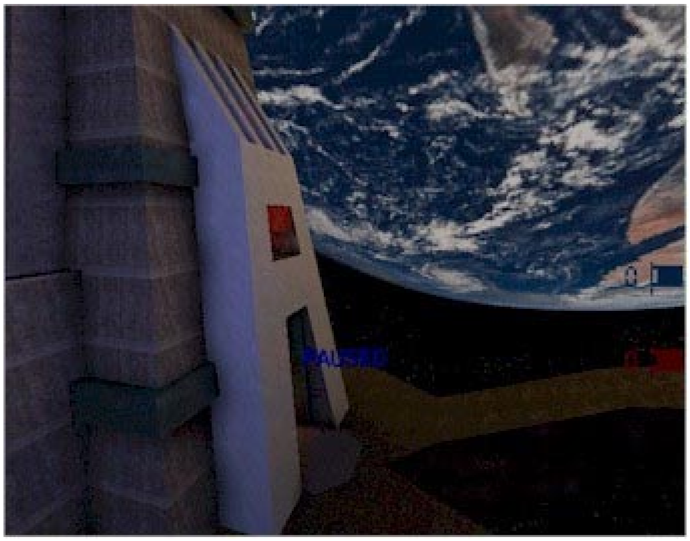
And add an ***effect*** to explode1.

```
action explode1
{
	my.facing = on; // face the camera
	my.near = on;
	my.flare = on;
	my.passable = on; // don't push the player through walls
	my.frame = 1;
	wait(1);

	play_entsound(my,explo_wham,1000);
	effect(particle_explo,50 * time,my.x,nullskill);
	my.red = 210;
	my.green = 100;
	my.blue = 100;
	my.ambient = 100;
	my.lightrange = 110;

	// use the new sprite animation
	while(my.frame < my._dieframes)
	{
		wait(1);
		my.lightrange += 15;
		my.red += 20 * time; // fade to red  // stay with framerate
		my.blue -= 20 * time;  // stay with framerate
		my.frame += time;  // stay with framerate
	}
	wait(1);
	ent_remove(my);
}
```

Now test it out!



Sky Spheres



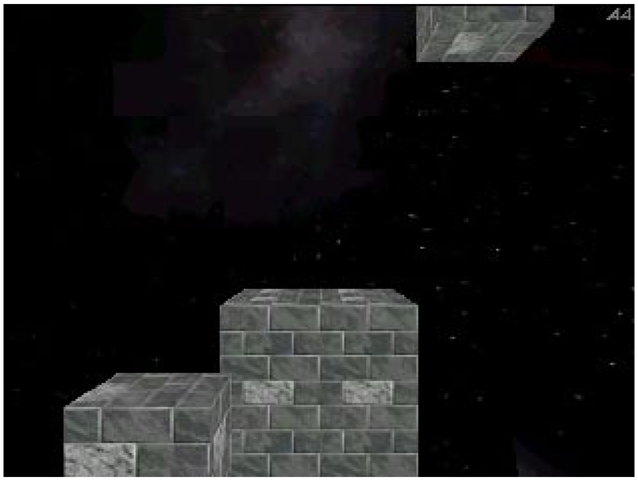
This introduces you the use of sky***spheres*** similar to the ones used in the *Spaceflight workshop*, but for shooters, arenas and the like. An example is Unreal Tournaments space arenas like ***Facing Worlds***. This allows for really cool space station levels where the sky textures look bad. We will start with the graphical part.

1. Open MED.
2. Create a large sphere, you can make a better one in milkshape and import it, it looks better in high-poly, MEDs **subdivide** doesn't do the trick.
3. Now texture it with a star texture or whatever your sky it supposed to look like.
4. Make it large enough to enclose your entire level (that's really big)
5. Now, you will be *inside* it, so the texture has to be on the inside! Switch to triangle mode and select all triangles.
6. Click the **flip normals** button and save it as **stars.mdl** or something like that.

Now we have an easy script to make:

```
action starsphere
{
        while(1)
        {
                my.pan +=  0.5 * time;
                my.tilt +=  0.5 * time;
                wait(1);
        }
}
```

If you want you can leave it without an action, the action just makes it look like your station or platform is spinning slowly in space.

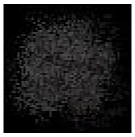

AA

Smoke

Wether its a smoking rocket launcher or a blazing fire, you are going to use smoke in your game, and I'm sure the templates paritcle_smoke isn't exactly what you call "Commercial Quality". So, lets make our own, but first, lets take a look at the template one.

```
function vec_smoke(&vec)
{
        vec[0] = random(1) - 0.5;
        vec[1] = random(1) - 0.5;
        vec[2] = random(1) + 1;
}

function smoke_func()
{
        if(my.size < 800) { my.size += time; }
        my.vel_x = my.vel_x / 2;
        my.vel_y = my.vel_y / 2;
}

function particle_smoke()
{
vec_smoke(temp);
vec_set(my.vel_x,temp);
my.size = random(5)+5; // 5 to 10 in size
my.bmap = smoke_map2;
my.flare = on;
my.lifespan = 100;
my.alpha = 50;
my.move = on;
my.function = smoke_func;
}
```

Okay, this doesn't look so bad after all. It just needs a little fixing. First, the smoke sprite looks like crap. Lets try the one from MISSION2.



This looks a little dark, but it looks light enough when the black is clipped and ***flare*** transparency is on. Now our smoke almost looks real, see how much the graphics can change a whole effect? But they just float up and BANG they disappear, lets try fading them out.

```
function smoke_func()
{
if(my.size < 800) { my.size += time * 0.1; }
my.alpha -= 2.1 * time;
if(my.alpha < 0) { my.lifespan = 0; }
my.vel_x = my.vel_x / 1.2;
my.vel_y = my.vel_y / 1.2;
}
```

Now we are subtracting 2.1 alpha each frame and making it fade out as it expands to a bigger size. Isn't that amazing? Now try coding your own from scratch! If you need help look at the particle section.

Gibbing!!

Oh yay! The wonderful GIBBING SCRIPT!!! This discusses the basic gib modifications I made to the one that is in war.wdl.

If you think it looks great, wait 'till you hear this I probably only modified a few lines of code to do it, but I had to get them all perfect so it looks exactly like it does.

The first thing was making some particles for blood.

```
bmap blood_map = <blood.bmp>;

function fade_func()
{
        my.alpha -= my.skill_x * time;
        if(my.alpha < 0) { my.lifespan = 0; }
}

function vec_blood(&vec)
{
        vec[0] = random(20)-10;
        vec[1] = random(20)-10;
        vec[2] = random(20)+10;
}

function particle_blood
{
        vec_blood(temp);
        vec_set(my.vel_x,temp);
        my.move = on;
        my.streak = on;
        my.flare = on;
        my.size = 10;
        my.alpha = 65;
        my.bmap = blood_map;
        my.lifespan = 35;
        my.skill_x = 2;
        my.gravity = 6;
        my.function = fade_func;
}
```

Hmmm... look familiar? Perhaps the scatter_speed would give you a clue! IT'S THE PARTICLE_SCATTER CODE! I just changed a few minor things.

my.bmap is now blood_map.

my.flare is enabled now.

Now I need use effect to release the particles from the body_gib_action.

```
function _body_gib_action()
{
        // scall the bits down by the actor_scale amount
        if (gibnow >= gibmax) { ent_remove(my); end; }
        gibnow += 1;
        vec_scale(my.SCALE_x,actor_scale);
        my.enable_block = on;

        // Init gib bit
        my._speed_x = 15 * (RANDOM(10) - 5); // -125 -> +125
        my._speed_y = 15 * (RANDOM(10) - 5); // -125 -> +125
        my._speed_z = RANDOM(35) + 15; // 15 -> 50
        my._aspeed_pan = RANDOM(35) + 5; // 35 -> 70
        my._aspeed_tilt = RANDOM(35) + 5; // 35 -> 70
```

```
    my._aspeed_roll = RANDOM(35) + 5; // 35 -> 70
    my._force = 0;
    my.roll = RANDOM(180); // start with a random orientation
    my.pan = RANDOM(180);
    my._force = -2;
    my.push = -1; // allow user/enemys to push thru

    abspeed[0] = 5 * my._speed_x;
    abspeed[1] = 5 * my._speed_y;
    abspeed[2] = my._speed_z * time * 5;
    move(me,nullskill,abspeed);        //move them away from the gib point so we
don't get floating blood splats

    // Animate gib-bit
    my.skill9 = 50;
    while(my.skill9 > -75)
    {
        abspeed[0] = my._speed_x * time;
        abspeed[1] = my._speed_y * time;
        abspeed[2] = my._speed_z * time;
        my.pan += my._aspeed_pan * time;
        my.tilt += my._aspeed_tilt * time;
        my.roll += my._aspeed_roll * time;
        vec_scale(absdist,movement_scale); // scale absolute distance by
    movement_scale
        move(me,nullskill,abspeed);
        effect(particle_blood,bloodlevel * time,my.x,nullskill);
        if(bounce.z)
        {
            create(<blooda.bmp>,my.x,blood_spat);
            my._speed_z = -(my._speed_z/2);
            if(my._speed_z < 0.25)
            {
                my._speed_x = 0;
                my._speed_y = 0;
                my._speed_z = 0;
                my._aspeed_pan = 0;
                my._aspeed_tilt = 0;
                my._aspeed_roll = 0;
            }
            gibvar = bloodlevel * splatlevel;
            effect(particle_blood,gibvar * time,my.x,nullskill);
            // my.skill9 -= 100;
        }
        my._speed_z -= 2;
        my.skill9 -= 1;
        wait(1);
        }
        my.passable = on;
        // Fade out
        my.transparent = on;
        my.alpha = 100;
        while(1)
        {
            my.alpha -= 5*time;
            if(my.alpha <=0)
        {
        // remove
        gibnow -= 1;
        ent_remove(my);
        return;
        }
```

```
		wait(1);
		}
	}
```

I added an *effect* in the while statement so it would constantly be dripping blood as the gib flies through the air. Also an *effect* to emit more blood when it *bounces*. Also "create(<blooda.bmp>,my.x,blood_spat);" was added to leave splat decals on walls. using the blood_splat function.

```
	function blood_spat()
	{
		scan_sector.pan = 360;
		scan_sector.roll = 360;
		scan_sector.tilt = 360;
		my_angle.pan = my.pan;
		my_angle.tilt = my.tilt;
		scan(my.x,my_angle,scan_sector);

		if(bullet_hole_counter <= kmaxbullethole && you != player)
		{
			bullet_hole_counter += 1; // inc bullet hole counter
			my.transparent = on;
			my.passable = on;
			my.oriented = on;
			vec_to_angle(my.pan,normal); // rotate to target normal
			waitt(160); // time blood stays before vanishing, wait a fixed amount
		of time with waitt(160), rather than 160 frames with wait(160)
			while(my.alpha > 0) // fade out
			{
				my.alpha -= time;
				waitt(1);
			}
			bullet_hole_counter -= 1; // decrease bullet hole counter
			ent_remove(my);
		}
		else
		{
			ent_remove(my);
		}
	}
```

Using the *kmaxbullethole* var used for maximum bullets in the level i also included blood splats as bullet holes. It is very similar to the bullet hole function. Also, rather than using wait(160) I used waitt(160). I this way the splats last 160 ticks, a fixed amount of time; rather than 160 frames, a varying amount of time.

Advanced

Now that you have gone over the whole tutorial you may want more advanced things. This is where you get to do you own scripts until my next tutorial. I hope you learned a lot from this tutorial and are now able to make your own scripts. I'll see your contributions on the User Forum!

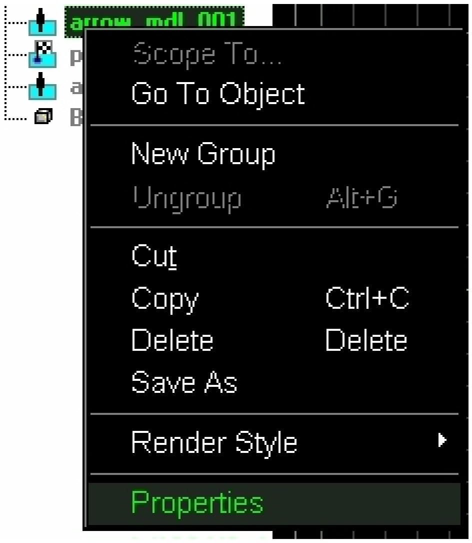
Farewell,
Nizzy

Appendix A

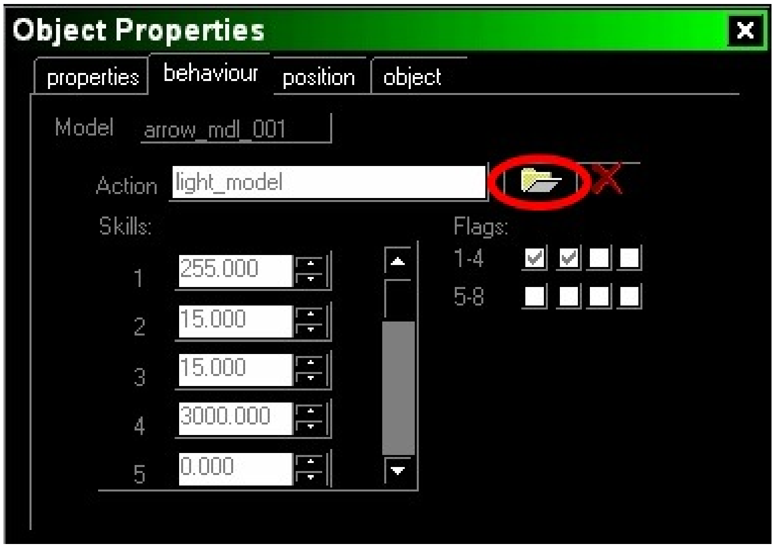
To use these effects, you simply *include* effects.wdl in your games main script, assign the corresponding action to the desired entity in WED, and set the skills and flags to further customize the effect.

Including effects.wdl
Create an new level. Now click File > Map Properties. Click on the new button next to the WDL script to create your new script. Now open the file in wordpad or any WDL editor. Add "include <effects.wdl>;" at the bottom of the include list. Now the effects will be in your action menu.

Assigning the action
Open WED and create a new level or open an existing one. Now add an entity and right click it in the object browser and select Properties.



This will open the properties window, go to the behavior tab and click on the open button next to the action entry line.



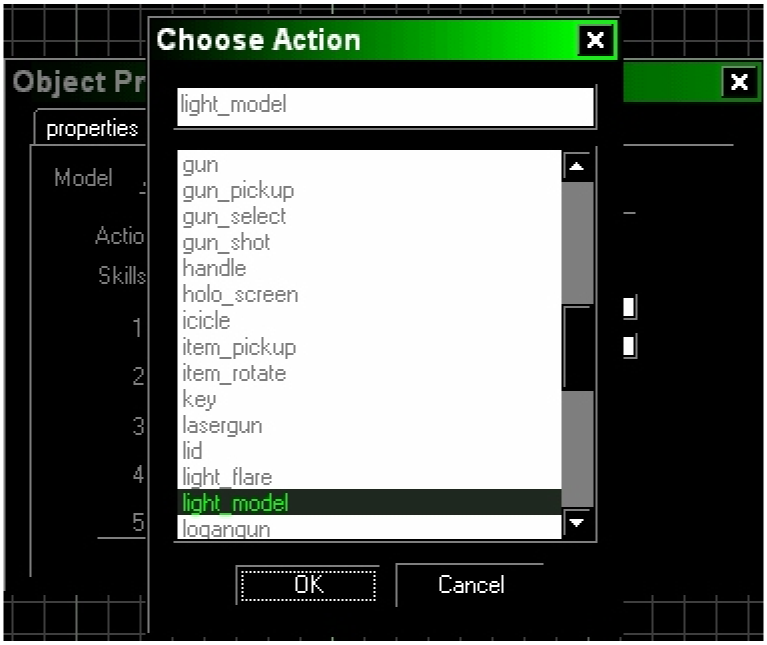
Model arrow_mdl_001

Action light_model

Skills:
1 255.000
2 15.000
3 15.000
4 3000.000
5 0.000

Flags:
1-4 ☑ ☑ ☐ ☐
5-8 ☐ ☐ ☐ ☐

This will display a list of all the actions *included* in your games script.

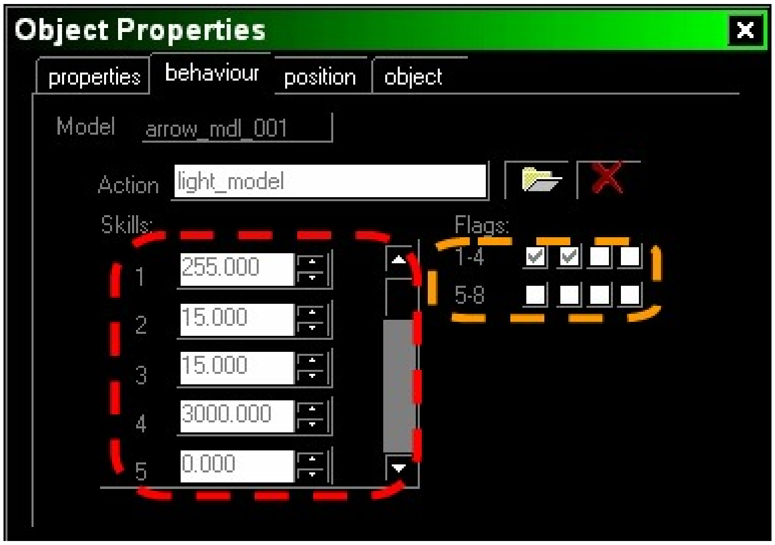


light_model

gun
gun_pickup
gun_select
gun_shot
handle
holo_screen
icicle
item_pickup
item_rotate
key
lasergun
lid
light_flare
light_model
logangun

Select whichever effect you like and click OK.

Now you can customize the effect by setting the *skills* and *flags*.



Model arrow_mdl_001

Action light_model

Skills:
1 255.000
2 15.000
3 15.000
4 3000.000
5 0.000

Flags:
1-4 ☑ ☑ ☐ ☐
5-8 ☐ ☐ ☐ ☐

Skill/Flag definitions & defaults

Note: The script may differ from the effects created in this workshop, but is made for a more general purpose. The effects made in this workshop were made customized so you can learn how to customize them to fit your game.

Light
Flag1 = flame-light
    Skill1-3 = red-green-blue light color (155,55,55)
    Skill4 = light distance (300)
    Skill5 = flame intensity (100)
Flag2 = busted-light
    Skill1-3 = red-green-blue light color (155,55,55)

Alpha_glow

Skill1 = min. alpha transparency (25)
Skill2 = max. alpha transparency (75)
Skill3 = glow speed (5)

Test_gib
No skills or flags, used for testing the gibs

Fire1
Skill1 = intensity (8)

Starsphere
No skills or flags, used for a skysphere model

Window
No skills or flags, attach to a window sprite and it will weaken and shatter when shot in the game.

Gem
Skill1 = alpha value (100)

Smoker
No skills or flags, used for a short pillar of smoke.

Appendix B

Crédits-

Game engine design- Conitec
WDL Script- Dan Niezgocki
Graphics- Dan Niezgocki
Graphics- Conitec
Particle help etc.- Gaehh

Special thanx- All those who gave ideas on the user forum, you are who made this possible!
Special thanx- Doug Poston, great support and help.
Very Special thanx- JCL, for your time, patience and everything else.