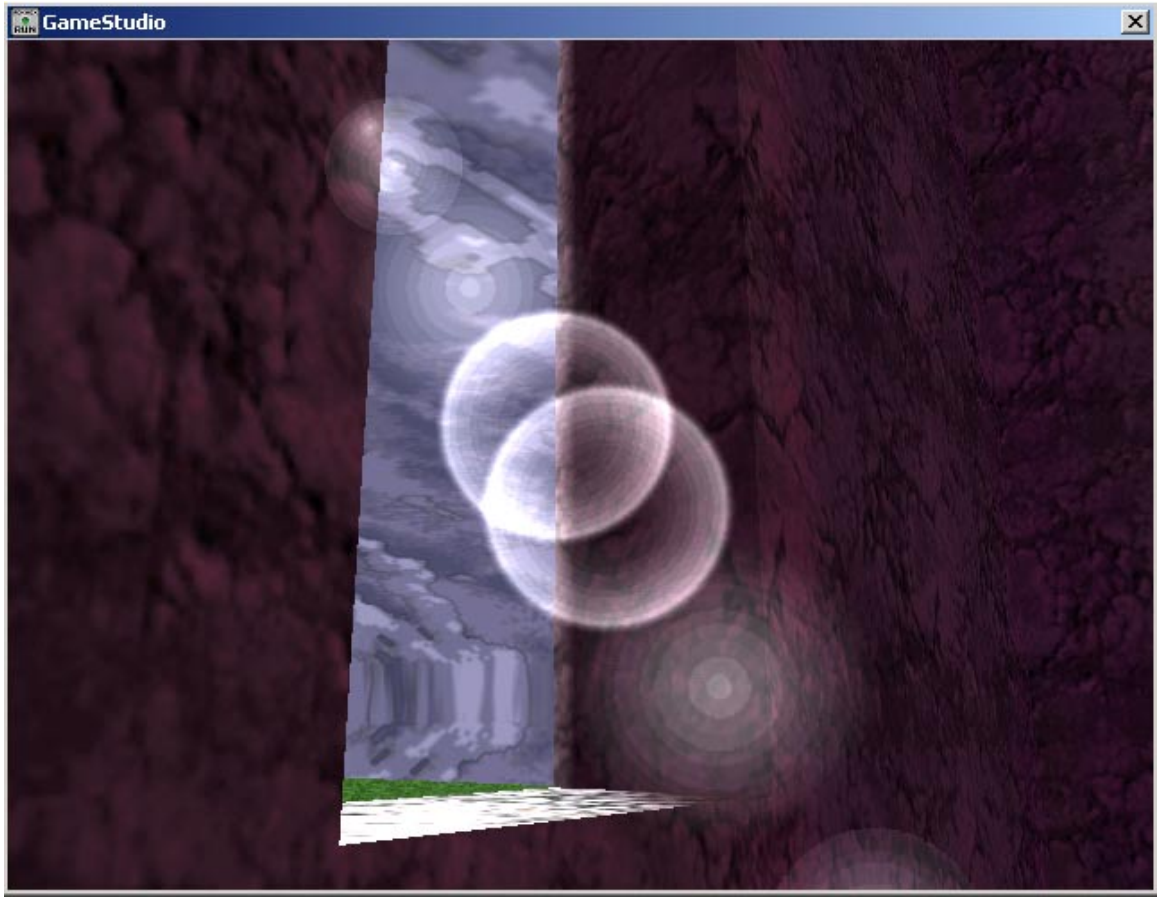
Lens Flare Microworkshop



Forward

Dear Reader,

Welcome to the first "Microworkshop". While the "normal" workshops cover large subject (i.e. the basics of RPGs or Flight Simulators) these workshops are designed to cover smaller subjects usually something you can add to an existing project. Our first subject will be "Lens Flares".

This workshop, like the normal workshops, is written for users with some previous 3DGameStudio experience. I assume that you have worked through the tutorials and understand at least the basics on how to use GameStudio and WDL.

This text is meant to complement the rest of the documentation that comes with 3DGameStudio, not replace it. If something in this workshop is unclear to you please read through the manuals that came with 3DGameStudio. I apologize in advance for any unclear wording, faulty code, errors, or omissions.

I hope you find these new microworkshops informative and enjoyable.

-Doug.

Get the latest version

Before you begin, it is very important to make sure you have the latest version of 3D GameStudio (engine, editors, and template scripts). I try to take advantage of the latest version so some of the commands/features I will use are only available with the latest updates.

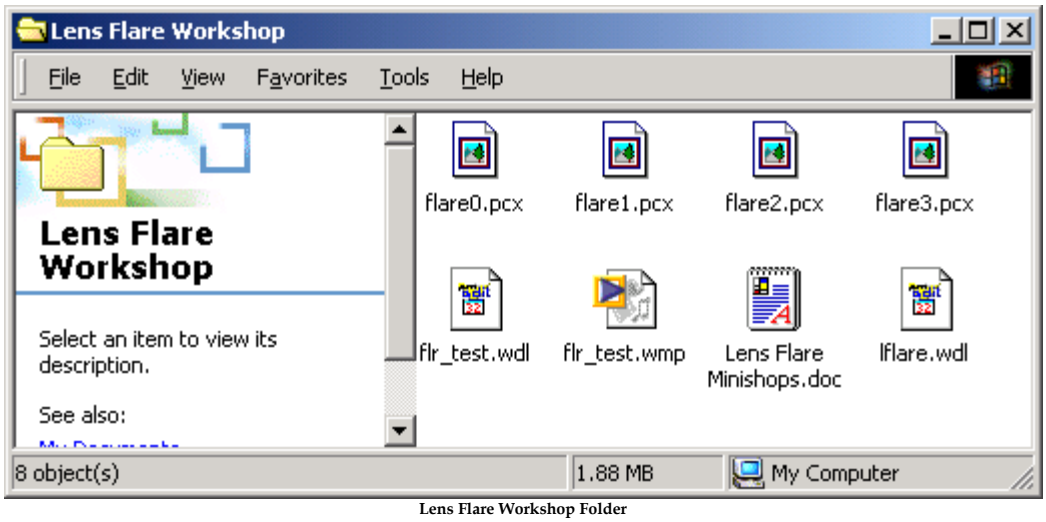
I will also try to take advantage of A5 features as well. When I use an A5 only feature I will note it and suggest a "work around" (if one is available). This workshop has been done with A5 Version **5.03**; if you try it with earlier engines, it's on your own risk.

Prepare your workspace

Create a folder called “Lens Flare Workshop” in your GStudio folder. This is the folder where all you game elements will be stored. Unzip the contents of the Len Flare workshop into this folder.

Your folder should now contain the following files:

Lens Flare Minishops.doc (this document)
flare0.pcx
flare1.pcx
flare2.pcx
flare3.pcx
flr_test.wmp
flr_test.wdl
lflare.wdl



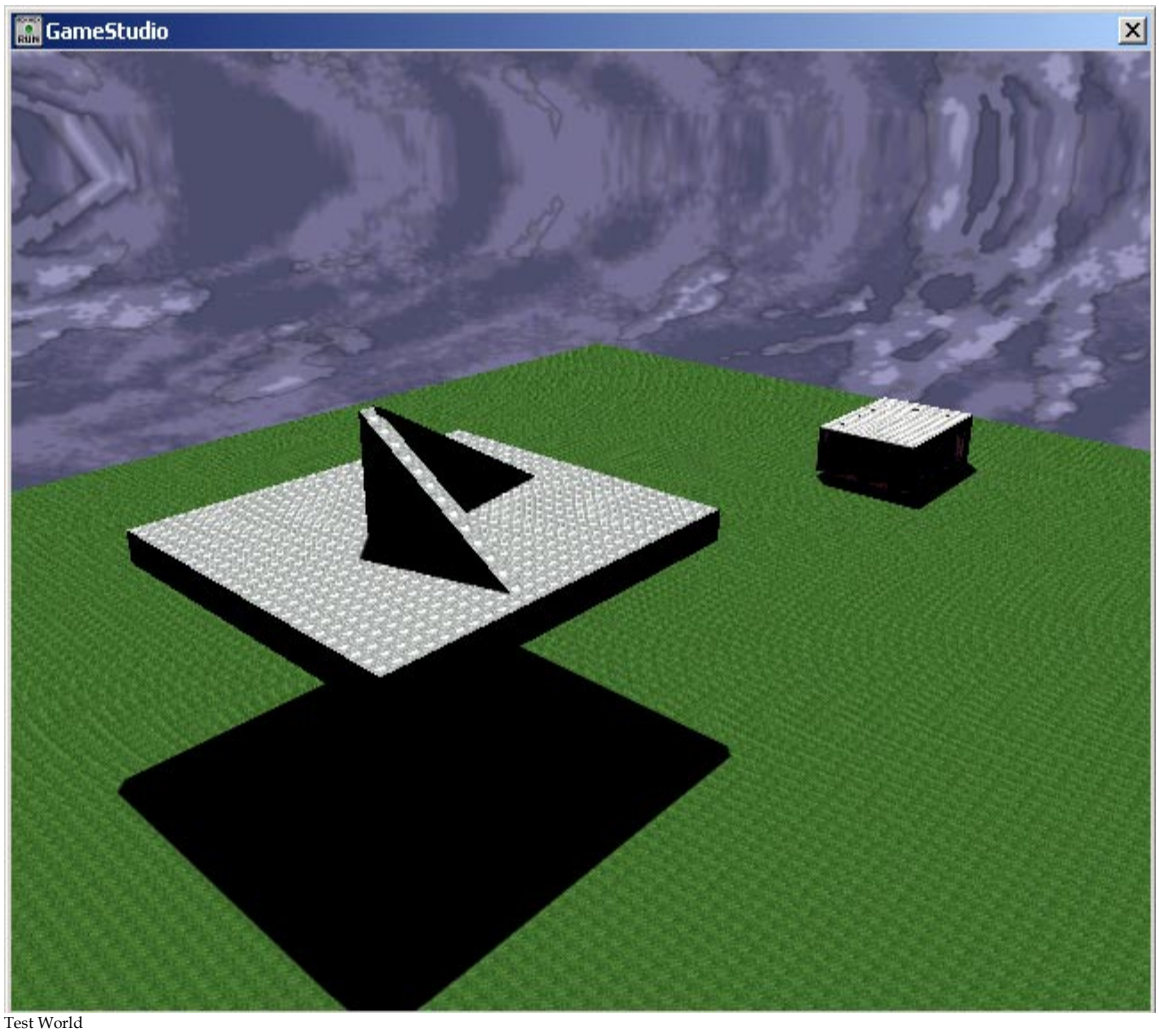
Lens Flare Workshop Folder

The "lflare.wdl" file contains the entire source for the lens flare code that we are coving in this workshop. If you want to see how this code works right now go ahead and build/run the flr_test level in WED. Move around the level and notice when the lens flare is visible and how it moves.

Creating the level

Because this is a mini workshop I am not going to make you spend your time following the steps to build a simple level. The code we are going to create should work with most 'outdoor' type levels. If you need a test level to get started (or for debugging) I've included two with this workshop (flr_tst.wmp and flr1test.wmp).

Note that you can set the sun position in the level (through Map Properties, for rendering sunlight and shadows), as well as in WDL (sun_angle). However those two positions are not necessarily the same! For sunlight, shadows and lens flares in your level to look right, take care to set your sun_angle.pan and sun_angle.tilt to the same sun azimut and declination angles you entered in WED.



Test World

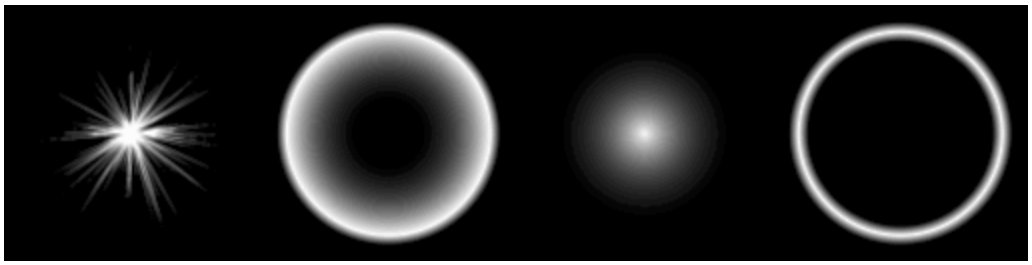
The important thing to note when using your own level is that there must be at least one area where you can 'trace' a line from the camera to the position value stored in `'flare_sun_pos'` (we will talk about this value later) or the lens flare will never show up.

What is "Lens Flare"?

The term lens flare has been used by different people to mean many different things. A good definition comes from the book "The Art and Science of Digital Compositing" (Ron Brinkmann, ISBN: 0121339602)

An artifact of a bright light shining directly into the lens assembly of a camera.

These can take the form of halos, spikes, or bright spots or halos, which seam to float in space. The intensity and positioning of these artifacts are dependent on the lens assembly and the position and intensity of the light in relationship to it.



Some Lens Flares

Unless you wear really think glasses or spend a lot of time with a camera or other optical devices, you don't normally see lens flare in you everyday life. They appear occasionally in film and television (the "X Files" is a great source for lens flares). Sometimes they appear by accident but often they are used for dramatic purposes (though more and more of these dramatic lens flares are added in postproduction by computers).

So why would you want to use lens flares in your game? Maybe your trying to get a movie like feel to your game, create the feeling like the player is behind a sheet of glass, or you just think the effect looks 'neat'. Used properly a lens flare can add to the look of your level or add to game play. Image the player selecting a position to shoot from that minimizes the lens flare on his/her sniper scope.

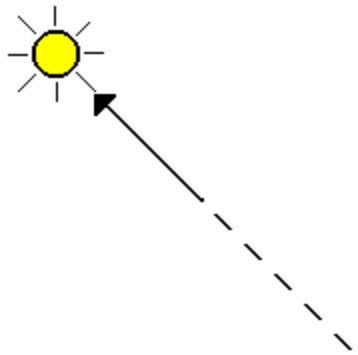
As neat as the effect can be beware of lens flare abuse! Now that lens flare effects are easy to add they are starting to look cheap and overused.

How Our Lens Flare Works

As you can see there are many different types of lens flare effects. The most dramatic type of lens flare, and the type most used in movies, television, and games are the multiple spot/halo effect. This is caused by shooting directly into a bright light source like the sun and is the result of reflections from the surfaces of the lenses and body of the camera.

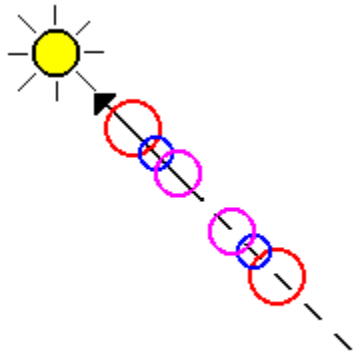
We could try to accurately model the lens and light to get the most "realistic effect" but this would take a great deal of time and we couldn't reproduce it in real time. The approach we are going to use is much simpler and behaves almost the same.

There are several different ways to "fake a flare" but the method I am going to use here is the "stick and sprite" approach. Calculate a vector (the stick) with its origin at the camera's center and pointing towards the sun.



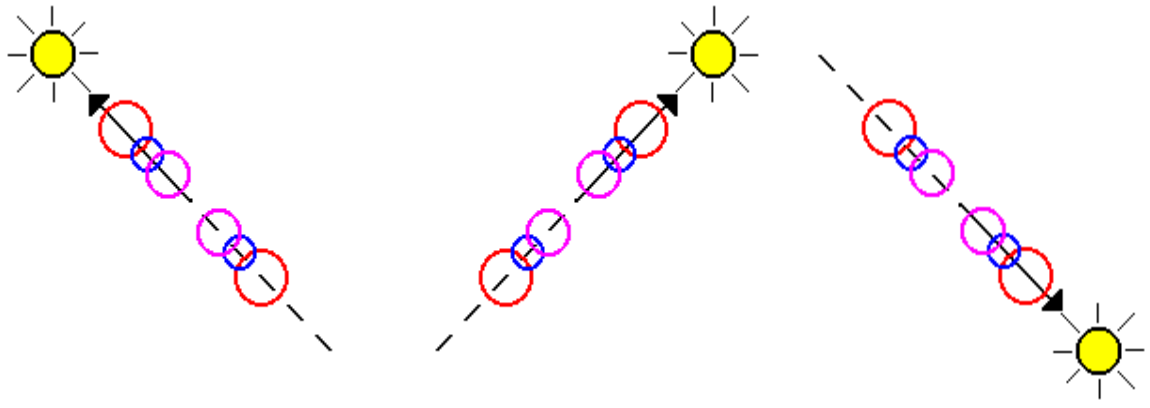
Stick to Sun

Take a series of flare images (the sprites) and arrange them on this vector so that half of them are in 'front' of the origin (positioned along the positive side of the vector) and the other half are symmetrically reflected on the 'back' (negative part of the vector).



Sprites on Stick

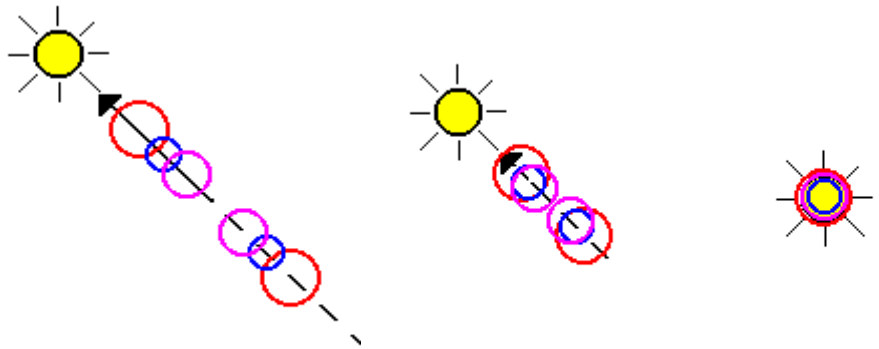
As the camera moves in relation to the sun the relative vector to the sun will rotate around the screen center like a hand on a clock.



Follow the Sun

As the sun approaches the middle of the screen, the vector will get shorter and the flare

images will "bunch up". When the camera is pointing directly at the sun the images will be drawn one on top of each other.



Flares Bunching

The result is a fairly convincing lens flare effect with very little effort. By adjusting the flare sprite images and their spacing we can product several unique effects using the same code.

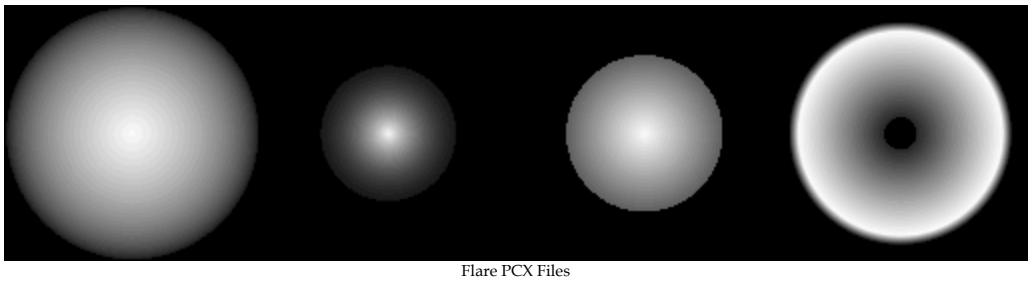
Getting Started

So where do we begin? Lets start by figuring out how we are going to display the projected flare images. You can do this several ways. My first version of this code used 2D Panels which worked okay but I found that using script defined entities gave me more flexibility.

The Flare Entities

Script defined entities behave a little differently then entities created in WED or by the 'create()' command in that they exist "outside" the level. This gives you the extra flexibility to set the layer and camera that they will appear in.

You should have four flare pcx included with this workshop folder. You can use any collection of entities to be your flares but spots and halos seam to work the best (note: some real camera lens flares are hexagonal in shape because they are cause by light reflecting off of the aperture blades). Here are the images we are using:



Flare PCX Files

Creating the Script

We want to design this code so we can add it to several different projects with a minimal amount of modification. To this ends we are going to create all our functions, vars, and entities in a separate file which can easily be included in any of our projects.

We will start by creating a new plain text file called "lflare.wdl". You can create this file in your project folder or you can create a new folder that you will use to store your own user defined template files.

The first thing we are going to add is a comment block to the top of our code so when we are looking at this code months later we will know what the functions in this file do.

```
////////////////////////////////////////////////////////////////////////
// File: lflare.wdl
//             WDL code for lens flare and lighting effects
////////////////////////////////////////////////////////////////////////
```

Next we are going to define a point to store the position of the light causing the flare effect. We will call this position `'flare_sun_pos'`. This point determines when and where the flares appear. We will set this value in the function `'lensflare_create()'`.

```
var     flare_sun_pos[3];       // position of sun in sky
```

The next var will be used to set the ‘trace_mode‘ when we trace from the CAMERA to the sun.

```
var     flare_trace_mode; // used to trace to the sun
```

Each flare sprite entity needs to contain a value that determines where along the vector (the stick) it will be placed. We will redefine the ‘`skin'` skill for the flare sprite entities to be used to store the percent distance between the screen center and the sun that the image will be placed (we will set the individual values when we define the sprites later on).

```
// use the skin parameter to store a % pivot distance in a sprite entity
```

```
define pivot_dist,skin;
```

Next we will define a status value to be used by our lens flare functions to turn on and off the effect and see what state it is in.

```
var     qLensFlare = -1;        // -1 == not created
                                // 0 == off
                                // 1 == on
                                // otherwise == turning off
```

Now lets create the sprite entities that will make up the flare effect. There are eight of them ('flare0_ent' to 'flare7_ent') plus a special flare that will be used for the sun itself.

```
// this is the sun itself
entity flareSun_ent
{
        type = <flare2.pcx>;    // 'sun' flare
        view = CAMERA;          // same camera parameters as the default view
        layer = -6;             // displayed beneath other entity layers
        pivot_dist = 1;         // percent from 'pivot point to sun' (1 == on sun)
        scale_x = 2;            // sun is two times as large as the flares
        scale_y = 2;
}
```

Notice that the 'pivot_dist' value is 1 (100%) this will place it directly over the sun position. For the other sprites we will use values between 0 and 1 for the 'front side' flares and 0 and -1 for the 'back side'.

```
// The 8 lens flare entities
entity flare0_ent
{
        type = <flare0.pcx>;
        view = CAMERA;
        layer = -6;
        pivot_dist = 0.75;      // at distance 0 is the pivot point - the screen center
}
entity flare1_ent
{       // 7 lens reflections
        type = <flare1.pcx>;
        view = CAMERA;
        layer = -6;
        pivot_dist = 0.55;
}
entity flare2_ent
{
        type = <flare2.pcx>;
        view = CAMERA;
        layer = -6;
        pivot_dist = 0.35;
}

entity flare3_ent
{
```

```
        type = <flare3.pcx>;
        layer = -6;
        view = CAMERA;
        pivot_dist = 0.15;
}

entity flare4_ent
{
        type = <flare0.pcx>;
        layer = -6;
        view = CAMERA;
        pivot_dist = -0.25;
}

entity flare5_ent
{
        type = <flare1.pcx>;
        layer = -6;
        view = CAMERA;
        pivot_dist = -0.45;
}

entity flare6_ent
{
        type = <flare2.pcx>;
        layer = -6;
        view = CAMERA;
        pivot_dist = -0.65;
}

entity flare7_ent
{
        type = <flare3.pcx>;
        layer = -6;
        view = CAMERA;
        pivot_dist = -0.85;
}
```

The values here define one type of lens flare effect, one where the flare images repeated half in front half behind the screen center (0,1,2,3*0,1,2,3). By changing the images, scales and '`pivot_dist`' values of each of the entities you can custom make your own flare effect.

Now we will write the functions that control and animate these flares sprites. These will be know as our "interface functions" since they are called from outside of our script (i.e. from the main script). The three interface functions we need are 'create' to set up the lens flare effect, 'start' to start displaying the effect, and 'stop' to stop the effect. Each of these functions will start with the prefix of ''`lensflare_`' so our function will be called ''`lensflare_create()`', '`lensflare_start()`', and ''`lensflare_stop()`'.

We will also write a couple of "helper functions" that will be called from our interface functions. These functions will be used to initialize, show/hide, and position our flare sprites. We will give each of these helper functions a ''flare_'' prefix.

Our first helper function ''flare_init' initializes the values of each our flare entities, setting their alpha channel if we are in D3D mode (transparency if we are in software mode). This function is called from "lensflare_create()".

```
// Desc: this function takes an entity as parameter.
function flare_init(flare_ent)
{
	my = flare_ent; // necessary because function parameters have no type
	my.visible = off;	// start with flares off
	if (video_depth > 8)	// D3D mode?
	{
		ent_alphaset(0,10);// create alpha channel (won't work with standard edition)
		my.bright = on;
		my.flare = on;
	}
	else
	{
		my.transparent = on;	// looks lousy in 8 bit, though
	}
}
```

This function takes an entity as a parameter (e.g. "flare_init(flare0_ent);""). If you haven't used function parameters much you might find this confusing. The WDL Manual says we can only pass "single-number-parameters" but in this case it looks like we are passing an entire entity. The WDL Manual also states that the "original parameter in the calling function remains unchanged" but we are using this function to change values. The reason this works is because each entity is identified in the engine by a single number (the entity's identification number). This number can be passed as a function parameter like any other number. To turn this identification number back into an entity we assign it to the synonym 'my'. Now we can use 'my' to initialize all our values.

The next helper function also takes a flare entity as a parameter. This time we are making the flare visible and positioning it along the vector from the screen center to the sun using the percent value store in it's 'pivot_dist' skill. The sun's screen position is stored in 'temp.x/temp.y'', which is calculated in the calling function 'lensflare_start()''. Because the sprite is projected from the screen to world coordinates using the command 'rel_for_screen()' the distance that the flares appear from the screen ('my.z'') will effect the size of the flare. This is another value you can adjust to change the effect.

```
// Desc: places a flare at temp.x/temp.y deviations from screen center
function flare_place(flare_ent)
{
	my = flare_ent;
	my.visible = on;
```

```
        // multiply the pixel deviation with the pivot factor,
        // and add the screen center
        my.x = temp.x*my.pivot_dist + 0.5*screen_size.x;
        my.y = temp.y*my.pivot_dist + 0.5*screen_size.y;
        my.z = 750;     // screen distance, determines the size of the flare
        rel_for_screen(my.x,camera);
}
```

Our last helper function is simply used to turn on and off (make visible or hide) all our flare sprites. This function is also called from '`lensflare_start()`' and takes a simple parameter of 'ON' or 'OFF'.

```
// Desc: this function turns all the flareN_ent and flareSun_ent on or
//      off depending on the value pass in 'on_off'.
function flare_visible(on_off)
{
        flareSun_ent.visible = on_off;

        flare0_ent.visible = on_off;
        flare1_ent.visible = on_off;
        flare2_ent.visible = on_off;
        flare3_ent.visible = on_off;
        flare4_ent.visible = on_off;
        flare5_ent.visible = on_off;
        flare6_ent.visible = on_off;
        flare7_ent.visible = on_off;
}
```

Our first interface function, "lensflare_create" sets up the position of the light source and lens flare sprite's alpha values.

```
// Desc: setup the lens flare effect
function lensflare_create()
{
```

First thing we will do is use our helper function '`flare_init()`' to initialize all of our flare sprites (including the sun sprite).

```
        // set alpha values for each entity
        flare_init(flare0_ent);
        flare_init(flare1_ent);
        flare_init(flare2_ent);
        flare_init(flare3_ent);
        flare_init(flare4_ent);
        flare_init(flare5_ent);
        flare_init(flare6_ent);
        flare_init(flare7_ent);

        flare_init(flareSun_ent);       // the sun flare
```

Next we will use the engine values for 'SUN_ANGLE' to set the 'flare_sun_pos' vector. This way the Sun Azimuth and Elevation values set in WED will be used to position our flare light source. You can replace this section with whatever value makes sense for your level. For example: a space sim might have the sun at the origin of the level (0,0,0). Whatever value you give to 'flare_sun_pos' it is important that you can trace to that point from some section of the level or the lens flare will never appear.

```
	// set the sun point (for tracing to this point to see if sun is visible)
	// x = (h*cos(tilt))*sin(pan);
	flare_sun_pos.x = (500000 * cos(sun_angle.tilt)) * sin(sun_angle.pan);
	// y = (h*cos(tilt))*cos(pan);
	flare_sun_pos.y = (500000 * cos(sun_angle.tilt)) * cos(sun_angle.pan);
	// z = h*sin(tilt)
	flare_sun_pos.z =  500000 * sin(sun_angle.tilt);
```

Next we will set the ''flare_trace_mode'. In our simple demo you only need to ignore passable blocks since the 'flare_sun_pos' lies outside the passable skybox. I've also added 'IGNORE_MODELS' because when you use this in a first person style game the CAMERA will be inside a model and therefore will never be able to trace to the sun.

```
	//  set the trace mode to be used to trace to the sun
	//IGNORE_PASSABLE needed for tracing through the sky box
	flare_trace_mode = IGNORE_PASSABLE + IGNORE_MODELS;
```

We will start with the lens flare effect turned 'off'.

```
	qLensFlare = 0; // start 'off'
}
```

The next function is the real core of the lens flare code. If you are typing this in as you are following along make sure to add this function after 'lensflare_create()'.

```
// Desc: start and animate a lens flare effect as long as qLensFlare == 1
function	lensflare_start()
{
```

The first thing we do is check to see if the user has already created the lens flares by calling 'lensflare_create()' directly or by calling this function previously. If not we call the 'lensflare_create()' code now.

```
	if(qLensFlare == -1) // create the lens flares
	{
		lensflare_create();
	}
```

Next we check to see if the lens flare is already active. With this code it only makes sense that one lens flare effect be running at a time so, if its already running, we will return at this point.

```
if(qLensFlare == 1)  // lens flare already started
{
	return;
}
```

We give the 'lensflare_create()' code time to set up and set the 'gLensFlare' var to show that the lens flare code is running.

```
wait(1);  // allow for setup time for "lensflare_create"
qLensFlare = 1; // mark lens flare as on
```

The main while loop for the effect will run once each frame cycle until something changes the 'qLensFlare' value to something other than '1'.

```
// place lens flares
while(qLensFlare == 1)
{
	// Animate lens flare
```

We will use the 'vec_to_screen()' command to make a quick check to see if the light source appears in the view (we have to make a copy of it first so we don't change it's value). If the point is not in the camera's view we hide the flare sprites using our helper function ''flare_visible(off)''.

```
vec_set(temp,flare_sun_pos);
if(vec_to_screen(temp,camera) == 0)
{
	// Outside of View cone... remove lens flares
	flare_visible(off);
}
else	 // check for trace to sun
{
```

If the light source passes the 'vec_to_screen()' test we can check to see if it is visible from the current CAMERA position by using the 'trace()' command. As I noted before, it is important that the 'flare_sun_pos' be positioned somewhere that the camera can trace to at least part of the time. In the case of the test map included with this workshop, a passable sky block surrounds the world so even though 'flare_sun_pos' has been set very far away we can trace to it from all the sunlit areas of the map. You may need to adjust the 'flare_sun_pos' value or the 'flare_trace_mode' parameters to make this work in your level.

```
	// trace to the 'sun'
	trace_mode = flare_trace_mode;
	// check if line to sun is blocked
	if (trace(camera.x,flare_sun_pos) != 0)
	{
```

If something is blocking the CAMERA from the sun we hide the flare sprites using our helper function 'flare_visible(off)'' like we did before.

```
                        // Something is blocking us.. hide lens flare
                        flare_visible(off);
                }
                else
                {
```

If we have made it to this section of the code then we are facing the sun light source and nothing is blocking it. We will use the sun's XY screen position (calculated from the earlier 'vec_to_screen()' call) and offset it by half the screen''s width and height to get the actual XY screen distance from the screen center.

```
                        // temp now contains the sun XY screen position
                        // subtract the screen center, needed for flare_place()
                        temp.x -= 0.5 * screen_size.x;
                        temp.y -= 0.5 * screen_size.y;
```

This is the temp value used by the helper function 'flare_place()'' to place the 8 flare a sun sprites.

```
                        // place flares according to deviation and their pivot distance
                        flare_place(flareSun_ent);
                        flare_place(flare0_ent);
                        flare_place(flare1_ent);
                        flare_place(flare2_ent);
                        flare_place(flare3_ent);
                        flare_place(flare4_ent);
                        flare_place(flare5_ent);
                        flare_place(flare6_ent);
                        flare_place(flare7_ent);

                }
        }
```

Wait one frame cycle before doing this all over again.

```
        wait(1);        // animate each cycle
}
```

We will only reach this section of code if something changes 'qLensFlare' to a value other than 1. In this case we should make sure that the lens flares are hidden (by calling the function 'flare_visible()'') and make qLensFlare equal to the 'off' value (0) to signal that we are done.

```
        // Remove lens flares
        flare_visible(off);
        qLensFlare = 0;     // mark lens flare as off
}
```

This last function is the simplest. To turn the lens flare off all we have to do is set the 'qLenFlare' value to a value other than 1. By setting the value to '.5' here the main while

loop in 'lensflare_start()' will exit.

```
// Desc: stop the lens flare effect
function lensflare_stop()
{
	qLensFlare = .5;	// signal to stop
}
```

How to use this Code

Now that we have the code finished it is real simple to use it to add lens flares to any level. Make sure "lflare.wdl" is in your path and then include it at the beginning of your main script (right under the template includes). To start the lens flare effect simply call "lensflare_start();" and call "lensflare_stop();" to stop the effect. If you plan to start the effect sometime other than the start of the game you might want to call "lensflare_create" in your main function to avoid any in game slowdown cased by setting the alpha values of the flare sprites.

```
////////////////////////////////////////////////////////////////////////////
// Lensflare test level
////////////////////////////////////////////////////////////////////////////
include <lflare.wdl>;// our flare code

////////////////////////////////////////////////////////////////////////////
// The engine starts in the resolution given by the follwing vars.
var video_mode = 6;
var video_depth = 16;
////////////////////////////////////////////////////////////////
// The MAIN function is called at game start
function main()
{
	load_level(<flr_test.wmb>);
	_move_straight();	// predefined camera movement

	sky_clip = -65;

	lensflare_create();	// init the lens flare
	lensflare_start();	// start lens flare

	wait(2);
	// start with lens flare in frame
	camera.pan = 25;
	camera.tilt = 15;
}

ON_2 = lensflare_start;
ON_3 = lensflare_stop;
```

Wrap Up

I hope you have enjoyed the first mini workshop. If you have any questions, comments, or improvements for this or any other workshop please post them to the Conitec User Forum: http://www.conitec.net/ubbcgi/Ultimate.cgi

Finished Level Running