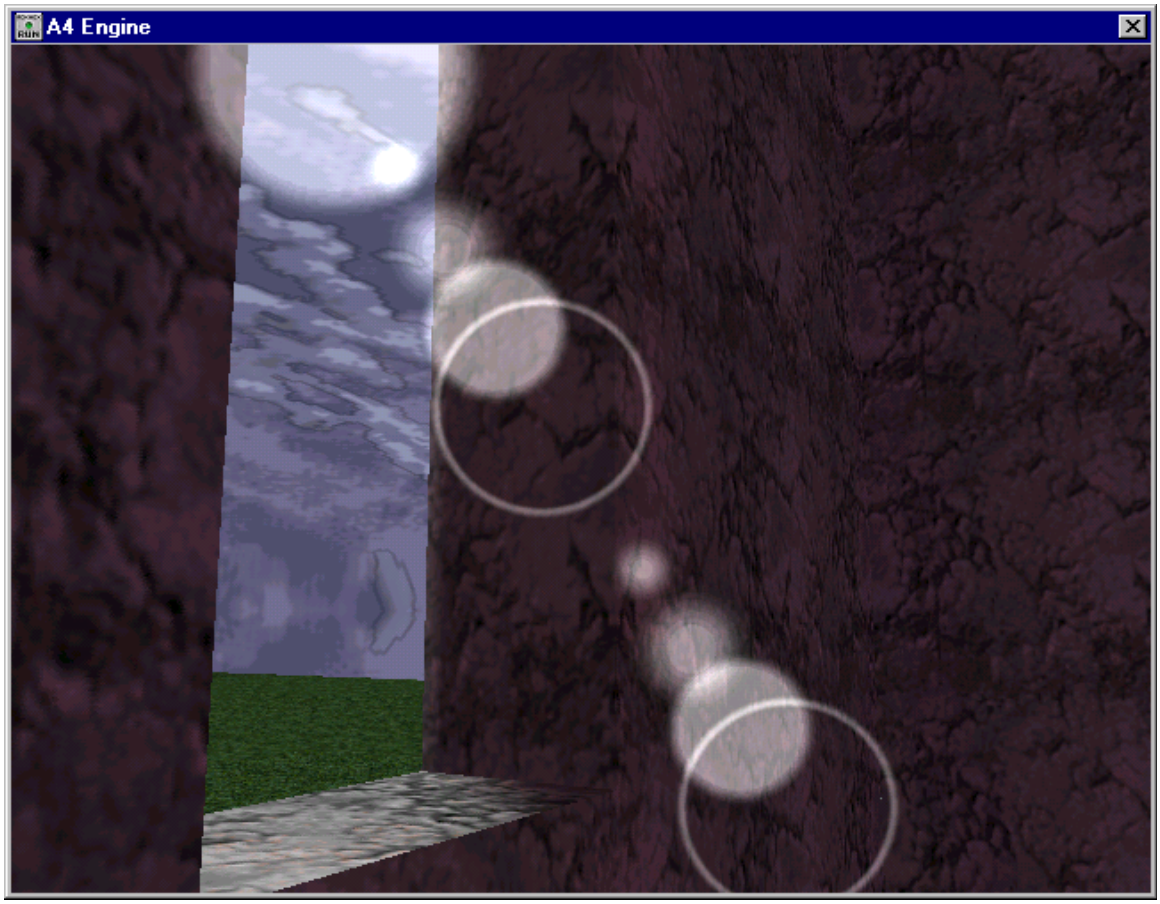
3D GameStudio

Miniworkshop Linsenreflexion



von Doug Poston / Conitec March 2001

Inhalt

Lieber Leser,

Willkommen zum ersten "Miniworkshop". Während sich "normale" Workshops mit grossen und in sich abgeschlossenen Themen beschäftigen (z.B. Grundlagen von Rollenspielen oder Flugsimulatoren), behandeln diese "Mini"-Kurse kleinere Teilbereiche, die Sie in der Regel einem bereits vorhandenen Projekt hinzufügen können. Unser erstes Thema ist also "Linsenreflexion".

Wie die normalen Workshops ist auch dieser für solche Anwender geschrieben, die bereits über eine gewisse Erfahrung mit 3DGameStudio verfügen. Ich setze voraus, dass Sie die Tutorials durchgearbeitet haben und zumindest die Grundkenntnisse besitzen, die man zur Arbeit mit GameStudio und WDL braucht.

Dieser Text ist zur Ergänzung der übrigen, mit 3DGameStudio mitgelieferten Dokumentation gedacht, nicht als Ersatz. Sollte Ihnen in diesem Kurs etwas unklar sein, lesen Sie bitte im 3DGameStudio-Handbuch nach. Für eventuelle unklare Formulierungen, fehlerhafte Codes, Irrtümer oder Versäumnisse, entschuldige ich mich im voraus.

Ich hoffe, Sie empfinden diese neuen Minisworkshops als informativ und sie machen Ihnen Spass.

-Doug.

Zuallererst:

Besorgen Sie sich die neueste Version

Bevor Sie anfangen vergewissern Sie sich, dass Sie auch wirklich die neueste Version von 3DGameStudio (Engine, Editoren und vorgefertigte (Template-) Skripte) haben. Das ist sehr wichtig, denn ich will die Vorteile der neuesten Version ausnutzen und verwende daher einige Befehle und Features, die erst mit dem neuesten Update möglich sind.

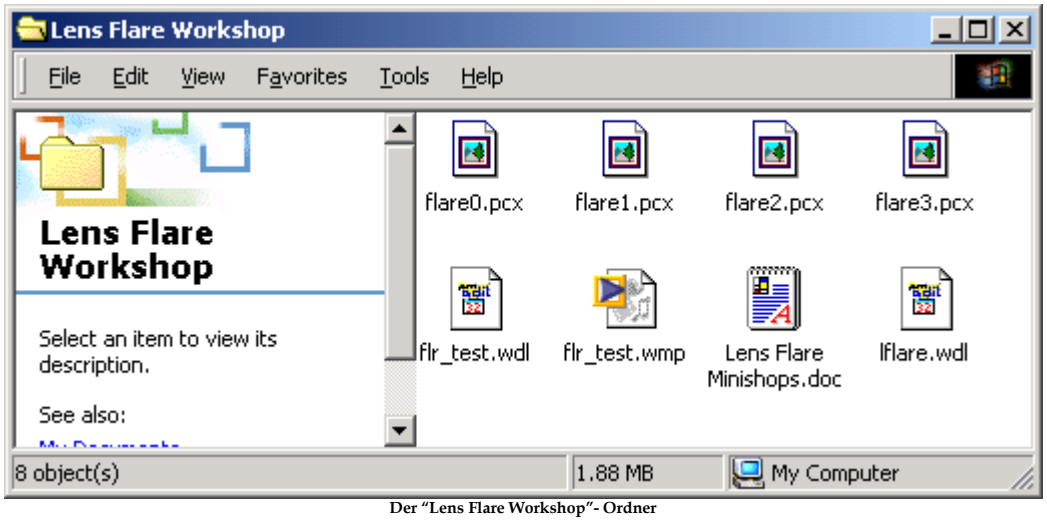
Für diesen Workshop wurde die A5, Version **5.05** verwendet. Ich mache mir also Features der A5 zunutze. Sobald ich ein solches Feature, das ausschliesslich mit A5 verfgübar ist, anwende, mache ich eine Anmerkung und schlage eine "Umgehungslösung" vor (falls es eine gibt). Wenn Sie das Ganze mit einer älteren Engine versuchen, tun Sie dies auf eigenes Risiko.

Bereiten Sie Ihre Arbeitsumgebung vor

Erstellen Sie in Ihrem Gstudio-Ordner ein Verzeichnis mit dem Namen "Lens Flare Workshop". Das ist jetzt der Ordner, in dem Ihre sämtlichen Spieleelemente gespeichert werden. Entzippen Sie den Inhalt des Linsenreflexions-Workshops in diesen Ordner.

Folgende Dateien sollten sich nun in Ihrem Ordner befinden:

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



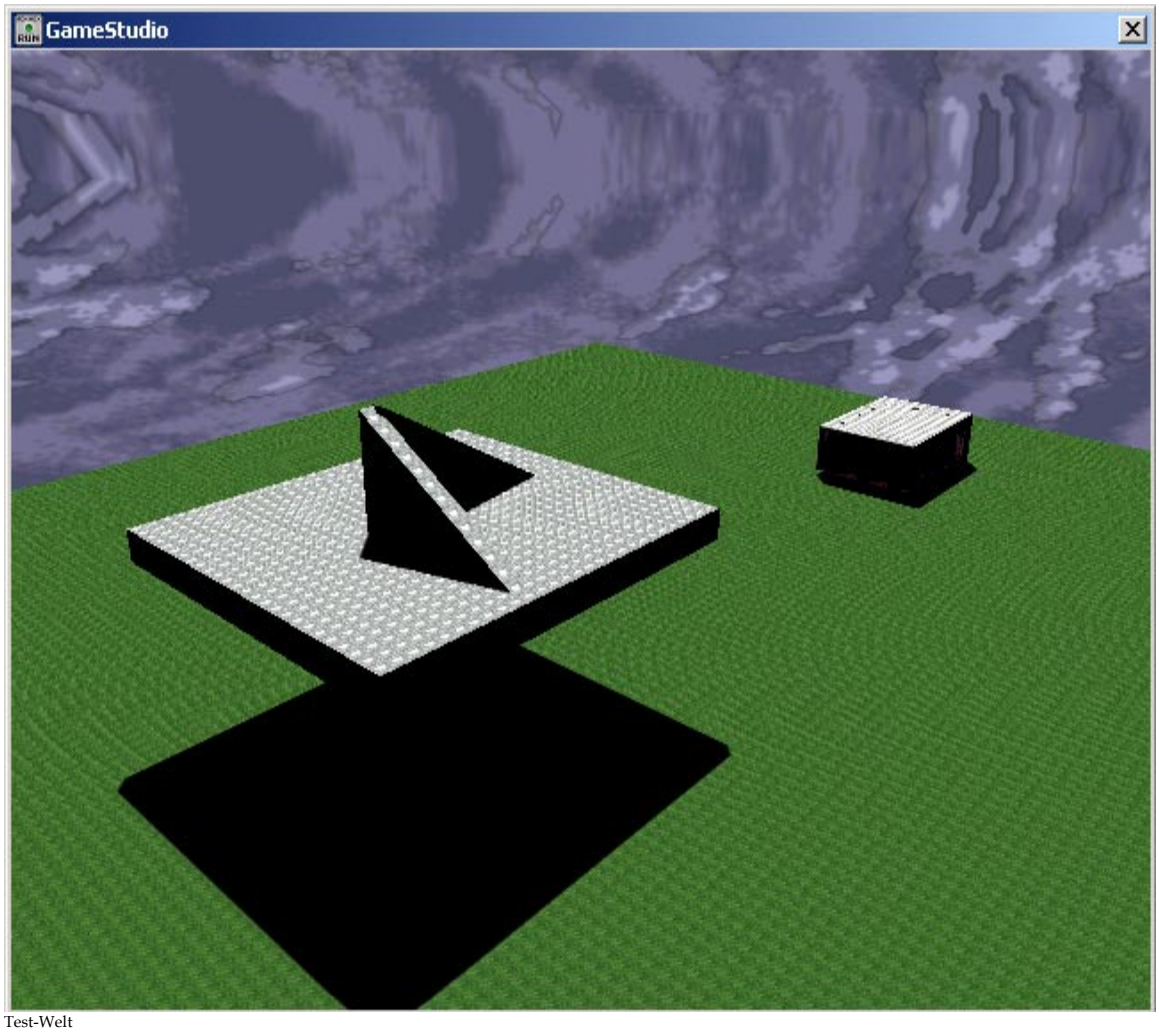
Der "Lens Flare Workshop"- Ordner

Die Datei **lflare.wdl** enthält sämtlichen Quellcode für die Linsenreflexion, die wir in diesem Workshop behandeln. Wenn Sie jetzt gleich sehen wollen, wie sich dieser Code bei der Arbeit macht, starten Sie den **flr_test**-Level über build/run in WED. Gehen Sie ein bischen spazieren und beobachten Sie, wann Linsenreflexionen sichtbar werden und wie sie sich bewegen.

Erstellen des Levels

Weil das hier ein Miniworkshop ist, werde ich Sie nicht mit einer Schritt für Schritt-Erklärung zum Erstellen eines einfachen Levels aufhalten. Der Code, den wir erzeugen werden, funktioniert mit den meisten Aussenleveln. Falls Sie einen Level brauchen, um überhaupt anfangen zu können (oder zum debuggen), habe ich diesem Workshop zwei beigefügt (**flr_tst.wmp** und **flr1test.wmp**).

Bedenken Sie, dass Sie die Position der Sonne sowohl im Level (über map properties zum rendern von Licht und Schatten), als auch in WDL (**sun_angle**) setzen können. Diese beiden Positionen sind jedoch nicht absolut identisch! Damit Sonnenlicht, Schatten und Linsenreflexion in Ihrem Level gut und korrekt aussehen, achten Sie darauf, Ihren **sun_angle.pan** und **sun_angle.tilt** auf dieselben Azimuth- und Deklinationswinkel zu setzen, die sie in WED eingegeben haben.



Test-Welt

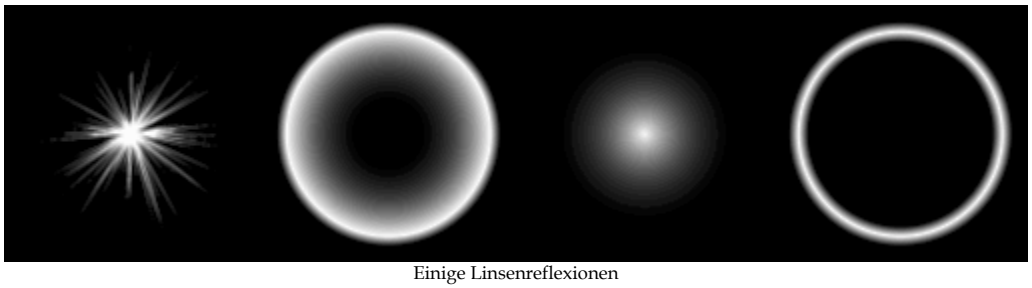
Wenn Sie Ihr eigenes Level verwenden, ist es wichtig, dass Sie mindestens einen Bereich haben, in dem Sie eine Verbindungslinie von der Kamera zu dem in **flare_sun_pos** angegebenen Positionswert ziehen können (**trace**), sonst wird die Linsenreflexion nämlich gar nicht erscheinen. (Näheres zu diesem Wert kommt später).

Was ist eine „Linsenreflexion“?

Der Begriff Linsenreflexion wurde von verschiedenen Leuten zur Beschreibung von einer Menge unterschiedlicher Dinge verwendet. Eine gute Definition stammt aus dem Buch "The Art and Science of Digital Compositing" (Ron Brinkmann, ISBN: 0121339602)

Die Reflexion eines hellen Lichts, das direkt in das Objektiv einer Kamera einfällt.

Diese können die Form von Ringen, Strahlen oder hellen Punkten annehmen und scheinen im All zu schweben. Intensität und Position dieser Artefeakte hängen vom Objektiv und seiner Relation zu Position und Intensität des einfallenden Lichtes ab.



Einige Linsenreflexionen

Wenn Sie nicht gerade sehr dicke Brillengläser tragen oder viel Zeit mit einer Kamera oder anderem optischen Gerät verbringen, sehen Sie im täglichen Leben normalerweise keine Linsenreflexionen. Gelegentlich tauchen solche Reflexionen in Film und Fernsehen auf (die "X-Files" sind ein gutes Beispiel dafür). Manchmal erscheinen diese Effekte aus Versehen, oftmals aber werden sie aus dramaturgischen Gründen eingesetzt (mehr und mehr werden diese der Dramaturgie dienenden Linsenreflexionen übrigends hinterher per Computer eingefügt).

Warum also wollen Sie nun Linsenreflexionen in Ihrem Spiel haben? Vielleicht möchten Sie ein ein filmähnliches Feeling vermitteln, wollen dem Player das Gefühl geben, er stünde hinter einer Glasscheibe oder Sie finden den Effekt einfach nur hübsch. Geschickt eingesetzt kann eine Linsenreflexion den Eindruck Ihres Levels und des Gameplays verbessern. Stellen Sie sich z.B. vor, der Spieler sucht sich eine Schussposition aus und muss dabei auf die Minimierung einer Linsenreflexion bei seinem Visier achten.

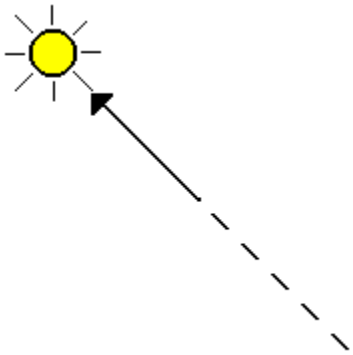
So nett der Effekt sein kann, vermeiden Sie Misbrauch! Jetzt, da Linsenreflexionen einfach einzubauen ist, besteht die Gefahr, dass ein Zuviel leicht billig und überfrachtet wirkt.

So funktioniert unsere Linsenreflexion

Wie Sie sehen, gibt es viele verschiedene Typen von Linsenreflexionen. Der am dramatischsten aussehende und damit auch der, der am häufigsten in Filmen oder Spielen eingesetzt wird, ist der Effekt mit mehreren Lichtsternen. Diesen Effekt erreicht man, indem man direkt in eine helle Lichtquelle, wie die Sonne filmt. Er entsteht aus der Oberflächenreflexion der Linsen.

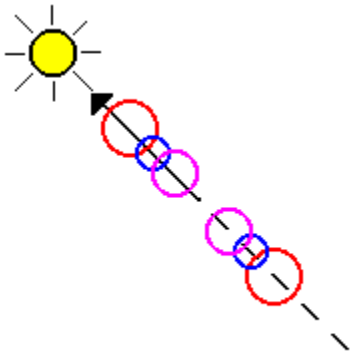
Um einen wirklich "realistischen Effekt" zu erzielen, könnten wir versuchen Linse und Licht möglichst exakt nachzubilden, aber das würde uns sehr viel Zeit kosten und liese sich in Echtzeit gar nicht reproduzieren. Die Methode, die wir verwenden ist wesentlich einfacher und verhält sich doch fast genauso.

Es gibt verschiedene Möglichkeiten eine Reflexion vorzutäuschen. Die, die ich hier benutze, ist die "Stab und Sprite"-Methode. Berechnen Sie einen Vektor (den Stab) dessen Ursprung im Zentrum der Kamera liegt und der zur Sonne zeigt.



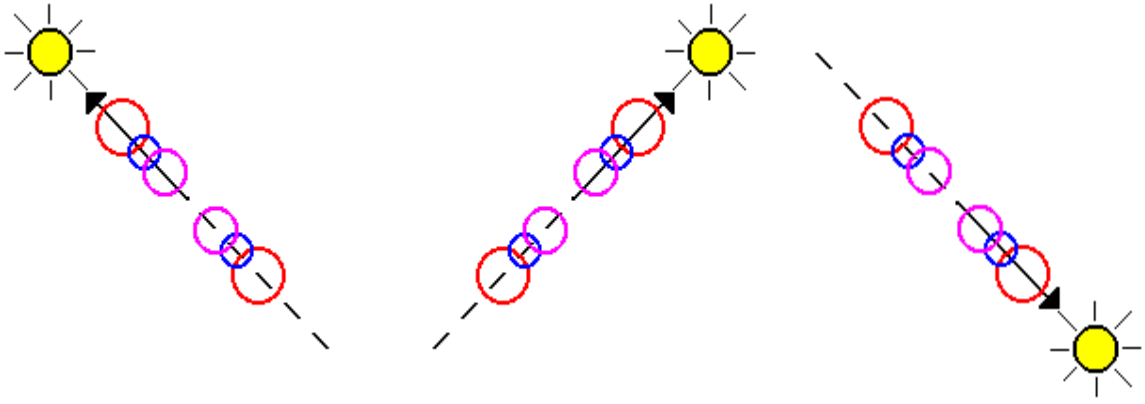
Vektor zur Sonne

Nehmen Sie eine Reihe von Reflexionsbildern (die Sprites) und arrangieren Sie diese so auf dem Vektor, dass die Hläfte davon 'vor' dem Ursprung (entlang der positiven Seite des Vektors) und die andere Hälfte symmetrisch reflektiert 'dahinter' (also auf dem negativen Teil des Vektors) liegt.



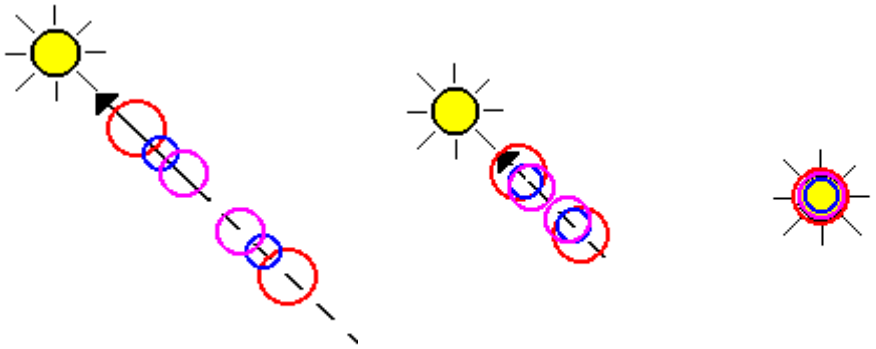
Sprites auf dem Stab

Wenn die Kamera sich relativ zur Sonne bewegt, rotiert der Richtungsvektor zur Sonne über den Bildschirm wie der Zeiger einer Uhr.



Folge der Sonne

Erreicht die Sonne die Bildschirmmitte, wird der Vektor kürzer und die Reflexionsbilder bündeln sich. Zeigt die Kamera direkt in die Sunne, werden die Bilder direkt übereinander gezeichnet.



Reflexionsbündelung

Das Ergebnis ist ein ziemlich überzeugender Effekt von Linsenreflexion und das mit wenig Anstrengung. Indem wir die Reflexionsbilder (Sprites) und Ihre Abstände einstellen, können wir mit ein und demselben Code verschiedene einzigartige Effekte produzieren.

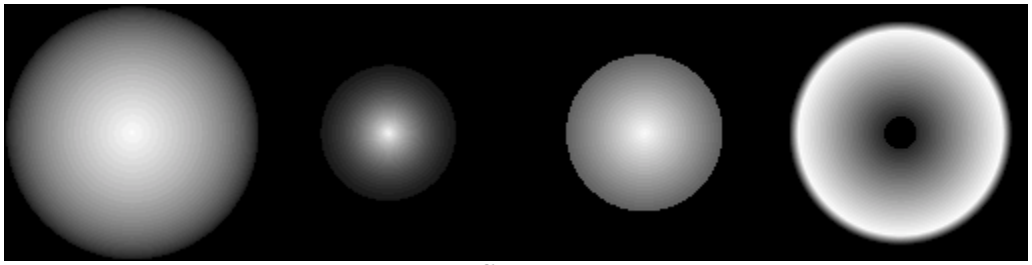
Fangen wir an!

Also, wo fangen wir an? Beginnen wir am besten damit, herauszufinden, wie wir die Reflexionsbilder darstellen sollen. Dies lässt sich auf verschiedene Weise bewerkstelligen. Meine erste Version dieses Codes verwendete 2D-Panels, was zwar ordentlich funktionierte, aber ich habe herausgefunden, dass im Skript definierte Entities mehr Flexibilität bieten.

Die Licht-Entities

Entities, die im Skript definiert sind, verhalten sich ein wenig anders, als solche, die in WED oder über den **`create()`**-Befehl erzeugt wurden, denn sie existieren "ausserhalb" des Levels. Das gibt Ihnen die zusätzliche Flexibilität Layer und Kamera, in denen Sie erscheinen sollen, zu setzen.

In diesem Worshop-Ordner sollten Sie vier **`flare.pcx`**-Dateien vorfinden. Natürlich können Sie jede beliebige Entity-Kollektion als Reflexionslichter verwenden, allerdings scheint es so, als funktionierten Punkte und Kreise am besten. (Anmerkung: einige reale Kamera-Linsenreflexionen sind sechseckig, was von der Rückreflexion des Lichts durch die Blendensegmente herrührt). Hier die Bilder, die wir verwenden:



PCX -Dateien

Erstellen des Skripts

Wir wollen den Code so schreiben, dass wir ihn in verschiedene Projekten eingliedern können, ohne grosse Veränderungen vornehmen zu müssen. Zu diesem Zweck erstellen wir alle unsere Funktionen, Variablen und Entities in einer separaten Datei, die sich leicht in jedes unserer Projekte einfügen lässt (**include**).

Zerst erzeugen wir also eine neue, leere Textdatei und nennen sie **lflare.wdl**. Entweder erstellen Sie diese Datei in ihrem Projektordner oder Sie erzeugen ein neues Verzeichnis, dass Sie zur Verwaltung ihrer eigenen Anwender-definierten Templates verwenden.

An den Anfang unseres Skriptes setzen wir einen Kommentarblock, damit wir auch ein paar Monate später noch wissen, wozu die Funktionen dieses Codes eigentlich gut sind.

```
////////////////////////////////////////////////////////////////////////
// File: lflare.wdl
//              WDL-Code für Linsenreflexionen und Lichteffekte
////////////////////////////////////////////////////////////////////////
```

Als erstes definieren wir einen Punkt um die Lichtposition zu speichern, die den Reflexionseffekt auslöst. Diese Position nennen wir **flare_sun_pos.** Der Punkt legt fest, wann und wo die Reflexionen erscheinen. Den Wert selbst setzen wir dann in der Funktion **lensflare_create()**.

```
var     flare_sun_pos[3];       // Position der Sonne am Himmel
```

Die nächste Variable, die gebraucht wird, ist **trace_mode**, mit dem wir den Modus festlegen, in welchem wir die Verbindungslinie von der Kamera zur Sonne messen.

```
var     flare_trace_mode; // Messlinie zur Sonne
```

Jeder Licht-Sprite-Entity braucht einen Wert sein, der festlegt wo sie entlang dem Vektor (Stab) gesetzt wird. Um die prozentuale Entfernung, in der das Bild zwischen dem Bilschirmmittelpunkt und der Sonne plaziert wird, zu berechnen, definieren wir den **skin**-Skill für die Licht-Entities neu. (Die individuellen Werte setzen wir später, wenn wir die Sprites definieren).

```
// verwende den Skin-Parameter, um den prozentualen Angelpunkt der Entfernung einer Sprite-Entitiy
// zu verwalten
define pivot_dist,skin;
```

Nun definieren wir einen Status-Wert, der von unseren Linsenreflexions-Funktionen zum An- und Abschalten des Effekts benutzt wird und anhand dessen man sehen kann, in welchem Zustand er sich befindet.

```
var     qlensflare = -1; // -1 == nicht erstellt
                         // 0 == aus
                         // 1 == an
                         // otherwise == ausschalten
```

Kreieren wir jetzt die Sprite-Entities aus denen sich der Lichteffekt zusammensetzt. Davon haben wir acht (**flare0_ent** to **flare7_ent**), zuzüglich eines spezielles Lichts, das wir für die Sonne selbst benutzen.

```
// das ist die Sonne höchstpersönlich
entity flaresun_ent
{
        type = <flare2.pcx>;     // 'Sonnenlicht'
        view = camera;           // dieselben Kameraparameter, wie im Standard-View
        layer = -6;              // unterhalb anderer Entity-Layers (Schichten) dargestellt
        pivot_dist = 1;          // Prozentualwert vom 'Angelpunkt zur Sonne' (1 == auf Sonne)
        scale_x = 2;             // Sonne ist doppelt so gross, wie die Lichter
        scale_y = 2;
}
```

Beachten Sie dass der **pivot_dist**-Wert **1** (100%) ist, was den Lichtsprite direkt über die Sonnenposition setzt. Für die anderen Sprites nehmen wir Werte zwischen **0** und **1** für die 'Vorderseite' und **0** und **-1** für die 'Rückseite'**.**

```
// Die 8 Reflexions-Lichtentities
entity flare0_ent
{
        type = <flare0.pcx>;
        view = camera;
        layer = -6;
        pivot_dist = 0.75; // bei einer Distanz von 0, ist der Angelpunkt die Bilschirmmitte
}
entity flare1_ent
{       // 7 Linsenreflexionen
        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;
}
```

Die Werte hier definieren einen Typus von Lichtreflexionseffekt bei dem die Lichtgrafiken zur Hälfte vor und zur Hälfte hinter dem Bildschirmmittelpunkt wiederholt werden (**0,1,2,3*0,1,2,3**). Wenn Sie die Skalierung und **pivot_dist**-Werte der Grafiken jeder einzelnen Entity verändern, können Sie sich Ihren eigenen Lichteffekt masschneidern.

Nun schreiben wir die Funktionen, die diese Licht-Sprites steuern und animieren. Diese kennen wir als sogenannte "Interface-Funktionen", die aus anderen Skripten (also ausserhalb des betreffenden) aufgerufen werden können (z.B. vom Main-Skript). Die drei Interface_Funktionen, die wir brauchen sind **create,** zum Erstellen der Linsenreflexion, **start,** um die Darstellung des Effektes zu initialisieren und **stop**, um das Ganze anzuhalten. Jede dieser Funktionen beginnt mit einem vorangestellten **lensflare_** . Unsere Funktionen heissen also **lensflare_create()**, **lensflare_start()**, und **lensflare_stop()**.

Zusätzlich schreiben wir noch ein paar "Hilfsfunktionen", die von unseren Interface-Funktionen aufgerufen werden. Wir brauchen diese Funktionen zum Initialisieren, zeigen/verstecken und Positionieren unserer Licht-Sprites. Alle unsere Hilfsfunktionen bekommen die Vorsilbe **flare_** .

Unsere erste Hilfsunktion heist flare_init und initialisiert die Werte einer jeden Licht-Entity. Falls wir im D3D-Modus sind, setzt sie ihre Alpha-Kanal-Werte, oder, wenn wir in einem Software-Modus sind, die Transparenz. Diese Funktion wird von lensflare_create()aufgerufen.

```
// Beschr: diese Funktion hat eine Entity als Parameter
function flare_init(flare_ent)
{
        my = flare_ent;  // nötig, da Funktionsparameter keinen Typus haben
        my.visible = off;        // starte mit abgeschalteten Lichtsprites
        if (video_depth > 8)     // D3D-Modus?
        {
                ent_alphaset(0,10);// erzeuge Alpha-Kanal (funktioniert nicht mit der
                                   // Standard-Version)
                my.bright = on;
                my.flare = on;
        }
```

```
        else
        {
                my.transparent = on;     // sieht in 8 Bit lausig aus, trotzdem
        }
}
```

Diese Funktion hat eine Entity als Parameter (z.B. **flare_init(flare0_ent);**). Das mag Ihnen vielleicht, vor allem, wenn Sie noch nicht viel mit Funktionsparametern gearbeitet haben, verwirrend vorkommen. Im WDL-Handbuch steht schliesslich, wir dürfen nur "einzahlige Parameter" verwenden, in diesem Fall aber sieht es so aus, als könnten wir eine komplette Entity einsetzen. Das Handbuch behauptet auch, dass der "Originalparameter in der aufrufenden Funktion unverändert bleibt", wir aber verwenden diese Funktion zum verändern von Werten. Der Grund aus dem das funktioniert, ist der, dass jede Entity in der Engine über eine einzelne Nummer identifiziert wird. Diese Identifikationsnummer einer Entity kann wie jede andere Nummer auch, als Funktionsparameter eingesetzt werden. Um aus der Nummer wieder eine Entity zu machen, weisen wir ihr das Synonym **my** zu. Nun können wir **my** zum Intitialisieren aller unserer Werte benutzen.

Auch die nächste Hilfsfunktion bekommt eine Licht-Entity als Parameter. Dieses Mal machen wir das Licht sichtbar und positionieren es entlang dem Vektor von der Bildschirmmitte zur Sonne, wobei wir den Prozentualwert verwenden, der im **pivot_dist**-Skill der Entity festgehalten ist. Die Position der Sonne am Bildschirm wird von **temp.x** und **temp.y** vorgegeben und von der Funktion **lensflare_start()** berechnet. Weil der Sprite mit dem Befehl **rel_for_screen()** mit den Welt-Koordinaten auf dem Bildschirm projiziert wird, beeinflusst der Abstand, in dem die Lichter vom Bildschirm erscheinen (**my.z**), ihre Grösse.

```
// Beschr: plaziert ein Licht an temp.x,temp.y Abweichungen von der Bilschirmmitte
function flare_place(flare_ent)
{
        my = flare_ent;
        my.visible = on;

        // multipliziere dir Pixelabweichung mit dem Prozentualwert
        // und addiere die Bildschirmmitte
        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;      // Bildschirmabstand, bestimmt die Grösse des Lichtes
        rel_for_screen(my.x,camera);
}
```

Unsere letzte Hilfsfunktion wird einfach nur zum An- und Abschalten (sichtbar machen oder verstecken) unserer Licht-Sprites gebraucht. Auch diese Funkton wird von **lensflare_start()** aufgerufen und nimmt die simplen Parameter **on** oder **off**.

```
// Beschr: diese Funktion schaltet in Abhängigkeit des in 'on_off' gegebenen
// Wertes, alle flaren_ent und flaresun_ent an oder aus

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;
}
```

Unsere erste Interface-Funktio, **lensflare_create** setzt die Position der Lichtquelle sowie die Alphawerte der Lichtsprites für die Linsenreflexion.

```
// Beschr: bilde den Linsenreflexions-Effekt
function lensflare_create()
{
```

Zuerst benutzen wir unsere Hilfsfunktion **flare_init()** und intitialisieren all unsere Licht-Sprites (inklusive des Sonnen-Sprites).

```
        // setze die Alphawerte für jede 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);        // Sonnenlicht
```

Als nächstes bedienen wir uns der Enginewerte für den Sonnenwinkel s**un_angle** und setzen den **flare_sun_pos**-Vektor. So werden die Sonnen-Azimuth- und Höhenwerte, die in WED gesetzt wurden zum Plazieren unserer Lichtquelle benutzt. Diesen Abschnitt des Codes können Sie durch solche Werte ersetzen, die für Ihr Level passend erscheinen. Eine All-Simulation könnte die Sonne z.B. im Urspung des Levels (**0,0,0**) haben. Welchen Wert Sie **flare_sun_pos** auch immer geben, es ist wichtig, dass sie von einigen Regionen des Levels aus eine Verbindungs-Messlinie zu diesem Punkt herstellen können. Denn sonst wird die Linsenreflexion niemals erscheinen.

```
        // setze den Sonnenpunkt(für die Verbindungslinie zum messen und um
        // festzustellen, ob die Sonne sichtbar ist).
        // 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);
```

Jetzt kommt der **flare_trace_mode**. In unserem einfachen Beispiel können Sie passierbare Blocks einfach ignorieren, da **flare_sun_pos** ausserhalb der passierbaren Himmels-Box liegt. Ich habe ausserdem noch **ignore_models** hinzugefügt. Wenn Sie das Ganze nämlich in einem Spiel der Ich-Perspektive (first person) anwenden, ist die Kamera ja innerhalb eines Models und wird daher nie in der Lage sein, eine Verbindungslinie zur Sonne herzustellen.

```
        //  setze den Verbindungs-Modus, um zur Sonne zu "tracen"
        //IGNORE_PASSABLE wird zum messen durch die Skybox hindurch gebraucht
        flare_trace_mode = ignore_passable + ignore_models;
```

Wir starten mit ausgeschalteter Linsenreflexion (**off**).

```
        qlensflare = 0;  // starte 'aus'
}
```

Die nächste Funktion ist der eigentliche Kern des Linsenreflexions-Codes. Wenn Sie das abtippen, achten Sie darauf, dass Sie diese Funktion im Skript nach **lensflare_create()** eingeben.

```
// Beschr: starte und animiere einen Linsenreflexionseffekt solange, wie qLensFlare == 1
function lensflare_start()
{
```

Als erstes prüfen wir, ob der Anwender die Linseneffekte bereits entweder direkt über **lensflare_create()** oder durch früheres Aufrufen der Funktion erzeugt hat. Falls nicht, rufen wir den **lensflare_create()**-Code jetzt.

```
        if(qlensflare == -1) // erzeuge die Linsenreflexionen
        {
                lensflare_create();
        }
```

Dann prüfen wir, ob die Linsenreflexion bereits aktiv ist. Bei diesem Code hat es nur Sinn eine Linsenreflexion auf einmal ablaufen zu lassen. Daher kehren wir wieder zu dem Punkt hier zurück, wenn sie bereits aktiv ist.

```
        if(qlensflare == 1)  // Linsenreflexion ist bereits gestartet
        {
                return;
        }
```

Wir geben dem **lensflare_create()-**Code Zeit zur Initialisierung und setzen die Variable **glensflare** um anzuzeigen, dass die Linsenreflexion läuft.

```
        wait(1);  // Zeit zur Initialisierung für "lensflare_create"
        qlensflare = 1;  // markiere Linsenreflexion als 'an'
```

Die Haupt-While-Schleife für unseren Effekt läuft pro Framezyklus einmal durch und zwar solange, bis der **qlensflare-**Wert zu etwas anderem, als **1** verändert wird.

```
        // plziere Linsenreflexion
        while(qlensflare == 1)
        {
        // Animiere Linsenreflexion
```

Mit Hilfe des **vec_to_screen()**-Befehls überprüfen wir kurz, ob die Lichtquelle im Blick erscheint (wir müssen erst eine Kopie davon machen, also ändern wir die Werte nicht). Ist der Punkt nicht im Sichtfeld der Kamera, verstecken wir die Licht-Sprites mittels unserer Hilfsfunktion **flare_visible(off)**.

```
        vec_set(temp,flare_sun_pos);
        if(vec_to_screen(temp,camera) == 0)
        {
        // Ausserhalb des Sichtfeldes... entferne Linsenreflexion
        flare_visible(off);
        }
```

```
        else     // check for trace to sun
        {
```

Wenn die Lichtquelle den **vec_to_screen()**-Test durchläuft, können wir über den t**race()**-Befehl herausfinden, ob sie von der gegenwärtigen Kameraposition aus sichtbar ist. Wie ich schon früher erwähnte, ist es wichtig, dass **flare_sun_pos** an einer Stelle plaziert wurde, an der die Kamera diese Position wenigstens zeitweise "tracen" kann. Im Falle der diesem Workshop beigefügten Test-Map, wird die Welt von einem passablen Himmels-Block umgeben. Daher kann, obwohl **flare_sun_pos** sehr weit weg positioniert wurde, von allen sonnenbeschienenen Stellen der Map aus "getraced", also die Verbindungslinie zur Sonne hergestellt und vermessen werden. Es könnte sein, dass Sie den **flare_sun_pos**-Wert oder den **flare_trace_mode** anders einstellen müssen, damit das auch in Ihrem Level funktioniert.

```
        // 'tracen' zur Sonne
        trace_mode = flare_trace_mode;
        // Prüfen, ob die Verbindungslinie zur Sonne blockiert ist
        if (trace(camera.x,flare_sun_pos) != 0)
        {
```

Wenn irgendetwas die Verbindung zwischen Kamera und Sonne unterbricht, verstecken wir, die Licht-Sprites, wie bereits zuvor, mit unserer Hilfsfunktion **flare_visible(off).**

```
        // Etwas blockiert uns.. unterbinde die Linsenreflexion
        flare_visible(off);
        }
        else
        {
```

Wenn wir in unserem Code bis hierher vorgedrungen sind, schauen wir ins Sonnenlicht und werden durch nichts blockiert. Wir benutzen die XY-Bildschirmposition der Sonne (wird vom früheren Aufruf von **vec_to_screen()** berechnet) und gleichen sie durch die Hälfte der Bildschirmgrösse und -Breite aus, um den aktuellen XY-Abstand des Bildschirms von dessen Mitte zu bekommen.

```
        // temp beinhaltet jetzt die XY-Position der Sonne auf dem Bildschirm
        // zieht die Bildschirmmitte ab, wird für flare_place() gebraucht
        temp.x -= 0.5 * screen_size.x;
        temp.y -= 0.5 * screen_size.y;
```

Das sind die **temp**-Werte, die von der Hilfsfunktion **flare_place()**zum plazieren der 8 Licht- und des Sonnen-Sprites benutzt werden.

```
        // plaziere die Lichter gemäss Abweichung und ihrer Distanz zum Angelpunkt
        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);
        }
}
```

Warte einen Framezyklus, ehe du das Ganze von neuem wiederholst.

```
        wait(1);        // animiere jeden Zyklus
}
```

Diesen Teil des Codes erreichen wir nur dann, wenn **qlensflare** durch irgendetwas auf einen anderen Wert als **1** verändert wurde. In diesem Fall sollten wir sicher gehen, dass die Linsenreflexionen versteckt sind (dazu rufen wir dei Funktion**flare_visible()**) und setzen zum Zeichen, dass wir fertig sind, **qlensflare** auf den Aus-Wert (**0**).,

```
        // Nehme die Linsenreflexion weg
        flare_visible(off);
        qlensflare = 0;     // markiere die Linsenreflexion als 'aus'.
}
```

Diese letzte Funktion ist die einfachste. Um die Linsenreflexion abzuschalten, brauchen wir den Wert von **qlenflare** nur auf einen anderen, als **1** setzen. Setzen wir den Wert hier auf **.5**, wird die Haupt-While-Shleife in **lensflare_start()**beendet.

```
        // Beschr: stoppe den Reflexionseffekt
        function lensflare_stop()
        {
        qlensflare = .5; // Signal zum beenden
        }
```

Wie benutzt man den Code?

Nun, da wir den Code fertiggestellt haben, ist es wirklich einfach, ihn zum Einfügen von Linsenreflexionen in jeden beliebigen Level zu verwenden. Vergewissern Sie sich, dass Sie in **lflare.wdl** Ihrem Pfad haben und fügen Sie diese dann per **include** in Ihr Haupt-Skript (gleich nach den Template-Includes) ein. Um den Linsenreflexionseffekt zu starten, rufen Sie einfach **lensflare_start()** auf und wenn Sie ihn stoppen wollen, **lensflare_stop().** Falls Sie vorhaben, diesen Effekt zu einem anderen Zeitpunkt als dem Spielestart beginnen zu lassen, könnten Sie **lensflare_craete** in Ihrer Hauptfunktion aufrufen und damit ein Verlangsamen des Spieleablaufs, der durch das Setzen der Alpha-Werte der Licht-Sprites verursacht würde, vermeiden.

```
////////////////////////////////////////////////////////////////////////////////
// Linsenreflexions-Test-Level
////////////////////////////////////////////////////////////////////////////////
include <lflare.wdl>;// unser Lichtreflexions-Code

////////////////////////////////////////////////////////////////////////////////
// Die Engine startet in der Auflösung, die durch folgende Variablen
// angegeben wird.
var video_mode = 6;
var video_depth = 16;
////////////////////////////////////////////////////////////////////
// Die HAUPT-Funktion wird bei Spielstart aufgerufen
function main()
{
        load_level(<flr_test.wmb>);
        _move_straight();       // vordefinierte Kamerabewegung

        sky_clip = -65;

        lensflare_create();     // initialisiere die Linsenreflexion
        lensflare_start();      // starte die Linsenreflexion

        wait(2);
        // Starte mit Linseneffekt im Hintergrund
        camera.pan = 25;
        camera.tilt = 15;
}

on_2 = lensflare_start;
on_3 = lensflare_stop;
```

Zum Schluss

Ich hoffe, der erste Mini-Workshop hat Ihnen gefallen. Wenn Sie irgendwelche Fragen haben, Anregungen oder Verbesserungen zu diesem oder einem anderen Workshop beitragen wollen, posten Sie diese auf den Conitec User Forum!

Das fertige Level in Aktion