

Light Flare Shader for rFactor

Tutorial & How-To

Created by **Siim Annuk (aka Some1)**

Version: **1.0**

Date: **03-09-2009**

Table of Contents

About this tutorial.....	3
About the light flare effect.....	4
Shaders in rFactor.....	5
Billboarding – the hard way.....	6
Prerequisites.....	8
Creating the light flare.....	10
Conclusion.....	12

About this tutorial

This tutorial explains how the light flare shader that can be seen in the rFactor vehicle mod **“Toyota Supra MKIV by Siim Annuk (aka Some1) and Niels Heusinkveld”** works and how to use it. The reader is required to have a basic understanding of 3D graphics and concepts as well as knowing simple trigonometry mathematics. This tutorial is geared towards 3D Studio Max users (version 7 and up), but might be applied to other 3D modeling software as well, provided that it has support for rFactor specific modding tools (shaders, exporters, etc). And of course, a working copy of rFactor is also needed :)

Please keep in mind, that I'm not a graphics professional, everything I know, I have learned by trial and error and via the internet. Thus, this tutorial might contain errors or incorrect information. If you find anything that doesn't really apply, please let me know!

About the light flare effect

The light flare effect is something that happens when very bright objects, such as headlights, appear to glow and flash if viewed directly at them. In rFactor, so far most of the headlights of cars and tracklights have been created mostly using some polygons placed at different directions and mapped with a light flare texture. While this mostly gets the job done, it looks unnatural and outdated for today's computer graphics standards.

The idea behind the light flare effect I have created is simple – make a quad (that is a square consisting of two triangles) face the viewer (camera) at every viewing angle and change its size according to the angle between the quad surface normal and the viewer direction. This means that for example, if the quad is facing forward (headlight), it is visible from the front and not visible from the back. This kind of effect has been used for ages in most games (Need for Speed series, Gran Turismo series etc) but unfortunately left out from rFactor.



Light flare effect in action

Because of certain limitations of the GMotor Engine (more on that later), the light flare shader is perhaps not the most optimized and **works only in DirectX 9** mode of rFactor (but hey, its 2009 and a true simmer has a good PC anyway!).

Shaders in rFactor

rFactor GMotor Engine uses DirectX vertex and pixel shaders to render geometric objects and materials on screen. These shaders can be found in **“GameData/Shared/coreshaders.mas”** file. A mas file is a (compressed) collection of files much like a zip file, and it makes loading of game resources faster and helps to conserve disk space. To operate with mas files, you need the MAS utility found in the rFactor mod development pack, which can be downloaded directly from the official rFactor website (www.rfactor.net), just look under the **“Dev Corner”**.

In case you are new to shaders, a vertex shader is a piece of code that is run for each vertex of the geometry it is applied on and a pixel shader is code that is run for each pixel in the screen that is rendered for a triangle. Actually, it is far more complicated than that and I haven't figured it all out myself either :)

Billboarding – the hard way

As previously mentioned, the idea behind the Light Flare effect is that a square quad (that is, an object consisting of two triangles) is rendered always facing the camera (this is called billboarding) and changing its size based on the angle between the camera and the light.

While rFactor 3DS Max plugin provides an option to export an object as 'billboard', it didn't fit the needs for the light flare effect. So I had to create a new shader that makes the objects it is applied to face the camera and change its size. A very important thing to keep in mind is that this shader must only be applied to objects consisting of quads, that is, the object should contain $n * 2$ triangles and $n * 4$ vertices, where n is the number of quads.

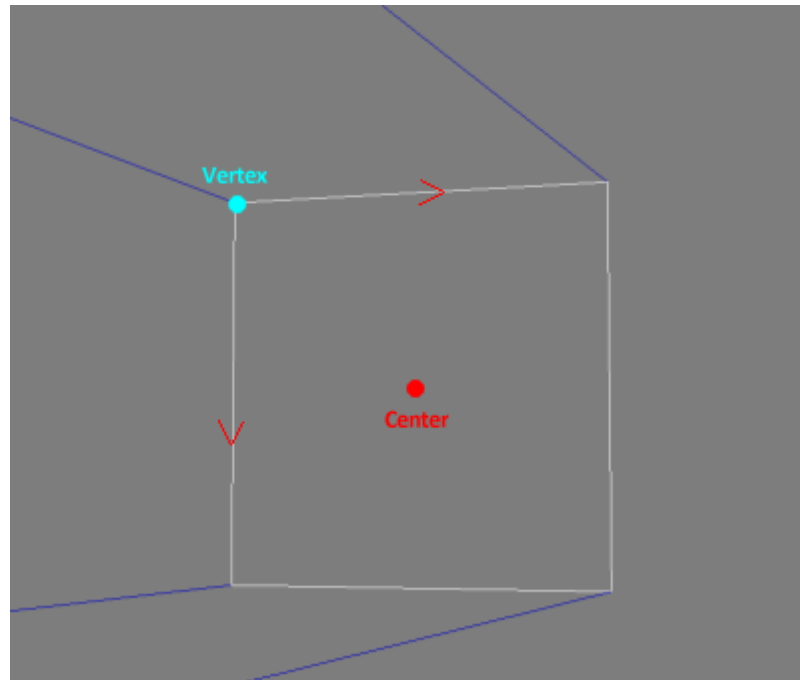
GMotor graphics engine doesn't provide a lot of ground to create a billboard shader. The assumptions I made for a quad is that it consists of 4 vertices, is completely flat and has 4 normals, and – **the most important thing** – the size of the quad is **1 unit**.

Most of the magic is done in the vertex shader (VS20_LIGHTFLARE.VSH). A vertex shader is a c-like program that is run for each vertex of the object this shader is applied to. One cannot reference other object's vertices from a vertex shader program, all that is known is the current vertex's position and normal and a bunch of other parameters.

Firstly we have to find out, which vertex we are dealing with. A vertex has coordinates in space (vertex coordinates, 3d) and coordinates in the texture (texture coordinates, 2d). DirectX texture coordinates are usually in the range of (0,0)...(1,1), where (0,0) is upper left corner of the texture and (1,1) is lower right corner of the texture. By knowing this we can identify, which corner a quad's vertex is. So, by subtracting 0.5 from the x and y texture coordinates we know that if the result (for x and y) is smaller than 0, the vertex is in the upper left corner etc.

When creating the billboard shader, the first difficult task I faced was to find out the center position of the quad. This is needed by the billboarding operations that we cover later. So, how do we find out the center of a quad, when we only know the vertex position, the corner it identifies and the normal of the vertex? Answer is simple trigonometry math.

Consider the following illustration:



The white lines are the borders of the quad, red spot marks the center of the quad we want to find out, the cyan spot marks the current vertex, the blue lines are each vertex's normals. Because we know the size of the quad, we just have to travel half of the size from the vertex position to inwards. It's a little hard to explain, but imagine how you would go to the center of a square from one corner. Because the quad is in 3D space and facing whichever way imaginable, simple add/subtract math of the vertex position doesn't work. Instead, we have to build two new vectors (having the same length as the size of the quad) that travel from the vertex position along the sides of the quad. Then we just multiply the position of the vertex with half of the vectors and we have the center.

Next task was to get the vertices to face the camera at all times. I found a good example from the ATI RenderMonkey samples package. To make the quad face the camera, we need to know the WorldViewMatrix, which can be calculated by multiplying ViewMatrix and WorldMatrix (these are known constants in rFactor vertex shaders). Now, we just do a multiplication and addition:

```
float3 pos = float3(0.0, 0.0, 0.0);  
pos += (particleSize * tx) * worldviewMatrix[0] + (particleSize * -ty) * worldviewMatrix[1];  
pos += particlePosition;
```

First we make the quad (the vertex) face the camera at the center of the space and then translate it to the particle position we previously calculated. Particle size is calculated according to the angle between the vertex normal and the camera. When the camera faces away from the quad, the size becomes 0 and the light flare disappears.

Prerequisites

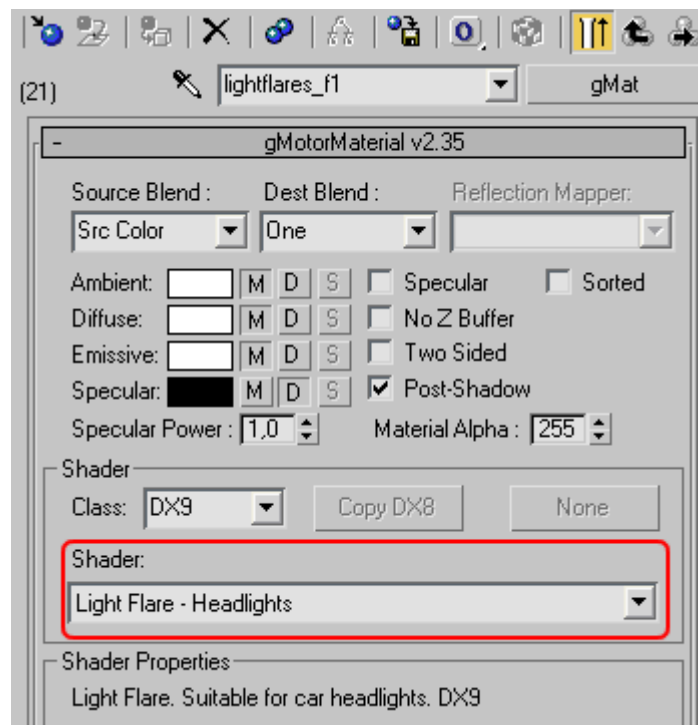
In order to create the light flare effect for your vehicle or track or whatever you are making, you need the following software installed and ready to use:

- 1) rFactor (latest version)
- 2) rFactor mod development pack
- 3) 3D Studio Max 7 or newer

I won't cover here the details on how to install the mod development pack (and gMotor tools for Max) and how to use it, because if you are new to it, the rest of the tutorial won't probably be very easy to understand (plus, my tutorial writing skills are not very good).

Because I have created new shaders, they must be installed for rFactor and 3DS Max in order to use them. rFactor looks for its shaders in the **“GameData/Shared/coreshaders.mas”** file and the folder **“GameData/Shared”** (this by the way, can be overridden in the mod rFm file). To install the new shaders, just extract the files included in the Shaders.zip file to rFactor's **“GameData/Shared”** folder and 3DS Max's **“HardwareShaders”** folder (just as described in the mod development pack installation guide).

You now should be able to use the light flare shaders. To verify this, open up 3DS Max, go to Material Editor, create a GMotor Material and see, if you can select **“Light Flare – Headlights”** for the DX9 shader:



Once you have verified that the shader is available, you have successfully installed the shader and are ready to create light flares. There are currently two (2) shaders for Light Flares available: “Light Flare” and “Light Flare – Headlights”. The first one is suitable for more general glow lights, for example rear lights and track lights. It simply changes its size according to the view angle. The other one is more suitable for car headlights, because it features a “sudden flash” when viewed straight to the light. Better explanation can be seen on the Toyota Supra mod, it uses the “Light Flare – Headlights” shader for the headlights (duh!) and “Light Flare” shaders for the rear lights.

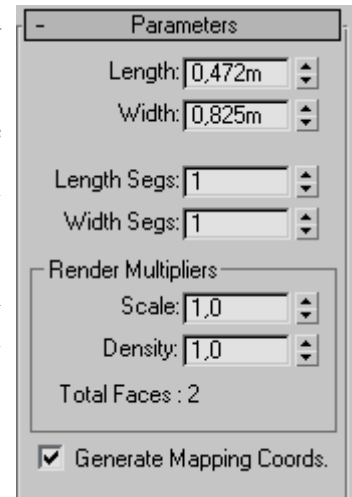
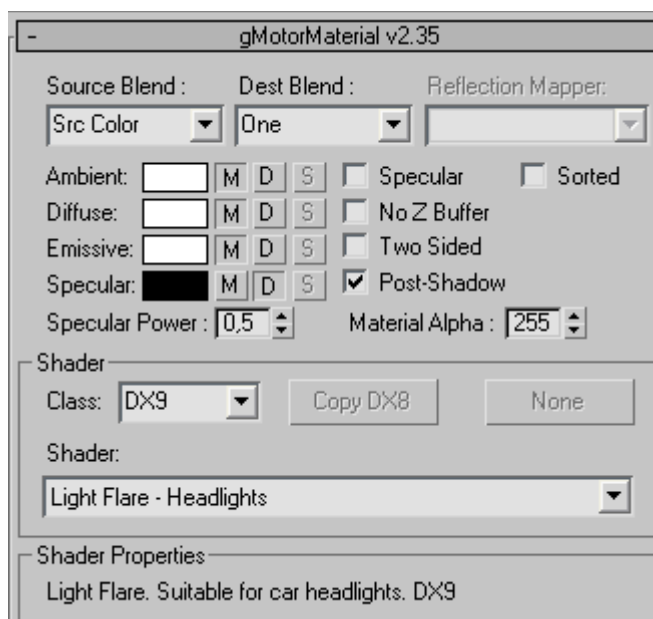
Creating the light flare

In case you skipped right here without reading the previous mumbo-jumbo about the inner workings of the shader, here are some things you need to consider when creating the light flare:

- the quad that represents the light flare has to be a square with the size of **1 unit!**
- the quad may be rotated only around Z or X axis (in 3DS Max)!

So, the first step is to create a quad that is 1 unit in width and length. Easiest way to do this is to select “Create → Standard Primitives → Plane”, and drag a plane on the “Front Viewport”. In the plane parameters, set length and width to 1 and segments to 1 as illustrated on the image. After that, convert the plane to an Editable Poly.

Next step is to create the material for the light flare. Open Material Editor, create a new Multi-Sub material and then create a new gMotorMaterial under that. I suggest the following settings for the material:



Using blendmode “Src Color : One” gives the best results and saves from using an alpha channel for the flare texture transparency. Although the size of the quad must always be 1 unit, it is sometimes necessary to have different size flares, for that purpose the specular power will be used to change the size of the quad by multiplying. So, if the specular power is 0.5 the size of the quad in game will be 0.5 units (because $0.5 * 1 = 0.5$). Also, you can use the emission to change the color of the quad. Now, that you have the material, assign it to the quad.

Place the quad wherever you like in your car or track and rotate it if you want it to be visible from front or back or from the sides. You can attach multiple flares to one object and this is a good thing, because rFactor only supports one file for each light.

Export the quad to GMT via the gMotor Tools for 3DS Max as always, fire up rFactor and you should now have light flares.

Conclusion

Light flares really give lights the feeling that they emit light and are bright. They add quite a lot of visual realism and are really easy to implement in a game. Too bad that rFactor and many other sims still don't feature such a primitive yet immersive effect.

This tutorial is still lacking in quite a few places and if you didn't succeed in creating your first light flare, let me know, what went wrong or what have I missed in this tutorial.