JustF Lighting Engine

Introduction

I found lighting and shadows in game maker a very lacking feature. So I decided to write my own lighting engine for game maker. In this (rather long~) tutorial I will explain how this engine works and how you could make it yourself and later I will explain how to use my engine.

> *NOTE: there are many variations on how to implement real-time lighting. I will just explain ONE of them here.*

I hope that they will implement something similar to this in the feature because that would be very neat. Hint developeres ;)

> *NOTE: I've made a lighting engine before. It was pretty popular on the game maker forums and is still out there. It is outdated however and slower than this one. This new one is better structured and can be implemented much more easily.*

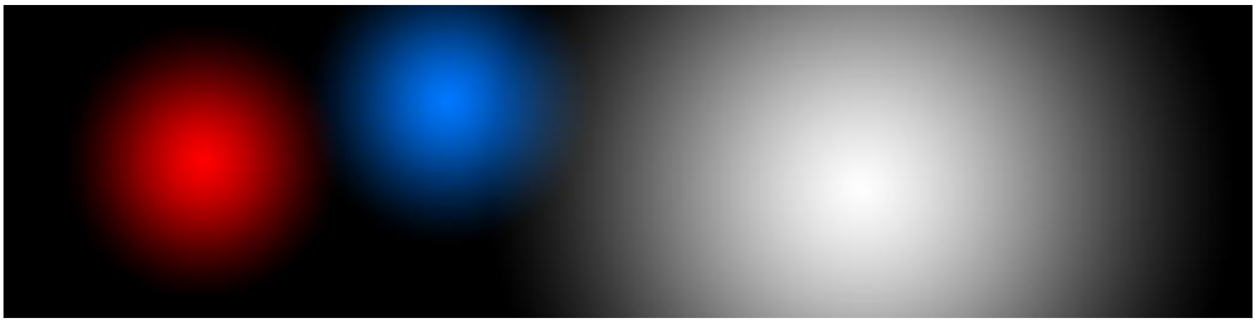
Basic Structure

In this implementation of lighting into your game you start off with one off-screen buffer (the lightmap). Every time you want to update your lightmap you basically follow these simple steps.

First off you clear your lightmap to the ambient color. The ambient color is the light color everything will get when there is no light (in that particular spot). The ambient color is usually a gray, not fully dark (like black), but also not fully lit (like white) because then you wouldn't need lights right?

This is what your lightmap will probably look like right now:

After that you will draw your lights (in all their colors and glory) onto the lightmap.



As you can see, the lights blend nicely together. In real life colors add up (this is how your monitor works as well). If you blend a red light with a blue light and a green light:

$$red + green + blue = \begin{pmatrix}1\\0\\0\end{pmatrix} + \begin{pmatrix}0\\1\\0\end{pmatrix} + \begin{pmatrix}0\\0\\1\end{pmatrix} = \begin{pmatrix}1+0+0\\0+1+0\\0+0+1\end{pmatrix} = \begin{pmatrix}1\\1\\1\end{pmatrix}$$

To achieve this sort of blending we use addictive blending (instead of the normal alpha blending). For a more detailed guide on blend modes click here https://www.yoyogames.com/tech_blog/68 and here https://www.yoyogames.com/tech_blog/69

Now to really light up your scene all you need to do is draw the lightmap to the screen. However, we will use one more blend mode here: multiply. The multiply blend mode will cause every pixel to be multiplied with the lightmap's pixel. In this case for each dark spot it will be multiplied with black:

$$color * black = \begin{pmatrix}R\\G\\B\end{pmatrix} * \begin{pmatrix}0\\0\\0\end{pmatrix} = \begin{pmatrix}R*0\\G*0\\B*0\end{pmatrix} = \begin{pmatrix}0\\0\\0\end{pmatrix}$$

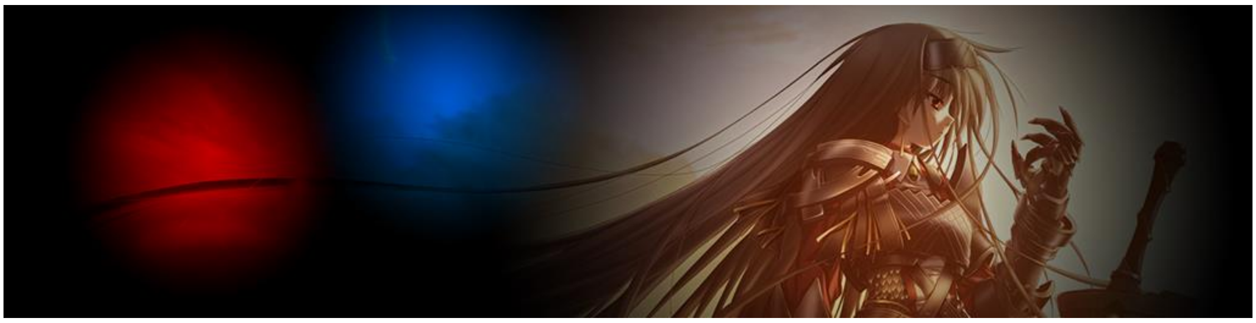
And for white

$$color * white = \begin{pmatrix}R\\G\\B\end{pmatrix} * \begin{pmatrix}1\\1\\1\end{pmatrix} = \begin{pmatrix}R*1\\G*1\\B*1\end{pmatrix} = \begin{pmatrix}R\\G\\B\end{pmatrix}$$

> *NOTE: these colors are in floating points, where 1 is the max value and 0 the min value (and everything in between is also permitted)*

So assuming your scene looks like $^{\text{(insert weeaboo joke)}}$

Now after drawing your lightmap (with multiply blending)

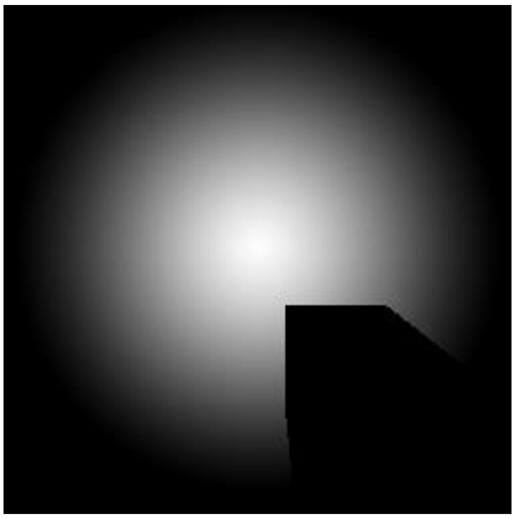


Wow! That looks neat! But, where are the shadows? Yeah, this is where everything gets a bit tricky, but stick with me, I will try to explain.

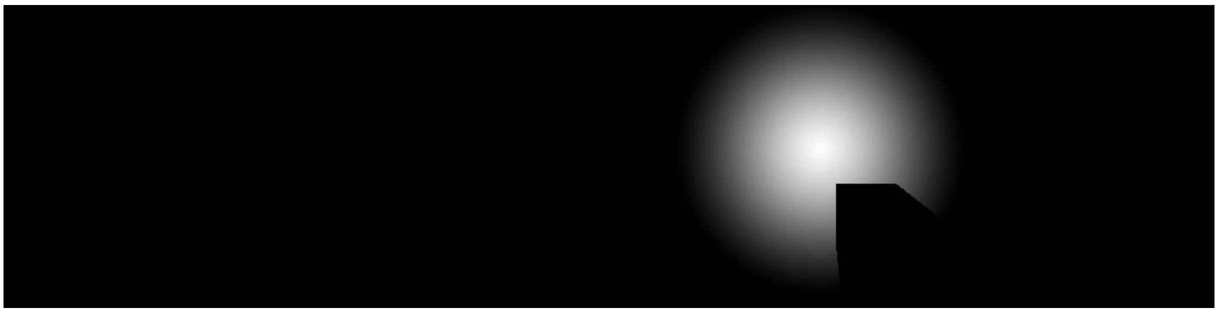
First off, we can't draw shadows directly to the lightmap. If we did we would get some very funky results (there are exceptions, you CAN do this, but you need to use a different type of light system).

To fix this, we introduce another buffer; the shadowmap (or shadow buffer). We first clear the shadowmap, then we draw the light in the same way we did with the lightmap; except the blend mode can be just normal. Then we draw the shadows your objects in your scene might cast onto this shadow buffer (I will explain how). Then we draw this later buffer to the lightmap, draw the rest of the lights in the same way and draw the lightmap to the screen the way we did before. As you might expect this buffer has the size of the largest light in your scene, if this light is very big it will drastically reduce performance.

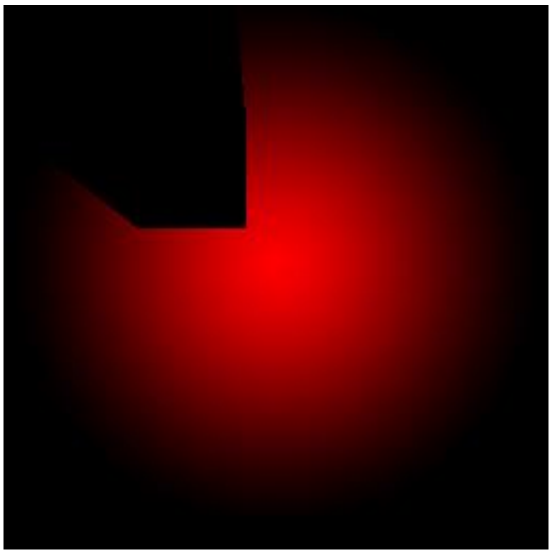
Example shadowmap for first light:



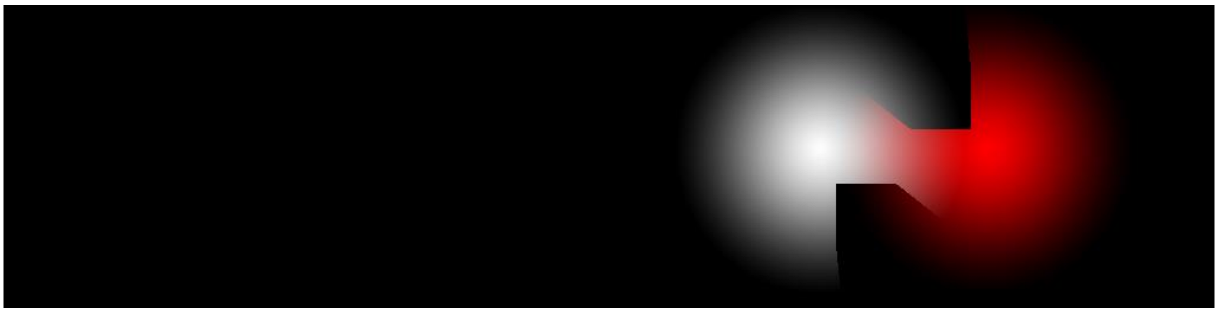
After drawing to the lightmap:



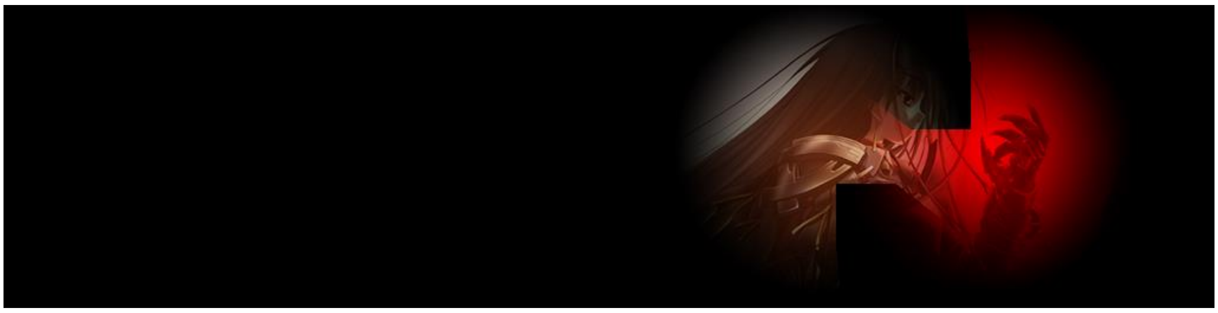
Example shadowmap for second light:



So after you have rendered all the lights onto your lightmap, your lightmap might look a little like this:



Then when you apply it to your scene it might look a little like this:



And there we go, our scene (with two lights, but not limited to) is done and lit up nicely.

Implementation

The implementation is made purely in game maker scripts. I will provide the download link somewhere on the game maker forums.

The scripts are provided with a lot of comments if you want to understand how the code works. If you want to use this engine in your game, you are completely free to do so. I would love to get some credits somewhere, but you have absolutely no obligations to. I do love to hear reactions and see what people make using this engine!

Furthermore, the scripts are organized in three categories:

- Lightmap
- Lights
- Casters

The lights speak for themselves, the light scripts are the scripts made for light objects. Caster scripts are for shadow casters, the objects that cast shadows. The lightmap scripts are for the lightmap itself, and all the settings surrounding it.

Basics

For the basic implementation you will need at least three objects (the names don't matter though!):

- A lightmap object (usually something like `obj_lightmap`)
- A parent light object (usually something like `obj_light_parent`)
- A parent caster object (usually something like `obj_caster_parent`)

For each different type of light you need you make a new light object and make the parent the light parent object.

This is the same for each caster object.

In most cases (haven't ever seen one at least) you won't need more than one lightmap object.

> *NOTE: To get multiple lightmaps to work properly you probably need to change some of the code yourself as I've never used of tested it.*

Lightmap

In the lightmap object you need to at least initialize the lightmap (with a size) and tell the scripts where to find the caster parents and light parents. Furthermore in the create event you probably should add some more settings you want. A typical create event looks a little like this:

```
lightmap_init(room_width, room_height);

lightmap_set_light_parent_object(obj_light_parent);

lightmap_set_caster_parent_object(obj_caster_parent);
```

However, in bigger scenes your lightmap would get insanely big. This would be a bad idea in many cases. The easiest fix for this problem is:

`lightmap_init(view_wview,view_hview);`

`lightmap_set_light_parent_object(obj_light_parent);`

`lightmap_set_caster_parent_object(obj_caster_parent);`

`lightmap_move_with_view(true);`

Note that we added the move with view to the code. This tells the lightmap to draw to the view location, instead of the room location. (Try experimenting with this, it will start to make more sense then)

Great! Our lightmap setup is done. There is just one thing left. In the draw event we need to update and draw the lightmap.

```
lightmap_update();

lightmap_draw();
```

Updating the lightmap will go through all of the lights and casters and start redrawing them onto the lightmap. The lightmap draw function actually draws the lightmap to the screen.

> *NOTE: You cannot update the lightmap in the step event because game maker forbids you to do drawing operations in the step event.*

Thing to keep in mind, the shadow buffer is only so big (by default 256). If you want to resize the shadow map you call:

```
lightmap_set_shadow_size(new_size);
```

That's it! The lightmap is done.

Lights

Lights are pretty straight forward. Any light object must have the light parent (as setup in the create event for the lightmap) as their parent. In the create event for each light you must decide what type of light it is.

You can either make this light a point light (a circular light in a particular color) by calling light init point, with the arguments radius and color.

```
light_init_point(256,c_white);
```

Or you can make it a sprite light. The sprite used for the light is also drawn with add blend mode (see the basic structure), so everything that is black won't be drawn. To init a sprite light you call light init sprite with arguments sprite_index and sprite_subimage

```
light_init_sprite(spr_light,0);
```

And that's all for lights, your light is done and will be draw to the lightmap by the lightmap object.

Casters

Finally there are casters. Casters, as said before, are the objects that cast the shadows. Casters can be initialized in a few different ways. From simple to most advanced:

```
caster_init_sprite();
```

This code initializes a caster as a square around the sprite. It is really straight forward and for most cases this will be sufficient.

> *NOTE: since casters are polygons and made out of just a few points, there is no such thing as caster rotations!*

Furthermore there is the caster init rectangle. This inializes a caster given the 4 values in this order: left, top, right, bottom.

```
 caster_init_rectangle(left, top, right, bottom);
```

Then there is the circle, given a radius and the amount of subdivisions (since it will be made out of triangles, most of the times around 10-20 is sufficient)

```
caster_init_circle(radius, subdivisions)
```

And finally there is one last way to create your own, custom shaped caster

```
caster_init_polygon();
```

This gives you a blank caster with no points. You can add points (in counter clockwise order) to the caster by calling

```
caster_add_point(x, y);
```

> *NOTE: both x and y are relative to the caster objects x and y position*

Rest

The rest of all the scripts are provided with a lot of comments and most of them kinda speak for themselves. If you still have questions about them you can always ask them on the forum post, even though I don't read them that often. There are loads of people wanting to help.

Footnote

Thank you for reading my tutorial on lighting. Remaking this tutorial cost me countless of hours and energy. I hope you liked reading this and that you learned something new. If you have feedback, feel free to