Game Development

GLSL Light (Attenuation, Color and intensity) formula

Asked 7 years, 7 months ago Active 6 years, 10 months ago Viewed 23k times

20

19

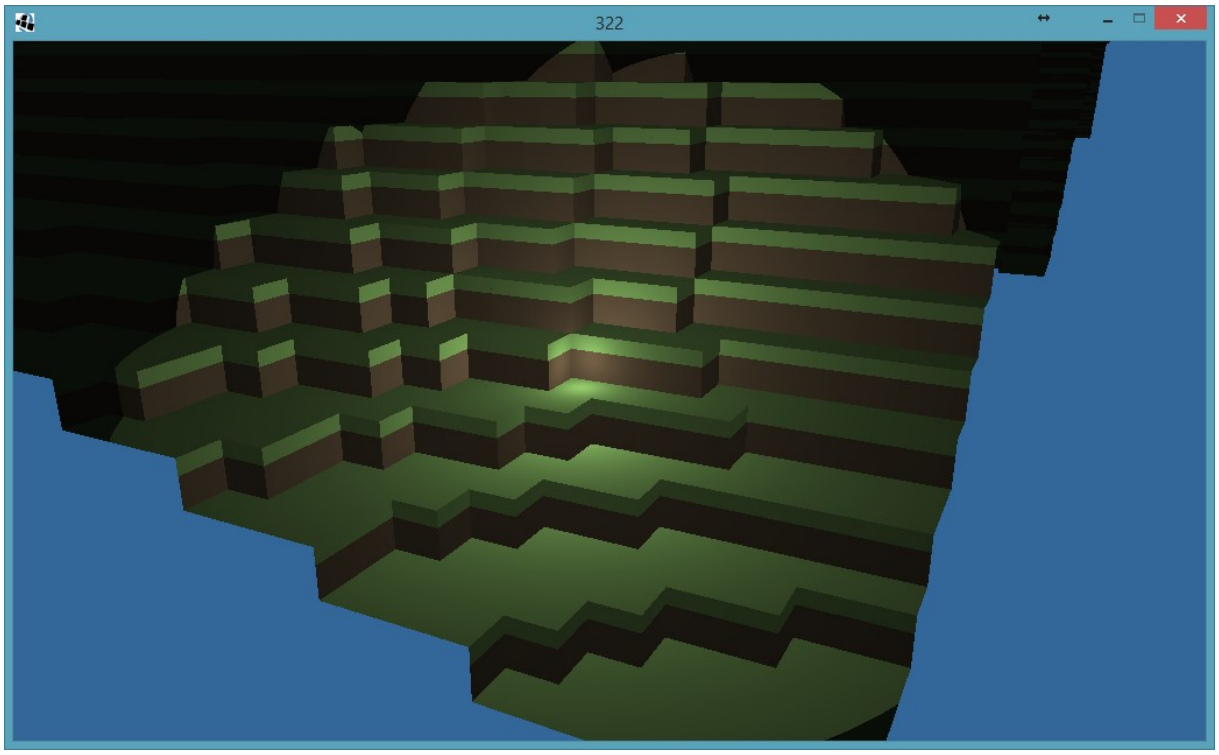
I'm implementing point lights in my Voxel engine, and I'm really struggling to get a good flow of light, from 100% near the light source to 0% at the light radius.

I have 5 arguments for the function:

1. Light color (Vec3)
2. Light intensity (distance from the light till the distance where falloff is 100%)
3. Distance from the light to the fragment
4. The angle from the fragment normal to the light
5. The position of the light

Can anyone push me in the right direction to create a function for the calculating of the fragment color?

Image of one of my experiments:



Edit (current code requested by Byte) **Note that this is just some experiment code from my side. I got the float att from a website, and it kinds works, but far from perfect.**:

```
void main()
{
// Light color
vec3 torchColor = vec3(1.0f, 1.0f, 1.0f);

float lightAdd = 0.0f;
for (int i=0; i<5; i++) {
    vec3 pos = lights[i];
    if (pos.x == 0.0f) continue;
```

6 You're still saying things like "struggling to get a **good looking**, **natural lights**" and " works, but **far from perfect**". You need to include specific, exact language. We don't know what good looking is to you, or what natural lights look like to you, or what perfect is. – MichaelHouse ♦ Jun 6 '13 at 1:56

2 Have you tried removing `if (dist < 9)` ? Alternatively you could calculate `att` with a function that returns 1 when distance is 0 and 0 when distance is 9. E.g. `mix(1.0, 0.0, dist / 9.0)` – msell Jun 6 '13 at 5:32

1 Answer

Active | Oldest | Votes

43

The attenuation function you've got,

```
att = 1.0 / (1.0 + 0.1*dist + 0.01*dist*dist)
```

is a fairly common one in computer graphics - or, more generally, `1.0 / (1.0 + a*dist + b*dist*dist))` for some tweakable parameters `a` and `b`. To understand how this curve works it's helpful to play with the parameters interactively. This curve is nice because it approaches the physically-correct inverse square law at large distances, but it doesn't shoot up to infinity at short distances. In fact, with `a = 0` it's a pretty good model of a spherical area light.

However, one disadvantage of it is that the light never quite goes to zero at any finite distance. For practical purposes in realtime CG we generally need to cut lights off at a finite distance, as you're doing with the `if (dist < 9)` clause. However, the radius of 9 is too short - with your settings in the attenuation function, the light doesn't get close to zero until dist is around 100.

You can calculate the radius of the light from the `b` parameter in the attenuation function (since the quadratic term dominates at large distances). Let's say you want to cut the light off when the attenuation reaches some value `minLight`, like 0.01. Then set

```
radius = sqrt(1.0 / (b * minLight))
```

That gives a radius of 100 for `b = 0.01` and `minLight = 0.01`. Alternatively, you can set the radius and calculate `b` to match:

```
b = 1.0 / (radius*radius * minLight)
```

For `radius = 9` and `minLight = 0.01`, that gives `b = 1.23`. You can set it up either way, but the key is to make the radius and the attenuation function match so that you don't cut the light off until the attenuation function is already very low, so you won't see a sharp edge.

All that said, there are alternative attenuation functions you can use. Another fairly common one is:

```
att = clamp(1.0 - dist/radius, 0.0, 1.0); att *= att
```

or the slightly fancier:

```
att = clamp(1.0 - dist*dist/(radius*radius), 0.0, 1.0); att *= att
```

Great! Though, I'd use `max` over `clamp` for performance reasons only. – Mike Weir Jun 26 '17 at 16:36

4 @MikeWeir Clamping to [0, 1] is free on many GPUs, actually. It's such a common operation that they have it as an instruction modifier. – Nathan Reed Jun 26 '17 at 18:58