Mike Horwath (Povray.advanced-users: 31-08-2015):
Do light sources in POV 3.7 use srgb or rgb? Do the colors need to be adjusted to the new system?

Clipka:

> ***Alain:***
> *You should use assumed_gamma 1 in your global_settings section and set display_gamma 2.2 in the master .ini file. Once it's done, you should never need to perform that kind of gamma "magic" in your scenes.*

Personally, I'd recommend trying to calibrate the display for sRGB and use "Display_Gamma=srgb". At any rate, the setting should of course match the display system's effective gamma, so a generic recommendation of setting Display_Gamma to 2.2 falls a bit short.

The question is a bit poorly worded, so instead of trying to guess your own understanding of that question, I'll try to provide you with information that should enable you to answer it yourself:

Light source colours in all versions of POV-Ray are specified in the very same manner as any other colours in that particular version.

Now first, let's look at what this means for POV-Ray 3.6:

* By default, POV-Ray 3.6 did not do any gamma handling; this can be interpreted in two ways:

(a) According to the design and implementation, and although this was certainly unintentional, POV-Ray 3.6 interpreted all colour expressions as colour coordinates in a linear RGB colour space, expected all input images to be in that same colour space, and generated output images in that same colour space. In this sense, POV-Ray 3.6 did everything right, but colour in- and output were in an uncommon format, a fact that was entirely unknown to most (if not all) users; as a result, users would create scenes that did not match what they wanted, and got results they wouldn't have wanted but, when misinterpreted in the most typical way, "happened" to look pretty much exactly like what they wanted.

(b) According to the users' experience, it would seem that POV-Ray 3.6 interpreted all colour expressions as colour coordinates in an RGB colour space gamma pre-corrected for the user's display system, expected all input images to be in that same colour space, and generated output images in that same colour space. In this sense, POV-Ray 3.6's colour handling matched user expectations, but (again entirely unknown to most users) it got a variety of colour computations outright wrong.

Most (if not all) legacy scenes (POV-Ray 3.6 scenes without explicit "assumed_gamma") would be written specifying all colours in sRGB coordinates (or something very close), but using the "rgb" family of keywords to specify them (the "srgb" keyword wasn't available back then).

* Alternatively, POV-Ray 3.6 already supported the "assumed_gamma" keyword. Back then, its only effect was on file output, and was a way to tell POV-Ray whether the above interpretation (a) or (b) was desired (by setting assumed_gamma to either 1.0 or 2.2), but

forcing POV-Ray to generate output images with properly gamma pre-corrected colours in either case. (There was also the possibility to choose a value in between, or even larger or smaller, but I won't go into detail about that.)

Many "semi-legacy" scenes (POV-Ray 3.6 scenes with explicit "assumed_gamma") would be written specifying all colours in something somewhat close to sRGB coordinates ("assumed_gamma 2.2"), but using the "rgb" keyword to specify them, just like in legacy scenes; others would be written specifying all colours in linear RGB coordinates ("assumed_gamma 1.0"), also using the "rgb" keyword to specify those, much like modern scenes.

Now let's examine POV-Ray 3.7:

* If you specify "#version 3.6", POV-Ray 3.7's colour handling is virtually the same as that of POV-Ray 3.6.

* If you specify "#version 3.7", POV-Ray 3.7 defaults to "assumed_gamma 1.0", but other than that there is */still/* no fundamental difference to the behaviour of POV-Ray 3.6, except that you get some */extensions/* to complement the "assumed_gamma" mechanism:

- When using the "rgb" family of keywords, the colour expression's interpretation will */still/* depend on the assumed_gamma setting; using the "srgb" family of keywords, however, will cause the colour expression to */always/* be interpreted as a colour coordinate in sRGB space. (This feature is even available when "#version 3.6" is specified.)

- POV-Ray will no longer expect input image files to be in the same colour space as colours specified using the "rgb" family of keywords; instead, POV-Ray will do its best to auto-detect what gamma the image data is pre-corrected for, and convert the data accordingly. (There's also a syntax to override the auto-detection.)

It is highly recommended that modern (POV-Ray 3.7) scenes use "assumed_gamma 1.0", and use the "srgb" family of keywords wherever colour coordinates in sRGB space are known, while using the "rgb" family of keywords where it is necessary or desired to supply colour coordinates in linear colour space.

Mike Horwath (Povray.advanced-users: 18-11-2016):
I'm still sort of confused. For a brand new scene designed from scratch in 3.7 and not needing to migrate from 3.6, should I aim to use "srgb" or "rgb" in the light color definition?

Clipka:
You should be using "assumed_gamma 1.0" if you want physical realism.

Whether you're using `rgb` or `srgb` is irrelevant. You can either specify

```
rgb <R,G,B>
```

or

```
   srgb <Rp,Gp,Bp>
```

where Rp=f(R), Gp=f(G) and Bp=f(B), with f(x) being a special function defined in the sRGB colour space specification. The rendering results are the same either way.

In the former case you're specifying the light source colour and brightness in terms of linear light intensity values. In the latter case you're specifying the light source colour and brightness in terms of something closer to how the human eye perceives the values.

For example, you can either specify

```
   rgb <0.0, 0.214, 1.0>
```

or

```
   srgb <0.0, 0.5, 1.0>
```

without any difference in the render result. Whatever floats your boat.

There's a caveat though when performing mathematical operations in conjunction with `srgb`; for instance,

```
   rgb <0.0, 0.214, 1.0> * 100
```

is equivalent to

```
   rgb <0, 21.4, 100>
```

which is equivalent to

```
   srgb <0, 3.726, 7.133>
```

which is equivalent to

```
   srgb <0.0, 0.522, 1.0> * 7.133
```

and thus results in a different hue than

```
   srgb <0.0, 0.5, 1.0> * 7.133
```

so if you're using `srgb` notation you'll need to take care when trying to adjust the bightness of light sources without changing the hue; the following should work:

```
   (srgb <0.0, 0.5, 1.0>) * 100
```

So my personal recommendation would be to prefer `rgb` over `srgb`.

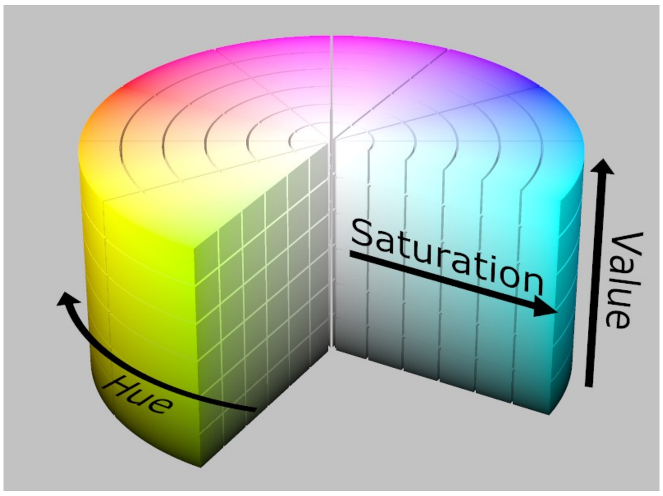
Gamma in version 3.7.1: the blend tools

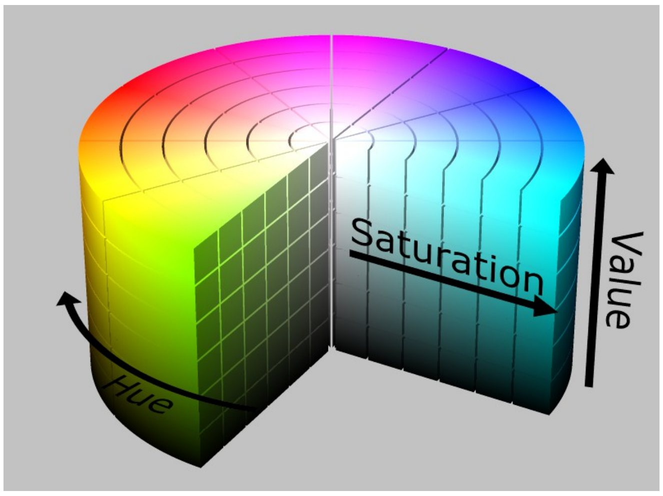
Mike Horwath (p.b.i. 27-12-2015):
I'm rendering the attached scene on Windows, but the default assumed_gamma value of 1.0 seems too light. Switching it to 2.2 is much better. Is there something wrong with my scene, because as I understand it, you're really not supposed to mess with assumed_gamma.

I also have these configured in the INI.

```
Display_Gamma=2.2
File_Gamma=2.2
```

[see pov file in Externals/Mike Horwath]





Clipka:
And that's particularly true when rendering images that are supposed to illustrate colour models.

As for how the image should properly look, that actually depends on the colour model you're trying to illustrate: If the "value" parameter is supposed to be linear, then it *must not appear* linear. If the parameter is instead supposed to be gamma-encoded, then approximate linear *appearance* is the right thing.

This is because when it comes to brightness, what *appears* linear to the eye actually isn't, and is actually close to a gamma of roughly 2.5.

Now while this would at first seem to imply that using assumed_gamma 2.2 would be the right thing to do if "value" is supposed to be gamma-encoded, there is one thing to be aware of: Blending *chromaticity* in this mode is pretty messy. Whether it would give the right results again would depend on the details of the colour model.

My suggestion would be to stick to assumed_gamma 1.0, and use POV-Ray 3.7.1's new colour_map/pigment_map **"blend_mode"** and **"blend_gamma"** parameters for more control over the colour gradients:

- Use **"blend_mode 1"** for entirely linear interpolation,
- **"blend_mode 2 blend_gamma 2.2"** for blending matching a gamma of 2.2, or
- **"blend_mode 3 blend_gamma 2.2"** to get gamma 2.2 interpolation of brightness but gamma 1.0 interpolation of chromaticity.

The defaults for these parameters are "blend_mode 0" (which performs gamma-agnostic blending, effectively matching the assumed_gamma setting) and "blend_gamma 2.5". Note that as opposed to most other gamma settings, "srgb" is not a legal value for "blend_gamma".

Kenneth (p.b.i. 28-12-2015):
This concept of chromaticity/brightness/gamma is completely new to me (as regards POV-Ray rendering); ditto the new 'blend' tools in 3.7.1. Clipka will (hopefully!) give us more info. A question I would have is this: Are the new blend tools specifically for creating accurate/specialized 'color models' like you're doing here (as well as for your Munsell color wheel render that you posted earlier)-- or are the tools meant to be used for our everyday, normal POV-Ray scenes as well?

Clipka:
Those colour and pigment map blend settings are indeed intended for everyday normal use. There are plenty of other ways to achieve what Mike may need, so I wouldn't have implemented the feature just for him.

Instead, I'd like to give credit to Warp, who sowed the seed for this feature back in the days when 3.7.0 was nearing completion, with his persistent refusal to accept that "assumed_gamma 1.0" is the holy grail of getting good images -- which of course it is, and his stubborn position was of course entirely unfounded... except, erm... well, not quite.

As you're new to the topic of gamma, let's give you a quick overview:

- Historically, computer graphics used cathode ray tube (CRT) displays. In a CRT the brightness of pixels on screen is controlled by modulating a particular voltage applied to the CRT, but the resulting brightness does not follow this voltage in a linear fashion, but according to a power-law, i.e. brightness = voltage^gamma, where gamma is typically somewhere around 1.8 to 2.2. To keep the display electronics and the computer's display interface simple, neither contained any mechanism to compensate for this distortion, so images intended to be presented on a computer display had to be */gamma corrected/* first to take that distortion into account. Usually this was already done during image creation, and the image saved in a */gamma pre-corrected/* format.

- This wasn't such a bad thing though, as the human visual perception is also highly non-linear; if we'd use an 8-bit linear brightness format, the precision would be far too low in the dark portions of an image (leading to visible colour banding), while being far too high in the bright portions of an image (leading to a waste of storage space); but it so happens that 8 bit provides just about enough precision in both dark and bright portions of */gamma pre-corrected/* images. Thus, this format also makes for an efficient encoding, and in modern file formats it is therefore typically referred to as */gamma encoding/*.

- In today's graphics industry, conceptually gamma pre-correction no longer plays any role, because correcting for the specific gamma of whatever display happens to be connected is instead done by the graphics application, operating systems, graphics driver and/or some calibration feature built into the display; gamma encoding still lives on however – even in the way colour information is input or presented in user interfaces for selecting particular colours.

- As a matter of fact, gamma pre-correction and later gamma encoding used to be so ubiquitous in the world of computer graphics, */and/* appeared to be so natural, that for a very long time people didn't even realize that it was there -- including the original authors of POV-Ray. This is a problem, because POV-Ray needs to do a lot of calculations with brightness values, and the formulae for those calculations were written under the silent (but wrong) presumption that colour information was encoded linearly.

- POV-Ray 3.6 was the first version of POV-Ray to ever tackle the issue of gamma, by introducing the "assumed_gamma" keyword and the "File_Gamma" (or was it "Display_Gamma"?) ini file setting. In this simple model, "assumed_gamma" told POV-Ray what gamma all the input colours were pre-corrected for, while ini file setting told POV-Ray what gamma to pre-correct all its output for (that is file output and preview display). However, this was essentially it -- POV-Ray still did all its internal computations as if all colours were linear, even if "assumed_gamma" was set to something else than 1.0. Also, setting "assumed_gamma" to 1.0 did solve this one problem, but it required all colours in the scene file to be specified using linear values and, worse yet, it meant all input image files were also assumed to use linear encoding.

- POV-Ray 3.7 solved (to my knowledge) virtually all the problems where the 3.6 gamma handling fell short: It introduced separate ini file settings for the two output paths (to file, and to preview display) so that output images could be pre-corrected for a different display gamma than the system it was previewed on; it introduced the "srgb" keyword so that colours in the scene file could be specified in a format familiar to most users and easily produced by colour picker tools; and it introduced a set of rules, as well as keywords to override them, to try to infer the gamma for which input images are pre-corrected for.

Except for the problems that Warp still saw. His annoying feedback had already inspired a good deal of the gamma improvements. Now he argued that "assumed_gamma 1.0" couldn't be right because gradients would look crappy.

He was wrong about "assumed_gamma 1.0"; it did a perfect job, and I tried to explain why it was right.

To no avail. He was adamant that they did look crappy, and that "assumed_gamma 2.2" did a much better job on them. Which, as I openly admit now, and silently admitted back then, was an entirely correct observation.

It wasn't the fault of "assumed_gamma 1.0" though, but rather the plain fact that for artistic purposes non-linear interpolation of gradients is desirable.

For brightness gradients, that is. When it comes to gradients between different colours with same brightness, using that same interpolation causes a dip in the brightness around the centre of the gradient.

I tried a whole bunch of different colour models to do the interpolation in, but none got both types of gradients right. I gave up, and 3.7 went out the door without a solution for this problem.

Later however, I found the time to revisit the issue, and found that good results */can/* be obtained; the key is to split up the colour information into brightness (i.e. the weighted sum of the colour channels) and chroma (i.e. the relative balance between the colour channels), interpolate the brightness non-linearly and the chroma linearly, and then re-combine the two properties into the new result colour.

That is exactly what "blend_mode 3" does: It was designed to give you the most aesthetically pleasing gradients you could ever ask for. "blend_mode 0" is only there for backward compatibility, "blend_mode 1" is there to do physically correct purely linear interpolation, and "blend_mode 2" is there just for completeness' sake.

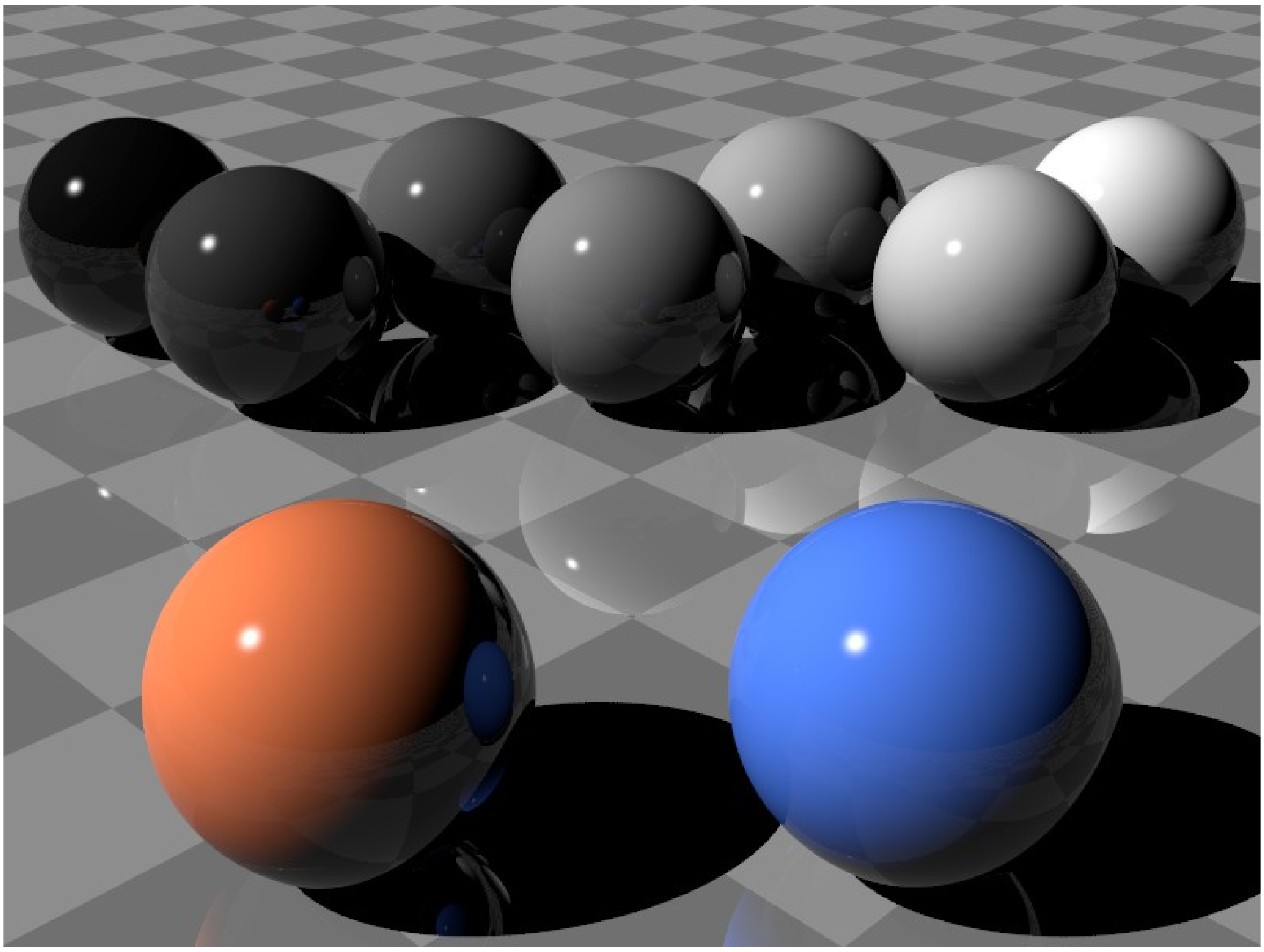
Gamma the smoking gun (clipka: povray.binaries.images, 21-12-2016)

Ever since I started working on gamma handling, I had been trying to concoct a scene that clearly and unambiguously demonstrates why anything other than "assumed_gamma 1.0" is doing it wrong. I */knew/* this was the case, but somehow it eluded all my attempts at nailing it down.

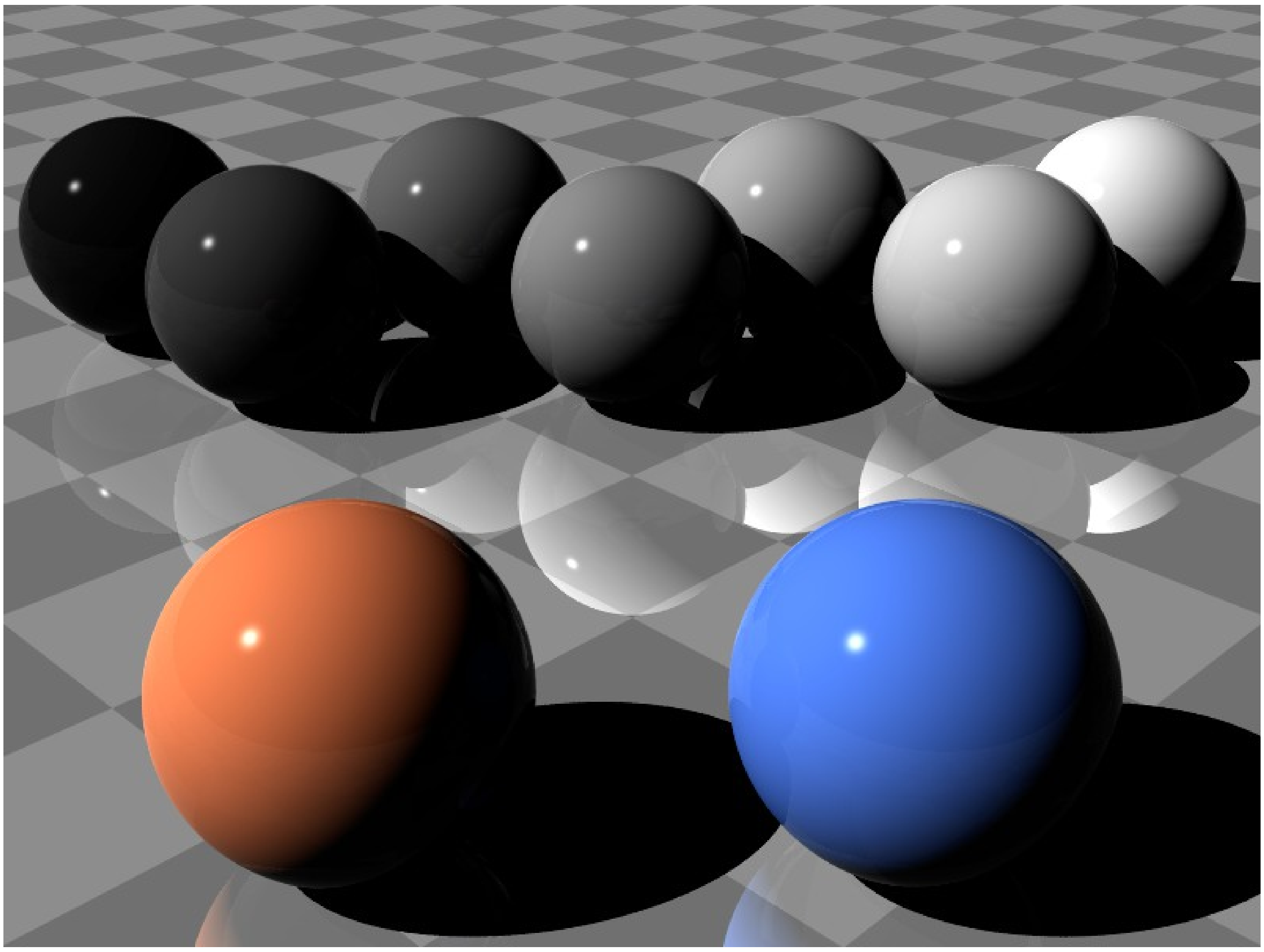
Until I stumbled across the description of the `exponent` reflection parameter in the docs: Here was a feature that probably had been invented for the sole purpose of working around a phenomenon that early POVers failed to understand, but which to me shouts "Gamma!" at the top of its lungs.

So inspired by that text, here it is now at last: The smoking gun of gamma handling.

The first image below shows the scene as it should be, rendered with `assumed_gamma 1.0` and proper colour math. What you see is a somewhat reflective chequered plane, a set of somewhat reflective spheres with highlights, and a black background. That's all, nothing else fancy in there. (I even turned off ambient and radiosity, as it would have made the next step far more difficult.)



The second image below shows essentially the same scene with `assumed_gamma 2.2`, with the diffuse settings (`diffuse` and `brilliance`) adjusted to get exactly the same diffuse effect out of the different colour math. Nothing else changed whatsoever.



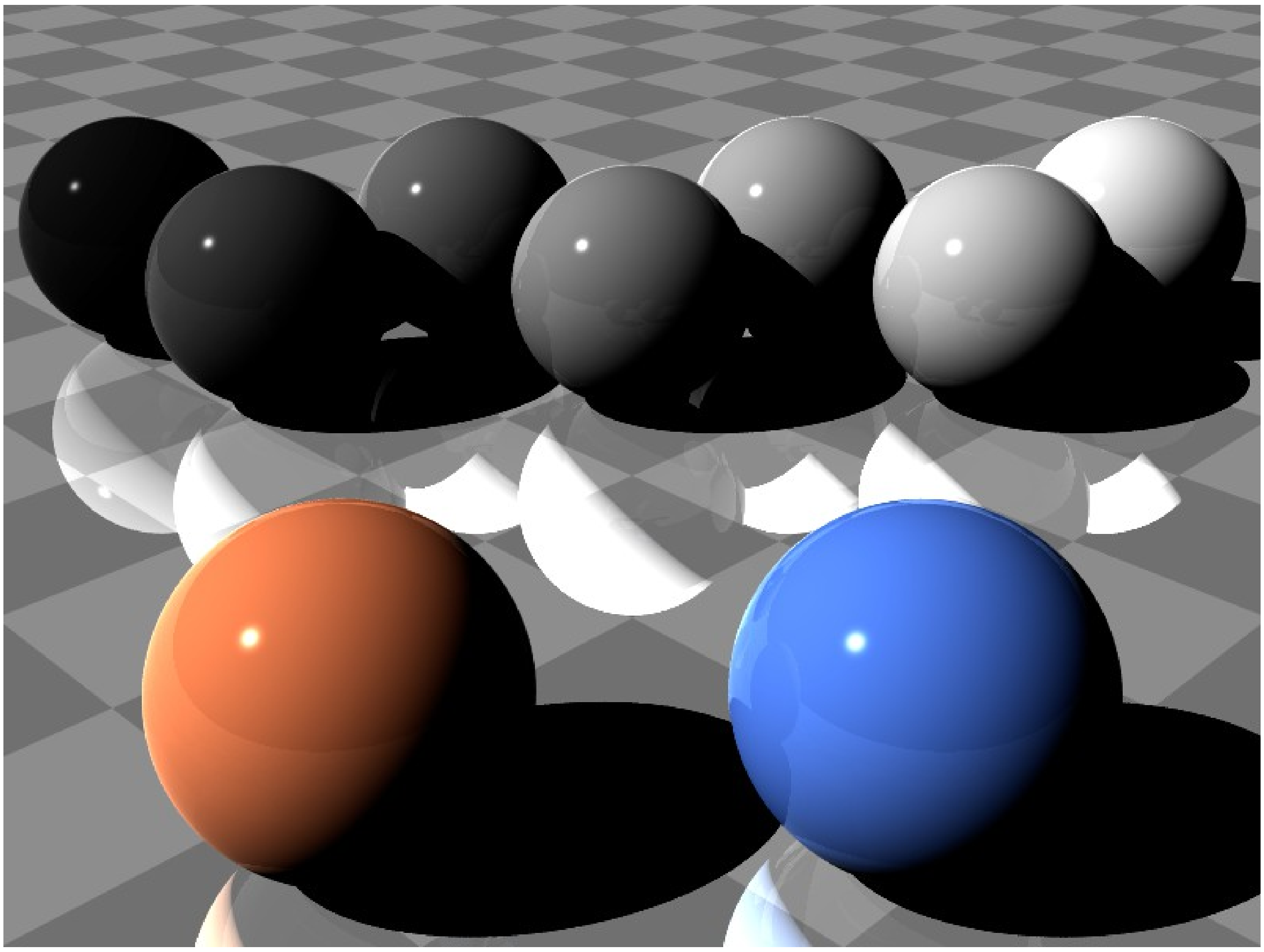
I'd like to draw your attention to the following details:

Exhibit A: The highlights. On the darker spheres they get dimmer, while staying at about the same brightness (actually even getting a bit brighter) on the bright spheres.

Exhibit B: The reflections. In the shadows and darker portions they seem to disappear almost */completely/*, while in the brightly lit portions they even get brighter.

Exhibit C: The terminators. In the reflections the transition between the illuminated and shadowed sides of the spheres become excessively sharp.

Just to scare your kids I've also attached a the third image below, rendered with "assumed_gamma 5.0" to exaggerate the effect. If the "assumed_gamma 2.2" image did not convince you, this one should: All the flaws visible here are due to non-linear colour math, and while they are a tad more subtle with a gamma of 2.2, they're still present.



I rest my case.

Dave Bandston (p.b.i. 21-12-2016):
Please forgive my ignorance, but does that mean the default settings for diffuse and brilliance are meant to give the most visually appealing results with an assumed_gamma of 2.2?

I never really gave assumed_gamma much thought. I just noticed that setting it to 1.0 produced a washed-out result so I picked 2.2 and never thought about it again.

Clipka (p.b.i. 21-12-2016):
Here are a few facts:

- The brilliance default of 1 fits */perfectly/* with `assumed_gamma 1.0`, because the developers back then naively implemented a formula that was designed for linear colours. The whole `brilliance` mechanism is an awfully hackish thing, and it so happens that it can be used to achieve the same proper look with other gamma settings (as far as diffuse goes), so my guess is that it was introduced specifically for the purpose of fixing the look of diffuse objects, in times when people probably didn't even know what gamma handling was.

- The diffuse default of 0.7 was presumably introduced in times when bad gamma handling was the norm, and it can be assumed that it was set in such a way as to get pleasing results in */that/* environment. In a gamma 1.0 scenario, that would correspond to a setting of about 0.45.

- As Warp demonstrated not long ago, one main reason (besides trying to use gamma-pre-corrected colours without the "srgb" keyword) for the washed-out look in gamma 1.0 mode seems to be the "ambient" default: That setting, too, was quite certainly designed for a gamma of about 2.2, and in a gamma 1.0 scenario that would correspond to an ambient setting of 0.006 (though that number is difficult to nail down, as ambient is always added to colours, and adding colours without proper gamma handling greatly distorts them, particularly if their absolute value is rather small.)

- Without proper gamma handling, there is stuff that you just simply */cannot/* get right simultaneously (as demonstrated with these images); so you may need a */lot/* of tweaking to get */somewhat/* close to a realistic look, and you'll have to do this */over and over again/* for virtually each and every scene, as you'll need to fine-tune your materials for the given lighting conditions and vice versa. On the other hand, with gamma 1.0 all it takes is some experience, and once you get your materials right you can re-use them quite easily in virtually every lighting condition. (Also, with proper gamma handling the number of knobs to tweak is smaller, since you never need to fiddle with any of those unrealistic hacks like brilliance, reflection exponent, or light source fade_power values other than 2.0.)