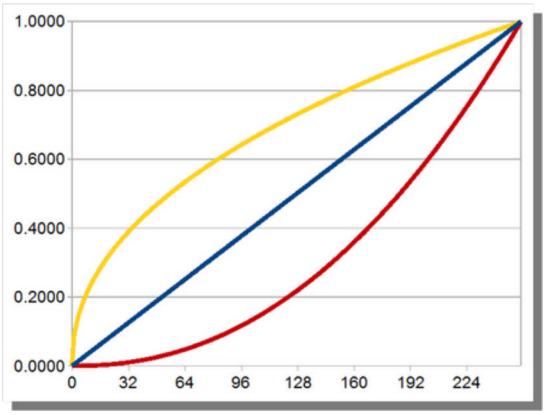
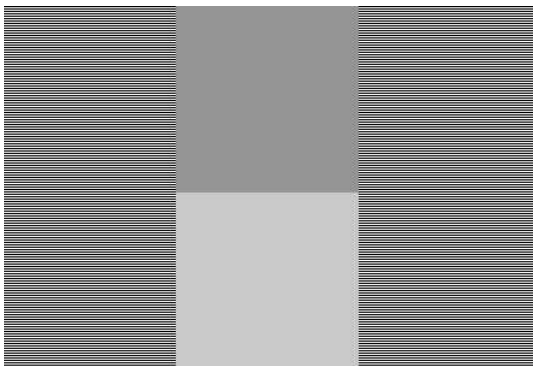
Linear-Space Lighting (i.e. Gamma)

June 21, 2010

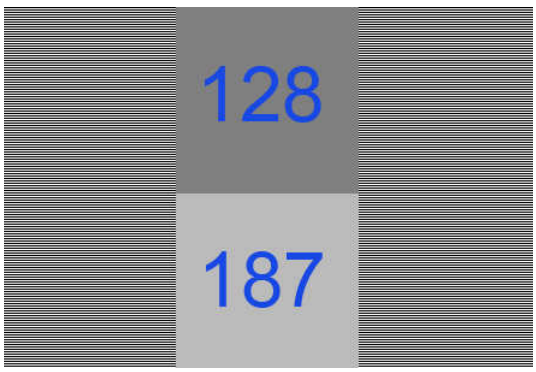


Linear-space lighting is the single most important thing for graphics programmers and technical artists to know. It's not that hard, but for whatever reason, no one really teaches it. Personally, I got a BS and MS at Georgia Tech, took basically every graphics class they had, and didn't hear about it until I learned about it from George Borshukov. This post is essentially the short version from my talk at GDC this year. You can check out the slides for more details.

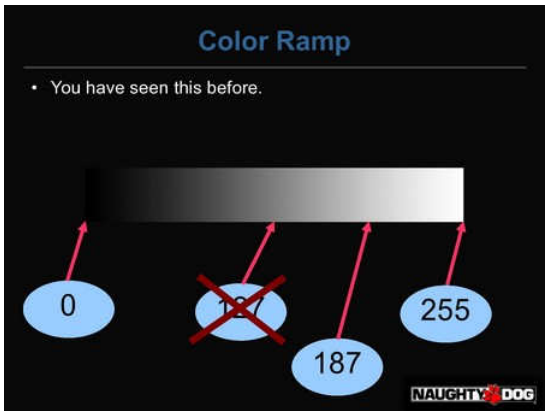
Here is the image I always start out with.



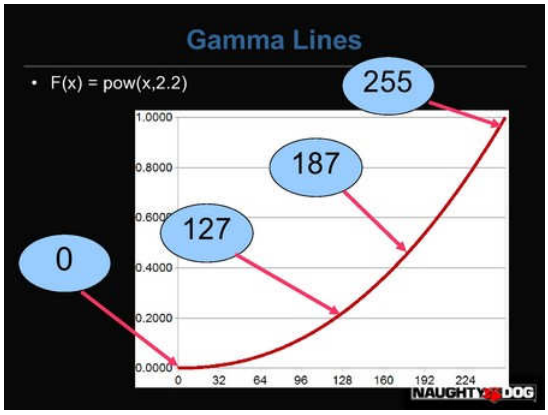
Hopefully your browser is not resizing it. The left and right are alternating horizontal lines of white and black. If you have a well-calibrated monitor, the middle-upper square should be much darker, and the middle-bottom square should be about the same intensity as the alternating lines. On the left and right side, you get essentially half-way between 0 and 255. So the color that is half-way between should be 127/128, right?



Nope. Actually, 187 is about half-way between 0 and 255. 128 is about much, much darker than half. Wtf? Btw, for all the images that look like slides, you can click on them for a higher-res version.

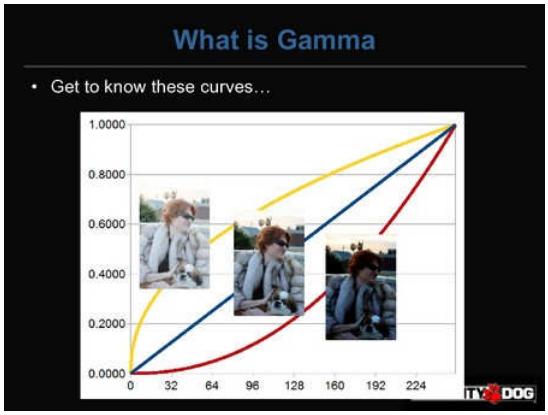


Here is a gradient from 0 to 255. Said another way, 128 is not actually half-way between 0 and 255. Rather, 187 is. Why? Well, there is this thing called gamma.

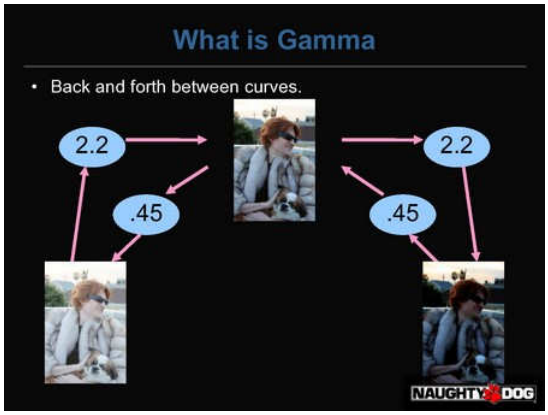


When you send an output value to your monitor, I always naturally assumed that the number of photons would increase linearly with the number I send it. I.e. the value 50 should send twice as many photons as the value 25. But in reality most monitors follow a gamma curve of about 2.2. If we think of our output value as being in-between 0 and 1.0 instead of 0 and 255, the number of photons will be about proportional to pow(x,2.2).

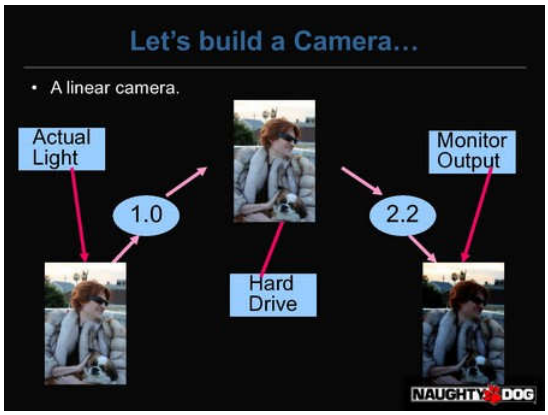
It can vary in different conditions. If you ever hear that Macs have a different gamma, it's because their default gamma is 1.8 (although I hear they recently changed it). Also, the sRGB curve is very similar, it is actually slightly different from a gamma 2.2.



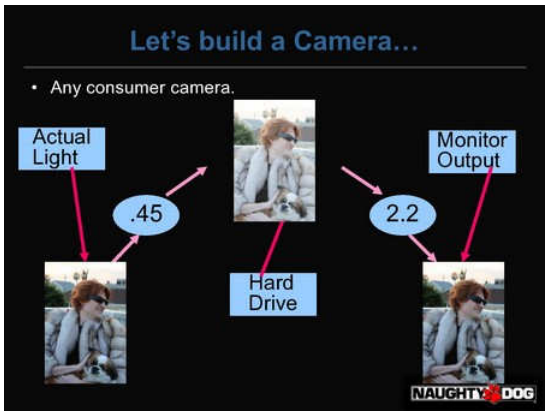
Here's an image of me wearing an expensive mink coat that is much too small for me. Notice how the sleeves barely go to my elbows? The image on the left is what I look like with a gamma of 1/2.2 (which is close to 0.45). The middle image is linear with no changes, or a gamma of 1.0. The dark image on the right is a gamma of 2.2.



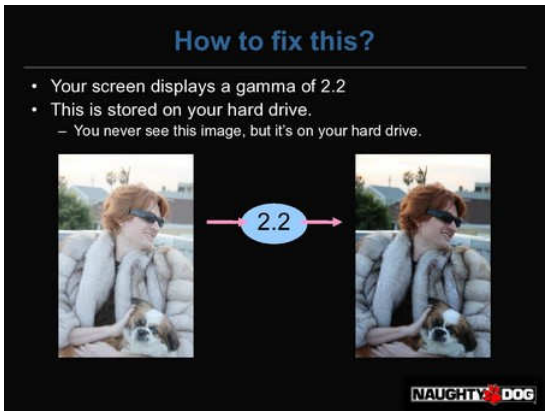
We can actually convert between gamma spaces with a power operation. Starting with the too bright image on the left, we can convert to the neutral image by applying a pow(x,2.2) to the image, and get to the one on the right by applying a pow(x,2.2) again. But, we can go the other way by applying the inverse function, which is pow(x,1/2.2).



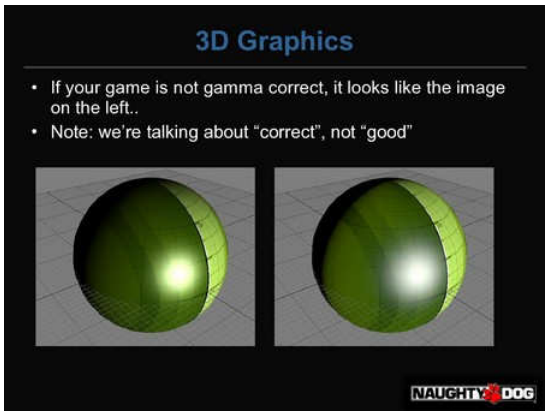
Suppose that we built a camera without any knowledge of gamma. We would build a sensor that would take incoming light (on the left) and split each pixel into 255 levels. That image would be stored on our hard-drive (middle). But, when the user would look at the image on their screen, the monitor would apply a gamma of 2.2 (right). If we did this, users would complain that our camera sucks because the image on their screen doesn't match what they see in the real world.



You could try to explain to them that the camera is correct and literally everyone's monitor is wrong (including the preview LCD on the back of your camera). Good luck with that. Instead, what EVERY consumer camera does is store the image in **gamma space**. When the light hits the sensor, the camera applies the pow(x,1/2.2) operation to it, and then breaks it into 255 levels, and stores the result on your hard drive. Your point-and-shoot does this. Your webcam does this. Your iPhone does this. You're SLR does this. The only cameras that do not do this are usually computer vision cameras. So the image is stored in gamma space, where it is actually kinda bright and pastel-ish. But when the user sees the image on their screen, it looks correct to them.

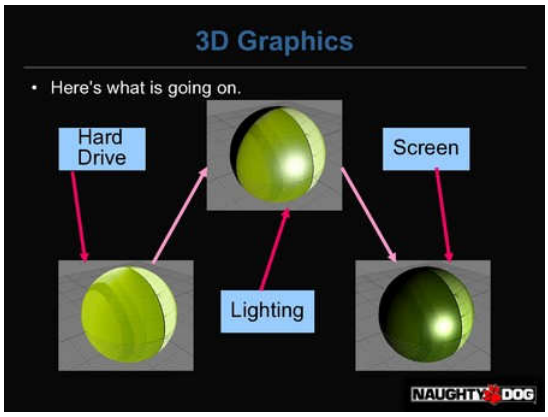


That's the single most important thing you need to know about gamma. Any time you see an image on the screen (like the one on the right), the image actually stored on your hard drive is bright, and pastel-ish, and looks like the image on your left. So how does this affect computer graphics?



The left image does all the lighting calculations in **gamma-space**. The right image does all calculations in **linear-space**. If you are on the PC/PS3/360, you should be doing your lighting in **linear-space**. If you are on the Wii/PSP/iPhone, you gotta do what you gotta do.

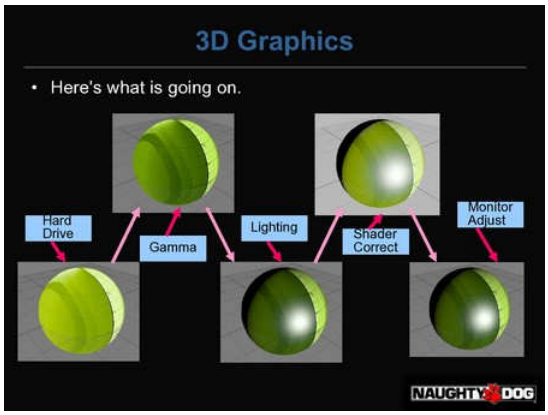
The image on the left is wrong for several reasons. First, it has a really, really soft falloff, which does not look correctly. Also, you can see a lot of hue-shifting, especially in the specular highlight. It seems to shift from white to yellow to green and back to yellow. When you look at the image on the right, it looks like a diffuse surface with a white specular highlight. It looks like what the lighting model says it should look like. Finally, I'm not talking about what looks "good", I'm talking about what is "correct". That's an entirely different discussion.



If you aren't paying any attention to gamma, you are essentially following this flowchart. Suppose you have a typical shader like so:

```
1. float specular = ...;
2. float3 color=tex2D(samp,uv.xy);
3. float diffuse = saturate(dot(N,L));
4. float3 finalColor = color * diffuse + specular;
5. return finalColor;
```

You are first reading the texture. BUT, the texture that is on your hard drive is in gamma-space. The texture is much brighter and more desaturated than the texture you see when you look at it on your screen. So you are taking this too-bright texture, and applying your lighting model to it, for the middle image. Your monitor then applies a pow(2.2) to it which darkens it and gives you the final result. How do you fix this?

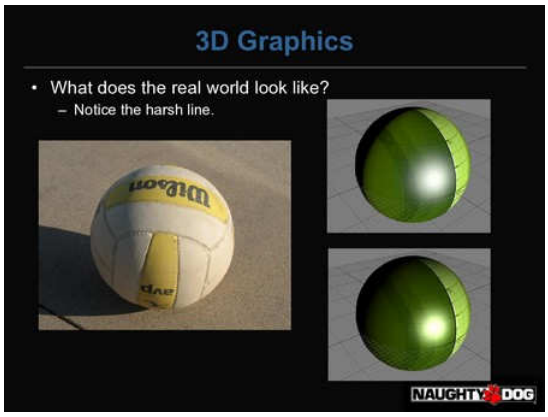


Here is the modified shader.

```
1. float specular = ...;
2. float3 color=pow(tex2D(samp,uv.xy),2.2);
3. float diffuse = saturate(dot(N,L));
4. float3 finalColor = pow(color * diffuse + specular,1/2.2);
5. return finalColor;
```

The only difference is the two pow instructions. When the texture is read by the hardware, it is in **gamma-space** (1st image). But the pow(x,2.2) converts the texture back into **linear-space** (2nd image). Then we do the lighting calculations (3rd image). Now that we have the final image that we want, we apply a pow(x,1/2.2) to convert it back into **gamma-space**. The **gamma-space** image is sent to the monitor, which applies it's own pow(x,2.2) and displays the final image.

Of course, those pow() functions aren't free in a shader. But they are in fact free if you use the hardware sampler states. For the texture read, you can use D3DSAMP_SRGBTEXTURE, and for the write to the framebuffer, you can use D3DRS_SRGBWRITEENABLE. So now your shader has the same code as before, and the hardware states give you the conversions for free.



Here is a real image of a volleyball. The top CG image is in **linear-space** and the bottom is in **gamma-space**. Note how the linear image matches the harsh falloff of the real image, but the gamma one does not.

So that's how to make your image look "correct". Making it look "good" is an exercise for the reader.

Filed under: Gamma! by admin

19 Responses to "Linear-Space Lighting (i.e. Gamma)"

1. **Matumbo**, on June 21st, 2010 at 1:26 am Said:

   When applying the two gamma corrections (back and forth), aren't we losing precision two times, once in the bright areas, and once in the dark areas ?

   Anyway, thanks for providing such a clear explanation on a really essential topic.

2. **francoisgfx**, on June 21st, 2010 at 1:50 am Said:

   this is a great explanation !!!
   it's always hard to explain this topic, I add it to my linear topic links : http://www.francois-tarlier.com/blog/index.php/2010/02/understanding-gamma-and-linear-workflow/

   thx

3. **francoisgfx**, on June 21st, 2010 at 2:08 am Said:

   @Matumbo if you are re-saving every time into 8bits yes. if you stay in 32bits I don't believe so. as much as you are not clamping the values at the end, you are good to go.

   @john on SLR (or any digital camera) if you save your picture in RAW file format, gamma are not saved into the color. So mostly saved as linear.
   Also, is there similar OGL function to fetch pixels in sRGB space ?

4. **Sebastien Lagarde**, on June 21st, 2010 at 4:14 am Said:

   Hi,
   Since some year now, there is a lot of post about gamma correct pipeline (Bungie conference on Halo, GPU gem 3, insomniac etc…), but once you get your gamma correct rendering engine, there still lot of work to do…

   How your team calibrate their screen ? What software do you used for calibration ? How calibrating screen for the PS3 ? What are the right settings to put in Photoshop to work in sRGB etc… Is there a person in the team in charge to follow the gamma settings of everyone else ?

   Lot of pratical questions which are important to get a beautiful game but never discuss in any article I read on the subject. I am curious about how game developer manage this process.

   Thanks for your great GDC talk!

5. **frank**, on June 21st, 2010 at 4:14 am Said:

   But what's a pain is that the PS3 doesn't do gamma correct blending whereas the 360 and most recent PC GPUs do. To have the same look on all consoles you have to emulate the PS3 bug (mostly by manually converting to gamma space at the end of the shader and letting the hardware blend in that space – gamma instead of linear)

   Do you have other tricks to circumvent that important issue?

6. **Sebastien Lagarde**, on June 21st, 2010 at 5:18 am Said:

   @frank
   There a "workaround". You can do all the rendering in linear space and convert to gamma space RGBA8 in the last postprocess.
   As an example take a look at the Naughty Dog slide (GDC2008), you can see that they do there opaque pass in linear LogLuv RGBA8 (This avoid the costly RGBA16F while keeping a good HDR range), then they still require RGBA16F for translucency then convert to gamma space at the end of the rendering pipeline.

7. **Diary of the Droune - Gamma Correction**, on June 21st, 2010 at 6:05 am Said:

   [...] recently pointed out by John Hable of Naughty Dog here, gamma correction is an extremely important topic for graphics, but a topic that you never learn at [...]

8. **Kester**, on June 21st, 2010 at 6:35 am Said:

   @francoisgfx OGL has EXT_framebuffer_sRGB
   and GL_EXT_texture_sRGB. You have to set an sRGB texture format to glTexImage, instead of setting a render state.

   The PS3 is quite happy to texture from the framebuffer… you could do a linear alpha blend in the shader.

9. **admin**, on June 22nd, 2010 at 12:57 am Said:

   @Matumbo: Yes, you are sacrificing precision in your high end to get more precision in the low end. This is actually a good thing because the human eye has more precision in the low end. For example, if you look at a solid gradient from 0-255, you will generally not see any banding in the higher areas, but you will probably see banding in the low areas.

   @Sebastien 1: Ideally you should calibrate for your screens, but that shouldn't stop you from doing it. Your screen is far closer to a gamma 2.2 than a gamma 1.0. (-: And of course, if you ship on consoles, most of your users will have screens that are too warm and contrasty.

   @Sebastien 2: Correct. That's why we do all blending in FP16. And since FP16 alpha blending isn't the PS3's strong suit, we do most of our particles in half-res and then upsample.

   @Kester: Setting gamma in glTex2D() is actually an artifact of the OpenGL language, not the hardware. Since we write code in libgcm, we can change sRGB reads and writes to on or off as render states at any time. But if you're coding in OpenGL on PC, you unfortunately have to specify your sRGB-ness during your glTex2D call.

   Btw, the most annoying issue for me with making your 360 and PS3 match is that the hardware gamma on the 360 is pretty bad. You can see that in this blog post.

10. **Arseny Kapoulkine**, on October 10th, 2010 at 2:45 am Said:

A quick question.
As far as I understand, the gamma space transformation only makes sense in [0..1] interval. This should have the following implications:
- for output, the linear-to-gamma conversion should be performed after the HDR color is converted to LDR one (so either after tone mapping or inside the tone mapping function?)
- for colors/intensities input (i.e. lights), the sensible thing to do is to separate light color (which is probably more intuitively tweaked in gamma space? or just tweak hue/saturation only) and light intensity (which is probably tweaked directly in linear space). Then the shader gets light-intensity * gamma-to-linear(light-color), if necessary.
- for high dynamic range inputs (i.e. lightmaps or environment maps), the input should be linear, because gamma space does not make any sense there.

Is that about right, or did I confuse everything?

11. **admin**, on October 11th, 2010 at 7:56 pm Said:

    Hi Arseny. Yes, yes, yes, and yes. (-:

12. **Yet Another Post about Gamma Correction » #AltDevBlogADay**, on June 2nd, 2011 at 4:35 pm Said:

    [...] Gamma in shaders http://filmicgames.com/archives/299 [...]

13. **Feeding a physically based shading model « Sébastien Lagarde**, on August 17th, 2011 at 6:00 pm Said:

    [...] [21] http://www.luxpop.com/ [22] Hable, “Linear-Space Lighting (i.e. Gamma)” http://filmicgames.com/archives/299 [23] Schüler, “An efficient and Physically Plausible Real Time Shading Model” Shader [...]

14. **Linear Workflow for Game ArtistsRuben Henares – Technical Artist**, on October 22nd, 2011 at 8:07 pm Said:

    [...] Further reading: Gamma Correction – Wikipedia GPU Gems 3 – Chapter 3 John Hable – Linear Space Lighting (Naughty Dog) [...]

15. **Gamma-correct rendering | Molecular Musings**, on November 21st, 2011 at 6:03 am Said:

    [...] rendering/lighting already, and there’s a few good resources on the internet, especially this blog post and a chapter from GPU Gems [...]

16. **Gamma校正和线性渲染管线 « fseraph's space**, on February 3rd, 2012 at 4:55 am Said:

    [...] http://filmicgames.com/archives/299 [...]

17. **Easy Linear Workflow in Maya Tutorial | Cambrand Designs 3D Blog**, on April 8th, 2012 at 4:13 am Said:

    [...] Linear-Space Lighting @ Filmic Games.com - In-depth guide [...]

18. **FluxHoudini Project: Part one | A newborn elderly**, on April 30th, 2012 at 3:57 pm Said:

    [...] range of the HDRI based lighting, fortunately Linear light space (more about Linear Workflow here, here, here and here) is the default for Houdini Mantra So, no worries about that [...]

19. **This is Sparta - Defrost games**, on December 5th, 2012 at 2:43 pm Said:

    [...] One of the issues when doing lightning is Gamma, this actually comes from how the monitor is displaying your image instead of from your code but you still will have to compensate for it. Due to how old CRT monitors worked a linear change in value for the RGB channels resulted in a non linear change in value on the screen. This might be a bit confusing so I’ going to explain it using some images created by John Hable [...]

« Graphics Research: Fast Enough For Games? Gamma and Mipmapping »

- ## Categories
  - CSG (1)
  - Gamma! (6)
  - GPUs (3)
  - Materials (1)
  - Misc (5)
  - Skin Shading (2)
  - Tonemapping (3)
  - Uncategorized (14)
- ## RSS Feeds!
  - Subscribe posts via RSS
  - Subscribe comments via RSS
- ## Meta Meta Meta!
  - Log in
  - Entries RSS
  - Comments RSS
  - WordPress.org
- ## Archives

- February 2012 (1)
- September 2011 (1)
- August 2011 (2)
- July 2011 (1)
- May 2011 (2)
- April 2011 (1)
- March 2011 (1)
- February 2011 (1)
- December 2010 (1)
- November 2010 (1)
- October 2010 (1)
- September 2010 (3)
- August 2010 (2)
- July 2010 (2)
- June 2010 (3)
- May 2010 (7)
- April 2010 (5)

Name:
John Hable

Known Alias(es):
The John

Status:
MIA

Last Known Contact:
filmicgames@gmail.com

Bio:
Little is known. Reliable sources have confirmed sightings at Georgia Tech, Electronic Arts in Vancouver and Los Angeles, and Naughty Dog. He is rumored to have worked on Uncharted 2 and Steven Spielberg's project, LMNO. Several eyewitnesses claim to have seen him compressing Tiger Woods's face, but said reports are dismissed as speculation.