“pixels” are the colored dots on a computer screen. “texels” are the colored dots in a texture image.

Yes – the driver, and its interface to OpenGL, adapts to the hardware. GPUs are made to be queried by the driver to determine the features. There are also OpenGL functions that can query the hardware so that your own software can adapt to the hardware it is running on.

The extensions that are available exist in the driver. Your OpenGL code can ask which ones are there.

It is the hardware vendors that write the drivers, so they definitely are given previews of new GPU hardware. Both the OpenGL and the Vulkan drivers take advantage of the same hardware, but the OpenGL drivers do more “hand-holding” and checking, which sucks up performance. Vulkan is leaner, and requires you (the app developer) to take more responsibility for things that would have been in the OpenGL driver, such as GPU memory management.

glBindTexture gets called in 2 places:

It first gets called in InitGraphics( ) to dock with the state so that the texture settings can be passed from your application to the area in GPU memory set aside for this texture.

It then gets called from Display( ) to make that texture and its settings part of the state so that it can be used.

Yes, well said!
OSU’s College of Engineering bought a massive Nvidia server with 96 Nvidia V100 cards in it. The V100 has no hardware graphics capability on the chip, just compute. It could be used for computational rendering, but right now it is mostly used by the Deep Learning folks.

The Oregon terrain map has no normal map, nor does it used baked lighting. What it does have is the height field data so that it can compute a surface normal at each pixel using a cross-product. It then applies the lighting equation to that surface normal.

Very cool video! Thanks for posting it.

I like it better when you do the voice-over because it is fun to hear how you describe your work. However, it is not required.

It’s not just putting a line there. It is adding 2 triangles to the empty 4-sided space on the bottom. Same with the sides. The top triangles, the side triangles, and the bottom triangles together make it a solid.

Yes. The height field came from the USGS NED dataset, which I wrote a program to interface to. The image came from a satellite photo. The 3D printing was done on a color ZCorp machine.

Yes, but we don’t need it to compute the surface normal at the blue dot.

I made those notes up this morning. I will post it with the Live lecture stuff on the Resources page.
13:12:25 From Mitchell-Nelson, Luke Andrew: Did I miss something, does the magnitude of the vector mean anything or just the direction?
13:12:45 From Sterrett, Matthew B: The length of normals don't typically matter
13:12:53 From Guidos, Job M: so it sounds like the origin of the normal is the vertex
13:12:54 From Haines, Grant A: Yeah just the direction matters
13:15:02 From Guidos, Job M: but how do you orient a vector without an origin :(
13:15:15 From Jacob Eckroth(He/him/his): What do you mean "orient a vector
13:15:16 From Haines, Grant A: It needs to be both a normal vector and a unit vector
13:15:32 From Guidos, Job M: normal vectors don't need to be unit vectors
13:15:50 From Haines, Grant A: For this they do, opengl wants them to be
13:16:07 From Land, Hunter: I think OpenGL converts to unit vectors automatically
13:20:59 From Guidos, Job M: It's nice that it give you the option to not normalize, in case you've already done it.
13:21:11 From Markwell, Cameron Douglas: why do you need a vector there in the glNormal3f?
13:21:25 From Jacob Eckroth(He/him/his): because it's asking for a vector

OpenGL lighting requires that the normal be a unit vector. As an option, you can get OpenGL to do this computation for you, but it will cause a small loss in performance.

Vectors only have direction (orientation) and magnitude, not a location. We like to draw diagrams of the normal vector attached to the vertex, but vectors really don’t have a location.

13:21:31 From Zach Parsons: So basically there is no consequence to doing per vertex normal except time to compile?
13:21:33 From Louie, Simon: that's what a normal is

That's right. It costs a little time in the extra glNormal3f() calls. But, in exchange, you have the opportunity to do smooth lighting.

13:31:07 From Neiger, Kevin Daniel: When do we get to do ray tracing?
13:31:52 From Neiger, Kevin Daniel: Oh, wow, I didn't think we would actually do ray tracing, lol
13:34:02 From Davis, Ryan Alexandra: Dumb question, but are we going to be doing global illumination as well

We aren’t going to officially “do it”, but we are going to talk about it in the Rendering notes. We could do it, however, if we do an ad hoc Blender session. The programming part of this class is really based on hardware-accelerated interactive 3D graphics.