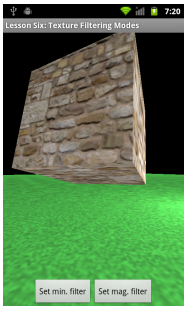
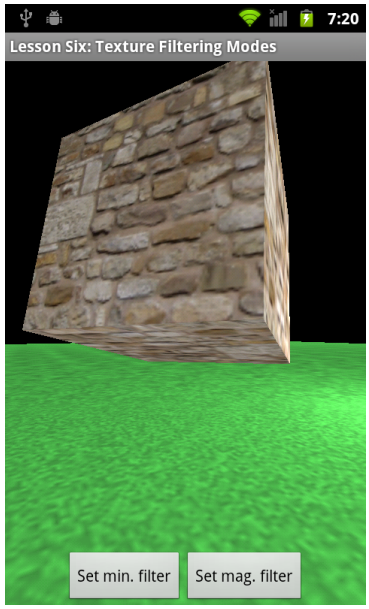
Android Lesson Six: An Introduction to Texture Filtering



In this lesson, we will introduce the different types of basic texture filtering modes and how to use them, including nearest-neighbour filtering, bilinear filtering, and trilinear filtering using mipmaps.



You'll learn how to make your textures appear more smooth, as well as the drawbacks that come from smoothing. There are also different ways of rotating an object, one of which is used in this lesson.

ASSUMPTIONS AND PREREQUISITES

It's highly recommended to understand the basics of texture mapping in OpenGL ES, covered in the lesson Android Lesson Four: Introducing Basic Texturing.

WHAT IS TEXTURE FILTERING?

Textures in OpenGL are made up of arrays of elements known as *texels*, which contain colour and alpha values. This corresponds with the display, which is made up of a bunch of pixels and displays a different colour at each point. In OpenGL, textures are applied to triangles and drawn on the screen, so these textures can be drawn in various sizes and orientation. The texture filtering options in OpenGL tell it how to filter the texels onto the pixels of the device, depending on the case.

There are three cases:

- Each texel maps onto more than one pixel. This is known as *magnification*.
- Each texel maps exactly onto one pixel. Filtering doesn't apply in this case.
- Each texel maps onto less than one pixel. This is known as *minification*.

OpenGL lets us assign a filter for both magnification and minification, and lets us use nearest-neighbour, bilinear filtering, or trilinear filtering. I will explain what these mean further below.

Magnification and minification

Here is a visualization of both magnification and minification with nearest-neighbour rendering, using the cute Android that shows up when you have your USB connected to your Android device:

Magnification

As you can see, the texels of the image are easily visible, as they now cover many of the pixels on your display.

Minification

With minification, many of the details are lost as many of the texels cannot be rendered onto the limited pixels available.

Texture filtering modes

Bilinear interpolation

The texels of a texture are clearly visible as large squares in the magnification example when no interpolation between the texel values are done. When rendering is done in *nearest-neighbour* mode, the pixel is assigned the value of the nearest texel.

The rendering quality can be dramatically improved by switching to *bilinear interpolation*. Instead of assigning the values of a group of pixels to the same nearby texel value, these values will instead be linearly interpolated between the neighbouring four texels. Each pixel will be smoothed out and the resulting image will look much smoother:

Some blockiness is still apparent, but the image looks much smoother than before. People who played 3D games back in the days before 3D accelerated cards came out will remember that this was the defining feature between a software-rendered game and a hardware-accelerated game: software-rendered games simply did not have the processing budget to do smoothing, so everything appeared blocky and jagged. Things suddenly got smooth once people starting using graphics accelerators.

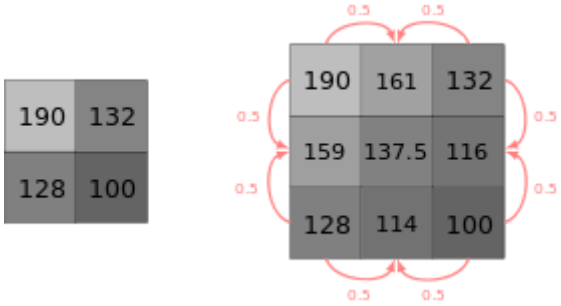


Image via Wikipedia

Bilinear interpolation is mostly useful for magnification. It can also be used for minification, but beyond a certain point and we run into the same problem that we are trying to cram far too many texels onto the same pixel. OpenGL will only use at most 4 texels to render a pixel, so a lot of information is still being lost.

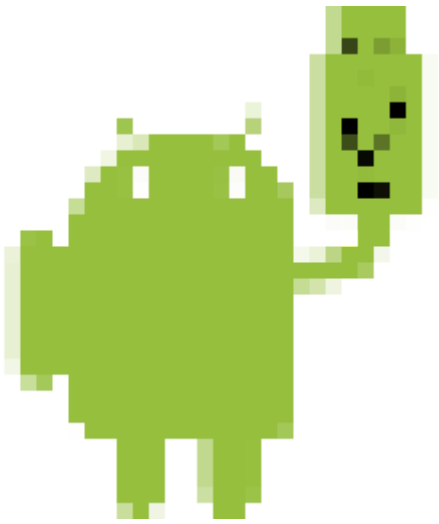
If we look at a detailed texture with bilinear interpolation being applied, it will look very noisy when we see it moving in the distance, since a different set of texels will be selected each frame.

Mipmapping

How can we minify textures without introducing noise and use all of the texels? This can be done by generating a set of optimized textures at different sizes which we can then use at runtime. Since these textures are pre-generated, they can be filtered using more expensive techniques that use all of the texels, and at runtime OpenGL will select the most appropriate level based on the final size of the texture on the screen.

The resulting image can have more detail, less noise, and look better overall. Although a bit more memory will be used, rendering can also be faster, as the smaller levels can be more easily kept in the GPU's texture cache. Let's take a closer look at the resulting image at 1/8th of its original size, using bilinear filtering without and with mipmaps; the image has been expanded for clarity:

Bilinear filtering without mipmaps



Bilinear filtering with mipmaps

The version using mipmaps has vastly more detail. Because of the pre-processing of the image into separate levels, all of the texels end up getting used in the final image.

Trilinear filtering

When using mipmaps with bilinear filtering, sometimes a noticeable jump or line can be seen in the rendered scene where OpenGL switches between different mipmap levels of the texture. This will be pointed out a bit further below when comparing the different OpenGL texture filtering modes.

Trilinear filtering solves this problem by also interpolating between the different mipmap levels, so that a total of 8 texels will be used to interpolate the final pixel value, resulting in a smoother image.

OPENGL TEXTURE FILTERING MODES

OpenGL has two parameters that can be set:

- GL_TEXTURE_MIN_FILTER
- GL_TEXTURE_MAG_FILTER

These correspond to the minification and magnification described earlier. GL_TEXTURE_MIN_FILTER accepts the following options:

- GL_NEAREST
- GL_LINEAR
- GL_NEAREST_MIPMAP_NEAREST
- GL_NEAREST_MIPMAP_LINEAR
- GL_LINEAR_MIPMAP_NEAREST
- GL_LINEAR_MIPMAP_LINEAR

GL_TEXTURE_MAG_FILTER accepts the following options:

- GL_NEAREST
- GL_LINEAR

GL_NEAREST corresponds to nearest-neighbour rendering, GL_LINEAR corresponds to bilinear filtering, GL_LINEAR_MIPMAP_NEAREST corresponds to bilinear filtering with mipmaps, and GL_LINEAR_MIPMAP_LINEAR corresponds to trilinear filtering. Graphical examples and further explanation of the most common options are visible further down in this lesson.

How to set a texture filtering mode

We first need to bind the texture, then we can set the appropriate filter parameter on that texture:

```
GLES20.glBindTexture(GLES20.GL_TEXTURE_2D, mTextureHandle);
GLES20.glTexParameteri(GLES20.GL_TEXTURE_2D, GLES20.GL_TEXTURE_MIN_FILTER,
```

How to generate mipmaps

This is really easy. After loading the texture into OpenGL (See Android Lesson Four: Introducing Basic Texturing for more information on how to do this), while the texture is still bound, we can simply call:

```
GLES20.glGenerateMipmap(GLES20.GL_TEXTURE_2D);
```

This will generate all of the mipmap levels for us, and these levels will get automatically used depending on the texture filter set.

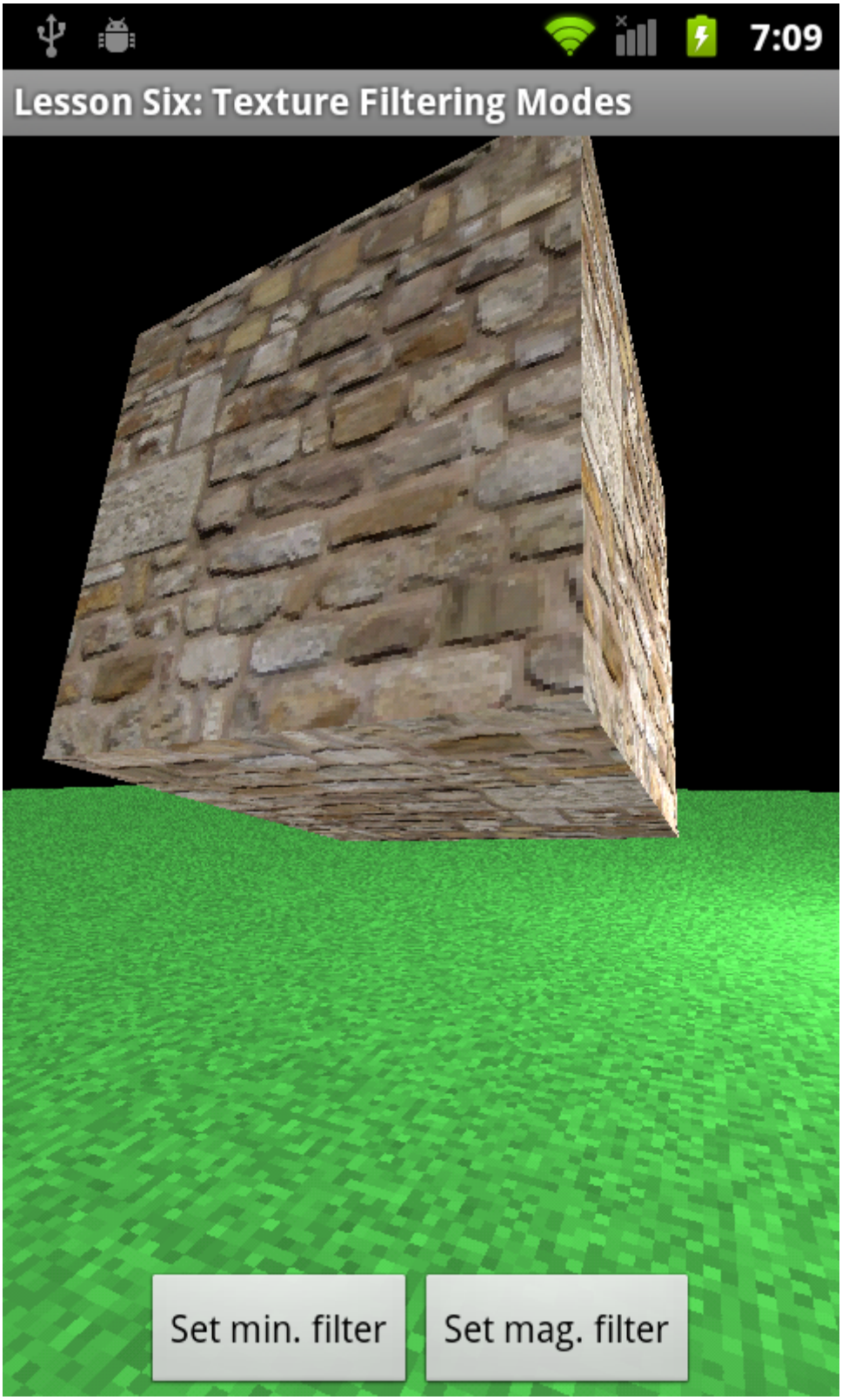
HOW DOES IT LOOK?

Here are some screenshots of the most common combinations available. The effects are more dramatic when you see it in motion, so I recommend downloading the app and giving it a shot.

Nearest-neighbour rendering

This mode is reminiscent of older software-rendered 3D games.

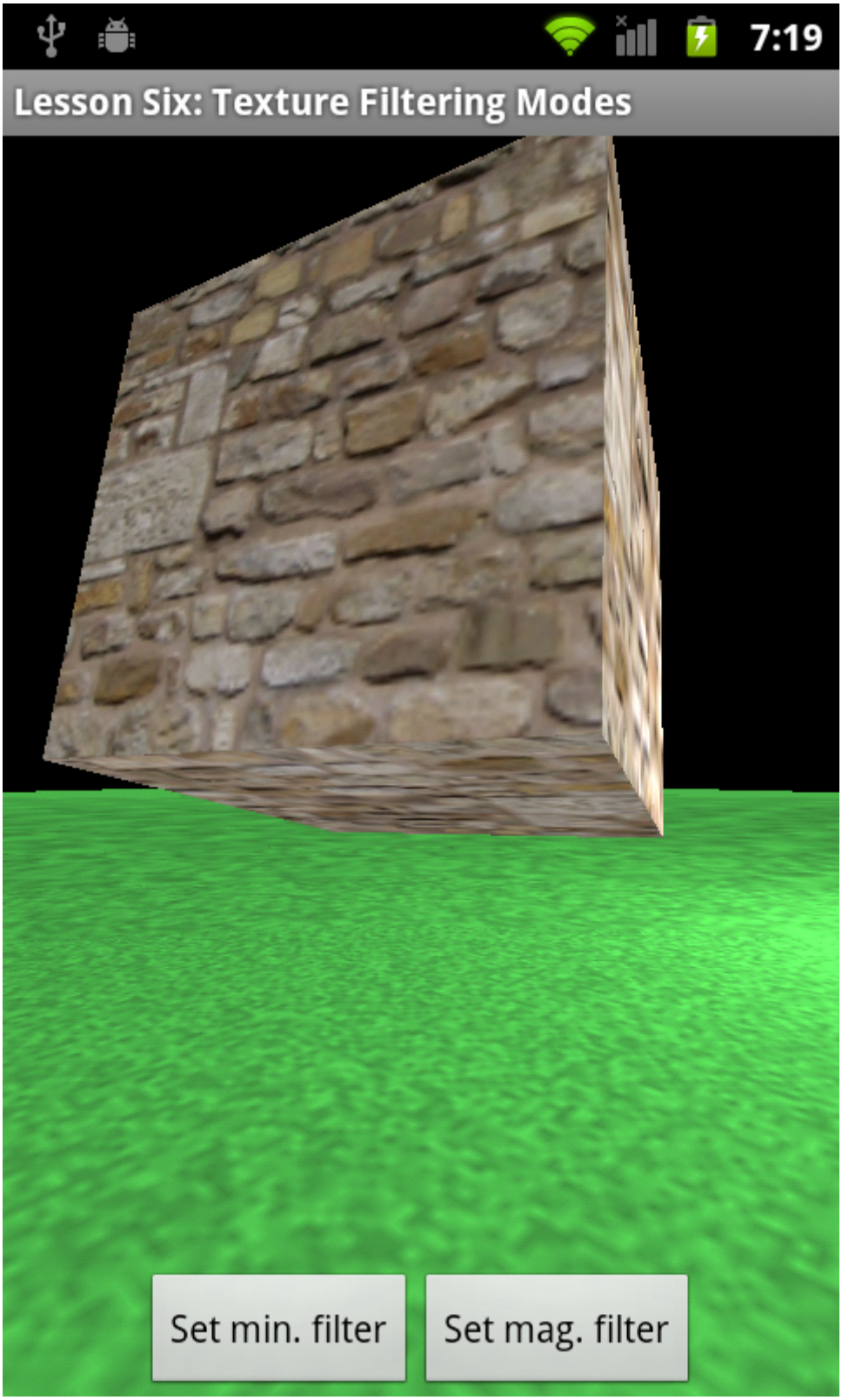
GL_TEXTURE_MIN_FILTER = GL_NEAREST
GL_TEXTURE_MAG_FILTER = GL_NEAREST



Bilinear filtering, with mipmaps

This mode was used by many of the first games that supported 3D acceleration and is an efficient way of smoothing textures on Android phones today.

GL_TEXTURE_MIN_FILTER = GL_LINEAR_MIPMAP_NEAREST
GL_TEXTURE_MAG_FILTER = GL_LINEAR

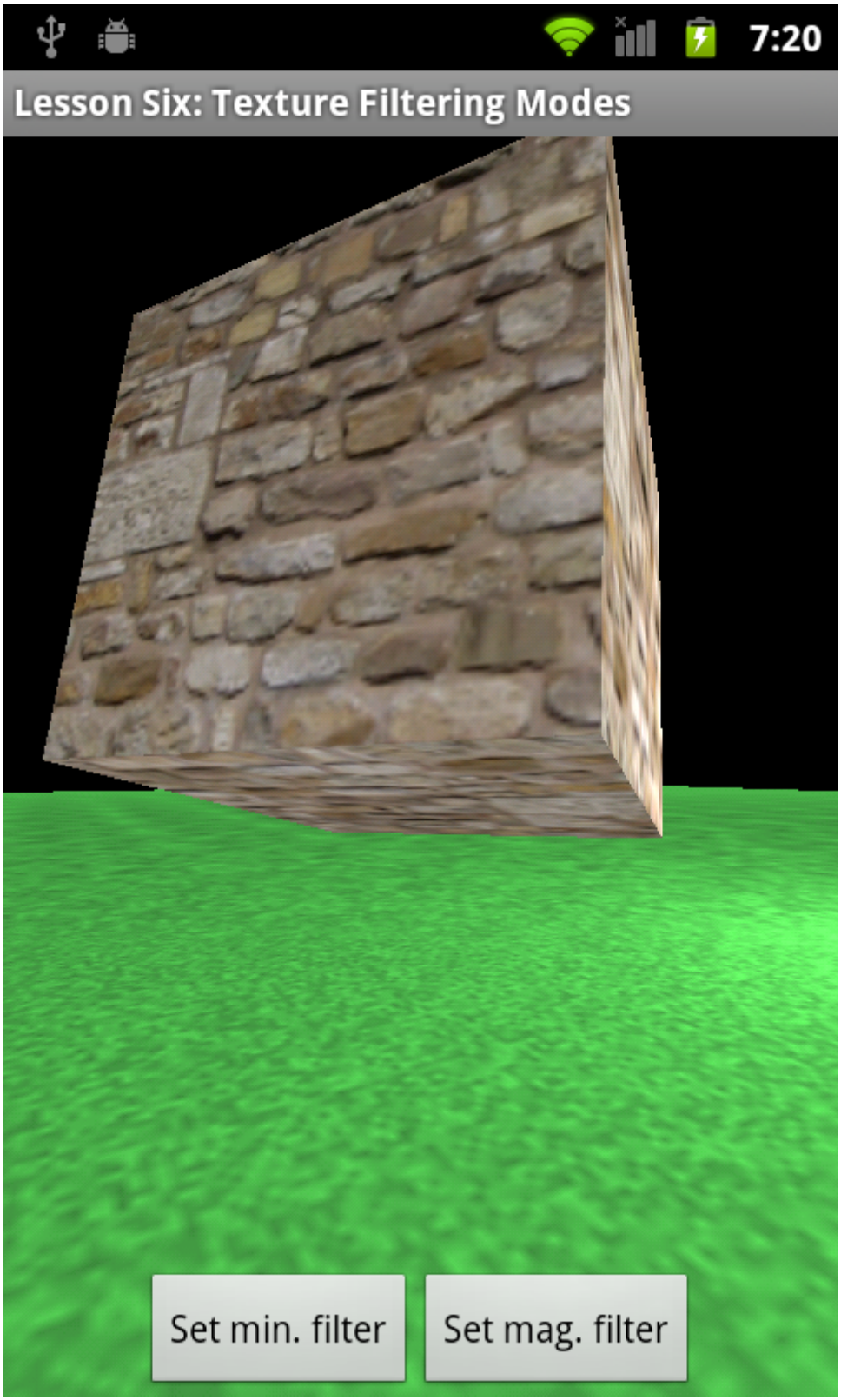


It’s hard to see on this static image, but when things are in motion, you might notice horizontal bands where the rendered pixels switch between mipmap levels.

Trilinear filtering

This mode improves on the render quality of bilinear filtering with mipmaps, by interpolating *between* the mipmap levels.

GL_TEXTURE_MIN_FILTER = GL_LINEAR_MIPMAP_LINEAR
GL_TEXTURE_MAG_FILTER = GL_LINEAR



The pixels are completely smoothed between near and far distances; in fact, the textures may now appear too smooth at oblique angles. Anisotropic filtering is a more advanced technique that is supported by some mobile GPUs and can be used to improve the final results beyond what trilinear filtering can deliver.

Further exercises

What sort of effects can you achieve with the other modes? For example, when would you use something like GL_NEAREST_MIPMAP_LINEAR?

WRAPPING UP

The full source code for this lesson can be downloaded from the project site on GitHub.

A compiled version of the lesson can also be downloaded directly from the Android Market:

I hope you enjoyed this lesson, and thanks for stopping by!

It's never been a better time to learn how to create your own 3D games and live wallpapers. If you can program in Java and you have a creative vision that you'd like to share with the world, then this is the book for you.

Author: Admin

Kevin is the author of OpenGL ES 2 for Android: A Quick-Start Guide. He also has extensive experience in Android development. View all posts by Admin

Admin / February 24, 2012 / Android, Android Tutorials / Graphics, Mipmap, Pixel, Texel (graphics), Texture mapping

19 thoughts on "Android Lesson Six: An Introduction to Texture Filtering"

Pingback: Rotating An Object With Touch Events | Learn OpenGL ES

Per

March 2, 2012 at 2:14 pm

Great introduction that is helping me out alot! Looking forward to more tutorials!

/Per

Admin

March 9, 2012 at 3:40 pm

Thanks, Per! If I can manage it I will try to reach 1 per week.

younes

July 8, 2012 at 12:54 am

thank you very much, that is so helpful and easy to understand ...
I want to ask what is the best filter for 2d mobiles games ??

Admin

July 9, 2012 at 3:54 pm

Hi Younes,

I believe that GL_LINEAR for magnification and GL_LINEAR_MIPMAP_NEAREST for minification is probably the most reasonable tradeoff between speed and quality for a 2D mobile game. The mip-mapping will improve texture render speed as the texture gets smaller, and the bilinear interpolation will make things look smooth. If you don't do any rotations or zooming, you might even be able to get away with GL_NEAREST for both.

tranthuan

October 21, 2012 at 9:01 am

first thank yout very much, that is so helpful for me.
Could you tell me, why you to add mAccumulatedRotation matrix in this lesson instead of just use mModelMatrix and mDeltaX, mDeltaY not assign zero.

Admin

December 6, 2012 at 3:49 am

Hi tranthuan,

I go into more detail on that in this post:
http://www.learnopengles.com/rotating-an-object-with-touch-events/

Eric Kuha

March 14, 2013 at 5:13 am

So, there are a few calls in the main activity in this lesson that are deprecated. Namely the onCreateDialog(int) and the showDialog(int) methods. They want people to use fragments instead. Just a heads up, in case you wanted to make updates!

Admin

March 14, 2013 at 6:08 pm

Yes, you are right. Android advanced a lot over the last 12-18 months or so. 😉

Pingback: MIP Mapping and Texture Sampling | CG Dev / OpenGL

lokendra

October 24, 2013 at 8:26 pm

sir need your help .i want to rotate 3D circle image in android opengl AND AGAIN TOUCH REPALCE THIS IMAGE FROM ANOTHER 3D IMAGE ON TOUCH MOVEMENT .

Admin

October 28, 2013 at 4:18 pm

Please try asking this question on StackOverflow; that way possible solutions will be more easily available and searchable for everyone.

Finiteresource

September 20, 2014 at 6:42 pm

For the record, this is the lesson I got to when I felt compelled to buy the book. This doesn't happen very often: The last computer book I bough was the "The indispensable PC Hardware Book, second edition" in 1996 – handy if you need to poke around in the programmable interrupt controller chip on an 8088.

Admin

September 22, 2014 at 5:48 pm

Thanks, I'm glad that it's helped you out! 🙂

Ariyawat

October 10, 2014 at 2:06 pm

Thank you so much for all of your tutorials.

Deepak

November 12, 2014 at 9:57 am

can we draw a 3D sphere instead of cube and rotate the same on move event.if so then how

---

Admin

November 12, 2014 at 6:42 pm

For a 3D sphere, you can try these resources for mesh construction:

http://stackoverflow.com/questions/18197936/how-to-draw-a-sphere-in-opengl-es-2-correctly
http://stackoverflow.com/questions/6072308/problem-drawing-a-sphere-

in-opengl-es
http://stackoverflow.com/questions/1695421/creating-spherical-meshes-with-direct-x (this last one uses C++ for the answer, but the logic will be similar)

Deepak

November 13, 2014 at 7:31 am

Thanks for the reply ..i will definitely take a look

Deepa

May 24, 2016 at 9:04 am

For my application i need to access texture in vertex shader. I am using the function texture2DLod(texture, cord.xy,mipmap), But this function for some reason gives me the constant value all the time and does not give varying texture. Is it required to set up the Frame Buffer Object for texture2DLod(..,..,..) to work? Do i need to give varying mipmap level or with constant value also it works?