# CS 330 Module Three Assignment Video Transcript

## Views, Projections, Matrices, and 3D Objects

Welcome to our presentation about views, projections, matrices, and 3D objects. By the end of this tutorial, you'll have a better understanding of how we create 3D objects, like this cube, and how we apply color to them.

First, we're going to discuss what is a model matrix. Then, we're going to look at what is the projection, what do we mean by that term. Then, we'll look at the meaning of “view.” Next, we'll look at the concept of perspective. And at the end, we'll bring all these together to show you how they work.

This presentation is a bit different in that we will be jumping to code as we talk about the theoretical part, just to enforce the knowledge. So, we're not going to wait all the way to the end to present the code. Although, we will present it at the end. But we will present parts of it as we speak about a certain subject.

So, what's a model matrix? A 3D object consists of triangles, each triangle consists of three vertices, and each vertex consists of three points, or three values, x, y, and z. There is another way to draw a cube or any 3D object using an indices array. And you can read about that in your reading material. Here, we're focusing on this method, which is creating triangles. Both are fine, and both generate the same result. It's just how it gets rendered. And using indices arrays saves space, since it requires less points.

If you look at this cube, you'll notice that this cube — let's look at the top side of the cube. It consists of two triangles. There's one here and one here. If we look at this triangle, the points are *negative 10, 10, 10*. And here it is declared in an array. The next point is *negative* *10, 10, negative 10*, which is right here. And then the third point is here.

Now, you must have guessed it by now. When we want to draw this triangle here, then we would go *10, 10, negative 10;* *negative 10, 10*; and then *10, 10, 10*. So if you notice, this gets repeated twice. And if you use the other way, indices, you don't do that. You don't repeat the same points.

So, what is a model matrix? Let's put it this way. Any 3D object, as we saw, can be represented using arrays. And when you apply a mathematical operation, a logic, onto an array, the object that gets drawn gets affected.

So, for example, you have an array and you apply the mathematical logic on it to rotate it. So, you're not rotating a matrix. You're just changing the values, assuming the values in there got rotated. And now, remember, these values are represented using the 3D object. So, the object rotates.

So, if you have a matrix and you, let's say, move it on the x-axis, you multiply it on the x-axis. And you shift it three units, and then you draw it. And you'll notice that the object shifts by three units, and so on. And same with scaling. So, any operation you do on an array will affect our object.

As you can see here, we can apply various geometric transitions on the object using matrices. The matrix M — and this is the model matrix, as we will see in a few minutes — it contains every translation, rotation, scaling applied to the object.

Basically, instead of sending the object to OpenGL and then performing — let's say this. Let's say we have a cube, and we want to rotate it 3 degrees, and then shift it to the right. You can apply the rotation and the shifting or translation on the array, and then send that array. And the object will appear in the proper location.

So, you can do more than one operation on the array, and then — or the model. I say array, but really, it's a matrix, and really, it's called a model. You apply that on the model, and then you send it to OpenGL. And it'll get translated into a 3D object.

Perspective. Perspective is how you view the world that you're creating, the viewpoint angle. And before we run it, just a note. Notice that this is the data points. Our world, at least our array, the boundary is *negative 1, negative 1, negative 1,* to *1, 1, 1*.

These are the colors. And if you'll notice, I changed the colors of two triangles in it. One is red. The other is blue. We covered this last week when we talked about coloring. So, we'll leave that for you to review.

And the three arrays — which we talked about one, which is the model, or perspective array, I think — and perspective array, and then the model array. And this is it. This is the model array, OK? Here it is.

So, the perspective array is here. This is where you identify your world, or how you look at the world, using specific parameters. So, for now, all we care about or we're going to focus on is this, the angle. Let me show you. Let's run it.

Here is our cube. Now notice that you're standing in the back, and you're looking at the cube. And your eye angle, it looks like it's 45 degrees. So, 45 degrees, again, this is it here. That's what your eye can cover, this world. See the blue around this cube and the cube in the middle? That's what you can see.

Notice if I change this angle, and I make it 90. Now, for 90, think of it this way. The angle is increasing, so it covers more of the world. Now remember, the world is not just the cube, it's everything in it. So, notice when I said 90, it's like you move back, so the angle can cover more — more objects, more items, and such.

Let me show you another one. Let's say we made this — I'm going to move it back to 45. Sorry. And let's run it. This is what it was before. Now, what if I said 10? Or let's say 20? See? Your view is even shorter. So, you got closer to the object. That's why your angle got reduced.

So, this controls how — by this I mean, let's look at the code. This matrix (and this is a mat4, so it's a matrix of 4 by 4) it identifies — or not identifies, it defines your view, OK?

Of course, it does a lot of other things, including ratio and the conversion between, what do you mean by 1 by 1, on the screen, it's 100 pixels by 100 pixels, and such. So, you can go to the OpenGL documentation to learn more about it.

So, this is perspective. And if you notice here, the degrees, as you increase the degree, the more you can cover from your world.

So, what's projection? A projection is how your object, which is 3D really, you think of it as a 3D world, how it gets projected on the screen. And the screen is a 2D world. It's a 2D image. So, a projection array enables us to — or not us, OpenGL — to translate your way of thinking of a 3D object on the screen into a 2D.

A lot of logic goes into this, including recalculating coordinates, applying shadow, determining what's clear, what's not. Remember, I mean, in OpenGL, objects that are closer to your eye are clearer than the ones farther away and such. So, a lot of math goes into that. A lot of logic, sorry.

Now, we are going to look at the view matrix. What is a view matrix? By default, in OpenGL, the viewer or the camera or your eye is positioned at the z-axis. The view matrix controls the way we look at the scene, at our world. And it's a function call that consists of three parameters.

The eye, the first parameter, determines the position of the viewer. The second parameter is the center, where the camera should aim. And the third one, it defines the direction of the camera. Is it up? Is it looking up or looking down?

The view matrix, which we call v or view, multiplies the model matrix and basically aligns the world or the objects to the camera. What does this mean in code? Let's check it out.

Let's look at our example. We changed the spec to 45. And let's run it. So here is the view, and here are the three parameters right there, right? If you look at the second parameter, it says your eye should look at *0, 0, 0*, which is the center of the world, OK?

Heads up. If you notice, the camera looks at the top. If I change this to negative 1, it'll look upside down. It will flip it. And that becomes a lot clearer in a few minutes. But for now, let's try to apply it to our code. But there is another tool that we'll use that will help you even more in a few minutes.

So now we change this to negative 1. Notice how — you notice that it's flipped upside down. Blue and red are to the right. So that's because your camera, it rotated upside down. And it will become clearer in a few minutes.

The original one is the place — sorry, the first parameter is the location of that camera, the starting position. There is a demo created by one of the students on YouTube. You can Google it and download it. And it shows you the perspective of the camera and the three parameters right here. But it does it visually. And this is your matrix.

So, if you remember, first, the camera target. If you remember, it was *0, 0, 0*. Let's look at it. Here we go, the camera target. So, if you look at it, here, if we change the target, notice that the target now is here, which is *2, 0, 0*, and the object moved. The object did not move. Your camera really moved.

So, if you move back, the object is placed — it's not even placed at *0, 0, 0* really. It's placed a little bit — I think it's at *1, 0, 0*? No. So it's placed like this, almost *0, 0, 0*, OK? So, notice, this controls the target, what your camera is looking at. And notice, it's a point, meaning its x, y, z. So, you could do it like this.

And if you changed — let's go back to here. If you changed this value, the y, that's what you're doing to your object. So, you're either looking above it or down, like this. That's because you're changing the y. Of course, same with x. X, you're either moving farther or — notice that here.

So, in this case, it's almost *0, 0, 5*. It's right here. That's where your center point — that's what you're looking at. Your object is farther from it. It's far away from that point, OK?

The second parameter is — the other parameter is the angle, the degree. And that's the one where I told you, if you change it, it'll rotate the camera. So, when we ran it, our cube, we did it like this. So, it was a cube, so it wasn't as clear, but that's what it looked like. The camera was flipped. That's because we changed this, the degree. And you can, of course, do it on y, x, and z.

Now, the first parameter is the camera position, notice. And no matter where you position it, it stays looking at whatever the target is at whatever degree it is. It's just the location of the camera changes the position of it, and hence the way you view your object. This was the first parameter, I think. We changed, now, a lot of things.

If you remember that, if you remember the degree — let me see if we can reset — reset the camera. If you remember the degree, we talked about the angle that was in the previous model. Notice that when we did 90 — I think the example we used was 90. Let's see, OK. You see? So, much larger angle. And hence the world you cover, it's much bigger.

But if you go down, if you make it a smaller angle, look at this here. I think we did 10 or 20, if you remember the cube. It was 20, I think. The cube was right in your face. That's because the camera position was closer than this.

Yeah, let me see. Sorry, the camera position was like this to the cube. So, when you increase the angle, see, you see more of the world. And as you reduce the angle, you see less of the world.

Let's see, bringing it all together. Let's look at our code. As you can see, first, we must set up the two parameters that are very important. We must read about them in our reading material and OpenGL documentation, which is the depth — the depth test, and the GL\_LESS.

This tells the OpenGL engine how to apply shaders and how to display your object, the close elements and the far elements. Usually, if you don't set up these things, you don't see anything on the screen. So just be careful.

We set up our arrays or matrices, and then we create a big matrix, a big one that consists of all three. Usually in the documentations they call it MVP: Model, View, Projection, OK?

Now this is — this array is our world. It doesn't have anything in it, doesn't have any cube or anything. It just sets up the camera, the units, the ratio of the objects, and the points. Next, you declare your array. You do the color. Sorry, not array, your vertices array.

If you notice, I changed two triangles, just to show you again, to emphasize what we learned before. You create the buffers. You attach the data to them. So, you have a vertex buffer. You have a color buffer. And then, in here, you link your world to what you want to display. And then you just — you upload your buffers to that world, and then you draw triangles, OK?

Now, I have an extra element that's outside our scope now. But if you notice, there's a variable that adds 1. That's an increment, so by 1. We don't care much about that variable. It's this line here. Let me show you, for example.

If I say, “Hey, first let's run the code.” If I want to apply rotation, like, “Hey, rotate this object, OK?” Then all I must do is just tell it, “Hey, on this array, apply rotation.” Rotate is a function, OpenGL function, that takes the matrix, your world, the degrees, and around which point to rotate.

So, if you do this, it will rotate the world and assign it back to your variable, which is then used to display. And if we run it, here it is, as you notice. I mean, it's going fast, because it's a loop. But you can do rotation now to your world, to your objects. That's not an object. We're rotating the whole world.

So, it would be interesting for you, as an exercise, to add two objects in there, another one, and then see what happens with the rotation. Now this is, again, outside your scope. But it's good to know for future weeks.

Now you have an idea of how we create a 3D cube. You should be able to work on creating a 3D pyramid using the same concepts, same commands. It's just you must visualize that object and apply what you learned to it. Good luck.