Good morning, students. Today we have two sections. In the first half, we are going to review some linear algebra. And in the second half, we will talk about Assignment 1. Now let's start 3D Math review. First, scalars. This concept is quite easy. Scalars are just numbers with some operations like addition and multiplication. I believe all of you are able to add two real numbers and multiply two real numbers. Is there anyone who can't do that? In the scalar field, there are some special elements like zero and one. If we add any number and zero, we can get the number itself. If we multiply any number and one, the product is the number itself. Each element has an opposite element. The sum of an element and its opposite element is zero. Also expect the zero element, each element has a reciprocal element and the product of these two is one. Now we extend scalars to vectors. Vectors have length and direction as this picture shows. And vectors also have addition and multiplication operations. When we multiply two vectors A and B, we let the start of B be the end of A, and the result vector C will be from the start of A and the end of B like Picture B shows. When we multiply a scalar α and a vector v, the length of the result vector will be α times as long as the length of v. If α is positive, the direction remains the same. If α is negative, the direction will reverse. How about α is zero? The product will be a zero vector. The start and the end of the zero vector are the same point. The length of the zero vector is zero and the direction of it is arbitrary. Now we can represent a equation of a straight line in the form of vectors. To specify a ray, we need a start point and a direction. If a point is on the ray, the direction from the start to this point is constant, d, but the length α is arbitrary. So the eqution of a ray can be written as $P0 + \alpha d$, for
α is non-negative. If we allow α to be negative, the point can go to the opposite direction, and it become a straight line. If we want to define a segment, we need the two end points P and Q, and the equation is $P(\alpha) = (1-\alpha) Q + \alpha R$ for $0 \leq \alpha \leq 1$
.

With the addition and multiplication operations, we can construct a coordinate system. In a 3D coordinate system, we have three special vectors v1, v2 and v3, which are constant and linearly independent. The most familiar example is the X axis, Y axis and Z axis. We call these special vectors, the basis. If the equation $\alpha 1v1+ \alpha 2v2+ \alpha 3v3$ equals to zero vector has only one solution, $\alpha 1$, $\alpha 2$ and $\alpha 3$ are all zeros, we call v1, v2 and v3 are linearly independent. On that condition, any vector in the 3D coordinate system can be represented as $\alpha 1v1+ \alpha 2v2+ \alpha 3v3$. Since v1, v2 and v3 are constant, we

can just write (α1, α2,α3), just like (x, y, z) as you usually do. Some times we may need to change the origin of the coordinate system, moving from (0,0,0) to P0. We call the new coordinate system Frame. And any point P will be P0 + α1v1+ α2v2+ α3v3. And in practice, the basis of frames are often orthogonal, v1, v2 and v3 are vertical to each other. We will talk about the advantage later. Besides changing the origin, we may also want to change to basis. When we want to change the basis from u to v, we just solve this linear equations and get the nine γ

.

For vectors, we have another two important operations, dot product and cross product, which scalars don't have. If the basis are orthogonal, the dot product of u and v is quite simple, just u1v1 + u2v2 + u3v3

. And that's why we are always using orthogonal bases. The dot product of two zero vectors is zero because all u and v are zero. If the dot product of two vectors are zero, we call these two vectors are orthogonal

. The length of a vector can be defined as the root square of the dot product of this vector and itself. Dot product can be used in projections. The dot product two vectors, u and v, can be also calculate by multiplying the length of u, the length of v and cos theta. Theta is here. By this equation, we can easily compute cos theta. So if we project u onto v, the length of the result vector is the length of u multiplies cos theta, and the direction is the same as v if cos theta is positive. If cos theta is negative, the direction will reverse. If cos theta is 0, we get the zero vector. And we can see from this formula, the dot product is zero and u and v are orthogonal. The other important operation is cross product. This formula is a little complicated and you should remember it. Do you remember this formula ($|a \times b| = |a|\,|b|\,|\sin(\theta)|$

), it's the area of a parallelogram. So cross product can be used to calculate the area of a parallelogram constructed by two vectors. Another conclusion is that the cross product of two vectors is vertical to both these two vectors. We can use the right-hand rule to see the direction of the cross product. Cross product is useful in the plane. A plane is defined by point P0 and two non-parallel vectors u and v. And any point on the plane can be written as $P0 + \alpha u + \beta v$

, where α and β are scalars. The normal vector of a plane is very important because the normal is vertical to any vectors on the plane. How to calculate the normal? Just do cross product and then the equation of a plane can be simply written as the product of

the normal and the difference between any point on the plane and P0.

The last concept is the convex hull. A convex hull is a convex polygon that covers all the given points. Of course you can easily find one of them, just drawing a very big convex polygon. Among all these, we have the smallest convex polygon, we call this the convex hull. The convex hull also has a formal definition like this formula, it's a set of points that can be written as $\alpha 1P1 + ... + \alpha nPn$
. P1...Pn is the given point set and the sum of all α is 1. Each α is no smaller than zero and not larger than 1.

OK, so many for the math review. I hope this review can let you remember what you have learned in linear algebra. Now let's move to discuss OpenGL. This topic is quite related to your assignment. So please pay attention. I have received many emails asking why my OpenGL window draws nothing. Now let's do it step by step.

Let's take rotating a color cube for example. In this problem, we need to draw a color cube, rotate it about x, y, or z axis depending on which mouse button you click and respond to your keyboard like stop rotating and quit.

Step One, it is your responsibility to calculate all positions of a cube and specify the colors on the eight vertices. Step Two, you need to initialize OpenGL by these following commands. GLUT_DOUBLE
means using a double buffering, GLUT_DEPTH
means I want depth test, which is used in removing hidden surfaces. Step Three, we need to set up all callbacks here. We need to specify the size of window. In your assignment, the width should be 640 and the height should be 480. You also need to specify the reshape, display, idle function, mouse function and keyboard function. Step 4, implement the reshape function. This function is called when you initialize OpenGL and resize the window. The two parameters are the width and the height of the window. In this function you need to set up projection, like setting viewport as the whole window and set the projection mode. Here the mode is orthogonal but you should use perspective mode in your assignment. At last, don't forget to switch the mode into model view because all transformations of an object should be in the model view mode. Step Five, display. You should first clean the window by calling glClear first. Clear both color buffer and depth buffer. The do transformation. In your assignment, you should also do glScale and glTranslate besides glRotate. After transformation, you can draw the object by specifying the positions and the color. Don't forget glutSwapBuffers()

, or you won't get any results. Drawing the six faces of a cube. We use GL_POLYGON to draw a cube. In your assignment, you may use GL_TRIANGLES, GL_LINES, GL_POINTS or any other as you need.
Step 8, animation. This is implemented in the idle function. Your can do transformation like rotate. Or take a screen shot to save a image. Don't forget glutPostRedisplay (). Step Nine, change axis according to mouse action. This mouse callback is called when you press or release a mouse button. The first parameter tells the program which button has been triggered, the left button, the middle button or the right button. The second state is whether you press it or release it. The last two parameters are the position of your mouse on the screen. Step Ten, respond to the keyboard. This function is called when your press a button on your keyboard. The first parameter is which button you are pressing and others are the position of your mouse. You can see, if you press q, the program exits. If you press SPACE, the stop state switches between doing nothing and calling spin cube.

In your assignment, you need to render a height field. So how to get a 3D position from a 2D image? In 2D image, the x, y coordinate is determined by the position of a pixel in the image. For example, if we have a 4 times 4 image, the x, y coordinate is zero and zero, this is one and zero, this is three and three, and so on. So where is the z coordinate, the height information? It's the color of a pixel. If the color is gray scaled, the z coordinate is just the value of its gray scale. How about RGB color for extra credit? Well, depends on you, write your function in your README file. Now we have the 3D positions of all vertices. What about the color. Generally, color is to represent the height, the value of z-coordinate in the example image. But you can design by yourself, like texture map an image on your height field for extra credit. Also write your achievement in your README file.

That's all. Thank you. Any questions to ask me or the producer?