All right, welcome to our first lecture on computer graphics. Today we're just going to do an overview of computer graphics with two main objectives. One is to try to understand broadly what computer graphics is all about, where does it show up in the world, and we're actually also going to really get our hands on a first algorithm for making images from 3D shapes. Also, I should say, all information about course logistics is on the webpage. Today, I'm really going to is on the webpage. Today I'm really gonna dive into the content. Okay, so what is computer graphics? Why are we here? What do we want to talk about this semester? When you think about computer graphics, probably an image like this one comes to mind, something from maybe an animated movie, visual effects. But what we really want to study in this class more fundamentally is how computer graphics shows up broadly in computer science, what its function is in digital computation, and why we need it. So if we think back to the earliest computers, they looked something like this, these huge machines that took up an entire room. And the way that you could interface with these machines was very crude. You might have to punch holes in a card that gets fed into the computer. It does some calculation and then it spits out some card also with holes punched in it and And this takes a lot of time, obviously, to set up the program, but also to interpret the results, to understand what is it that came out of the computer. And so you think, okay, well, there must be a better way to get digital information in a form that we can understand, in a form that we can digest. So if you start looking at the history of computation, you'll see that over time, very naturally, people gravitated toward having some kind of visual display to convey what information was being stored or was computed by a machine. And eventually we got to the point where people really started thinking about this problem of display and interaction as a study or topic all on its own. So this is one of the earliest examples of somebody who really was starting to think about computer graphics, human-computer interaction. graphics human-computer interaction. So this is a demo of something called the Sketchpad, which was developed by Ivan Sutherland. I'll place a short little clip here. ...communicate with a machine. Now how do you actually go about communicating with a computer in a graphical sense? Well, we are using an oscilloscope here, which is much like a TV set, except it's being given by the computer. In order to get the information into the computer, we have to draw somehow on this display. And we use the light pen. Right, so people had really started thinking about how to, getting higher and higher resolution, getting richer color, to the point where we're no longer just getting a little punch card with holes with a tiny bit of information coming out of the machine, but now a single image on a monitor, an 8K monitor, might convey about 95 megabytes of information. So we can really pump out a lot of information from the machine. And if we start looking at even newer devices that are coming down the pipe, like virtual reality headsets, augmented reality headsets, these are the devices where we have to supply a phenomenal amount of information in order to drive them. So a headset that you might purchase today has two displays, one for each eye, that are each 2160 by 2160 pixels. They have some number of bits to convey color, and they have to display an image to get some kind of realistic reproduction of something visual. They have to display an image 90 times a second. something visual, they have to display an image 90 times a second. So if you do a little computation, that's saying we're pumping out about 2.3 gigabytes per second of information from the computer. A lot more than we were at the dawn of computing. So one natural question to ask here is why is visual information so important? There are a lot of other ways that we could have tried to get information out of the machine. We could have played sound or had some kind of vibration. Why are we so focused on visual information in computer science and computer graphics well one big reason is that's just the way your brain works about 30 percent of your brain is dedicated to visual processing and so if you think about it from a kind of system level point of view, your eyes are kind of the highest bandwidth port into the head. So if you're trying to get information from a machine into your mind, doing something visual makes a whole lot of sense. Doing something visual makes a whole lot of sense. Okay, and so this kind of explains, perhaps, why computer graphics was developed, why it was such an important part of computer science. And we can start to put together at least a tentative definition of computer graphics. What is computer graphics? Well, we might say that computer graphics is the use of computers to synthesize visual information. We have digital information, just ones and zeros, and we want to perform some kind of computation to turn it into something that human beings can consume, some kind of visual information. Now, if we frame it that way, if we say the point is to create information for human beings to consume, then it's also natural to ask, well, why only, again, why only concentrate on visual information? Sure, the brain has a lot of visual computing power, but there are other things that we could do. And in fact, graphics has evolved a lot since its early days to focus on a much broader set of problems and questions than just the question of how to turn pixels on and off on a high resolution display. So there have been people who've worked on simulating sound, right, generating sound from fictional or virtual 3D models. There are people who've worked on synthesizing or simulating touch so that you could feel the way an object might feel if it existed in physical reality. And so we can broaden the definition of computer graphics and say, really we're interested in using computers to synthesize and manipulate sensory information. Okay, you could go on and maybe think about taste and smell and

other senses might be fun, but the key word here is synthesis. Computer graphics is kind of the unique discipline in computer science that takes information and synthesizes perceptual stimuli. There are other disciplines, let's say computer vision, that look at the inverse problem. I have stimuli in the real world and I want to consume them. I want to interpret them and convert them into digital information. So in that sense, you might think of computer graphics and computer vision as being kind of inverses of each other. vision as being kind of inverses of each other. And the interplay of those two disciplines actually is very important. Something that is very contemporary, going beyond turning pixels on the screen, is turning digital information into another thing we can experience, which is physical matter. So there's a lot of work in computer graphics, for instance, on 3D printing. I have a digital model and I want to print it out here. I'm printing this bunny rabbit slice by slice until I have a solid physical model I can hold in my hands. But it goes much further than just 3D printing. People are looking at all sorts of different ways that you can turn digital models into physical models, whether that's turning digital objects into food or into clothing or into buildings. The bottom right example here is a duck who was born without one of their feet. And so somebody went and scanned the other foot, reflected it on the computer, printed it out to create a prosthesis, right? So there's a lot of interesting things you can do with the technology from computer graphics that goes beyond image generation and is really starting to have a major impact on the broader world, okay? the broader world. Okay, so this is going to be our working definition for this class, how to use computation to turn digital information into sensory stimuli. Although I'd say if you talk to somebody who does a lot of computer graphics, they would still say, you know, that definition is still a bit too narrow. I don't think it really captures all the interesting things that go on in computer graphics. And so if you want a really good picture of what's happening in computer graphics, a great place to look is at the ACM SIGGRAPH conference, where all of the new results in computer graphics are presented every year. Lots of exciting new research papers, and they produce a nice technical papers trailer that just highlights some of the cool stuff that's going on in graphics. So let's take a look at that. Welcome to the SIGGRAPH 2020 Technical Papers trailer. We'll present clips from just a few of the exciting breakthroughs in the Technical Papers program. A learning-based approach to keyframe video stylization enables this artist to craft a personal look in real time. By using four fisheye monochrome cameras, we can track someone's hands in space, enabling them to sort these blocks and win the game. Using clever hinge design, a flat lattice of thin flexible strips can be deformed into a complex 3D structure and then flattened back out again. To create a realistic knit bunny, this algorithm uses an energy density function to drive a thin shell simulator. It can also knit armadillos. By interpolating a sparse series of 3D poses, this algorithm produces smooth, complex motion that still allows for artistic control. This algorithm supports immersed bubbles in free surface flow simulators, producing the familiar glugging action of a water cooler. A method for managing the dynamics of non-linear deformable objects lets us bounce this hairy elastic toy against a wall. Neural networks can improve photographs of people's faces by removing external shadows and simulating a soft fill light. Efficiently coupling the interaction between solid objects and turbulent flows lets us unleash a tornado on toy animals. Generative adversarial networks and deep reinforcement learning help this quadruped navigate a maze to find delicious stars. navigate a maze to find delicious stars. By animating the dynamic fracturing of isotropic and orthotropic materials, we can pull off the top layer of a piece of pork belly. An algorithm that exploits tunnels and configuration space can solve complex puzzles made up of entangled, rigid shapes. A method to implicitly model the space of plausible faces lets us discover new faces by sketching or by merging existing parts. By planning the motion of its two robot arms holding a deformable hot wire, this device can cut complex 3D shapes, like a bunny. This system measures the forces we feel in real and simulated worlds, and then applies those forces with a fingertip device, so we can feel what we see. This block of soil was simulated by a massively parallel material point method for physical materials. It can also collide armadillos. By processing photographs on a mobile device, this algorithm lets us view our scenes from novel viewpoints. This technique models large numbers of colliding elastic bodies with friction, so running characters can enjoy rich, lustrous hair. A particle-based approach to fluid simulation lets us apply neural textures to fluids, even while they're moving. A material method for two-way coupling of nonlinear solids and fluids lets us drop balls full of water and orange juice. Okay so hopefully from that you get the sense that computer graphics is really a rich and diverse discipline that goes far beyond just turning pixels on and off on the screen. And it touches all sorts of different areas of life. Computer graphics really is everywhere. computer graphics really is everywhere. So a lot of us, again, first think of computer graphics as something that shows up in entertainment and movies and games, and that certainly is a very important use of computer graphics. But even within entertainment, it shows up in places you might not even be thinking about. So not just kind of cartoon animation, but lots of movies you go to see these days might have effects in them that are so good you don't realize they're there. Maybe removing wrinkles from the skin or inserting somebody into an old photograph who wasn't really there. Kind of scary. Computer graphics, of course, is used

a lot these days in art and design. People, in addition to traditional media, are almost always doing some kind of digital media painting or modeling or something. In similarly industrial design, you want to make products that are not only beautiful, but have some particular functions. So you're balancing aesthetic aspects with functional or engineering aspects. That's very core to the kinds of questions people look at in computer graphics. How do you balance the physical and mechanical constraints with aesthetic considerations? Also in trying to do engineering, you might just need to visualize data. That's a very important thing. You do a car crash simulation, you need to understand what happens. There's so much data that goes on in these simulations, it can be hard to really understand what happened. And visualization techniques are very important for that. Architecture is being revolutionized by techniques from computer graphics, in particular an area called discrete differential geometry. It has a lot to say about structures that can be easily built, but still can be designed in a very free-form way. In scientific and mathematical visualization, again, you have tons and tons of data that might come out of some high-resolution simulation. Okay, it's great that the computer can do all that. It's great that we can build systems that can solve problems at that scale. But once you've solved those problems, how does a human being interpret the answer? There's just so much information to make sense of. Likewise, in medical or anatomical visualization, right? We have devices that can take very, very detailed scans of the body. How do you come up with techniques and algorithms that let you efficiently look at all that data and understand what's going on? Computer graphics plays a huge role in navigation and the way that you get around the world, right? Anything from just simple 2D maps to more sophisticated 3D maps to autonomous vehicles doing autonomous driving or interesting things like taking collections of photographs that are taken by different tourists and assembling them in some way that lets people get a sense of what that location looks like. Lots of interesting things you can do there. And just broadly in communication, there are things that you don't think about as computer graphics, but require some pretty amazing technology. So actually every single letter that you see drawn on your screen when you're reading the newspaper, when you're reading a website, there's some pretty sophisticated algorithms that are needed to draw all that typography and draw it very, very quickly. And it kind of works like magic. You don't even realize that there's some sophisticated algorithm running because people have done such a good job at developing this technology, but it is sitting there in the background. Likewise, people are exploring all sorts of new mechanisms for communication, for telecommunication beyond the written word. So like virtual avatars that you might sit in front of your camera and turn you into a 3D model. I mean, the possibilities are endless. So there's so much to do. What are we going to do in this class? Well, we're really going to look at the foundations behind all of this, the computational foundations behind all these different applications, right? Because all these applications, whatever domain you're excited about, demand sophisticated theory and a sophisticated treatment of computer systems. So in the theory category, we're going to talk about things like basic representations. How do you digitally encode shape or motion? A big theme in this class is going to be sampling and aliasing. So how do you navigate acquiring a signal and reproducing a signal so that the thing you reproduced faithfully represents the thing you acquired. We're going to talk a lot about numerical methods. So how do you manipulate signals numerically? How do you solve equations numerically? solve equations numerically. In a lot of computer science education, there's a focus on discrete combinatorial problems, working with integers and so forth. This is really going to be a deep dive, your first real hands-on experience perhaps, in working a lot with floating point representations of numbers, which can be a little tricky. We're going to talk about radiometry and light transport. So how do we, in a very precise way, talk about how light looks, what color it is, how bright it is, how it's distributed in the scene. These questions are all very important for trying to create photorealistic images. And we're also going to connect this a little bit, at least, to perception. perception, right? Remember, again, the goal is to take digital information and convert it into stimuli that a human being can consume. Well, if that's the case, then understanding how human beings perceive visual information and other stimuli is really going to factor into our algorithms in an important way. For instance, if we say we want to compress an image so that a person doesn't notice anything changed, we really have to understand something about human psychology. understand something about human psychology. On the system side, you know, if we want to create algorithms that pump out gigabytes of data into a VR headset, well then we really need to think about performance and the way that information is exchanged by different processes and algorithms in our pipeline. So we're going to talk about things like parallel or heterogeneous processing that might be needed to run fast graphics algorithms. We're going to talk a little bit about graphic-specific programming languages, like shader languages. graphic specific programming languages, shader languages, and more generally we're going to talk about how we can formulate problems in computer graphics in terms of a pipeline. What are the right primitives to break down our problems into so that we can efficiently stream computation through a piece of hardware. Okay, so that's it for the high-level overview. Let's really dig in and get our hands on some concrete problem. And so what we're going to do for most of the rest of this lecture is we're going to try to think about how we can model a three-dimensional object,

encode it as digital information, and then turn that digital encoding of the object into a picture, into a two-dimensional picture that we can look at. Okay? And in particular, we're going to think about a very simple piece of geometry, just a cube. Okay? So we want to generate a somewhat realistic drawing of a cube, and we want to do that algorithmically, not just by sketching it on paper. So we have these two key questions we'll have to answer. One is a modeling question. How do we describe or encode the cube on the computer? And then we have a question of rendering. That word rendering will come up a lot in this class. When we say rendering, we mean something about how do you take a digital description and convert it into a concrete picture. So how do we visualize the model of our cube? Okay, so just to get us going, this is obviously a simple example, but just to get something concrete, let's say that we have a cube that's centered at the origin, 0, 0, 0, in three-dimensional space. It is a 2 by 2 by 2 cube, 2 wide by 2 tall by 2 deep, okay? okay and the edges of the cube are going to be parallel to the x y and z axes it's kind of an axis aligned cube so the first question we could ask you know how how do we encode this? In some sense, we've already encoded the cube. We've given a description of the cube that's pretty precise. We know where it sits in
space. We know how big it is. We know how it's oriented. Okay? But this description doesn't generalize very well. If we want to start describing other objects, like a face or a car, it's not really enough to just give the position, the rough size, and the orientation. Somehow we're going to need to encode more detailed information about the shape in order to eventually make a picture of it, to render it. in order to eventually make a picture of it, to render it. So, maybe think for just a moment, what's a more explicit description I can give of the cube that really spells out in great detail what its geometry looks like? what its geometry looks like. What kind of information might I encode? Okay, well it's a little bit of a tough question. It's kind of an abstract question. But let's try this. So one thing we could do that's pretty concrete is we could say, first of all, what are all the corners, or what are all the coordinates of the corners of the cube, or the vertices of the cube? of the corners of the cube, or the vertices of the cube. So given this description of the cube, we should be able to figure out where in space its eight corners live. Okay. And what might be an example? Maybe think about what's the coordinate for just one of the corners? Can you think of any one? Okay, so if you think about this for a second, you think, okay, it's 2 by 2 by 2, but it's centered at the origin, so it's kind of a radius of 1. And so at least one of the corners will be at 1, 1, 1. Another one might be at minus 1, minus 1, minus 1. And then we have all the other possible combination of signs. We have three coordinates, two possible signs, 2 to the 3 is 8, so we get the 8 vertices of the cube. And kind of the point here is, again, if we think, okay, this was easy for the cube and it seems almost unnecessary for the cube. But again, if I was trying to convey the shape of a face or a car, I could also list a a bunch of points on that shape. So the point here is that we have a description that will generalize to more interesting geometry. Okay? But we're not quite done yet. So far, we don't really have a description that captures the essence of a cube. We just have these eight points that are floating out in space, right? And if I just show you these eight points, if I drew a picture now, if I rendered these eight points onto a two-dimensional canvas, it might be pretty hard to tell that this is a cube. I would just have these eight dots. It's sort of somewhat strange locations. So what additional information might I provide to really capture the shape of the cube. Okay, well there are a couple natural possibilities here. One that's pretty straightforward is I could list all the edges. I could say not only where are the eight corners in space, but which of those points are connected to each other by an edge of the cube. And one simple way to specify that would be to say, okay, an edge has two endpoints. So if I have these eight points, A through H, I'll specify an edge as just a pair of letters. So for instance, AB is an edge, CD is an edge, EF is an edge, and so forth. You can pause the video if you like and check me on this. What do you want got it right with one of these edges. Well, I guess what you should check to see is that only one of the coordinates changes when I go from one endpoint to another. The edges are aligned with the x, y, and z axes, meaning as they go from one edge endpoint to another, only the x coordinate is changing, or only the y coordinate, or only the z coordinate. Okay, but I'm pretty sure that those are all the right edges for my cube. So, we now have a digital description of the cube. It's no longer a concept. It's no longer just the word or the string cube, but it's a very, very explicit digital encoding of the cube. And we could encode this if we like all the way down to binary data, right? Rather than these text strings that we see on the slide. strings that we see on the slide. Okay, but this is what we mean by a digital encoding of the geometry or the scene that we're working with. So we're kind of done now with the modeling task. It was a little bit of a pain, right? We probably in general don't want to describe our models this way. And later on in the semester, we'll see how to build tools that give us a lot more ability to easily design and model three-dimensional objects. But for now, we have a reasonable digital description of our cube. we have a reasonable digital description of our cube. So the next question is, how do we draw this cube as a flat 2D image? How do we render it? Well, what do you think? I mean, the basic problem here is we have three coordinates for every vertex, but if we want to draw it on a 2D piece of paper, we need two-dimensional coordinates. So how could we take these three coordinates and turn them into two? That's our basic rendering problem, at least for

today. What do you think? Okay, well, there are a lot of possibilities here. For one thing, we could just throw away one of the coordinates, right? We could just say, if I want to plot a three-dimensional point on a two-dimensional plane, I'll just toss out the z-coordinate, and I'll just draw x and y. And that works okay, but we don't really get a three-dimensional sense of this cube if we do that, right? We would just see, well, what would you see in that case? Let's say I threw away the z-coordinate, I plopped these points in the plane and then I connected any pair of points that belong to an edge. Well, you'd just see a square, right? It'd be looking, it'd be like looking at the cube from the side but without any perspective. It would be this really boring image. So we want to do something a little more interesting. So our basic strategy is going to be to map the 3D points to 2D points in some way, in some interesting way. okay? And then we're going to connect those 2D points with straight lines. Is that an algorithm? Is that really something we can go and implement? Not really. It's not yet an algorithm because we haven't really broken it down into atomic operations that a computer knows how to do. But it's a good sketch of the algorithm that we want to design. Okay? So let's talk in a little more detail about how each of those steps is going to look. The first one is perspective projection. So we said that this really, really simple idea of just tossing out a coordinate, that's not going to give us a very interesting image. What we'd like to do is capture things that we know are true from our day-to-day experience of looking at the world. So one thing we know from walking around the world is that objects tend to look smaller as they get further away. This is the phenomenon of perspective. And, you know, it's such a natural thing. It's such a thing that is just obvious that most people never really think about, well, why does that happen? really think about why does that happen? Why is it that if I'm looking at a building downtown and I I walk up to the top of the hill? No, I mean, this is crazy, right? The building doesn't change. So, so why does it look so tiny? Have you ever thought about this? So let's really think about this. And one way we can get our heads around what's happening is to consider what's called a pinhole model of a camera. Actually, this is a real kind of camera you can build. So the way this works is you can imagine you have a cardboard box that's completely dark inside, and then you poke just one tiny little hole to let light in. And on the opposite side from where you poke the hole, you put a piece of film. Okay, so traditional film was a photosensitive material that had silver halide crystals in it or something like that. And the way it works is every time light comes in, every time a photon hits that film, it causes a chemical reaction to occur. Those little crystals get stuck to the film paper, and anywhere that light doesn't hit, the crystals don't get stuck. And so the stuff that didn't crystallize gets washed away. All the stuff that got stuck remains on the photo paper. Really, this is your negative, right? And it creates an impression of the brightness or darkness that was shining through that hole. That's how traditional photography worked before digital cameras came around. Okay? So, not super important, actually, all this chemistry and how the film works, but that's what's going on in your pinhole camera. What we want to understand, again, is this phenomenon of perspective. Why is it given this setup that models that are, or objects that are closer look bigger and smaller look further away. If I move the box away from the tree or toward the tree, why should the size of the tree change in my two-dimensional image? change in my two-dimensional image. Maybe this is a little easier to understand if we look at it from a side view. Okay, so the pinhole now is along this horizontal axis. It's on the right side of the box along the horizontal axis. And we can ask for any point P equals XYZ in three-dimensional space, so maybe the tip of some leaf on the tree. Where is that going to end up on the image? If there's green light kind of shooting off of that leaf you can imagine there's some light that bounces off and shoots off of that leaf into the camera what is the position it's going to end up at on the film the position Q equals UV. So we're going to say the point P is where it starts, Q is where it ends up, but where exactly does it ends up vertically. So if V is the vertical coordinate on the two-dimensional film, how could you figure out the V coordinate of this point Q based on this little diagram. Okay, well, some of you may have thought, ah, actually, this is something about similar triangles. I can notice that there are sort of two similar triangles in this picture., and there's the one on the left with base 1 and height v. Okay, so what do we know about similar triangles? Similar means, for instance, that the ratios of edge lengths are the same. So we know that y
over z is the same as v over 1. Right? Y over Z is the same as V over 1. So if we assume that the camera has unit size, it's a 1 by 1 by 1 box, and its pinhole is hole is at the origin of space, 0, 0, 0, then we just need to solve the equation v over 1 equals y over z for the coordinate v. v and that's equal to y over z in other words the vertical coordinate on the film is just the slope y over z okay how do we get the other coordinate how do we find the other coordinate? How do we find the u-coordinate where the light hits the film? Well, that's easy. We just do exactly the same, but we look at the box from the top this time rather than from the side. Again, we'll get the same kind of picture with two similar triangles. We carry out the same calculation, and we'll discover that the horizontal coordinate is x over z. Okay, hopefully that makes sense. If not, pause, rewind, take another look. But the important thing is the result we got agrees with our day-to-day intuition. What do we notice? That if we have this point on the tree, x, y, z, and z starts getting bigger and bigger

and bigger and bigger, what happens to x and y? I'm sorry, what happens to u and v? Right? u and v start getting smaller as z gets bigger. As the tree goes off into the distance, as z gets bigger. As the tree goes off into the distance, the coordinates on the image plane shrink. The apparent size of the image shrinks. As z gets smaller, x over z gets bigger and y over z gets bigger and the image gets bigger. Okay? So this actually gives some real concrete explanation for why it is that things look bigger when they're closer and smaller when they're far away. The same kind of thing is happening in your eyeball. Your eyeball is not quite the same as a pinhole camera, but it's not so different either. You have a pupil, which is like the pinhole, and you have a retina on the back of your eyeball, which is kind of like the film in your camera. And exactly the same kind of calculation explains why things get smaller in the distance. Okay. The good news too is that this is an easy calculation. If I want to go from 3D to 2D now, all I have to do is divide by Z. So now we can really go ahead and draw our three-dimensional cube as a two-dimensional image. And we can do this by a completely algorithmic procedure, something that doesn't require that we're good artists. We don't have to have any background in painting or sketching or anything like that to get this illusion of perspective. We just have to repeat the same symbol algorithm 12 times, and I will encourage you to actually do this, because it's a lot more fun to see that this works out at home, to actually do the calculation, draw the picture and see that yes, indeed the algorithm works, than to just sit there and, you know, nod your head and believe that it all happens the way that it should. Okay. So what you're going to do is repeat the same very simple algorithm 12 times, once for each edge of the cube. This is a little monotonous, so if you want to write a little piece of code, you know, by all means, to carry out these calculations for you, or you can do it with a calculator, it doesn't matter. these calculations for you or you can do it with a calculator it doesn't matter so for each edge we're going to imagine that we have a camera sitting at 2 3 5 okay so some interesting location in space not zero zero zero just because we want a better view of our cube view of our cube, and we're going to convert the three-dimensional coordinates x, y, z of each endpoint to two-dimensional coordinates u, v in the following way. The first thing we're going to do is we're going to subtract the camera location c from the vertex location xyz. So we have our vertices listed at the bottom. Those are our capital xyz. Right? So we're going to do, for instance, 111 minus 235. minus 235 and that gives us the 3d locations of those points relative to the camera you see how that works if our camera was sitting at the origin 0, 0, 0, then the vertices we've listed are already the coordinates we care about. But if we move the center of the camera a little bit, we also have to change the coordinates of the vertices. We're working in a different coordinate system now centered around the camera. OK? So subtract c from capital xyz to get lowercase xyz. Then how do we go from 3D to 2D? We divide the little x and y coordinates by the little z coordinate to get u and v. And if you want to plot this on a piece of graph paper, you'll make your life a little easier by just writing it out as a fraction. Okay? Once you have those coordinates, just draw a line, let's say on a piece of graph paper, between the two coordinates you computed. That's the little subroutine that that'll draw a single edge of the cube. Your overall rendering algorithm then is to just do a loop over all the edges and execute this subroutine 12 times. Okay? And so then you could go ahead and draw this on graph paper. In fact, if you don't have graph paper, but you do have a printer, you could pause the video and you could print out this moment of the video on your printer. Lots of ways you could do it, or you could draw in a drawing program on your computer but give it a shot okay and if you go ahead and do this maybe you make a mistake or two but that's okay probably you're going to get an image that looks something like this. Okay, so that is the output of our first graphics algorithm. It turns a digital description of a cube into a real two-dimensional image. And you can check here if you got the same 2D coordinates. But what's really cool about this is it looks pretty good, right? This really does look like a three-dimensional rendering of a cube. three-dimensional rendering of a cube. And we did it completely by an algorithmic deterministic procedure. We let math do all the hard work for us. Eventually, we'll let code and computation do all the hard work for us. So if we now wanted to make an animation of this cube, if we wanted to imagine the camera is moving around the world, we would get this beautiful perspective correct image of the cube from different points of view. It's pretty cool. the cube from different points of view. It's pretty cool. Here's just a fun picture produced by a previous class where everybody contributed a different edge. So 12 different students sat and computed 12 different edges. And hey, they got it right. Okay? So again, the point here is you don't need higher level reasoning to come up with these images. It's really about breaking down the rendering process into lower and lower level subroutines and components that can be used to execute this kind of drawing algorithmically. Okay, so success! We turned purely digital information into purely visual information using a completely algorithmic procedure. That's what we want to do in this class in general. But actually, it's a little bit of a lie, right? There was one part of our process that was not really very digital. It was pretty analog, let's say, because we didn't really talk about how you would draw lines on a computer. We relied on maybe drawing lines with a pencil on paper. Okay? So, to even start thinking about this question, how do we draw lines on a computer, start thinking about this question, how do we draw lines on a computer, we have to start talking about how an image is

represented on a computer. And in fact, you can start getting an answer to this question if you just just come closer. Come, come closer to the screen. Come really, really, really close to the screen. Okay. So if you came really close to your computer monitor, and especially if you have a low resolution monitor, you're going to see that the display is actually not a continuous image like you might see in the real world but it's actually this grid of little picture elements. And they look something like this. So they look like these little blocks of lights that have been turned on or turned off or turned on to varying degrees that generate the image. generate the image. And actually, you notice even some interesting and weird stuff about these pixels, which is they look like they're striped in three colors. They have red, green, and blue stripes, like a little flag, right? So they're not just a continuous color. And we'll talk a little bit later in the class about color and how it is that little red, green, and blue lights manage to produce all sorts of different colors. But it's worth knowing that this is kind of what's going on when you look at a screen, whether it's your laptop or your cell phone or whatever it is. So even though displays are pretty complicated, we can come up with a pretty simple abstraction for displays or for digital images that's good enough for working with algorithms. Okay? And this abstraction is to say that we have a raster display, meaning things get drawn or rasterized onto a grid with some number of rows and columns. The image is essentially comprised of the values that are stored in this grid, the numerical values that are stored in this grid. and each little cell of the grid represents some color, some color value. Okay, again we'll talk a little bit later on about how exactly to encode colors, right? If I said please give me the digital value, the numbers encoding the color orange or green, you may not know exactly how to do that. But later on, we'll see there's a very straightforward mapping between these colors, or maybe sometimes not so straightforward mapping between these numbers and color values. Okay? But this is what you should think. When you think of a digital image, you think of a grid with some numbers in it. Maybe the easier version is to think not about colors, but just a grayscale or black and white image where you have a one if the pixel is white and a zero if the pixel is black or 0.5 if the pixel is a medium gray. Okay, so that's how we think about a digital image. The next question is, if we want to draw a line into a digital image,

which of these pixels should we turn off or on? If you hand me only the two endpoints of the edge, you say the UV coordinate of one endpoint and the UV coordinate of another endpoint, which pixels in the grid need to get turned on. That process of turning the high-level description of the line segment into a set of pixels is called rasterization. Rasterization is a process of converting a continuous object to a discrete representation on a raster grid or pixel grid. So how might you answer this question? The question at the top of the slide. What pixels should we turn on to depict this line? What do you think? Well, there's a lot of different answers. In fact, there's no right or wrong answer here, but one reasonable answer might be, how about just turning on any pixel that touches the line? So if the line passes through that pixel at all, that's going to be part of the image for the line. Okay, that works okay. You might notice that in this example, the line looks maybe a little chunky. There's some pixels that are barely touched by the line, but still the whole pixel gets turned on. So is that going to look funny? there's another rule, a rule that's actually used by a lot of real modern graphics APIs and hardware and so forth, called the diamond rule, which says you should turn on a pixel not if the line passes through the square box around the pixel, but if it passes through the diamond made by connecting the midpoints of the pixel. This is called the so-called diamond rule. Okay, so you notice compared to the previous rule, or simpler rule, this line is a little thinner. Right? Pixels that are just grazed by the line don't get turned on, and maybe it looks a little nicer. You could also ask, you know, is there really not a right answer is it really not a best way to rasterize a line well one way to answer this question is to say well what properties would you like that algorithm to have what should be true about a line rasterization algorithm. For instance, you might say, oh, if the two endpoints are in the same location, only one pixel should get turned on, or maybe no pixel should get turned on. You could come up with criteria for what it means to have a good line rasterization algorithm first, and then try to figure out if there is a good rule or a good algorithm for drawing lines that satisfy all those criteria. Another perspective that we're going to talk about a lot when it comes to rasterization is to think about coverage. So you could say, how about we turn on a pixel with intensity proportional to what fraction of the pixel is covered by the object we're drawing. Now this is a weird question for a line because a line is, it's like infinitely thin, right? It covers zero percent of any pixel. any pixel. So maybe you say, well, okay, but I'm going to give my line a width. I can think of my line, not as an infinitely thin line, but actually a little skinny, skinny rectangle that has some width. Ah, so now I could start asking, well, what fraction of each pixel is covered by this skinny rectangle? And lighting up the pixel by that amount, that starts to sound pretty reasonable. And it also sounds nice because I can deal with things like thick lines and so forth. Okay? But I'm really getting ahead of myself. I just want to stress that in computer graphics, there's often many, many ways to do things. You shouldn't just believe that there's one right way. Lots of different possibilities to explore. right way. Lots of different possibilities to explore. So let's go back to

this simple view of a line as this thing that has two endpoints and is somehow no thickness. Okay. And let's say we care about something like this diamond rule. So if we do that, how algorithmically now could we find the pixels that satisfy a given rasterization rule? Let's say we've already decided what we think is the right way. How do we actually turn on the right pixels? So one stupidly simple algorithm would be we just check every single pixel in the image to see if it meets our rasterization rule. Okay? So, for instance, let's say I have an 8K monitor, you know, millions and millions of pixels in my huge high-resolution monitor, and I just want to draw one little line segment, one short line segment. Let's say my image, in fact, is n by n for some very large n, and I'm going to rasterize it by going to every single cell in my pixel grid, every single pixel in my pixel grid, and checking, does the line pass through the diamond in this pixel? In terms of n, the size of the image, how many operations do I have to do to do this rasterization? Okay, I have to do n squared operations. I have to check all n times n pixels in the image. I have to check all n times n pixels in the image. And this is a little weird because I expect that a line or a line segment is going to cover at most about n pixels. A line is a linear kind of thing. It can't cover the whole image if it has no width. So it feels like I'm doing way more work than I need to be doing. There must be a way to do better. There must be a way to do an amount of work proportional to the number of pixels in the output, rather than the number of pixels in the image. Okay? So this leads to an idea of incremental line rasterization. So let's say a line is represented, again, with two endpoints, u1, v1, and u2, v2. There they are. And we're going to, for simplicity, assume that these are integer coordinates, and those integer coordinates correspond to the centers of pixels. onto the centers of pixels. The slope of the line, hopefully you remember about slope. Slope is rise over run. In this case, it's V2 minus V1 over U2 minus U1. Okay, and we want to light up the pixels along this line segment. How do we do that? Well, let's consider an easy special case. Just one of many things that could happen. Let's assume that u1 is less than u2 and v1 is less than v2. And v1 is less than v2. So v1 is at the lower left and v2 is at the upper right as in this image. In this case, the slope is between 0 and 1. There's more change in x than there is in Y. And so we can come up with a pretty simple little algorithm for walking along the line and figuring out which pixels to light up. In particular we can start at height v1 and then going from left to right from U1 to U2 we're just going to increment one column at a time. In each column we're going to increase our height by the slope s. So for a unit change in u, v changes by s. We're going to draw the pixel, or we're going to light up the pixel at u and round v. So round v is going to give us the integer closest to our current v value. And then repeat. So we move from left to right. We gradually increment v and we just draw the closest integer to our current v value until we finally reach the end of the line segment. Pretty straightforward. This is not a general algorithm because in general s is not between 0 and 1. We have to think about these different cases. Okay, what if there's more change in V than there is in U? Well, then we need to write a little piece of code that looks very similar to this one. Right, but where some U's and V's are swapped out around and so forth. But that's the basic idea. So, there you go. We have a concrete algorithm now for actually drawing the lines. Okay. And in fact, this algorithm is pretty easy to implement. It's not by the way, how lines are really drawn in modern graphics software or in modern graphics hardware. There are a lot more clever and interesting things you do to draw lines. You want to handle things like lines with width, as we've already talked about. But this is a really nice algorithm, a nice idea of incremental line rasterization that actually does show up in lots of other algorithms in computer graphics. computational procedure for turning our digital information, just a list of vertices and edges, and maybe the camera grid, on an image grid. Okay? So really, just from this one lecture alone, you really could go home and write up pretty easily a piece of code that draws three-dimensional images or draws three-dimensional models into a two-dimensional image. It's pretty cool, right? And now you have the freedom. You can change the input. You don't have to draw the cube. You could mess around with the vertices. You could mess around with the edges. You could draw all sorts of interesting, cool 3D models. Okay, so this whole exercise that we've just gone through is really what computer graphics is all about. How do we enc that digital information into an actual image or sensory stimuli? So, so far, we've only talked about a very, very simple line drawing of a cube. Okay. If we want to generate more realistic pictures, we're going to need a much richer model of the world. We're going to need to talk about geometry. We're going to need to talk about materials. How do you distinguish leather from paint from carpet or whatever? We need to talk about lights, different kinds of illumination. We need to talk about camera models. We need to talk about motion. We need to talk about all sorts of things and all the algorithms to translate that data into images. And we will see all of this and more as our course progresses. Just to give a little taste of how things will go in this class, how will you learn all this stuff? Of course, part of it will be by lectures and conversations and so forth. We're also going to learn a lot by making and doing. So a big part of your assignments in this class is going to be building up what's called the Scotty 3D package. This is a software package that's kind of a toy software package that's meant to emulate kind of a toy software package that's meant to emulate real 3D packages that people use for creating visual

effects and so forth. So we're going to give you kind of a skeleton code for Scotty 3D, and you're going to fill in all the essential algorithms. the essential algorithms. And this is broken up into four major assignments. The first one is on rasterization, on the kinds of things that we've been talking about today, but not just drawing black lines on a white image, really drawing much more rich and interesting graphics that you might find, for instance, in what's called an SVG file, a standard vector graphics file. And why do you want to do this? Well, because it lets you display all sorts of beautiful images and animations, like you might see around the web, like you might see on posters, and again typography, just text itself, involves a process of rasterization. The next assignment isn't going to be about geometric modeling. So we've already seen a little bit of this today. Okay, we wanted to model the cube, so we input the vertices and the edges. We're going to talk a lot more about tools to generate and design interesting 3D shapes, and then to process those shapes. You created the mesh in one way, but a certain algorithm needs it to be broken up in a different way. The motivation for tools like these is to create models like these. Models coming up not only, again, in entertainment, but also in product design and engineering and in all sorts of other things. in engineering and in all sorts of other things. And even with the simple subdivision modeler that you're going to build in Scotty 3D, you're actually going to be able to create some pretty sophisticated models. People have made some really cool 3D models with their own 3D modeler in previous years. The third part has to do with our rendering process. We saw a, again, really simple rendering process of drawing lines, but the third part is going to be about photorealistic rendering. If I have a description not only of geometry, but also of lights and materials and cameras, how can I predict what that scene would look like in real life? and hitting a surface and shining this way and that way to create this beautiful reality that we live in. And this kind of technology gets used all over the place again in the movies, of course, to make virtual environments look real, but also in architecture. You've designed a house and you want to see before you build it, hey, does it look right if the light is shining in through the south window? Does the room look the way that I'd really like it to? Or if I'm doing some kind of product design or engineering, how can I get a sense of how this thing will look before I even build it? That gives you a lot of freedom to design and explore without having to sink millions of dollars into actually building the thing. And finally, we're going to want to go beyond static images and talk about motion, talk about animation. So just like we need digital encodings of 3D geometry, what kind of information do we need to encode to describe to the computer an animation. Okay, and again, the motivation is lots of different things, animating characters in movies, of course, but not only the bodies of the characters, you also have to think about complicated things like, how does the hair of a character move around and respond to its environment? And this starts to get into connecting computer graphics to physics and actually taking laws of physics that you might have learned in your intro classes and breaking them down into numerical procedures that you can actually use to approximate how things move in the real world. Animations show up also in other places like robotics. So here we're seeing an example of sort of animating a robot, but that animation is being used to actually control a physical object. So talking about digital encodings of motion is really, really important beyond just, you know, making animations for the movies. Okay, so that's it for today. Okay, so that's it for today. Next time we're actually going to start diving into a review and preview of the kind of mathematics that you'll need for this class. So the prerequisites are some background in linear algebra and vector calculus. But there are a few things that we'll need to review. There are a few things that show up a lot in computer graphics that might not quite have been emphasized in the courses you took previously. So we want to make sure you have the right tools to guarantee your success as you go through this course. All right, see you then.