slide1:

Hello everyone, and welcome back.

Today, we will give the last MatLab lecture to explore how it can be used to demonstrate CT image reconstruction, that you've been learning in this course.

The image you see here is an example from GE's "Revolution CT." This technology allows us to capture extremely high-resolution images of the human body. In this case, you can see a detailed 3D view of the skull, including the blood vessels—arteries, veins, and even tiny capillaries.

If you're curious, the link at the bottom of the slide will take you to the website where this example came from. They have several other fascinating images that show how CT technology is evolving to provide ever-sharper and more detailed views inside the human body. These advances are not just technical achievements—they are vital for clinical use, helping doctors diagnose and treat patients with much greater precision.

As we go through this lecture, I'll also remind you that all the MATLAB code examples we'll be using are available on the LMS. If you'd like to follow along or try them yourself later, you'll find everything there.

slide2:

Let's take a quick look at our course schedule to see where we stand.

As you can see, we've been moving steadily through the topics—starting with the basics, like introduction, systems, and Fourier series, and then building up to signal processing and MATLAB practice.

Currently, we are on track with the section on computed tomography, or CT. This means you already have the mathematical and signal processing foundation, and now we are applying it directly to medical imaging.

So yes, we are on schedule, and today's focus on MATLAB and CT fits exactly into the larger flow of the course.

slide3:

Now, before we dive into today's material, just a quick reminder.

If you'd like to follow along with the examples, please make sure you have MATLAB installed, along with the Signal Processing Toolbox and the Image Processing Toolbox. These are essential for running the code.

If you haven't set this up yet, don't worry—it's simple. Just download the zip file I've posted on LMS, extract all the files, and place them into your MATLAB folder. Once that's done, you'll be ready to run the scripts, either as we go through them in class or later on your own time at home.

Alright, with that setup in place, let's move on to the theory of CT.

slide4:

Now, let's step into the theory of CT.

What I'll give you here is a general overview, much of which should feel like a review from our earlier classes.

The basic idea is this: CT uses X-ray projections taken from many different angles around the body. As the X-rays pass through, they are attenuated—that means they are weakened—depending on the different properties of the tissues they pass through. Dense tissues like bone attenuate more, while soft tissues attenuate less.

By collecting all these projections from multiple views, we build up enough information to start reconstructing what's inside the body. That's the foundation of computed tomography.

slide5:

So, let's look a little closer at the general theory of CT.

When X-rays pass through the body, different tissues attenuate the beam in different ways. For example, bone blocks attenuate X-rays much more strongly than muscle or soft tissue. This difference is exactly what gives us useful information.

Now, the challenge is that each X-ray image we take is really just a 2D projection. You can see that in the picture on the left. If we only had this single projection, we would miss most of the information about what lies behind or to the side of the structures.

So what do we do? We take projections from many different angles—like the images you see in the middle. Each projection adds a different piece of information.

Then comes the key step: we solve what's called an inverse problem. Mathematically, this is done using the Radon transform and its inverse. By combining all of those projections through the inverse Radon transform, we can reconstruct a detailed image of the body's interior.

That's what you see on the right: a 3D CT reconstruction where bones and tissues are highlighted in different colors.

In this class, to keep things simpler, we will mostly work with 2D imaging cases when applying the Radon transform in MATLAB. But remember, the same principle extends to full 3D CT in clinical practice.

slide6:

Now let's talk about the resolution of CT reconstruction.

Two key factors determine how sharp or detailed your reconstructed image will be.

First, the number of rays, or the number of X-rays passing through the object at each angle. This directly affects the radial component of spatial resolution—that is, how finely we can distinguish details along a line from the center outward.

Second, the number of views, or the number of different angles, is often denoted by theta. This controls the circumferential component of resolution—how well we can distinguish details as we go around the circle.

On the cartoon CT slice shown here, you can see both components: the radial resolution going outward from the center, and the circumferential resolution wrapping around the image.

So, to summarize: the more rays you use, and the more views you collect, the higher the resolution of your reconstruction. On the other hand, using fewer rays or fewer views leads to blurrier images with less detail.

slide7:

Now let's talk about the Radon transform.

The Radon transform is essentially a way to represent X-ray projection data, and the result is something called a sinogram. A sinogram is just an image that shows how the projections vary as we rotate around the object.

This representation also tells us something about the characteristics of the sample. For example, objects that are closer to the center of the field of view show up as regions of higher amplitude in the sinogram.

Here you can see the mathematical definition of the Radon transform. Don't worry about memorizing the equation—what matters is understanding that it describes how the function f of x and y, our object, is projected along different angles, which we denote by theta.

On the left, you see the same setup we discussed earlier, with X-rays passing through the object at angle theta. And on the right, you see an example sinogram. Notice the bright, wavy line: that corresponds to the yellow circular object in the field of view. Because it's closer to the center, its contribution to the sinogram has a higher amplitude, and it traces this curve as the angle changes.

So the sinogram is the bridge between the raw projection data we collect and the reconstructed CT image we want to build.

slide8:

So far, we've looked at the Radon transform. But to actually reconstruct an image, we also need the inverse Radon transform. This process relies on something called the Fourier slice theorem.

Here's the idea in simpler terms. If we take the one-dimensional Fourier transform of a Radon transform projection profile at a specific angle phi, that result is the same as taking the two-dimensional Fourier transform of the object and looking along a line at that same angle phi.

That's powerful because it means each projection we acquire gives us a slice, or a line, through the object's 2D Fourier transform. When we combine all these slices from projections taken at many different angles, we can fill in the complete 2D Fourier transform of the object.

And once we have the full 2D Fourier transform, we can simply apply the inverse two-dimensional Fourier transform to reconstruct the original image.

The diagram here shows this step by step. Each projection corresponds to one line in Fourier space. Change the angle, and you get another line, and then another. As we add up all these lines at different angles, the Fourier space is gradually filled. Finally, applying the inverse Fourier transform brings us back to a reconstructed image in real space.

slide9:

The next key idea is back-projection, which is the standard method for reconstructing CT slices.

Here's how it works: we start with the sinogram, which contains all the projection data. For each projection, we take its information and "back-project" it—that means we spread that projection back across the image space at the angle where it was acquired.

Let's look at this cartoon example. Imagine we have a large oval, and inside it are two smaller red ovals. If we take a projection from the top down, the sinogram shows two bright spots that correspond to those red ovals. When we back-project this data, we get a blurry reconstruction along that one angle.

Now, let's do the same from a side view. Again, the sinogram shows two high-amplitude areas, and when we back-project that projection, we get another blurry reconstruction, but this time along the side angle.

At this point, we have two back-projections. When we combine them, the overlap starts to suggest the locations of the two smaller ovals inside the larger one. Of course, with only two projections, the result is quite rough—in this case, it looks more like two squares.

But as we add more and more projections from many different angles, the combined back-projections gradually approximate the true shapes. With enough views, the reconstruction recovers the original sample very accurately.

slide10:

Now let's move from simple back-projection to filtered back-projection, or FBP.

As we saw in the last example, back-projection is the standard way of reconstructing CT slices. We take the sinogram, back-project each view, and combine them. But here's the problem: if we simply use unfiltered back-projection, the resulting image often looks blurry. That's because the projections overlap in a way that smears out the details.

To fix this, we apply a filter in the sinogram space before back-projecting. Filtering helps sharpen the edges and preserve the fine details that would otherwise be lost.

Look at the images here. In the top row, we have a sinogram and its back-projected image. Notice how the back-projection produces just a fuzzy, grayish blob with poorly defined boundaries.

Now, in the bottom row, the sinogram has first been filtered. After reconstruction, the image is much clearer—you can actually see the circular structure with sharp edges, and even a smaller circle inside it, which could represent something like a tumor.

On the right side, you see examples of different filters that can be applied, such as the Ram-Lak filter, the Shepp-Logan filter, and the Hamming filter. Each of these adjusts the frequency content of the sinogram differently, but they all serve the same purpose: to enhance contrast and reduce blurring in the final reconstructed image.

So in summary, filtered back-projection is one of the most widely used methods in CT, because it gives us images that are both accurate and sharp enough for clinical use.

slide11:

Here's something pretty cool to look at—this is filtered back-projection in progress.

What you're seeing here starts with the original sinogram. Then, as the algorithm processes each view, the back-projections are gradually added together. Step by step, the image begins to take shape.

It's almost like watching the reconstruction unfold in real time. As more and more views are included, the overlapping lines come together, and the original phantom—or the sample image we started with—slowly reappears.

This gives you a visual sense of how CT actually builds up an image: not from a single snapshot, but from many projections combined through filtered back-projection.

slide12:

Now, how do we use MATLAB to demonstrate these CT principles?MATLAB gives us two distinct simulation paths: parallel-beam CT and fan-beam CT. We'll explore both, step by step.

For our demonstrations, we'll work with three simple phantoms—that is, test images used to evaluate reconstruction methods:

Square-in-square phantom: a small bright square centered inside a larger dark square.

Circle phantom with a nodule: a large gray circle with a small bright circle near the edge—think of that dot as a mock "tumor."

Shepp–Logan phantom: a standard test object that comes with MATLAB. It's widely used in our field, so researchers can compare results fairly—if two groups reconstruct the same Shepp–Logan phantom, their images are directly comparable.

We'll first apply parallel-beam tools and then switch to fan-beam tools, so you can see how the algorithms behave on the same phantoms.

slide13:

Let's begin with parallel-beam CT.

In MATLAB, the main tool for simulating this is the Radon transform function, simply called radon. This function takes an image—our phantom—and generates its parallel-beam projections at different rotation angles.

In a parallel-beam setup, the X-ray beams are arranged so that they travel in perfectly parallel lines as they pass through the object and reach the detector. Each projection corresponds to summing the values of the image pixels along those parallel paths.

So when you hear me say "parallel-beam CT," think of it as collecting straight, evenly spaced rays, rotating around the object, and storing all of that information in a sinogram. MATLAB's radon function gives us exactly that.

slide14:

Now let's go deeper into how MATLAB's radon function works for parallel-beam CT.

Think of the simulated beams as rays spaced one pixel apart. These beams are projected from the source, travel in straight parallel lines, pass through the object, and arrive at the detector on the other side.

In practice, both the beams and the detector array rotate together around the center of the image. The amount of rotation is controlled by the angle theta, which we provide as an input to the function. So by sweeping through many angles, we gather projections from all around the object.

The MATLAB function looks like this:R, x p equals radon I, theta;

Here's what each part means:

R is the result—it contains the projection values, essentially the amplitudes of the rays.

X p gives the detector positions, if you need to keep track of them.

I is simply your input image, or phantom.

theta is the set of rotation angles you want to use.

So, in one line of code, MATLAB takes your image and simulates what the parallel-beam projections would look like at the specified angles.

slide15:

Let's look at our first MATLAB example for parallel-beam CT.

If you've downloaded the materials, the code is called E x 1 parallel dot m. You can try running it now or later on your own. Inside the script, at line 5, there's a variable called p type. By changing the p-type to 1, 2, or 3, you can switch between the three phantoms: the square, the circle with a small circle inside, or the Shepp–Logan phantom.

Here's what happens when you run the code. On the left, you see the image domain, in this case, just a simple white square. As the red arrow rotates, it represents the projection angle, which we call theta. Each angle contributes new data to the sinogram.

On the right, you see the sinogram domain. Notice how information is gradually added as theta changes. For the square, the sinogram has a symmetric, crisscrossing pattern.

If you switch to the second phantom—the large circle with a small white dot—the sinogram looks different. The small dot produces a wavy curve in the sinogram because the detector gets closer to and farther from the dot as it rotates.

Finally, with the Shepp–Logan phantom, the sinogram becomes much more complex. There are many structures inside that phantom, and each one leaves its signature on the sinogram. This complexity is much closer to what you would see in a real CT scan of the human brain, where multiple tissues and structures contribute overlapping patterns.

So the sinogram is really just another way of saying: here's how the object looks from every possible angle.

slide16:

Now let's try a simple experiment with the code.

In the script, there's a parameter called the theta step. This sets how finely we sample the rotation angles. For example, if the step is 1, MATLAB computes projections at 0, 1, 2, 3 degrees, and so on, all the way to 180. But if we change the step to 5, then MATLAB only samples at 0, 5, 10, 15 degrees, and so on.

Here's what happens when we make that change. The sinogram becomes much coarser. You can see it here for the square phantom: instead of smooth, continuous curves, the sinogram looks blocky, because we've skipped over many intermediate angles.

What does this mean for reconstruction? It means lower resolution. If we try to reconstruct an image from this sinogram, the result will appear streaky or blocky, because we don't have enough angular information to fill in the details.

So the takeaway is: to get a high-resolution reconstruction, we want a small theta step, which gives us many angular views. Fewer angles make the computation faster, but the image quality suffers.

slide17:

Now let's see what happens if we change the maximum theta value.

Previously, we collected projections from 0 all the way up to 180 degrees. But what if we stop early—say, at just 45 degrees?

When we do that, the sinogram is essentially truncated. Instead of showing information from a full half-circle of projections, it only covers a small portion of the angles. In other words, we've only scanned part of the sample.

What does that mean for reconstruction? If we only gather projections from 0 to 45 degrees, then when we try to reconstruct the image, we're missing most of the information. The result will be incomplete and inaccurate, because the algorithm never saw the object from the other directions.

So, limiting the maximum theta is like taking photos of a sculpture but only from one side—you lose the complete picture. To get a full and accurate reconstruction, we need projections across the whole angular range.

slide18:

So far, we've seen how to use the forward Radon transform with MATLAB's radon function. But to actually reconstruct an image, we need the inverse Radon transform, which in MATLAB is implemented as the function iradon.

This function performs back-projection for parallel-beam sinograms, and it can also carry out filtered back-projection if we choose.

Here's the format of the function:I equals I radon R, theta, interp, filter;

I is the reconstructed image, the output we want.

R is the sinogram, the input data we've collected.

theta is the set of rotation angles used.

Interp is the interpolation method—this tells MATLAB how to fill in values between samples.

The filter is the filter we want to apply for filtered back-projection.

The last two arguments, interpolation and filter, are optional. That means you could just write I radon R, theta to perform simple, unfiltered back-projection. But if you include a filter, such as Ram-Lak or Shepp–Logan, MATLAB will automatically implement filtered back-projection, giving you a sharper and more accurate image.

So, in short, radon simulates the projections, while iradon reconstructs the image from those projections. Together, they form the foundation of CT simulation in MATLAB.

slide19:

Let's walk through Example 2: Parallel-beam back-projection.

The code for this is called E x 2 parallel back dot m. Inside the script, line 32 runs a normal back-projection with no filter, and line 35 applies a filtered back-projection.

In this case, I've included the Hamming filter as an example. But MATLAB also supports other filter types, such as Ram-Lak, Shepp–Logan, Cosine, and Hann. You can even design your own filter and add it as an input if you want to experiment.

Now let's compare the results. On the left, you see the reconstruction of a simple white square using unfiltered back-projection. Notice how the image shows streaks and a kind of "cross" pattern—that's the artifact caused by overlapping unfiltered projections.

On the right, you see the result with filtered back-projection. The square is much cleaner, with sharper edges and less background noise. The filter removes the smearing and helps reveal the actual structure.

If you change the phantom, such as using the Shepp–Logan phantom, you may notice that different filters give slightly different results. Some filters are better for enhancing edges, while others help reduce blurriness or noise. This is why in practice, the choice of filter depends on what details are most important for your application.

So this example shows us the power of filtering—without it, back-projection images can be full of artifacts, but with it, we get reconstructions that are much more useful and accurate.

slide20:

Now, let's take a closer look at how changing the maximum theta or the theta step affects reconstruction.

By default, our code uses a maximum theta of 180 degrees and a step size of 1 degree. But what happens if we adjust those values?

On the left, you see the case where max theta is 180 and the step is 5. Here, we're still covering the full range of angles, but we're sampling them more coarsely. As a result, the back-projected image shows more artifacts—notice the diagonal streaks in the unfiltered image. Filtering helps reduce some of these, but the square edges are still less crisp compared to the default case.

On the right, we have max theta reduced to 90 degrees with a step of 1. Now the issue is different: we've cut the angular range in half, so the reconstruction is incomplete. The square looks more rounded at the edges, and you see stronger cross-like artifacts in the background.

So here's the key takeaway:

Increasing the step size (fewer sampled angles) makes the reconstructions less defined.

Reducing the maximum angle (smaller angular coverage) lowers resolution and produces incomplete images.

In real CT scanning, both of these parameters matter. More angles mean better images but also more computation and radiation dose. Fewer angles save time and dose but reduce quality.

slide21:

Now let's turn to the second type of CT geometry that MATLAB can simulate, which is fan-beam CT.

Unlike parallel-beam CT, where all the X-ray beams are perfectly parallel, in fan-beam CT, the rays spread out from a single source point, forming a fan shape as they pass through the object. This setup is much closer to how real CT scanners work today, because the source rotates around the patient while the rays diverge outward.

As I mentioned in class, fan-beam geometry introduces some additional considerations compared to the parallel-beam case, but it also gives us more realistic simulations. MATLAB provides dedicated functions to handle fan-beam CT, which we'll look at in the next slides.

slide22:

Now let's look at MATLAB's fanbeam function.

In this geometry, the X-rays don't travel in parallel lines like before. Instead, they spread out in a fan shape from a single point called the beam vertex. You can see that illustrated here: the source is at the top, and the rays diverge outward, passing through the sample.

On the opposite side, we have the detector array that records how much of the X-rays make it through.

There's also an important parameter here called D. This is the distance from the fan-beam vertex—that is, the source position—to the center of rotation of the sample. For something simple, like a square phantom, D might just be the length of the diagonal. For a human body, D would be the distance from the source to the center of the patient.

As in parallel-beam CT, both the source and detector rotate together around the object at different angles, which we denote by theta. By sweeping through all these angles, we collect the fan-beam projection data needed for reconstruction.

So, the fanbeam function in MATLAB allows us to simulate this geometry directly, bringing us closer to how modern CT scanners actually operate.

slide23:

Now let's look at the fanbeam function in MATLAB in more detail.

This function calculates projection data for a specified fan-beam geometry. Unlike the parallel-beam case, here the rotation angles are fixed from 0 to 360 degrees. That means you can't restrict the scan to only half a circle, like 0 to 180, or just 0 to 45. The fanbeam function always assumes a full rotation.

Here's what the function looks like:

F1, sensor pos 1, fan rot angles 1] equals fan beam P, D, 'Fan Sensor Spacing', dsensor1;

Let's break it down:

F1 is the fan-beam projection data, the main output you need.

Sensor pos 1 gives the positions of the detectors.

Fan rot angles 1 lists the rotation angles used.

P is your input image, or phantom.

D is the distance from the source vertex to the object's center of rotation.

'Fan Sensor Spacing' is a property name, which tells MATLAB you're specifying the spacing between detectors.

Dsensor 1 is the actual spacing value.

Those last two inputs are optional. You can run the function with just the image and the distance, and MATLAB will use default detector spacing. But if you want more control—say, to change the resolution or field of view—you can specify these additional arguments.

So in short: fanbeam gives you projection data over a full 360 degrees, with flexibility to adjust sensor spacing if needed.

slide24:

Now that we've seen how MATLAB generates fan-beam projections, the next step is to reconstruct an image from them. For that, we use the I fan beam function.

Here's what happens under the hood: I fan beam first converts the fan-beam data into equivalent parallel-beam projections. Once that's done, it applies the same filtered back-projection algorithm we studied earlier to perform the inverse Radon transform and reconstruct the image.

The function looks like this:

I fan 1 equals I fan beam F1, D, ‘Fan Sensor Spacing', dsensor 1);

Breaking it down:

Ifan1 is the reconstructed image, the output.

F1 is the sinogram data produced by the fanbeam function.

D is the distance from the source to the object’s center.

‘Fan Sensor Spacing' is the property name that tells MATLAB you’re specifying detector spacing.

D sensor 1 is the actual spacing value.

The last two inputs are optional—you can let MATLAB use default spacing if you don’t need to adjust it.

So, just like with parallel-beam CT, we have a pair of functions: fan beam to simulate the projections, and I-fan beam to reconstruct the image. Together, they allow us to model fan-beam CT entirely within MATLAB.

slide25:

Now let’s move to Example 3: Fan-beam CT.

If you run the script E x 3 fan beam dot m, you’ll see how MATLAB handles this geometry. Inside the code, at line 25, there’s a parameter called D. By convention, D is set to be slightly larger than half the diagonal length of the image. Then, at line 26, the D sensor value is set to 1, which specifies the spacing between detector elements.

On the left, you can see the fan-beam sinogram for a simple square phantom. On the right, you see the reconstruction using filtered back-projection. The square shape is clear and recognizable, but you may also notice small dots or speckles near the corners. These are reconstruction artifacts, and if you run this code on your own computer, you’ll probably see them more clearly.

For more complex phantoms, the sinogram looks similar to what we saw in earlier examples, but extended over a full 0 to 360 degrees.

The reconstructions in those cases often show diagonal streaks or artifacts around the edges. Even so, the important structural information inside the phantom—like the outlines of shapes—remains fairly accurate.

So, fan-beam CT in MATLAB gives us realistic results, but it also shows us that no reconstruction is perfect. Artifacts are always a part of the process, and learning how to recognize and minimize them is a big part of CT imaging.

slide26:

Now let’s experiment with sensor spacing in fan-beam CT.

In our code, this is the D sensor value on line 26. By default, we set it to 1. But what happens if we change it?

On the left, we set D sensor equals 0.5. This means the detectors are closer together, giving us finer sampling. The result is higher resolution—you can clearly see the square shape, with only minor artifacts.

In the middle, we keep the D sensor equal to 1, the default value. The reconstruction is still decent, but not quite as sharp as when we used the smaller spacing.

On the right, we set D sensor equals 5. Now the detector spacing is wide, so we lose a lot of detail. The reconstruction looks blurred and distorted—the square shape is almost unrecognizable, more like a rounded blob.

So the key point here is: resolution is directly related to detector spacing. Smaller spacing means higher resolution. Larger spacing means lower resolution.

In practice, of course, reducing sensor spacing increases data size and computational load. But if the spacing is too large, the resolution drops too much to be useful. Finding the right balance is part of designing and operating a CT system.

slide27:

Now let's test what happens when we change the parameter D in our fan-beam simulation.

Remember, D is the distance from the fan-beam source vertex to the center of the object. In our code, this is set in line 28. The default value is 191, which is roughly half the diagonal length of the image.

Here's what happens:

On the left, with D equals 191 (default), the reconstruction looks good. The square phantom is sharp, and the resolution is reasonable.

In the middle, when we increase D to 300, the resolution decreases. The edges become less clear, and faint artifacts start to appear in the background.

On the right, with D equals 500, the resolution drops significantly. The square no longer looks crisp—it appears distorted, and the background shows strong artifacts.

So the rule here is: the farther away the source is from the object, the lower the reconstruction resolution.

This simulation is useful not only for learning but also for system design. It shows that if we set the source too far from the patient, image resolution suffers. By keeping the source-object distance within a good range, we can achieve sharper reconstructions.

slide28:

That wraps up our MATLAB section for today.

You'll use MATLAB's radon and iradon functions to work with an ellipse phantom—generate the projections, form the sinogram, and reconstruct the image.

Please make sure your submission clearly shows:

the input ellipse image,

the sinogram you generated, and

The reconstructed image (try both unfiltered and filtered back-projection, and note which filter you used).

If you run into issues, double-check that the Signal Processing and Image Processing Toolboxes are installed, and verify your angle settings and interpolation options.

Great work today. Next time, we'll build on this and discuss how to interpret artifacts and improve reconstruction quality.

slide29:

And finally, let me remind you about your second homework assignment.

From the Green Book, please complete problems 1.14, 1.15, and 1.18.

In addition, there's an optional project for those of you who want to take on something more creative. As part of the Art_X initiative, you can design a concept for a portable CT scanner—for example, one mounted inside a self-driving car, or a CT scanner designed for a completely novel application.

To get inspired, I've included two references here at the bottom of the slide, which showcase some innovative CT scanner designs. These papers, including some from Dr. Wang's group, may give you ideas for how CT technology can be adapted beyond the traditional hospital setting.

Again, the design project is optional—but the Green Book problems are required.

slide30:

And with that, we'll wrap up today's session.

Thank you all for joining the class and following along. We've covered a lot—from the basic theory of CT, to the Radon transform, back-projection, filtered back-projection, and finally MATLAB implementations for both parallel-beam and fan-beam geometries.

Please remember to finish your homework assignments and, if you're interested, explore the optional design project for something creative.

That's all for today. Once again—thank you, and I look forward to seeing you next time.