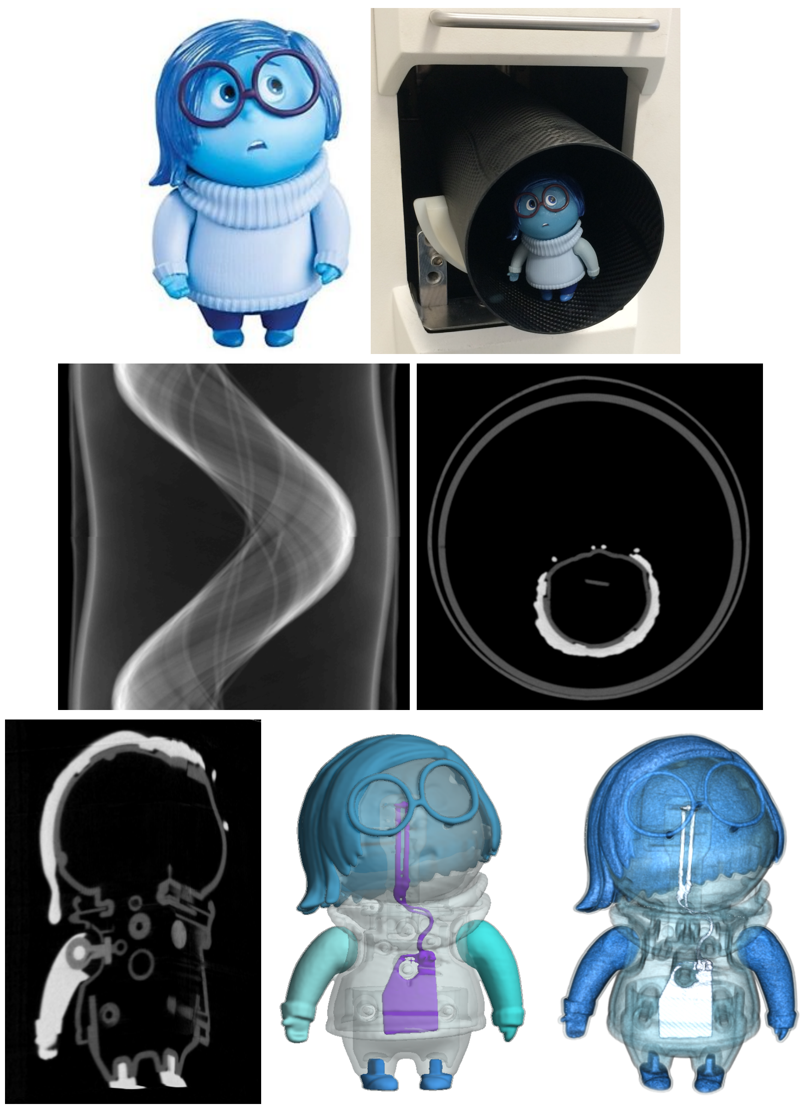
# GG2: CT reconstruction and visualisation: PYTHON version (Re-designed for remote working)

Graham Treece

May 2020



## 1 Introduction

The aim of this project is to follow the whole process of 3D medical imaging using X-ray Computed Tomography (CT), starting with a scan of a real object, right through to the creation of a new object from the scan. It will cover the physics of X-ray material interaction, the design of CT scanners, the maths behind CT reconstruction, the use of computer graphics in CT visualisation, and the preparation and issues involved in creation of physical models from CT data using 3D printing.

The first half of the project will introduce you to CT scanning and reconstruction by the development and testing of a simple CT simulator. Starting with an image defining the location and type of various materials, this will be 'scanned' with a typical CT geometry, and a set of X-ray source and material parameters, and finally the data reconstructed from the scanned measurements using the common technique of filtered back-projection. The emphasis is on learning what is involved, and experimenting with the various options (for instance resolution, interpolation techniques, etc.), rather than on writing lots of software: most of the components will be supplied.

The second half of the project will concern a variety of 'real life' scenarios. At this point a real object will be scanned, and you will be given the data for this object from the CT scanner. Each group will then be tasked with making the best use of the CT data for a scenario of their choice: for instance radiotherapy planning for cancer treatment, or design and assistance with an artificial limb replacement. Each scenario will involve some element of reconstruction (only the raw measurement data from the CT machine will be provided), visualisation (using any of a number of techniques), and preparation for model-building from the data (using 3D printer software). Again, the focus is on the investigation rather than writing software. Reconstruction of the raw data will make use of results from earlier in the project, whereas the visualisation and modelling will make use of the many free programs available for this task: selecting and learning the appropriate programs and techniques is a part of the project.

The project will finish with a brief presentation so that each group can show how they have addressed their task to the other students.

## 2 Overview and rules

An overview of the project and submission times is given in Fig. 1. For most of the project you will work in groups of three. In the first part of the project, everyone in the group will be expected to work on the CT simulator so everyone can acquire the necessary background knowledge. As the project develops each student will take on one part of the collective tasks but will be expected to work in collaboration and present the results as a group at the end. The CT simulation work will be based around some provided Python functions.

The overall arrangement of the project is as follows:

**Week 1** Introductory work on X-ray generation, scattering and detection, CT geometry and scanning. Development of a CT simulator and experiments using this simulator.

**Week 2** CT reconstruction using filtered back-projection and completion of basic CT simulator, which each group will demonstrate is working correctly. Extensions to simulator to include CT noise, beam hardening correction and Hounsfield Units, with associated experiments, leading to individual interim reports.

**Week 3** Real CT scan, provision of raw data, and start of task-based work.

**Week 4** Continue task-based work, group presentation of results and individual final reports.

During the project period, 6 hours per week are timetabled when you will be particularly expected to work on the project (9.00 – 11.00 each Friday, Monday and Wednesday). During each of these sessions, you will

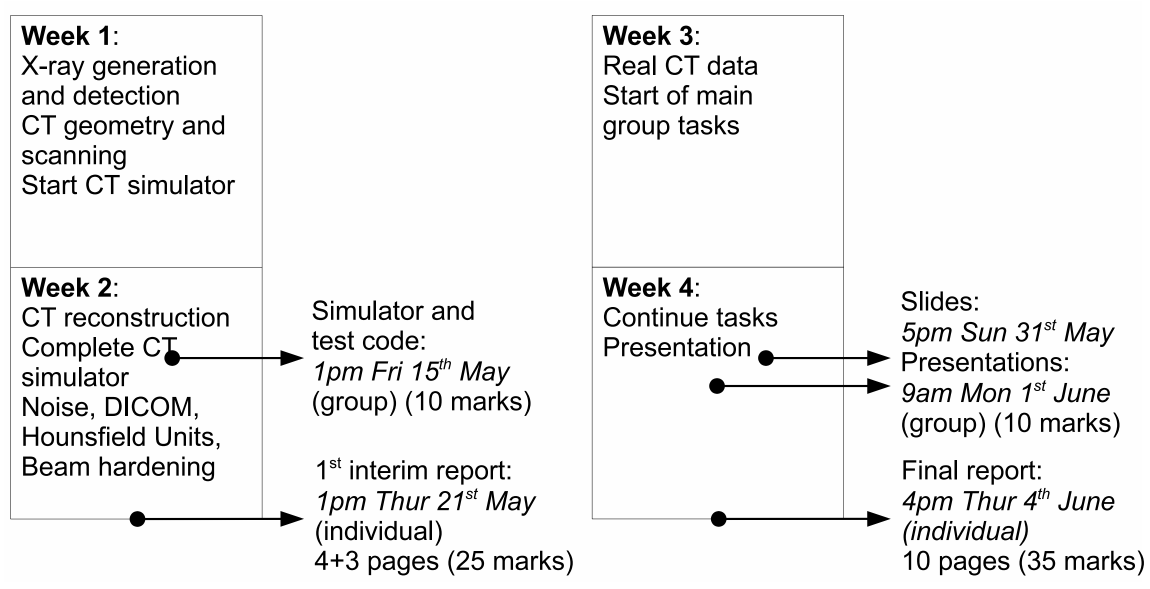


Figure 1: Overview of project stages and submission dates.

need to ‘sign-in’ by clicking on the relevant link for that session at the bottom of the GG2 Moodle page. Note that these links can only be clicked during the actual project session. You can check which sessions you have ‘attended’ (and which other elements of the course you have looked at) using the ‘Check your attendence and progress’ link at the start of the session list.

Demonstrators will be available to give guidance and help at any point during each of these sessions, via zoom. We will also arrange for a short chat with each group of three students at a specific point during each of these sessions, so we can all keep up to date. All students are expected to be available during all of these sessions, but there will be no specific penalty for missing them.

The project is designed for students to spend some additional 14 hours per week per project working on their own, though you may have more time than that this year, since only one project is running. This is expected to include appropriate background reading as suggested in this handout, continuation of experiments and tasks, preparing presentations and writing reports. Clearly most of this will involve working at a computer: it is your responsibility to **keep appropriate backups**: whilst it is understood that internet connections are not always completely reliable, computer issues are not regarded as an acceptable excuse for the complete lack of work, or for late submission of reports.

Reports and presentations are detailed in this handout at the appropriate stage. There is a **penalty of 3 marks per day** or part thereof for late submission of the group or interim report. **Final reports must not be late**. In summary there will be:

- Group submission containing evidence that the CT simulator works, consisting of the **simulator and test code** from each group, due by 1 pm Friday 15th May. Feedback on this will be given for the session beginning 9 am Monday 18th May. **10 marks in total** (awarded as a group).
- Individual **interim report** due by 1 pm Thursday 21st May, summarising the main experiments so far. Maximum 4 pages + 3 pages for appendices (for additional figures and tables only). **25 marks** (awarded individually).
- A brief **group presentation** during the last session beginning 9 am Monday 1st June of no more than

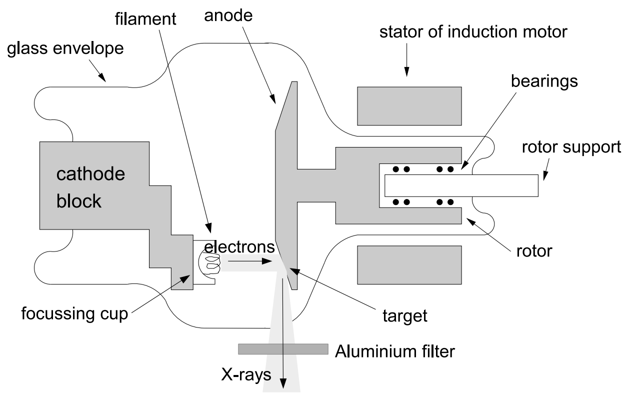


Figure 2: Schematic of a typical X-ray source.

10 minutes per group. This is a chance to show each other the results of the task each group has taken on. **10 marks** (awarded as a group).

- Individual **final report** on the task work, due by 4 pm Thursday 4th June. Maximum 10 pages in total. **35 marks** (awarded individually).

In preparing reports, students are required to adhere to the page limits. The interim and final reports are to be written individually. However, the simulator test code and the presentation of your task work should be organised by the whole group. Further details are provided elsewhere in this handout and in the online document 'Third Year Project Guide'. However, note that the 'Third Year Project Guide' only suggests typical requirements: it is this document, the GG2 project handout, which you need to adhere to.

In each section, the background material is designed to give a context for what you need to learn and also the important terms which need to be understood. The intention is that you **research for yourself** to discover more: these are all fairly fundamental concepts and there is generally lots of information available on them from various sources. Where this is not the case, specific sources are given as a starting point.

## 3 X-ray generation, scattering and detection: first part of week 1

By the end of this section, you should know roughly how X-rays can be generated and detected, and how they are attenuated by various materials relevant to clinical imaging. You should also have developed the provided simulator such that, given an X-ray source energy distribution and an amount of various materials, it can calculate the residual X-ray energy after passing through these materials.

### 3.1 Background

CT scanning depends on the interaction of X-rays with materials: X-rays are produced and sent through the subject in various directions, and different materials affect these X-rays in different ways. The residual energy after the X-rays have passed through the subject is measured, and this can be used to reconstruct certain material properties of the subject in 3D. More dense materials attenuate the X-rays more strongly, so it is mostly the density of the material which is seen in the reconstructed data.

X-rays are at the extreme end of the electromagnetic spectrum, with very low wavelengths (much less than a nanometre) and very high energies (more than 10 kilo-electron volts (keV) per photon). Figure 2 shows a typical technique for generating them: an anode (often made of tungsten) is bombarded with electrons at

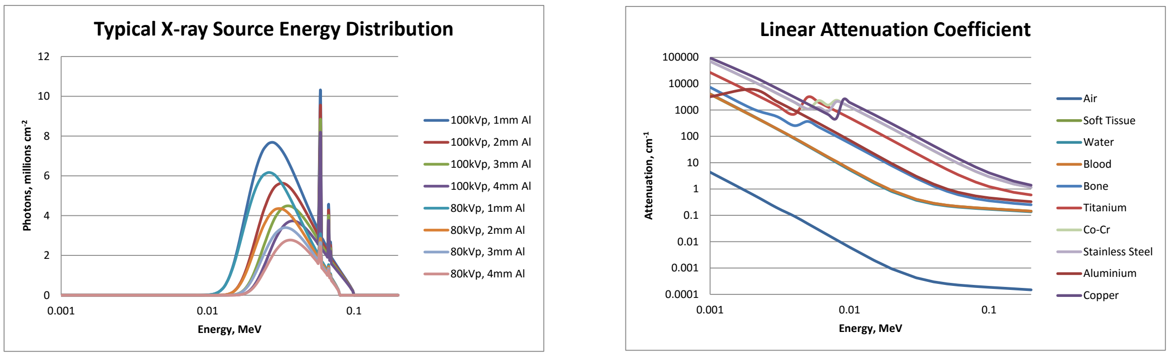


Figure 3: Typical source energies (left) and attenuation coefficients (right) in clinical imaging.

a particular energy, given by the potential difference between the anode and the cathode: this is actually measured in electron-volts, but often abbreviated to kVp, i.e. ‘peak’ keV.

Some of the electrons interact with the anode in a process called *bremsstrahlung* to generate photons of various energies. Hence the generated X-rays actually have a distribution of energies, as shown in Figure 3, though these energies cannot be greater than the kVp of the electrons. As we will see later, it is a good idea to reduce the range of these energies, so a ‘filter’ (i.e. a chunk of material, often aluminium, of a particular thickness) is usually placed in front of the X-ray beam, which has the effect of removing much of the lower-energy radiation. See Fig. 3 for the effects of using 1 mm to 4 mm of aluminium (Al) on two sources at 100 kVp and 80 kVp.

Since the number of electrons is proportional both to the tube current and also the amount of time that the X-ray beam is active, the X-ray ‘dose’ is usually expressed in terms of these two quantities multiplied together. A typical setting would be 100 mAs (milli-amp-second) which is equivalent to 100 mA for 1 second, or a greater current over a reduced time: the aim is always to minimise this dose in order to reduce the amount of radiation the subject receives.

The X-rays interact with material in a variety of different ways, but the key ones of interest in medical imaging are *Compton scattering* and the *photoelectric effect*. In the former, higher energy photons collide with a free or an outer shell electron, which escapes with some kinetic energy, and the photon then continues with reduced energy. In the latter, lower energy photons can eject an electron from an inner shell and be completely absorbed as a result. In both cases, either the reduced energy photon, or the ejected electron, can continue to interact with the surrounding material.

Both of these lead to a reduction in X-ray energy, and all interactions are summarised in terms of the *linear attenuation coefficient* $\mu$ such that:

$$I = I_0 e^{-\mu x} \tag{1}$$

where $I_0$ is the incident energy, $x$ is the thickness of material in cm, and $I$ is the reduced energy after passing through this material. For Compton scattering $\mu \propto \frac{\rho}{E}$, whereas for the photoelectric effect $\mu \propto \frac{\rho}{E^3}$ (where $E$ is the energy). The linear attenuation coefficient can also be expressed per material density $\rho$, in which case it is called the *mass* attenuation coefficient, $\frac{\mu}{\rho}$.

Since $\mu$ is summarising lots of different and complex effects, it varies with the initial X-ray energy: Fig. 3 shows some typical examples for different medically-relevant materials. Hence to get the *total* residual energy after passing through a subject, we have to sum the intensity at all photon energies, as well as the linear attenuation for all materials (presuming a discrete energy distribution to match the simulator we will be developing):

$$I_{\text{tot}} = \sum_E I_{0,E}\, e^{-\sum_m \mu_{m,E}\, x_m} \tag{2}$$

where $E$ are the energies, $m$ the materials and $\mu_{m,E}$ the linear attenuation for material $m$ at photon energy $E$. This can also be represented as the product over different materials:

$$I_{\text{tot}} = \sum_E \left\{ I_{0,E} \prod_m e^{-\mu_{m,E}\, x_m} \right\} \tag{3}$$

The residual X-ray energy can be detected by various means, but one of the most common, called a *scintillator*, depends on turning the remaining photon energy into light, which can then be turned into electricity using a photo-diode array in much the same way as a digital camera. The scintillator and photo-diode usually also have a *collimator* in front of them: this is an array of very fine holes drilled into an appropriate material, which ensure that the detector only responds to photons which travel directly towards it, rather than multiply-scattered photons which might arrive from other angles.

## 3.2 Resources

The following resources should help with this and subsequent sections. They can be loaded by moving to the folder containing the provided code, starting `ipython` in a terminal, and then typing the command `'run ct_include.ipy'`. `ipython` is an interactive version of python which allows you to run python commands one at a time, rather than having to write a python script. You will need to have installed the `pydicom` library in addition to `matplotlib, numpy, scipy` which will probably already be installed. This script also forces functions to be re-loaded each time they are run, which makes editing and testing them easier.

The easiest way to use python is to install the anaconda distribution, and from this also install Visual Studio (VS) Code, which is a lightweight coding environment from which code can be edited and debugged, and ipython can be run. Instructions for installing these are on the Moodle GG2 site: note that in Windows you may need to run the software as an adminstrator in order to install additional libraries. Whilst you can then just run VS Code directly, it sometimes works better to run the anaconda `navigator` first, then run `VS code` from there.

Alternatively, you are welcome to run a local copy of Jupyter, or create your own python scripts directly. In this case, you can look at the contents of `ct_test_example.py` to see what modules you are likely to need to import. Scripts run in this way do not need to use `ipython`.

**`mass_attenuation_coeffs.xlsx`** is a spreadsheet containing material linear attenuation coefficients, and the photon energy distributions from several X-ray sources. The linear attenuation coefficients have been gathered from information at the NDT Resource Center[1]. New materials can hence be added from this site if necessary. The source energy distributions were simulated using `SpekCalc`[2]. The graphs in Fig. 3 are reproduced from this spreadsheet.

**`material, source`** are classes containing this information. After running (`'ct_include.ipy'`), typing `'material.name'` will reveal the list of materials defined in `'material.coeffs'` for the energies in `'material.mev'`. This is accessed by the function `'material.coeff()'`. Likewise typing `'source.name'` will reveal the list of sources defined in `'source.photons'` for the energies in `'source.mev'`. This is accessed by the function `'source.photon()'`.

[1]`https://www.nde-ed.org/GeneralResources/MaterialProperties/Xray/Rt_matlprop_index.htm`
[2]`http://spekcalc.weebly.com/`

**`fake_source`** is a function for generating an approximate source energy distribution for any electron energy kVp and any thickness of filter material. Typing `'help(fake_source)'` gives instructions for using this function (and similarly for subsequent python functions). For instance, `'y = fake_source(material.mev, 0.12, material.coeff('Aluminium'), 2)'` can be used to generate an energy distribution in `y` for a 120 kVp source filtered by 2 mm of aluminium. You can show this distribution by typing `'plot(y)'`.

**`attenuate`** is an incomplete function which, when given a range of photon energies, a set of material linear attenuation coefficients (such as are provided by `material.coeff('Bone')`) and a depth (i.e. amount of material), should calculate the residual intensity at each photon energy after passing through that depth of the given material `'Bone'`, based on eq. (1). It is more efficient to calculate this for multiple samples at the same time.

**`ct_detect`** is a function which uses `attenuate` to calculate the summed residual energy for a variety of paths, each path consisting of a variety of materials with different depths. This is based on the product formulation in eq. (3) and will form the core of the CT simulator.

**`ct_lib.py`** also contains some small utility functions to make loading, saving and plotting of data easier.

## 3.3 Experiments

In this and subsequent sections, the text only describes what it is that you should be investigating or trying to demonstrate: the intention is that you will design and carry out appropriate experiments to acomplish this in each case.

- Ensure that you understand the basic idea of X-ray generation, material scattering and detection, by appropriate background reading. In particular you should understand the terms *X-rays*, *bremsstrahlung*, *Compton scattering*, the *photoelectric effect*, *linear attenuation coefficient*, *scintillator* and *collimator*.

- Take a look at the material linear attenuation coefficients, either using the data in variables or in the provided spreadsheet. Which part of these distributions is dominated by Compton and photoelectric scattering, and how wide is the transition region between these two?

- Complete the provided function `attenuate` so that it performs as suggested in the associated help text. For the moment, you do not need to worry about the `mas` parameter and this can be ignored. You should then be able to use `ct_detect` to assess the attenuation of a particular material, for example, `'y = ct_detect(source.photon('100kVp, 2mm Al'), material.coeff('Water'), np.arange(0, 10.1, 0.1), 1)'` will store in `y` the residual energy after a 100 kVp X-ray source, with a 2 mm aluminium filter, has passed through water for each of the depths 0 cm to 10 cm, in 0.1 cm intervals. Also ensure you understand what `ct_detect` does and how it works.

- Use `ct_detect` as above to investigate how different depths (amounts) of a material attenuate the source energy. Try plotting the logarithm of the residual energy, using `'plot(np.log(y))'` : given eq. 1, should this be a straight line? Make sure you repeat this with different source energies (either those provided in the `source` structure, or using the `fake_source` function), and also with single energy sources (using the `'ideal'` option in `fake_source`). Also look at how these results vary when using different materials.

Figure 4: A typical medical CT scanner (left) and ScanCo Xtreme CT for ankles and wrists (right).

## 4 CT scanning and the sinogram: second part of week 1

By the end of this section, you should know about the physical layout of a typical CT scanner, and how it uses multiple X-rays and detectors to form a ‘sinogram’ which contains sufficient information to be able to reconstruct a single slice through the ‘subject’ (i.e. whatever it is that is being scanned). We will also discuss how to simulate a subject using a discrete array of materials.

### 4.1 Background

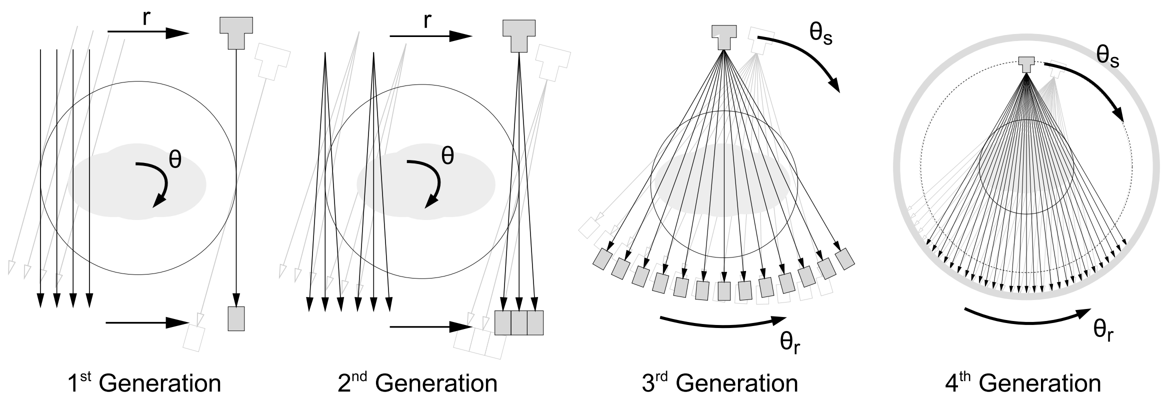


Figure 5: Historical configurations of CT scanners.

The discussion in the previous section only concerns how a single beam of X-rays is generated, interacts with materials, and is detected. We can imagine a small part of an X-ray beam (from the source to a specific detector) as a single path through the subject, or a single line in any of the diagrams in Fig. 5. This only gives us a single measurement: the residual energy after the X-ray has passed through all the material in this path. It turns out that, in order to be able to map the attenuation throughout a cross-section of the subject, we actually need this for a large set of path offsets $r$ and angles $\theta$, as in the 1st generation diagram in Fig. 5.

There are a variety of ways of achieving this: the simplest (and the one we will use to develop the reconstruction technique later) is 1st generation, with a single source and detector which are fixed relative to each other, but can move to different offsets and angles. 2nd generation scanners involve a fan-beam source and multiple detectors, and 3rd generation scanners have a fan-beam and sufficient detectors on an arc such that the whole subject cross-section can be irradiated at once. In this case there is a detector 'angle' $\theta_r$ rather than an 'offset' $r$. The source and detector are fixed with respect to each other and the whole source-detector combination is rotated by more than $180^\circ$ to generate the different source angles $\theta_s$. This is more common, and the raw data later in the project will come from this sort of scanner setup.

There are also more complex scanners involving complete rings of detectors: they are around the circumference of the hole in the left-hand scanner in Fig. 4: the subject lies on the table and is gradually moved through to create a whole 3D volume, with the source moving around half the circle for each location of the table. Real 3D scanning can be done with multiple 2D arrays of detectors, cone-beam shaped X-rays (i.e. a fan in two dimensions) and a helical movement where the table and source rotation are simultaneous: but we will keep it simple here and presume each cross-section is dealt with independently.

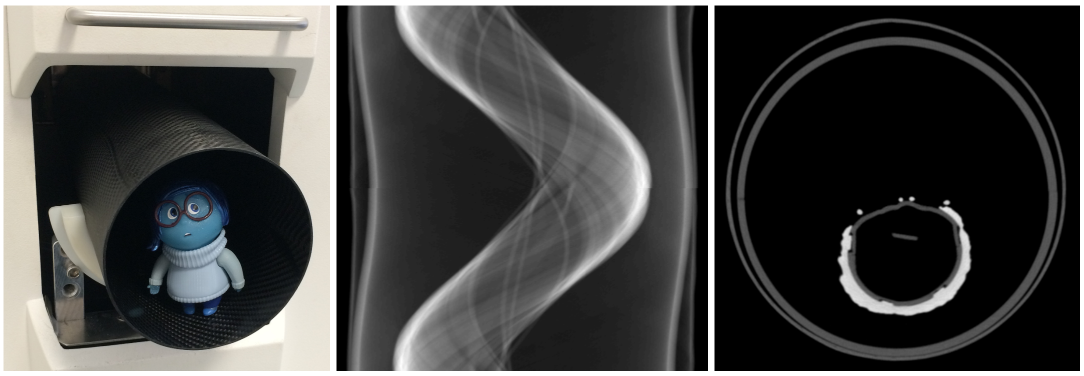


Figure 6: Sample sinogram and reconstruction. The subject can be seen within the scanner (left) and one slice of the 3D data is shown, as a sinogram (centre) and also as a reconstructed image (right).

A 'sinogram' is essentially the raw data that comes from the scanner for a single cross-section: the measured energy at each detector, for each offset $r$ (or $\theta_r$ for fan-beams) and source angle $\theta$. An example sinogram is shown in Fig. 6, where $r$ varies in the horizontal direction and $\theta$ in the vertical direction. It is called a 'sinogram' because a fixed point in space generates a sine wave in the sinogram.

We are actually interested in the *total attenuation* for each path, rather than the residual intensity $I_{\text{tot}}$. Re-arranging this measured energy from eq. (2) gives:

$$\mu_{\text{tot}} \equiv \sum_m \mu_{m,E}\, x_m \approx \sum_m \mu_m x_m = -\log_e \frac{I_{\text{tot}}}{\sum_E I_{0,E}} \tag{4}$$

which means that we need to know the total source energy $I_0$ (as received by the detectors) as well, usually from a calibration scan where only air is present, and hence there is very low attenuation. However, eq. (4) is only possible because of the approximation in the middle where we ignore the fact that $\mu$ varies with photon energy! We will come back to this later.

In a real CT scanner anything the X-ray passes through, at any scale, can affect the residual energy. In our simulator, we are going to describe the 'subject' to be scanned as an array of different materials: i.e. a discrete 2D representation, where each 'pixel' of the array has a certain size and is all associated with one specific material. This simulated subject is often referred to as a 'phantom' image. We therefore have to be able to 'scan' this phantom, i.e. to work out which pixels, containing which materials, an X-ray at a particular offset $r$ and angle $\theta$ will pass through. We will presume that the detectors are exactly the same size as the pixels in the phantom.

Even in a linear 1st generation scan, the X-ray locations and orientations are not in general going to be aligned with the phantom 2D array. Figure 7 in the next section shows an example of this: the source angle $\theta$ varies, and hence so does the coordinate system for the measurements, however the simulated phantom data stays in a fixed (world) coordinate system. So we are going to need to *interpolate* or *approximate* the phantom image to generate new data which is aligned with the X-ray at each angle $\theta$: this is the process of guessing a data value which is not exactly coincident with any of the samples in the phantom array. Once we have done this, we can simply sum the material contributions for each pixel in the *interpolated* data to get the total depth (amount) of each material along a specific X-ray path.

How we do this interpolation will affect the accuracy of our results. Various methods are possible, from basic nearest-neighbour interpolation, through linear interpolation, to cubic B-spline approximation. These vary in how much surrounding data they use to 'guess' the nearby data value, and what assumptions are made in terms of the smoothness of the interpolated data.

## 4.2 Resources

The following resources should help with this and subsequent sections:

**`ct_phantom`** is a function which will generate a variety of phantom images for testing the CT simulator. These images are 2D arrays of pixels where each pixel has a value which corresponds to an entry in the `material.name` field and indicates what material is at that pixel. Various resolution and anatomical phantoms can be created, for instance typing `'y = ct_phantom(material.name, 256, 3, 'Titanium')'` would generate a 256 × 256 phantom in `y` simulating a single hip replacement made of titanium.

**`ct_scan`** is a function which simulates a CT scan of a phantom containing various materials. For each angle in turn, and for each material, it interpolates the phantom in line with the scanning angle, then sums along each ray to discover the depth (amount) of this material in each ray. It then uses `ct_detect` to calculate the residual energy after passing through these materials, and outputs this as a sinogram. It uses the `scipy` function `ndimage.map_coordinates` to perform linear interpolation: see the help for this function for alternative interpolation schemes. For instance `'y = ct_scan(source.photon('100kVp, 2mm Al'), material, X, 0.1, 256)'` will generate a sinogram in `y` for the phantom `X`, using 256 angles and presuming a pixel size of 0.1 cm.

**`ct_calibrate`** is an incomplete function which needs to take the sinogram from `ct_scan` and turn it into a set of total attenuations as in eq. (4).

**`draw`** is a utility function for conveniently showing a 2D array as an image, with a colour key indicating the data values. For instance typing `'draw(X)'` will display the phantom data in X. The displayed range of data *values* can be limited using `draw(X, caxis=[-1,2])`: this will display values from $-1$ (in black, or the first colour in the colourmap) to 2 (in white, or the last colour in the colourmap).

Note that the utility function (e.g. `plot`), and other plotting functions from `matplotlib`, are also very useful for displaying a single line through the results: this can sometimes reveal subtle variations in image value which are not visible when just displaying the whole image using `draw` above.

## 4.3 Experiments

- Ensure that you understand the layout of typical CT scanner sources and detectors and also the most common approaches to interpolation. In particular you should understand the various *CT generations*

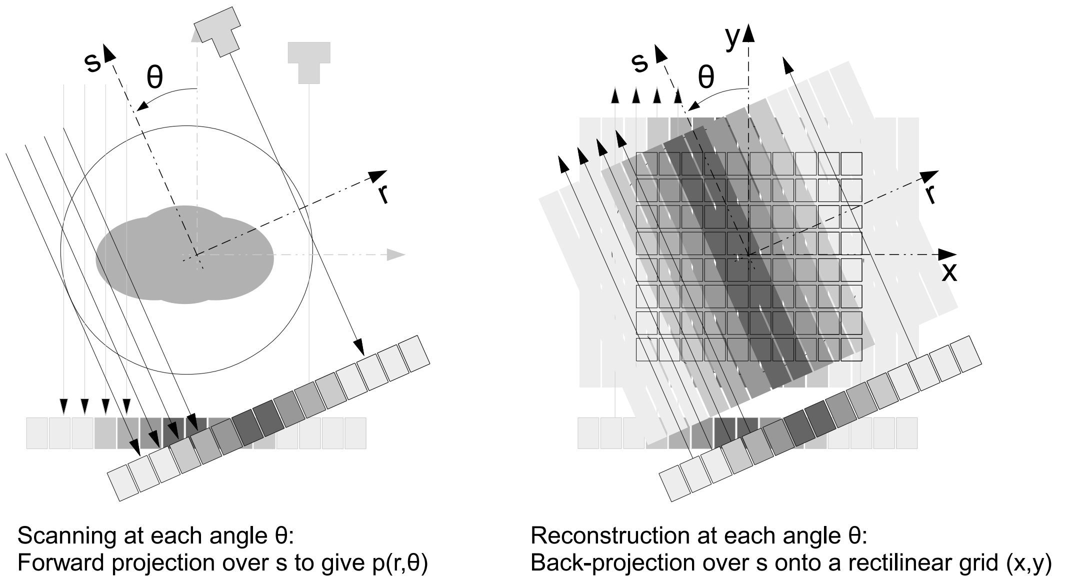

s
θ
r
Scanning at each angle θ:
Forward projection over s to give p(r,θ)
y
s
θ
r
x
Reconstruction at each angle θ:
Back-projection over s onto a rectilinear grid (x,y)

Figure 7: Scanning (forward projection) and reconstruction (back-projection).

and also the difference between *nearest-neighbour*, *linear* and *cubic* interpolation. If you are interested in finding out more about 3D scanning strategies, you could also read about *cone-beam CT*, *helical CT* or *multi-slice CT*.

- Check that you understand the working and the output of the `ct_phantom` function, and also how the `ct_scan` function interpolates this phantom data and calculates residual energy.

- Construct a sinogram using the `ct_scan` function and investigate what it looks like for various simple shapes — for instance an impulse (single point) in various places and a disk — as well as the more complex phantoms. Also investigate the effects of varying the number of angles and changing the default interpolation technique.

- Complete the provided function `ct_calibrate` so that you can convert the sinogram from residual energy to total attenuation values. This will involve simulating a 'calibration' scan in air. Confirm that the result scales with actual material attenuation as you would expect.

# 5 Reconstructing cross-sectional data: first part of week 2

Having simulated the creation of a sinogram from some phantom data, we now consider how to use this sinogram to re-create a cross-sectional image of the phantom. This process is called *reconstruction*: an example reconstructed image is shown in Fig. 6.

## 5.1 Background

The aim of reconstruction is to use the sinogram to create a discrete linear attenuation distribution $\mu(x, y)$, which is a cross-sectional image related to the distribution of densities in the subject. In order to work out how to do this we also need to define the scanning process, which is actually a *forward projection*, as in the left-hand image in Fig. 7. If we define the calibrated sinogram from the last section as $p(r, \theta)$, and the scanning direction at each angle as $\vec{s}$, then we can represent the scanning process as a line integral of the attenuation in this direction:

$$
\begin{aligned}
p(r,\theta) &= \int_{\vec{s}} \mu(x,y)\,\mathrm{d}s && (5)\\
&= \int_{\vec{s}} \mu(r\cos\theta - s\sin\theta, r\sin\theta + s\cos\theta)\,\mathrm{d}s \\
&\equiv \mathcal{R}\left\{\mu(x,y)\right\} && (6)
\end{aligned}
$$

where $\mathcal{R}\{\mu\}$ is called the *Radon transform* of $\mu$, and the relationship between $(x, y)$ and $(r, s)$ is shown in Fig. 7. Hence what we would like to do is invert the Radon transform so that we can get back from the sinogram $p(r, \theta)$ (which is what we actually measure) to $\mu(x, y)$:

$$\mu(x,y) = \mathcal{R}^{-1}\left\{p(r,\theta)\right\} \tag{7}$$

Intuitively, since the scanning process is a forward projection, it would make sense for the reconstruction process to look like a *back-projection*. This is shown in the right-hand image of Fig. 7: for each line in the sinogram, we 'smear' this back across a Cartesian grid in $(x, y)$ and accumulate the result over all angles. If we call this back-projection $b(x, y)$, we can represent the result for each pixel $(x, y)$ as an integral over all the scanning angles:

$$
\begin{aligned}
b(x,y) &= \int_0^\pi p(x\cos\theta + y\sin\theta, \theta)\,\mathrm{d}\theta \\
&= \int_0^\pi p(r,\theta)\,\mathrm{d}\theta && (8)
\end{aligned}
$$

This intuition turns out to be nearly, but not quite, right. We need some maths to see why, which is largely based on the treatment by Suetens (2009). In fact we can make progress by considering two different Fourier transforms. Firstly, the 2D Fourier transform of the linear attenuation distribution $\mu(x, y)$ which is:

$$\mathcal{F}_{\mathrm{2D}}\left\{\mu(x,y)\right\} = \iint_{-\infty}^{\infty} \mu(x,y)\, e^{-i(\omega_x x + \omega_y y)}\,\mathrm{d}x\,\mathrm{d}y \tag{9}$$

where $\omega_x$ and $\omega_y$ are spatial frequencies in each of the $x$ and $y$ directions.

Secondly, the 1D Fourier transform (in the $r$ direction) of the sinogram $p(r, \theta)$:

$$\mathcal{F}_{\mathrm{1D}}\left\{p(r,\theta)\right\} = \int_{-\infty}^{\infty} p(r,\theta)\, e^{-i\omega r}\,\mathrm{d}r \tag{10}$$

where $\omega$ is the spatial frequency in the $r$ direction (which varies with the angle $\theta$).

These two expressions can actually be related: you can see this by substitution of the appropriate parameters below, and noting that, because the $(x, y)$ to $(r, s)$ coordinate transform is just a pure rotation, its Jacobian

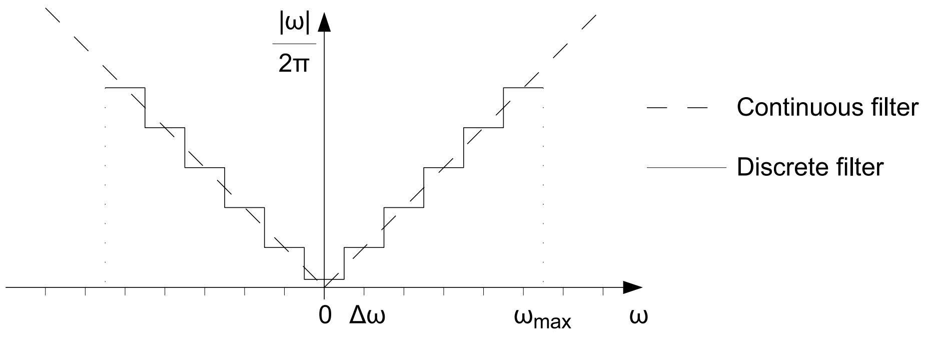


Figure 8: Filter in back-projection.

is unity. We first use eq. (5) to substitute for $p(r,\theta)$ in eq. (10):

$$\begin{aligned}\mathcal{F}_{\mathrm{1D}}\{p(r,\theta)\} &= \iint\limits_{-\infty}^{\infty} \mu(x,y)\,e^{-i\omega r}\,\mathrm{d}s\,\mathrm{d}r \\ &= \mathcal{F}_{\mathrm{2D}}\{\mu(x,y)\} \quad \text{where} \begin{cases} x = r\cos\theta - s\sin\theta \\ y = r\sin\theta + s\cos\theta \\ \omega_x = \omega\cos\theta \\ \omega_y = \omega\sin\theta \end{cases} \end{aligned} \tag{11}$$

This is known as the *projection theorem* or *central slice theorem*. What it says is that the Fourier transform of the projection of $\mu(x,y)$ at an angle $\theta$ (i.e. $\mathcal{F}_{\mathrm{1D}}\{p(r,\theta)\}$) is the same as first taking the 2D Fourier transform of $\mu(x,y)$ (i.e. $\mathcal{F}_{\mathrm{2D}}\{\mu(x,y)\}$) and then just looking at a slice through it at whatever angle $\theta$ the projection was in (i.e. the given substitutions for $x$, $y$, $\omega_x$ and $\omega_y$). There are lots of pictorial explanations of this online, since it is also very important for Magnetic Resonance Imaging (MRI).

We can use this to invert the Radon transform as follows:

$$\begin{aligned}\mu(x,y) &= \mathcal{F}_{\mathrm{2D}}^{-1}\{\mathcal{F}_{\mathrm{2D}}\{\mu(x,y)\}\} && \text{– self evident} \\ &= \mathcal{F}_{\mathrm{2D}}^{-1}\{\mathcal{F}_{\mathrm{1D}}\{p(r,\theta)\}\} && \text{– from eq. (11)} \\ &= \int_0^{\pi}\int_{-\infty}^{\infty} \mathcal{F}_{\mathrm{1D}}\{p(r,\theta)\}\,\frac{|\omega|}{2\pi}\,e^{i\omega r}\,\mathrm{d}\omega\,\mathrm{d}\theta && \text{– using polar version of } \mathcal{F}_{\mathrm{2D}}^{-1} \\ &= \int_0^{\pi} \mathcal{F}_{\mathrm{1D}}^{-1}\left\{\frac{|\omega|}{2\pi}\mathcal{F}_{\mathrm{1D}}\{p(r,\theta)\}\right\}\mathrm{d}\theta && \text{– noting this is an inverse Fourier transform} \quad (12) \\ &= \int_0^{\pi} \mathcal{F}_{\mathrm{1D}}^{-1}\left\{\frac{|\omega|}{2\pi}\right\} \star p(r,\theta)\,\mathrm{d}\theta && \text{– can also be expressed as a convolution} \quad (13)\end{aligned}$$

where eq. (13) is very similar to our intuitive guess in eq. (8), except that we are first filtering the sinogram $p(r,\theta)$ with $\mathcal{F}_{\mathrm{1D}}^{-1}\left\{\frac{|\omega|}{2\pi}\right\}$. This is hence called *filtered back-projection* or FBP, and remains probably the most common technique in CT reconstruction.

The required filter with frequency response $\frac{|\omega|}{2\pi}$ is shown in Fig. 8: the continuous form is infeasible since the gain is infinite at infinite frequencies, so the discrete version has a cut-off frequency $\omega_{\max}$. Actually

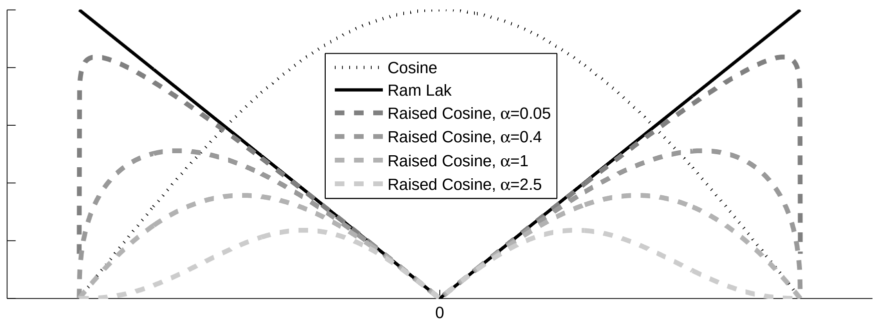


Figure 9: Raised cosine filter.

the discrete version is not *exactly* the same as the continuous version sampled at the discrete frequencies: the reasons for this are complex, but relate to the need for the filter to be periodic. The exact form is given by Zeng (2014), but an approximate correction can be made simply by replacing the zero at $k = 0$ with $\frac{1}{6}$ of the value at $k = 1$, where $k$ is the frequency index. This is called a *Ram-Lak* (Ramachandran and Lakshminarayanan, 1971) filter.

Implementation hence usually follows eq. (12), except with a summation over the discrete measured angles, rather than an integral:

1. Measure and calibrate the sinogram $p(r, \theta)$.
2. Take the Fourier transform in the $r$ direction.
3. Multiply each frequency by the appropriate coefficient in the Ram-Lak filter.
4. Take the inverse Fourier transform in the $r$ direction.
5. Back-project each angle over the $(x, y)$ grid and sum the result, to give $\mu(x, y)$.

Step 5 in this procedure will involve interpolation since, as can be seen from the right-hand diagram in Fig. 7, the attenuation $\mu(x, y)$ that we are creating is not aligned with the sinogram $p(r, \theta)$ that we are back-projecting. There are hence some important decisions to be made regarding how we interpolate the data from one coordinate system to the other, similarly to the discussion in the previous section regarding the forward projection of the sinogram. The difference here is that, whereas the forward projection was only an issue for the CT simulation, the back-projection has to be performed in the real CT scanning reconstruction process.

There are lots of variations on the Ram-Lak filter which mostly seek to reduce the high-frequency response a bit, since this can otherwise emphasise noise in the reconstructions. One particularly useful case is the raised-cosine:

$$f(\omega) = \frac{|\omega|}{2\pi} \cos\left(\frac{\omega}{\omega_{\max}} \frac{\pi}{2}\right)^{\alpha} \tag{14}$$

which is shown in Fig. 9. Using very small values for $\alpha$, e.g. $\alpha \ll 1$ means that $f(\omega) \approx \frac{|\omega|}{2\pi}$, but larger values of $\alpha$, e.g. $\alpha > 1$ attenuate high frequencies.

## 5.2 Resources

The following resources should help with this and subsequent sections:

**`back_project`** is a function which will take a filtered sinogram and back-project it over a reconstruction array. Typing '`y = back_project(x)`' creates an array `y` of size $n_r \times n_r$, given the sinogram `x` of size $n_\theta \times n_r$.

**`ramp_filter`** is an incomplete function, which should take the calibrated sinogram and filter it using an appropriate filter.

**`scan_and_reconstruct`** is an incomplete function which should make use of the other functions you have written to simulate the whole scanning and reconstruction process. For instance, typing '`y = scan_and_reconstruct(source.photon('80kVp, 2mm Al'), material, X, 0.1, 256)`' will result in simulating the CT scanning of a phantom in `X` (presumed to be a square array) with a scale of 0.1 cm per pixel. This will use the material class and the first provided radiation source, and scan with 256 angles from 0 to $\pi$.

**`ct_test_example`** is an incomplete function to provide a template for your simulator test code in the group submission.

## 5.3 Experiments

- Check that you understand the basics of *filtered back-projection* and how back-projection is implemented in the `back_project` function, as well as the meaning of *Ram-Lak filter*. If you are interested, other techniques include *iterative reconstruction*, which is becoming more popular again: the simplest version is the *algebraic reconstruction technique* or ART.
- Complete the `scan_and_reconstruct` function so that it will perform all the operations required to scan and reconstruct a phantom image in `X`. Ignore the filtering step for the moment.
- Try experimenting with this function (i.e. without filtering) on simple phantoms containing an impulse or a disk, and note how the result compares to the original data.
- Complete the `ramp_filter` function and calculate the impulse response of this filter, i.e. how this filter alone affects a single impulse (you can display the results using `plot`). Once complete, add this to the `scan_and_reconstruct` function, and repeat the previous experiment on simple phantoms, but now including this filter before reconstruction. Note the difference this makes to the reconstructed data.
- Investigate the effect of varying the number of angles on the reconstructed image, how many angles are necessary for a good result, and how this varies with the size of the phantom image.
- Investigate the impact of varying the type of interpolation in the `interpolate.interp1d` function in `back_project`.
- Try multiplying the Ram-Lak filter coefficients with a cosine raised to the power of the `alpha` parameter, as in equation (14). How does varying `alpha` affect the results of the previous two experiments?

## 6 Group submission: simulator and test code

At this stage your group should have a functional version of `scan_and_reconstruct` which can take a phantom image and produce a reconstruction which will look very similar, though will have data levels corresponding to linear attenuation, rather than an index relating to material type.

In order to test this code, you should design some *end-to-end* tests, based on the provided template `ct_test_example`. An *end-to-end* test is one in which you set some initial conditions (for instance related to the phantom, materials, or source), run the simulator on them, and check that the final output (reconstructed image) is what you would expect it to be, given the initial conditions. Each of these tests should save results (images or text files), clearly named according to the test number, in a `results` sub-directory. These tests are *not* intended to cover all the experiments in the previous sections (these are covered by the first interim report, see Section 8), just to confirm that the completed simulator works correctly. It is up to you which tests to include, but you should at least have one which assesses the reconstructed *geometry* and one which assesses the reconstructed data *values*.

Two things are required for the group submission:

- A zip file containing a working version of your simulator code up to this point (i.e. not including any modifications you have made for the later sections in this handout), and also your version of the test code `ct_test_example`. The test code should be appropriately commented to carefully explain what each test is checking, what the initial conditions and the test are, and what you expect the results to be. You should also comment on any modifications to the original simulator code.
- An additional zip file containing the results in your `results` directory after running your test code.

These files are due in by 1 pm on Friday 15th May. The primary goal of this submission is to ensure that each group has a working CT simulator at this stage, and understands how it works.

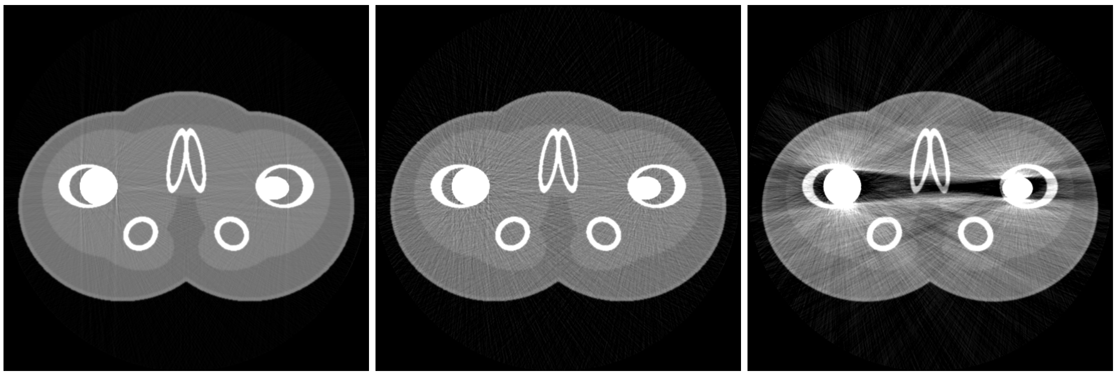


Figure 10: Modelling noise sources. These all use a single-energy source at 74 keV with: no noise (left), transmission noise (centre) and also noise from multiple scattering and background radiation (right).

## 7 Noise, beam hardening and Hounsfield Units: second part of week 2

Having finished the basic CT simulator in the previous section, the aim here is to add three refinements to this simulator, covering noise, calibration for multiple-energy sources, and conversion to Hounsfield Units. You may prefer to work independently at this point so one each from your group addresses each of these refinements.

### 7.1 Background

#### 7.1.1 CT noise and current

Up to this point we have been ignoring sources of noise in the CT scanning process, and hence also the tube current and time taken to scan, given by the `mas` parameter in several of the provided functions.

In reality, the energy in an individual photon is not reduced according to eq. (1): a single photon will either end up interacting with an atom (in which case scattering will occur) or not (in which case no energy is lost at all). The effect of many of these interactions tends towards a binomial distribution. Equally the *creation* of photons at the source actually follows a Poisson distribution. It is only when we consider the energy from lots of photons in the X-ray beam that we can model the *combined* energy reduction. In this case it turns out that the number of *transmitted* photons also follows a Poisson distribution, with a mean given by eq. (1). However, we also need to adjust the number of source photons: the expressions we have been using are photons per milli-amp-second per square centimetre, i.e. the units are $\text{mAs}^{-1}\text{cm}^{-2}$, where the area is the cross-section of the beam as seen at each detector. Hence increasing the current-time-product, or the size of each detector, should also increase the number of source photons.

In addition to this, the detectors can respond to photons which have not travelled directly from the source: either as a result of background radiation from the external environment, or multiple scattering events. The collimators are designed to limit these effects, but they don't remove them entirely, so the residual energy will have an additive fixed component (the amount of background radiation) and one which scales with the number of source photons (the multiple-scattering). Both of these can also be modelled using a Poisson distribution.

Figure 10 has some examples of simulated scans which add in each of these components one at a time. The left-hand image is a simulation with no noise at all.

The presence of non-direct radiation means that the relative noise level increases if the actual transmitted radiation is smaller: so materials which are very attenuating, like bone or metallic implants, can generate quite high levels of noise in the reconstruction. This is the cause of the black streak in the right-hand image

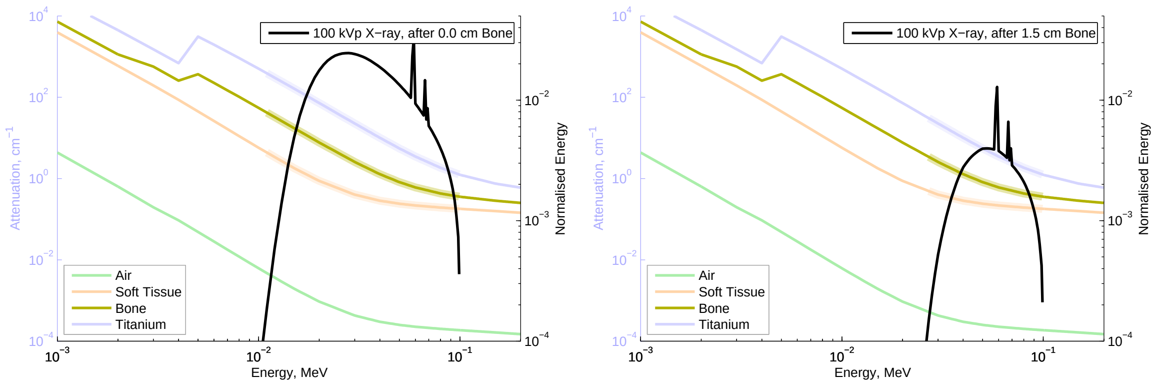


Figure 11: Residual energy distribution after passing through different bone thickness.

of Fig. 10, which is between two highly attenuating Titanium implants. In practice we also have to limit the detector signal at some level above zero (for instance at 1 photon), since the logarithm in eq. (4) would otherwise imply an infinite attenuation.

### 7.1.2 Beam hardening correction

In the earlier sections we noted that the source radiation had quite a wide energy distribution, and also that the material linear attenuation varies quite considerably with energy. This was expressed by eq. (2), however in the later eq. (4) and in the reconstruction maths in the previous section, we ignored this variation in energy, and presumed that the material only had a single linear attenuation value. This is a valid assumption if the source only contains photons of a single energy, or if the material linear attenuation is constant with energy.

The reality is more like Fig. 11, which shows both the linear attenuation of several materials and the typical energy spectrum of a source before and after travelling through 1.5 cm of bone. If you consider the average attenuation over the source energy distribution, this is higher in the left-hand graph than it is on the right: hence the more bone that the radiation travels through, the smaller the effective attenuation is. Another way of expressing this is that the lower energy radiation gets attenuated much faster than the higher energy radiation. This is known as *beam hardening*, where the 'hardening' relates to the higher average beam energy after it has passed through a material, which is then attenuated less.

The result is that the attenuation per cm is not actually a constant, but varies with the amount of material: so a CT image of a disk of the same material will show a 'cupping' pattern: apparently lower attenuation at the centre of the disk which is actually the result of the radiation having to travel through more of the material to get to that point. Hence if we plot the total attenuation against the thickness of a material, this is a curve rather than the straight line we might naturally expect. The physical filter (e.g. Aluminium) placed in front of the X-ray source helps by reducing the energy range of the source, but it doesn't eliminate the problem.

Whilst it is difficult to eliminate this effect for *all* materials in a scan, it is possible to correct it for a *single* material by appropriate calibration of the detected attenuation values. Since in medical CT much of the subject is soft tissue or muscle, and the attenuation is very similar to water, CT machines are usually calibrated to remove beam hardening artefacts for water. This can be done by measuring the recorded attenuation by eq. (4), $\mu_w$, for a range of different water thicknesses, $t_w$ and fitting a function $f$ to this data, where:

$$t_w = f(\mu_w) \tag{15}$$

During a real scan, for each measured attenuation value $\mu_m$, we use this function to determine what equivalent

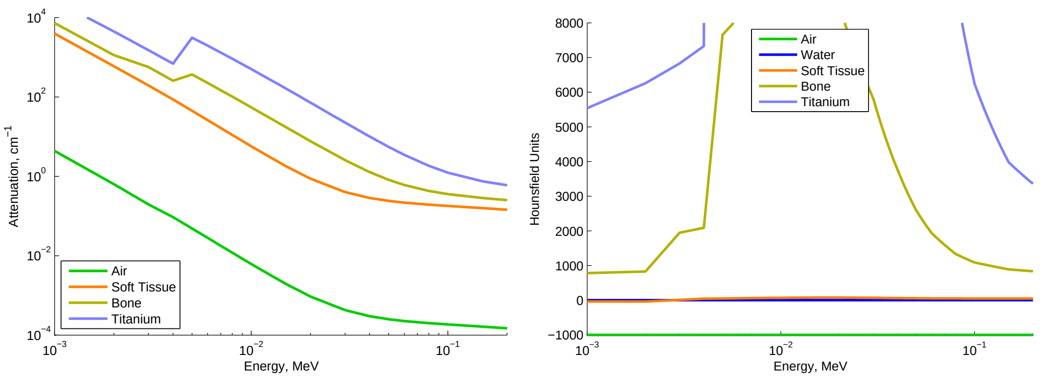


Figure 12: Attenuation distribution with energy (left) and after conversion to Hounsfield Units (right).

water thickness would be needed to get this value, i.e.:

$$t_{w,m} = f(\mu_m) \tag{16}$$

The *calibrated* attenuation is just a scaled version $\mu_c = Ct_{w,m}$, where $C$ is chosen so that $\mu_c \approx \mu_m$ at some arbitrary low thickness of material. If the conversion to Hounsfield Units (see next section) is performed correctly, then the value of the scaling parameter $C$ should not matter once the data has been converted.

### 7.1.3 Hounsfield Units, windowing and DICOM

Since linear attenuation is dependent on the source energy as well as the material, see the left-hand graph in Fig. 12, the measured attenuation would be different depending on what range of energies a CT scanner used. The source energy is itself dependent on the peak energy of the electrons, which can vary from around 60 to 140 kVp, and also the material and thickness of the filter in front of the X-ray beam.

Clearly it is not ideal to have patient data which differs depending on which CT machine was used to scan with: so in an attempt to standardise the values in such images, the Hounsfield Unit (HU) was developed, defined as:

$$\text{HU} = \frac{\mu - \mu_w}{\mu_w} \times 1000 \tag{17}$$

where $\mu$ are the values of the reconstructed image (i.e. the measured, calibrated attenuation), and $\mu_w$ is the equivalent value for water. This results in the right-hand graph in Fig. 12: by definition, water is now always at 0 HU, air at -1000 HU (since eq. (4) has calibrated the attenuation with reference to air), but other materials can still be variable: bone and Titanium are particularly striking examples.

For medical CT, Hounsfield Units are usually stored as integers between the 12-bit range of -1024 and 3072, with any values outside this range set to the nearest limit. This is a much greater dynamic range than most computer displays can cope with, since they usually only have 256 different brightness levels. Hence the Hounsfield Units are also *windowed* before they are displayed, to highlight a particular range of values:

$$g = \frac{\text{HU} - c}{w} \times 128 + 128 \tag{18}$$

where $g$ is the brightness value to display for a particular measurement HU (restricted to the range $0 \leq g \leq 255$), and $c$ and $w$ are the window centre and width respectively. These take standard values for different clinical situations: so for instance to display soft tissue you might set $c = 0, w = 200$ since it is similar to water at 0 HU but can vary from this by about 100 HU.

Medical imaging data is usually stored in DICOM[3] format, which is very complex and covers other areas like data transmission as well as storage. Whilst the entire definition of DICOM is well beyond the scope of this project, it is impossible to avoid it entirely since medical imaging viewers (which we will be using later) usually expect data in this format. In the case of 3D CT data, a DICOM file would store various patient and hospital information, CT scanner parameters, and positional information about the image and patient location, as well as the image data itself. A 3D data set is usually stored as a sequence of DICOM files (one per cross-section), which all contain reference numbers (UIDs or 'Unique IDentifiers') which mark each file out as being a part of that 'sequence' (single set of 3D data) and 'series' (multiple sets of 3D data but all of the same patient, scanned at the same time and with the same scanner).

## 7.2 Resources

Most of the resources needed for this section have already been discussed, but in addition:

**`hu`** is an incomplete function which should take a reconstructed attenuation image, the source energy, and material properties, and return the image converted into Hounsfield Units and clipped to the appropriate range of values for storing DICOM data.

**`create_dicom`** is a function for saving the reconstructed data, presumed to be the output of `hu`, in DICOM format. For instance, typing `'create_dicom(X, 'ct_data', 0.1)'` will create the DICOM file `'ct_data_0001.dcm'` containing the data `X` with a pixel size of 0.1 cm. The function can also be used to create multiple slices for 3D data sets.

## 7.3 Experiments

### 7.3.1 CT noise and current

- Add in calculation of the actual number of source photons, estimation of the transmitted scatterer distribution, and also additional detection due to indirect scattering. This will involve modification of the `attenuate` and `ct_detect` functions, and use of the `mas` and `scale` values.
- Investigate the effects of using both low and high `mas` settings. Does it matter how physically large the phantom is (i.e. the pixel size)? What happens when there is very high attenuating material in the scan?
- To what extent can low `mas`, and hence higher noise, be compensated by using a Ram-Lak filter with a narrower band (i.e. increasing the power `alpha` in the raised cosine so $\alpha > 1$)?

### 7.3.2 Beam hardening correction

- Try reconstructing disks or anatomical phantoms, using either 'real' sources or 'ideal' single-energy sources (which shouldn't exhibit beam hardening). How do the reconstructions differ when using a 'real' source?
- Extend the `ct_calibrate` function to include beam hardening correction for water. To do this you will need to implement eqs. (15) and (16), and also simulate the water calibration scan using `ct_detect` or `attenuate`. You can either calculate lots of different values for $t_w$ and $\mu_w$ and then use `scipy.interpolate.interp1d` to find $t_{w,m}$, or you can model $f$ as a function, for instance using `numpy.polyfit`.

[3]Digital Imaging and Communications in Medicine, `http://dicom.nema.org/`

- Check that this function generates the correct result when scanning a disk made entirely of water. Then investigate what difference your new calibration makes to an anatomical phantom, and to multiple disks made out of various other materials.

#### 7.3.3 Hounsfield Units and DICOM

- Complete the `hu` function so that it implements eq. (17) and add this to the `scan_and_reconstruct` function. You will need to use `attenuate` or `ct_detect` to establish the value of $\mu_w$. Check by using `draw` that the reconstruction has sensible values.
- Save the data in DICOM format using `create_dicom` and confirm that you can load the image into a typical DICOM viewer (see Section 10 for some suggestions).
- Investigate how stable the HU values are for water and other materials at different beam energies. Also note the effect of *windowing* the data, which can be achieved by using `draw` and then changing the mapping of value to brightness using the `caxis` argument.

## 8 Individual submission: first interim reports

This as an individual report, minimum font size 10 pt, and a maximum of 4 pages, plus 3 for appendices (additional figures and tables only). You should summarise the interesting background reading and experiments and results up to the end of Section 7. Ensure that you have answered all the questions raised in the experiments and discussed the suggested investigations. The report should include *all* the investigations from Sections 3 to 5, and also those investigations in Section 7 in which you were actually involved. Be concise: keep the discussion brief where things are straightforward, but pay more attention to detail in areas where you think something interesting is happening. Do not include large chunks of this handout: you can refer to it where necessary.

It is important that this report is **written entirely by yourself**. You can include statements that have been derived from papers or the internet, or images that you have downloaded, so long as they are all **clearly cited stating their source**. Since this is a group project, it is expected that you will have shared some code between you as a group, and hence some images, and the data from plots, will be identical between different group members. However, you should always create the figures yourself, and write your own figure captions, and you should **not share your reports between group members**.

This report is due in by 1 pm on Thursday 21st May.

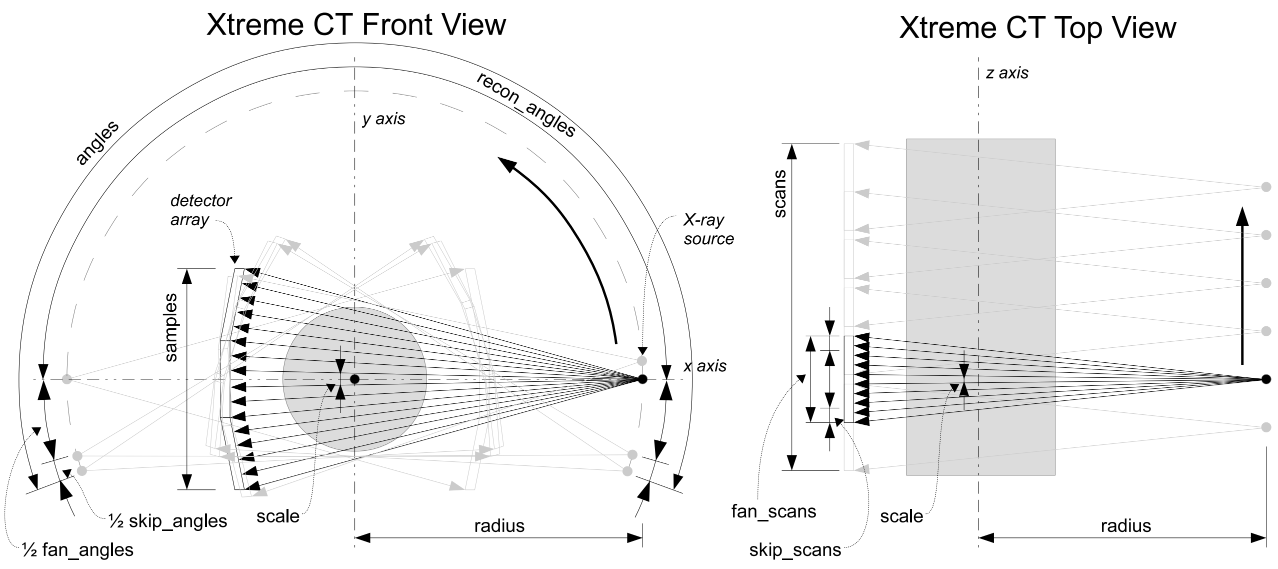


Figure 13: Schematic of the ScanCo Xtreme CT scanner (not to scale: both radii are actually larger).

## 9 Real 3D CT data

### 9.1 Background

Since it is difficult to access medical CT scanners in the hospital, we will be using data from an XtremeCT[4] high resolution micro-CT peripheral scanner usually used for ankles and wrists: see the right-hand scanner in Fig. 4. Compared to a whole-body medical CT, the peak energy is lower: 60 kVp, rather than 120 kVp or so for a medical scanner. In addition, the scanning area is much smaller: roughly a cylinder of diameter 12 cm and length 15 cm. However, the resolution is also higher, so the resulting data is quite similar, though at a different scale. This scanner has the advantage that we can access the raw data, i.e. the actual measurements at the detectors.

A schematic of the Xtreme CT is in Fig. 13. In this case the detector is a 2D array (actually three arrays side by side), and the X-rays are arranged as a fan scan such that the whole sample is irradiated at one time: so in this sense a 3rd generation geometry. The fan is in two dimensions, so as well as multiple angles, there are also multiple slices in the $z$-direction in a single shot. The source is then rotated: in fact it is necessary to rotate this slightly more than 180° to account for the fan angle. After each rotation, it is moved in the $z$-direction and the process is repeated. This is a type of what is known as *cone-beam CT*.

The important parameters of the acquisition are shown in Fig. 13: these correspond to the return values from the `xtreme` class described below. `samples`, `scans`, `radius`, `fan_scans` and `skip_scans` are all in the same units, corresponding to pixel numbers on the 2D detector array. The pixel width in the $y$ or `sample` direction, measured at the $y$-axis[5], is the same as the pixel height in the $z$ or `scan` direction[6]. Hence multiplying by `scale` will convert all these to equivalent millimetres on each of the axes. `angles`, `recon_angles`, `fan_angles` and `skip_angles` correspond to angle numbers in the rotation. Multiplying these by `dtheta` will convert them to angles in radians.

The raw data is stored in an `'.rsq'` file: this contains all of the detector measurements, as well as one

[4]ScanCo Medical, `http://www.scanco.ch/en/systems-solutions/clinical-microct/xtremect.html`
[5]Note that since this is actually a fan scan, the distance between each sample in the $y$ direction varies slightly along the $x$ direction.
[6]Again, this is measured at the $z$-axis.

scan with the X-ray source turned off, and one with it on but nothing in the scanner. These two scans give the reference intensity $I_0$ and also the noise floor for the detector: i.e. the amount of noise and/or background radiation that it measures irrespective of the source. It can be accessed as a fan-based sinogram (or 'slice') at a particular $z$ location, in which case the data size will be `angles` × `samples`. Alternatively it can be accessed as a sort of X-ray image (or 'scan') at a particular angle, in which case the data size will be `scans` × `samples`. In the former case, the `angles` contain all those in one rotation: 180° plus the `fan_angles` and additional `skip_angles`. In the latter case, the `scans` contain every scan from every fan in the $z$-direction: each of these fans contains `fan_scans`, but many of these scans will actually overlap, and there will not necessarily be a whole number of fans.

The fan-based reconstruction in the $x$-$y$ plane is a bit more complex than the parallel reconstruction discussed in Section 5. However, all X-rays are straight lines, and each X-ray in each fan hence corresponds to a (different) X-ray in an equivalent parallel reconstruction: you might be able to see this from Fig. 13. Hence a fan-based sinogram can be converted into an equivalent parallel-based sinogram by re-organisation and interpolation of the data. The fan in the $z$ direction is a little harder to deal with, but we can note that the angle is only very slight (less than shown in Fig. 13) so one option is to simply ignore it and presume that the X-rays are parallel in this direction.

## 9.2 Resources

At this stage you will be provided with some raw scan data and various functions to help you work with it. These are not a complete set of replacements for the simulator you have already developed, but they are sufficient to read and interpret the raw data from the Xtreme scanner.

**`raw_data.rsq`** is the raw data from the Xtreme CT scanner.

**`data_????.dcm`** are reconstructions of the raw data, stored in DICOM format files. There is one file per cross-sectional slice ($x$-$y$ plane), and the files are numbered sequentially for each slice location ($z$ axis). These files are reconstructed at $\frac{1}{4}$ of the actual scanner resolution so are significantly less clear than the reconstructions you will be creating from the raw data above. They are provided in order to allow progress to be made on visualisation and investigations into 3D printing while you are developing the higher quality reconstructions.

**`xtreme`** is a python class for handling the raw data from the Xtreme scanner. Initialising the class using, for example, '`xt = xtreme('raw_data.rsq')`' will open the file and initialise the structure with information from that file. Typing '`vars(xt)`' will show you the parameters this contains and their values, and using this class with the following functions will allow access to the data from this file.

**`xt.get_rsq_slice`** is a function which will return the fan-based sinogram, or slice, for a given $z$ location. Typing '`f, fmin, fmax = xt.get_rsq_slice(100)`' will return the 100th slice `f` of size `xt.angles` × `xt.samples`. The corresponding calibration values `fmin` and `fmax` are arrays of size `1` × `xt.samples` since the calibrations are the same for every angle.

**`xt.get_rsq_scan`** is a function which will return an X-ray, or scan, for a given rotational angle. Typing '`y, ymin, ymax = xt.get_rsq_scan(100)`' will return the 100th reconstruction angle scan `y` of size `xt.scans` × `xt.samples`. The corresponding calibration values `ymin` and `ymax` are arrays of the same size as `y`.

**`xt.fan_to_parallel`** is a function which turns the fan-based sinogram into an equivalent parallel-based sinogram. Typing '`p = xt.fan_to_parallel(f)`' will take the fan-based sinogram `f` with size `xt.angles` × `xt.samples` and return the parallel-based sinogram `p` with size `xt.recon_angles` × `xt.samples`. The new sinogram `p` will cover exactly 180°.

**`xt.reconstruct_all`** is an incomplete function which should reconstruct all the data and output DICOM files for each reconstructed slice.

## 10 Reconstruction, visualisation and modelling tasks: weeks 3 and 4

The remainder of this project will focus on the completion of a hypothetical task based on the raw CT data which has been provided. Each group will work together on one task: some suggestions for possible tasks are given in Section 10.4. These suggestions are not intended to be prescriptive and you are welcome to think of your own. However, each task must cover some element of **all** of the following:

- Reconstruction - reconstruct and correct the raw CT data from the Xtreme scanner as well as possible and store this in DICOM format.
- Visualisation - take the reconstructed data and use volume rendering, reslicing, surface rendering, with any additional cropping, filtering etc. as required to generate appropriate visualisations of the data.
- Modelling - create a computer-based surface model or models derived in part from the CT data and in part designed by yourself and use these to simulate the process of 3D printing from CT data.

These elements can either be handled separately by each group member, or in combination: however the task will be presented as a group.

### 10.1 Reconstruction

Reconstruction of a single slice of the Xtreme data can be achieved in a simliar fashion to the CT simulator which has already been developed. With the additional provided functions for reading the raw data and also converting the fan-based sinogram to a parallel-based sinogram, you will also need to modify at least the `ct_calibrate` and `hu` functions as well as the appropriate part of the `scan_and_reconstruct` function. The incomplete `reconstruct_all` function provides some hints as to how this single-slice reconstruction can be run over all the appropriate slices in the data to generate a complete 3D data set.

You might want to investigate possible reconstruction filters other than the Ram-Lak filter, or raised-cosine filter, and also what the appropriate filter cut-off frequency $\omega_{\max}$ is for this data. You could also investigate to what extent beam hardening is present in this data, and how you might be able to correct for this.

Reconstructing in this manner is not quite correct, since we have had to presume that the X-rays were parallel in the $z$ direction (right-hand diagram in Fig. 13). There are a variety of techniques for reconstructing this data correctly: the most well known is *Feldkamp-Davis-Kress* (Feldkamp et al., 1984) or FDK reconstruction. Various techniques exist to implement this for cone-beam CT. If you like a challenge, you could try the *Parallel FDK* or P-FDK technique by Turbell (2001)[7], which is also based on converting to parallel-based sinograms in the $x$-$y$ plane, but makes use of multiple slices ($z$ values) for each reconstruction. The idea is, for each reconstruction, to just use data from each angle and each slice in a particular $z$-fan where it actually intersects the reconstruction plane.

Alternatively, or in addition, to the full reconstruction above, you could also investigate to what extent there are errors in the reconstructed data due to the assumption of parallel slices. You will need to look at the

[7] `http://people.csail.mit.edu/bkph/courses/papers/Exact_Conebeam/Turbell_Thesis_FBP_2001.pdf`

reconstructed data very carefully, and consider where these errors are most likely to be, in order to determine these correctly.

## 10.2 Visualisation

Fortunately, there is quite a lot of free software for visualising medical 3D data sets and the intention is that you will use appropriate software to achieve the best visualisations you can from the reconstructed 3D CT data in DICOM format. Appropriate selection of software, and learning how to use it to visualise the CT data you have, forms a part of the work of this project: in practice this is exactly what would be required if you were faced with the problem of needing to visualise a medical data set. Equally, getting the data into an appropriate format for the software is an important issue which can often occupy as much time as learning to use the software. Fortunately most medical visualisation software will understand the DICOM format, and that is why we have made use of this format for this project.

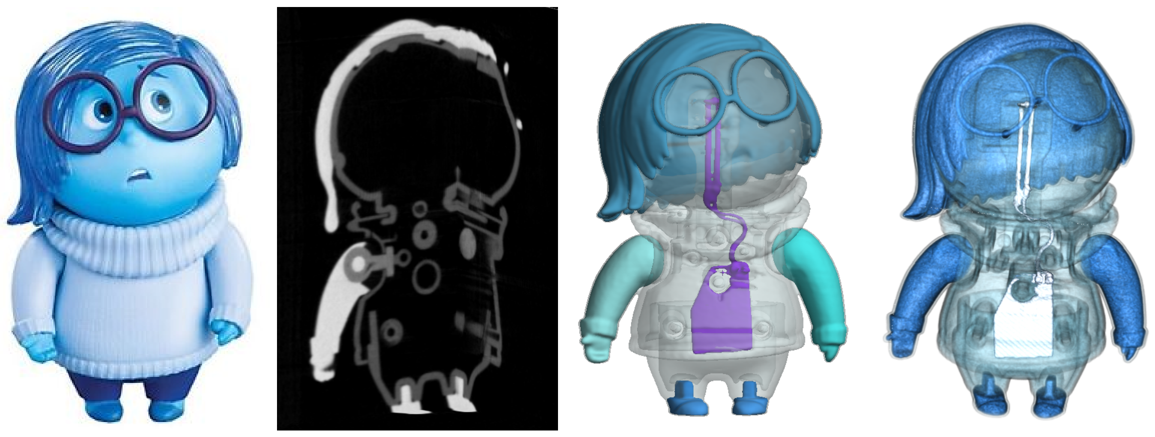


Figure 14: Sample visualisations. The subject (left) is visualised using multiple reslices (centre left), surface rendering (centre right) and volume rendering (right). Colours in the renderings are entirely false, i.e. made up — these can't be seen using CT.

Since you are required to *use* rather than *implement* appropriate visualisations, you only need an overview of the possibilities rather than the implementation details. An appropriate overview was given in one of the lectures from the Bioengineering option of Part IB Paper 8: see pages 15–28 from handout 2 of 'Imaging the Eye'[8]. There are also many overviews of this material online. The main three options are *reslicing*, *surface rendering* and *volume rendering*, examples of which are shown in Fig. 14:

- *Reslicing* involves creating a 2D cross-sectional image by cutting through the 3D data at any location or angle. This can be displayed on its own or in the appropriate location in 3D. Since the pixels on this image are not exactly aligned with the 3D data, some *interpolation* or *approximation* is required, the type of which will affect the quality of the generated reslice. There is an example of three orthogonal reslices in the second image of Fig. 14.

- *Surface rendering* involves extracting and displaying a computer-based representation of a surface contained in the data. Extraction involves first *segmenting* the data to define what is inside and what is outside the surface: this can be by simple selection of data thresholds (for instance anything over 1000 HU

[8]https://www.vle.cam.ac.uk/course/view.php?id=69951

would normally represent bone or metal), or by manually drawing contours on each of the original CT cross-sections, or by a number of other means. Once the 'object' of interest has been defined, the surface is then *triangulated*: a set of interconnecting triangles are created which exactly represent the border between the object and the surrounding tissue, e.g. by the *marching cubes* algorithm. The surface can then be displayed in 3D by assigning some material properties (for instance colour and shininess) and simulating light sources at various positions. What view you get of the surface is defined by a simulation of a camera location, and usually the rendering is fast enough that the view can be changed interactively. There is an example of several surfaces in different colours in the third image of Fig. 14.

- *Volume rendering* involves creation of a 3D visualisation which potentially involves every point in the 3D data. Every possible data value (Hounsfield Unit) is assigned a colour (for instance white-yellow for bone, red for blood, etc.) and also a transparency (so for instance you might want to 'see through' tissue but not bone). The rendering is constructed by pretending that light is travelling through the data and responding to these parameters at each point. The appearance of embedded surfaces can also be added by picking out regions in the data with high gradients, and 'lighting' these in the same way as in surface rendering above. Hence volume rendering requires the setting of a lot of parameters for each HU value, and this is usually controlled by user interfaces which allow the drawing of curves, for instance material colour against density. There is an example of a volume rendering in the right-hand image of Fig. 14.

You might also want to investigate *filtering* of the data before such visualisation to remove noise or enhance certain features in the data, or *cropping* to remove some regions of data and hence allow better visualisation of other areas of interest.

Some of the free software which might be useful for this task is listed below: this is not intended to be exhaustive and you will probably need to use several of these:

**MicroDicom** This is a good general purpose viewer just for looking at DICOM slices and the extra information in DICOM files: it does not perform 3D rendering (Windows).

**3DimViewer** This can produce simple volume and surface renderings of DICOM data (Windows/MacOS).

**Osirix** A general purpose medical DICOM viewer which will reslice and volume-render the data. There is also a freeware version called **Horos** (MacOS).

**ImageVis3D** This is quite a sophisticated volume rendering application which can load DICOM data, and can also export a surface mesh (Windows/Linux).

**Stradview** Can reslice a DICOM data set or a sequential set of images. Can also be used to render and export surfaces, particularly useful if you need some manual interaction in segmenting the data (Windows/Linux).

**3DSlicer** Can load DICOM and other types of data. Will do volume rendering and also data filtering and interpolation which can be overlayed in 3D. It can also extract surfaces from the data (Windows/MacOS/Linux).

**Paraview** Mainly a general purpose data visualiser and surface editor but can also load DICOM data. Volume rendering is a fairly recent addition (Windows).

**Meshlab** This is a good general purpose tool for processing and visualising surfaces (meshes) once they have been created (Windows/MacOS/Linux).

**Blender** This can produce extremely impressive production quality surface visualisations, though doesn't load data in DICOM format. Quite a complex user interface (Windows/MacOS/Linux).

**3D Builder** This is an app available for Windows 8 and later which can edit and change the location of surfaces: it is particularly designed for preparing objects for 3D printing. Windows 10 also includes some similar software: **Mixed Reality Viewer** and **Paint 3D** can also now view models (Windows).

## 10.3 Modelling

Whilst, sadly, it will not be possible to actually create physical 3D printed models from CT data this year, we can still go through the many steps required before a 3D model can be created, and we can still simulate the process of actually producing a 3D print.

The first step before creating a physical 3D model from the CT data is to extract a computer-based surface, or mesh, from the data. To do this well is actually quite difficult, and you will need the software listed in the previous section to help with this. You will almost certainly need the mesh to be processed in some way, either to smooth it, or ensure there are no holes or very thin parts, before a physical model can be created from it.

You should also ensure that you have changed the mesh in some way, for instance creating a new part for the scanned model, or creating a support designed to fit part of the model. Once again, the software in the previous section can help with this. You can either directly edit a surface which has been created from the CT data, or you can make careful measurements from the CT-derived surface and then use these measurements to create additional surfaces which are designed to fit, or otherwise work with, the CT-derived surface. You may also need to convert the mesh into an appropriate file format.

The Dyson Centre for Engineering Design has several RS IdeaWerk 3D printers, and also some Ultimaker printers, which are all described by guides which are available from the Dyson Centre website[9]. You will see from these guides that, before a surface can be printed, it has to be checked for consistency, and then converted to a set of instructions which actually tell the 3D printer exactly what to print. The following are examples of software which can be used for this purpose:

**Ultimaker Cura** Fairly simple interface, good positioning and scale of model, 3D preview features for various options, plus estimated time and material use for the Ultimaker printer (Windows/MacOS/Linux).

**Slic3r** Very flexible interface, options to control just about everything, though a little harder to use. Lots of previews, including 2D and 3D slices of various forms. Generic software, not designed for a specific printer (Windows/MacOS/Linux).

**Repetier** Similar to Ultimaker, but designed for a different printer, does give estimated time and material use (Windows/MacOS/Linux).

You should use the software above to investigate all aspects of the printing process, including for instance the initial model alignment and size, what difference it makes to use different diameter (profile) materials, infills and the infill percentage, use of a base (raft) and also whether additional supports are needed. You should also investigate the different scales at which the model can be printed, and how this affects the accuracy of the printed model. Whilst 3D printing is relatively inexpensive, it can potentially take several hours to produce a model of a few centimetres height. Hence it is also useful to know both how much material is being used and how long the estimated print time is.

[9]`http://www.dysoncentre.eng.cam.ac.uk/3d-printing`

### 10.4 Suggested tasks

Some suggestions for tasks follow — these are designed to link together some element of each of the preceding sections, given the available CT data. They are not restrictive and in all cases you should investigate the best reconstructions, visualisations and models that you can: do not let the specific task limit what you produce.

- Limb replacement - create a new part which will connect to the subject at some point but is possibly altered at others, for instance a new false leg. The visualisations could help with designing this, showing what it would look like when attached, or assisting with any required ‘surgery’ to attach it.
- Radiotherapy plan - during radiation treatment for cancer, it is vital to know exactly what is being treated, and the radiation dose at various distances from this in 3D. So outline a feature which you want to ‘treat’, and produce visualisations which show this *in situ* as well as what data is within a range of distances (e.g. 5 mm and 10 mm) from it. Once the ‘plan’ has been completed, you also need to build an appropriate ‘jig’ which will keep the right part of the subject ‘still’ during treatment.
- Catch the criminal - the subject has stolen something but somehow managed to get themselves CT scanned in the process, so all we have is the CT data. From this, attempt to reconstruct appropriate visualisations that might help you catch them, and various physical properties (height, weight, etc.). You could also for instance create a physical boot in order to check for matches for any boot impressions that might have been found.

## 11 Group presentation

Presentations are intended to give your group the chance to show your visualisations and models to the other students, and explain why and how you created them. During the last morning session, each group will have ten minutes to present their work, including a brief opportunity for questions. The format is flexible but it is expected that everyone in the group will take some part, either in preparation of the slides or presentation of the material.

Given that these presentations will need to be remote via Zoom this year, you will need to hand in a complete copy of your slides (one set per group, please save these as a pdf document) by 5 pm on Sunday 31st May.

## 12 Individual submission: final report

This as an individual report, minimum font size 10 pt, and a maximum of 10 pages in total. Your final report should describe the task-based activities in the last two weeks of the project, from Sections 9 and 10. You should place this in the context of the whole project, but do not repeat all the details from your interim report: you can refer back to it where necessary.

The report should summarise the whole task, including aspects of the work of other group members, but you should concentrate on those parts in which you were directly involved, and be clear what was your own work and what was not. In addition to describing what you have done and why (including the selection and use of any software you needed), it should also contain discussion of what you have learnt regarding the various advantages and limitations of CT data; visualisation techniques and what can and can not be achieved with them; and the potential benefits or otherwise of 3D printing from medical data. You should also include a brief discussion of the benefits and issues of working in a group, how your group was organised and how well it functioned.

The instructions about writing and referencing in Section 8 also apply to this report. Where you want to summarise work performed by another group member, you are welcome to borrow appropriate images / photos / plots from them, but **you should clearly state that these were from another group member**. The captions, and all the accompanying text, must always be written by yourself.

This report is due in by 4 pm on Thursday 4th June.

## References

Feldkamp, L., Davis, L., Kress, J., 1984. Practical cone-beam algorithm. JOSA A 1 (6), 612–619.

Ramachandran, G., Lakshminarayanan, A., 1971. Three-dimensional reconstruction from radiographs and electron micrographs: application of convolutions instead of fourier transforms. Proceedings of the National Academy of Sciences 68 (9), 2236–2240.

Suetens, P., 2009. Fundamentals of medical imaging, 2nd Edition. Cambridge University Press.

Turbell, H., 2001. Cone-beam reconstruction using filtered backprojection.

Zeng, G. L., 2014. Revisit of the ramp filter. In: 2014 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC). IEEE, pp. 1–6.