Pinnacle

Infrastructure and connectivity
by Michael Kantor

Beta servers

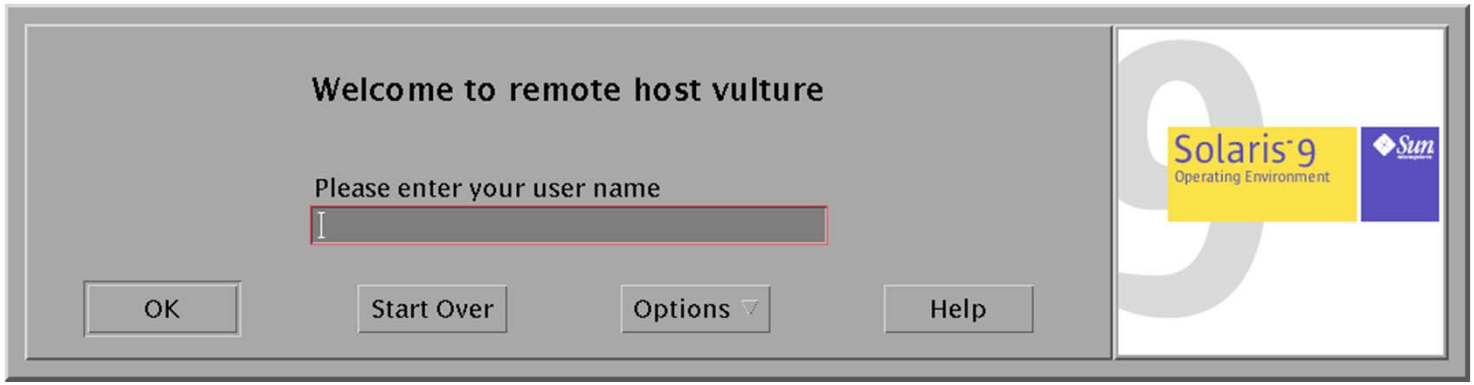
- Older: SPARC Solaris9
  - vulture
  - vulture2
- Newer: x86 Solaris10
  - These share a patient database: raptor, raptor2
- Latest: Pinnacle v9, x64 Solaris10
  - raptor4

Solaris display

- X-Window system is used for all displays
- Any computer can display X-Window sessions with appropriate software
  - Windows: X-Win32, Cygwin, X386, XFree86, etc.
  - Linux / UNIX: native support
  - Apple OS X: somewhat native, needs some configuration
    - http://developer.apple.com/Darwin/runningX11.html
  - More info
    - http://en.wikipedia.org/wiki/X_Window_System

Login accounts

- Nearly all accounts have the password, **p3plan**
- To use a beta version of Pinnacle, login using the **pinnbeta** username
- Clinical versions use the **p3rtp** username

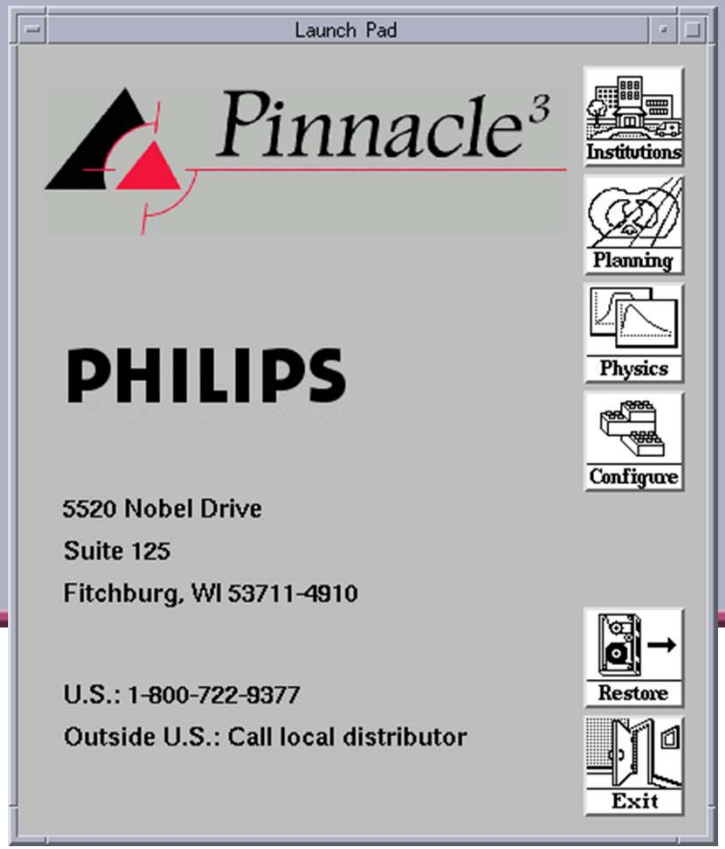


StartPinnacle

- The command **StartPinnacle** can be used to start the LaunchPad from the terminal.
  - Messages from Pinnacle are output to the terminal, very useful for debugging your scripts or exploring
  - Everything in Solaris / UNIX is case-sensitive!
- The LaunchPad is a “parent” application that browses Institutions and Plans, imports/exports files, and starts Pinnacle on a chosen plan.

StartPinnacle

```
xterm
raptor.P3RTP <1>: StartPinnacle
/usr/local/adacnew
Warning: Cannot convert string "-linotype-palatino-medium-r-normal--0-300-75-75-
p-*-iso8859-1" to type FontStruct
Warning: Cannot convert string "-urw-itc zapfdingbats-*-*-*-*-20-*-*-*-*-*-*-*"
to type FontStruct
stty: : Invalid argument
stty: : Invalid argument
stty: : Invalid argument
stty: : Invalid argument
stty: : Invalid argument
stty: : Invalid argument
```



Pinnacle organization

- Database
  - Institutions
    - Patients
      - Image Datasets
      - Plans
        - Trials
- No patient can be opened by more than one instance of Pinnacle, even if working on different plans

Pinnacle

scripting

by Michael Kantor

Scripting

- All of Pinnacle runs on scripts
  - Every button, list, box, etc. uses scripts to interact
- Internally, scripts are listened to and acted on by C code and plugins.
  - If you choose to “create” for Pinnacle in C, using the plugin interface, there are more options to get creative and a lot more math, BUT your plugin will not run in a clinical version
- Pinnacle has a low tolerance of scripts that don’t behave like it expects them to, it is very easy to cause a crash
- Everything here is relevant to versions 8, and probably 9. Everything here should function in versions 6 and 7, tread carefully with anything older.

Scripting

- Root objects [case-sensitive!]
  - PlanInfo
  - TrialList
  - RoiList
  - PoiList
  - DVHList
  - VolumeList
  - WindowList
  - ViewWindowList
  - KeyDependencyList
  - CtToDensityTableList
  - PrinterList
  - Store
  - …

Scripting

- General notes
  - Scripts are plain-text, no need to compile
  - A command can span multiple lines, but MUST end with ;
  - There can only be one = for any single command
  - There are no **for** or **while** loops, but it is possible to target a command at each item in a list
  - Comments follow // or can be grouped between /* and */ to span multiple lines
  - # is an inline escape, telling Pinnacle that what follows is a literal string or a special command, but not an object name

Scripting

- Objects have hierarchy and contain other objects and properties
  - TrialList Object
    - Trial Object
      - Name: Trail_1
      - PrescriptioList Object
        - Prescription Object
          - Name: Prescription_1
      - BeamList Object
        - Beam Object
          - Name: Beam_1
        - Beam Object
          - Name: Beam_2

```
TrailList = {
    Trial = {
        Name = "Trial_1";
        PrescriptionList = {
            Prescription = {
                Name = "Prescription_1";
            };
        };
        BeamList = {
            Beam = {
                Name = "Beam_1";
            };
            Beam = {
                Name = "Beam_2";
            };
        };
    };
};
```

Scripting

- We can access various settings and values
  - Find out how many beams the trial has
    ```
    Echo = TrialList.Current.BeamList.Count;
    ```
  - Store and print the active trial's name to the console
    ```
    Store.StringAt.trial_name = TrialList.Current.Name;
    Echo = Store.StringAt.trial_name;
    ```
  - Get the volume (cm^3) of the first ROI
    ```
    Echo = RoiList.#"0".Volume;
    ```
  - Sum the volume of the first and last ROIs, then print to console
    ```
    Store.FloatAt.volume_A = RoiList.First.Volume;
    Store.FloatAt.volume_B = RoiList.Last.Volume;
    Store.At.volume_A.Add = Store.FloatAt.volume_B;
    Echo = Store.FloatAt.volume_A;
    ```

Scripting

- Store
  - The Store is an object where you can place temporary variables
  - Nothing in the Store will be saved with the plan
  - You can view and modify the Store with this command:
    ```
    WindowList.MyStoreEditor.CreateStoreEditor =
    "Store.Address";
    ```
  - The .Address property is object's memory pointer
  - Stores in other objects can be used, like in a beam, so the Store Editor needs to know which one you want to view:
    ```
    TrialList.Current.BeamList.First.Store.StringAt.my_str
    ing = "hello world";
    WindowList.MyStoreEditor.CreateStoreEditor =
    "TrialList.Current.BeamList.First.Store.Address";
    ```

Scripting

- A note on At and FloatAt or StringAt
  - If there is an action being performed on the variable, use At, otherwise, use FloatAt or StringAt, whichever is appropriate
  - Example
    - Echo a variable
      ```
      Echo = Store.StringAt.some_string;
      ```
    - Same thing, but .Value is an action that gets the value of the variable, which is then passed to Echo
      ```
      Echo = Store.At.some_string.Value;
      ```

Scripting

- Commands on the same object can be nested
  - ```
    Store = {
        FloatAt.alpha = 0.007297;
        FloatAt.e = 2.7183;
        StringAt.foo = "bar";
    };
    ```
  - ```
    TrialList.Current.PrescriptionList.First = {
        Name = "RUL 60Gy";
        PrescriptionDose = 200;
        NumberOfFractions = 30;
    };
    ```

Scripting

- All lists share some properties and actions
  - `.Count`, the size of the list, n
  - `.Current`, reference to the currently selected object
  - `.CurrentIndex`, current integer index into the list
  - `.First`, object at index 0
  - `.Last`, object at index n-1
  - `.IsCurrent`, logical check
  - `.NextCurrent`, set the next object to be current
  - `.LastCurrent`, set the previous object to be current
  - `.MakeCurrent`, action to set some other index to be current
  - `.#"*"`, this will apply the command to every item of a list
    - For example, update Isocenter of all beams in the plan
      `TrialList.#"*".BeamList.#"*".IsocenterName = PoiList.First.Name;`
  - etc… there are many

Scripting

- Logical operators
  - ```
    IF.RoiList.Count.LESSTHAN.#"10".
    THEN.CreateNewROI.
    ELSE.Echo = "There are now 10 ROIs";
    ```
  - ```
    Store.StringAt.orientation = "Transverse";
    IF.ViewWindowList.Current.Orientation.
    STRINGEQUALTO.Store.StringAt.orientation.
    THEN.ViewWindowList.Current.Orientation = "Sagittal";
    ```
- Numeric logic test
  - EQUALTO, NOTEQUALTO, GREATERTHAN, LESSTHAN, GREATERTHANOREQUALTO, LESSTHANOREQUALTO
- String logic test
  - STRINGEQUALTO, STRINGNOTEQUALTO, IS, ISNULL, Contains

Scripting

- Loops
  - For any list, there is a way to sequentially set each item current and run a script
  - Example: echo the name and volume of each ROI
    - Create a second file, I will call it echo_roi_volume.Script, and it will contain:

```
Echo = RoiList.Current.Name;
Echo = RoiList.Current.Volume;
```

    - Then in my main script file

```
RoiList.ChildrenEachCurrent.#"@".Script.ExecuteNow =
"/some/path/echo_roi_volume.Script";
```

  - Example: set the color to red for any empty ROI

```
RoiList.ChildrenEachCurrent.#"@".
IF.RoiList.Current.#"!HasCurves".
THEN.RoiList.Current.Color = "red";
```

Scripting

- Key Dependency
  - Tell Pinnacle to do something whenever some value is changed
  - Not all keys respond to this, but most will
  -
```
KeyDependencyList.CreateChild = "";
KeyDependencyList.Last = {
    Name = "new_trial_alert";
    KeyString = "TrialList.Count";
    AddAction = "Echo=#Hey, you just added or removed a Trial!";
};
```

Scripting

- Executing other scripts
  - From file
    ```
    Script.ExecuteNow = "/some/path/my_script.Script";
    ```
  - From the content of a variable
    recall, the # is to make sure Pinnacle treats A as a string, not an object
    ```
    Store.StringAt.command = "TrialList.Current.BeamList.First.Name
    = #A";
    Store.At.command.Execute = "";
    ```
- Executing UNIX programs
  - `SpawnCommand = "env | grep DISPLAY";`
  - `SpawnCommand = "uptime";`
- Execute UNIX command, but let Pinnacle continue without waiting for an exit code
  - Get environment variables from the server vulture2
    ```
    SpawnCommandNoWait = "rsh vulture2 env";
    ```

Scripting

- Saving data
  - The `.Save = "/some/path/file.txt";` action, when applied to nearly any list or object, will save all contents to the given path
  - For best practice, create a directory inside the home directory, using your last name. Keep scripts there and save files there as well.
    - For example, I tend to use /home/pinnbeta/kantor or /home/p3rtp/kantor
  - Do not place 100s of Mb there, or you risk crippling the system for all users. Monitor your own usage and delete what you no longer need.
- Accessing data
  - FTP can be used to access the file system, using the same usernames, passwords, and hostnames, like:
    - ftp://pinnbeta:p3plan@vulture2

Scripting

- Some practical examples
  - Show the Beam Settings Panel window
    ```
    WindowList.BeamsPanel.Create = "";
    ```
  - Save the current Trial
    ```
    TrialList.Current.Save = "/tmp/current_trial";
    ```
  - Turn on verbose status messages
    ```
    IsVerbose = 1;
    ```
  - Warn if there is a problem
    ```
    IF.TrialList.Current.Name.Contains.#"$".
    THEN.WarningMessage = "Please remove the $ from the Trial Name";
    ```
  - Save the dose grid
    ```
    TrialList.Current.DoseGrid.SaveDataToFile = "/tmp/the_dose";
    ```
  - Save ROI voxel mask
    ```
    RoiList.Current.SaveBinaryVolumeAs = "/tmp/roi_mask";
    ```

Pinnacle

data manipulation with MATLAB

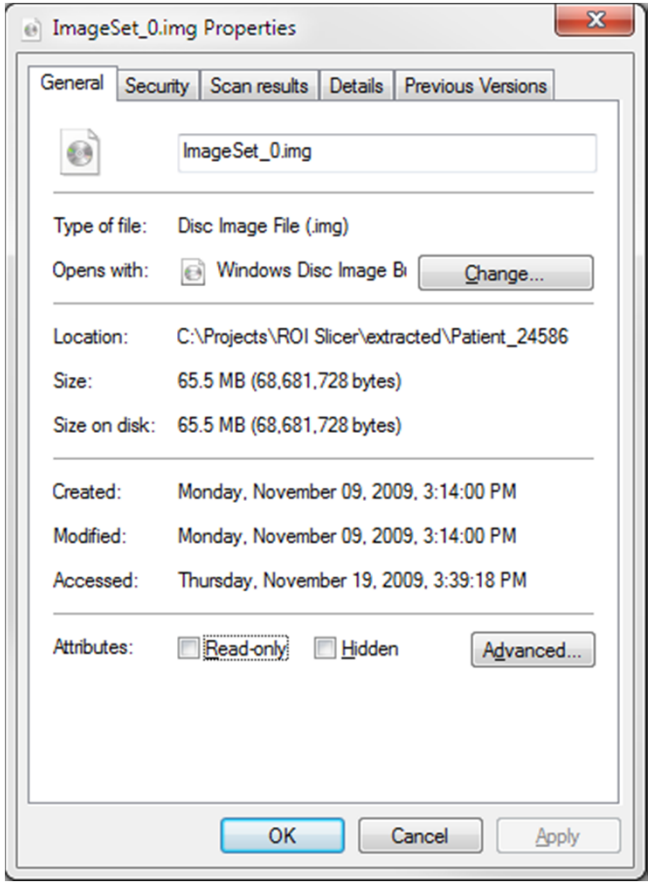
by Michael Kantor

Read Image Datasets

- The ImageSet_#.header files, in the Patient folder, contain important information regarding the ImageSet_#.img files:
  - `byte_order = 0`
    - 0 means Little Endian, 1 is Big Endian
  - `bytes_pix = 2`
  - `x_dim = 512`
  - `y_dim = 512`
  - `z_dim = 131`
  - `x_pixdim = 0.097656`
  - `y_pixdim = 0.097656`
  - `z_pixdim = 0.300003`
  - `x_start = -24.951172`
  - `y_start = -68.651169`
  - `z_start = 93.000000`
- Origin is top-left-rear

Read Image Datasets

- File only contains the values, increasing fastest in order by X, Y, Z
- Check:
  bytes_pix × x_dim × y_dim × z_dim
  2 × 512 × 512 × 131 =
  68,681,728 bytes
- 2 bytes per voxel, it's a signed integer
  In MATLAB, this is int16



Read Dosegrids

- Important Script keys
  - `TrialList.Current.DoseGrid.VoxelSize.X`
  - `TrialList.Current.DoseGrid.VoxelSize.Y`
  - `TrialList.Current.DoseGrid.VoxelSize.Z`
  - `TrialList.Current.DoseGrid.Dimension.X`
  - `TrialList.Current.DoseGrid.Dimension.Y`
  - `TrialList.Current.DoseGrid.Dimension.Z`
  - `TrialList.Current.DoseGrid.Origin.X`
  - `TrialList.Current.DoseGrid.Origin.Y`
  - `TrialList.Current.DoseGrid.Origin.Z`
- The files are in 4-byte (single precision) float format

Read Dosegrids

- All files are in Trial.Binary.### filenames in the Plan folder
- Summed and weighted Dosegrid
  - `TrialList.Current.DoseVolume = \XDR:0\;`
    means the dose is in Trial.binary.000
    but, this key is not guaranteed to exist
- Individual beam Dosegrids
  - `TrialList.Current.BeamList.Current.DoseVolume = \XDR:4\;`
    Dose specifically from this beam is in Trial.binary.004
  - `TrialList.Current.BeamList.Current.MonitorUnitsValid`
    If this is 1, then dose is in units of cGy/MU, otherwise cGy

Read 3D volumes in MATLAB

```
% open file in read mode, using Little Endian
file_handle = fopen('ImageSet_0.img', 'r', 'l');

% read x_dim*y_dim*z_dim*bpp bytes from file_handle in the int16 format
data = fread(file_handle, x_dim*y_dim*z_dim*bpp, 'int16');

% release the file
fclose(file_handle);

% convert from 1D to 3D
data = reshape(cast(data, 'single'), x_dim, y_dim, z_dim);
```

View a slice

- From the CT grid already loaded, select slice 50 and display

```
image = data(:, :, 50);
imshow(image);
```

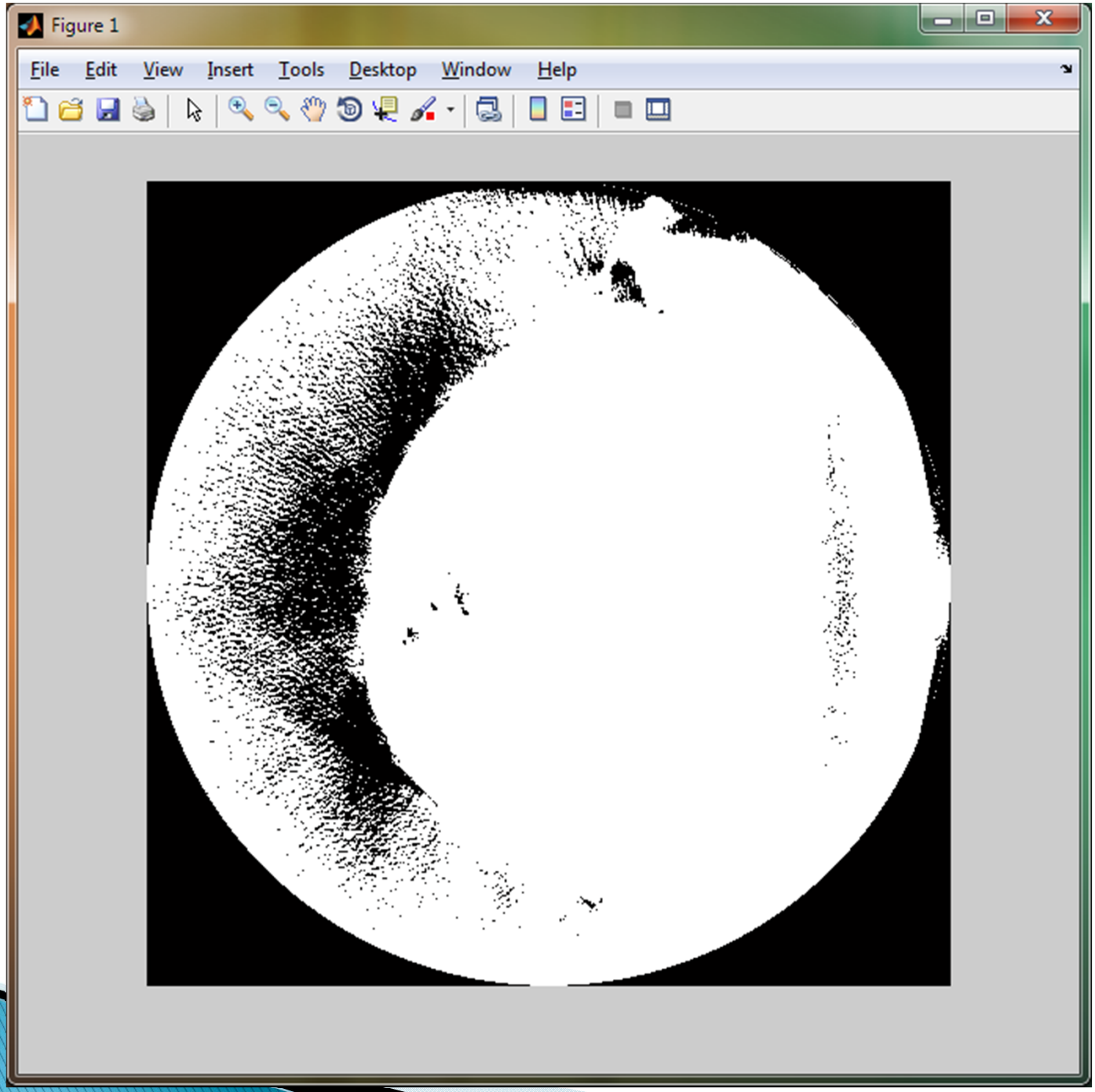


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help

View a slice

- Select a better window

```
imshow(image, [797 1199]);
```

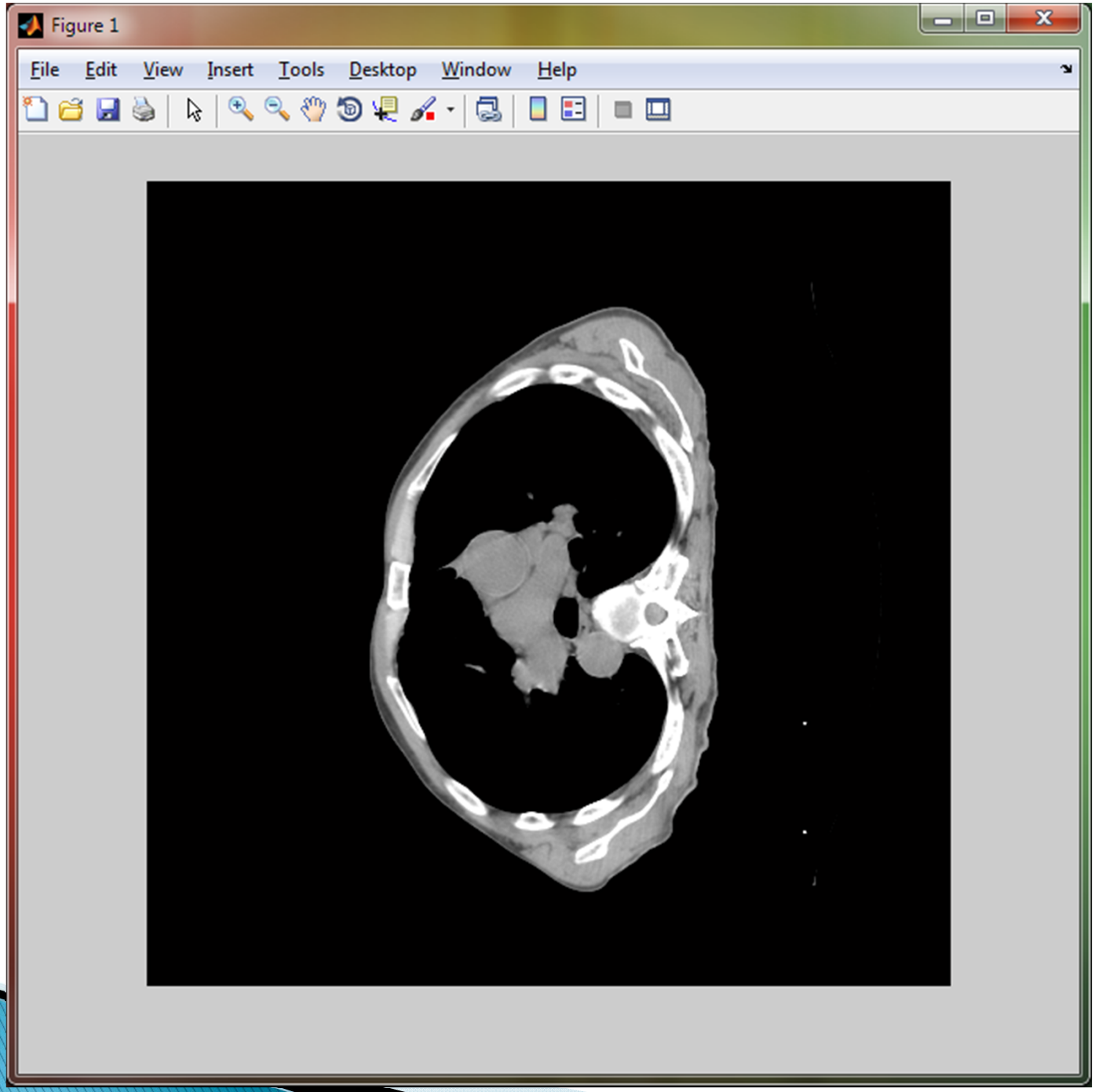


Figure 1
File Edit View Insert Tools Desktop Window Help

View a slice

- Almost there, just transpose

```
imshow(image', [797 1199]);
```

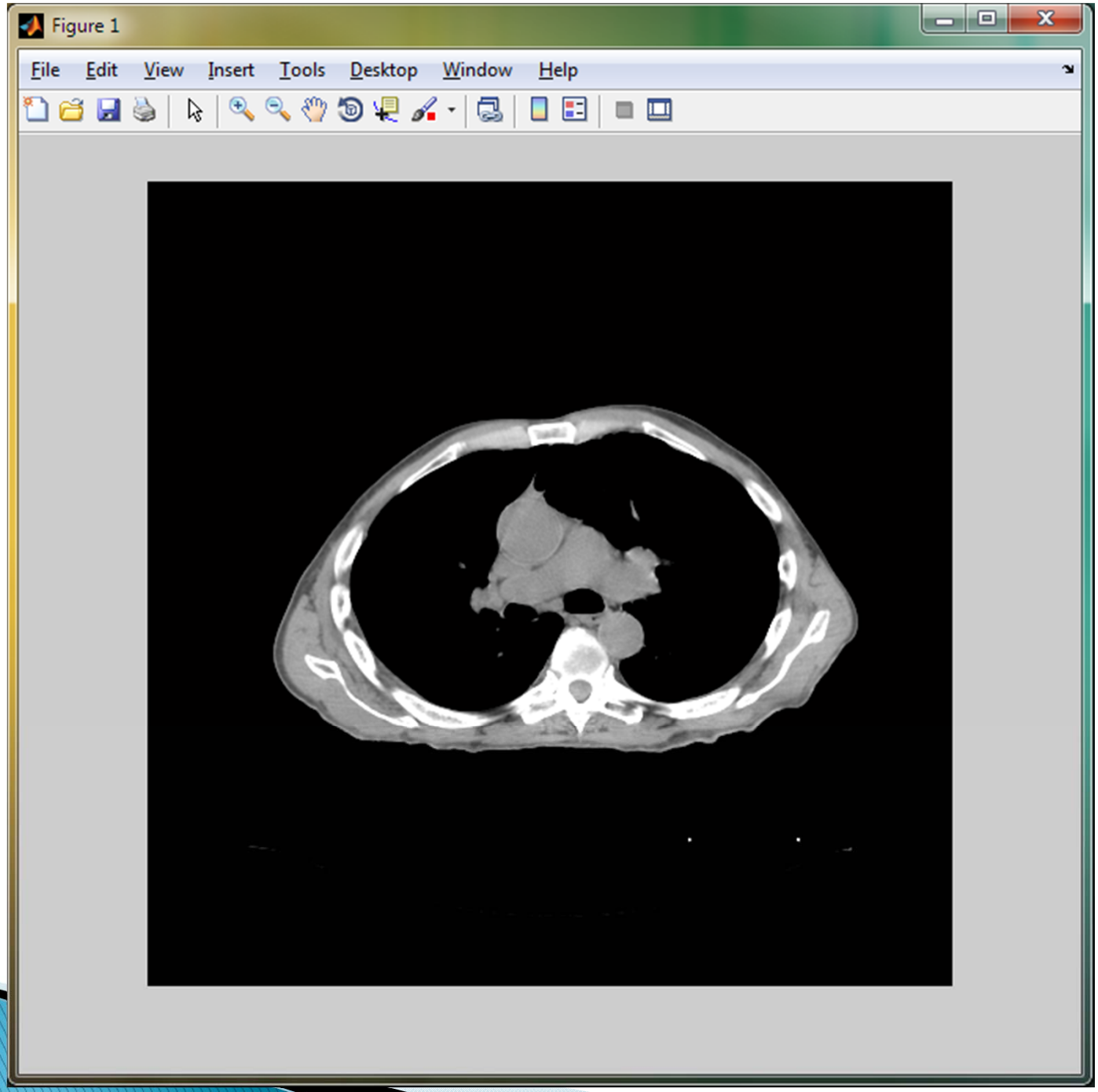


Figure 1
File Edit View Insert Tools Desktop Window Help

Make your own Dosegrid

- Isotropic

```
% make it match the dose grid size in the target plan
x_dim = 130; y_dim = 95; z_dim = 135;
% highest dose, in cGy
max_dose = 6000;

% origin
center = round([x_dim y_dim z_dim]/2);
% approx. distance from center to a corner
max_distance = norm(center);

% gradient distributions in X, Y, Z
[my, mx, mz] = meshgrid(...
    1-center(2):y_dim-center(2),...
    1-center(1):x_dim-center(1),...
    1-center(3):z_dim-center(3)...
);

% populate
dose = (mx.^2 + my.^2 + mz.^2) .^ 0.5;
% normalize
dose = abs(dose - max_distance) / max_distance;
% scale
dose = dose * max_dose;
```

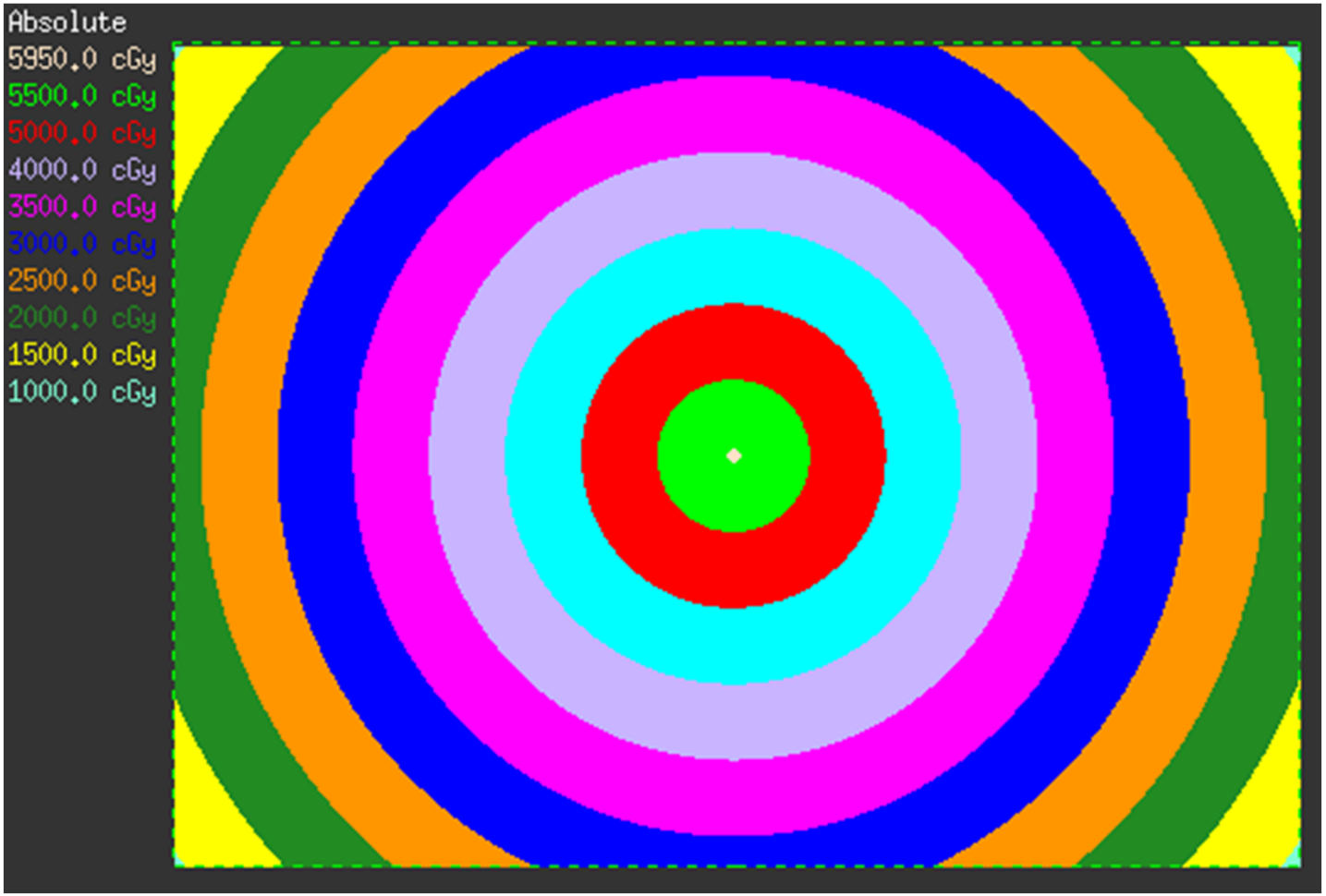
Make your own Dosegrid

- Write the result to a file that Pinnacle can read

```
dose_out = fopen('output.img', 'w', 'l');
for z=1:z_dim
    for y=1:y_dim
        fwrite(dose_out, dose(:, y, z), 'float32');
    end
end
fclose(dose_out);
```

- Read in the dose, which will stay in the plan while it is open, and until a beam has to be recalculated

```
TrialList.Current.DoseGrid.LoadDataFromFile = "/some/path/output";
```


Absolute
5950.0 cGy
5500.0 cGy
5000.0 cGy
4000.0 cGy
3500.0 cGy
3000.0 cGy
2500.0 cGy
2000.0 cGy
1500.0 cGy
1000.0 cGy

Make your own ROI

- ROIs are composed of curves, which are closed polygons of n>0 points, each with 2 coordinates.
- The slice coordinate is stated independently
  - This is usually in the Z; Pinnacle appears to support curves drawn in Sagittal or Coronal, but this is rarely used in practice
- The point coordinates are given to a Curve Editor, which then adds the new curve to a ROI
- Curves are converted to a voxel mask by Pinnacle, which is how it determines which dose voxels are inside which ROIs
- The union of overlapping curves is removed from the ROI mask

Make your own ROI

- Example

```
CreateNewROI = "";
RoiList.Last.Name = "Triangle";
RoiList.Last.EditCurve = {
    SliceCoordinate = 76.5;
    Orientation = "Transverse";
    Curve = {
        NumberOfPoints = 3;
        Points[] = {
            -5.71294, -33.9919,
            -5.61528, -33.9919,
            -5.51763, -34.0895
        };
    };
};
RoiList.Last.CopyEditCurveToNewCurve = "";
```

Make your own ROI

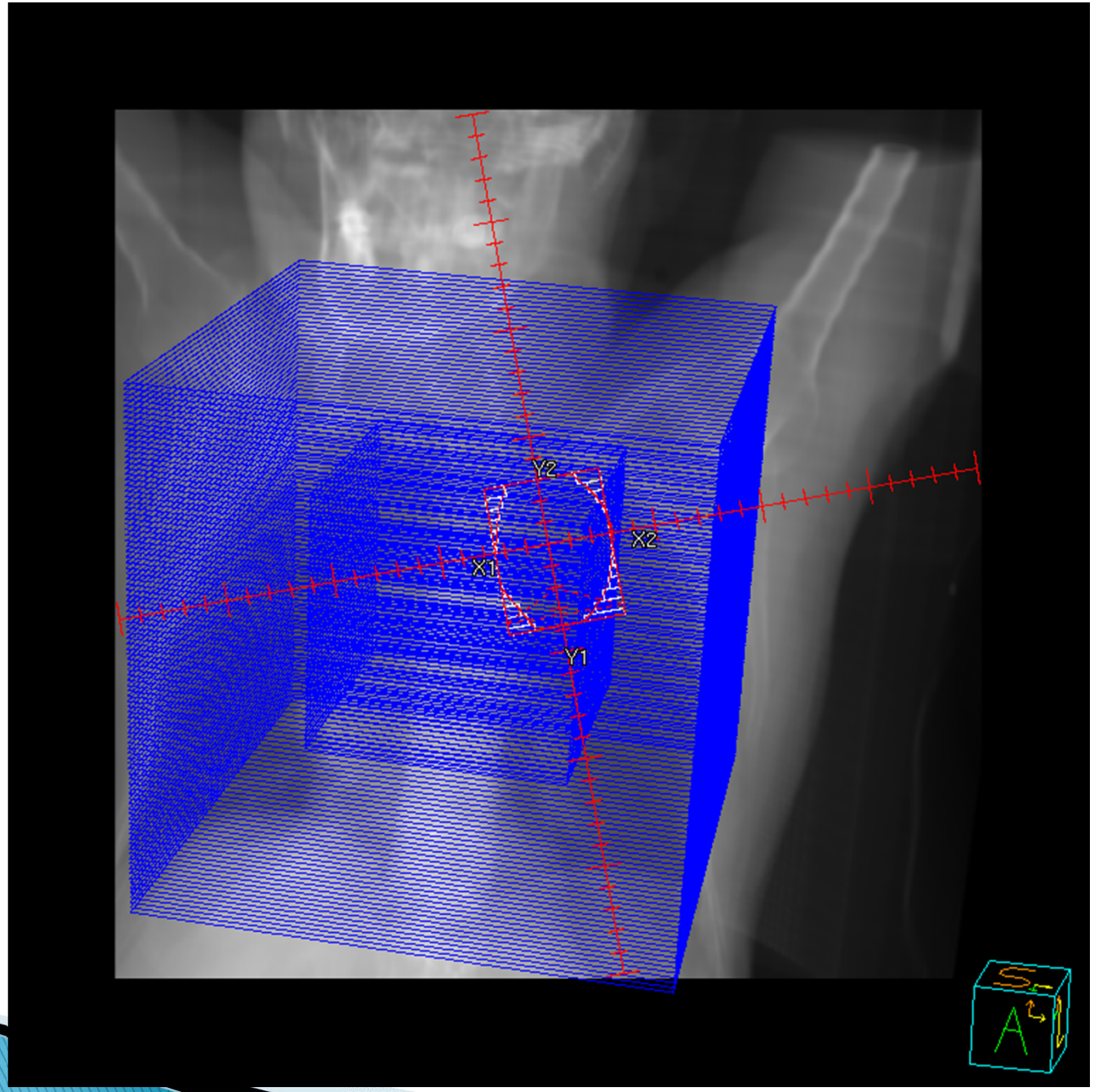
- In MATLAB, `bwboundaries` becomes very useful

- Example, a rectangular shell ROI

```
% the volume should be defined at CT coordinates
x_dim = 512;          y_dim = 512;          z_dim = 213;
x_pixdim = 0.097656;  y_pixdim = 0.097656;  z_pixdim = 0.250000;
x_start = -24.951172; y_start = -74.951149; z_start = -21.000000;

% allocate memory
roi_mask = zeros(x_dim, y_dim, z_dim, 'int8');

% mask off the shell
roi_mask(128:384, 128:384, 50:150) = 1;
roi_mask(192:320, 192:320, 75:125) = 0;
```

```
output = fopen('new_roi.Script', 'w');
fprintf(output, 'CreateNewROI = "";\n');
fprintf(output, 'RoiList.Last.Name = "shell";\n');
fprintf(output, 'RoiList.Last.Color = "blue";\n');

% iterate over z, curves will be transverse
for z=1:z_dim
    B = bwboundaries(squeeze(roi_mask(:, :, z)));

    if isempty(B); continue; end

    for r=1:length(B)
        coords = B{r};

        fprintf(output, 'RoiList.Last.EditCurve = {\n');
        fprintf(output, ['SliceCoordinate = ' num2str(((z - 1) * z_pixdim + z_start)) ';\n']);
        fprintf(output, 'Orientation = "Transverse";\n');
        fprintf(output, 'Curve = {\n');
        fprintf(output, ['NumberOfPoints = ' num2str(size(coords, 1)) ';\n']);
        fprintf(output, 'Points[] = {\n');
```

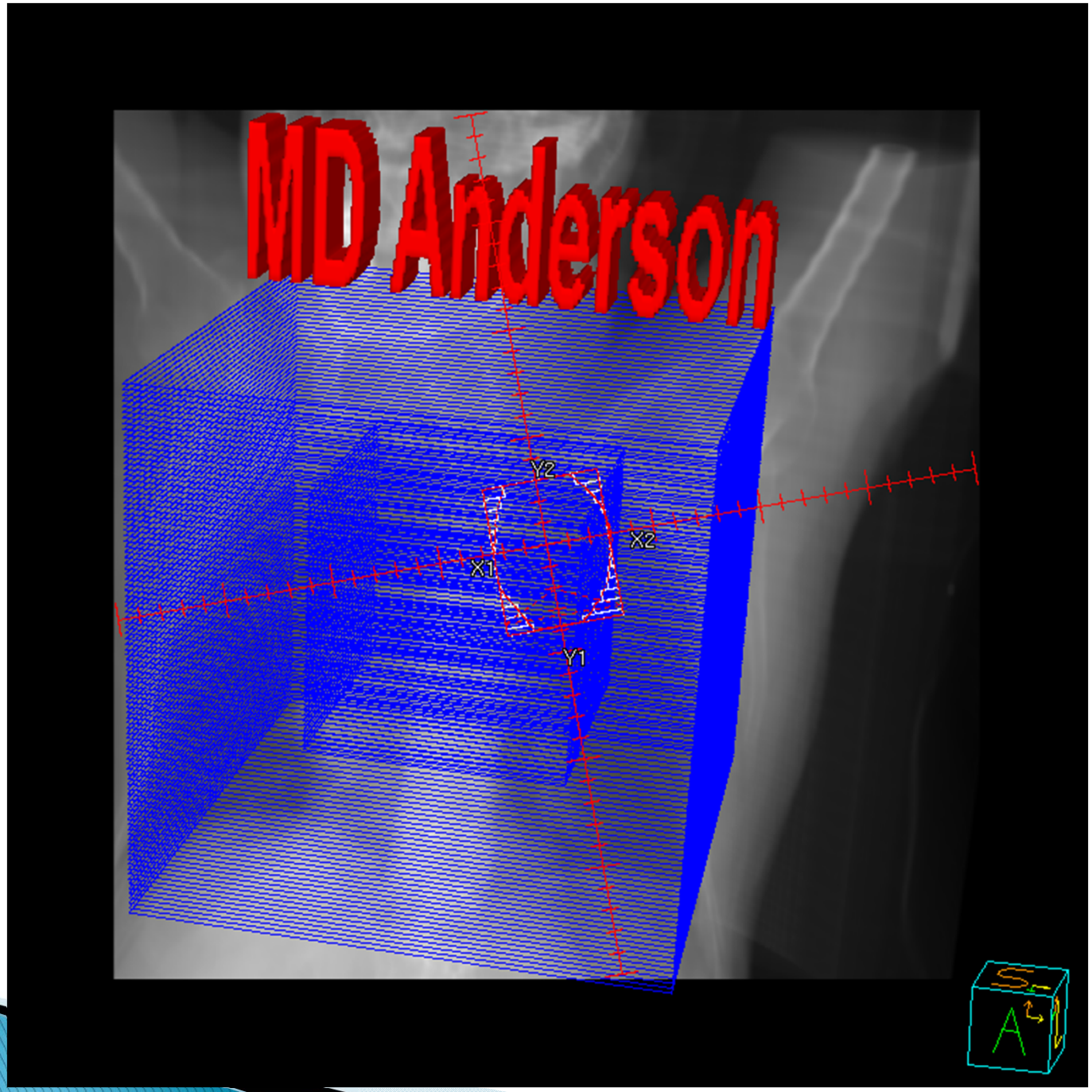
```
        % Go from MATLAB's Y-X to Pinnacle's X-Y, adjust for scale & origin
        % -1 is there because MATLAB indexes from 1, but Pinnacle from 0
        x = ((coords(:, 2) - 1) .* x_pixdim) + x_start;
        y = ((coords(:, 1) - 1) .* y_pixdim) + y_start;

        % convert to formatted string of X-Y data
        curve = [x y];
        temp = curve';
        temp = regexprep(num2str(temp(:)'), '\s*(\S+)\s+(\S+)', '$1, $2,\n');

        % print w/o trailing command and newline
        fprintf(output, temp(1:length(temp)-2));

        fprintf(output, '\n};\n');
        fprintf(output, '};\n');
        fprintf(output, '};\n');
        fprintf(output, 'RoiList.Last.CopyEditCurveToNewCurve = "";\n\n');
    end
end

fclose(output);
```


Y2
X2
X1
Y1
S
A

You can get creative!

- Just mask the area differently, and repeat the rest

```
% draw text in a figure
text(0.1, 0.1, 'MD Anderson', 'FontWeight', 'bold',
'FontSize', 30);
% capture the text
text_mask = getframe(gcf, [100 60 270 50]);
close all;
% re-structure the image data
text_mask = squeeze(text_mask.cdata(:, :, 1));
text_mask = 1 - text_mask / max(text_mask(:));
% note, size(text_mask) = [50 270]
% fill the mask volume with the text information
for x=223:255
    roi_mask(x, 121:390, 1:50) = text_mask';
end
```


MD Anderson
Y2
X2
X1
Y1