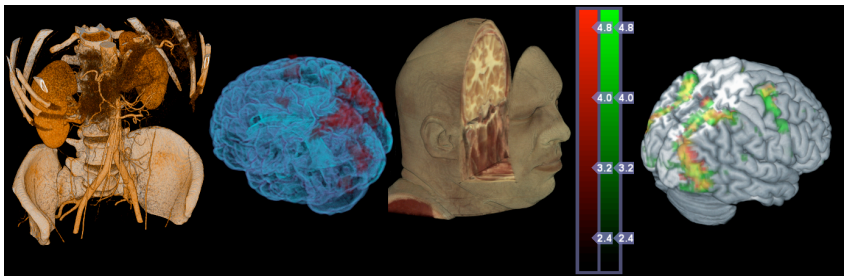


MRIcroGL manual

By Chris Rorden
Version 16 June 2017

1. Introduction

MRIcroGL is an intuitive viewer for medical images. This free and open source project provides tools for 2D and 3D display of images.

MRIcroGL supports the Windows, OSX or Linux operating systems. However, it requires a graphics card that supports volume rendering, with the appropriate driver installed.

2. Quick start

Double-click on the MRIcroGL program to launch the application. MRIcroGL can view NIfTI format images (these have the file extensions .hdr, .nii or .nii.gz). To view an image, simply drag-and-drop the image onto the program. You can change between rendering and slice modes by selecting the desired view from the "Display" menu (Figure 2.1).

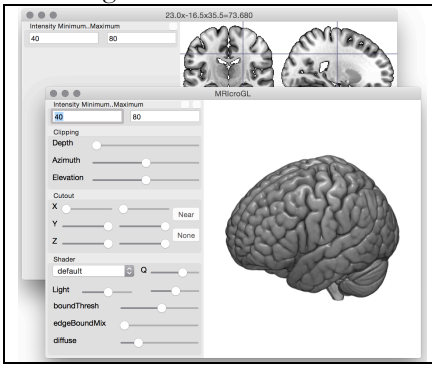


Figure 2.1 An example of the mni152_2009bet image that is distributed with MRIcroGL. On the left (back) is the 2D view, while the right (top) panel shows the 3D rendered view.

3. Adjusting color and transparency

Typical MRI scans and CT scans save data as a monochromatic intensity range. Crucially, different tissue types have different image intensities. For example, bones appear bright on a CT scan while soft tissue appears darker. Therefore, since different types of tissue have unique image intensities, you can choose to make some tissues invisible (transparent) while revealing other types of tissue. In addition, you can assign unique colors to specific image intensities, further helping to define different types of material.

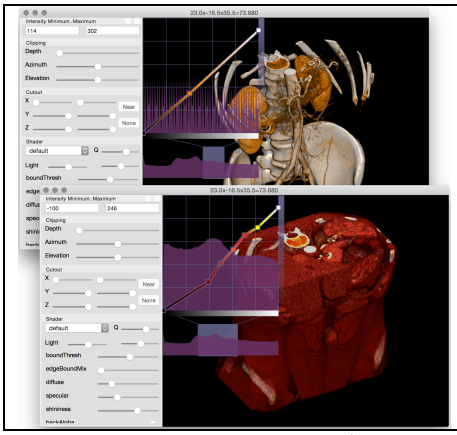


Figure 3.1 You can highlight different tissue using the 'color and transparency' window. Both the top and bottom panels show the abdo256 image. The top panel uses the Color/Scheme/Kidneys command to hide the darker tissue, revealing the bones and kidneys. In contrast, the bottom panel uses the Color/Scheme/Muscles color mapping that shows the darker tissue, so one cannot see beneath the muscles. Note that the top panel has selected intensities between 114 and 302, while the lower panel is adjusting values in the range –100..246. Within this range the color schemes also differ in the colors used, with the top scheme using oranges while the bottom scheme has predominantly reds and yellows.

Note that the Color/Scheme menu allows you to select from several predefined color schemes. For example, the top panel of Figure 3.1 shows the 'CT_Kidneys' color scheme, while the lower panel shows the 'CT_Muscles' color scheme. However, you can also manually change the color scheme. You can manually edit the intensity range you are interested in by adjusting the "Intensity Minimum…Maximum" values shown in the upper left of the toolbar. For example, note that the kidney color scheme is selecting a color range of 114..302 while the muscle color scheme includes darker material (-100..246).

If you want to further customize the color scheme you can choose the Color/ColorEditor menu item. A new histogram appears, as shown in Figure 3.1. Note that the color schemes are composed of nodes. For example, the muscle color scheme has a node that makes some tissue intensities to appear yellow. To change the color of a node, double click on the node. For example, you could make this muscle node appear blue instead of yellow. You can also use the mouse to drag the node to a new location. Dragging the node to the left or right will change the image intensity selected by that node, while dragging the node up and down changes the transparency of the selected image intensity. A sample video showing advanced usage of the colors is linked from www.mccauslandcenter.sc.edu/mricrogl/tutorials.

4. Colorbars

An intensity colorbar is useful for recognizing the range of intensities displayed. Selecting Color/Colorbar brings up a window that allows you to change the appearance of the colorbar. Note that when viewing overlays, you will see one colorbar for each overlay image (e.g. see Figure 7.1).

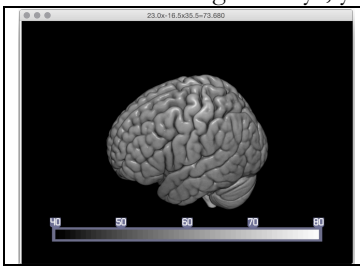


Figure 4.1 An example of the mni152_2009 image with a colorbar visible on the bottom. Clicking the colorbar changes its position on the screen.

5. Clipping

As mentioned earlier, MRIcroGL can either show 3D renderings or 2D slices of an image. However, the clipping tool allows the best of both worlds showing a 2D slice cut through a rendered image. Further, while MRIcroGL's 2D slices are always orthogonal planes (e.g. cut perpendicular to the image, yielding sagittal, coronal or axial views), the clipping tool allows oblique cutting angles.

To see the clipping tool, open an image in the rendering mode (if you are in the 2D slices mode select the Display/Render menu item). Adjust the 'Depth' slider to set the depth of the clipping plane. Additional sliders allow you to specify the azimuth and elevation of the clipping plane.

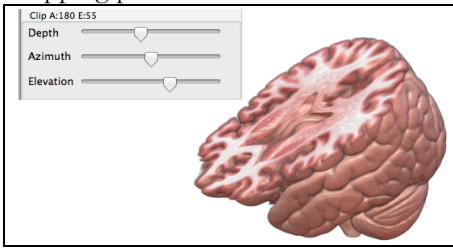


Figure 5.1 An example of the mni152_2009 image with a clipping plane applied. To see this view, drop the 'clip.gls' script onto MRIcroGL.

6. Cutouts

The clipping option described in the previous section allows us to cleave off a 2D slice from the image. The cutout option removes a 3D box from our image, allowing us to see inside. To see the cutout tool, open an image in the rendering mode (if you are in the 2D slices mode, toggle the View/2D menu item). There are six trackbars that adjust the position and size of the cutout. The top two adjust the left-right position, the middle two adjust the anterior-posterior location, and the bottom two adjust the superior-inferior dimension. A good place to start is to press the 'nearest sector' button, which removes the chunk of the image facing you. This is useful, as now you can see any changes made as you adjust the trackbars.

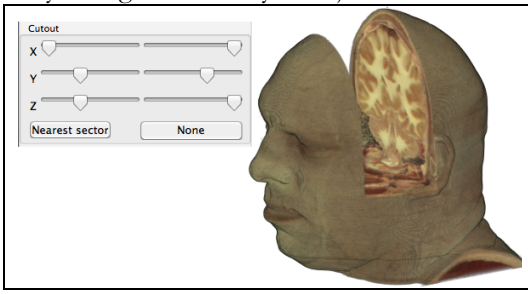


Figure 6.1 An example of a cutout applied to the visible human image (from cutoutdemo2.gls).

7. Overlays

The overlay function allows you to superimpose images on top of your background image. A common usage is to load statistical maps that show brain activity on top of anatomical scans. To add an overlay, first open the overlay window, by choosing Overlays/AddOverlay. As shown in Figure 7.1, you can load multiple overlays and set a unique colorscheme and intensity range for each. You can also use the Overlay/Transparency menus to control how opaque the overlays appear relative to each other and to the background.

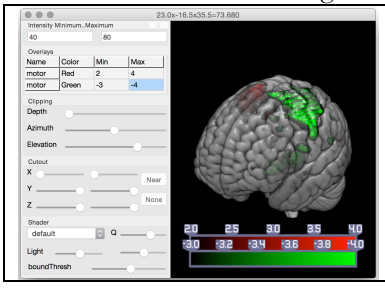


Figure 7.1 An example of the mni152_2009 image with the 'motor' statistical map loaded twice as overlays. In this example, we are showing voxels with a brightness of more than 2 as Red and less than -3 as Green. This is from the 'overlay.gls' script.

The overlay menu has several menu commands that allow you to adjust how multiple overlays interact with each other. You can interact with the Overlays spreadsheet in the Tool Panel to select the color scheme and intensity. There are a couple of non-obvious commands: clicking on the name of an overlay changes its order, so it can be either 'above' or 'below' another overlay. Shift or right-clicking on the name makes the overlay invisible (and the name will appear in red) – shift or right click again to show the overlay.

8. Mosaics

The mosaic function allows you to show multiple 2D slices at once. The easiest way to create a mosaic is by using the mosaic designer form (though you can also use scripting). To see the mosaic designer panel select the Desiplay/Mosaic menu. You can then set the number of rows and columns, as well as the horizontal and vertical overlap. As you adjust these controls, you will see the selected image as well as the corresponding script. You can edit the script and then press 'Run script' for more control. For example, you could arbitrarily switch between axial, sagittal and coronal orientation between rows.

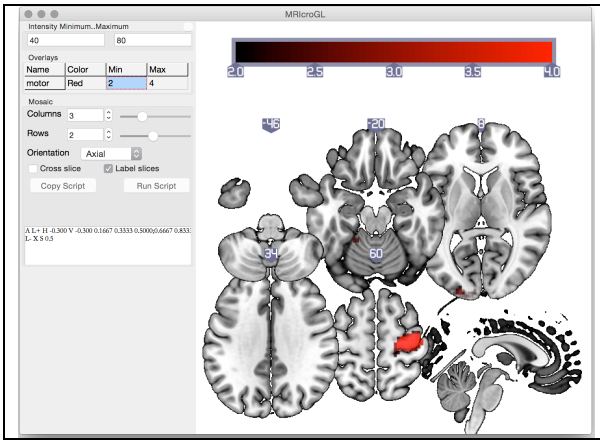


Figure 8.1 A mosaic created from the mni2009_152 image. Note that this mosaic shows both axial and sagittal slices, and that the rows and columns have been set to overlap a little. This was created with the following script "A L+ H -0.300 V -0.300 0.1667 0.3333 0.5000;0.6667 0.8333 L- X S 0.5", as described in the scripting section.

9. GLSL Shaders

MRIcroGL uses files called shaders to generate renderings. Whenever you are in the rendering mode (Display/Render menu item) you will see a panel named "Shader" in the tool panel. You can use the drop-down menu in this panel to select between different shaders, and the sliders on this panel to adjust each shader. Each shader is simply a text file in a folder called 'shaders' (for OSX users, you will have to select the application and choose 'show package contents' to see this folder). Power users can edit the shader text files to create unique effects. The shader minimal.txt shows how a shader can be created with less than 50 lines of code. Here are some useful hints:

- MRIcroGL chooses the initial default shader based on alphabetically sorted filenames. So if you prefer the 'simple.txt' shader to 'default.txt', you could simply rename it 'basic.txt'. This might be a good idea if you have a slow computer, as the 'simple' shader renders faster than the 'default' one.
- Most shader text files start with a 'pref' section that provides the user with adjustable values. You can easily edit these to make the default shader settings more to your liking. For example, the default.txt shader has a preference "specular|float|0.0|0.3|1" – which means that the user will see a slider that allows them to adjust the amount of glistening from 0 to 100%, with a starting point of 30% (the first number is the minimum, the second is the default, the third is the maximum). If you prefer more lighting hints, you could edit the middle number, e.g. if you prefer a matter image you would set this to "specular|float|0.0|0.0|1"

- You can copy shader files and give each a different file name, this would allow you to set custom preferences.
- You can adjust MRIcroGL's ini file to set default shader settings (open the 'About MRIcroGL' menu item and press "Abort" to open this file): RayCastQuality1to10 allows you to choose the quality of raycasting from 1 (fast but poor quality) to 10 (slow but excellent quality). Changing RayCastViewCenteredLight=0 will have the lighting specified relative to the object, not the viewer.

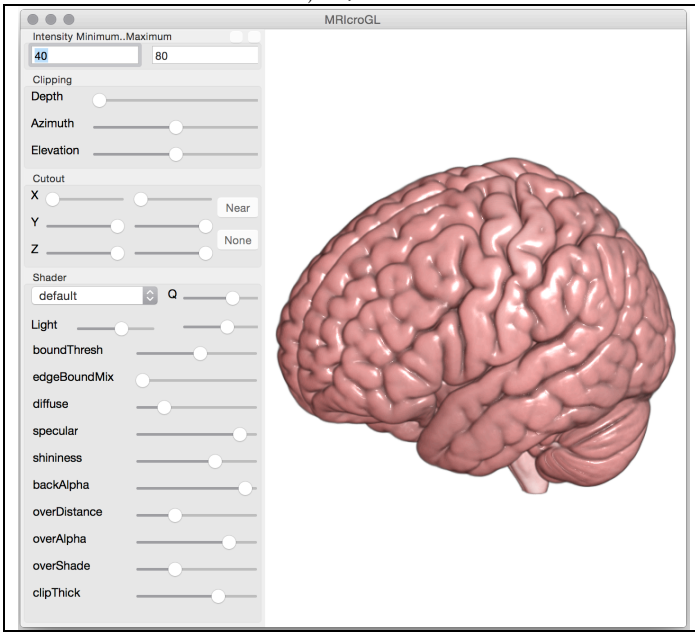
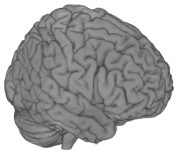
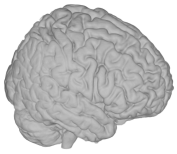
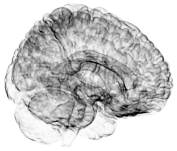


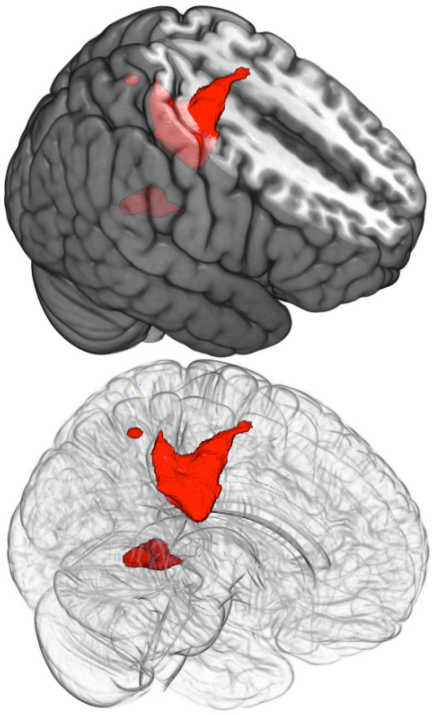
Figure 10.1 GLSL shaders are compact programs that can harness the capabilities of your graphics card. To select and tune a shader, choose Render/Shader to view the shader form. The drop down list at the top of the form allows you to select from the shader plug-ins (in this case the 'phong' shader which provides illumination cues). Some shaders will provide additional user adjustable settings, for example this shader has the 'specular' trackbar that allows the user to interactively adjust the glint of the final image. This image is from the shader.gls demonstration scripts.

Individual shaders work by combining different properties of the image. The table below shows a few popular properties.

Property	Example	Notes
Volume		Brightness depends on image intensity.
Ambient Light		Constant brightness, independent of all other factors.
Diffuse Light		Illumination based on surface orientation with respect to light. Surfaces pointing toward light are brighter than those facing away. (Lambertian reflection).
Specular Light		Illumination based on reflection between light source and viewpoint. Emulates glossy, reflective material. Includes additional adjustment 'shininess': mirrors have small intense specular highlights, while duller surfaces have larger highlights
Edge Shading		Edges parallel to viewing direction are made darker. This emphasizes edges. Only depends on viewpoint (independent of light position).
Boundary Opacity		Regions where image intensity is varying are made opaque, whereas regions with consistent image intensity appear transparent. A simple way to create a 'glass brain' (see also edge enhancement).

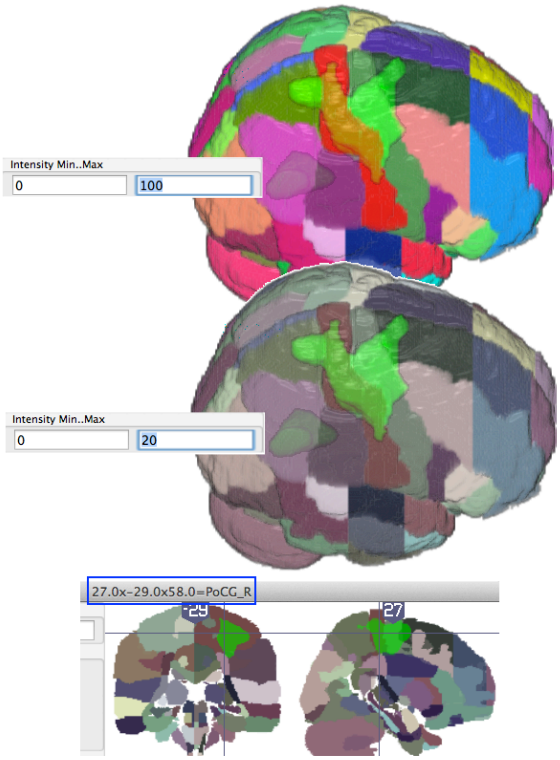
11. Overlay Shader

Recent releases of MRIcroGL include a new shaders named 'overlay', 'overlay_glass' and 'overlay_toon' that are specifically designed for displaying functional or connectivity overlays on top of anatomical scans These overlays allows you to independently set the transparency of the background and overlay images. Note that these shaders are much slower than the conventional shaders, and require twice the memory on the graphics card (and typically render half as fast). The 'Default' shader is also an overlay shader (and therefore it is a lot slower than the 'Simple' shader). See www.mccauslandcenter.sc.edu/mricrogl/about for more details. A YouTube video tutorial is also available from http://www.mccauslandcenter.sc.edu/mricrogl/tutorials



12. Atlas Templates

MRIcroGL includes several template images that show the brain parceled into discrete regions (see section 15 for details on the provided templates). By default these images appear in bright colors (as shown in the top right). This can make it hard to discern overlay images. To adjust the saturation of the template, simply adjust the image intensity values. By default these have a range of 100 (0..100), where the lower images on the left show the image desaturated to 20% (0..20). If you set the image intensity range to zero, the template will appear as a grayscale image. Also note that clicking on any location in the 2D slices view will reveal the region name (in the lower left this is "PoCG_R" for 'posterior central gyrus of the right hemisphere'). A YouTube video illustrating these effects is linked from http://www.mccauslandcenter.sc.edu/mricrogl/tutorials



13. New Features

Version 10/2011 : Thanks to Martins Upitis, David V. Smith & McKell Carter for suggestions/code

- When viewing multiple overlays, click on the overlay's filename in the overlays window to change stacking order of overlays (Note: this feature has no influence if you have selected Transparency/OnOtherOverlays/Additive)
- If the file /script/startup.gls exists, this script will be launched when the program starts.
- To disable this feature, edit the mricrogl.ini file and change "StartupScript=1" to read "StartupScript=0"
- If you are having problems running MRIcroGL, launch the program with the shift key down (OSX and Windows) or with the right mouse button depressed (Linux). You can then restore the default settings. You will also be given the option of disabling the volume rendering (if your graphics card can not support that feature) and enabling power-of-two upsampling (e.g. if you load an image with 192 voxels in one dimension, it will be loaded as 256 voxels, this can dramatically improve performance on older {~2006] graphics cards, but will be slightly deleterious for newer graphics cards)

Version 7/2014 :

- Ability to display template images emulating the 'random rainbow' color scheme of FSLView. These images can include text labels so that the region name is displayed when the user clicks on that location of a 2D slice. Adjusting the 'brightness' of these templates influences the saturation (e.g. a brightness range of 0..50 will have 50% of the full saturation).
- Ability to open non-NIfTI formats. Freesurfer images (MGH/MGZ format), NRRD images (NRRD/NHDR format), AFNI (HEAD format), VTK/ITK MetaIO format (MHA/MHD). The NRRD and MetaIO formats specify a large number of formats – this software should read most images with at least 3 dimensions stored either uncompressed or with GZ compression.
- Overlay shader (see section 11)
- Template images (see section 12)
- Overlay window function added: File/AddOverlay(removeSmallClusters), also scripting function overlayloadcluster (see section 15)

Version 11/2015 :

- Ability to save bitmaps to higher resolution than current display (select the 'Preferences' menu command to select scaling).
- Overlay shaders improved.
- Ability to 'Yoke' different instances of MRIcroGL to show the same slice on different brains. Use the 'Yoke' command in the 'Display' menu. This requires running multiple instances of MRIcroGL simultaneously (choose 'New window' from the 'File' menu).
- OSX will now show retina quality text for controls, 64-bit OSX version. Provides 3D acceleration for modern VirtualBox installations.
- 'Default' shader is now an overlay shader (previous default has been renamed 'simple').
- 'Occlusion' shader.

Recent Versions

- See https://github.com/neurolabusc/MRIcroGL for recent changes

14. Sample data

The software is distributed with sample images

- The Visible Human Project Photographed cryo-sections of a male ('visiblehuman').
- T1-weighted_MRI scan of human brain ('chris_t1' Chris Rorden).
- T2-weighted MRI scan of human brain ('chris_t2' Chris Rorden).
- Magnetic resonance angiography Time-of-flight of human head ('chris_MRA' Chris Rorden).
- Functional magnetic resonance imaging statistical maps showing regions active left versus right hand movements ('motor').
- X-ray computed tomography scan of human cadaver head ('ct' University of North Carolina Volume Rendering Test Data Set).
- Average T1 scan from 152 individuals from the Montreal Neurological Institute www.bic.mni.mcgill.ca/ServicesAtlases/ICBM152NLin2009 Fonov et al. (2009) NeuroImage 47: S102. Saved as mni152_2009bet (0.5mm isotropic). If you are working with a large number of overlays or using a computer with a weak graphics card you may want to reslice this to a lower resolution (e.g. use nii_reslice1mm Matlab script).
- AAL is the Automated Anatomical Labeling atlas from Tzourio-Mazoyer et al. (2002) www.ncbi.nlm.nih.gov/pubmed/11771995
- Brodmann is Krish Singh's speculative Brodmann regions, described at www.mccauslandcenter.sc.edu/mricro/mricro/lesion.html
- HarvardOxford-cort-maxprob is the Harvard-Oxford atlas described at http://fsl.fmrib.ox.ac.uk/fsl/fslwiki/Atlases
- The INIA19 images are from the INIA19 Primate Brain Atlas, www.nitrc.org/projects/inia19/ (see also PMID: 23230398)
- The natbrainlab image is a discrete version of the images from www.natbrainlab.com.

15. Scripting

New users can choose MRIcroGL's View/Script menu item to show the scripting window, the choose a demo script from the scripting window's "File" menu and finally choose the Script/Run menu item to run a script. This is a nice way to observe features. Advanced users can develop their own macros to automate repetitive tasks or create nice demos.

Pascal is used as the scripting language. To write a script, simply select View/Scripting to open the script engine. Type your commands into the text window and select Script/Compile to execute your script. You can also save and open scripts using the File menu. Here is a very simple script. It does one thing, which is to open the file named ch256.

```
Program Simple
Begin
    LOADIMAGE('mni152_2009bet');
End.
```

Note that MRIcroGL is pretty intelligent. It will use files named mni152_2009bet.nii, mni152_2009bet.nii.gz, and mni152_2009bet.hdr in the program folder and the script's folder. If no file is found, a dialog box allows the user to find the file. You could also be more explicit by running "Loadimage('c:\mri\dataset.hdr')", but this is not required.

Below is a list of all of the commands that are specific to this program, but one can also use regular Pascal commands (like for loops, variables and constants). The included sample scripts show some of these properties.

SCRIPTING COMMANDS

ADDNODE(INTENSITY, R,G,B,A: byte) This command adds a new point to the color table. Consider the default color table that has only two nodes - black is node zero and white is node one. Running ADDNODE(192,255,0,0,64) would make images with 75% intensity (192/255) appear bright red, and be 25% opaque (64/255).

AZIMUTH (DEG: integer) This command rotates the rendering. For example, AZIMUTH(-20) rotates the image 20 degrees counter-clockwise

AZIMUTHELEVATION (AZI, ELEV: integer). Sets the viewer location. For example, AZIMUTHELEVATION(90,10) will show a sagittal (right side) view from a slightly inclined viewpoint. Zero degrees azimuth refers to posterior, while 180 degress is directly anterior. Note that these values are absolute, while the AZIMUTH and ELEVATION commands refer to relative changes in viewpoint.

BACKCOLOR (R,G,B: byte) Changes the background color, for example BACKCOLOR(255, 0, 0) will show images on a bright red background

BMPZOOM (Z: byte) Zoom factor applied to bitmaps (Edit/Copy, File/Save and SAVEBMP). For example, BMPZOOM = 2 will save images at twice the screen resolution.

CAMERADISTANCE (Z: single) Sets the viewing distance from the object. Values near zero will appear inside the object (near the center of the object), values near 1 will appear at a reasonably close distance, and larger values will make the object appear quite small.

CHANGENODE(INDEX, INTENSITY, R,G,B,A: byte) This command adjusts a point in the color table. Consider the default color table that has only two nodes - black is node zero and white is node one. Running CHANGENODE(1,255,255,0,0,128) would change the formerly black-to-white color table to be black to bright red. Note that node zero can only have an intensity of zero and the final node must have an intensity of 255.

CLIP (DEPTH: single) Creates a clip plane that hides information close to the viewer. For example, CLIP(0.5) will hide all surfaces on the nearest half of the image.

CLIPFORMVISIBLE (VISIBLE: boolean) Shows or hides the clipping form. For example CLIPFORMVISIBLE(TRUE) shows the form, CLIPFORMVISIBLE(FALSE) hides the form.

COLORBARCOORD (L,T,R,B: single). Sets the position of the colorbar based on the Left, Top, Right and Bottom coordinates. The left and right coordinates are in the range 0..1 from the left to right of the screen, while the top and bottom components range from 0 near the bottom to 1 near the top. If the distance between L-R is greater than T-B then a horizontal colorbar will be shown, else a vertical colorbar is displayed.

COLORBARFORMVISIBLE (VISIBLE: boolean) Shows or hides the window that allows the user to interactively control the size and location of the colorbar. For example COLORBARFORMVISIBLE (TRUE) shows the form, COLORBARFORMVISIBLE (FALSE) hides the form.

COLORBARTEXT (VISIBLE: boolean). If set to true, then colorbars will include text that indicates intensity range. For example COLORBARTEXT(true).

COLORBARVISIBLE (VISIBLE: boolean). Shows a colorbar on the main images.

COLORNAME (Filename: string) Loads the requested colorscheme for the background image. For example, running COLORNAME(CT_KIDNEY) will apply the kidney color scheme.

CONTRASTFORMVISIBLE (VISIBLE: boolean) Shows or hides the contrast and color window. For example CONTRASTFORMVISIBLE (TRUE) shows the form, CONTRASTFORMVISIBLE (FALSE) hides the form.

CONTRASTMINMAX (MIN,MAX: single); Sets the minumum nd maximum value for the color lookup table. For example, consider CONTRASTMINMAX(-200,500). In this case, a voxel with the value of -200 will be displayed with the darkest value in the color table, and voxel with a value of 500 will be shown with the brightest value

CUTOUT (L,A,S,R,P,I: single) Selects a sector to remove from rendering view. For example CUTOUT(0,0,0,0.5,0.5,0.5) will hide the left-anterior-superior hemiquadrant.

CUTOUTFORMVISIBLE (VISIBLE: boolean) Shows or hides the cutout window. For example CUTOUTFORMVISIBLE (TRUE) shows the form, CUTOUTFORMVISIBLE (FALSE) hides the form.

ELEVATION (DEG: integer) changes the render camera up or down. For example, Elevation(20) moves the viewpoint 20-degrees above the object.

EXTRACT(LEVELS,DILATEVOX:integer; ONEOBJECT: boolean); Attempts to remove noise speckles from dark regions (air) around object. Levels=1..5 (larger for larger surviving image), Dilate=0..12 (larger

for larger surround). You can also specify if there is a single object or multiple objects. Default values are 5,2,true.

EXISTS(lFilename: string); Function reports whether a file exists. For example, EXISTS('c:\img.nii') returns TRUE if this file is present and false if the file is absent.

FRAMEVISIBLE (VISIBLE: boolean) Shows or hides the cube that appears around the rendered object. For example FRAMEVISIBLE(false) will hide the frame.

LOADIMAGE (lFilename: string) Opens a NIfTI format image to view. For example, LOADIMAGE('c:\mri\image.nii') will open the file named 'image.nii'.

MAXIMUMINTENSITY (MIP_ON: boolean) Changes the rendering mode between standard (which highlights surfaces of objects) and Maximum Intensity Projection that shows the brightest object, regardless of depth. For example MAXIMUMINTENSITY(false) switches to standard rendering.

MODALMESSAGE (STR: string) Displays a dialog box with a message. The script will wait until the user responds prior to continuing.

MODELESSMESSAGE (STR: string) A text message is shown on the main window. This message requires no response from the user, and the script does not pause for a response. The message is visible until the script sends a MODELMESSAGE('') or until the application is restarted.

MOSAIC(Str: string) Shows a series of 2D slices. For example MOSAIC(V 0.1 H 0.2 A 0.3, 0.6, 0.9; A 0.5 S 0.5 C 0.5) shows two rows of images, each with three columns - the top row shows three axial views, while the bottom shows an axial, coronal and vertical slice. The vertical slices overlap 10% (V 0.1) and the horizontal slices overlap 20% (H 0.2).

- A: subsequent slices in axial orientation
- C: subsequent slices in coronal orientation
- H: Horizontal overlay, e.g. H 0.5 means each slice has 50% overlap. Values can range from –1 to 1.
- L: turns on (L+) or off (L-) text labes for slices
- S: subsequent slices in sagittal orientation
- V: Vertical overlap, from –1 to 1, e.g. V 0 means no vertical overlap
- Z: mirrored sagittal slice
- Numbers are used for each slice, with semicolons denoting new rows, so 0.1 0.2 0.6; 0.8 0.9 is a mosaic with five images, three in the first row and two in the second.

MOSAICFORMVISIBLE (VISIBLE: boolean) Shows or hides the mosaic designer window. For example CLIPFORMVISIBLE(TRUE) shows the form, CLIPFORMVISIBLE(FALSE) hides the form.

ORTHOVIEW (X,Y,Z: single) Shows a 2D projection view of the brain, with an axial slice at the Z coordinate, a coronal slice at the Y value, a sagittal slice at the X coordinate. For example ORTHOVIEW(0.5,0.5,0.5) shows a slice in the middle of the volume.

ORTHOVIEWMM (X,Y,Z: single) Shows a 2D projection view of the brain, with an axial slice at the Z coordinate, a coronal slice at the Y value, a sagittal slice at the X coordinate. For example ORTHOVIEWMM(0, 0, 50) shows a slice 50mm superior to the anterior commissure.

OVERLAYCLOSEALL. This function has no parameters. All open overlays will be closed. The background image (if any) will remain opened.

OVERLAYCOLORFROMZERO(FROMZERO: Boolean). If set to false, then the full color range is used to show the overlay.If set to false, then the color range spans from zero to the most intense color. For example, consider an overlay with a color scheme that goes from black to bright red and a OVERLAYMINMAX(1,1,2). With colorfromzero set to false, a voxel with an intensity of 1 will appear black (minimum in the range 1..2). On the other hand, if colorfromzero is true, then this same voxel will appear dark red (as 1 is half way between 0 and 2). Note that in this case voxels darker than 1 will still be invisible (as they must exceed the 1..2 threshold set by overlayminmax).

OVERLAYCOLORNAME (lOverlay: integer; lFilename: string); Set the colorscheme for the target overlay to a specified name. For example OVERLAYCOLORNAME(1,' CT_Kidneys') will make the first overlay show intermediate brightness regions as red and very bright regions as white.

OVERLAYCOLORNUMBER (lOverlay,lLUTIndex: integer) Sets the color scheme for a overlay. For example, consider that you load a single overlay and want it to make it appear blue, you could call OVERLAYCOLORNUMBER(1,2). The LUTindex refers to the order of the colorscheme in the overlay menu's drop down menu, such that 0=grayscale, 1=red, 2=blue, etc.

OVERLAYFORMVISIBLE (VISIBLE: boolean) Shows or hides the overlay window. For example OVERLAYFORMVISIBLE (TRUE) shows the form, OVERLAYFORMVISIBLE (FALSE) hides the form.

OVERLAYLOAD (lFilename: string): integer; Will add the overlay named filename and return the number of the overlay. Will return zero if unable to find the file or if there are too many overlays already loaded.

OVERLAYLOADCLUSTER (lFilename: string; lThreshold, lClusterMM3: single; lSaveToDisk: boolean): integer; Will add the overlay named filename, only display voxels with intensity greater than threshold with a cluster volume greater than clusterMM and return the number of the overlay.

OVERLAYLOADSMOOTH (SMOOTH: boolean) Determines whether overlays are interpolated using trilinear interpolation (SMOOTH = true) or nearest neighbor (SMOOTH= false). Smoothed images will not appear jagged, but a thresholded map (where values below say Z=2.3 are set to zero) may appear to have dark edges. Note the this command does not influence currently loaded images, rather it sets how future images will be smoothed when overlayload is called.

OVERLAYMASKEDBYBACKGROUND (MASK: BOOLEAN). If true, than a overlay will be transparent on any voxel where the background image is transparent. If false, the overlay will always be shown, regardless of the background image. Sometimes it is nice to mask an overlay, so for example brain activity does not appear outside the brain. On the other hand, consider the glassbrain script, where we want to see overlays that are inside the brain's shell. In this case, we would want to use set this value to false.

OVERLAYMINMAX (lOverlay: integer; lMin,lMax: single) Sets the color range for the overlay. Values outside this range but closer to zero will not be displayed, voxels with more extreme values will appear with the maximum color from the color scheme For example OVERLAYMINMAX(1,2 9) will set the first overlay to have an intensity range of 2..9.

OVERLAYTRANSPARENCYONBACKGROUND (lPct: integer). Controls the opacity of the overlays on the background. For example, setting this to 50 will equally blend the overlays with the background, while 90 means that the overlays will be barely visible. Use –1 for an additive mixture (where the largest red, green and blue components are taken from each image).

OVERLAYTRANSPARENCYONOVERLAY (lPct: integer); Controls the opacity of the overlays on other overlays. For example, setting this to 50 will equally blend the overlays, while 90 means that the top overlay will be barely visible against a lower overlay. Use –1 for an additive mixture (where the largest red, green and blue components are taken from each image).

PERSPECTIVE (USEPERSPECTIVE: boolean) Toggles the usage of perspective on or off. If set on, closer objects will appear larger than more distant objects in the rendering window. If off, an orthographic projection is used (where size is not influenced by distance from viewer).

RESETDEFAULTS. Sets all of the user adjustable settings to their default values. Calling this at the beginning of a script ensures that an image will always look identical, regardless of user-implemented changes.

SAVEBMP (lFilename: string) Saves the currently viewed image as a PNG format compressed bitmap image.

SCRIPTFORMVISIBLE (VISIBLE: boolean) Shows or hides the scripting window. For example CLIPFORMVISIBLE(TRUE) shows the form, CLIPFORMVISIBLE(FALSE) hides the form.

SETCOLORTABLE (TABLENUM: integer) Changes the color scheme used to display an image. For example, SETCOLORTABLE(0) applies the default gray-scale color scheme. To determine the color scheme number, open the Color and Transparency Window and check the pull-down menu. For example, the first item is the default table, so it has a value of zero, while Kidneys is the second item and has a value of 1.

SHADERFORMVISIBLE(VISIBLE: boolean) Shows or hides the GLSL shader control window.

SHADERNAME(lFilename: string). Will load the named GLSL shader and set the shaders default variables. For example, shadername('phong') will load the 'Phong' shader (assuming 'phong.txt' is in your shader folder).

SHADERADJUST(Property: string; Val: single). Some shaders allow the user to interactively adjust settings (using the shader form) – this command allows you to change these settings via a script. For example, shaderadjust('specular',1.0) would set the value specular to 1. Some shader properties are integers or floats (both shown as trackbars), whereas others are Boolean (which are displayed as checkboxes). For Boolean values, shaderadjust('name',0) sets the value 'name' to false, whereas any other value (shaderadjust('name',1)) will set the property to true.

SHARPEN Applies Unsharp-mask to make background image sharper but noiser. Can be applied multiple times for exaggerated effects.

SLICETEXT (VISIBLE: boolean) If true, the 2D slices will be displayed with text indicating which side is left and numbers indicating slice coordinates.

VIEWAXIAL (STD: boolean) creates rendering from an axial viewpoint. VIEWAXIAL(true) shows a bird's eye view (from directly above), while VIEWAXIAL(false) shows an image from directly below.

VIEWCORONAL (STD: boolean) creates rendering from a coronal viewpoint. VIEWCORONAL(true) shows a view from directly in front, while VIEWCORONAL(false) shows an image from directly behind.

VIEWSAGITTAL (STD: boolean) creates rendering from an sagittal viewpoint. VIEWAXIAL(true) shows a profile view with the anterior dimension on the right side, while VIEWSAGITTAL(false) has the anterior dimension on the left side.

WAIT (MSEC: integer) The program pauses for the specified duration. For example WAIT(1000) delays the script for one second.

- mricrogl
- Discussion
- View Source
- History

mricrogl:MainPage

From NITRC Wiki

MRIcroGL user interface.

Contents

Introduction

MRIcroGL is medical image viewer that allows you to load overlays (e.g. statistical maps), draw regions of interest (e.g. create lesion maps). Details can be found on the MRIcroGL home page. MRIcroGL runs on Windows, Linux and OSX. This wiki provides some additional notes on using MRIcroGL.

Requirements

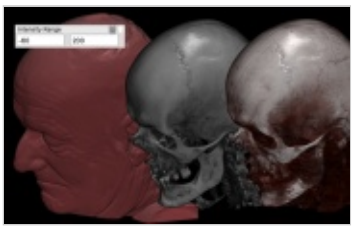
MRIcroGL runs on Linux, Windows, and MacOS. It requires OpenGL 2.1 (released 2006) or later. If you have problems see the trouble shooting web page.

Installation

To install MRIcroGL download it from NITRC. If you have problems running MRIcroGL, you may want to visit the Installation Notes and troubleshooting page. One basic hint: if you are having trouble running MRIcroGL, try launching the program with the **shift key** down (or the **control key** if you are using Unix). This will allow you to choose between different settings to match your computers video card.

MRIcroGL is open source, and if you want you can download download and compile it yourself.

Viewing images



Adjusting the image contrast and brightness can change the appearance.

You should be able to view most medical images by dragging and dropping the image onto MRIcroGL, or by using the File/Open command. The software should be able to read most medical images, including those in the following formats: NIfTI (.nii, .nii.gz, .hdr/.img), Bio-Rad Pic (.pic), NRRD, Philips (.par/.rec), ITK MetaImage (.mhd, .mha), AFNI (.head/.brik), Freesurfer (.mgh, .mgz), basic DICOM (extensions vary). Since DICOM images are usually saved as 2D slices, it is usually a good idea to convert them to NIfTI format first, as described in the 'Converting Images' section.

You may want to adjust the image intensity range values and the color scheme to emphasize different tissues. For example, the image shows a high-quality CT scan from Philips, where we can adjust the contrast to either show the scalp, the bones or the soft tissue.

Basic tasks are shown in the basic usage and color settings videos.

Loading Overlays

After you open an image, you can add overlay images on top of the background image. To do this you will want to choose File/Overlays which displays an overlay control window and then select File/OpenOverlays.

For demonstrations, see the basic overlays video. For advanced usage see the overlay shaders and the overlay glass videos.

Drawing Regions of Interest

After you open an image, you can draw regions of interest. This is useful for mapping lesions (for analysis with vlsm or NiiStat), or for analyzing your fMRI data (for analysis with marsbar).

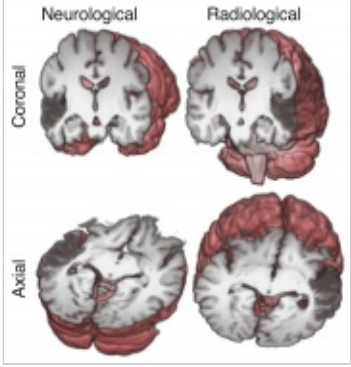
The basic details for drawing a region of interest are described on the MRIcroGL drawing page. In brief you will want to do the following

1. Load an image with MRIcro

2. Choose a 2D view of your image (use the Display menu to select sagittal, coronal, axial or multislice views, but **not** the render view).
3. Choose Draw/DrawColor/Red
4. You can now drag the mouse on the image to create a filled object
5. Drag the mouse with the control key down to create an unfilled object
6. Alt+click to fill a contiguous region
7. Drag with the shift key down to erase a filled region
8. Drag with the shift and control keys down to erase an unfilled region
9. Alt+Shift+click to erase a contiguous region
10. Remember, you can use the arrow keys and the home/end keys to change which slice you are viewing. This allows you to draw one slice after the other. For OSX Mac computers, the command+left_arrow and command+right_arrow keys simulate home and end presses.

You may want to watch the drawing video.

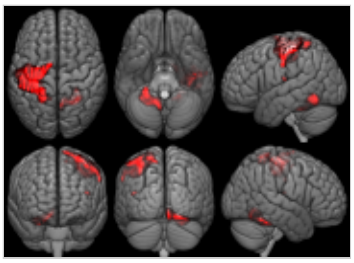
Neurological versus radiological convention



Neurolgical versus radiological convention as seen from the perspective of 3D rendering.

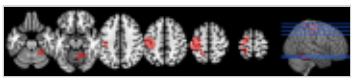
Recent versions of MRIcroGL allow you to choose whether the 2D axial and coronal slices are shown in neurological or radiological convention. You can select your preference with the Display/Radiological(FlipLR) menu item. When this is unchecked, a left hemisphere lesion is shown on the left side of coronal and axial slices (neurological convention). When this item is checked, a left hemisphere lesion will appear on the right side of the screen. Note this selection does not mirror reverse the 3D renderings. To understand this, take a look at the figure. Neurological convention shows coronal/axial slices with the camera posterior/superior to the brain. Radiological view shows coronal/axial slices with the camera anterior/inferior to the brain. Therefore, this left hemsiphere lesion appears on the left side of the image in the neurological views, but on the right side in the radiological views. Another way to think about this is that neurological and radiological convention are reversed with respect to the depth dimension (e.g. what is closer or further from the viewer). To help you remember the convention the "L" and "R" buttons in the 2D slices panel will be either in the order "LR" (neurological) or "RL" (radiological). Due to the brains' symmetry, left-right errors are easy to make, so be careful with this feature. This explains why your choice of radiological versus neurological convention does not influence the 3D renderings. With the 3D renderings we explicitly set the camera position with respect to the brain, and we can see the depth dimension: near occlude more distant features.

Mosaic Views



Mosaic renderings.

Choosing "Mosaic" from the "Display" menu shows the mosaic view, where multiple slices of the same image can be seen simultaneously. A panel named "Mosaic" appears in the tool panel, and this allows you to interactively adjust the mosaic view with sliders, check boxes, pulldown options and text editing. You can also create advanced mosaics using scripting. Unlike the other views, the appearance of mosaics may have substantially different aspect ratios when you view them onscreen versus when you use File/Save to save an image. The mosaic view is designed for generating publication quality images, and the resolution is driven by the volume you are viewing rather than your screen (to reduce interpolation artifacts). While you can add text labels in this mode, for publications you probably want to generate images without labels and add labels later using a vector-art tool that can create sharper text than is possible with bitmaps.



Mosaic slices.

Scripting

MRIcroGL can use scripts to demonstrate features and automate laborious tasks. Scripts can be run from the graphical interface (the View/Scripting menu item) or can be invoked from the command line (allowing other programs to control MRIcroGL). See Github for a full listing of functions. The scripts below show the in-built Pascal scripts, but if you prefer you can use the popular Python language.

Glass Brain

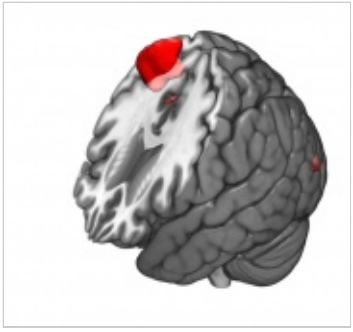
A script showing glass brain projection.

This script uses the provided example dataset from an individual conducting a tapping task whenever they see a visual stimulus. We expect to see the contralesional motor cortex, supplementary motor area and visual cortices active during the task. We overlay this on a standard brain. This script demonstrates the 'overlay_glass' shader released in 2014 that allows you to independently control the transparency of the background and overlay images. For a tutorial on using this see the demonstration YouTube video.

To reproduce this script choose View/Scripting and then insert the following code. When you are done choose Script/Run.

```
BEGIN
        LOADIMAGE('mni152_2009bet');
        OVERLAYLOADSMOOTH(true);
        OVERLAYLOAD('motor');
        OVERLAYMINMAX(1, 2.5, 2.5);
        BACKCOLOR(255, 255,255);
        SHADERNAME('overlay_glass');
        SHADERADJUST('edgeThresh', 0.6);
        SHADERADJUST('edgeBoundMix', 0.72);
END.
```

Clip Plane

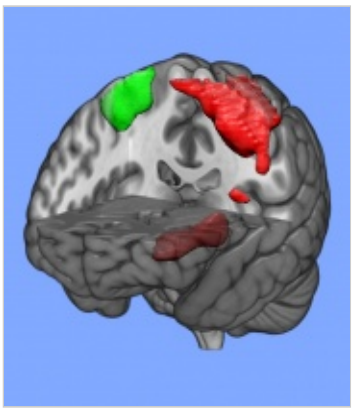


A script showing a clip plane.

This script demonstrates the 'overlay' shader released in 2014 that allows you to independently control the transparency of the background and overlay images. This shader allows you to decide whether the clipping is only applied to the background (as shown) or to both the background and overlay images. You can also watch the overlay shader demonstration video on YouTube for hints on how to use this shader.

```
BEGIN
        LOADIMAGE('mni152_2009bet');
        OVERLAYLOADSMOOTH(true);
        OVERLAYLOAD('motor');
        OVERLAYMINMAX(1, 2.6, 2.6);
        BACKCOLOR(255, 255,255);
        SHADERNAME('overlay');
        CLIPAZIMUTHELEVATION(0.4, 0, 120);
END.
```

Cutout

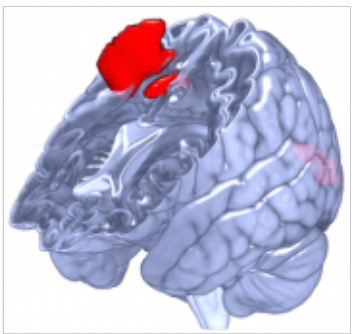


A script showing a cutout.

This script demonstrates the ability to generate cutouts. Note that I load the statistical map 'motor' twice – once to show the regions more active when the left versus right hand was active and once to show the regions that were more active for movements of the right versus left hand (this was a [1 -1] t-test).

```
BEGIN
        LOADIMAGE('mni152_2009bet');
        BACKCOLOR(128, 169, 255);
        OVERLAYLOADSMOOTH(true);
        OVERLAYLOAD('motor');
        OVERLAYMINMAX(1, -4, -4);
        OVERLAYLOAD('motor');
        OVERLAYMINMAX(2, 4, 4);
        CUTOUT(0.0, 0.45, 0.5, 0.75, 1.0, 1.0);
        SHADERNAME('overlay');
END.
```

Shell

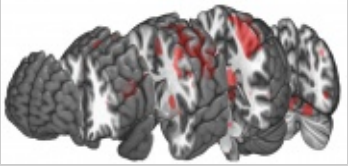


A script showing a shell effect.

This script emphasizes the tissue surfaces (regions with strong gradient magnitude where bright voxels are near darker voxels). This hides regions where the image brightness is not changing, resulting in the cortex and ventricles appearing as thin shells with a hollow interior.

```
BEGIN
        LOADIMAGE('mni152_2009bet');
        OVERLAYLOADSMOOTH(true);
        OVERLAYLOAD('motor');
        BACKCOLOR(255, 255, 255);
        OVERLAYMINMAX(1, 2, 2);
        SHADERNAME('overlay_shell');
        SHADERADJUST('colorTemp', 0.0);
        CLIPAZIMUTHELEVATION(0.35, 0, 140);
END.
```

Mosaic

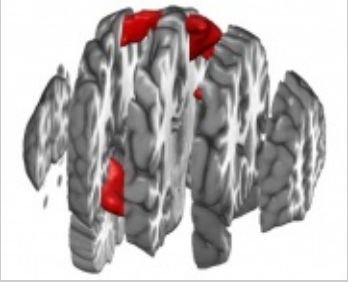


A script showing merged slabs.

This is a way to create a 3D mosaic with any shader. I prefer the 'exploded brain' method as it is simpler to use interactively and you are not restricted to orthogonal slices. See the exploded brain script for common notes. You can show either the left or right hemisphere by setting AZIMUTHELEVATION(130, 15) or AZIMUTHELEVATION(230, 15).

```
CONST
        kSegments = 5;
VAR
        i: integer;
        start, thick: single;
BEGIN
        thick := 1/kSegments;
        RESETDEFAULTS;
        COLORBARVISIBLE(false);
        BACKCOLOR(255, 255, 255);
        LOADIMAGE('mni152_2009bet');
        OVERLAYLOAD('motor');
        OVERLAYMINMAX(1, -2, -2);
        AZIMUTHELEVATION(130, 15);
        FOR i := 1 TO kSegments DO BEGIN
                start := (i-1)*thick;
                CLIPAZIMUTHELEVATION(start, 0, 180);
                CUTOUT(0.0, 0.0, 0.0, 1.0, 1.0-start-thick, 1.0);
                SAVEBMP('sector'+inttostr(i));
        END;
END.
```

Exploded Brain



A script showing the exploded brain effect.

The script generates a series of bitmaps that you can weld together to create a nice image. Unlike the '3D Mosaic', this requires the 'Overlay' shader which has a slider named 'clipThick' that allows us to adjust the thickness of the clip plane slab. You can also press the 'overClip' check-box to select whether the overlay sticks out of the slice (or for a script use 'SHADERADJUST('overClip', 1)' and 'SHADERADJUST('overClip', 0)'). The critical value is 'kSegments' – here I generate 12 slabs, you can adjust this for thinner or thicker slices. This script will generate a series of bitmaps that you stack together using your favorite image editor (Photoshop, Acorn, Pixelmator, etc). The trick is to feather the edges between each image (these programs usually include a 'magic wand' that allows you to make regions transparent).

```
CONST
        kSegments = 12;
VAR
        i: integer;
        start, thick: single;
BEGIN
        thick := 1/kSegments;
        RESETDEFAULTS;
        COLORBARVISIBLE(false);
        BACKCOLOR(255, 255, 255);
        LOADIMAGE('mni152_2009bet');
        OVERLAYLOAD('motor');
        OVERLAYMINMAX(1, -4, -4);
        SHADERNAME('overlay');
        SHADERADJUST('specular', 0.3);
        SHADERADJUST('overAlpha', 0.6);
        SHADERADJUST('clipThick', thick * 0.75);
        AZIMUTHELEVATION(255, 25);
        FOR i := 1 TO kSegments DO BEGIN
                CLIPAZIMUTHELEVATION(0.001+((i-1)*thick), 0, 180);
                SAVEBMP('sector'+inttostr(i));
        END;
END.
```

Cluster Thresholds

A script showing a shell effect.

The statistical maps generated by tools like SPM and FSL can either be unthresholded (green top left image) or thresholded (green bottom left). Since we typically reslice these low resolution images to map onto higher resolution images it is generally a good idea to use the unthresholded image: the thresholded image is artificially surrounded by bogus values so we can not interpolate the boundaries easily. The Overlays/Option/SmoothWhenLoading menu allows you to choose your interpolation method. If this option is unchecked, nearest neighbor interpolation is used and you image will appear jagged (left glass brain) – this is most appropriate if your data was previously thresholded. However, MRIcroGL has commands to remove these small clusters (right image). If the SmoothWhenLoading item is checked, the image will be smoothed with a trilinear filter – this looks very nice (middle glass brain) and is typically appropriate if your raw data was not previously thresholded. However, closer inspection of these reveals that a few tiny clusters appear. If your original analysis used a cluster threshold, you will want to use the overlay windows' File/AddOverlay(RemoveSmallClusters) option. This will remove any clusters smaller than the specified size. You can also achieve this by calling the 'overlayloadcluster' function from a script. You will supply the overlay name, the brightness threshold, and the minimum cluster size (in mm^3), and select whether you also want this thresholded overlay saved to disk (true or false). Note that the cluster size is specified in volume not voxels: if you estimated a random field theory familywise error corrected threshold for clusters of at least 32 voxels with data that was 3mm isotropic (3x3x3 = 27mm^3 per voxel) then you would specify 864mm^3. In the example below we preserve voxels that exceed a brightness of 2.0, are part of clusters of at least 1cc (1000mm^3) and choose not to save the resliced image to disk.

```
BEGIN
        LOADIMAGE('mni152_2009bet');
        OVERLAYLOADCLUSTER('motor', 2, 1000, false);
        SHADERNAME('overlay_glass');
END.
```

Scripting from the command line and other programs

You can also call scripts from the command line or other programs, including those you create in Python and Matlab. MRIcroGL's scripting is similar to that of Surf Ice, so you may want to look at the Surf Ice wiki. Here is a simple example:

```
/Users/rorden/Documents/osx/MRIcroGL.app/Contents/MacOS/MRIcroGL –S "begin LOADIMAGE('ct.nii.gz'); end."
```

Supported Formats

MRIcroGL's native format is NIfTI. However, it can also read images stored in other formats. Not all features of these formats may be supported, so use with caution:

- AFNI format (.head/.brik)
- Analyze format (predecessor to NIfTI, .hdr/.img files)
- BioRad PIC format (.pic)
- Blender voxel format (.bvox)
- BrainVoyager (.vmr/.v16)
- DeltaVision (.dv)
- DICOM - simple drag and drop display or conversion to NIfTI (see below).
- ECAT (.v)
- Guys Image Processing Lab (.gipl)
- ITK format (.mha, .mhd)
- MGH format (.mgh, .mgz)
- MRtrix format n.b. currently only saves images saved as axial slices (.mif, .mih)
- NRRD format (.nrrd, .nhdr)
- Some tagged image file format variations (.tif, .tiff, .lsm)
- VTK legacy format (.vtk)

Converting Images

Use the Import menu item to start dcm2nii. A new window appears, and you can set up your preferences by adjusting the values in the dcm2nii window toolbar. You then drag-and-drop the folder containing you DICOM images onto the dcm2nii window (note: if you drag the DICOM images onto the main MRIcroGL window you will view rather than convert the image). For more details see the dcm2nii wiki.

Special Images

MRIcroGL includes a few NIfTI format images to illustrate the software. Here are some additional images that are available for download.

- Philips CT scan (60mb). MRIcroGL comes with a low resolution version of this image (198x231x256 voxels) to support a wider range of graphics cards. The version linked here is at native resolution and is 384x448x539 voxels. This image provided by Philips Medical and distributed with permission. The DICOM images for this scan are also available.
- mni152_2009ns (34mb). MRIcroGL includes copies of of the mni152_2009 atlas. While this atlas was based on non-linear deformations, it retains the overall size and shape of the original MNI/ICBM 152 template. That template is used for normalization by many tools. However, the MNI template is actually larger than the average brain. In contrast, SPM12's segmentation/normalization generates brains that are roughly aligned but are more typically sized. Therefore, the SPM New Segment ('ns') image here is more appropriate if you use SPM's normalization (though the differences are subtle).

New Features

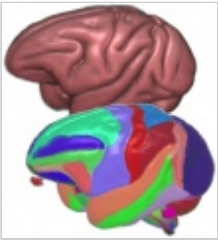
If you have used MRIcroGL before, but have updated to the latest version you may want to briefly find out what new functions have been added.

1. The MRIcroGL download includes a PDF file named "manual.pdf" - section 13 of the manual is named "New Features" and it describes changes and improvements to the software.
2. The latest/upcoming features are described on the beta page.

Gallery

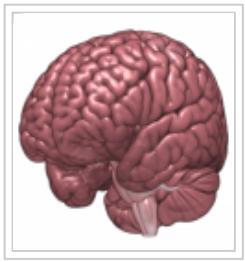
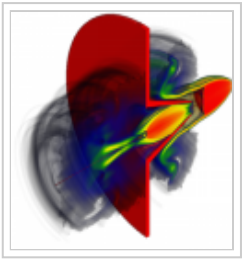
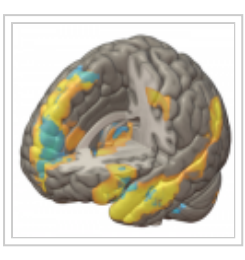
Photographs

CT



primate atlas

mouse

			
Gift	Stag beetle	Daisy pollen	MNI Template
			
Jet	Overlays	[1]	