Edinburgh
Mouse
Atlas
Project

Medical Research Council
MRC
Human Genetics Unit

EMAP Technical Report

A 3D Paint Program for the Mouse Atlas and Gene Expression Database

Author:	**Richard Baldock**
Email:	R.Baldock@hgu.mrc.ac.uk
Date:	**25th March 2004**
Distribution:	**Open distribution is permitted**
Status:	**Final**
Source:	**Report/000010-MAPaint**
Project:	**DG401**
Report number:	EMAP/Report/000010
Keywords:	**3D voxel images, painting, segmentation, data mapping, 3D reconstruction**

Abstract

At the core of the gene-expression database are a time series of grey-level reconstructions of the mouse embryo with every voxel identified as part as an anatomical component. By this means the gene-expression database can be mapped into a common geometric space or “standard mouse” and therefore enable search and comparison of the complex expression patterns. The 3D paint program is one of a number of tools to assist the creation of this atlas and the mapping of the anatomical structures.

The program allows the definition of arbitrarily complex 3D spatial domains by user interaction either directly by “painting” the reference image or by using the image processing and analysis tools as a form of “power-assist”. The program can be used for defining any spatially organised data within the 3D space and in particular will be one of the means by which gene- expression will be entered into the database.

Note this document was written originally in 1996 and has been updated since but the current date reflects when the final version was generated and installed into the CVS repository.

Contents

1 Introduction

The essence of the gene-expression database being developed at the MRC Human Genetics Unit is the ability to map the complex patterns of expression, see figure 1, from many experiments at different times and places onto a standard geometric space in order to allow comparison both with the expression patterns of other genes, the extent ot developing anatomy and indeed any other spatially organised information of interest. Furthermore by mapping onto a standard framework it becomes possible to enter the information into a database and to pose both textual and spatial (including image processing) queries. Much of the existing gene expression data is known only in textual terms by reference to the anatomy but in general the anatomical descriptions will be limited in detail and less specific than actual image data. To enable the inclusion of this data and to allow comparison with the known anatomical structures and tissue properties we need a digital *Atlas* of the developing mouse embryo onto which we can independently map the anatomy and gene-expression. The development of this atlas is part of Mouse Atlas and Gene Expression Database project[1, 7, 2] at the MRC in collaboration with the Department of Anatomy, University of Edinburgh. -

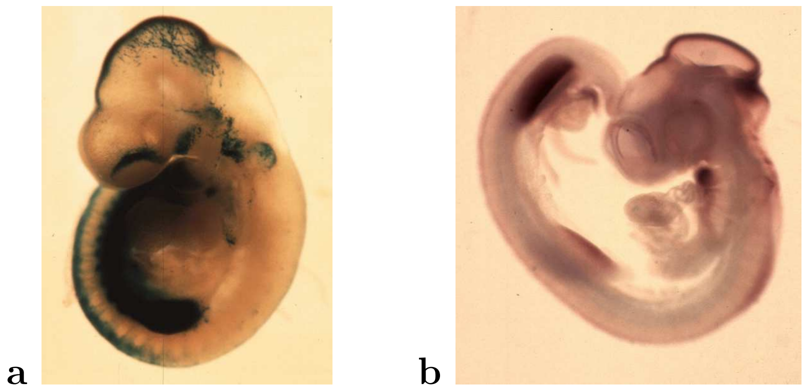

Figure 1: Example expression patterns in a: the 12 day and b: the 9-day mouse embryo

The underlying reference image at each stage must be a full grey-level reconstruction of the histology so that the atlas can be resectioned in any arbitrary plane to compare with the experimental sections. The anatomy forms a rich spatial and temporal structure with each anatomical component identified as a 3D spatial domain at every developmental stage at which it is present so it is possible to go from anatomical name to spatial region. The reverse is also possible because every voxel is part of an anatomical domain so by straightforward search it is possible to go from spatial region to anatomy. For this to be possible we must segment and label the entire 3D volume image for each embryonic stage in the atlas. The 3D paint program has been developed for this purpose. The links between text, anatomy domain and histology as represented by the grey-level voxel reconstruction is shown in figure 2.

The basic assumption is that the final decision for the labelling of any particular voxel must be made by the biologist and therefore the result of any image processing or analysis tools must be subject to review. The structure of the program reflects this assumption and it is assumed that the user will be actively "painting" the reference image, calling on "power-assist" as required.

A second important aspect of the program is that from a pilot studies it is clear that the optimum

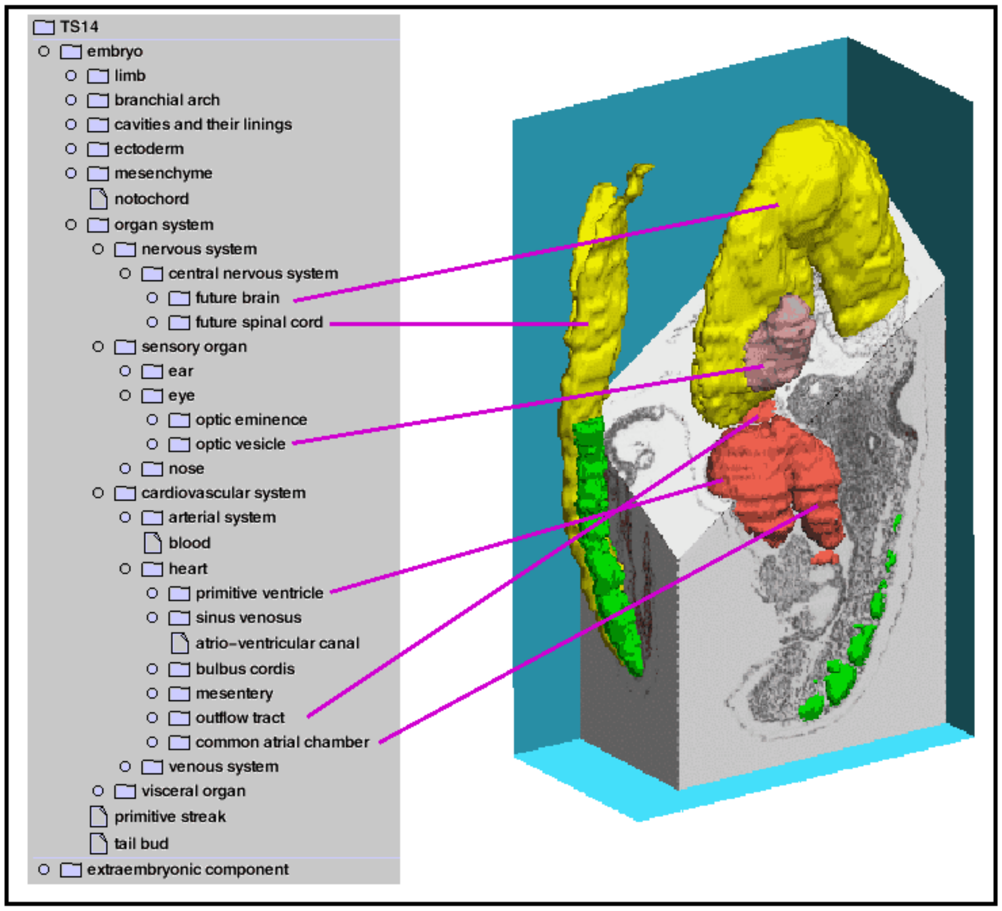

Figure 2: The LHS of the figure shows a screen shot of the anatomy database browser with a selection of the 9-day tissues. The RHS shows a 3D view of the reference image in “cut-block” view to display orthogonal and arbitrary sections through the data, none of which are in the original sectioning planes. The RHS also shows in surface rendered mode selected anatomical components identified by the connecting lines.

viewing plane for identifying and painting structures is often not one of the standard sectioning planes and that it must be possible to paint in any arbitrary planes, preferably more than one at the same time. This is also required when the painting method is used for inputting section gene-expression data. It will not be possible or desirable to constrain the sectioning of the experimental embryos to the section planes of the atlas reconstructions.

The purpose of this document is to describe in some detail the mechanics of the painting program in order to provide a deeper understanding for the user and to get down in writing the precise definitions of the geometric quantities used to define the views. It is not intended as documentation of the code or as a user-guide. A user help/manual is provided as a set of html documents accessible externally or from within the program, the code is documented as manual pages and within the source code files.

2 3D Images

The paint program is based on the image processing library known as *woolz*[8, 5, 6, 4] developed over the years at the MRC Human Genetics Unit. Internally the images are in woolz format which is interval coded and therefore compact for sparse image data and very efficient for the most frequent IP operations required for this program and the gene-expression database, and in particular binary operators - union, intersection, difference; morphological operations - erode, dilate, opening, closing; domain operations - segmentation, labelling, thresholding.

The 3D image structure in woolz is a simple extension of the 2D structure. In 2D an image (one of a number of possible woolz *objects*) is defined over an arbitrary region of a discrete 2D space with coordinates (k, l) where k is the column coordinate and l the line coordinate. For each line in the image there is a list of intervals which give the start and end points of the image along that line. There is a list (possibly empty) of intervals for each line and it is clear that an arbitrarily complex region of the discrete space can be defined in this way. It is assumed that the discretisation in the x and y directions is at fixed regular intervals, constant in both directions but not necessarily equal. The 3D structure is simply a stack of 2D images again with the constraint that the planes are evenly spaced and that within the plane bounds there must be a 2D image for every plane, although the image could be the empty set. The plane coordinate is defined to be p.

The advantage of the woolz encoding is that only grey-level information within the domain of the image is stored rather than for the whole rectangular box defined by the column, row and plane bounds. For the mouse atlas this can reduce the memory requirements in some cases by about a factor of five which is very significant for the later embryo stages where the grey-level image data may exceed 500 MBytes. Even with woolz encoding we may have to consider some form of lossy compression.

Each reconstruction will have its own internal discrete coordinate system with an associated affine transform which will provide the link between internal coordinates and external, biologically relevent coordinates. Furthermore it is likely that each anatomical component will have its own coordinate frame defined in order to establish the localised meaning of terms such as *anterior-posterior etc.*. These coordinate systems are not discussed here. The geometry detailed below is entirely within the internal coordinate system and is concerned with how arbitrary sections are presented to the user, how fixed point constraints are imposed on a view, the definitions of the viewing parameters and miscellaneous geometric calculations for example calculating the lines of intersection of views.

3 Geometry

In this section we discuss the basic geometric transformations used within the program and provide the definitions of viewing parameters.

3.1 Coordinate Transformation

There are many ways to define an arbitrary rotation, scaling and translation of one coordinate frame into another. For the purposes of the paint program we define a set of parameters for viewing arbitrary planes through reconstructions which seem to be the most useful in terms of the way in which the paint program will be used and which should be easy to visualise for the user. The relationship between these transforms and the general affine transform used elsewhere is most easily found be direct comparison of the transform matrix elements.

We define a viewing plane by defining a new set of coordinate axes such that the new z-axis is along the "line-of-sight". This axis is fully determined by defining a single fixed point which is the new coordinate origin. The actual view plane is then defined to be perpendicular to this axis and is determined by a scalar distance parameter along the new axis. In this way the transformation between the original, $\mathbf{r} = (x, y, z)$, and viewing coordinates, $\mathbf{r}' = (x', y', z')$, is determined by a 3D rotation and translation with the viewing plane defined as a plane of constant $z' = d$. These parameters are dipicted in figure 3.

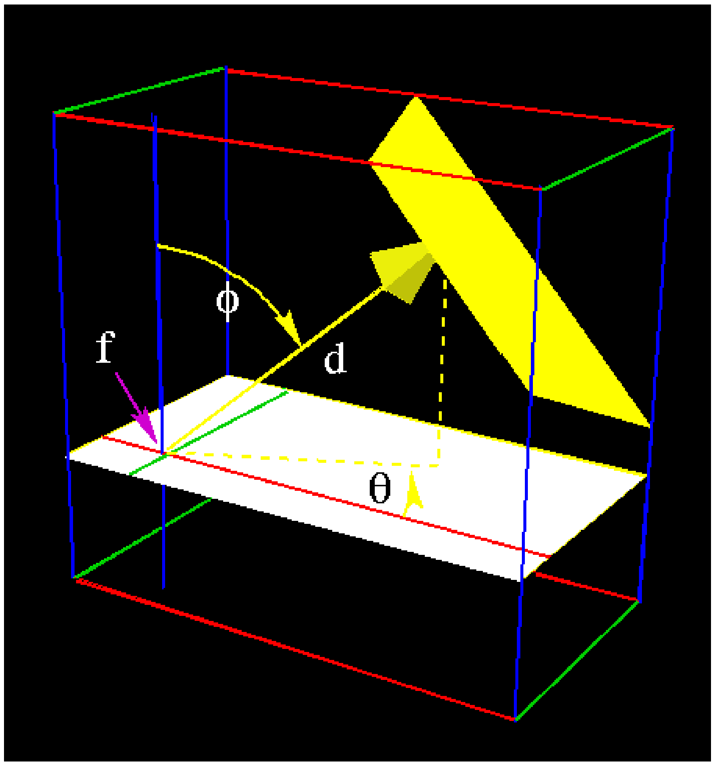

Figure 3: The viewing plane is defined to be perpendicular to the viewing direction given by angles θ and ϕ which are yaw and pitch respectively. The actual plane is distance d from the fixed point $\mathbf{f}$.

A 3D rotation can be defined in terms of Eulerian angles[3, page 107] which are not consistently defined in the literature, but for which we assume the usual British definition[9, page 9], where a rotation about an axis is *clockwise* in the direction of the axis and the second rotation is about the new y-axis:

1. rotate by angle ξ (xsi) about the z-axis
2. rotate by angle η (eta) about the new y-axis,
3. rotate by angle ζ (zeta) about the new z-axis.

This sequence is depicted in figure 4.

The rotation matrix defined by these angle is most easily determined as a product of three rotations:

$$R = R_\zeta R_\eta R_\xi \tag{1}$$

where in matrix notation

$$R_\xi = \begin{pmatrix} cos(\xi) & sin(\xi) & 0 \\ -sin(\xi) & cos(\xi) & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{2}$$

$$R_\eta = \begin{pmatrix} cos(\eta) & 0 & -sin(\eta) \\ 0 & 1 & 0 \\ sin(\eta) & 0 & cos(\eta) \end{pmatrix} \text{ and} \tag{3}$$

$$R_\zeta = \begin{pmatrix} cos(\zeta) & sin(\zeta) & 0 \\ -sin(\zeta) & cos(\zeta) & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{4}$$

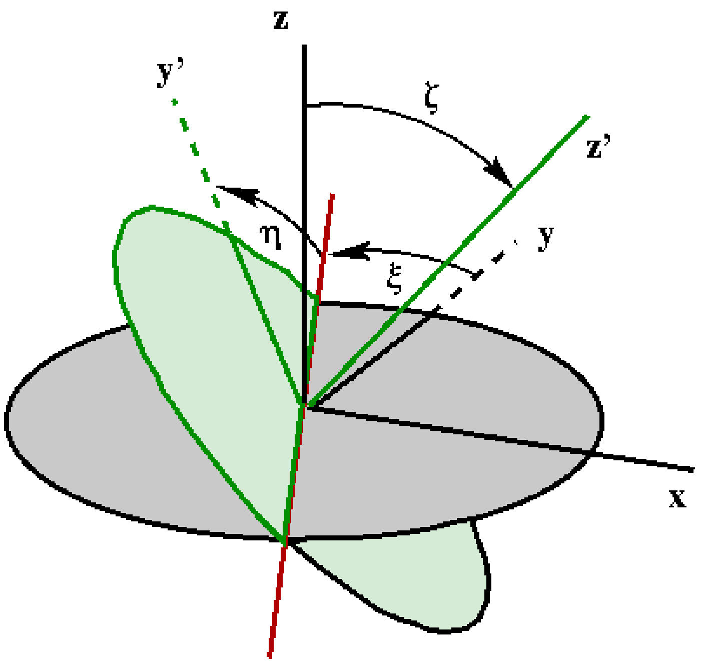

Figure 4: The Eulerian angles. The line of intersection between the original x-y plane and the new x'-y' plane is known as the "line of nodes" and shown extended in red in the figure. The x' axis is below the original plane and not shown.

Multiplying the individual matrices yields

$$R = \begin{pmatrix} cos(\zeta)cos(\eta)cos(\xi) - sin(\zeta)sin(\xi) & cos(\zeta)cos(\eta)sin(\xi) + sin(\zeta)cos(\xi) & -cos(\zeta)sin(\eta) \\ -sin(\zeta)cos(\eta)cos(\xi) - cos(\zeta)sin(\xi) & -sin(\zeta)cos(\eta)sin(\xi) + cos(\zeta)cos(\xi) & sin(\zeta)sin(\eta) \\ sin(\eta)cos(\xi) & sin(\eta)sin(\xi) & cos(\eta) \end{pmatrix}$$

A rigid body transformation can be described as a rotation followed by a translation. In our case we may wish to display the section with magnification and therefore the transformation from screen to object coordinates will also involve scaling. With this in mind we define the transform from object to viewing coordinates as

$$\mathbf{r}' = sR(\mathbf{r} - \mathbf{f}), \tag{5}$$

where s is the scaling and $\mathbf{f}$ is the fixed point in the original coordinates. Most of the calculations we require involve the inverse transform

$$\mathbf{r} = \frac{1}{s}R^{-1}\mathbf{r}' + \mathbf{f}, \tag{6}$$

where $R^{-1} = R^T$ (transposed matrix)[3].

3.2 Viewing Planes

The viewing or normal direction plus the fixed point and distance only define the viewed image to within an arbitrary viewing angle, there is still a choice of the orientation of the image on the screen. There are a number of possibilities here. One is to consider the 3D image as an object in space and display a projection of the viewing plane as seen by an observer in the same 3D space. This has the merit of providing clear feedback of the position of the plane within the whole but is not ideal for painting because for some angles the projection will introduce perspective distortion of the image. There are two solutions adopted by the paint program both of which display the view-plane "flat" on the screen, *i.e.* the view orthogonal to the viewing direction. The difference in the two is the model used to establish the rotation around the viewing direction. In the first versions of the program the plane was displayed as if the viewer was "walking" around the structure on the $x - y$ plane. This proved confusing and therefore a different model has been included which is to ensure that the projection of an arbitrary but predefined vector $\mathbf{u}$ (up) onto the viewing plane will be parallel to the y-axis of the displayed plane (actually "up" on the screen is in the direction $(0, -1)$ in screen coordinates because traditionally raster coordinates start at the top left corner of the screen). Both of these have their advantages and disadvantages and therefore the either model can be selected.

3.2.1 Walking Around the Statue

The initial solution adopted in the 3D paint program is to always display the viewing plane "flat" but to orient the image as if the viewer were "walking around" the object. The actual displayed image is then obtained by rotating the viewed plane about an axis parallel to the line of intersection of the view plane and the "horizontal" which is defined to be a plane of constant z. This is depicted in figure 3 where two viewing angles are defined: the angle to the x-axis of the projection of the viewing direction onto the horizontal, θ, and the angle from the viewing direction to the z-axis, ϕ. These are the usual spherical coordinate angles and are known under many aliases *e.g.* azimuth and declination or pitch and yaw. Only two angles are required to define the viewing direction and the third Eulerian angle is determined by using the rule outlined above. It can be seen for example that a $y - z$ plane is displayed with $y' = y$ and $x' = z$, and the $x - z$ plane is displayed with $x' = x$ and $y' = -z$. It is straightforward to show that the Euler angles defined using θ, ϕ and rotation about the line of intersection with the horizontal are

$$\xi = \theta, \tag{7}$$
$$\eta = \phi \text{ and} \tag{8}$$
$$\zeta = -\theta. \tag{9}$$

Substituting these into equation 3.1 yields

$$R = \begin{pmatrix} cos^2(\theta)cos(\phi) + sin^2(\theta) & cos(\theta)sin(\theta)(cos(\phi) - 1) & cos(\theta)sin(\phi) \\ cos(\theta)sin(\theta)(cos(\phi) - 1) & sin^2(\theta)cos(\phi) + cos^2(\theta) & sin(\theta)sin(\phi) \\ -cos(\theta)sin(\phi) & -sin(\theta)sin(\phi) & cos(\phi) \end{pmatrix} \tag{10}$$

For each new view it is necessary to calculate the corresponding point in the reference image for each point on in the view plane. This is done by defining a set of look-up tables so that the trigonometric functions are only calculated once for each viewing direction. Because the transform angles can be arbitrary floating point numbers it is necessary to calculate the corresponding point in the reference image in real (floating point) coordinates. This means that 6 LUT's are required namely a vector LUT for x' and y'. These LUT's, the maximum values for x', y' and z', the view angles, fixed point and distance are each stored in the `ViewStruct` data-structure held with each view window and are used for updating values as the distance or other parameters are modified.

3.2.2 Up is Up

In this model the projection of a predefined direction "up" will always be displayed as the vertical in the section view. If the viewing direction is parallel to this vector then the angle of rotation around the viewing direction is not defined and an arbitrary choice can be made. A consequence of this is that for small changes in viewing direction around the "up" vector may give rise to arbitrarily large changes in the display orientation. This effect occurs for any rule which determines the screen orientation solely from the static viewing orientation. If a smooth transformation is required then it is necesary to select the orientation not just in terms of the viewing direction but also in terms of the previously viewed section, *i.e. dynamic* information is required. This of course means that the viewing direction does not uniquely define the section orientation.

To establish the viewing transformation we need to calculate the Euler angle ζ so that the projection of the up vector, $\mathbf{u}$, is parallel to the displayed y-axis. To do this we calculate the angle ω between the component of the vector $\mathbf{u}$ perpendicular to the viewing direction and the y-axis in the new coordinate system for the transformation $R(\xi, \eta, \zeta) = R(\theta, \phi, 0)$. The angle ζ is then ω or $\omega - \pi$ depending on the orientation of the y-axis on the screen. This angle can be easily established by rotating the up vector by $R(\theta, \phi, 0)$ and calculating the angle of the projection of this vector to the transformed y-axis. Thus if

$$\mathbf{w} = R\mathbf{u}, \quad \text{then} \tag{11}$$
$$\omega = \tan^{-1}(\frac{w_x}{w_y}). \tag{12}$$

Given the Euler angles the view transformation is calculated in exactly the same way as for the other viewing mode.

3.2.3 Look up Tables for Display to Image Coordinates

To make the calculation more efficient we can minimize the number of floating point operations by pre-calculating the trigonometric terms and storing them as look-up tables (LUT). In this case we are transforming from the 2D display frame back to 3D image coordinates and to avoid round-off error we require 6 LUTs for each of x' and y' to (x, y, z). Note x' and y' are always integer coordinates in the display frame. In our coordinate system we want the transform from (x', y', d) to the original image coordinates therefore using equation 6 we get

$$\begin{aligned} x &= \frac{1}{s}(R^{-1}_{11}x' + R^{-1}_{12}y' + R^{-1}_{13}d) + f_x \ , \\ y &= \frac{1}{s}(R^{-1}_{21}x' + R^{-1}_{22}y' + R^{-1}_{23}d) + f_y \ \text{ and} \\ z &= \frac{1}{s}(R^{-1}_{31}x' + R^{-1}_{32}y' + R^{-1}_{33}d) + f_z \ . \end{aligned} \tag{13}$$

We define LUTs for x' to (x, y, z) which include the constant terms:

$$\begin{aligned} T_{x'x}(x') &= \frac{1}{s}(R^{-1}_{11}x' + R^{-1}_{13}d) + f_x \ , \\ T_{x'y}(x') &= \frac{1}{s}(R^{-1}_{21}x' + R^{-1}_{23}d) + f_y \ , \\ T_{x'z}(x') &= \frac{1}{s}(R^{-1}_{31}x' + R^{-1}_{33}d) + f_z \ . \end{aligned} \tag{14}$$

and similarly for y' to (x, y, z):

$$\begin{aligned} T_{y'x}(x') &= \frac{1}{s}R^{-1}_{12}y' \ , \\ T_{y'y}(x') &= \frac{1}{s}R^{-1}_{22}y' \ , \\ T_{y'z}(x') &= \frac{1}{s}R^{-1}_{32}y' \ . \end{aligned} \tag{15}$$

To determine the new section image the transformations can be calculated by look-up plus one addition:

$$\begin{aligned} x &= T_{x'x}(x') + T_{y'x}(y') \ , \\ y &= T_{x'y}(x') + T_{y'y}(y') \ , \\ z &= T_{x'z}(x') + T_{y'z}(y') \ . \end{aligned} \tag{16}$$

If the viewing direction does not change and only the distance parameter is incremented by δd then the y' LUTs are unchanged and the x' LUTs become:

$$\begin{aligned} T_{x'x}(x', d + \delta d) &= T_{x'x}(x', d) + \frac{1}{s}R^{-1}_{13}\delta d \ , \\ T_{x'y}(x', d + \delta d) &= T_{x'y}(x', d) + \frac{1}{s}R^{-1}_{23}\delta d \ , \\ T_{x'z}(x', d + \delta d) &= T_{x'z}(x', d) + \frac{1}{s}R^{-1}_{33}\delta d \ . \end{aligned} \tag{17}$$

Note $R^{-1} = R^T$. The LUTs are calculated over the maximum range possible for x' and y' which is determined by forward transform (equation 5) of the bounding box of the original image.

3.3 Fixed Point Constraints

Navigation through the volume can be difficult especially since arbitrary sections give rise to unfamiliar views of well-known structures. To aid the navigation the paint program give a “cartoon” 3D feedback display showing the rough position of each view with respect to the bounding box (see the section on 3D display). A control within the view windows that can provide assistance for navigation is the option of setting fixed-point constraints. The basic idea of the transformation used is that it may often be possible to identify one or more points within the image that the user wishes to be definitely visible and thereby reduce the search space to setting the viewing angles. If one point is fixed then there are two degrees of freedom left to set the view and if there are two fixed points then there is only one degree of freedom.

The transformation is defined so that by setting one fixed point, $\mathbf{f}$, the orientation parameters, θ, ϕ, will rotate the view plane about this point. If two points are fixed then θ and ϕ are dependent and can be represented in parametric form using a third angle parameter, ψ, which corresponds to the angle around the line joining the two fixed points.

The two fixed points $\mathbf{f_1}$ and $\mathbf{f_2}$ define vector

$$\mathbf{n_1} = \frac{\mathbf{f_2} - \mathbf{f_1}}{|\mathbf{f_2} - \mathbf{f_1}|}$$

which must remain in the view plane. The values of θ and ϕ which define the plane in which the two fixed points were initially defined, θ_0 and ϕ_0, define an axis perpendicular to $\mathbf{n_1}$

$$\mathbf{n_2} = R_0^{-1}\mathbf{z},$$

where R_0 is the rotation matrix defined by θ_0, ϕ_0 and $\mathbf{z} = (0,0,1)^T$. A third axis, $\mathbf{n_3} = \mathbf{n_1} \wedge \mathbf{n_2}$, is perpendicular to both $\mathbf{n_1}$ and $\mathbf{n_2}$ and can be used to define the normal to a viewing plane which contains the two fixed points:

$$\mathbf{n} = cos(\psi)\mathbf{n_2} + sin(\psi)\mathbf{n_3},$$

where $0 \leq \psi \leq 2\pi$, and $\psi = 0.0$ is the original fixed plane. If we demand that $\mathbf{f}$ is equal to one of the two fixed points and that the viewing direction is $\mathbf{n}$ then both fixed points will be visible. The corresponding viewing angles are given by:

$$cos(\phi) = n_z, \quad (18)$$

$$tan(\theta) = \frac{n_y}{n_x}. \quad (19)$$

Note if $\mathbf{n} = \mathbf{z}$ then θ is ill-defined. In the program paint the third Euler angle ζ is determined by requiring that the vector $\mathbf{n_1}$ maintains a constant angle to the vertical on the screen. This angle is determined by when the fixed line is first set.

In the 3D paint program the fixed point constraints are implemented so that they remain in force whilst only the angles are being modified. Resetting the first fixed point or changing the distance parameter will cancel the second fixed point constraint.

3.4 Lines of Intersection

To make the painting easier on an undistorted image the arbitrary sections are not displayed with any sort of orthographic or perspective projection but as seen from the viewing angle. This can be somewhat confusing and to help navigation the line of intersection of the view that has the current "input-focus" (usually when the cursor is in the view window) with all other views is displayed and will move as the controls are adjusted. To determine the line of intersection is straightforward geometry and requires solving a set of three simultaneous equations to find a point in each view that is common to both.

For two given planes p_1 and p_2 with viewing angles (θ_1, ϕ_1) and (θ_2, ϕ_2) respectively we want to establish the angle, α, and a point, $\mathbf{p} = (p_X, p_Y)$, of the line of intersection of plane 1 in plane 2. With these two quantities we can then draw the line of intersection in plane 2:

$$(y - p_y) = tan(\alpha)(x - p_x).$$

If $\mathbf{n_1}$ is the normal to viewing plane 1 and $\mathbf{n_2}$ is the normal to viewing plane 2 then the line of intersection, which must lie in both planes, is parallel to the vector $\mathbf{n_1} \wedge \mathbf{n_2}$. To establish the angle in plane 2 we rotate this vector to plane 2 coordinates, $\mathbf{l} = R_2(\mathbf{n_1} \wedge \mathbf{n_2})$, and

$$tan(\alpha) = \frac{l_y}{l_x}.$$

To calculate a point of intersection of the two planes we solve for the point of intersection of a line in plane 1 with plane 2. A line in plane 1 which must intersect plane 2 is a normal to the line of intersection. If $\mathbf{f_1}$ and d_1 are the fixed point and distance parameters of plane 1 then the normal to the line of intersection which passes through the origin in plane 1 is

$$\mathbf{r} = \mathbf{n_1} \wedge (\mathbf{n_1} \wedge \mathbf{n_2})s_1 + \mathbf{f_1} + \mathbf{n_1}d_1, \tag{20}$$

where s_1 is a scalar parameter. Plane 2 can defined in terms of a vector parameter $\mathbf{s_2} = (x_2, y_2, d_2)^T$ where x_2, y_2 are free parameters and d_2 is the distance parameter of plane 2 by the equation

$$\mathbf{r} = R_2^{-1}\mathbf{s_2} + \mathbf{f_2}. \tag{21}$$

The point of intersection between this plane and the line in plane 1 is found by solving

$$R_2^{-1}\mathbf{s_2} + \mathbf{f_2} = \mathbf{n_1} \wedge (\mathbf{n_1} \wedge \mathbf{n_2})s_1 + \mathbf{f_1} + \mathbf{n_1}d_1, \tag{22}$$

for the parameters s_1, x_2 and y_2. This vector equation results in a set of three simultaneous equations which can be expressed in matrix form

$$A \begin{pmatrix} x_2 \\ y_2 \\ s_1 \end{pmatrix} = \mathbf{B} \tag{23}$$

where $\mathbf{B} = \mathbf{f_1} - \mathbf{f_2} + \mathbf{n_1}d_1 - \mathbf{n_2}d_2$ and

$$A = \begin{pmatrix} (R_2^{-1})_{11} & (R_2^{-1})_{12} & -(\mathbf{n_1} \wedge (\mathbf{n_1} \wedge \mathbf{n_2}))_x \\ (R_2^{-1})_{21} & (R_2^{-1})_{22} & -(\mathbf{n_1} \wedge (\mathbf{n_1} \wedge \mathbf{n_2}))_y \\ (R_2^{-1})_{31} & (R_2^{-1})_{32} & -(\mathbf{n_1} \wedge (\mathbf{n_1} \wedge \mathbf{n_2}))_z \end{pmatrix}.$$

The equations are solved in the paint program using standard linear equations algorithms.

3.5 3D Orientation Feedback

The lines of intersection of each plane with respect to the others provides a form of feedback to the relative positions of each viewing plane however because the views are not projections from 3D onto the screen there are no perspective cues to help the user orient the different views. This is not very important for the normal orthogonal sections which presumably will be familiar to most users of the database but for non-standard viewing directions could make easy use of the atlas and recognition of structures more difficult.

As an additional aid to orientation a “cartoon” 3D perspective display of the enclosing rectangle plus the various viewing planes is shown in the main window of the paint program. These geometric objects are displayed using OpenGL which is now emerging as the standard 3D graphics API for all Unix/X11 systems and has been adopted as standard within Microsoft Windows systems. In the paint program we are currently using a public domain package “Mesa” which provides virtually all of the OpenGL functionality on a raw X11 server unless OpenGl extensions are available for the server. So that the 3D display is reasonably efficient only very simple graphical objects are used - rectangles to represent the viewing planes, boundary displays of the domains which can be toggled on or off. It is important that the 3D display does not dominate the cpu time otherwise the reasonably interactive browsing through the volume data and painting will be lost. Figure 5 shows the top-level window of the program MAPaint v1.00 with the boundary cartoon of the 9-day mouse embryo with a number of tissues (domains) displayed.

3.6 Digital Sectioning

Paint has been developed to define arbitrary 3D domains within a grey-level voxel image. In many cases it is convenient to define a large structure for later sub-division. In many cases the sub-division can be acheived by a planar cut or “digital knife”. In pain this has been implemented by using a particular section view as the knife and the user can select which of the previously defined domains should be cut. The algorithm in terms of the woolz binary images is very straightforward. The 3D domain is defined by a set of *intervals* which can be wholey on one side of the section or cut by it. To establish this the end points of each interval are transformed using equation 5 and the resulting z' values compared with the section distance d. If both values are greater than d then the interval is part of the object on one side, if both values are smaller then the interval is part of the object on the other side and if one is larger d and the other smaller then the interval must be split into two parts. The dividing point is proportional to the respective distances from each end point.

Dividing a volume defined in terms of intervals is very efficient and the time is proportional to the number of intervals which scales as the number of planes times the number of lines.

3.7 Ray Tracing or “Laser” View

With respect to a given section a continuous line on that section can be converted to a surface by considering the set of perpendicular rays through each point on the line. The view will correspond to a “curtain” view or prism generated by translating the curved line through 3D space in a direction perpendicular to the section. The term laser view has been termed because, in analogous fashion to the section view, this view of the data can also be used to cut existing domains. There is the

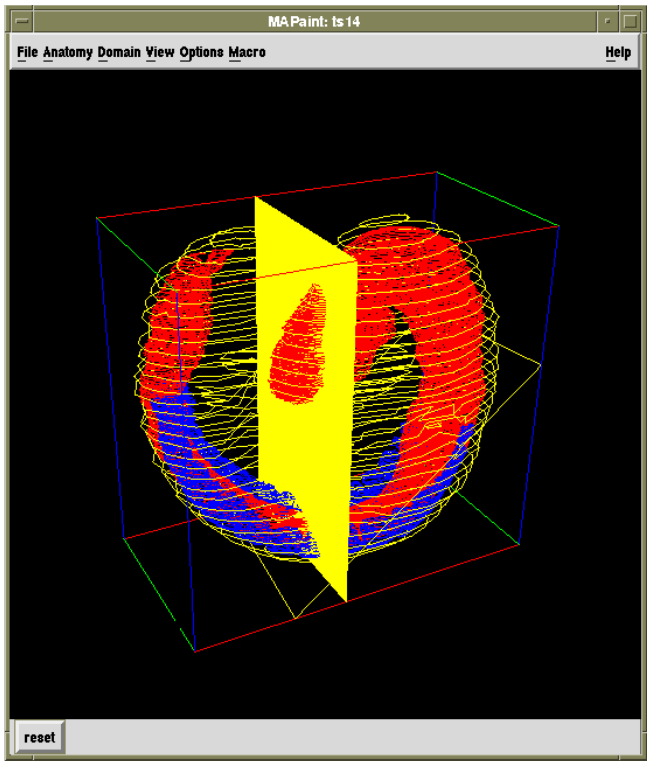

Figure 5: Top-level window of the computer program MAPaint v1.00 showing the 3D feedback available. The 9-day mouse embryo is shown as a series of outline contours, a number of anatomical domains are similarly depicted (red: neural tissue, blue: somites). Feedback for the position of two section views (not shown) are displayed, in one case as a solid plane, in the other just as the intersection polygon between the section plane and the voxel image 3D bounding box.ues to help the user orient the different views. This is not very important for the normal orthogonal sections which presumably will be familiar to most users of the database but for non-standard viewing directions could make easy use of the atlas and recognition of structures more difficult.

constraint however that the laser view must pass right through the domain to be cut or be a closed curve.

To define the laser view a set of points at unit distance along the curve are selected and for each point, (x'_i, y'_i, z'_i), we know that $z'_i = d$. At each point the value of z' is allowed to vary in unit steps from d_{min} to d_{max} and the correspondig value $\mathbf{r}$ in the original image coordinates is calculated using equation 6. The grey value for that point can then be determined. By scanning in this fashion the grey-values over the entire laser view can be established. The individual line images are aligned by the points corresponding to $z'_i = d$.

4 Painting

The basic function of the paint program is to allow interactive segmentation of a reference image to generate a series if binary images which map directly onto the original reference and (presumably) correspond to structures of some sort within the original. At the lowest level the segmentation is by interactively marking - painting - the reference image to delineate the region of interest. This is not actually changing the original, merely creating a binary mask which is displayed a coloured "wash" over the original grey-values. Within the paint program it is possible to delineate up to five different regions at the same time, more than that is difficult because the program is designed to run with the colours displayed using overlay planes for efficiency. On an 8-bit colour display with a reduced colour availability and the requirement that the overlays appear as a wash over the grey-values this effectively reduces the number of possible simultaneous paint domains to five - three overlay planes allows 32 colours per wash, reserve 32 colours for other applications, reserve 32 values for the grey-values leaving 5x32 for the overlays plus 32 colours for "solid" overlays. In the current version the user can select upt to 5 "wash" overlays and 27 "solid overlays.

For the atlas it will often be the case that the edge of one structure will delineate the boundary of the adjacent structure and therefore for efficiency and consistency each boundary should be defined just once. To aid this process the program will enforce boolean "rules" on the domains defined interactively as a simple dominance hierarchy. By default the overlays are ordered so that domain-1 has dominance over domain-2, domain-2 has dominance over domain-3 and so on. Within the program there is the option to change this dominance ordering (by "drag and drop") so that the user can modify the behavior as required.

There are currently two painting modes. The first is referred to as "drawing" in which the user draws a boundary, either by dragging which will result in an arbitrary polyline or by point selection which will result in a line consisting of straight-line segments. Once the polyline is complete the inside of the contour defines a region that is used to modify the current domain. The new domain will be added (set union) or subtracted (set difference) according depending on which button is pressed or if the modifier key `Meta` is pressed.

The second mode is to use the "paint-ball" which is equivalent to a "paint-brush" and "eraser" depending on which button is pressed or if the `meta` modifier key is pressed. With both of these options there are a variety of parameters - cursor appearence, paintball size and shape. Painting a screen resolution is very demanding therefore is is possible to temporarily bring up magnification windows to any level which allows painting to single pixel level when necessary. In the magnified windows the painted domain is constrained to the pixel regions of the base image and will therefore have a "blocky" appearence.

The default parameters controlling the painting and drawing process can be controlled from a tool control dialog. This dialog switches as the selected painting tool changes to reflect the parameters available for that option. In the case of drawing th only options area choice oc cursor - single pixel dot, cross bars or arrow. Usage to date suggests that the single dot is most useful and is therefore set as the default.

The supplementary parameters with the drawing option are paint-ball size, paint-ball shape (circle or square) and a choice of contrasting boundary for the paint-ball which can be useful on dark images.

5 Power Assist

Power assist is the term coined for using image processing techniques to help the painting process. This ranges form simple erosion and dilation of the painted region to automatic tracking of structures through the volume. The list of IP support tools is currently:

Geometry object: this is not yet implemented but will allow the user to place simple geometry objects (circle, square, etc) on the image. This is intended for applications such as marking dots, nuclei etc where the presence and distribution is the information of interest and not the detailed shape.

Threshold: select a point and the domain to add (or subtract) is defined by the set of pixels with grey value within a certain range of values around the selected pixel grey value and connected to the selected pixel by other members of the set. The range can be adjusted by dragging the mouse right and left and the resulting new domain is interactively displayed. Releasing the mouse button accepts the changes. By pressing the left mouse button with the <control> key pressed the mode is switched to allow the user to define a constraint domain within which the thresholded domain is restricted. This allows holes in an otherwise complete boundary to be blocked.

Fill: the displayed section can be considered to be segmented into number of regions according to the existing painted domains. In this sense "no-paint" can be considered a region. The fill option will add (or subtract) the singly connected region containing the selected point. The tool controls with fill allow the user to select between 4- and 8-connectedness to define the connected regions. 8-connected tends to allow too much "leaking" and therefore 4-connected is the default.

Erode, Dilate: apply erosion or dilation to the current domain with a structuring element which can have different sizes and distance metrics. The size options are of a radius between 1 and 10 and the distance options are 8-connected, 4-connected, octagonal and Euclidean. Pressing left mouse button over a region will erode it, <control> left mouse-button will dilate it. In this way the user can execute combinations of morphological operations on the defined domains within the 2D section. These are useful for defining thin boundary regions and for smoothing.

Affine transform: allows affine transform (currently only translate and rotate) of previously defined regions. The tool controls allow selection of which type of transform. In the current version the choice is between translate and rotate. To move a domain component click the left mouse-button within the required domain and drag. In translate mode the domain can now be moved into place. In rotate mode (indicated by the cursor shape) the point at which the mouse-button was pressed will act as the centre of rotation and the domain will rotate by the angle between an imaginary lever from the centre of rotation to the cursor and the x-axis. For small rotations drag out a longer lever to the right hand side of the centre of rotation.

Image Tracking: modify the domain propogated from the previous operation (usually the previous section) by using the image information associated with that domain to match to the existing image. The method matches small parts of the image near to the domain and establishes the best match at a series of points along the domain boundary. This is closely related to the "snake" techniques developed for flexible model matching but with a number of differences. The first is that there is no global assumption about the image match cost, the matching at each point on the boundary depends on the image used to define the initial boundary. This breaks the requirement that the snake must have the same appearence at all parts. The second is that there is no constraint on the absolute curvature of the boundary, only a constraint on the *change* of curvature with respect to the original. This allows the preservation of specific sharp features in the boundary which may be real. There is a control panel which allows the modification of a set of control parameters for the searched area and the snake cost function. The cost function is optimised using Dynamic Programming and is an approximation to the snake cost (making this optimisation globally correct is straightforward and will be implemented soon).

Edge Tracking: this is essentially identical to image-tracking but now the cost is derived from the local edge strength which is currently defined as the modulus of the gradient. The gradient is calculated using a derivative of gaussian filter with an adjustable width. The gaussian suppresses high frequency noise. This edge strength is not ideal because it does not include the directional component and it is possible to result in a new domain which switches between two closely parallel edges of opposite polarity.

Tablet: in many cases it is easier and more convenient to draw on printed pictures of the sections which can be done using a graphics tablet. This is useable simply as a means to control the cursor, the use of a pen to draw is more natural and easier to control than a mouse, but the main use is to draw from the printed sections. To link tablet coordinates with image coordinates the user defines two fixed points on the tablet which are transformed to two given points within image space. In our case the fixed points are generated as crosses on the printed picture and are typically at opposite ends of a diagonal. The user first marks each point then draws around the required region. The tool controls allow the user to define the image coordinates of the reference points. These points could of course match fiducial markers within the sectioned material if they exist. The tablet controls are for a Wacom cordless tablet.[1]

Domain Review: allows the user to loop throught the segmented parts of a given 3D domain and add that region to a selected domain by pressing a button. The selection also include return to grey, *i.e.* delete, and discard which will leave any regions under the reviewed region unchanged. The domain to be reviewed can be read in directly or may be generated within the program. For example it is possible to select "Check Overlap" on the read domain dialog which will then check the domain read for overlaps with any existing domain. If an overlap is found then the overlap region is registered with the domain-review controls which when activated allow the user to check each overlap region in turn and select the required destination domain. Alternatively the review domain can be read directly from disc.

The domain-review control dialog provides a complete list of the non-NULL planes within the review-domain object and selection of a plane will change the view to that plane. Pressing "review" initiates the review from the selected plane and continues until all planes are complete or "Stop" is pressed. In review mode the region to be reviewed is flashed on the screen and the user simply selects which domain that region should be added to. Very small regions (less than 10 pixels) are highlighted by adding a large square around the region which also flashes. It is not required that all the parts of a review-domain are allocated but the review-domain dialog must be dismissed before other a painting tools can be used. Any remaining regions are saved for later review unless overwritten by an operation that can result in a review-domain (*e.g.* read domain).

Domain Surgery implemented but no description.

[1]We use the Wacom UltraPad A4 Serial Tablet with the cordless ultrapen which can correct for pen tilt. Contact WACOM Computer Systems GmbH, Hellersbergstr. 4, 41460 Neuss, Germany, WWW site `http://www.wacom.de/`

The tracking tools have only recently been implemented and need tuning to establish the best default parameters for different applications.

6 Miscellany

6.1 Colour maps

Paint is designed to run an an 8-bit display using pseudo-colour or colour look-up table to map pixel values to displayed colours. This mapping can be adjusted to suit particular grey-level reference images and to allow the overlay colours to be modified. The controls are the gamma setting and a minimum and maximum value which enable certain ranges of grey-values to be enhanced.

6.2 Autosave

Like any large program paint is not bug-free therefore an autosave option has been included which saves all of the painted domains at predetermined intervals which defaults to 15 minutes. This time is user settable and autosave can be switched off but is not recommended. Data can be retrieved from an old autosave file and the user is presented with the domains from the saved file and prompted to assign the domains as required.

7 Program Help Pages

The program help pages are written in HTML the World-Wide-Web hypertext markup language and are accessed using Netscape (see `http://www.netscape.com/` if help is requested from within the program or with any html browser otherwise. Paint provokes the required page to be displayed on the netscape browser on the users terminal, if necessary starting up Netscape.

References

[1] R.A. Baldock, J. Bard, M.H. Kaufman, and D Davidson. A real mouse for your computer. *BioEssays*, 14:501–502, 1992.

[2] D Davidson, J Bard, R Brune, A Burger, C Dubreuil, W Hill, M Kaufman, J Quinn, M Stark, and R Baldock. The mouse atlas and graphical gene-expression database. *Seminars in Cell and Developmental Biology*, 8:509–517, 1997.

[3] Herbert Goldstein. *Classical Mechanics*. Addison-Wesley, Reading, MA, 2 edition, 1950.

[4] Liang Ji, Jim Piper, and Jing-Yan Tang. Erosion and dilation of binary images by arbitrary structuring elements using interval coding. *Pattern Recognition Letters*, 9:201–209, 1989.

[5] J. Piper and D Rutovitz. Data structures for image processing in a c language and unix environment. *Pattern Recognition Letters*, 3:119–129, 1985.

[6] Jim Piper and Denis Rutovitz. An investigation of object-oriented programming as the basis for an image processing and analysis system. Technical report, MRC Human Genetics Unit, Edinburgh, 1988.

[7] M Ringwald, R.A Baldock, J Bard, M.H Kaufman, J.T Eppig, J.E Richardson, J.H. Nadeau, and D Davidson. A database for mouse development. *Science*, 265:2033–2034, 1994.

[8] Denis Rutovitz. Data structures for operations on digital images. In G Cheng, C R S Ledley, D K Pollock, and A Rosenfeld, editors, *Pictorial Pattern Recognition*, pages 106–133. IAPR, 1968.

[9] E T Whittaker. *A Treatise on the Analytical Dynamics of Particles and Rigid Bodies*. Cambridge University Press, London, 3 edition, 1927.