Abstract: We have been examining a new sketching paradigm that allows direct free-hand creation of immersive graphics. Based on Apple’s QuickTime VR Technology it enables one to paint in a cylindrical panoramic space. To increase drawing speed we make extended use of the cylindrical geometry of the panoramic space, particularly the fact that a straight line on screen is a sinusoid on the flattened cylinder. We have added a secondary window to the QuickTime viewer to indicate the view angle and orientation, and various grids to help structure the space. Standard drawing functions such as colour pickers, shaped, straight and curved line drawing tools, flood fills, and cloning tools have all been implemented. In addition to these, brush controls have been added with parameterised opacity, hardness, size and spacing. The software handles a background read-only image plus an overlay (with alpha channel) containing new marks. There is also an undo buffer. To aid with the initial creation of an image, VRML models can be directly imported as wireframes. Many possible extensions of this application are discussed with the aim of opening up the digital panorama as a fully interactive VR experience.

Introduction:

Our interest in panoramas grew from the difficulties of using traditional conventional CAD and sketching tools in the architectural design process. [Dorsey 98] The creation of a 2D representation of a 3D space is a long standing problem. Constructing a 3D model is slow and laborious and does not provide the instant visual feedback that designers need, whilst painting and sketching in perspective has long been constrained by the limits of a flat canvas in which the observer has a fixed viewpoint and view direction. The interactive painting of panoramas would represent a compromise between a 3D painting system and a flat 2D system.

The word Panorama was coined in the late 18th century from the Greek words pan (all) and horama (view)[Oetterman 97]. The following century saw the construction of many cylindrical rooms as viewing chambers for enormous panoramas some with canvasses in excess of 4 000 m² [Atkinson 89] taking up to five years to paint. Many variations followed with longitudinal panoramas, cosmoramas, myrioramas, dioramas and panstereoramas all being constructed on both large and small scales [Oetterman 97]. It was not however until the advent of QuickTimeVR [Chen 95] that the panoramic medium has become globally available. As a quick web search will now show, there are many software tools available for viewing and creating panoramic images. They allow users to smoothly
view an image and click on parts of the image to link to other files. In some viewers sounds and animations can be embedded.

In terms of computer software there are two basic forms of panorama; the cylindrical panorama - analogous to the panoramic image being painted on the inside of a cylinder - and the spherical panorama in which the panoramic image is on the inside of a sphere. Practically, a cube is used for the source image in the spherical panorama. In both cases the user has a fixed viewpoint in the centre of the cylinder or sphere, and the net effect is of a limited VR world. We will be looking largely at the case of the cylindrical panorama and from here onwards we shall take the word panorama to refer to cylindrical panoramas unless explicitly stated otherwise.

In order to view and paint on a panoramic image, the cylinder must be flattened (fig.1) to create a raster image in panoramic space. However straight lines as seen on the screen are not necessarily straight lines as seen in the panoramic space, so sketching on the unwrapped image may well have unexpected effects when the image is rewrapped and seen in a panoramic viewer.

Related Work

There is now a profusion of software tools available for editing and painting 2D images [Lansdown 95]. There are also many tools to create panoramas from photographs [www.panoguide.com]. However the art of editing and painting panoramas is less advanced. We shall now discuss the software packages that do exist for the editing of panoramic images.

PanoTools [Panotools] is one of the best known image editing tools available to work directly on wrapped panoramas and is supplied as a filter plugin to Adobe Photoshop [www.adobe.com]. PanoTools asks the user to define a view within the panorama. This view is then transformed into a flat 2D image that can be edited with the full range of Photoshop’s image editing tools. At the end of drawing, the image must be then transformed back into panoramic space.

SkyPaint [www.skypaint.com] is a second application that allows the user to edit an existing panoramic image. The user can examine the panorama to be edited before selecting a view to be edited. This view is then directly exported to a standard image-editing package, where it can be edited before being reimported into the panorama. Unfortunately, in both SkyView and PanoTools, the user is only able to work on one view of the panorama at a time, as a result the painting experience loses its interactivity and tools such as selections, flood fills and line drawing are limited to the chosen view. This leads us on to the need for a painting tool that allows painting directly onto the panorama with functions to allow panning, tilting and zooming into the image during drawing.

An application [Tolba 99] developed at MIT by O. Tolba et al is the first interactive panoramic sketching tool. It uses 3D pixel coordinates and allows the user to sketch with hard brushes. It also allows the import of existing images and allows the user to extrude a predrawn plan. Extensions to this software [Tolba 01] include the ability to construct 3D shapes, reproject the painting, cast shadows and rerender the image with various techniques. However it does not include the wide variety of tools that are now
thought of as standard in painting packages: soft and opaque brushes, straight line drawing tools, multilevel undo commands, area selection tools, flood fills and cloning tools. This application is not currently commercially available although a nice Java applet for sketching is available on the web [Tolba (online)].

Software architecture:

Our application has been developed in C/C++, using the Advanced Programmable Interface (API) for QuickTimeVR [Chen 95]. QuickTimeVR uses a flattened cylinder (fig. 1) as its source image. From here on we will refer to this image as the panorama space image. It distorts this image to create a pre-screen image which is then cropped and resized to create an image on the display (fig. 2). Our software paints all strokes in the panorama space image, whilst some tools make temporary strokes in the pre-screen space for simplicity and speed. QuickTimeVR provides the interface and the tools to display the panorama with continuously variable panning, tilting and zooming.

Points in panorama space can be defined in two ways, either as 2D pixel coordinates or as angles of pan and tilt, whilst points in screen space are defined in screen pixel coordinates. Functions are provided in the QuickTime API to convert points in screen space into points or angles in panorama space. However, the reverse transformations have had to be written as they are not provided.

![Figure 1. The cylinder is flattened to make a raster image](image1)

![Figure 2. Displaying a line in a cylindrical panorama](image2)

One of the most crucial issues surrounding painting in panoramas is the order in which the various actions involved in a digital pen stroke are carried out. fig. 3 shows a flow chart of a simple drawing stroke. At the beginning of a stroke we must first save the current image to allow the brush stroke to be undone afterwards. For a standard brush stroke we must run tight loop for the duration of the stroke, alternately getting the current cursor position and painting the brush shape at the relevant position in the panoramic
image. Every time a mark (or digital ink spot) is made in the panoramic image, the area containing the digital ink spot must be distorted into the screen space.

The concept of layers will be familiar to anyone acquainted with CAD or painting systems. In using layers, the users’ paint stroke is superimposed onto all layers below the current active layer. In practice this is achieved by painting the current active layer though a layer mask onto the source image. Due to speed concerns our application does not implement layers in this way. Instead as a temporary measure, we paint simultaneously on an image containing only the strokes and on a secondary image containing he original background image as well as the strokes.

This application is currently running on a 450MHz Pentium II processor with 128-Mb ram, which provides adequate power. The maximum supported size is currently a 2500 pixel wide panoramic image. In order to draw at reasonable speeds our software makes extended use of the geometry of the panoramic image. We shall now discuss the underlying geometry of the panorama and indicate how some essential results have been found.

*Figure 3. Flow chart for a simple drawing stroke*
Theoretical basis:

Showing that a straight line is a sinusoid in panorama space

Let us begin by showing that a straight line in screen space will be represented by a segment of a sinusoid in panorama space:

![Diagram of a cylinder with points P1 and P2 and an observer's eyepoint O.](image)

Figure 4. Plan and elevation of the cylinder showing two points P1 and P2 which along with the Observer’s eyepoint in the centre of the cylinder define a plane. In fig 4b we take an elevation of the cylinder with the plane lying perpendicular to the page.

The two points, P1 and P2, at either end of a line segment, and the user’s eyepoint, O, define a plane that passes through the centre of the cylinder. Any other points on this line will also lie on this plane. If we choose an elevation (fig 4b) such that we are looking at the cylinder from a direction perpendicular to the plane normal, We obtain the following equation for the height, d of a the point P1 above the eyepoint for a plane inclined at $\varphi$ to the horizontal.

$$d = l \tan \varphi$$

Where $l$ is as drawn in fig 4a and fig 4b. Looking in plan at the cylinder (fig 4a) gives the following equation:

$$l = r \cos \omega$$

So

$$d = r \tan \varphi \cos \omega$$
Any point on the line that is defined by the plane inclined at angle $\phi$ to the horizontal and passing through the eyepoint of the cylinder will be a sinusoid. In order for this to be useful for our sketching tool we need to know how to derive the phase, $\phi$, amplitude, $A$, and offset, $C$, of such a sinusoid.

**Finding the phase amplitude and offset of the sinewave**

We begin by writing the equation for a sinusoid relating the $x$ and $y$ coordinates of the point $(x_i, y_i)$ in panoramic image space, (see fig 1)

$$y_i = A \sin(\alpha x_i + \phi) + C$$  \[1\]

Where the variable $\alpha$ is related to the width, $w$, (in pixels) of the panoramic image by the relation

$$\alpha = 2 \pi / w$$  \[2\]

By symmetry arguments, the sinusoid is offset so that half of it lies above the horizon and half lies below the horizon, thus $C$ is known – and by choice of axes can be set to 0. For a line defined by two points $(x_1, y_1)$ and $(x_2, y_2)$, we can now write out two simultaneous equations from [1] and [2] and we obtain the following equation for the phase:

$$\phi = \tan^{-1} \left( \frac{(C_0 \sin \alpha) x_2 - (\sin \alpha) x_1}{(\cos \alpha) x_1 - (C_0 \cos \alpha) x_2} \right)$$ \[3\]

Where $C_0$ is defined as:

$$C_0 = \frac{y_1}{y_2}$$ \[4\]

And the amplitude can be computed by substituting equation [3] into equation [1].

$$A = \frac{y_i}{\sin(\alpha x_i + \phi)}$$ \[5\]

When drawing we need to be able to ensure even spacing between successively drawn points. When the spacing is small compared to the width of the panoramic image, it is most efficient to take a starting point on the line and add the gradient times the change in $x$, $\Delta x_i$. The gradient of the sinusoid is simply found by differentiating equation [1], to give the following recursion relation which allows us to find the next point $(x_{i+1}, y_{i+1})$ on a sinusoid from the point $(x_i, y_i)$

$$y_{i+1} = y_i + (A \alpha \cos(\alpha x_i + \phi)) \Delta x_i$$

$$x_{i+1} = x_i + \Delta x_i$$ \[6\]
We are now able to draw the line in panorama space corresponding to a straight line in screen space.

**Drawing Tools:**

**Soft Brushes**

An important aspect of this painting system is the ability to use soft brushes. When a brush stroke is made, the software working in panorama space linearly interpolates between successive mouse points. At a spatial interval regulated by the spacing control, a ‘paint spot’ is made along the interpolated path. Each brush has an associated alpha mask. This is created when the user alters the brush controls. This alpha mask is a bitmap the same dimensions as the brush. When a paint spot is to be made, the brush is blended into the current working panorama image at the current location using the standard alpha blend for each of the red, the green and the blue (rgb) channels [Trinder 98]

\[
I = F \alpha + (1 - \alpha) B
\]

Where \(I\) is the destination pixel, \(F\) is the current brush colour, \(\alpha\) is the product of the current transparency and the alpha mask’s opacity scaled to lie in the range \(0 \leq \alpha \leq 1\) and \(B\) is the background image colour.

Controls have been added to give the user control over the transparency, spacing of between interpolated paint spots and the profile of the mask. Limited control has been given over the brush size.

*Figure 5a*  *Figure 5b*  *Figure 5c*

**Figure 5** The same paint spots when viewed from differing pan angles will appear distorted towards the edges of the screen. Note the view orientation window in the bottom left hand corner of each screen

When we paint a series of large spots we discover that they do not appear to be the same size across the screen. This effect is visible in **Fig. 5**. On panning or tilting the paint spots appear to change size. This is because the paint is applied in panorama space with each spot being painted with the same number of pixels across. In mapping the panorama
space image to screen space, the software distorts all spots but those drawn in the view centre. This is the most consistent behaviour as it avoids view dependence of the brush size.

**Straight Line Tool**

![Flowchart for a drawing a straight line.](image)

The straight line tool offered is a significant advance on existing panoramic editing packages. As, given two input points, the equation of what is a straight line in screen space can be derived in panorama space, it becomes possible to extend straight lines beyond the confines of one given view of the panorama. The implementation of the painting of straight lines also illustrates how painting in the pre-screen buffer can be used as a means to increase the drawing speed. **Fig 6** shows the basic drawing order for the creation of a straight line. When drawing a straight line the user first clicks on a point to...
denote the beginning of the line. A line is then continuously drawn from current cursor position to the start point of the line in a process known as rubber banding [Foley 90]. The rubber banding is all done in the pre-screen buffer so that no transformations need be done on panorama space pixels until the line is complete. When the start point of the line is outside the current view, a point on the edge of the view must be calculated as the starting point of the rubber band. The inverse transformation from panorama space to screen space is required for this operation. When the line is complete, paint spots are drawn along the path between the start and end points of the line using the same masking technique as for brush strokes.

When drawing thick lines across a large view angle the user notices that lines appear thicker at the ends than at the middle, even though the line thickness is constant in panorama space. This is expected behaviour, as a user panning along a predrawn straight line will notice that it maintains constant thickness in the view centre. To help predict this effect the rubber banding is more intelligent than simply being a thick straight line – it adapts its thickness to match the mean thickness of the line. Ideally the thickness of the rubber band would vary in the same way as the final stroke.

**Pipette Tool**

The pipette tool samples the rgb value of the selected pixel in screen space. As the QuickTime viewer antialias the onscreen image, the pixel needs to be sampled directly from the panoramic image. To do this, the screen coordinate is simply transformed into a panorama space coordinate and the rgb value of the pixel becomes the current brush colour.

**Flood Fill Tool**

The program written includes a flood fill tool. This tool fills all of the pixels surrounding the point clicked as long as their colour is within a given tolerance of the seed pixel. In the case of panoramas the flood fill must also fill across the ends of the panorama. The technique for flood filling used here was adapted from the Seed Fill algorithm written by P. Heckbert in Graphics Gems [Glassner 90]. The first adaptation made was to test that any given pixel value lay within a given tolerance of the seed pixel value, rather than the original test which was that they had to be the same. The second adaptation that had to be made was to cater for the case when the pixels to be filled cross the ends of the panoramic image as in fig. 7a. In order to do this, a list of filled pixels at the ends of the panorama needs to be maintained. When the fill has initially finished, we run down the right hand side of the list of filled points until we come to a filled point. We then start a fill with a seed point at the same height but at the extreme left hand end of the panoramic image as in fig. 7b. We now continue down the list of filled points until we find the next filled point. This process is repeated for all of the points at the right hand end and then repeated for all of the points at the left hand end. **Fig 7a-e.** We continue inspecting and seed filling until we have inspected both ends of the panoramic image with no seed filling occurring. A computationally more efficient approach could be to rewrite the algorithm with modular arithmetic so that when one end of the panorama space image was reached, the flood fill would continue at the other end.
Figure 7. Flood filling. The fill starts with a seed pixel (fig 7a). all pixels within a given
tolerance are filled up to the ends of the panorama (fig7b) when this operation is complete
we find the first filled pixel in the column and take the pixel on the extreme left end of the
panorama as our new seed pixel. This process is continued until an inspection of the ends
of filled pixels at the ends of the panorama leads to no new filling (fig 7e)

Clone Tool

The clone tool requires some thought when applied to panoramic painting. Its
action is to copy one part of a panoramic image to another part of the panoramic image.
Fig. 8 shows what happens when the clone operation is carried out when simply copying
one part of panorama space to another part of panorama space. If the section to be cloned is
at a different distance from the horizon than the location it is being copied to, then copying
a straight line will create a curved line in the panorama. This is the action of the clone tool
as currently implemented. Special care needs to be made with the onscreen cursor whose
position relative to the brush will change as the brush moves across the panorama. In order
to do this, the cloning point in panorama space must be transformed into screen space.
Figure 8. Cloning in panorama space can lead to some unexpected effects in screen space. In this diagram, region A has been cloned into region B in panorama space. When we transform the panorama into screen space, the straight line in region A' appears as a curve in region B'.

A possible enhanced version of the clone tool would be one that does the cloning operation in screen space during the stroke. At the end of the stroke it would then warp the entire stroke in panorama space so that it appeared as on the screen. The biggest problem with this implementation would occur when the source pixels were near the edges of the image and the destination pixels were in the centre of the image. In this case the source pixels would not be sufficiently dense to fill the destination pixels.

Selection Tool

The selection tool is a very useful tool and occurs in some form or other in all leading painting packages. In using a selection tool we have used the Vector Codec [Apple 99] that comes as part of the QuickTime API. This allows automatic filling of a boundary line defined as a series of vectors. In our implementation we draw the vector outline using the mouse points as vertex points of the outline. The vector shape is then painted into a mask and brushed into the panorama space image in the same way as individual paint spots are through the brush mask. A special case arises when the vector shape crosses then ends of the panoramic image (see fig 9). In this case we copy the vector shape into a mask that is offset so that the entire vector shape can lie on the mask in panorama space. This mask is then offset as it is copied into the panoramic image so that the selection is painted in the correct place. By using a mask, selections can be grown by simply creating a temporary secondary mask and copying it into the primary mask.
Figure 9. Copying a filled vector object in panorama space when the object crosses both ends of the panorama space.

**Viewing Tools**

In order to aid the drawing process, a simple dialog box to indicate the pan, tilt and field of view has been added. This can be seen in the bottom left of fig. 5a-c. Users find this a very important aid in drawing, particularly when dealing with a blank canvas with no geographical points of reference. This dialog box also gives the user complete control over the view – clicking or dragging in the view dialog box will cause the main view on the panorama to pan and tilt as well. Ideally this window would show the vanishing points as well.

**Gridlines**

Fig 10 The Various grids supported. Fig 10a shows the perspective grid, fig 10b shows the screen grid and fig 10c shows the panorama grid.
In addition to the view dialog box it was found necessary to include various grids as an aid to the drawing of the images. Three grids have been experimented with – a screen grid, a cylindrical grid and a perspective grid (fig. 10a-c) of these three grids, the perspective grid and the vertical grid lines from the cylinder grid are undoubtedly the most useful. The screen grid is a rectangular grid drawn in screen space and is of limited use. The cylindrical grid is created by drawing a rectangular grid in the panorama space image. When the panorama has been tilted so that the view is no longer centred on the horizon, we need to use 3–point perspective so vertical lines off the view centre will converge. The vertical lines of the cylindrical grid then become very useful.

The Perspective grid is perhaps the most useful of the three grids. It appears as a series of perspective lines converging at a user defined vanishing point. The user can define two vanishing points and hence create two perspective grids.

Our software currently orientates the perspective gridlines by asking the user to click and drag to define a vanishing point. It is also useful to be able to define the vanishing point by defining two lines along the perspective grid to be created. Care also needs to be taken that the gridlines are not drawn too strong so that they become distracting to the user.

The perspective gridlines are drawn as a series of sine waves in panorama space. The user has the option to create two vanishing points simultaneously. We will now discuss how they should be spaced.

**Even spacing of perspective gridlines**

*Figure 11a.*  
*Figure 11b.*

*Figure 11. In fig. 11a We see the spacing between the verticals decreases as they approach the vanishing point. In fig 11b we see that the verticals are evenly spaced along the line P’Q’ in world space.*

When drawing a perspective grid, the gridlines must intersect a horizontal line above or below the horizon at regular intervals (see fig. 11a). This leaves us with the
problem of finding the phase, offset and amplitude of a succession of sinusoids in panorama space as in fig. 11b so as to achieve this even spacing.

From fig. 11b we can see that evenly separated gridlines in screen space will be represented by a series of sinusoids in panorama space that all pass through the vanishing point \((x_0,y_0)\) and the points \((x_j,y_j)\) which lie on another sinusoid corresponding to the line \(PQ\). This is in fact offset another sinusoid that is at a maximum/ minimum at the value \(x_0\). From fig. 11b the sinusoid \(P'Q'\) is given by

\[ y_j = A \cos(\alpha x_j) + C \]  

[7] Where we have an arbitrary fixed amplitude \(A\).

If we consider the panoramic cylinder as viewed from above (fig 11c), with a separation of \(r\) pixels in screen space, then we can obtain the following equation for the pan angle \(\phi_j\) of the \(j^{th}\) point

\[ \phi_j = \arctan \left( \frac{jr}{l} \right) \]  

[8] Here \(l\) is the ‘pixel distance’ between the screen and the cylinder centre is a constant for a given view. We also note that \(j\) can be both positive and negative.

Combining equations [7] and [8], we obtain the following equation for the intercepts \((x_j,y_j)\) of the sinusoids that pass through the vanishing point \((x_0,y_0)\) and the line \(P'Q'\)

\[ y_j = A \cos \left[ \arctan \left( \frac{jr}{T} \right) \right] + C \]  

\[ x_j = x_0 + \alpha \arctan \left( \frac{jr}{T} \right) \]  

[9] So we now know two points – the vanishing point and the point \((x_j,y_j)\) that define each individual gridline. We can now derive the phase, amplitude for the gridlines in panorama space by using equations 1-5.

In practice setting the variable \(\frac{r}{l} = \frac{\sqrt{A}}{N}\), where \(N\) is the total number of lines to draw produces sensible results. This however leads us to the issue of when successive gridlines are so similar they appear to lie on top of one another. To test prevent gridlines from being drawn too close, we set the condition for \(x_n,y_n\) a point on the the \(n^{th}\) grid line, that if

\[ (y_n - y_{n-1})^2 + (x_n - x_{n-1})^2 < T^2. \]  

[10]
for some tolerance $T$, then the gridline will not be drawn.

Following equation [9] we can create gridlines that appear evenly spaced onscreen. We use equation [10] to ensure that the grid lines do not appear so close to one another that they merge together.

**VRML Import**

We have added the ability to import wireframe VRML (Virtual Reality Markup Language) models directly into a panorama as a means of quickly importing 3D geometry. We use Satoshi Konno’s libraries, CyberVRML 97 [Konno 97], to read the files. We then use simple trigonometry to convert geometry in Cartesian coordinates into polar coordinates, or angles of pan and tilt in our panorama. The user’s eyepoint corresponds to a view as defined in the VRML file. There are simple controls to correctly locate the imported wireframe in the panoramic cylinder. The VRML model is drawn as a wireframe model screen space. Simple controls exist to translate and rotate the input VRML model so that it aligns as expected with the panoramic image. The screen space image is updated each time the model is transformed. When the user has transformed the model into the required location, the VRML model is then drawn in panorama space. It is clearly possible to allow the imported files to be raytraced or rendered as hidden line images.

**Import of bitmaps**

Functions have been added to import and place bitmap images within a panorama, with a form of transparency being supported. The ability to import images is very helpful in the rapid creation of panoramas. It is also useful to be able to save a panorama painting project so that the stroke buffer can be returned to. In the future it would be very nice to use masks to paint in opacity on imported images and to warp them so as to appear undistorted when imported into the panorama. The ability to transform an imported image would be useful as would a tool to perspective distort them so that they lay in line with the panoramic grid.

**Extensions to cubic panoramas.**

In spherical panoramas, the geometry can be simplified by using a flattened cube as the panoramic space image. Fig 12 shows how in a spherical panorama, the user’s eye point, $O$, the start point, $A$, and the finish point, $B$, of any straight line in screen space define a plane in real space. The intersection of this plane with the edges of the cube which lie at known angles of pan and tilt can then be found by standard linear geometry. The straight line can then be drawn in panorama space by simply joining these points. The intersection points with these edges correspond to pixel coordinates on the edges of the flattened cube. By finding these points and simply joining them up with straight lines in panorama space we can draw a straight line. Similar techniques can be employed when drawing grids.

The consistency of brush sizes would cause an additional problem when painting in spherical panoramas. Maintaining constant brush size across the face of the panoramic cube, would lead to brushes appearing $\sqrt{2}$ times smaller near the corners of the cube than at
the centre of the faces of the cube. It may be necessary to scale the brushes to avoid this effect. A similar scaling and distorting effect would occur with the clone tool.

The extension to spherical panoramas of the techniques outlined above for flood fills and for the creation of a selection tool is clearly possible. However this approach does lead to the additional complexity of having to deal with not just the two ends of the panorama, but the twenty-four edges of a cubic template. However the authors cannot see any theoretical problems to this approach although note that a clear object orientated framework would be useful in dealing with the six faces of the cube.

![Image of spherical panorama diagram]

**Figure 12. A ‘spherical panorama’ is created from a cube as the image in panoramic space. Drawing the line AB in screen space will define a plane passing through the observer’s eyepoint, O. We can use geometry to derive the points of intersection with the cube and hence can draw the line in panorama space.**

**Future Work and Extensions:**

Currently many 3D rendering packages can output cylindrical projections directly. We have shown how a 3D model can be directly imported into a panorama, opening up many possibilities. Although better controls for importing wireframe models from VRML are needed, the obvious enhancements are through hidden line and rendered imports. A possible extension is to work using a z-buffer and a material buffer as in *Piranesi* [www.informatix.com], either directly imported into the panorama or prerendered in a modelling system. This development would bring interactive rendering into a VR environment. The research carried out by O. Tolba et. al at MIT [Tolba 99] allows panoramas to be constructed from an input floor plan. This is certainly an interesting way to progress. This functionality could certainly be added to our application with a little additional refinement to the VRML importing utility.

Textures and cutouts are now an essential part of any image editing software. As textures must be warped in order to work correctly in panorama space, so as to avoid too
much loss of image quality, they must be of a higher resolution than the destination image. The exact warping that occurs is dependent on the distance of the image from the horizon in panoramic space. Ideally cutouts and images would be aligned with a plane implied by the grid lines from a given vanishing point, or with surfaces implied by imported models.

In our application the entire panorama is loaded as one image. Five entire copies of the panoramic image are constantly maintained – the acting panorama space image, the marked panoramic image, the strokes image, and two undo images. Improvements could certainly be made to reduce the amount of memory used. In the future we will use multiple layers with layer masks as well as a multi-level undo facility.

Hot spots are a very important part of panoramic software, as they allow a user to navigate between scenes. Hotspots would be simple to implement within PanoPainter and would extend the panoramic world beyond the limits of one cylinder. It would become very interesting when a 3D model was imported into all linked panoramas simultaneously.

Animations are now a commonly used part of QuickTime. Animations could easily be added to PanoPainter, although the problems of warping as discussed with textures and cutouts have already been raised. The animation could be created as a prerendered movie, or the VRML importing ability could be extended to aid with initial animation creation. In terms of interactive VR presentations and computer games, this opens new possibilities. Users can make alterations to the panorama and those alterations will remain consistent as they navigate within the panoramas. As computers get faster it will become realistic to view panoramic films in which the user chooses what to look at. A panoramic editing tool would be an essential part of such an industry.

Conclusions:

In German text of 1794, it was noted that the composition of a panorama “cannot be concentrated about a single point as in a conventional picture, instead the artist must create a number of centres to draw the eye again and again” [Schneider 97]. This holds true for panoramas on a computer screen as well as for large panoramas on canvas. In developing a fully interactive set of computer panoramic painting tools we have made it possible to continuously work over the whole digital canvas allowing the digital artist fuller control over the composition than has hitherto been possible.

The software differs from traditional painting in two ways. First it has to negotiate between two different spaces, the cylindrical panorama space (which contains the data), and the flat perspective screen space, in which it is presented. Coordinate conversion and image warping are needed in both directions. For example a line drawn straight on the screen becomes a sinusoid on the panorama. Generally a painting gesture is composed in screen space as a ‘digital ink’ sequence of time-stamps and mouse coordinates, translated to panorama space, and stroked into the panorama. Finally the updated region is warped back to screen space. The second difference is that the panorama has cylindrical topology, which changes things like the flood-fill algorithm, and means that ‘straight’ lines can be closed curves. A number of familiar actions need to be entirely re-thought, such as auto scrolling while drawing a line, and the action of a ‘clone’ tool.

The results of being able to interactively paint in panoramic space are both surprising and gratifying. One can literally “paint oneself into a box” and draw what is
behind one’s head. Perspective suddenly becomes fluid. Lines drawn parallel start to converge as the scene rotates. Conversely lines drawn to a vanishing point will become parallel and then converge in the other direction as the rotation continues.

As with most innovations in interactive graphics, this software must be experienced in order to be appreciated.

Acknowledgements:

The authors would like to thank Dr. Michael Trinder for his instrumental and ongoing thoughts and comments on PanoPainter. Extended thanks go to Informatix Inc for their funding of this project, Arvan Prichard for further comments and a final word for Dr Benachir Medjdoub and Simon Ruffle for their continuous support.

References


Comparison of software for creating panoramas available at www.panoguide.com


Panotools – available from www.fh-furtwangen.de/%7Edersch/

Photoshop available from www.adobe.com

Piranesi software available from www.informatix.com


SkyPaint available from www.skypaint.com


Tolba O., Dorsey J., McMillan L. A Projective Drawing System online at graphics.lcs.mit.edu/~tolba/sketch/
