For Macintosh Programmers & Developers
MacTech MAGAZINE

QUICKDRAW 3D TRICKS

By Tom Djajadiningrat and Maarten Gribnau, Delft University of Technology

Desktop VR using QuickDraw3D, PartI

Using the View Plane Camera for Implementation of a Head-Tracked Display

About the authors...
Tom Djajadiningrat (J.P.Djajadiningrat@io.TUDelft.nl) is an industrial designer interested in products and computers which are intuitive in use. When not trying to convince others that he will now finish his PhD thesis on interfaces using head-tracked displays 'really soon', or hassling Maarten and the QuickDraw 3D mailing list with silly programming questions, he dreams of designing the 25th anniversary Macintosh.
Maarten Gribnau (M.W.Gribnau@io.TUDelft.nl) is an electrical engineer interested in Computer Graphics and Interaction Design. He has successfully delayed Tom's research project so they can now both convince others that they will complete their PhD theses 'really soon'. Apart from his research on two-handed interfaces for 3D modeling applications, he occasionally drinks a strong cup of Java when he is trying to program the ultimate internet golf game.

SUMMARY

Wouldn't it be cool to be able to look around three dimensional objects displayed on your monitor by moving your head, just as if the objects were standing there? Kind of like a hologram, but with the flexibility of 3D computer graphics. Futuristic and expensive? It could be easier and cheaper than you think. In a two part article we explain how to implement such a system, also known as a head tracked display, on a PowerMacintosh. Head-tracked perspective provides the user with a sense of depth without the use of stereoscopy. To facilitate implementation we use QuickDraw 3D, Apple's 3D graphics library. In terms of hardware all you need is a PowerMac with QuickDraw 3D, an absolute position-measuring device with three degrees of freedom and, preferably, a QuickDraw 3D accelerator board. This month we discuss the graphics-related aspects of a head-tracked display. Next month we discuss the hardware-related aspects.

INTRODUCTION

When we move about in everyday life we see objects in our environment from different perspectives. As we do this it appears that objects at different relative depths shift with respect to each other. This phenomenon, called movement parallax, is a very strong depth cue. It is possible to mimic movement parallax in virtual reality systems. In immersive Virtual Reality (VR) (where the user wears a helmet) movement parallax is combined with stereoscopy. Apart from immersive systems there is also another category of VR systems called Desktop VR which uses a conventional monitor. Desktop VR is a rather broad term which can mean anything from a simple system showing a perspective "walk-through" with mouse interactivity, to a complex system using both stereoscopy and movement parallax. With immersive VR the user's real environment is completely replaced by a virtual one, while with desktop VR a virtual scene is embedded within the user's real environment. One of the good things about a desktop VR system which uses movement parallax only is that the user's 3D impression is considerably improved without the need for some kind of 3D glasses and stereo rendering. For movement parallax, all that is needed is a way of determining the user's head position. As a result only a minimum of headwear is required.

When the user looks around a virtual scene displayed on a monitor, his head position changes, which can be detected by means of a position sensor attached to his head. In response the computer can update the perspective of the scene shown on the monitor in accordance with his new head position. The result is that the user can look around the objects in the virtual scene as if they were standing in front of him (Figure 1).

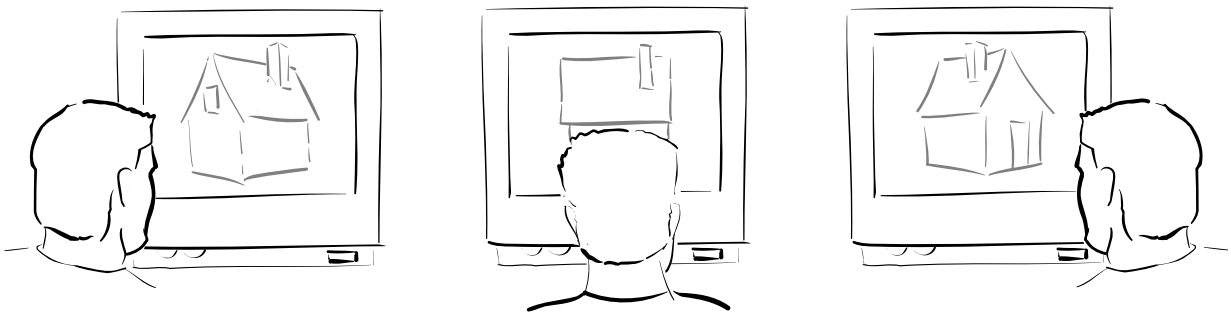

Figure 1, An observer looking at a virtual house displayed on a head-tracked display. By moving his head to the right, he views the house from the right. By moving his head to the left, he views the house from the left.

Perception psychology uses the term movement parallax to describe a particular depth cue. In the human-interfacing community many terms are

used to describe desktop VR systems which make use of the movement parallax depth cue. These terms include head-tracked display, head-slaved virtual camera, animated perspective and virtual window system. In this article we will use the term head-tracked display.

A HEAD-TRACKED DISPLAY ON THE MAC

Required and recommended software and hardware

What you need is a PowerMacintosh, QuickDraw 3D 1.5.3, a position sensor with three degrees of freedom, and preferably a QuickDraw 3D accelerator card. We use QuickDraw 3D because it facilitates communication with input devices, and implementation of the correct coupling between head position and camera movement.

The position sensor needs to detect position with three degrees of freedom and needs to be suitable for attaching to the head. A low cost option is to use a FreeD (formerly known as the “Owl”) ultrasonic tracker by Pegasus Technologies. Other, more accurate, but also more expensive options are, for example, a Dynasight infra-red tracker by Origin Instruments or a Flock of Birds electro-magnetic tracker by Ascension Technologies. Please note that we supply basic driver applications and source code for the FreeD, the Dynasight and the Flock of Birds. We also provide information on how to connect these devices to a Macintosh computer.

A QuickDraw 3D accelerator board is recommended because, with movement parallax, frame rate is quite important. The lower the frame rate, the longer the delay between establishing the sensor position and the corresponding perspective being displayed on the monitor. In the meantime the user may have moved to a different location. As a result of this lag, the perspective which is displayed does not match the user’s viewing position. To the user this mismatch expresses itself as distortion and instability of the virtual scene.

What you should know

We assume that you are familiar with the basics of QuickDraw 3D programming. If you have not dealt with QuickDraw 3D before we suggest that you have a look at the introduction to QuickDraw 3D in Develop 22 (Thompson and Fernicola, 1995) or at chapter nine "QuickDraw 3D" of "Tricks of the Mac Game Programming Gurus" (Greenstone, 1995).

OVERVIEW OF THE TWO-PART ARTICLE

There are two software components which together form our head-tracked display: a viewer application, called MacVRoom, and a driver. As mentioned previously, the implementation of the head-tracked display is described in two parts. In this month’s graphics issue we give an explanation of how to

control the camera by the user's head position to show the corresponding perspective on the monitor. We also show you how to actually get the head-tracked display up and running. This section tells you how to use MacVRoom, how to attach the sensor to your head, and how to troubleshoot.

Next month, we will explain how to write the driver and how to use the Pointing Device Manager to handle the communication between the driver and MacVRoom. We will also discuss a number of calibration methods to get the best possible results.

Depending on your needs you may wish to read parts of or all of this and next month's article. See whether you fit into one of the following categories and have a look at the overview (Figure 2). Note that for some of the information you will have to wait till next month.

1. I would like to see what this head-coupled perspective is about. I don't have a three DOF tracker though.

   Read the section "Using your head-tracked display" to get an idea of what a head-tracked display involves. Play about with MacVRoom, it switches to mouse control in absence of a three DOF tracker.

2. I have one of the tracking devices mentioned and would like to try out the head-coupled perspective.

   Read the sections "Using your head-tracked display" and "Calibration". Hook up your tracker, start up the appropriate driver and play about with MacVRoom.

3. I have a three DOF tracker, different from the ones you mentioned, and I would like to try it with MacVRoom.

   Read the section "The QuickDraw 3D Pointing Device Manager" and build a driver for your particular tracker. Then read the sections "Using you head-tracked display" and "Calibration". Hook up your tracker, start up the appropriate driver and play about with MacVRoom.

4. I have an serial input device and I am interested in using it in conjunction with the Pointing Device Manager. I am not interested in head-tracked displays.

   Read the section "The QuickDraw 3D Pointing Device Manager" and build a driver for your particular input device.

5. I have one of the tracking devices mentioned and I want to incorporate a head-tracked display into my own QuickDraw 3D application.

   Read the section "Head-tracked Camera Methods" and incorporate the code into your own app. Then read the sections "Using your head-tracked display" and "Calibration". Hook up your tracker, start up the appropriate driver and your head-coupled perspective enabled app.

6. I have a three DOF tracker but it is not one you mentioned. I also want to incorporate movement parallax into my own QuickDraw 3D application.

Lucky you! You'll have to read both parts of the article completely!

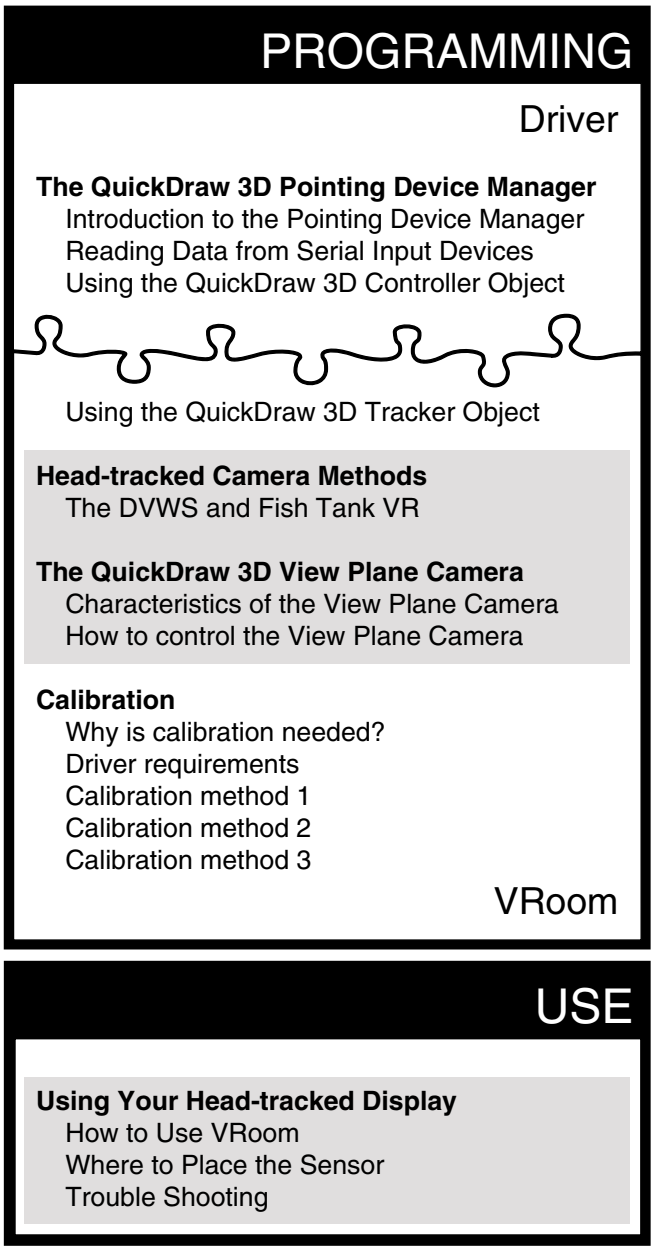

Figure 2, Overview of the two-part article. The subjects which are covered this month are marked with a grey block.

HEAD-TRACKED CAMERA METHODS

The Delft Virtual Window System and Fish Tank VR

When it comes to implementing movement parallax we could use a conventional perspective camera, position it in the virtual world to

correspond to the user's head position, and orient it in such a way that it always looks at the center of the virtual scene (Figure 3). This is what happens when you choose Delft Virtual Window System (DVWS) (Overbeeke et al., 1987; Smets et al., 1987) from the projection menu. Such a projection method is also known as "on-axis" projection, because the center of the image coincides with the camera's viewing axis.

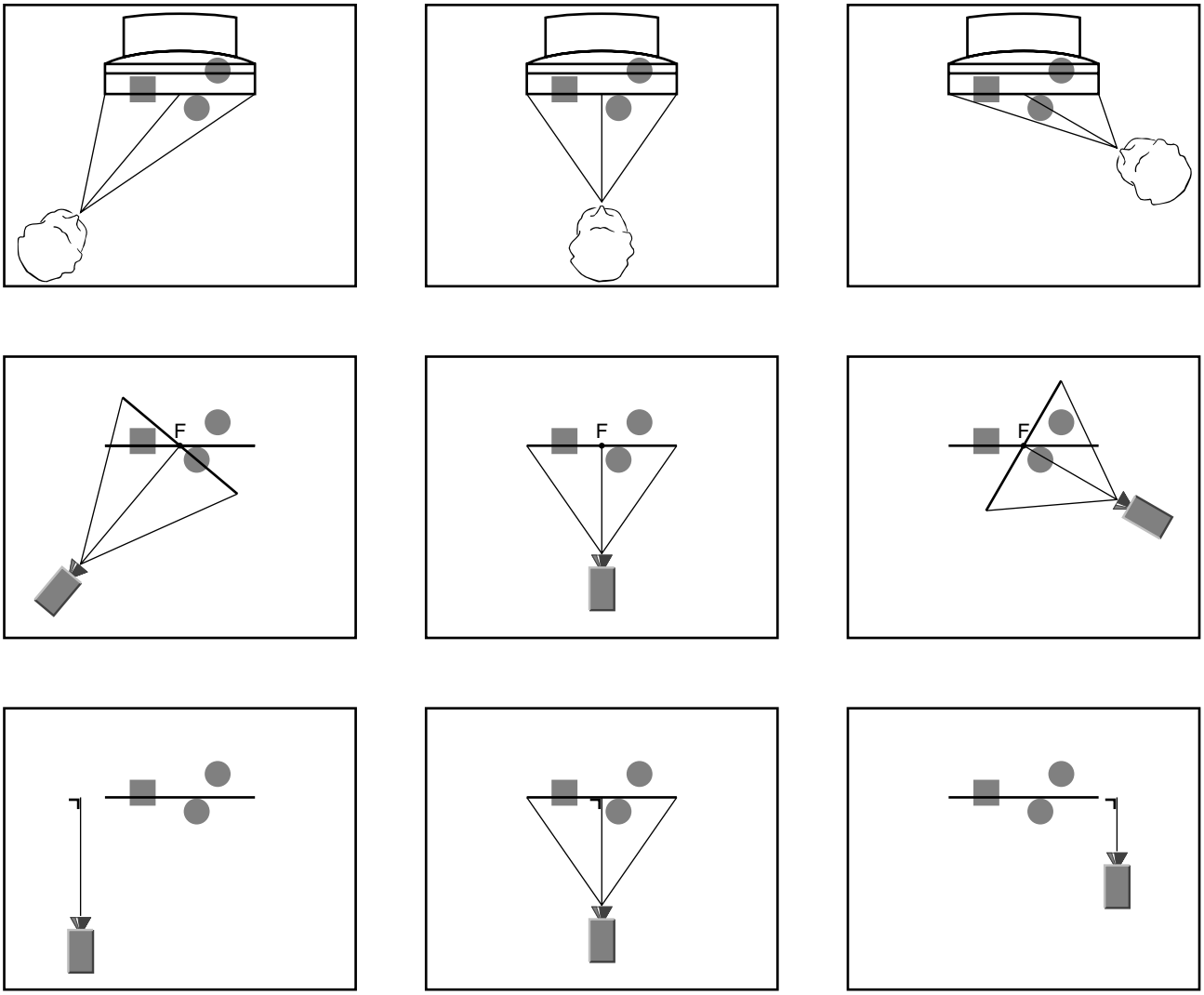

Figure 3, Three different observer positions (top row), the corresponding camera positions for the DVWS projection method (middle row), and the corresponding camera positions for the Fish Tank VR projection method (bottom row).

The top row shows three different positions of an observer in front of a monitor. The middle row shows the virtual camera which always is pointed at the fixation point F. The main advantage of this projection method is that it is perceptually robust as the virtual scene is always displayed in central perspective. Even if the system is not properly calibrated the resulting image does not appear distorted. As Figure 4 shows, the perspective is that of a

regular photograph. This means that camera movement can also be scaled compared to observer movement. So the observer can look around the virtual scene completely and even view it from the back with relatively small head movements, though the scene does not appear to be stationary. Also, observers who are not wearing the tracker and who therefore do not control the perspective, still get an undistorted view on a rotating scene.

Implementation of the DVWS requires only placement and zooming of an ordinary QuickDraw 3D aspect ratio camera. We assume you are familiar with this type of camera and therefore do not explain it any further in this article Please have a look at the routines NewAspectRatioCamera (in ViewCreation.c) and AdjustAspectRatioCamera (in AspectRatioCamera.c) if you need more help.

The disadvantage of an on-axis projection is that it is difficult to perfect the illusion that the virtual scene is rigidly connected to the physical world. When we use an on-axis projection with a conventional perspective camera the result is a perspective image of the virtual scene. However, the user already looks in perspective at the monitor screen which displays the image. The result of the compounding of perspectives is that lines in the virtual scene and lines in the real world which are meant to stay parallel do not appear to stay perspectively parallel when the user views the virtual scene from different angles. Therefore the virtual scene does not seem rigidly connected to the monitor. Instead it appears to rotate relative to the monitor. Perhaps the easiest way of thinking about this is as follows. Take a rectangular sheet of paper and tape it to your monitor so that the edges of the paper run parallel to the edges of the monitor. From whatever viewpoint you look at the monitor and the sheet of paper, the edges of the monitor and the paper will always run perspectively parallel. In other words, they always intersect at the same vanishing point. If we create a virtual equivalent of the sheet of paper and look at it with an Aspect Ratio Camera from different angles, the resulting 2D image of the virtual sheet of paper will be perspectively distorted. But this is not what we want! After all, the physical sheet of paper never changed its shape. If we wish to avoid this compounding of perspectives we need to use a different projection method.

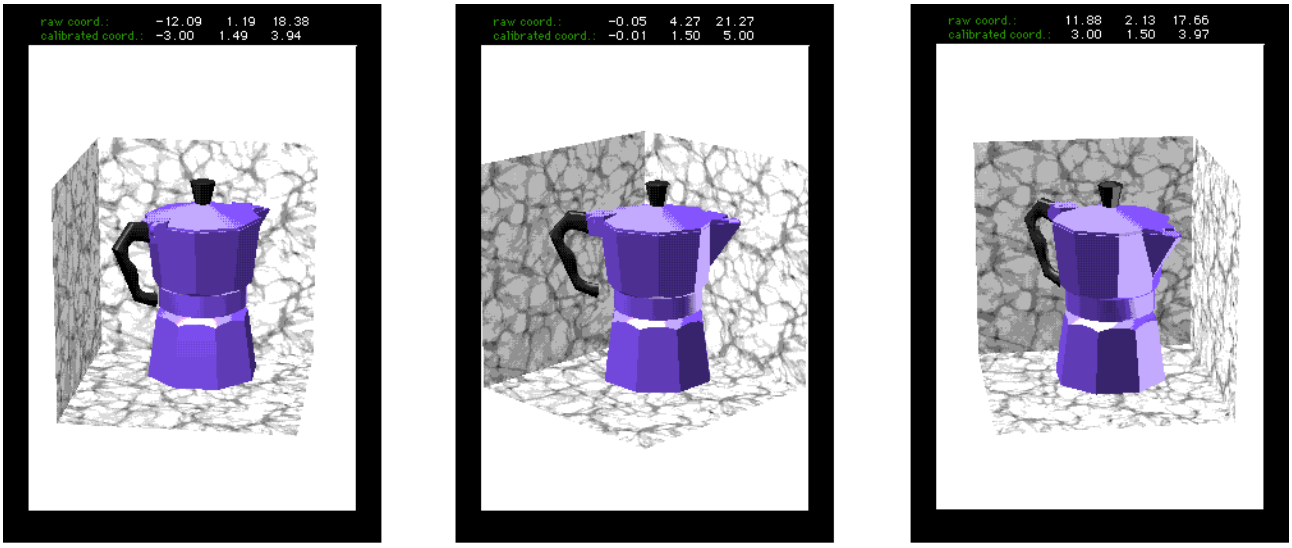

Figure 4, Three perspectives according to the Delft Virtual Window System projection method generated by three different head positions. From left to right: viewed from the left, from the middle and from the right.

This projection method, which is called an "off-axis" projection method and is often referred to as Fish Tank VR (Ware et al., 1993), is shown also in Figure 3.

The bottom row shows the three virtual camera placements which correspond to the observer positions in the top row. The line of sight of the virtual camera is kept perpendicular to the display by translating the camera without rotating it. This will prevent the compounding of perspective. The resulting images are shown in Figure 5. Although the pictures look distorted at first instance, you can find the head position from which they appear to look right. To find this position, use the numbers in the calibrated coordinate fields of each picture in the following manner. The numbers are x,y,z coordinates, multiples of the width of the pane with the white background. The origin of the coordinates is in the center of this pane. For example, to position your head correctly for the left picture, move your head three pane widths to the left, one and a half width to the top of the page and four widths out of the page.

Figure 5, Three perspectives according to the Fish Tank projection method generated by three different head positions. From left to right: viewed from the left, from the middle and from the right.

A problem is that as the camera moves to one side, the scene will move off the monitor in the other direction. We need to be able to specify a part of the imaging plane which is off-axis. QuickDraw 3D conveniently provides a View Plane camera for this purpose.

THE QUICKDRAW 3D VIEW PLANE CAMERA

Characteristics of the View Plane Camera

Figure 6 shows a View Plane Camera and Listing 1 describes the View Plane Camera data structure. Just as the familiar Aspect Ratio Camera, a View Plane Camera uses a *struct* of type TQ3CameraData to specify its placement, hither and yon planes, and view port. The view plane is situated perpendicular to the camera vector at a distance specified by the view plane parameter. The point where the camera vector intersects the view plane defines the origin of the view plane coordinate system. We can specify which part of the view plane we wish to render through the halfWidthAtViewPlane (dx), halfHeightAtViewPlane (dy), centerXOnViewPlane (Cx) and centerYOnViewPlane (Cy) parameters.

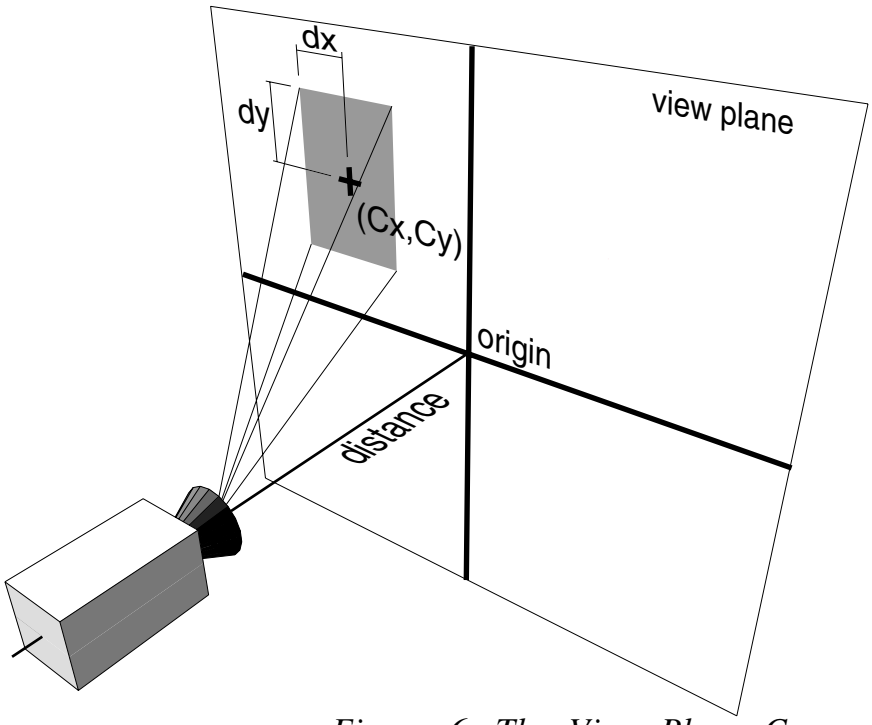

Figure 6, The View Plane Camera

Listing 1: View Plane Camera data structure

```
typedef struct TQ3ViewPlaneCameraData
{
  TQ3CameraData  cameraData;
  float          viewPlane; // distance to view plane
  float          halfWidthAtViewPlane; // dx
  float          halfHeightAtViewPlane;// dy
  float          centerXOnViewPlane;  // Cx
  float          centerYOnViewPlane;  // Cy
} TQ3ViewPlaneCameraData;
```

How to control the View Plane Camera

We have chosen to use a right-handed coordinate system with the positive Y-axis pointing upwards and the positive Z-axis pointing out of the monitor. This is convenient as its orientation matches that of the FreeD and the Dynasight when they are put on top of a monitor. The virtual camera always remains parallel to the Z-axis so that its line of sight is always perpendicular to the xy plane. The view plane coincides with the XY plane and the part of view plane which is rendered is centred around the world origin (Figure 7).

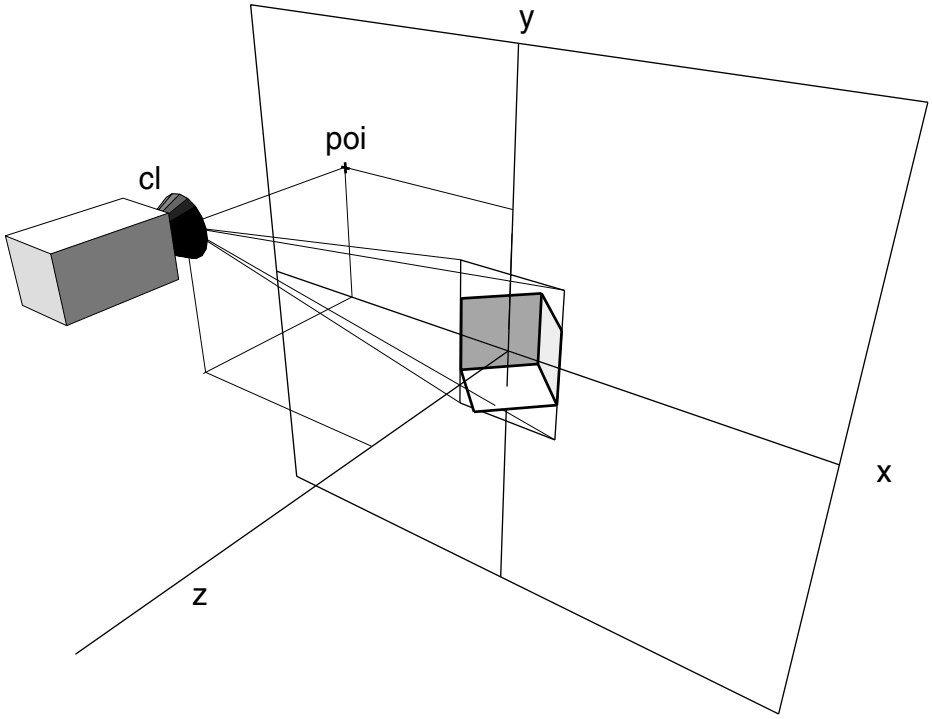

Figure 7, View Plane Camera looking at the display space in which the model appears. The line of sight of the camera always remains parallel to the z-axis. cl is camera location, poi is point of interest.

There are two parts to controlling a View Plane Camera for an off-axis projection. We need some set-up code and some code which adjusts the camera on every nullEvent.

The set-up code is from ViewCreation.c and is in Listing 2. Although the camera placement used in this set-up function is adjusted immediately after start up to the tracker data, it is good practice to use values which ensure that the scene will be visible. That way if we see an image on start up which disappears immediately afterwards, we know that the app is rendering all right but that afterwards the camera placement has become garbled.

The hither and yon planes truncate the viewing pyramid to a viewing frustum. Virtual objects are clipped against this viewing frustum. To make absolutely positive that we do not get Z-buffer problems with acceleration cards which have only 16 bit Z-buffering we put the hither plane just in front of the display space and the yon plane just behind it. The display space is one QuickDraw 3D unit deep, and positioned half in front of the view plane and half behind it. Therefore the frontmost point is at z=0.5 and backmost point is at z=-0.5. On each nullEvent, we adjust the values of hither and yon, which are relative to the position of the camera, to keep the hither and yon planes at the same location in the world coordinate system.

The viewPort parameter lets you choose which part of the area you have cut out of the view plane is mapped to the pane. In our case the full view port is used and is not adjusted on a nullEvent.

When the camera is created the centerXOnViewPlane = 0 and centerYOnViewPlane = 0 so that the part of the view plane which is rendered is centred around the world origin. The part of the view plane which is rendered is made one QuickDraw 3D unit wide, so the halfWidthAtViewPlane = 0.5. When we get to the section on calibration we will see why this is convenient. The ratio of the halfWidthAtViewPlane and halfHeightAtViewPlane parameters should equal that of the pane width and height. We have made it 1:√2. Thus the width of the display space takes up the full width of the pane, while the height of the display space is less than the height of the pane. This extra height is necessary to prevent clipping of the background planes. For example, if the pane was made to fit the height of the cubic display space, the front half of the ground plane would be clipped by the bottom of the pane, as soon as the user would move his eye above the bottom of the pane. If you find that clipping still occurs, you can decrease the pane width to height ratio to, for example, 1:2.

Listing 2: ViewCreation.c

NewViewPlaneCamera

```
TQ3CameraObject NewViewPlaneCamera(void)
{
  TQ3Status              returnVal = kQ3Failure ;

  TQ3ViewPlaneCameraData perspectiveData;
  TQ3CameraObject        camera;

  // cameraLocation is on the z-axis
  TQ3Point3D             from = { 0, 0, 5 };

  // view plane origin at world origin
  TQ3Point3D             to = { 0, 0, 0 };
  TQ3Vector3D          up = { 0.0, 1.0, 0.0 };

  float                  paneWidth = kPaneWidth;
  float                  paneHeight= kPaneHeight;


  perspectiveData.cameraData.placement.cameraLocation=
    from;
  perspectiveData.cameraData.placement.pointOfInterest=
    to;
  perspectiveData.cameraData.placement.upVector=up;
```

```
    // The display space is 1 QuickDraw 3D unit deep.
    // We put the hither pane just in front of the display space...
    perspectiveData.cameraData.range.hither = from.z - 0.5;

    // and the yon plane just behind it.
    perspectiveData.cameraData.range.yon = from.z + 0.5;

    // use the full viewPort
    perspectiveData.cameraData.viewPort.origin.x = -1.0;
    perspectiveData.cameraData.viewPort.origin.y = 1.0;
    perspectiveData.cameraData.viewPort.width = 2.0;
    perspectiveData.cameraData.viewPort.height= 2.0;

    // the distance from the virtual camera to the view plane equals
    // the z-coordinate of the camera position, since the view plane coincides with
    // the xy plane, and the camera vector is parallel to the z-axis.
    perspectiveData.viewPlane       = from.z;

    // the aspect ratio of these parameters should equal
    // that of the paneWidth and paneHeight.
    // For convenient calibration we've made widthAtViewPlane = 1,
    // so halfWidthAtViewPlane = 0.5
    perspectiveData.halfWidthAtViewPlane  = 0.5;
    perspectiveData.halfHeightAtViewPlane=
      0.5*paneHeight/paneWidth;

    // image centred around center of the xy plane
    perspectiveData.centerXOnViewPlane    = 0;
    perspectiveData.centerYOnViewPlane    = 0;

    camera = Q3ViewPlaneCamera_New(&perspectiveData);

    return camera ;
}
```

The code which adjusts the camera on every nullEvent is in Listing 3. Although MacVRoom is meant to be used with a three DOF position-measuring device and its QuickDraw 3D driver (explained next month), we do provide some code to couple the camera to the mouse. This allows you to see the effect of the View Plane Camera without a tracker being present. Of course you can only benefit from the head-coupled perspective when you have a position-measuring device with three DOF. With a mouse the perspective will look strange and distorted, though you can try to move your head to find the eye position at which the perspective for the current mouse position looks correct.

When a three DOF position-measuring device and its QuickDraw 3D driver are present, we first get the position of the tracker from the tracker object. To this raw position the calibration matrix is applied (Calibration is

necessary because it is neither possible to put the sensor in the middle of the eye, nor the tracker in the middle of the pane. We will discuss calibration extensively next month). Now we can use the resulting position to control the camera. To keep the camera parallel to the Z-axis we make the point of interest the same as the camera position, with the only difference that the z-coordinate is set to zero. Now we need to specify which part of the imaging plane we wish to record. We do this by setting the viewPlane, centerXOnViewPlane and centerYOnViewPlane parameters. The viewPlane parameter is the distance from the camera to the viewPlane. We can simply set it to the value of the z-coordinate of the camera position. The centerXOnViewPlane and centerYOnViewPlane are set to the minus x-coordinate and the minus y-coordinate of the virtual camera. This ensures that we're always looking at an imaging area which is centred around the world origin. As the cubic display space is also centred around the world origin, it does not shift within the pane, even though the camera is translated without being rotated.

Listing 3: ViewPlaneCamera.c

AdjustViewPlaneCamera

```
TQ3Status AdjustViewPlaneCamera( DocumentPtr theDocument)
{
  Point               mousePosition ;

  TQ3CameraObject     myCamera ;
  TQ3CameraPlacement  myCameraPlacement;
  TQ3Point3D          from, to;
  TQ3Vector3D         up     = { 0.0, 1.0, 0.0 };
  float               viewPlane;
  float               centerX;
  float               centerY;

  TQ3Status           status;

  TQ3Boolean          positionChanged;


  // If no controller could be found
  // during initialization, fTracker was set to NULL.
  // In that case we're using mouse control.
  // This allows us to check whether things are working
  // alright without a tracker present.

  if (theDocument->fTracker == NULL)
  {
    GetMouse(&mousePosition) ;
    LocalToGlobal(&mousePosition) ;
```

```
      // Set camera position based on mouse coordinates.
      // Since the mouse has only got two DOF we're
      // setting the Z-coordinate to a fixed value.
      // We have chosen to set it to 5.0 * the pane width,
      // which is a reasonable approximation of the observer-screen distance.
      from.x =(float)(mousePosition.h-gScreenMiddle.h)/10;
      from.y =(float)(mousePosition.v-gScreenMiddle.v)/10;
      from.z =5.0;
   }
   else
   {
      // Get the position from the tracker object.
      status = Q3Tracker_GetPosition(
            theDocument->fTracker,
            &from,
            NULL,
            &positionChanged,
            NULL);

      // If it fails or doesn't bring us a new position
      // we're not bothering with adjusting
      // the virtual camera.
      if ((status != kQ3Success) ||
        (positionChanged == kQ3False)) goto bail;

      // apply the calibration matrix on the raw position
      Q3Point3D_Transform(&from, &gCalibrationMatrix,
&from);
   }

   // Set the point the camera looks at
   // the line of sight of the camera is always
   // parallel to the Z-axis, so we can simply
   // set the Z-coordinate to zero.
   to.x    =  from.x;
   to.y    =  from.y;
   to.z    =  0.0;

   // Fill in the camera placement.
   myCameraPlacement.cameraLocation  = from;
   myCameraPlacement.pointOfInterest   = to;
   myCameraPlacement.upVector      = up;

   // The distance to the viewPlane is simply
   // the value of the Z-coordinate.
   viewPlane = from.z;

   // We're cutting out a piece of the viewPlane
   // centered around {0,0,0}.
   centerX = -from.x;
```

```
    centerY = -from.y ;

    // Work out the range of the hither an yon
    // This is to make sure we don't get Z-buffer problems
    myRange.hither = from.z - 0.5;
    myRange.yon = from.z + 0.5;

    // Get the camera from the view
    Q3View_GetCamera (theDocument->fView, &myCamera);

    // Fill in the fields of the camera
    Q3Camera_SetPlacement(myCamera,&myCameraPlacement);
    Q3ViewPlaneCamera_SetViewPlane  (myCamera, viewPlane);
    Q3ViewPlaneCamera_SetCenterX    (myCamera, centerX);
    Q3ViewPlaneCamera_SetCenterY    (myCamera, centerY);
    Q3Camera_SetRange(myCamera, &myRange);

    // Dispose of the camera object
    Q3Object_Dispose( myCamera ) ;

    return kQ3Success ;

    bail:
    return kQ3Failure ;
}
```

USING YOUR HEAD-TRACKED DISPLAY

How to Use MacVRoom

MacVRoom allows the user to calibrate the tracker, load a 3DMF model, display it inside a concave, cubic space and look at it from different points of view by moving his head. If a position tracker and driver can be found, MacVRoom starts with a calibration procedure. Please follow the on-screen instructions. The user is asked to keep the sensor twice the width of the pane in front of the top-left corner of the pane and to press return. This process is then repeated for the bottom-right corner of the pane (Next month we will provide you with much more info on calibration, as well as a more accurate calibration method). If no position tracker and driver can be found, MacVRoom switches to mouse control, and the calibration procedure is skipped.

To create an undisturbed backdrop for the rendering, MacVRoom hides the Finder and any other application by a window which covers the whole screen, and creates a QuickDraw 3D pane inside that window. The cubic space in which the model is displayed is formed by three square polygons (Figure 8). Its center is at the center of the pane and the front-rear diagonal of its ground plane runs parallel to the z-axis.

From the projection menu the user can choose between two different projection methods called the Delft Virtual Window System (DVWS) (Figure 4) and Fish Tank VR (Figure 5). Please read the section "Introduction to Head-tracked Camera Methods" for an explanation of these terms. With the DVWS, the middle of the left-right diagonal plane of the cubic display space appears to be pinned to the monitor screen. With Fish Tank VR, the whole of the diagonal plane coincides with the monitor screen. The left and right vertical edges thus appear to be stuck to the screen.

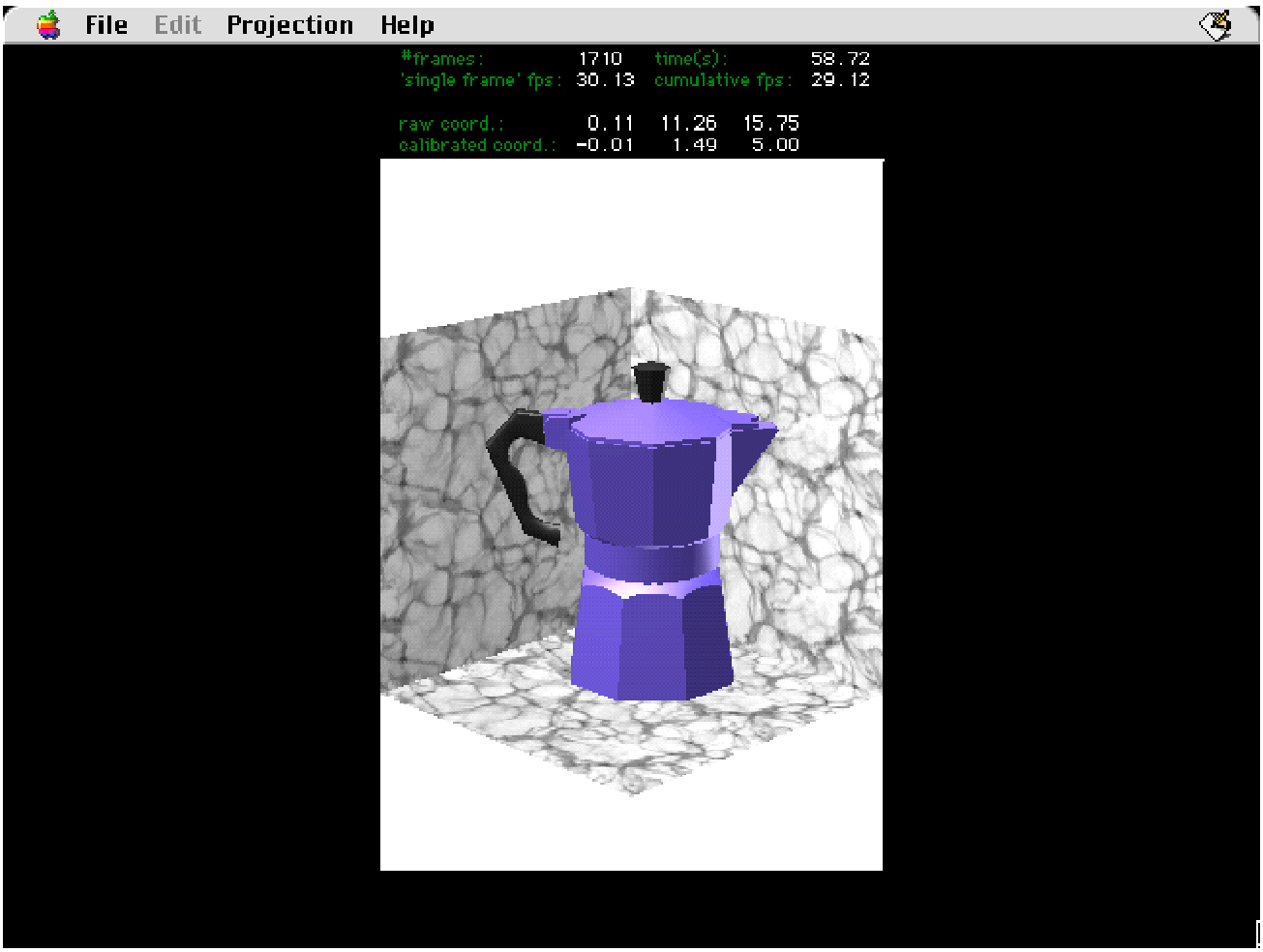

Figure 8, A screenshot of the MacVRoom application.

When everything is up and running you will see a status bar above the rendering pane. This status bar gives you the following information:

Frame count: The number of frames that have been rendered so far.

Time: The time which has elapsed since rendering the first frame.

'Single frame' fps: This frame rate in frames per second is the inverse of the time it takes to render a single frame. It does not take into account the time it takes to complete other tasks, such as event handling and adjustment of the camera. Note that, depending on the complexity of the

model and the speed of the Mac, you may get figures exceeding the screen refresh rate. While the Mac can render the model at a frame rate greater than the screen refresh rate, it can never display the rendered images at such a rate.

Cumulative fps: This is the frame rate in frames per second calculated by dividing the elapsed time by the number of frames rendered since start up. This cumulative frame rate suffers from start-up overhead. You'll notice that it slowly increases.

Raw coordinates: These are the x, y and z coordinates as they come in from the driver application.

Calibrated coordinates: These are the x, y and z coordinates of the virtual camera.

Where to Place the Sensor

OK, so you've got a PowerMac with QuickDraw 3D, a three DOF tracker, a driver and MacVRoom, and everything is calibrated. All you need to do now is to attach the sensor to the head in some way. You can choose between viewing the scene with one eye or with two eyes. Some people find that, in absence of stereo imaging, one-eyed viewing of the virtual scene results in a more convincing depth impression than two-eyed viewing. This may be because with two-eyed viewing there is a depth cue conflict between the stereoscopically perceived surroundings (Mac, monitor, table etc.) and the monoscopic virtual scene. A disadvantage of viewing the scene with one eye is that you either need to keep the other eye closed or wear an eye patch, which means more headwear and thus discomfort.

If you use one eye, try to mount the sensor as close to the viewing eye as possible (Figure 9 and 10). Remember that we're trying to establish the position of the eye. As we cannot mount the sensor in the eye we have to make do with mounting the sensor near the eye. As a consequence there will always be some amount of viewpoint dislocation. Since we're using a sensor which provides only position and no rotation information we cannot accurately compensate for this dislocation as we do not know which way the user's head is tilted. If you're viewing with two eyes, you may wish to consider mounting the sensor between the eyes. In either case you could you use a headband or just the frame of a pair of spectacles. Before you mount the sensor to headband or glasses, make sure that the orientation and position of the sensor is such that it stays within track of the base unit in the region in front of the monitor.

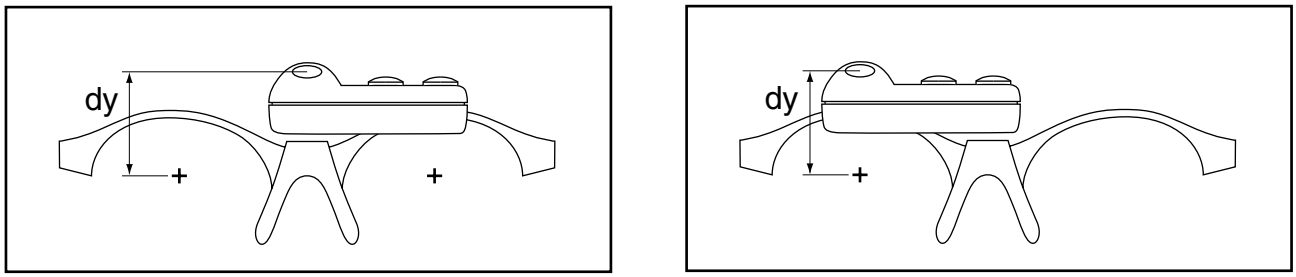

Figure 9, The FreeD sensor mounted in the middle for two-eyed viewing (left), and above the right eye for one-eyed viewing (right).

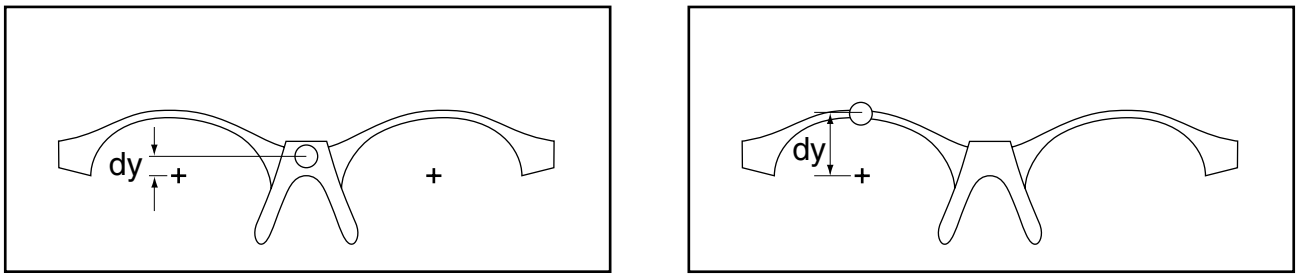

Figure 10, The Dynasight reflector mounted in the middle for two-eyed viewing (left), and above the right eye for one-eyed viewing (right). Note that the offset in the vertical direction (dy) differs.

Trouble shooting

What if it kind of works but you don't find it really convincing? The following hints may give some improvement:

1. Have you accurately followed the calibration procedure? If the sensor is not accurately calibrated the virtual camera does not correspond to the user's head position. To the user this misalignment appears as distortion. You can check whether the system is properly calibrated by looking at the calibrated coordinate field in the status bar, and holding the sensor stationary in the following locations. The left edge of the pane should give x=-0.5 and the right edge x=0.5. As we're working with a width:height ratio of 1:√2 the top edge of the pane should give 0.71 and the bottom edge -0.71. Holding the sensor in front of the monitor by the width of the window, should give a z-coordinate of z=1. Of course these figures are only approximate. It is unlikely that you will find exactly these values, but at least you can find out whether calibration is OK-ish or has gone completely haywire. If the values are off, restart MacVRoom to calibrate the system anew.
2. Is the frame-rate acceptable on your machine? Below 15 fps you'll probably suffer from a lot of delay. Make sure you are running with the QuickDraw 3D runtime extensions rather than the debug extensions as the latter are much slower. Try switching off the textures on the background

planes by changing the conditional compiler statement "#define TEXTURE 1" to "#define TEXTURE 0". Try loading a less complex model with fewer polygons and fewer textures. Try running in thousands rather than millions of colors. Make sure that AppleTalk is inactive and that you don't have any extensions active which cause noticeable interrupts, such as fax extensions. Anything which overlaps the rendering pane, either another window or the control strip will cause performance degradation. Remember that QuickDraw 3D does not like virtual memory. Your choice of geometry can have a considerable influence on frame rate. With the interactive renderer trimeshes yield the best performance, meshes the worst, with polyhedrons somewhere in between (Schneider, 1996; Zako et al., 1997). 3DMF Models can often be made to render significantly faster by using 3DMF Optimizer (Pangea). Finally, if you are running without hardware acceleration and use a PCI PowerMac, consider adding an accelerator board. There is lots of information available on the QuickDraw 3D home page.

CONCLUSIONS

In this month's Part I we documented the graphics-related aspects of the implementation of a head-tracked display on PowerMacintosh using QuickDraw 3D. A head-tracked display gives the user a depth impression of a 3D scene without the use of stereoscopy. We showed you how to use the View Plane camera to achieve a perspective projection which takes the user's head position into account. We also showed you how to use the viewer application and how to troubleshoot. In next month's Part II we will have a look at the hardware-related issues of our head-tracked display.

ACKNOWLEDGMENTS

We gratefully acknowledge P.J. Stappers, J.M. Hennessey, C.J. Overbeeke and A. van der Helm, for their constructive criticism during the preparation of this article.

BIBLIOGRAPHY AND REFERENCES

Apple Computer, Inc. (1995). *3D Graphics Programming With QuickDraw 3D*. Reading, MA: Addison-Wesley Publishing Company.

Fernicola, P. and Thompson, N. (1995, June). *QuickDraw 3D: a New Dimension in Macintosh Graphics.* Develop 22, pp.6-28

Greenstone, B. (1995). QuickDraw 3D. In McCornack et al. (eds.), *Tricks of the Mac Game Programming Gurus* (pp. 491-546). Indianapolis, IN: Hayden Books.

Pangea Software, http://www.realtime.net/~pangea/.

Overbeeke, C.J., Smets, G.J.F. and Stratmann, M.H. (1987). *Depth on a flat screen II*. Perceptual Motor Skills 65, p.120.

Quickdraw 3D home page, http://www.apple.com/quicktime/qd3d/

Schneider, P.J. (1996, December). *New QuickDraw 3D Geometries*. Develop 28.

Smets, G.J.F., Overbeeke, C.J. & Stratmann, M.H., (1987). *Depth on a flat screen*. Perceptual Motor Skills 64, pp.1023-1034.

Ware, C., Arthur, K. & Booth, K.S. (1993). *Fish tank virtual reality*. Proceedings of the INTERCHI'93, pp.37-42.

Zako, R. and the Artifice Support and Testing Team (1997). http://www.artifice.com/tech/geometry_performance.html.