Methods for Generating and Displaying Stereo Images on VR Systems using Quad-Buffered Graphics Adapters

Dimo Chotrov
Technical University Sofia
Sofia, Bulgaria
dchotrov@tu-sofia.bg

***Abstract*—This paper considers approaches for generating stereoscopic images by using the left and right video buffers of a quad-buffered graphics adapter. Sample OpenGL code is given for specifying camera projection to generate correct stereo pair images. The paper also shows how by using the outputs of a quad-buffered graphics adapter the stereo pairs can be displayed on different 3D capable display systems like monitors, 3D TVs and Power-wall projection systems. Example setups are presented and discussed.**

Keywords—Virtual Reality; Stereo Visualization; Quad-Buffered Graphics Adapters

I. Introduction

It is well known that people perceive most of the information about their surrounding environment through their eyes. Following, if an artificially generated world, usually referred to as virtual environment (VE) or virtual reality (VR), wants to be convincing to the user one of the most important perception channels that has to be stimulated by it is the visual channel. The more convincing the visualization – the greater the feeling of immersion, of being part of the VE. This is where stereo visualization comes into place. Most of the displays nowadays still show a two dimensional (2D) image. Even if the virtual scene that is being visualized is three dimensional (3D) what the viewer sees is a 2D image of the 3D scene. Thanks to the natural depth perception that we have stereo displays can be used to present the viewer with a 3D representation of the scene increasing significantly the immersion.

This paper presents an overview over the general workflow for generating and displaying stereo images for virtual environments by utilizing quad-buffered graphics adapters. First a short explanation of how do we actually perceive depth and how this knowledge is used by VR applications to visualize artificial worlds in depth is provided. After that a more practical approach is assumed by giving examples for stereo pair generation, including sample OpenGL code, and pointing out the differences when rendering images for mono and for stereo visualization. At the end different possibilities for the setup of stereo visualization systems are described and again examples for such systems are given.

II. Background

A. Depth Perception

There are two ways in which people perceive distance and depth [2][3] – one is due to personal experience and different relationships between the observed object and its surroundings. Some of these depth cues are object occlusion, movement parallax, perspective, relative size to known objects, etc.

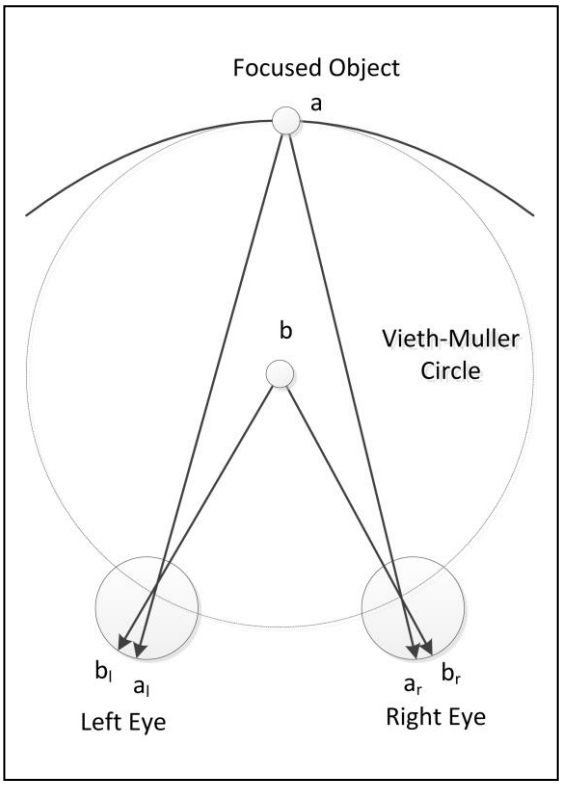

Fig. 1. Binocular perception of objects

The other reason for depth perception is the so called binocular disparity – because our eyes are horizontally separated from each other they actually see and transmit to the brain slightly different images of our surroundings. From this difference, called binocular disparity or parallax, the brain extracts the depth information [1][2].

When we look at something our eyes automatically focus on that object (see Fig. 1) in this way defining the focus distance. Depending on how object images are projected on the retinas the brain decides whether the object is closer or further away than the focused one.

B. Stereo Projection

Stereo projection is concerned with how to simulate human visual perception when rendering computer generated images. For the purpose two cameras are placed in a virtual scene to imitate the view points of the viewer's left and right eye. The resulting left and right images are often regarded as a stereo pair. The cameras are spatially displaced with a horizontal distance that should correspond to the user's eye separation. They should look in parallel directions and have a common projection plane (see Fig. 2), otherwise the resulting images lead to increased eye strain and uncomfortable feeling for the viewer. To achieve this an asymmetric frustum (also off-axis projection) has to be calculated for the two cameras.

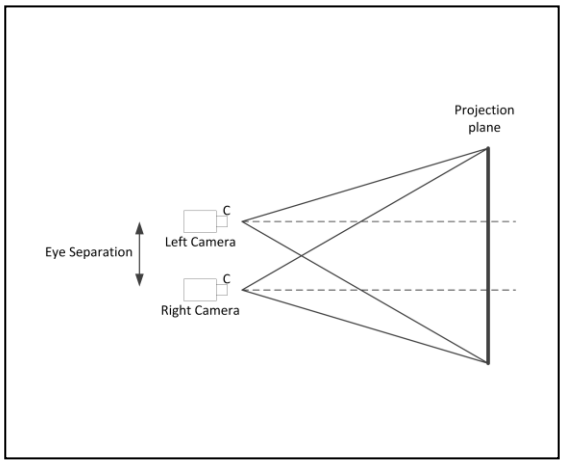

Fig. 2. Asymmetric (off-axis) stereo projection, redrawn from [3]

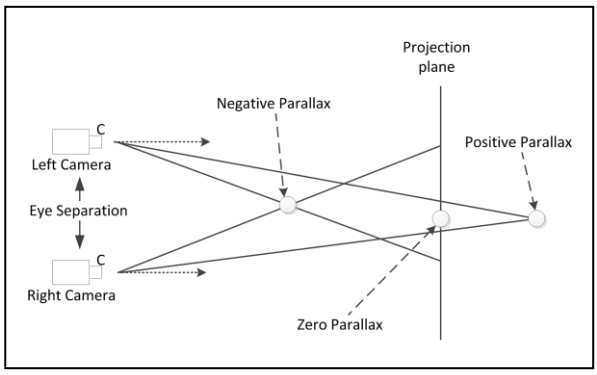

Fig. 3. Zero, positive and negative parallax, redrawn from [3]

Objects lying on the projection plane have zero parallax, objects in front of the projection plane – negative parallax and objects behind the projection plane – positive parallax. With the parallax being measured as the difference between the projection of the object on the projection plane as seen from the left and right viewpoints (see Fig. 3). Or in other words if the rays from the two viewpoints towards the object cross before they reach the projection plane – the object generates negative parallax, if they cross at the projection plane – zero parallax, and if they cross after the projection plane – positive parallax.

Respectively when the user views the images rendered by the two cameras objects having zero parallax will appear to be on the screen surface, objects with negative parallax – in front of the screen and objects with positive parallax – inside the screen.

More detailed description and explanation about camera setup for stereo projection can be found in [3] and [4].

C. Stereo Displays

Very important for the successful perception of stereo images is that the left image must be visible only for the left eye and the right image – only for the right eye respectively. Otherwise the viewer will see simply two flat overlapping images instead of one fused image with depth cues. There are different ways in which the stereo pair can be transmitted to the viewer's eyes separately:

- Anaglyph stereo – the simplest (and already somewhat outdated) type of stereo presentation using color filters. The two images are processed with complement colors (for example cyan-magenta) and the viewer wears glasses with the same colors.
- Head Mounted Displays (HMD) – the viewer wears a special device with two screens positioned directly in front of one of the viewer's eyes. Thus each eye can only see the image that is designated for it.
- Active stereo – the left and right images are displayed alternately on a screen with high refresh frequency. The viewer wears glasses (called active or also shutter glasses) that are synchronized with the screen so that when the left image is displayed the right eye is blocked by the glasses and vice versa.
- Passive stereo – the left and right images are presented to the user at the same time. In front of the source of each image (i.e. beamer) a light polarization filter is placed with different polarization direction. For example with horizontal polarization for the left and vertical polarization for the right image. The viewer wears respectively polarized glasses (called passive glasses) so that the horizontally polarized light coming from the source of the left image can only pass through the also horizontally polarized glass in front of the left eye and not through the vertically polarized glass in front of the right one.
- Auto-stereoscopic displays – these types of displays use different techniques (parallax barrier, lenticular

sheet, etc.) to direct the light from alternating pixel columns of a display so that when the viewer is positioned at specific places in respect to the screen his / her eyes will see the correct image, thus avoiding the need for glasses. A detailed description of different types of auto-stereoscopic displays and how they work can be found in [5].

D. Quad-Buffered Graphics Adapters

The majority of today's graphics adapters have two video buffers – front and back. The idea is that the display is showing the image in the front buffer while the application renders the next image in the back buffer. When the next image is ready the contents of the two buffers are swapped and the next image is shown onscreen. This ensures that the user always views at a valid image instead of observing the next image being rendered. The difference by quad-buffered graphics adapters is that instead of two there are four video buffers – a pair of front and back buffers for the left eye and another pair for the right one.

III. Generating Stereo-Pairs using Quad-Buffered Graphics Adapters

A. Mono Rendering with Two Video Buffers

When rendering with mono projection the image from a single camera viewpoint is rendered to the back buffer and when rendering has finished the front and back buffers are swapped. A diagram for mono projection is shown on Fig. 4. The view frustum is symmetrical along the view direction of the camera. It is defined by near and far distances, the field of view (FOV) and width to height ratio (aspect ratio). The view frustum defines the part of the virtual scene visible to the camera.

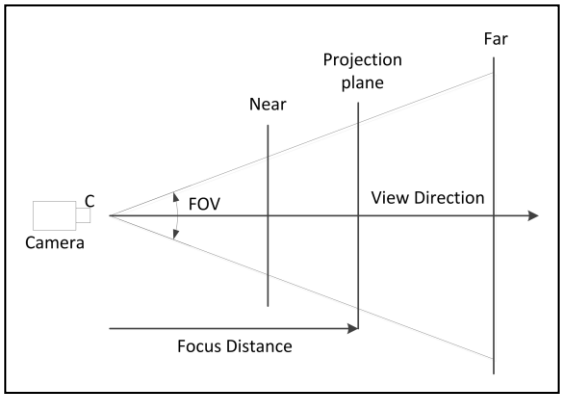

Fig. 4. Mono projection camera setup

Below sample OpenGL code follows for setting up the camera view frustum:

```
glDrawBuffer(GL_BACK);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glViewport(xpos, ypos, _width, _height);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
//gluPerspective(cam.getFOV(),cam.ratio,cam.getZNear(),
cam.getZFar());
double rad = ANG2RAD * cam.getFOV() / 2;
GLdouble top = tan(rad) * cam.getZNear();
GLdouble bottom = -top;
GLdouble left = bottom * cam.ratio;
GLdouble right = top * cam.ratio;
glFrustum(left,right,bottom,top,cam.getZNear(),cam.getZFar());
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(cam.getPos().x, cam.getPos().y, cam.getPos().z,
        cam.getTarget().x, cam.getTarget().y, cam.getTarget().z,
        cam.up.x, cam.up.y, cam.up.z);
/*
 ... draw scene ...
*/
GL_SwapBuffers();
```

The first two lines specify the active buffer to write into and clear its contents. After that a viewport is defined where the scene will be rendered. The *cam* variable is a camera object instance describing a viewpoint in the scene. The *glMatrixMode* function is used to define which matrix is currently adjusted – the projection or the modelview matrix. OpenGL provides two functions for setting up the view frustum – *gluPerspective* and *glFrustum*. The difference between the two is that *gluPerspective* defines always a symmetric view frustum, while the *glFrustum* can be used to define also asymmetric view frustums (which are necessary when performing stereo rendering). The top, bottom, left and right variables describing the view window are calculated for the near distance of the camera which places the projection plane on the near distance. The *gluLookAt* function defines the position and view direction of the camera. After that the actual rendering of the scene follows. When the render process is finished the result is displayed by swapping the front and back buffers.

B. Stereo Rendering with Four Buffers

When performing stereo rendering the scene has to be rendered twice – once from the viewpoint for the left eye and a second time from the viewpoint for the right eye. It is important that the view directions of the two cameras are parallel. Otherwise if they are converging to a common target point (the so called toe-in method) the resulting stereo images will be uncomfortable for the viewer causing eye strain (the resulting images will be as if the user keeps his or her eyes crossed all the time). The correct projection setup for the two cameras is the asymmetric view frustum, or off-axis projection.

On Fig. 5 the symmetric frustums of the left and right cameras are given with black dashed lines. The cameras are positioned at a horizontal distance equal to the eye separation. Some considerations about the actual eye separation value and

its relation to the screen size are given in [4]. The eye separation value should be adjustable as different users can perceive different parallax amounts comfortably. The actual image that has to be displayed onscreen is where the two symmetric frustums overlap on the projection plane. To achieve this the actual view frustums of the two cameras have to be offset to receive the asymmetric frustums shown with red solid lines.

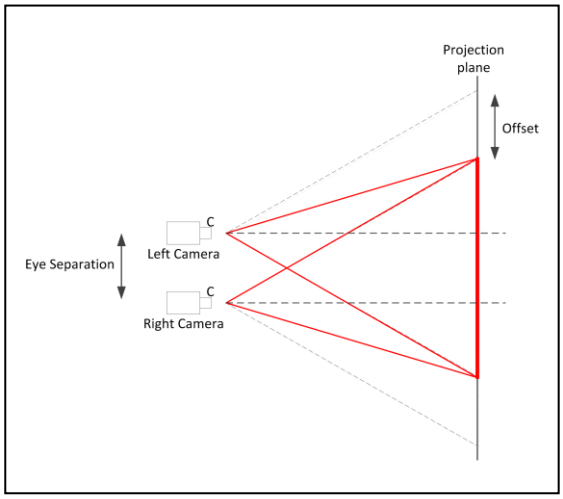

Fig. 5. Asymmetric view frustum

Sample OpenGL code follows for setting up the asymmetric camera view frustums for the left and the right cameras. Some of the lines are skipped when they are the same as the code for the symmetric frustum from the previous section.

```
vec_right = cam.X * cam.eyesep / 2.0; //cam.X is a unit vector = view x up
//top and bottom the same as by symmetric frustum
glDrawBuffer(GL_BACK_LEFT); //draw in left back buffer
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
//offset left and right of the view window
left  = - ratio * top + 0.5 * cam.eyesep * cam.getZNear() / cam.getFocus();
right =  ratio * top + 0.5 * cam.eyesep * cam.getZNear()/ cam.getFocus();
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(left, right, bottom, top, cam.getZNear(), cam.getZFar());
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(cam.getPos().x – vec_right.x, cam.getPos().y – vec_right.y,
        cam.getPos().z – vec_right.z, cam.getTarget().x – vec_right.x,
        cam.getTarget().y - right.y, cam.getTarget().z – vec_right.z,
        cam.up.x, cam.up.y, cam.up.z);
/*
 ... draw scene from left view ...
*/
glDrawBuffer(GL_BACK_RIGHT); //draw in right back buffer
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
//offset left and right
left  = - ratio * top - 0.5 * cam.eyesep * cam.getZNear() / cam.getFocus();
right =  ratio * top - 0.5 * cam.eyesep * cam.getZNear()/ cam.getFocus();
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(left, right, bottom, top, cam.getZNear(), cam.getZFar());
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(cam.getPos().x + vec_right.x, cam.getPos().y + vec_right.y,
        cam.getPos().z + vec_right.z, cam.getTarget().x + vec_right.x,
        cam.getTarget().y + vec_right.y, cam.getTarget().z + vec_right.z,
        cam.up.x, cam.up.y, cam.up.z);
/*
 ... draw scene from right view ...
*/
GL_SwapBuffers();
```

Notice that this time the scene is first rendered in the left back buffer and then once again in the right back buffer. The front and back buffers (both left and right) are swapped only after both left and right images are ready. In order to keep camera view directions parallel both the camera positions and camera directions are moved left and right with half the eye separation from the original camera position by the call to *gluLookAt*. The other important difference from the symmetric frustum is the coefficient with which the left and right window definitions are multiplied – this is the offset that makes the frustum asymmetric and unifies the view window of the two cameras.

Some considerations – as the resulting images are saved in different places and the information about the scene is only being read during the render process the rendering of the stereo pair images can be seen as two separate processes that can be easily parallelized. Because the rendering of the two images can be independently often two computers are used so that one renders the left image and the other the right. In such a case sometimes these computers are equipped with quad-buffered video cards which is unnecessary as each computer actually outputs only one image. It is better that the computers be equipped with a higher class dual-buffer video card.

IV. Connecting Stereo Displays to Quad-Buffered Graphics Adapters

There are multiple ways in which the stereo images rendered with a quad-buffered video card can be displayed in stereo. Depending on available or affordable hardware and software and especially the display systems a suitable

configuration can be chosen. Bellow some examples for stereo display systems using quad-buffered graphics adapters follow.

A. *Easy Solutions*

An example of an "easy" solution is using the NVidia 3D Vision driver. It can automatically generate stereo pairs for any visualization application which uses the Direct3D graphics library. Unfortunately this doesn't apply for OpenGL used by the majority of science directed stereo visualization applications. But for quad-buffered graphics adapters the NVidia 3D Vision driver can take the output written by the OpenGL application in the left and right buffers and forward it in the necessary format to the output. Then the 3D Vision driver and hardware can be used to synchronize a 3D Vision ready display connected to the output with 3D Vision shutter glasses. Fig. 6 shows an example with a 3D Vision ready beamer connected to an NVidia Quadro graphics adapter. The 3D Vision infra-red emitter synchronizes the image from the beamer with NVidia's shutter glasses.

Fig. 6. An NVidia 3D Vision ready beamer showing stereo image

B. *Third Party Drivers*

This part covers setups by which a third party driver is used to transfer the stereo pairs in an appropriate format to the graphics adapter output. Examples for such drivers are the NVidia Quadro video card drivers and the iZ3D drivers. In this case the application writes the stereo pairs to the video buffers, the third party driver reads them from there and converts them to the appropriate format – for example side-by-side, top-bottom, frame sequential, etc. (see [6] for more information), and forwards them to the graphics adapter output. The format should correspond to a format that can be read by the display connected to the graphics adapter output.

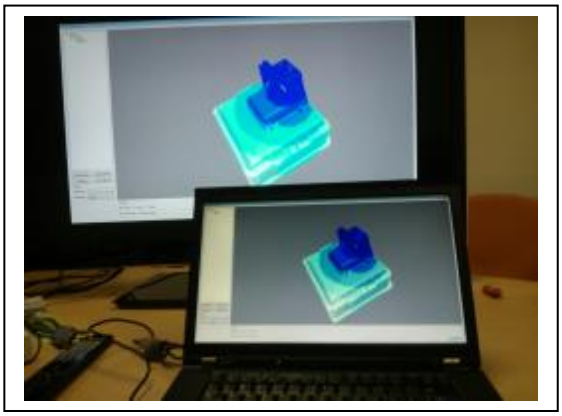

Fig. 7. A laptop with NVidia Quadro video card connected to a 3D TV display

Fig. 7 shows an example of a laptop with an NVidia Quadro graphics adapter connected to a 3D TV Display. The NVidia Quadro driver is setup to deliver a frame sequential signal which can be interpreted by the 3D TV and displayed in stereo. On Fig. 8 the necessary settings for the NVidia Quadro drivers can be seen. The *Generic active stereo* format corresponds to a frame sequential (also time-multiplex) ordering of the left and right stereo images. In this case to connect the 3D TV to the laptop also a DP (Display Port) to HDMI (High-Definition Multimedia Interface) adapter was needed. More information about the exact setup can be found in [7]. Such configuration is very suitable for creating mobile virtual reality systems.

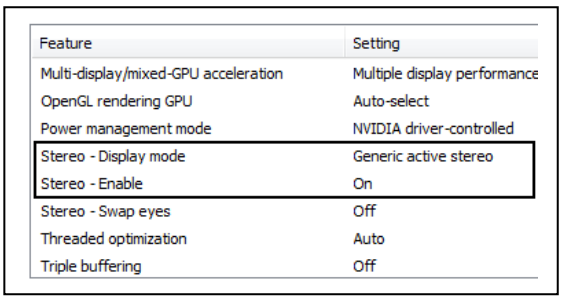

Fig. 8. Settings in the Manage 3D Settings tab of the NVidia Quadro Control Panel to connect to a 3D capable display in frame sequantial mode

C. *Using Dual Outputs*

Quad-buffered video cards usually also have two or more outputs. For example two DVI outputs or two Display Port outputs as with some NVidia Quadro cards. This means that two displays can be connected to the outputs. This is a typical setup for example for a power-wall stereo projection system with two beamers where usually two computers render the images for the left and right eye and the stereo pairs are then projected by the two beamers on the projection wall. Instead of a cluster of two or three computers in this case a single computer with quad-buffered graphics adapter can be used where the two beamers are connected to the two outputs of the graphics adapter.

Table C.3 List of Supported Stereo Display Modes (continued)

Option	Description	Hardware Examples
nView Clone Mode	Uses projectors from two displays in nView Clone mode-left image on one display, right image on the other. Passive polarized filters (glasses) isolate the left and right images to the corresponding eyes of the viewer.	Dep3D System

Fig. 9. Excerpt from NVidia Control Panel Quick Start Quide

Fig. 9 is an excerpt from the NVidia Control Panel Quick Start Guide showing that nView Clone Mode can be used to achieve the necessary result. Fig. 10 shows two beamers with passive polarization filters connected to the outputs of an NVidia Quadro video card. To connect the beamers two DP to DVI (Digital Video Interface) adapters were needed.

Fig. 10. Two beamers with passive polarization filters connected to the outputs of a quad-buffered graphics adapter

At the end Fig. 11 shows the resulting stereo pair projected on a power-wall.

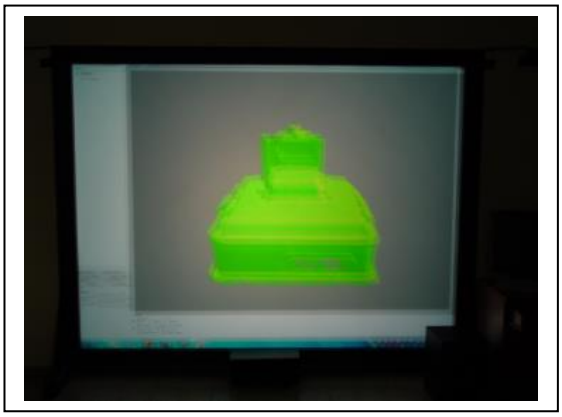

Fig. 11. Stereo image projected by the beamers on the power-wall

v. Conclusion

This paper provides a necessary foundation for readers that are interested in stereo visualization and virtual reality. It explains the basics, gives examples, points out differences and directs the reader to further useful literature with practical orientation.

The different visualization systems described at the end show some of the benefits of using quad-buffered graphics adapters for stereo visualization which in general are:

- Easy to setup;
- Suitable for creating portable or mobile visualization systems.

References

[1] N. Qian, “Binocular Disparity and the Perception of Depth”, Neuron, Vol. 18, Cell Press, 1997, pp. 359-368

[2] J. Behr, D. Reiners, “Class notes: don't be a WIMP: (http://www.not-for-wimps.org)”, ACM SIGGRAPH 2008 classes, ACM, USA, 2008

[3] P. Bourke, P. Morse, “Stereoscopy, Theory and Practice”, Workshop at the International Conference on Virtual Systems and Multimedia VSMM’07, Brisbane, Australia, 2007

[4] S. Gateau, D. Filion, “Stereoscopic 3D Demystified: From Theory to Implementation in STARCRAFT II”, NVIDIA GDC 2011

[5] N. A. Dogdson, “Autostereoscopic 3D Displays”, Computer , vol.38, no.8, 2005, pp.31-36

[6] R. Piroddi, “Stereoscopic 3D Technologies”, Snell Ltd. April 2010

[7] Chotrov D., Maleshkov S., “Methods and Tools for Design and Implementation of an Affordable Mobile Virtual Reality System for Solving Engineering Problems”, Proceedings of Technical University of Sofia, Volume 62, Issue 4, 2012, pp. 161 – 168