Volume Rendering in an Optical Tracking based Virtual Environment

Balázs Domonkos*
Attila Egri†

Department of Control Engineering and Information Technology
Budapest University of Technology and Economics
Budapest / Hungary

Abstract

In the last five-eight years the three dimensional displaying come into focus again. When it is amended by a tracking system it is called *mixed reality*. This paper presents a mixed reality environment (its hardware and software elements) developed by us on the Budapest University of Technology and Economics. Our system uses a dual projector stereo display for three dimensional visualization and a computer-vision based tracking system. The hardware configuration consists of two projectors, polar filters, video cameras, cheap passive coloured markers and an average PC.

Volumetric data visualization in a virtual environment has several benefits. It decreases the depth information loss caused by the $3D \rightarrow 2D$ projection, tracked tools can affect many properties of the visualization, that can be set or modify using the classic keyboard, mouse input in a difficult way.

The final goal of this project is to improve this virtual environment with general tracking therefore it can be used for augmented reality applications.

Keywords: augmented reality, computer vision-based tracking, real-time volume rendering

1 Introduction

Mixed Reality (MR) systems require tracking of the user's head movement and in many applications other object tracking can be useful. The word "registration" means allocation of the position of real and virtual objects in a common coordinate frame. The problem of registration are liable to occur in all mixed reality technologies both in Virtual Reality (VR) and in *Augmented Reality* (AR).

AR brings the existing physical space and some virtual space into one common space in real-time. It differs from virtual reality basically in not replacing, but extending the user's environment. The tasks of the two technologies are similar, but while in the case of virtual reality, the real challenge is to create the most realistic visualization possible, for augmented reality the accuracy of the registration matters. For example a virtual "cup" have to be displayed *on* not above or under the real "table" in order to ensure believable virtual environment.

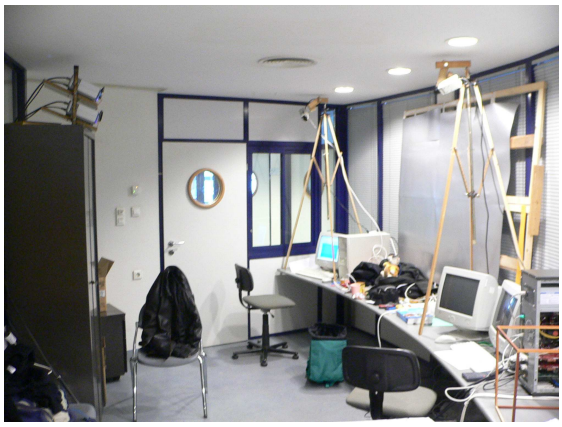

Figure 1: Test system configuration for cvbARlib

The goal of our work is to develop a system realizing a general purpose AR environment on a simple hardware – can be used for visualization large volumetric data sets in a projected AR environment.

The hardware configuration of an AR system is made of three parts [2]:

visualization: according to the principle of the 3D visualization the images for the two eyes are produced separately. In our realization this means a pair of projectors mounted with circular polar filters and suitable spectacles [19, 18].

registration: realized by computer vision-based tracking of passive coloured markers (pieces of high-coloured paper, ping-pong balls or "Kinder" eggs). Reconstruction is done using stereo vision. Therefore two cameras are used.

computing power: we used a desktop PC with video capture card and two video cards.

*domonkos.balazs@lanten.hu
†vregath11@gmail.com

From the software view, our goal is to develop a *general purpose programming library* (*cvbARlib*: *C*omputer *V*ision *B*ased *A*ugmented *R*eality *Lib*rary[1]). This library is responsible for all functionalities that need to be provided in a projected AR system.

There are numerous benefits of applying such AR technology in visualization of large amount of volumetric data (highly accurate 3D medical images, images created in geology, industrial radiology, scientific simulation data):

- The stereo rendering technique is appropriate for decreasing the depth information loss of the three dimensional data caused by the $3D \rightarrow 2D$ projection. [8]
- Due to the registration, one can walk around the displayed objects, a natural, perceptual interface is provided for setting the external geometric parameters of the visualization.
- Cheap passive coloured markers can be attached to any real objects affecting the volumetric data (user's hand, real tools). With these tools light sources, slicing planes and other properties of visualization can be defined and modified in a perceptual way.

Certainly the above mentioned system has disadvantages according to the classic, monitor based volume visualization:

- Stereo rendering doubles the rendering time.
- Tracking increases the computation time, and can make delays in the system.
- Computer-vision algorithms are usually not robust and accurate.

The next section summarizes briefly the system and its metrics. It is followed by the detailed descriptions of the hardware and the software elements. After that we introduce the volume rendering application. At the end of the paper the results are presented and the conclusions are summarized.

2 Our Virtual Environment

The following metrics are set up for our system:

- the system must be able to update the displayed image at a rate of 20-25 frames per second to sustain the illusion of reality.
- The system must respond to the user's movement in near real-time.
- The displayed virtual objects must have correct real sizes, specified position and orientation.

[1] http://www.cvbarlib.net

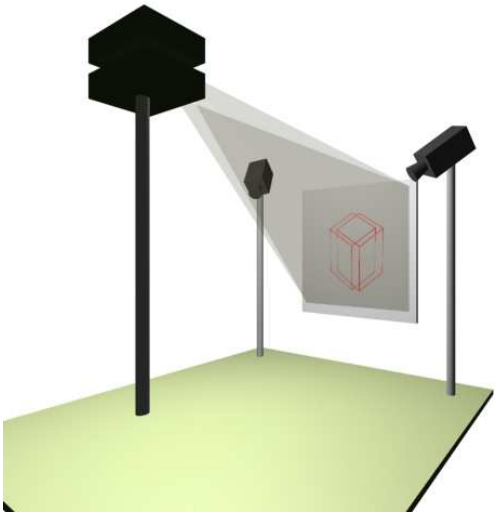

Figure 2: Physical system design draft

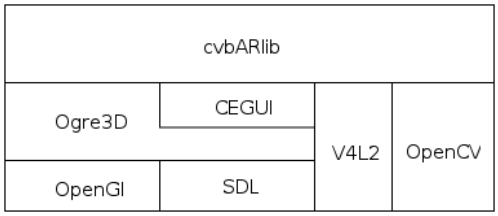

Figure 3: Libaries used by cvbARlib

For visualization, two projectors, polar filters and a front projectable canvas are used [19, 18]. The displayed images are generated by the *DLP (digital light processing)* projectors. DLP technology is necessary because the LCDs have in-built polar filters which would interfere with the external filters. The projected images are separated by two *circular polar filter pairs*. One of the pair is mounted in front of the lens of the projector the other one is assembled into the glasses worn by the user. The canvas has to preserve the polarity of the light and its amplifying characteristic has to be almost direction-independent. We found a low-cost paperboard to assemble a test system that satisfies the specified requirements.

The registration subsystem is based on stereo vision techniques, it uses two cameras and passive coloured markers. The user is monitored by two cameras, head position value are calculated using the camera images. This head position is used for rendering the appropriate images for the two eyes. These images are displayed by the projectors and are separated by the polar filters 2.

From software view implementation of the visualization and the registration is a challenge, however there are some additional tasks to perform. A world model is required for containing attributes of real and virtual objects. The cameras and the canvas have to be calibrated before the system is used. For comfortable usage of the system a graphical user interface is needed.

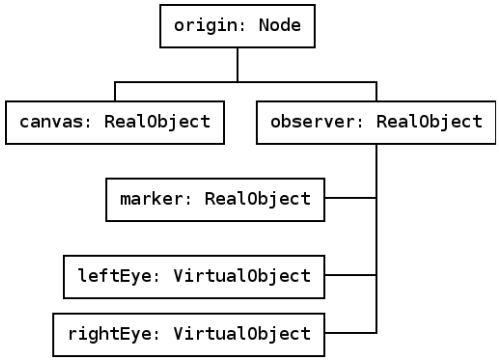

Figure 4: cvbARlib sample world model

Our library creates a uniform layer on the top of the libraries shown in Figure 3. The following task have to be accomplished:

1. world model,
2. registration,
3. visualization,
4. user interface,
5. calibration.

In the next sub-sections we discuss these tasks in detail.

2.1 World modell

The *world model* of cvbARlib suitable for defining geometry and optical attributes of both real and virtual objects is simplified from the scene graph of Ogre3D[2] [16] game engine. This framework defines a common metalevel on the top of the DirectX and OpenGl rendering engines and the Win32 and X windowing systems.

The `Node` class gives a node of the scene graph. The geometric and optical attributes of the inherited classes (`RealObject`, `VirtualObject`) are described separately by `Mesh` and `Material` classes.

The advantage of this model lays in its generality and flexibility: the real and virtual objects can be freely attached to each other (the tracked coloured markers move the model of the user which moves the virtual cameras used in rendering, see Figure 4). The only exception in hybrid structure creation is attaching real objects to virtual ones because the constraints are affected between the two worlds in only one direction.

Ideally whole augmented world is *known and displayed*: the virtual objects are displayed by the rendering system, their position and orientation are known; the locations of real objects are determined by the tracking system, and their rendering is done by the reality.

[2]Object Oriented Graphics Rendering Engine

2.2 Visualization

There are lots of possibilities to display objects in three dimensions. These solutions differ in technology, size, indoor or outdoor usage, the number of concurrent users etc. One of these technologies is the *Head Mounted Display* (HMD). It can be used for both VR and AR environments (*optical* and *video see-through* technologies). Its advantages are the good resolution and the nice generated depth cue. Since the HMD is mounted on the user's head, there are no limits in augmentation of the environment, since the rendering canvas is set near the user's eye. Therefore the position and orientation of the canvas (namely the lens of glasses) are changing during the user's head movement. Thus the good position and orientation tracking is very important in this technique [2].

Another technology is the *holo-TV*. This is a relatively new device delivered by the Hungarian Holografika Ltd. [15]. It visualizes a calculated holography from a voxel-addressable framebuffer. Since practically every ray is displayed for all pixels and all directions, head tracking is unnecessary and the number of users is unlimited in this technology. But the width, height and depth sizes of the three dimensional image is limited by the screen size, the graphics memory and the bandwidth.

The *projected VE technologies* are located between the HMDs and the holo-TV. Here orientation tracking is unneeded since the canvas is fixed, as at the holo-TV, but only two images are calculated for only one user just like at the HMDs. The "window to the virtual world" is limited by the fixed canvas, but depth is almost unlimited (only limited by the largest disparity can be rendered). Since the image is projected one or more meters screen diameter can be reached easily. It is a noticeable benefit, that only position tracking is needed, because orientation is usually calculated from positions, therefore errors in position can cause huge error in orientation.

In addition, our hardware configuration has two CRT displays too for displaying two dimensional information (GUI, camera images, debug information) and for conventional user input. The device-independent display used by cvbARlib is based on X Windowing system features (XF86, Xorg) accessible on most Unix-based systems.

There are two technologies for assembling multi-display systems:

1. The *TwinView* [20] enables joining two video outputs into one logical X screen on dual-display cards.
2. The *Xinerama* [21] merges two different X servers (even on different GPUs) into one logical X server.

The disadvantage of the TwinView according to the Xinerama is the limit of the connectable video outputs, but it has many advantages:

- It uses one X screen, the *n*Vidia driver hides the multi-display hardware-specific information from the X server which sees one screen.

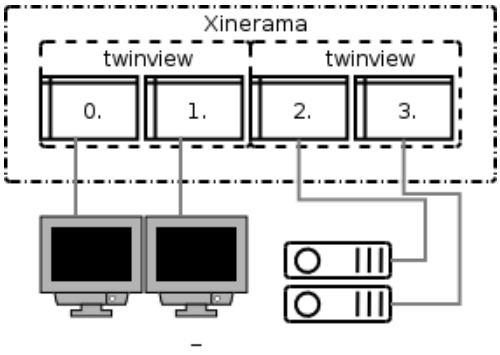

Figure 5: X server configuration in cvbARlib

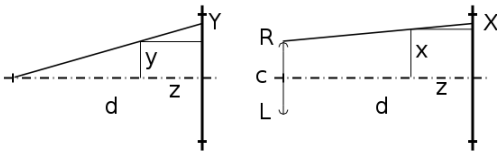

Figure 6: Parallel axis asymmetric frustum perspective projection

- All graphic devices use one frame buffer and accordingly all functionality of the single display appear under TwinView too.
- Emulating one logical screen does not make any noticeable software overhead.

The empirical X server configuration of cvbARlib for maximal TwinView exploiting is shown in Figure 5. The common type devices are connected to the same graphics cards under TwinView; the TwinViews are merged into one logical screen by Xinerama on the top level.

For the proper three dimensional rendering we use the *parallel axis asymmetric frustum perspective projection* (PAAFPP) [4, 1].

The PAAFPP calculates the X,Y projected coordinates by the following formulas.

For left camera:

$$X = \frac{(x+c/2)d}{d-z} - c/2, \quad (1)$$

$$Y = \frac{yd}{d-z} \quad (2)$$

For right camera:

$$X = \frac{(x-c/2)d}{d-z} + c/2, \quad (3)$$

$$Y = \frac{yd}{d-z} \quad (4)$$

Here (x,y,z) is the three dimensional coordinates of the point, (X,Y) is the projected point, d is the distance between the camera and the canvas and c is the eye offset (see Figure 6).

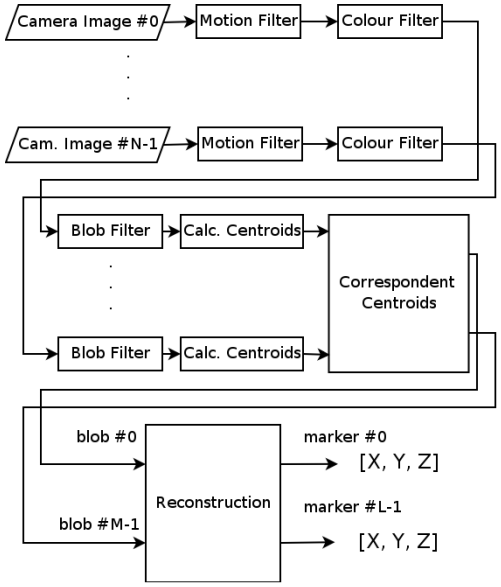

Figure 7: Tracking pipeline

The stereo rendering of augmented world mentioned above and tracking of real objects (markers on the user) is achieved in a parallel way and real-time ensuring continuous registration and three dimensional experience.

Our rendering engine uses the modified Ogre3D metarenderer. To implement the PAAFPP method, we added some methods to the Ogre3D's `Camera` class to overwrite the camera matrix directly with the new values calculated by the PAAFPP (similar to the `glFrustum` OpenGl function).

The logical screen of the windowing system is handled by the `Screen` class, the physical devices are represented by different `ViewPorts` (`ViewPort2D` for two dimensional user interface and `ViewPort3D` for projection). The offsets and sizes of these viewports can be set for each one. The `Canvas` has references to the received `ViewPort3D`s, it sets the viewport attributes at projector calibration.

2.3 Registration

We use computer-vision based algorithms for interest of metrically registered system. Knowing the accurate position of the canvas, the user and the encroaching units are very important both for proper 3D visualization and possible user interaction.

The tracking system of cvbARlib uses coloured markers have to be fixed on the tracked objects. Now the head tracking is implemented, but the number of the tracked objects is theoretically and algorithmically unlimited in our solution. The whole tracking pipeline is shown in Figure 7.

First, moving pixels are searched in the camera images (*Motion Filter*). There is a simple way to find the contours of moving areas with background estimation using three equal-sized images (background, last and current image):

1. Computing two differences:
   - BCDiff: between background and current image and
   - LCDiff: between last and current image.
2. Calculating the new background image: a pixel is determined to be a background pixel when is does not change in the last x images, that means LCDiff is less than threshold Δ_1 y times ($y <= x$).
3. Finding moving pixels: when BCDiff is greater than threshold Δ_2.

The algorithm has four tunable parameters: x, y, Δ_1 and Δ_2. The background is learned during the motion detection. As the texture of objects are often homogeneous, small motions can change the colour values of the inner pixels slightly, the above steps usually find only the contours of moving areas. Accordingly moving area (a line) can be assumed between two pixels marked as moving and the algorithm can be amended in the following way:

When a moving pixel is found, a temporary list is created to which further pixels are appended until another moving pixel is found or the margin is reached. When another moving pixel is found, the pixels in the temporary list are marked as moving pixels, otherwise the list is dropped.

The *Colour Filter* algorithm is based on the hue value uses the grey world algorithm [3] to estimate the lighting. The output of the colour filter is processed by the *Blob Filter*, which generates marker-coloured continuous pixel sets (*blobs*).

With this *centroids of corresponding blobs* have to be associated. More than one same marker-coloured centroids in the same image can raise a correspondence problem. In this case, epipolar constraints can be applied (described for two cameras): an epipolar line can be computed from the two camera matrices and the position of blob center in the selected camera image. The nearest one is chosen from the blobs on the other camera image [12].

If the corresponding centroid pairs have been found, the three dimensional coordinates of the blob center can be calculated using a linear triangulation with SVD algorithm, otherwise the last found spatial coordinates are returned.

We use the functionalities of openCV [17] computer vision library to implement the tracking system. The results of these algorithms are shown in Figure 8.

2.4 Calibration

For the system to function properly, two calibration steps are needed. The tracking subsystem requires *metrical calibration of the cameras*, the correct 3D visualization needs *canvas calibration*. Since the cameras are used for canvas calibration, they need to be calibrated first.

Figure 8: Working tracking algorithm

Camera Calibration

Zhang's algorithm [11] is used for camera calibration. The advantage of this calibration algorithm is that it can be done by an inexpert user, (s)he only has to move a flat pattern in front of the cameras. The results of the calibration can be saved since the intrinsic parameters does not change frequently. Only the extrinsic parameters calibration is necessary in most cases, which needs only one image to be taken by each camera.

Canvas Calibration

The calibration of the canvas is necessary for a dual head stereo rendering system. The static binocular depth cue comes from the disparity[3] on the two projector images. A few pixel displacement causes disparity failure which leads to depth errors. Therefore the canvas calibration has two goals:

- to determine the distortion characteristic of projection,
- to determine the coordinate system and sizes of the screen, thus the projection metrics can be derived in pixels/meters resolution), thence the sizes of the projected objects can be set metrically.

During the calibration, the characteristic of the projection is determined by projecting running horizontal and upright lines on the canvas. One of the cameras, turned perpendicularly in front of the canvas, takes images simultaneously. The *largest usable area of the canvas (I)* and the *displacement map* of the projected images can be retrieved using the following algorithm (see Figure 9):

Let $P_i(u,v)$ be the projected pixel on the ith projector, $I_{cam}(x,y)$ is the corresponding pixel on the raster image of the largest usable area of the canvas (I) in the camera image and $I_{P_i}(x,y)$ in the image of the ith projector. From the

[3] the distance between the correspondent points on the images

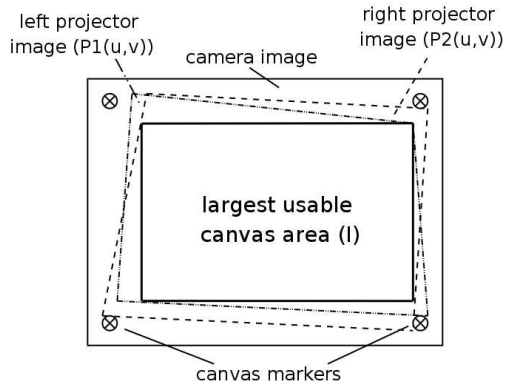

Figure 9: Canvas calibration

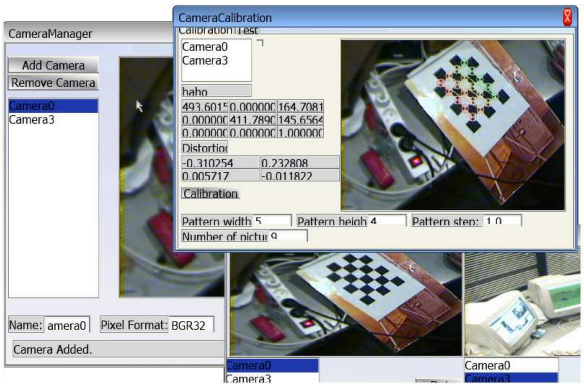

Figure 10: Two dimensional cvbARlib user interface

rectificated images[4] the projection function can be calculated for each projectors (Figure 9):

$$P_i(u,v) \quad \rightarrow \quad I_{cam}(x,y) \tag{5}$$

The projection of I, from the image of the projector to the camera image, can also be retrieved from the measured image pairs:

$$I_{P_i}(x',y') \quad \rightarrow \quad I_{cam}(x,y) \tag{6}$$

Inverting projection (5) using interpolation, the $I_{P_i}(x,y) \rightarrow P_i(u,v)$ projection can be retrieved which can be used as displacement pixel shader lookup table (stored in a texture) for each projector.

2.5 User interface

The main interface of the system is *perceptual*: the external parameters of the rendering virtual camera can be set by the user's movement. cvbARlib also provides *conventional two dimensional* user interface on `ViewPort2Ds` with keyboard and mouse inputs (see Figure 10). This functionality of our library is built on the CEGUI[5] GUI system [13]. The GUI consists hierarchical system of windows. The widget structure is defined in separate XML files while the event handlers are the descendants of the `GuiWindow` class.

The library contains in-built windows for the following tasks:

- camera management,
- camera calibration,
- canvas calibration,
- teaching marker colours,
- reconstruction debugging.

[4] undistorted camera images using the information of the camera matrix

[5] Crazy Eddie's GUI System

3 Volume rendering using cvbARlib

Direct volume rendering visualization technique is a method for directly displaying a sampled three dimensional scalar field (density distribution function) without any conversion steps (fitting geometric primitives to the samples, eg. using the Marching Cubes method [10]). Our application achieves *texture mapping (direct) volume rendering methods*. During texture mapping techniques the volumetric information is loaded into the texture memory of the graphics card, and the hardware realizes a mapping from the object (or viewport) space to the texture space.

Approximating the Direct Volume Rendering Integral (DVRI) [7], the composited shells can be represented by parallel polygons. The visualization process gives a projection between the texture space and the object space by setting the spatial and the texture coordinates of the polygon vertices.

The parallel polygons can be defined either in object or in viewport based coordinate system [5]. Better results can be reached using the viewport based approach, but it needs three dimensional texture support (today it is widespread). Using three dimensional textures also allows trilinear interpolation. Even better results can be achieved using vertex and pixel shader programs – in our implementation we use conventional alpha blending and filtering.

It should be remarked that the texture transfer speed is different for different texture dimensions (2D or 3D) and voxel types (8/16 bits intensity, 24 bits RGB, 32 bits RGBA). According to our benchmark we use I8 and RGBA32 colour modes.

High Resolution Volume

The following specification can be set up on an AR volume renderer:

- The displaying must be achieved in real-time frame rate (at the least 20 fps is required). If it is not possible, undersampling has to be done.
- The sizes of the projected internals should be real. It

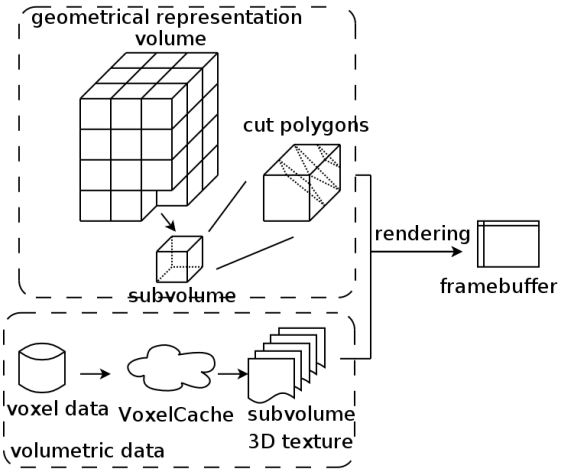

Figure 11: VoxelCache

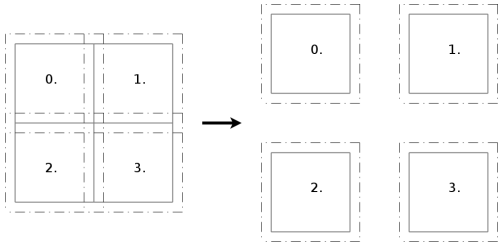

Figure 12: Sharing voxels among neighbour subvolumes

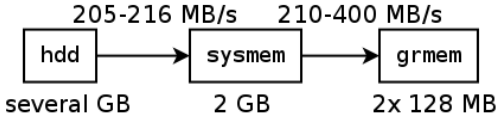

Figure 13: Bandwidths in our test system

can be reached using proper canvas calibration (see section 2.4)

Visualization of high resolution volumetric data raise bandwidth problems since the whole data (which can be one gigabyte or more) can not be stored in the texture memory. We did several benchmarks on our test system for determining the bandwidths between the hard drive, the system memory and the texture memory using optimized PNG compression[6] (see Figure 13). Considering the 400MByte/sec memory bandwidth swapping is not worth during a rendering pass.

There are at least three methods for handling this problem (from the simplest to the most complicated):

- undersample the whole data in order to fit into the video memory,
- divide the data into blocks and do swapping between the rendering passes,
- compress the data; hierarchical wavelet transformation is very suitable for this problem [6]

We used a hybrid method of the first and the second techniques inspired by Google (former *n*Vidia) Keyhole project [14]. Our algorithm (called *VoxelCache*, Figure 11) divides the whole Volume into *SubVolume* cubes with sizes of power of 2 using regular subdivision. The subvolumes are stored in different resolution (like three dimensional mipmaps) in a preprocessing step. The regular 2-powered subdivision has two benefits:

- the graphics card uses textures with dimensions of powers of 2,
- the regular decomposition makes the composition easier.

This method allows to mark subvolumes with too low or too high average voxel density as ”black” and ”white” blocks. These subvolumes will not be rendered or will be rendered without texture in the rendering pass.

The VoxelCache registers the subvolume instances:

- the *devices* where the subvolume is storeed: hard disk, system memory or graphics memory,
- the *resolution* of the instance.

Using this registry the algorithm manages the swapping scheduler using

- *inner strategy elements*: considering the capacity of the devices and the bandwidths between them,
- *outer strategy elements*: relative position of the subvolume in the user's frame which gives the required maximal spatial resolution of the subvolume.

Displaying the subvolumes is achieved using trilinear interpolation. To avoid artifacts, the neighbour subvolumes share the common voxels (Figure 12) [6]

4 Results

Our test hardware is an AMD Athlon64 3200+ with 2GB system memory, 2x 80 GB SATA-1 hard drives in RAID0[7], 2x *n*Vidia 6800 graphics cards with 1.0-7667 kernel module, IVC-200 video digitalizer card, 4x 800x600 video output devices (2x DLP projectors + 2x CRT monitors) in 3200x600 arrangement. The cvbAR-lib programming library with the AR volume renderer is implemented and experimented under the Debian Etch operating system with kernel 2.6.12.3 patched with Mingo Preemptive patch. The physical layout of the environment

[6] `http://pmt.sourceforge.net/pngcrush/index.html`

[7] 90-98 MB/s read bandwidth (205-216 MB/s can be reached taken into account the PNG compression)

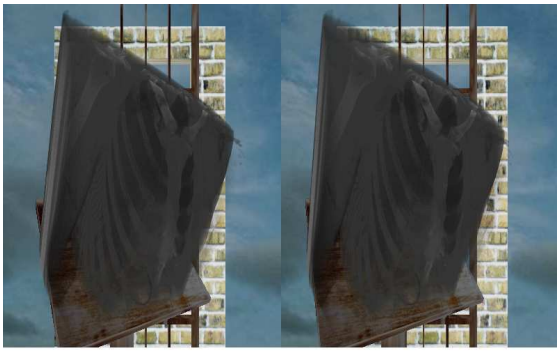

Figure 14: Human FrozenCT cadaver in our virtual environment, stereo image pair

Polygons per subvolume	Worst value	Average value	Best value
64	8.96	34.74	34.75
128	5.94	17.67	17.68
256	4.67	9.00	9.02
384	3.89	6.12	6.13

Table 1: The measured frame rates

is shown in Figure 1. The bandwidths in our test system are presented in Figure 13.

Our test data are The Visible Human Data Set [9] Male FrozenCT volume: decoded from GECT format to I8 and RGBA32 we get a volume with size 470 MB and 1880 MB. Our empirical subvolume size is 128^3 voxels. The subvolume data are organized into PNG images with dimensions $l \times l^2$ since the voxels order is the same as in in $l \times l \times l$ 3D textures. The minimal/maximal average voxel density limit is 1% / 99%.

The screenshot of the rendered images using simple alpha blending is shown in Figure 14, the alpha filtered rendering in Figure 15 (The fullscreen antialiasing filter (FSAA) of Ogre3D is disabled). Rendering and tracking frame rates can been seen in Table 1 and in Table 2.

To benchmark the tracking speed, 1000 image-pairs are taken and three different modes are defined: no operation (camera image is only grabbed), tracking without motion filter and full tracking. To determine the accuracy of tracking, the following simple benchmark test is used: the chessboard pattern for camera calibration is continuously moved in front of the camera-pair at the distance of approximately 1.5 meters (this is the average distance from the user). Three dimensional coordinates of two fixed chessboard corners (at the distance of 6 chessboard units) are calculated for 1000 image-pairs. The results of this test is shown in Table 3.

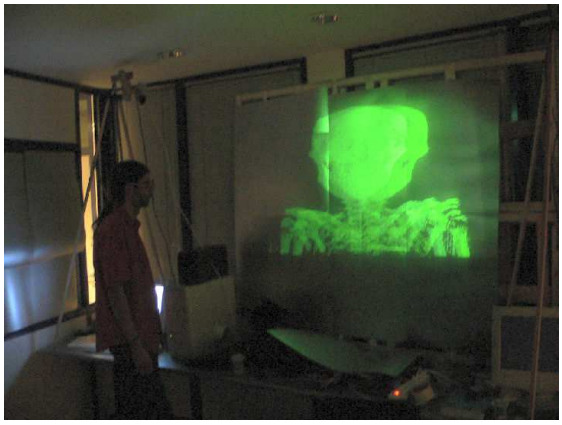

Figure 15: Stereo projection in working

Mode	Worst value	Average value	Best value	Std.dev
NOP	25.00	25.00	25.00	0.00
No Motion Filter	18.73	21.15	22.01	0.69
Full Tracking	16.65	19.63	20.77	0.92

Table 2: Tracking frame rates

5 Conclusion, future Work

Our goal is to create a hardware and software environment with which AR based applications can be developed, tested and run. Our Augmented Reality approach differs from classic ones using HMDs. It is an indoor, projected virtual environment, uses simple and light glasses, cheap tracking markers and we focus on metrical registration not just providing ”3D feeling”. Our environment has both advantages and disadvantages against the HMD and the holo-TV technologies. The system can be calibrated semi-automatically by pattern-recognition algorithms. All parts are commercially available at relatively low cost or as free software.

Now the user can walk around the projected objects, but we plan implementing gesture recognition algorithms in order to interact with the visualized objects. The bottleneck of the renderer is the hdd-sysmem bandwidth. For better performance we also plan implementing wavelet transformation.

6 Acknowledgement

This work has been supported by the National Office for Research and Technology (Hungary).

References

[1] Akka, Bob. Writing Stereoscopic Software for StereoGraphics(R) Systems Using Microsoft Win-

	Avg. dist. in ch. units	Avg. dist. in cms	Std.dev in cms
input	6	30.00	0.00
tracked (avg)	5.93	29.67	0.32
error	1.17 %	1.17 %	1.07 %

Table 3: Accuracy of the tracking

dows(R) OpenGl. 1998. http://www.stereographics.com/support/developers/pcsdk.htm.

[2] Azuma, R. T. A Survey of Augmented Reality. *Presence Cambridge Masachusetts*, pages 355–385, 1997.

[3] Barnard, Kobus. Modeling Scene Illumination Colour for Computer Vision and Image Reproduction: A survey of computational approaches. Submitted for partial fulfillment of the Ph.D. depth requirement in Computing Science at Simon Fraser University.

[4] Bourke, Paul. 3D Stereo Rendering Using OpenGL (and GLUT). 1999. http://astronomy.swin.edu.au/~pbourke/opengl/stereogl.

[5] G. Greiner and K. Engel and M. Bauer and T. Ertl. Interactive Volume Rendering on Standard PC Graphics Hardware Using Multi-Textures and Multi-Stage Rasterization. 2000.

[6] Guthe, Stefan and Wand, Michael and Gonser, Julius and Straßer, Wolfgang. Interactive Rendering of Large Volume Data Sets. 2002.

[7] Meißner, Michael and Huang, Jian and Bartz, Dirk and Mueller, Klaus and Crawfis, Roger. A Practical Evaluation of Popular Volume Rendering Algorithms. 2000.

[8] Mora, Benjamin and Ebert, David S. Instant volumetric understanding "order independent" volume rendering. 2004.

[9] The National Library of Medicine. *The Visible Human Project*. http://www.nlm.nih.gov/research/visible/visible_human.html.

[10] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169, 1987.

[11] Zhang, Zhengyou. A Flexible New Technique for Camera Calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence Pami*, pages 1330–1334, 1998.

[12] Zisserman, Andrew and Hartley, Richard. *Multiple View Geometry in Computer Vision (Second Edition)*. 2004.

[13] *CEGUI Crazie Eddie's GUI System*. http://www.cegui.org.uk.

[14] *Google keyhole*. http://www.keyhole.com/body.php?h=products&t=keyhole2NV.

[15] *Holografika Ltd.* http://holografika.com.

[16] *Ogre3D Object Oriented Rendering Engine*. http://ogre3d.org.

[17] *Open Source Computer Vision Library*. http://www.intel.com/technology/computing/opencv/index.htm.

[18] *The CAVE project*. http://cave.ncsa.uiuc.edu.

[19] *The Geowall Consortium*. http://geowall.geo.lsa.umich.edu.

[20] *TwinView Dual-Display Architecture*. http://www.nvidia.com/object/feature_twinview.html.

[21] *Xinerama, X Window System exension project page*. http://sourceforge.net/projects/xinerama.

All URLs are last visited at 2006-03-12.