7-2012

View-Dependent Rendering to Enhance Natural Perception for Augmented Reality Workstations

Rafael Radkowski
Iowa State University

James H. Oliver
Iowa State University, oliver@iastate.edu

View-Dependent Rendering to Enhance Natural Perception for Augmented Reality Workstations

Abstract

This paper presents a novel method for view-dependent rendering for monitor-based Augmented Reality (AR) applications that enhances natural visual perception. A typical monitor-based AR workstation incorporates a large monitor that acts as a window to the physical workspace in front of it. Although these workstations are often used in industrial AR applications, they unfortunately do not provide natural visual perception. This research focuses on the development of AR system and a novel method for real-time rendering of the augmented scene that incorporates the user's point of view. Thus, the user gets the impression of natural visual perception while moving in front of the monitor. This paper introduces the hardware setup, geometric registration to determine the correct view, and image processing. A sequence of images is presented that demonstrates the advantages of this method.

Disciplines

Computer-Aided Engineering and Design | Graphics and Human Computer Interfaces

Comments

This is a conference proceeding from *ASME 2012 11th Biennial Conference on Engineering Systems Design and Analysis* 1 (2012): 359, doi:10.1115/ESDA2012-82608. Posted with permission.

Proceedings of the ASME 2012 11th Biennial Conference on Engineering Systems Design and Analysis
ESDA2012
July 2-4, 2012, Nantes, France

ESDA2012-82608

VIEW-DEPENDENT RENDERING TO ENHANCE NATURAL PERCEPTION FOR AUGMENTED REALITY WORKSTATIONS

Rafael Radkowski
Heinz Nixdorf Institute
University of Paderborn
Paderborn, NRW, Germany

James Oliver
Virtual Reality Application Center
Iowa State University
Ames, Iowa, United States

ABSTRACT

This paper presents a novel method for view-dependent rendering for monitor-based Augmented Reality (AR) applications that enhances natural visual perception. A typical monitor-based AR workstation incorporates a large monitor that acts as a window to the physical workspace in front of it. Although these workstations are often used in industrial AR applications, they unfortunately do not provide natural visual perception. This research focuses on the development of AR system and a novel method for real-time rendering of the augmented scene that incorporates the user's point of view. Thus, the user gets the impression of natural visual perception while moving in front of the monitor. This paper introduces the hardware setup, geometric registration to determine the correct view, and image processing. A sequence of images is presented that demonstrates the advantages of this method.

Keywords: augmented reality, viewing device, view-dependent rendering

INTRODUCTION

Augmented Reality (AR) technology is a human-computer interaction technique that superimposes the natural visual perception of a user, generally provided with a video camera, with computer-generated information (i.e., 3D models, annotation, and texts). AR presents this information in a context-sensitive way to the user. Special viewing devices are necessary to use AR. A common viewing device is the so-called head mounted display (HMD). An HMD is a device similar to eyeglasses that use small displays instead of lenses.

AR systems generally support natural visual perception. An interface is described as natural if its technical realization is effectively veiled from the user [1]. With HMD-based AR natural visual perception is generally achieved with stereoscopic viewing integrated with head tracking to generate graphics view frusta that correspond exactly to the user's eye positions in the physical world. This enables virtual geometry to be rendered in exactly the same perspective as the video stream onto which it is superimposed. Humans perceive the world stereoscopically, which enables us to estimate distances and to accurately evaluate three-dimensional shapes, etc. Stereoscopic viewing in AR leads to similar advantages: distances and the size of virtual objects can be better estimated (see [2] [3] [4]).

In industrial applications that require a relatively large workspace and perhaps multiple users, a simpler AR implementation is often used. A typical industrial AR workstation is a large monitor that acts as a window to the physical workspace in front of the monitor [5] [6] [7]. In these applications, the video camera and the monitor are arranged back-to-back so that the user looks through the monitor onto the workspace as through a window. The computer-generated objects augment the retrieved video stream.

However, the current generation of AR workstations do not provide a natural visual perception. The typical setup captures monocular images from the physical workspace behind the monitor and the user is not tracked. The retrieved images remain static regardless of the viewer's position. Thus, the user loses one of the main advantages of AR applications.

An important visual perceptual cue is perspective viewing. View-dependent rendering can simulate this cue. View-dependent rendering is a method that, in general, adapts the perspective of the computer-generated information to the viewing position of the user. It emerges in the field of virtual reality and usually is based on head tracking. Different methods exist that facilitate the generation of a plausible view for responsive workspaces, projector-based applications, such as CAVEs, and so called "fish-tank" VR systems based on monitors. However, these VR approaches do not adapt the view of the physical environment, which in AR, typically come from video images.

This research develops a novel method for monitor-based AR based on view-dependent rendering that presents a the combined video/virtual AR image corresponding to the user's head position. Thus, the user sees a geometrically correct view when moving his/her head in front of the monitor. The paper is

structured as follows. The next section introduces the relevant related work. Section 3 describes the system setup. Section 4 explains the view-dependent rendering. Section 5 presents implementation details and the results. The paper closes with a summary and description of future research.

RELEATED WORK

The related work focuses on two fields: AR workstations and view-dependent rendering. The literature review shows that many basic methods exist. However, no view-dependent rendering methods for monitor-based AR workstations have been developed to date.

Augmented Reality Workstations

A representative example of a contemporary industrial AR system was introduced recently by Geißel et al [8]. The authors present an AR application for the review of 3D models within the context of a physical prototype. In general, the main view of the application shows a physical car (a section of it) via video. The video image of the physical car is superimposed by 3D models. Their system includes a monitor and a camera mounted on a pivot arm. The pivot arm measures the position and orientation of the camera, which enables the correct alignment of the 3D models relative to the objects captured in the video. The application facilitates the design review of the construction and aids during assembly training.

Similar setups are introduced in [5], [6], and [7]. In general, these AR workstations implement a monocular video stream on a monitor. The monitor screen acts as a window to the physical workspace and virtual objects are superimposed onto the scene. However, the combined video/virtual (AR) view remains static when the user moves in front of the monitor. Thus, the user looses the advantages of a natural viewing, such as better estimation of distances and sizes, which facilitate accurate pick and place operations as well as assembly operations [2] [3] [4]. In the case of design reviews and assembly training this capability may be critical.

Two conclusions can be made: 1) Typical industrial AR applications utilizes a monitor-based approach. However, many of the advantages of AR are lost when using a monitor. 2) Natural viewing enhances AR applications like assembly training applications. Today, this can be realized using HMDs only. This demonstrates the need for a natural visual interface for monitor-based AR workstations.

View-dependent rendering

View-dependent rendering is widely used in the field of computer graphics. Many different methods for large-scale models, virtual reality, animation, and network rendering have been introduced. For example, view-dependent rendering for VR involves tracking the user's head position in physical space, with offsets locating each eye position. The bounds of the VR display (bench top display surface, CAVE wall, or fish-tank monitor extents) are used with the eye position to construct a view frustum for the virtual scene. The next frame is rendered with a similar frustum created with the alternative eye position. The images rendered are presented to the appropriate eye in either a frame-sequentially fashion (with liquid crystal shutter glasses) or continuously (via polarization filters on projectors and glasses). See for example, [9][10][11][12].

However, these methods do not facilitate view-dependent rendering of AR scenes that are shown as video stream. This requires view-dependent rendering of images.

View-dependent rendering for AR applications can potentially be accomplished by adapting an image. The foundation of this is a method was first introduced by Debevec in 1998 [13], referred to as the view-dependent texture map, which is essentially a way to render different views of one scene. The approach from Debevec requires a 3D model of the real scene. The images of the scene are transformed to conform to the new point of view.

Heigl et al. present an enhanced approach that estimates a model automatically [14]. The estimated model is a depth map representing the real scene. As result, they are able to compute different views of a scene based only on one image.

In [15] the authors present a similar approach. They estimate a depth map by moving a binocular camera. Thus a depth map is estimated which is the basis for the view-dependent texture mapping.

Mori et al. [16] present an approach for so-called multi-view images. They use image-warping techniques to create different views of one image. These images are arranged with respect to their spatial position. Thus, a user can see the image from different positions. A similar method was introduced by Würmllin et al. [17]. Two comprehensive surveys of this field are presented in [18] and [19].

While these methods suggest a possible approach for AR, they are typically used for single images only. In addition, the methods do not facilitate spatial registration between real and virtual objects. Smolic [20] presents the first solution that works with video streams.

Bimber et al. [21] introduces the first projection-based AR application. They use a projector to augment a physical background with computer-generated images and thus developed a method for spatial registration. However, only the perspective of the virtual objects are adapted with respect to the background. The perception of the physical background does not need to be changed.

In summary: several methods exist that facilitate view-dependent rendering of virtual objects only, based on head tracking. In addition, several methods for view-dependent rendering of images also exist. These methods are based on view-dependent texture maps. However, most of the methods facilitate the adaption of single images only. The coherence and plausibility of the visual appearance in a sequence of images is not warranted. The AR application of Bimber superimposes a physical background and does not need a to adapt the physical background.

As a consequence, there is a need for a novel method that integrates two aspects of view-dependent rendering for AR applications: 1) the adaption of the AR scene shown in a video stream and 2) the registration between real and virtual objects.

SYSTEM SETUP OVERVIEW

This section introduces the hardware and software system developed to address these needs. First, the hardware setup is described. Afterwards, the software setup is introduced.

Hardware Setup

Figure 1 shows the hardware setup for the view-dependent rendering AR workstation. The main user working area is a table top. A monitor is mounted on one side of the table such that the angle between table and monitor is 50°, and an opening between table and monitor gives access to the working area. The opening from the table to the lower edge of the monitor measures 30cm. The monitor is a 40" Samsung Series 5 TV with a resolution of 1920 x 1080 dpi and a physical size of 88cm x 59cm. A 15cm reflective hemisphere is attached to the back of the monitor. The hemisphere reflects the entire environment, including the working area, and provides an omnidirectional image of the environment. It is mounted at the center of the monitor in order to reduce the distance between the center of the screen and the center of the sphere. A video camera stands on the opposite side of the table. In this implementation, a Creative Webcam Chat HD with a resolution of 1280 x 720 pixel (aspect ration 16:9) at 30 fps was used. The video camera captures images of the table top via the reflective sphere.

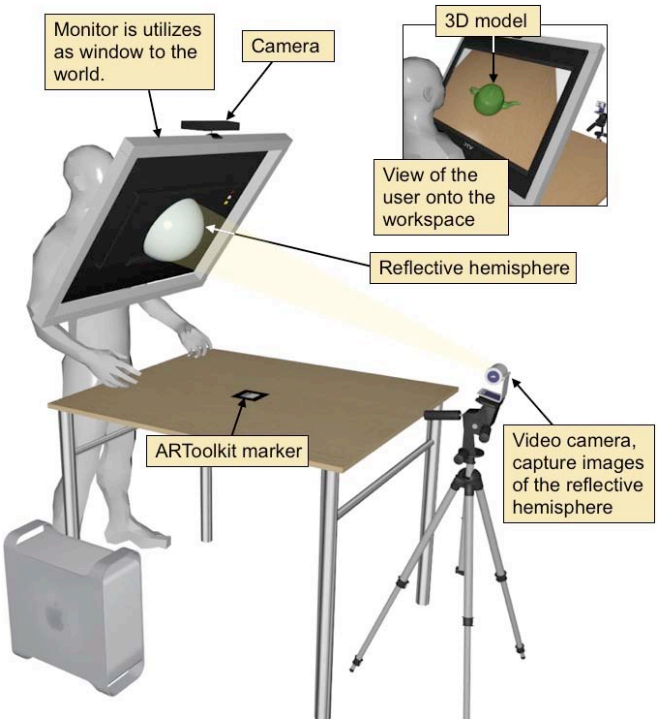


Figure 1. Overview of the hardware setup

A second video camera is attached on top of the monitor. It observes the user, particularly the movements of his/her head. It is also a Creative Webcam Chat HD with a resolution of 1280 x 720 pixel (aspect ration 16:9) at 30 fps. The entire setup works for one user only. The prototype system is implemented on a PC with a 3.5 GHz Intel Xeon processor, 6GB RAM and an NVIDIA Quadro 5000 graphics processing unit (GPU).

The position and orientation of all video cameras, the sphere, and the TV are aligned manually. As a consequence, the entire system cannot be moved after calibration. The calibration step is also carried out manually.

Software Setup

This subsection provides an overview of system software. Figure 2 shows the software architecture of the entire system, which consists of two major components: 1) an AR application and 2) a head tracking application.

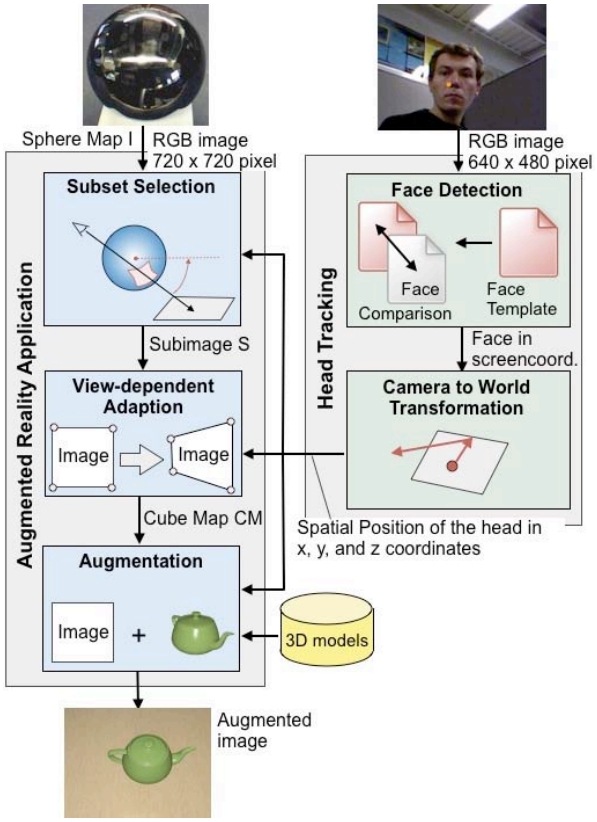


Figure 2. The software system of the application

Augmented Reality Application

The main task of the AR application is to retrieve images from the main camera, select a subset of this image corresponding to the users view point, adapt this subimage to apply proper perspective, and finally augment it with computer-generated information. Therefore, the software is separated into three modules: subset selection, view-dependent adaption, and augmentation. The three modules work in a sequential order. The output of the previous module is the input of the next module.

The processing of these three modules is referred to as view-dependent AR rendering and is described in detail in the following.

The task of the subset selection module is to select the

subset of the omnidirectional image reflected from the sphere that corresponds to the head position of the user. The spatial head position of the user and the image of the sphere are the input data. The sub-image is the output. This module is written in C++, the transformations are realized as GLSL shader code. The image management and auxiliary functions are based on OpenCV (http://opencv.willowgarage.com), an image-processing library.

The task of the view-dependent adaption module is to modify the retrieved subimage to fit to the required view perspective of the user and to the form factors of the screen. Therefore, it needs the head position of the user as input as well as the geometry of the hardware setup.

The next (augmentation) module applies the spatially correct augmentation of the 3D model with respect to the head position of the user and the form factors of the subimage. Therefore, it modifies the position and orientation of a virtual camera and the 3D models. ARToolkit, a computer vision-based tracking system written in C, is implemented for tracking [22]. For rendering, Open Scene Graph (OSG, www.openscenegraph.org) is used, a scene graph library that provides functions for the representation of 3D models and their rendering. Finally, this module provides the superimposed image as output.

Head Tracking

The head tracking application tracks the head of one user in front of the monitor and provides the position of the head P_{user} (Figure 3). It is subdivided into two modules: face detection and a camera-to-world transformation.

The face detection module is implemented with a tree-based technique: the so-called Haar classifier [23]. Since it is a state-of-the-art detector, it is only briefly summarized here. The head detector is a so-called supervised detector in that it needs training samples to be able to detect and track faces. These training samples are the source of reference images which are arranged in a tree. Each node of this tree represents a so-called rejection cascade; a test which checks whether an area of an input image is a head or not. The test is essentially a comparison: does an area of the input image meet the reference image? The nodes are ordered from the least to the most complex reference image. This minimizes the number of computations because many areas of an input image can be rejected (i.e., not part of the searched object) on an early cascade level. When an area passes all nodes, it is considered as a face. OpenCV provides a set of functions to train reference images and carry out the test. The module provides the position of the face in screen coordinates.

The camera-to-world transformation transfers the position of the head from image coordinates into Cartesian world coordinates. The center of the screen is the origin. This is done using a pinhole camera mode [24] (Figure 3):

$$P_{user} = K \cdot T_{camera} \cdot P_{user,screen} \quad (1)$$

The input is the position of the user's face in screen coordinates $P_{user,screen}$, T_{camera} describes the position of the video camera in world coordinates, and K is the projection matrix of the camera lens.

The head tracking applications operates at 30 frames per second and submits its data to the AR application using a network connection (UDP/IP).

VIEW-DEPENDED RENDERING

This section describes the method for view-dependent rendering. Figure 3 shows an overview of the geometric relations of the AR workstation. From left to right, the figure shows the user, the screen, the hemisphere, and the camera. The origin is the center of the hemisphere and the center of the monitor. Both positions are considered to be physically identical. The variables of Figure 3 are explained in the following subsections.

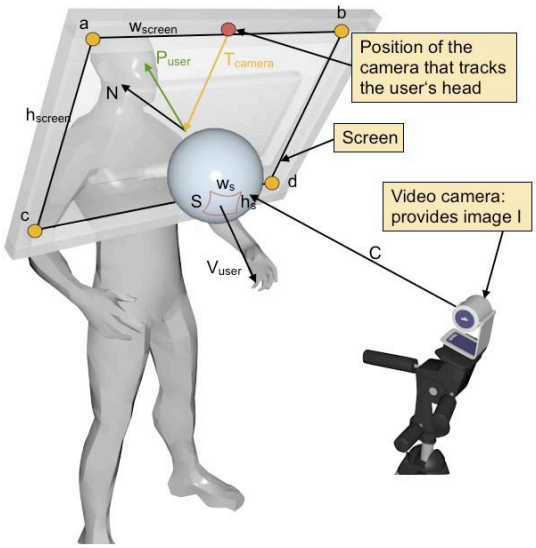


Figure 3. Geometrical relations of the setup

The input image for the view-dependent rendering is denoted as image *I*. It shows the reflection of the sphere. The output is a view-dependent image that is shown on screen and presents the working surface beneath the monitor in a correct perspective. It appears to the user as if looking through a window.

In order to realize this, three steps are necessary. First, a subset of the omnidirectional image *I* that corresponds to the user's view needs to be determined. Second, this image must be corrected and transformed into the correct perspective. The result is an image that shows the work space from the users's viewpoint. Third, a 3D model has to be superimposed onto this image. This requires its spatially correct rendering to facilitate a plausible looking scene. In the following, the method is described in this order.

Determination of the subimage

The objective is to determine the subset of the entire image *I*, which is retrieved from the video camera. The subset is denoted by *S(u, v)*, the entire image is denoted by *I(u, v)* where

u and v are the pixel coordinates. Since the image *I* represents an omnidirectional image of the environment a so-called sphere mapping [25] technique is necessary.

The following formulation assumes an input image that shows the sphere. In addition the center of the image corresponds to the center of the sphere. Thus, the influence of the distance between camera and sphere *C* and a possible displacement of the camera can be ignored.

First, the position of the center of the subimage *S* is determined, which is indicated by:

$$\mathrm{V}_{user} = -P_{user} \tag{1}$$

The vector V_{user} describes the viewing direction of the user with respect to the sphere. Input for this calculation is the head position of the user P_{user}. At this time, it is assumed that the head tracking application provides this vector. Eq. 1 assumes that the user is looking through the center of the screen.

Second, the size of the subimage *S* needs to be calculated. The size depends on the field of view of the user and the radius of the sphere. The horizontal field of view is calculated using the two vectors that indicate the corners of the screen and the scalar product:

$$\mathrm{a} = \mathrm{SC}_w - P_{user} \tag{2}$$

$$\mathrm{b} = \text{-}\ \mathrm{SC}_w - P_{user} \tag{3}$$

$$\mathrm{FoV}_h = a \circ b \tag{4}$$

with

$$\mathrm{SC}_w = \begin{pmatrix} \frac{w_{screen}}{2} \\ 0 \\ 0 \end{pmatrix} \tag{5}$$

The vertical field of view is calculated in a similar manner by using the vectors *a* and *c*.

Knowing the field of view, the size of the subimage *S* can be calculated by:

$$\mathrm{w}_s = 2 \cdot r \cdot sin(\tfrac{FoV_h}{2}) \tag{6}$$

$$\mathrm{h}_s = 2 \cdot r \cdot cos(\tfrac{FoV_v}{2}) \tag{7}$$

with w_s and h_s, the height and the width of the subimage *S* and *r*, the radius of the sphere.

This calculation defines the subimage as a spherical rectangle, which can be mapped to a corresponding planar rectangle via a spherical mapping technique. However, directly employing a general spherical mapping (topologically from a sphere to a cube) could result in a subimage rectangle that spans two faces of the cube, as shown in Figure 4a. Furthermore, spherical maps generally introduce image distortion, which reaches its maximum near the boundary of the spherical image. So positioning the subimage near the center of a cube face will minimize image distortion. Thus, an additional transformation is required before the sphere map is applied.

View-dependent Adaption

The view-dependent adaption undistorts the sphere map image (i.e., positions the spherical subimage so that the mapped image is centered on one cube face), retrieves the subimage *S*, and fits it to the form factors of the hardware setup. The basis for the entire transformation is a cube map and a sphere to cube map transformation according to Reinhard et al. [25]. A cube map represents a texture as a six-sided cube. Each side of the cube is aligned with one principal direction and shows an image with respect to its alignment (Figure 4a). The cube map is denoted by *CM*, where CM_i is one face of the cube map and $cm_{i,j} = (x, y, z)$ the corner *j* of face *i* in *x, y, z* coordinates. $CM_i(u_c, v_c, t_c)$ denotes the image with texture coordinates u_c, v_c, t_c of side *i*.

Three steps are necessary to retrieve the subimage *S*. First the cube map is rotated by a transformation that moves the subimage to one face of the cube map. Second the sphere map is transformed into the cube map and the subimage is retrieved. Third, the subimage is sheared to comply with the users current perspective and the hardware form factors.

Step1

In the first step the cube map is transformed by two rotations (Figure 4). These transformations will move the center of the front face of the cube map to the center of the subimage. This is carried out by:

$$CM' = R_x \cdot R_y \cdot CM \tag{8}$$

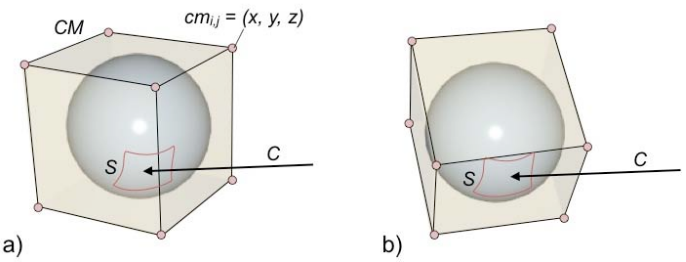


Figure 4. a) The cube map is transformed from its initial orientation to b) the center of the subimage S.

where R_x and R_y are two transformation matrices that rotate the image with respect to the head position of the user P_{user}.:

$$R_x(\varphi_h) = \begin{pmatrix} cos(\varphi_h) & -sin(\varphi_h) & 0 \\ sin(\varphi_h) & cos(\varphi_h) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{9}$$

$$R_y(\varphi_v) = \begin{pmatrix} cos(\varphi_v) & 0 & sin(\varphi_v) \\ 0 & 1 & 0 \\ -sin(\varphi_v) & 0 & cos(\varphi_v) \end{pmatrix} \tag{10}$$

with φ_h and φ_v, the horizontal and vertical angles between the screen and the screen normal *N* (Figure 5).

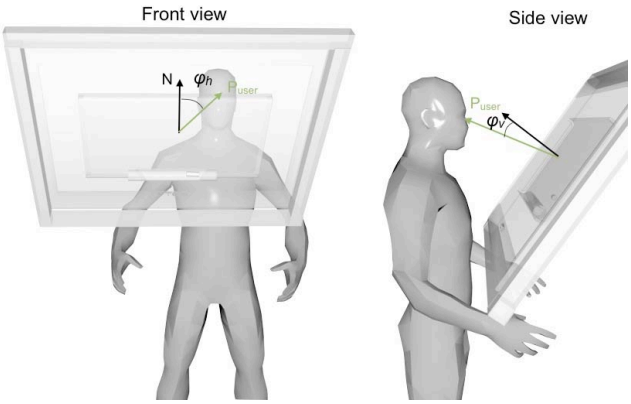


Figure 5. Position of the user in front of the screen.

Step 2

In the second step the sphere-to-cube map transformation is applied (Figure 6a). This transformation copies every pixel of the sphere map into a pixel of one side of the cube map. Therefore, the sphere-to-cube map transformation calculates for each pixel $CM_i(u_c, v_c, t_c)$ of each side of the cube map the related pixel $I(u,v)$ in the sphere map [25]:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} 0.5 + r \cdot u_c \\ 0.5 - r \cdot v_c \end{pmatrix} \tag{11}$$

$$r = \frac{sin(0.5 \cdot acos(-t_c))}{2.0 \cdot \sqrt{u_c^2 + v_c^2}} \tag{12}$$

The result of this transformation is the cube map. However, due to eq. 8, only one side of the cube map is necessary (Figure 6b). This side shows the relevant scene.

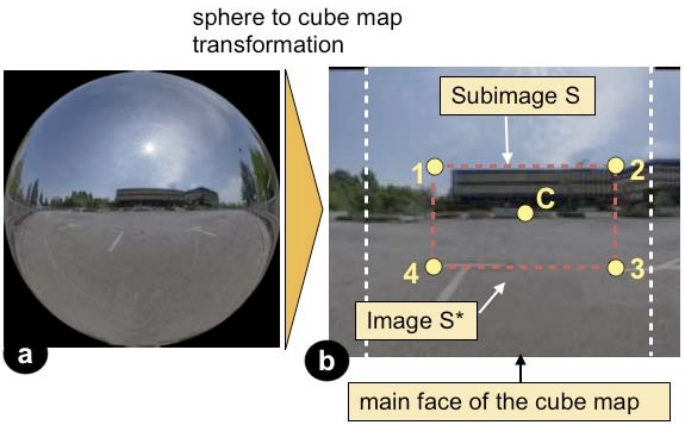


Figure 6. Sphere to cube map transformation and subimage copy.

Knowing each texture coordinate, the subimage S is retrieved per pixel by limiting the texture coordinates with respect to w_s and h_s. The boundaries are computed by applying the width w_s, the height h_s and the center to equation 11 and eq. 12. A world-to-texture coordinate transformation provides the center C and the boundaries 1 to 4 (Figure 6b). The texture coordinates are used to crop the image.

Prerequisite for this is to know the size of the physical scene, which is reflected by the sphere. This could be calculated. However, an initial manual calibration provides better results. This calibration is described in the next section.

Step 3

The objective of the third step is to transform the image so that it complies with the view of the user. When the user's head position P_{user} does not comply with the screen normal N the field of view must be sheared. The shearing increases with an increasing angle between N and P_{user}. To get a proper view onto the physical workspace, this shear is simulated by a perspective transformation that causes a tangential displacement of the image. Figure 7 shows this displacement. The left figure shows the nominal projective view when P_{user} aligns with N. Thus, the image is a rectangle. When the user moves his/her head to the left/right and top/bottom, a projective transformation corrects the view of the subimage S by adapting the texture coordinates:

$$\mathrm{P} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & d & 2d \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{13}$$

$$d = \frac{P_{user} \circ N}{|N|^2} \cdot N \tag{14}$$

$$\mathrm{P} = (\mathrm{P}_{user} \circ N) \cdot P \tag{15}$$

$$\mathrm{CM}_i'' = P \cdot CM_i' \tag{16}$$

where CM' and CM'' are the cube maps.

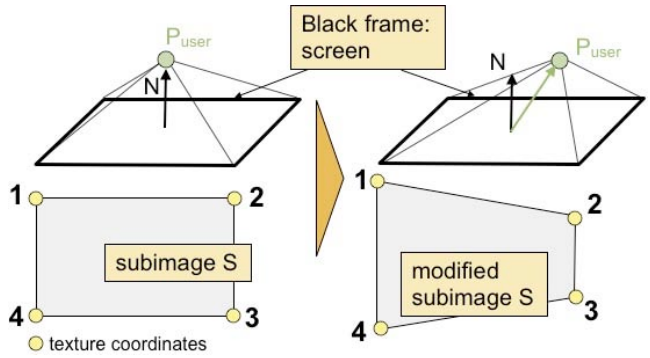


Figure 7. Perspective transformation with respect to the viewing position of the user.

The resulting image is mapped to camera coordinates to get a visual correct aligned image and then mapped to the form factors of the screen. The final rendering process generates a rectangular image again, which show the correct view. Finally this image is shown on the screen.

Augmentation

Spatially correct augmentation means that a virtual object must be superimposed onto the subimage S in a such a way that it is correctly aligned with respect to the physical objects that are shown in S. To ensure this, the subimage S itself must comply with the physical form factors of the monitor and conform to the perspective of the user's view onto the

workspace. Both requirements are fulfilled by the method described above.

Virtual objects are rendered by defining a virtual camera. Thus the definition of the virtual camera must conform to the view of the user, i.e., its position and orientation as well as its view frustum must be defined appropriately. Figure 8 shows a schematic representation of the virtual scene. The figure shows the virtual camera and a 3D model. The work space and the monitor screen are also indicated. The origin of the scene lies at the center of the screen. This origin also corresponds to the origin of the physical scene.

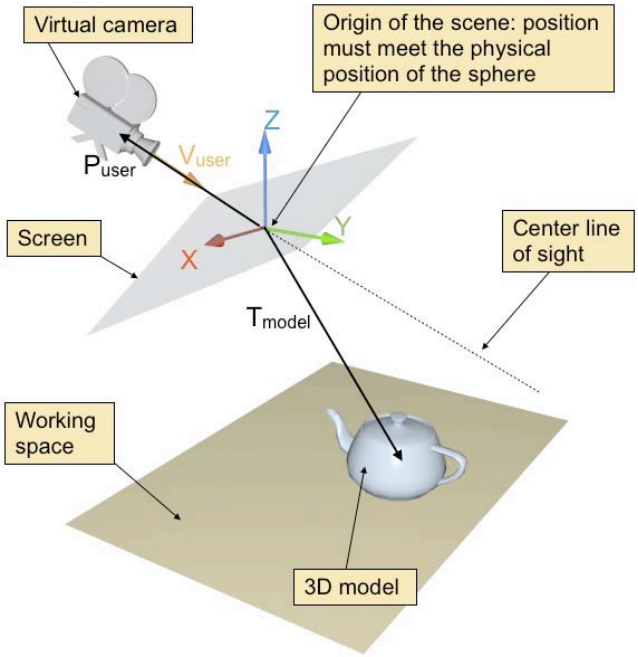


Figure 8. Virtual scene of the AR application

It is assumed that the user looks onto the physical workspace through the center of the physical sphere. Thus the virtual camera position corresponds to the head position of the user P_{user}, and the viewing direction of this virtual camera is V_{user}.

In addition to the viewing position, the field of view of the virtual camera is adapted with respect to the viewing position and the physical form factors of the screen. The field of view has already been determined in equation 4, the adaption is represented by the equations 13 through 15. This adaption is reused to define the proper parameters for virtual camera.

The position and orientation of the 3D model is specified using a common transformation matrix in homogeneous coordinates. This matrix is denoted as T_{model} (Figure 8). This matrix is determined by the ARToolkit in order to move and rotate the model on the workspace.

IMPLEMENTATION AND RESULTS

This section presents the application as well as the results. Figure 9 shows the prototype setup of the AR workbench, which has been used to generate the results. The hemisphere is attached on the back of monitor. The video camera stands on the opposite site on a tripod.

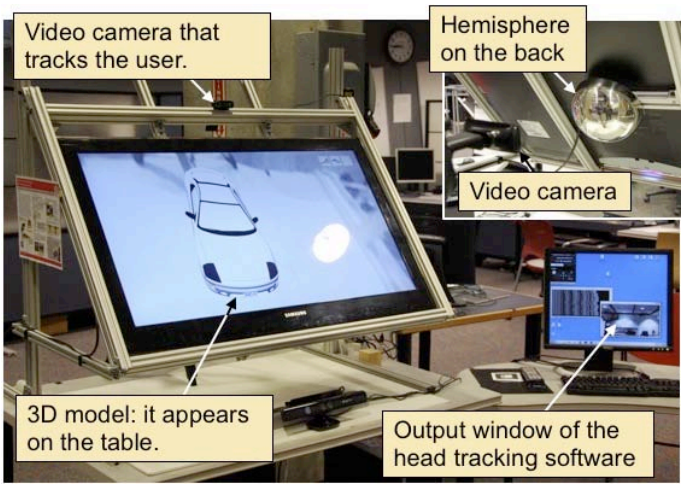


Figure 9. The prototypic setup of the AR workbench

The following section is separated into three subsections. The first subsection describes relevant implementation details. The second subsection presents the results. The results are discussed in the third subsection.

Implementation Details

The view dependent AR rendering method has been implemented as scene graph-based application. Figure 10 shows its simplified scene graph. The circles represent the nodes of the scene graph, the triangles the geometry nodes, and the convex boxes referenced node objects.

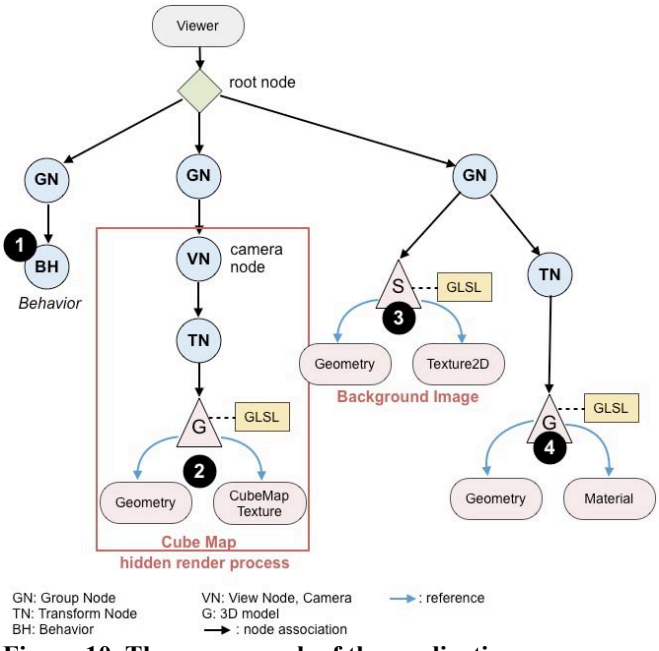


Figure 10. The scene graph of the application

The scene graph has three major branches. From the left to the right: the first branch is used to retrieve video images from the camera, the second branch contains the sphere-to-cube map transformation, and the third branch renders the background image and the 3D models. The application processes the scene graph from the left to the right. The numbers indicate the processing sequence.

To retrieve a camera image, a node with a callback function is used (a function that is called once per frame). The callback function keeps a reference of the camera and asks for one image per frame (1). The video camera process itself runs asynchronously in a separate thread and stores the images at a distinct position. The scene graph, in particular the callback node, retrieves the latest image frame-synchronously. Finally, this branch provides an image of the sphere map.

The second branch implements the subimage selection and the sphere-to-cube map transformation (2). The cube map is realized as texture. OSG provides a CubeMap class for that purpose. The subset selection and the transformation are implemented using OpenGL Shader Language (GLSL). Thus, everything is carried out during a render process. Therefore, the relevant side of the cube map is assigned to a geometry, a plane with four vertex points. Listing 1 shows the vertex program and Listing 2 shows the fragment program of the cube-to-sphere map transformation.

Listing 1: GLSL vertex program of the sphere-to-cube map transformation

```
uniform mat4 transformation;
void main()
{
        gl_FrontColor = gl_Color;
        gl_TexCoord[0] = gl_MultiTexCoord0;
        gl_Position = gl_Vertex;

        gl_TexCoord[0] =vec4(transformation
    * vec4 ( gl_TexCoord[0].xyz, 1.00) );
}
```

The variable *transformation* contains the rotation matrices of eq. 9 and eq. 10, and the last line of the code implements eq. 8.

Listing 2: GLSL fragment program of the sphere-to-cube map transformation

```
uniform sampler2D tex;
void main()
{
    vec3 cubeVectorNorm = normalize(gl_TexCoord[0].xyz);
    float r1 = sin ( 0.5 * acos(-cubeVectorNorm.z));
    float r2 = 2.0 * sqrt( pow(cubeVectorNorm.x, 2.0) +
        pow(cubeVectorNorm.y, 2.0) );
    float r = r1 / r2;
    vec2 v = vec2(0.5 + r * cubeVectorNorm.x,  0.5-r *
        cubeVectorNorm.y);
    gl_FragColor = texture2D(tex, v);
}
```

The texture *sampler2D tex* is the input sphere map. The code implements eq. 11 and eq. 12. The result is one face of the cube map. The whole branch is rendered as a hidden render step. The resulting image remains in the texture memory of the GPU.

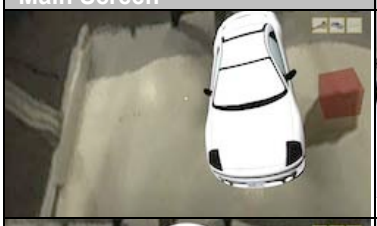
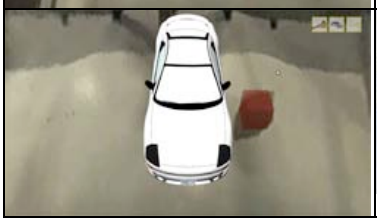
The third branch implements the rendering process of the background image and the rendering of the 3D models. The cube map image generated during the hidden render process is used as the background image. This image is assigned as a texture to a rectangle plane (3). Eq. 15 is used to determine the texture coordinates. Thus, the correct section of the image is shown on screen. At last, the 3D models are rendered (4).

Results

This subsection presents the results achieved using the prototypical setup. Table 1 presents a series of images from the main screen. The left column shows the images of the main screen, i.e., the AR scene the user sees when looking through the monitor onto the table. The right column presents the image of the head tracking software. This image appears mirror-inverted, but it captures the user from an opposite position. The values beneath this image are the position of the user's head, expressed by the angles φ_h and φ_v (Figure 5). The first three rows show a movement of the user from the left to the right. The next two rows show the view from a center-top and center-bottom position. The next picture shows the view from the upper right corner and the last picture the view from the lower right corner.

The AR scene shows a virtual car in a scale of 1:4 on the table. In addition to this virtual car, a physical cube is placed on top of the table. The size of the cube is 50 x 50 x 50 mm. This cube is used as a reference. Thus, a virtual red box with the same size is superimposed on the cube. When the user moves his/her head, the virtual cube should cover the real cube. To recognize a movement, the box is rendered semi-transparent.

Table 1: Series of screenshots that demonstrate the view of the user

Main Screen	Head Tracking
	
	

The images show that the method provides a view onto the table with respect to the user's head position. The view changes from the left to the right and from the top to the bottom. The entire application works in real time. The view to the table changes immediately when the user is moving.

In addition, a slight displacement between the red virtual cube and the physical cube can be observed. The images in Table 1 shows that the virtual and the physical cube fit well. Nevertheless, there is a slight displacement of the virtual cube: the edges of the virtual cube do not meet the edges of the physical cube exactly.

Discussion

The development of this AR workbench and the implementation of the method show, that view-dependent rendering is a viable solution to provide natural visual perception for monitor-based augmented reality. The entire view of the scene behaves the user expects. The image on the main screen shows the expected section of the working space. Thus, one finding of this work is the correctness of the presented method.

In addition, the utilization of a sphere and a sphere-to-cube map transformation fulfills the real time requirements. The implementation of the sphere-to-cube map transformation is fast enough to calculate 30 images per second. Thus, the view of the working area changes immediately. The user does not register a noticable delay.

At this time we assume that the user looks through the center of the screen to the table. This limits the effect and the usage of the AR workstation. However, this is not a limitation of the method rather than a limitation of the head tracking / face tracking system. It tracks the position of the face only.

One drawback is the relatively low resolution of the resulting image. Because the presented image is only a subset of the entire image captured by the camera, its resolution is lower than the that of the whole image. The current solution generates an image with a physical resolution of approx. 800 x 400 pixel. An image with that resolution, shown on a screen with a physical resolution of 1920 x 1080 pixel, results in a blurred image without sharp edges.

Furthermore, the subset selection works like a zoom. Image errors, noise, and other effects are also magnified. In particular camera noise is disruptive because it appears as constantly flickering snow. The only way to reduce this effect is to use a high quality camera with a low signal-to-noise ratio.

SUMMARY AND FUTURE WORK

This paper presents a method for view-dependent rendering for AR workstation. The method adapts the combined video/virtual image to the viewpoint of the user. The novel solution is to use a hemisphere and to select a subset of this hemisphere with respect to the position of the user's head. The mathematical model, as well as implementation details are described. The AR workbench and the method have been realized and tested. In summary, results indicate that the method facilitates view-dependent AR rendering that provides a plausible image of the workspace. This novel method can enhance natural visual perception for monitor-based AR workstations. This work focuses in particular on enhancing the perception of one visual cue for natural viewing: perspective.

Further enhancements will focus on the simulation of additional visual cues: depth cues. The current solution shows a flat image, without any depth cues. The proper perspective facilitates estimation of the size of a virtual object. However, additional visual cues may be helpful. For example, it may be possible to estimate the distance between the camera and the objects on the table using an optical flow approach. The calculated depth values could be used to simulate visual cues like the perspective of single objects (rather than adapting the perspective of the entire scene), shadows, and motion parallax.

Finally the entire system will be tested by a group of users. Comparing human performance using this system compared to text or 2D graphical instructions for a relatively detailed task such as manual assembly may expose the benefits of enhanced visual-spatial perception.

In addition, future work will strive to address the problems described above, such as the slight displacement of the 3D

model and the low resolution. The displacement offset can be corrected by a non-affine image transformation. This transformation will change the viewpoint of the user. In addition, the calibration procedure will be enhanced. At this time, the camera, the monitor, the sphere, and the workplace are measured manually. The dimensions are used as parameters in the software. This will be replaced by a template-based calibration procedure. A template with a known geometry will be placed on the table. A set of images of this template will be used to calibrate the entire system accurately and automatically. Finally, to address low resolution a second camera can be added that captures the more of hemisphere so that both images are merged onto one sphere map.

REFERENCES

[1] J. Blake, "*The natural user interface revolution*," in *Natural User Interfaces in .NET*, no. 1, 2011, pp. 1–33.

[2] Neumann U. and Majoros A., 1998, "*Cognitive, performance, and systems issues for augmented reality applications in manufacturing and maintenance,*" in Proceedings of the IEEE Virtual Reality Annual International Symposium, 1998, pp. 4–11.

[3] Tang A., Owen C., Biocca F., and Mou W., 2003, "*Comparative effectiveness of augmented reality in object assembly,*" in Proceedings Conference on Human factors in computing systems (CHI '03), Ft. Lauderdale, FL, USA, April 05 - 10, 2003, pp. 73-80

[4] Biocca F., Tang A., Lamas D., Gregg J., Brady R., and Gai P., 2001, *"How do users organize virtual tools around their body in immersive virtual and augmented environments?: An exploratory study of egocentric spatial mapping of virtual tools in the mobile infosphere,"* Technical Report, Media Interface and Network Design Labs, Michigan State University, 2001.

[5] Katzenbach A. and Haasis S., 2008, "*Virtual and Mixed Reality in a SOA Based Engineering Environment,"* Design Synthesis, CIRP Design Conference, April, 7-9, 2008, p. 29.

[6] Wittke M., 2007, *"AR in der PKW-Entwicklung bei Volkswagen,"* IFF Wissenschaftstage - Virtual Reality und Augmented Reality zum Planen, Testen und Betreiben technischer Systeme, Magdeburg, pp. 51–56.

[7] Geißel O., Longhitano L., and Katzenbach A., 2008, *"Operativer Einsatz von Mixed Reality Technologien,"* in Augmented & Virtual Reality in der Produktentstehung. 7. Paderborner Workshop Augmented & Virtual Reality in der Produktentstehung. 5.-6. Juni 2008, Paderborn, pp. 175–189

[8] Geißel O., Longhitano L., Katzenbach A., and Resch M., 2010, *"Automotive Mixed Mock-up – Eine neue Entwicklungsplattform der Automobilindustrie,"* in IFF Wissenschaftstage - Digitales Engineering und Virtuelle Techniken zum Planen, Testen und Betreiben technischer Systeme, 15.-17. Juni 2010, Magdeburg, pp. 41–46.

[9] Ugur B., Sahiner A. V., and Fidaner I. B., 2009, *"Off-Axis Stereo Projection and Head Tracking for a Horizontal Display,"* in Proc. of Conference in Advances in Computer Entertainment Technology ACE'09, October 29–31, 2009, Athens, Greece, pp. 1–4.

[10] Grey J., 2002, *"Human-computer interaction in life drawing, a fine artist's perspective,"* in Proc. of the Sixth International Conference on Information Visualisation, July 10-12, 2002, London, UK, pp. 761–770.

[11] Krueger W., Bohn C., Froehlich B., Schueth H., Strauss W., and Wesche G., 1994, *"The Responsive Workbench,"* in IEEE Computer Graphics and Applications, 14(3), pp. 1–14 IEEE Computer Society Press, Los Alamitos, CA, USA.

[12] Garstka J. and Peters G., 2011, *"View-dependent 3D Projection using Depth-Image-based Head Tracking,"* IEEE (Eds.) Proceedings of the 8th IEEE International Workshop on Projector-Camera Systems (PROCAMS) CVPR 2011 Colorado Springs, 22-25. 6. 2011, pp. 52–58.

[13] Debevec P. and Yu Y., 1998, *"Efficient view-dependent image-based rendering with projective texture-mapping,"* in 9th Eurographics Rendering Workshop, Vienna, Austria, June 1998, pp. 105-116

[14] Heigl B., Koch R., Pollefeys M., Danzler J., and Van Gool L., 1999, *"Plenoptic Modeling and Rendering from Image Sequences taken by a Hand-Held Camera,"* Proceeding of Mustererkennung 1999, 21. DAGM-Symposium, Springer Publishing, London, pp. 94-101.

[15] Evers-Senne J.-F. and Koch R., 2003, *"Image Based Interactive Rendering with View Dependent Geometry,"* Computer Graphics Forum, Volume 22, Issue 3, pp. 573–582, Sep. 2003

[16] Mori Y., Fukushima N., Fujii T., and Tanimoto M., 2008, *"View Generation with 3D Warping Using Depth Information for FTV,"* in 3DTV Conference: The True Vision - Capture, Transmission and Display of 3D Video, 2008, May 28-30, 2008, Istanbul, Turkey, pp. 229–232.

[17] Würmllin S., Lamboray E., Waschbüsch M., Kaufmann P., Smolic A., and Gross M., 2005, *"Image-space Free-viewpoint Video,"* in Proceedings of Vision, Modeling, and Visualization (VMV) 2005, Erlangen, Germany, November 16–18, 2005, Erlangen, Germany, pp. 665-673

[18] Zhang C., 2004, *"A survey on image-based rendering—representation, sampling and compression,"* in Signal Processing: Image Communication, Vol. 19, No. 1. (January 2004), pp. 1-28

[19] Shum H. and Kang, S.B., 2000, *"A review of image-based rendering techniques,"* IEEE/SPIE Visual Comm-unications and Imaging.

[20] Smolic A., Mueller K., Merkle P., Rein T., Kautzner M., Eisert P., and Wiegand T., 2004, *"Free viewpoint video extraction, representation, coding, and rendering,"* International Conference on Image Processing, 2004, ICIP '04. 2004, Singapore, Oct 24-27, 2004 pp. 3287–3290.

[21] Bimber O., Wetzstein G., Emmerling A., and Nitschke C., 2005, *"Enabling view-dependent stereoscopic projection in real environments,"* Proceedings on Fourth IEEE and ACM Int. Symposium on Mixed and Augmented Reality, 2005., 2-5 November 2004, Arlington, USA. pp. 14–23.

[22] H. Kato and Billinghurst, M., 1999, *"Marker Tracking and HMD Calibration for a Video-based Augmented Reality Conferencing System,"* in 2nd International Workshop on Augmented Reality, October 20-21, 1999, San Francisco, USA, p. 10

[23] R. Lienhart and J. Maydt, 2002, *"An Extended Set of Haar-like Features for Rapid Object Detection,"* IEEE Int. Conf. on Image Processing (ICIP 2002), Rochester, NY, Sep. 22-25, 2002, pp. 900–903.

[24] Szeliski, R, 2010, *"Computer Vision"*. Springer Verlag, Berlin, 2010

[25] Reinhard E., Ward G., Pattanaik S., and Debevec P.,*"High Dynamic Range Imaging",* Morgan Kaufmann, 2004

To all Patrons:

This article will be purged in 14 days. *If you need pages re-sent, please notify us within this time frame. Otherwise, we will have to relocate the item and you may be charged again.*

*****

Pictures, graphs, and charts may not maintain optimal resolution after scanning. Please call 216-752-0300, or email, if hardcopy is needed.

Thank you

The Research Investment, Inc.
Phone: (216) 752-0300
Fax: (216) 752-0330
E-mail: orders@researchinvest.com