MR Platform: A Basic Body on Which Mixed Reality Applications Are Built

Shinji Uchiyama, Kazuki Takemoto, Kiyohide Satoh, Hiroyuki Yamamoto,
and Hideyuki Tamura
MR Systems Laboratory, Canon Inc.
2-2-1 Nakane, Meguro-ku, Tokyo, 152-0031 Japan
{uchiyama.shinji, takemoto.kazuki, sato.kiyohide, yamamoto.hiroyuki125}@canon.co.jp, HideyTamura@acm.org

Abstract

This paper describes a platform package, called "MR Platform," which we have been implementing for research and development of augmented reality technology and applications. This package includes a parallax-less stereo video see-through HMD and a software development kit (SDK) for a Linux PC environment. The SDK is composed of a C++ class library for making runtime MR applications and related utilities such as a camera calibration tool. By using the SDK, the following functions are available; capturing video, handling a six degree-of-freedom (DOF) sensor, image processing such as color detection, estimating head position and orientation, displaying the real world image, and calibrating sensor placement and camera parameters of two cameras mounted on the HMD.

1. Introduction

New technologies are making substantial progress to seamlessly merge virtual and real worlds into a mixed reality (MR) space [1]. Recent advances in mixed reality, especially in augmented reality (AR), are summarized in [2]. Although it is expected that this kind of technology will be successfully applied to industry, entertainment, and various other fields, there are presently almost no products intended for use in an MR system and very few toolkits, such as ARToolKit [12], available to research and prototype such products. In order to construct such a system, various processing such as real-time video capturing, high-speed image processing, sensor handling, and virtual world handling management must be realized efficiently. This poses great obstacles for constructing a practical applied system.

We participated in the Key Technology Research Project on Mixed Reality (MR Project) in Japan [3]. Through this project of 4 years and 3 months, we constructed MR technology by developing head mounted displays (HMDs) specifically designed for AR [4], studying registration methods [5][6] merging real and virtual worlds seamlessly, and also developing practical AR systems, such as "RV-Border Guards [7], " "Clear and Present Car [8], " and "2001: An MR-Space Odyssey [9]." Since the MR project was completed in March 2001, a platform environment for research and development of MR systems has been implemented based on our achievements in the project. This package is called "*MR Platform*" and is expected to be available to support and promote R&D activities in the field of MR/AR.

This paper describes the configuration and features of this platform, and details the software structure and functions mainly from the view of application construction.

2. MR Application

Various types of MR/AR applications are studied by many researchers. Some of them utilize handheld displays as display units, some use projection type displays. As to the tracking device, six degree-of-freedom (DOF) tracking sensors are adopted by many systems, but some utilize vision-based methods that do not require any physical tracking devices. In order to design *MR Platform*, a hardware and software environment of MR application should

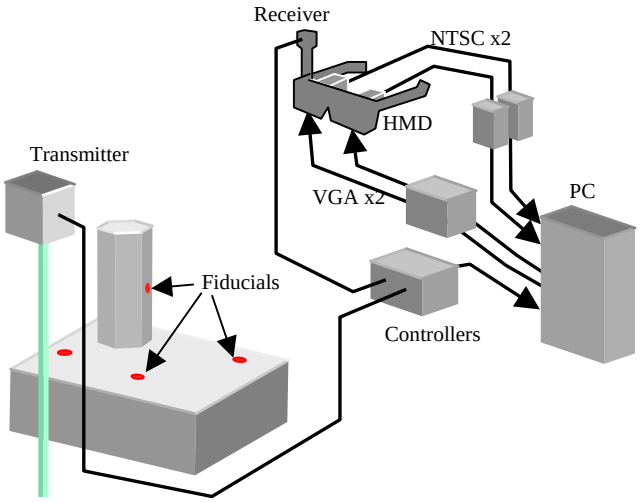


Figure 1. Typical hardware setup of MR/AR system

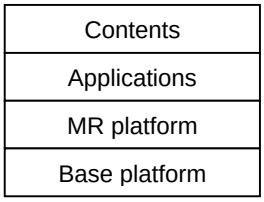


Figure 2. Four layers of MR application

be clarified.

As to the hardware environment, a typical configuration shown in Fig. 1 is assumed. In this configuration, a video see-through HMD (ST-HMD) is utilized as a display unit, and a 6-DOF tracking sensor attached to the HMD tracks the head position and orientation of the user. The application program running on the PC realizes a registration, which matches the real and virtual world coordinates, based on the position and orientation obtained by the sensor and image information captured by a camera attached on the HMD. It then renders images based on the registered position and orientation. The program should do all this process in real-time.

As to the software environment, each MR system consists of 4 layers: the base platform layer, the MR platform layer, the application layer, and the contents layer as shown in Fig. 2. The base platform layer is the lowest layer of the system, and consists of a computer, graphic and video hardware, other related devices, an operating system and system level libraries. The MR platform layer provides equipment and functions necessary for providing services to support the MR system. The video ST-HMD and software libraries such as registration, video mixture, even graphics are categorized in this layer. These two layers are independent of applications and content. The application layer is a software program implemented on the MR platform layer. The program may be designed for a specific application field, such as a shooting game, but is independent of its content. The contents layer is data used in the application program.

3. Configuration of "MR Platform"

Under the hardware and software environment described in the last section, "*MR Platform*" provides equipment and functions unique to the MR application. It consists of a video ST-HMD with a software development kit (SDK) corresponding to the MR platform layer.

As the base platform layer, an i386 based PC and Linux operating system are adopted because this type of environment competes successfully with graphics workstations (GWS) such as those provided by SGI. This is a consequence of technological advances, including:

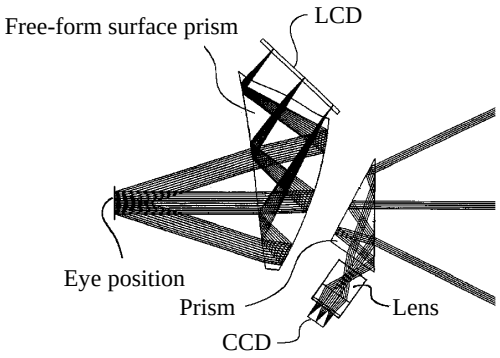
- Recent rapid enhancement of PC CPU performance,
- Great enhancement of the real-time graphic performance on Linux and OpenGL,
- Enriched software environments supporting virtual world development, such as Open Inventor and OpenGL.

More details on the hardware of the base platform are described in Section 6.

3.1. COASTAR Type Video ST-HMD

One of the major achievements of the MR Project is a series of ST-HMDs specially designed for MR and AR applications. Among them, the video ST-HMD based on COASTAR (Co-Optical Axis See-Through for Augmented Reality) technology received great interests from AR researchers [10]. This HMD has a feature that the optical axes of the built-in cameras and displays coincide as shown in Fig. 3 (a), thus providing true 3D sensation without a adverse effects (e.g., sickness or loss of balance), a result that is very difficult to achieve with off-the-shelf equipment.

The *MR Platform* provides the newest COASTAR-type video ST-HMD that has been redesigned and improved based on the prototype described in [10]. The HMD shown in Fig. 3 (b) offers integrated stereoscopic cameras



(a) Optical configuration

(b) Appearance

Figure 3. COASTAR type video see-through HMD

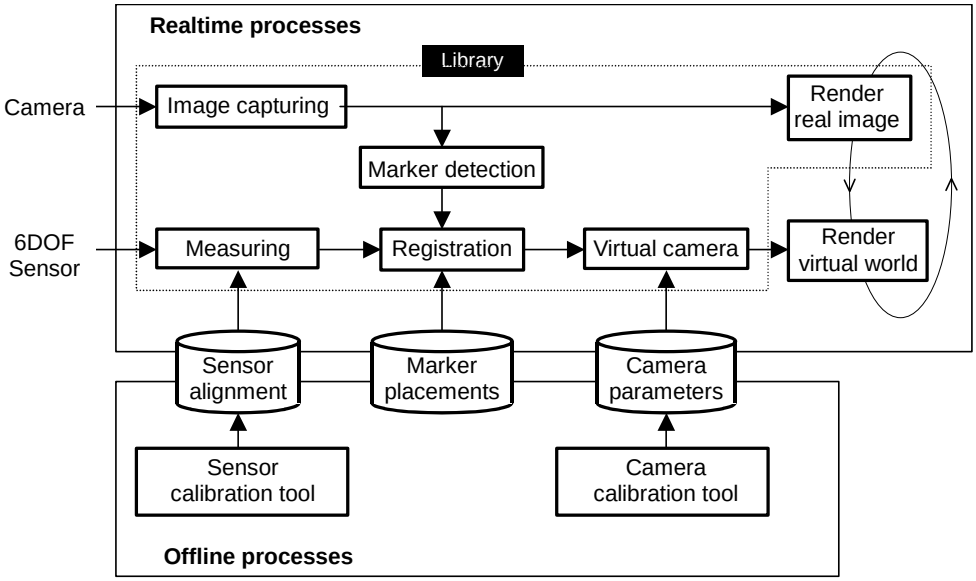


Figure 4. Basic processing elements of an MR/AR system

whose axes coincide with its display axes, stereoscopic display, simultaneous video outputs, and other features, including;

- Wide field of view: 51 degrees in horizontal direction, 37 degrees in vertical direction,
- High resolution: VGA (640 by 480),
- Light weight: 327 grams,
- Standard video output: NTSC.

This HMD may contain a receiver of a magnetic sensor such as a Polhemus FASTRAK for head tracking.

3.2. MR Platform SDK

In order to construct an MR/AR system, various processing such as real-time video capturing, high-speed image processing, sensor handling, and virtual world handling must be realized efficiently. In addition, a precise registration process, such as matching the coordinate system in a real world with one in a virtual world, is necessary. Such a situation poses great obstacles for constructing a practical applied system. Thus the SDK in *MR Platform* is provides functions and tools unique to such an MR/AR application. Figure 4 shows the basic processing elements necessary for realizing an MR/AR system. They are divided into two parts, library and utility tools. Please note that the library does not support any of the virtual world rendering shown in Fig. 4 because various dedicated libraries, such as Open Inventor and OpenGL Performer, are already available and our platform assumes their use.

3.2.1. Library of MR Platform SDK

The design policies for the library of *MR Platform SDK* are as follows:

Independent of CG rendering

As described, the library does not support virtual world rendering. However, the library provides a function to acquire viewing information (OpenGL-style viewing frustum, projection matrix, camera position and orientation, viewing matrix, and so on) necessary for rendering virtual world.

Object oriented programming style

The processing elements shown in Fig. 2 are highly independent of each other in function, but deeply related to in data flow. Therefore, individual processing elements are implemented as classes in C++, and designed to realize an object-oriented programming style that expresses the data flow directly by inter-class connections.

Easy to implement new algorithms

Since methods used in MR/AR are still the subjects of ongoing research, the library should be flexible to enhance its functions with new algorithms and methods. For example, a user may want to implement his/her idea on registration method. In that case, he/she can enhance the registration method of the corresponding class in the library. Implementing a class inheriting a corresponding sensor class can support a new tracking device.

The library is implemented according to the above basic design policies. The relation between the library and the base platform and applications is shown in Fig. 5. In the figure, V4L stands for Video4Linux, the Linux standard

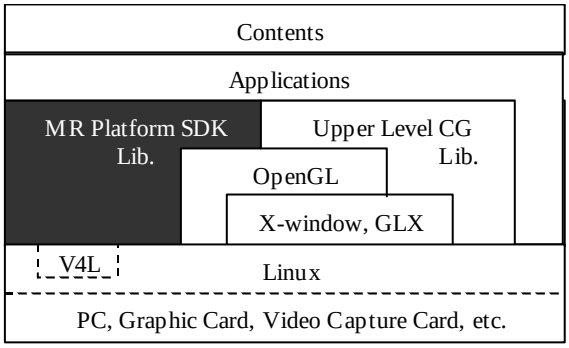


Figure 5. Library layer

video capture API. The upper level CG library is used for handling a virtual world. More specifically, this refers to Open Inventor , OpenGL Performer or others like them.

3.2.2. Utility Tools of MR Platform SDK

The sensor alignment information, marker placement information, and camera intrinsic parameters should be pre-calibrated in offline mode. These parameters vary for each environment or each time the real world is changed. Therefore, these parameters must be calibrated on the user side.

It is not an easy task, however, to get marker placement information in various environments, thus the SDK provides the calibration tools in order to get the other two types of information. Marker placement information should be prepared by some method, such as using a tape measure or a surveying instrument, not provided in the SDK. This marker placement information should be prepared in advance of the sensor calibration.

4. Functions of Library

To provide each function element as a module, the functions are installed as a C++ class library. The library consists of 127 classes that can roughly be classified into seven groups. The roles and function of each class are as follows:

Video device class

This class provides video device operation interfaces to specify the camera input signal type and the byte configuration of each pixel and to instruct the start/stop of capturing.

Image classes

This class provides basic image processing functions to generate images from a video capture device or a file, to detect color regions, and to label regions in a color detected image.

Sensor device classes

These classes are used for controlling position and orientation sensors and for handling the measurement results such as coordinate transformation.

Marker classes

A marker is a kind of a label that becomes an image feature placed in a real world for sensor error corrections. These classes manage marker placement in real world and detect markers from images according to marker placement information.

Camera classes

These classes perform geometric calculations to estimate the physical camera positions and orientations from information of the sensor device class and marker classes. Physical camera estimation results are converted into virtual camera information (parameters directly available to

```
... include header files

int main( int argc, char *argv[] )
{
    ... Open a window, create and initialize the virtual world
    // Generate and initialize the video device class
    MrpVideo video;
    video.initialize( MrpVideo::SVIDEO );
    // Generate the image class to refer the video device as the input source
    MrpCaptureImage capture( video );
    // Generate the marker class to read a marker file
    MrpMarker marker("marker.dat");
    marker.detector->setImage( capture );
    // Generate the sensor handling class
    MrpFastrak sensor;
    // Generate the camera class to refer the sensor class and the marker class
    MrpCamera6DOFSAndMarkers camera( sensor, marker );
    // Generate the rendering image class
    MrpImageRenderer renderer;

    // Main visual loop
    while (1) {
        // Measure by the sensor
        sensor.update();
        // Capture image
        capture.update();
        // Detecte marker
        marker.detector->detect();
        // Estimate camera position and orientation
        camera.update();
        // Render the captured image (real world)
        renderer.render( capture );
        ... Render virtual space based on the camera
    }
}
```

Figure 6. Simple sample program

OpenGL).

Rendering classes

The image rendering class renders a captured image as the real world under compensating the lens distortion using the camera intrinsic parameters. For affinity with any OpenGL based CG libraries, this class is coded only using low-level drawing instructions of OpenGL.

Other classes

This library also has the matrix-vector arithmetic operation classes for geometric operations and the communication class for data transmission to achieve a stereoscopic view by using two PCs.

Figure 6 shows a sample program using the functions of the above classes. Class objects are referred with the input and output of each processing element shown in Fig. 4 to create an MR/AR application reflecting the data flow described in Section 3.

5. Functions of Utility Tools

Utility tools are prepared for calibrating the tracking sensor and the parameters of the cameras build-in HMD.

5.1. Camera Calibration Tool

The calibration tool, which calibrates the intrinsic parameters of two cameras build-in HMD and the relative transformation between them, is provided in the SDK. Here the intrinsic parameters consist of the distortion of the imaging system and its perspective transformation.

Each camera built-in the HMD has some optical distortion, and does not shows the ideal pinhole camera characteristics. Without compensating for this distortion, incorrect registration arises using sensor data with vision HMD. This is one of the major factors of misalignment between the real and virtual worlds.

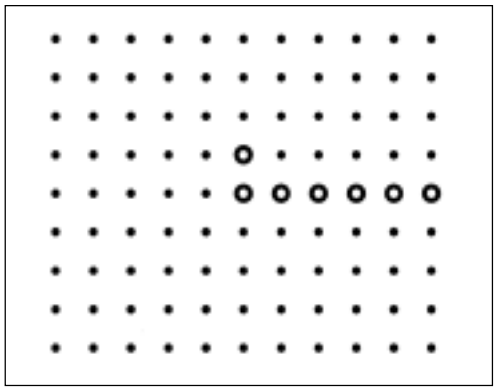


Figure 7. Pattern for camera calibration

In addition, the perspective transformation at rendering a virtual world must have the same perspective effect as the HMD-build-in camera imaging system. Even if the position and orientation of the camera is correctly tracked, the error of perspective transformation makes the registration incorrect.

Since two cameras are installed in the HMD for stereoscopic view, the intrinsic parameters should be calibrated for each camera. The relative transformation between the two camera coordinate systems should also be calibrated since an incorrect transformation gives us an incorrect 3D sensation.

If any special equipment, such as an X-Y stage or a special gauge cube, is required, this calibration is very difficult to be done by users. Thus a vision-based calibration tool, which does not need any special equipment, is provided in the *MR Platform SDK*. This tool uses a pattern printed on plane board as shown in Fig. 7. Each circle in the pattern is placed on a regular grid. A set of concentric circles makes the coordinate system on the plane. Thus the position of each circle can be determined from an image if the central part of pattern is in the image.

From circle positions on the image and their corresponding coordinates on the plane, camera calibration is done as follows:

(1) Estimate radial lens distortion

The lens distortion is modeled as follows:

$$r_0 = k_2 r_i - k_1 (k_2 r_i)^3$$

$$r_0 = \sqrt{(x_o - c_x)^2 + (y_o - c_y)^2}, \quad r_i = \sqrt{(x_i - c_x)^2 + (y_i - c_y)^2}$$

where (x_i, y_i) is a position on the ideal (non-distorted) image, (x_o, y_o) is a position on the actually observed (distorted) image, and (c_x, c_y) is the center of distortion. The calibration tool estimates k_1, k_2, (c_x, c_y) using several captured images with an iterative algorithm.

(2) Estimate perspective parameters

(2-1) Calculate the 2D homography from physical circle centers on the pattern plane to its captured and inverse-distorted positions. The 2D homography is in the form of a 3x3 homogeneous matrix.

(2-2) Calculate the focal length using an initial principal point (320, 240) and the homography. At the same time, the camera position and orientation can be calculated on the pattern plane in the 3D space.

(2-3) Based on the results of the step (2-2), project all circle centers on the pattern to the image plane. Update the principal point to reduce the displacement between the projected circle positions and the detected positions in image.

(2-4) Recalculate the focal length and the position/orientation of the camera.

(2-5) Based on the results of the step (2-4), update the principal point. Repeat the step (2-4) until the displacement is enough small.

(3) Base on the results of the step (2-4), calculate the relative transformation between two cameras.

In practice, a sequence of images captured while moving an HMD is used to make the calibration process precise.

5.2. Sensor Calibration Tool

The sensor calibration tool can calibrate the alignment information of the standard 6-DOF position and orientation sensor — the 3 SPACE FASTRAK from Polhemus Inc., the Flock of Birds or miniBIRD from Ascension Technology Corp. The parameters, shown in Fig. 8, that can be derived with this tool are as follows:

- The position and orientation of the transmitter in the world coordinate system (transformation from the world to the transmitter), and
- The position and orientation of the receiver in the camera coordinate system (transformation from the camera to the receiver).

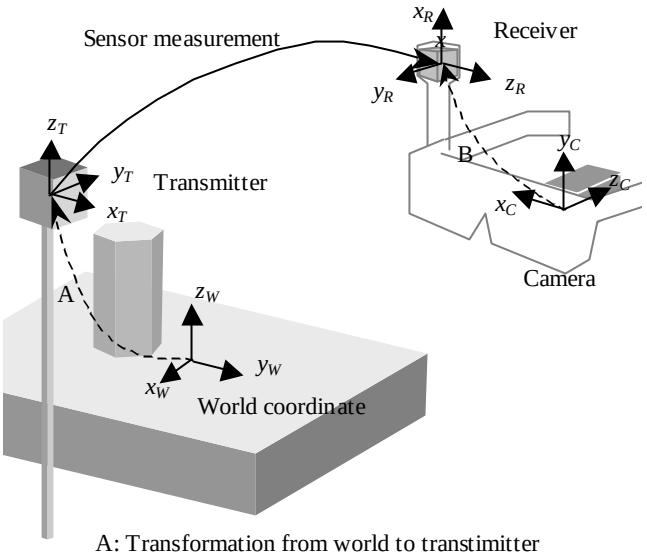
These parameters are necessary to find the position and orientation of the camera in the world coordinate system.

These parameters are estimated from:

- World coordinates of three or more non-collinear markers,
- Image coordinates of markers in images captured from several different positions and orientations,
- Sensor measurement values when the images are taken, and
- Initial values of the sensor alignment information (approximate values).

Of the above items, the world coordinates of the markers are data that should be known prior to this tool's preparation stage. The image coordinates and the sensor measurement values are data obtained while using the tool. In fact, most of the operations of this tool are intended to find these data. The initial values of the sensor placement information are not always required. However, in some cases, if the initial values are not set, an appropriate solution cannot be obtained.

Taking these data inputs, this tool projects each marker's world coordinate to the image based on the sensor measurement values and the sensor alignment information. The tool, then, calculates the sensor alignment information that minimizes the sum of errors between the calculated image coordinates and the actual image coordinate in the image, by using a Netwon-Raphson method.



A: Transformation from world to transtimitter
B: Transformation from camera to receiver

Figure 8. Sensor calibration

Figure 9 shows snapshots of these tools.

6. Operating Environment

The following hardware is required for *MR Platform* and development with it:

PC with the following components

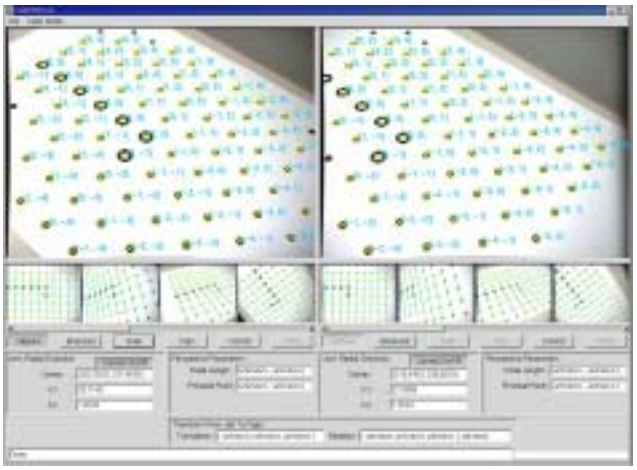
- Accelerated Graphics Port (AGP) bus slot.
- More than two Peripheral Component Interconnect (PCI) bus slot.
- A Pentium III 600 MHz or better CPU (A Pentium III 1 GHz or better CPU is recommended.)
- More than 256MB of RAM in total.
- Red Hat Linux 7.1 or later as its standard operating system.

Video capture cards

- Two PCI-based video capture cards equipped each with a Conexant Bt878 video decoder chip.

Graphic card

- A high-speed AGP-based graphics card equipped with a nVIDIA graphics processor with the TwinView or nView dual-display architecture.
- In order to obtain the full performance of the graphic card, the standard distribution X drivers,



(a) Camera calibration tool

(b) Sensor calibration tool

Figure 9. Snapshots of calibration tools

supplied with Red Hat Linux, must be replaced with the drivers supplied by nVIDIA Corp.

Tracking device

- A FASTRAK by Polhemus Inc. or a Flock of Birds, miniBIRD by Ascension Technology Corp. A receiver of these devices can be attached on the ST-HMD included in *MR Platform*.

By using TwinView or nView from nVIDIA Corp., X server enables OpenGL acceleration in the condition that the display size is 1280x480 and it has a single X screen in a single X display across dual VGA ports. The primary port can output the left half of the 1280x480 screen, and the secondary port can output the other side. By using this technology, an application uses a single 1280x480 GLX context window in order to display stereoscopic images.

7. Example Systems Using MR Platform

This section exemplifies two MR systems using the *MR Platform*. Both systems use the same configuration PC that has Dual Pentium III 866MHz CPUs, a TwinView GeForce2 MX graphics card and two video capture cards.

7.1. Simple Example

Figure 10 shows the behaviors of a simple example. The views are from the right eye but both the right and left eyes are used for capturing and drawing. In this example, the four markers shown in Fig. 10 (one of them is hidden under a virtual windmill) are used for the registration. A virtual building and the virtual windmill are placed on the table and the hexagonal column, respectively. Including the right-eye and left-eye processing, the one PC can refresh 63 frames or over per second. (Video-captured images can be refreshed at 30 fps only.)

By using the *MR Platform SDK*, this application was implemented in only about 450 lines of code, including virtual world rendering and window operations. Moreover, Open Inventor 2.1.5 for Linux by SGI is used as an upper level CG library, which can be used to handle a scene graph, for rendering a virtual world.

7.2. AquaGauntlet : An Entertainment in Mixed Reality Space

Figures 11 shows a more practical example where our mixed reality entertainment [11] was ported onto the *MR Platform SDK*. For handling and rendering the virtual objects, OpenGL Performer 2.4 for Linux by SGI is used.

8. Conclusions

This paper described a platform package, called "*MR Platform*," for research and development of augmented reality technology and applications. This package includes a parallax-less stereo video see-through HMD and a software development kit (SDK) for a Linux PC environment. A user of this package will be able to implement an MR/AR application without worrying about MR/AR specific functions such as video capturing, registration, and so on. This enables us to rapidly prototype MR/AR applications and to inject new life into R&D activities in this field.

As of this writing, the package is yet to be released publicly. However, as described in Section 7, the package is practically used in our systems and makes efficient development possible.

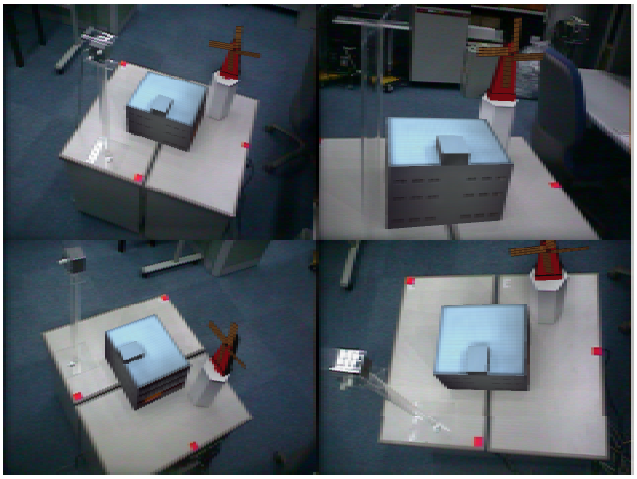


Figure 10. MR Platform based simple system

Figure 11. AR game "AquaGauntlet™"

Although the package described in the paper provides basic functions for MR/AR applications, additional functions would be useful for practical applications. A scene graph application program interface (API), interaction API that supports an interactive device, and shared mixed reality environment API are some of these. We are in the process of enhancing the package with these new functions.

Acknowledgments

The authors would like to thank Prof. Haruo Takemura of Osaka Univ., Prof. Yasuyoshi Yokokoji of Univ. of Kyoto, and Prof. Hirokazu Kato of Hiroshima City Univ. for their valuable comments on the design of *MR Platform*. Mr. Mahoro Anabuki, Miss Rika Tanaka and the other members of MR Systems Laboratory of Canon Inc. gave us direct or indirect support in the development of this package.

References

[1] H. Tamura et al.: "Mixed reality: Future dreams seen at the border between real and virtual worlds," *IEEE Computer Graphics and Applications*, vol.21, no.6, pp.64-70, 2001.

[2] R. Azuma et al.: "Recent advances in augmented reality," *ibid*, pp. 34-47, 2001.

[3] H. Tamura: "Overview and final results of the MR project," *Proc. ISMR2001*, pp. 97-104, 2001.

[4] A. Takagi et al.: "Development of a stereo video see-through HMD for AR systems," *Proc. ISAR 2000*, pp.68-77, 2000.

[5] T. Ohshima et al.: "AR^{2}Hockey: A case study of collaborative augmented reality," *Proc. IEEE VRAIS'98*, pp.268-275, 1998.

[6] K. Satoh et al.: "Case studies of see-through augmentation in mixed reality projects," *Proc. IWAR '98*, pp.3-18, 1998.

[7] T. Ohshima et al.: "RV-Border Guards: A multi-player entertainment in mixed reality space," Poster session, *IEEE IWAR'99*, 1999.

[8] C. Stratmann et al.: "Clear and present car: an industrial visualization in mixed reality world", *Proc. ISMR2001*, pp. 199-200, 2001.

[9] T. Ohshima et al.: " '2001: An MR-Space Odyssey'– An application of mixed reality technology to VFX for film production," *Proc. ISMR2001*, pp.201-202, 2001.

[10] H. Kato et al.: "Virtual object manipulation on a table-top AR environment," *Proc. ISAR2000*, pp.111-119, 2000.

[11] http://www.mr-system.co.jp/project/aquagauntlet

[12] http://www.hitl.washington.edu/research/shared_space/download/