3D VIEW: Designing of a Deception from Distorted View-dependent Images and Explaining interaction with virtual World

*Syed Muhammad Ali, **Zeeshan Mahmood, and *** Dr. Tahir Qadri

***Abstract*- This paper presents an intuitive and interactive computer simulated augmented reality interface that gives the illusion of a 3D immersive environment. The projector displays a rendered virtual scene on a flat 2D surface (floor or table) based on the user's viewpoint to create a head coupled perspective. The projected image is view-dependent which changes and deforms relative to user's position in space. The nature of perspective projection is distorted and anamorphic such that the deformations in the image give an illusion of a virtual three dimensional holographic scene in which the objects are popping out or floating above the projection plane like real 3D objects. Also, the user can manipulate and interact with 3D objects in a virtual environment by controlling the position and orientation of 3D models, interacting with GUI incorporated in virtual scene and can view, move, manipulate and observe the details of objects from any angle naturally by using his hands. The head and hand tracking is achieved by a low cost 3D depth sensor 'Kinect'. We describe the implementation of the system in OpenGL and Unity3D game engine. Stereoscopic 3D along with other enhancements are also introduced which further improves the 3D perception. The approach does not require head mounted displays or expensive 3D hologram projectors as it is based on perspective projection technique. Our experiments show the potential of the system providing users a powerful, realistic illusion of 3D.**

***Index Terms*- 3D projection, head coupled perspective, head tracking, Kinect, interaction**

I. INTRODUCTION

Humans possess 3D vision to perceive the structural information about their surroundings. The shape and size of object, lighting, view, perspective and stereoscopic vision, etc. allow us to judge the characteristics of the world around us. As the technology develops, computer systems are aimed to duplicate the real world in a computer environment. In order to give the illusion of 3D, we have to simulate the perceptual effects that can create realistic experience. "The mind has a strong desire to believe that the world it perceives is real" –John Lanier.

The origin of virtual reality dates back to 1500, when Italian artists painted two-dimensional surfaces mostly walls or ceilings to give the optical illusion of a 3D space by using some perspective tools and spatial effects. In 1968, first mechanical based head tracking experiment completed successfully which provided ambitious concept for head-coupled virtual reality

Systems Colin Ware implemented Fish Tank virtual reality to view stereo images of 3D scene on a monitor using a perspective projection coupled to the head position of the observer [1], [2]. Rekimoto presented a robust method for head tracking for Fish Tank VR using computer vision by processing image frames from the camera [3]. Johnny Lee in [4] proposed a system which uses head tracking for desktop VR displays (television, smart phone, tablet). He used set of Infrared LEDs on wearable glasses with an infrared camera in the Wii-mote to track the user's head position. Changing the view based on the head position created a realistic impression of Fish Tank VR. C. Cruz-Niera in [5] has described the CAVE system in detail where projectors are directed towards the walls and ceiling of a cube-shaped room to make immersive virtual reality. The trackers inside the CAVE track the user's head position precisely and images are displayed by projectors based on where the person is standing in the room.

Our project focuses to design an intuitive and interactive computer simulated augmented reality that gives the illusion of a 3D immersive environment. The projector displays a rendered virtual scene on a flat 2D surface (floor or table) based on the user's viewpoint to create a head coupled perspective. The projected image is view-dependent which changes and deforms relative to user's viewpoint. The nature of perspective projection is distorted and anamorphic, which need to be looked from a particular point as shown in Fig.1. When viewed at a

*Department of Electronic Engineering Sir Syed University of Engineering &Technology, Karachi, Pakistan. malisq@hotmail.com,
**Department of Electronic Engineering Sir Syed University of Engineering &Technology, Karachi, Pakistan. engineer.zeeshan@outlook.com,
***Department of Electronic Engineering Sir Syed University of Engineering &Technology, Karachi, Pakistan. mqadri@ssuet.edu.pk

certain angle, the deformations in the image give an illusion of a virtual three dimensional holographic scene in which the objects are popping out or floating above the projection plane like real 3D objects.

Fig.1. Anamorphic illusions created from distortion of 2D image (source: http://julianbeever.net)

The estimation of the user's head position in space is carried out by using inexpensive 3D depth sensor 'Kinect'. The depth image is processed with OpenNI framework along with NITE middleware to retrieve the 3D position of the user's head [6], [7]. The information is then used to create a view-dependent projection of a virtual scene by applying an off-axis perspective projection. In addition to implementing head coupled perspective, stereoscopic 3D with anaglyph rendering is used to further improve the 3D depth perception. Also, projecting shadows of the objects and applying texture and parallax mapping greatly optimizes the user experience. In this paper, the approach is implemented in OpenGL and Unity3D game engine [8], [9]. To build an immersive and interactive augmented reality experience, natural full-body interactions are proposed. The user can manipulate and interact with 3D objects in a virtual environment by controlling the position and orientation of 3D models, interacting with GUI incorporated in the virtual scene by either his/her hands or can use pen or wand based interaction. By using 3D hot points with hand-tracking technique, the user is able to interact with UI elements in the scene and can view, move, manipulate and observe the details of objects from any angle naturally by using his hands.

The outline of the rest of the paper is as follows, Section II will investigate the system model of the system relating to each of the main stages of the work in detail. Section III concludes the paper with applications, limitations and future enhancements of the project.

II. SYSTEM MODEL

The overall work is composed of five fundamental stages. The first stage describes the approach to track the user's head. The second stage investigates the transformation from Kinect camera coordinates to projection space coordinates. In the third stage, the off-axis perspective projection or skewed camera is explained. The fourth stage comprises of implementation of the effect in 3D engine i.e. OpenGL and Unity 3D along with some illusion enhancements. In the last stage, it is described how a user can manipulate and interact with virtual objects by using his body. The complete block diagram of the proposed system is illustrated in Fig.2.

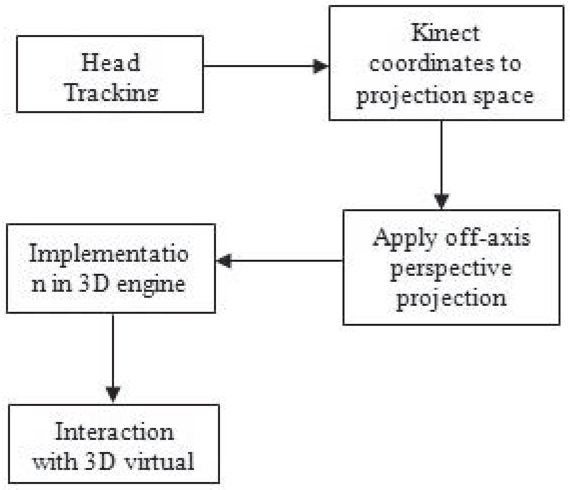
Head Tracking
Kinect coordinates to projection space
Apply off-axis perspective projection
Implementation in 3D engine
Interaction with 3D virtual

Fig.2. System model of the system

Head Tracking

The first stage of the system deals with the estimation of the position of the user's head in the environment. In order to develop an augmented reality system in which the perspective of the scene changes as the user moves his head, the position of the eyes or head of the user must be known. There are different ways to implement the desired approach. The head tracking methods can generally be classified into two types. One method requires wearable sensors while the other implies vision based techniques.

In head mounted sensor tracking, users have to wear sensors which use technologies like Infrared, RF or Ultrasound etc. The location of the head is then computed based on the position of the sensor. An example of such system is implemented by Johnny Lee in which he used a set of Infrared LEDs on wearable glasses with an infrared camera in the Wii mote to track the user's head position [4] [10]. Also, there is a commercial product available called Track IR which allows to track the user's head with six degrees of freedom [11]. The product consists of an optical capture device or camera and a head mounted sensor called Track Clip. The Track Clip consist of Infrared LEDs which is detected by the camera. The disadvantage of such systems is that users have to wear some sort of device and usually require line of sight visibility, which limits the natural way of controlling view-dependent perspective. The effect should work seamlessly for everyone all the time without the need of exchanging the sensors again and again. So, considering the nature of the project, using head mounted sensors to track the head position is inappropriate.

Fig.3. Track IR + Track Clip

On the other hand, vision based interfaces are feasible and cheap nowadays and are most suited to the essence of our research. There is a variety of algorithms available for face detectors, body part detectors, pose estimation and gesture training-testing. The Face API library by Seeing Machines offers remarkable face-tracking along with facial features [12]. It requires no hardware, but a color camera. The limitation is that it needs good lighting conditions and the user has to face the camera so that his face is tracked. This is too restrictive for the project application so we investigated depth cameras for implementing head tracking.

With depth cameras or 3D time-of-flight sensors, we can use different techniques to track the user's head in 3D depth images. Ehsan Parvizi proposed a head detection algorithm which is based on contour analysis on depth images to extract the curves of moving regions [13]. Another 3D head tracking system was described by Salih Burak which is based on recognition and correlation-based weighted interpolation [14]. Moreno also proposed an adequate head tracking algorithm in which the head shape is modeled by an ellipse with a trained color histogram of skin and hair samples [15].

The majority of the available depth sensors is quite expensive. In 2010, Microsoft launched Kinect which is the first 3D depth sensor that allows users to interact naturally through body gestures. It was made for the Xbox360 console for a richer gaming experience, but by taking the advantage of open USB connection of Kinect, the open source drivers for PC were released. The drivers provide access to Kinect depth and RGB image data streams and soon it proved to be more than just a gaming device. Now, it is widely used by developers to take control of a wide variety of applications and robots also. Also, Prime Sense Company released its own programming framework called OpenNI (Open Natural Interaction) to deal with Kinect data streams. Implementing head tracking on 3D depth images from scratch is complicated and requires the use of different algorithms. Therefore, we used Kinect as a low cost depth camera solution with an OpenNI framework along with NITE (Natural Interface Technology for End-user) to track user's skeleton (head joint for now) in depth image sequences. NITE is a high level middleware developed by Prime Sense which provides algorithms for effective hand and body joint tracking. Prime Sense distribute these algorithms for commercial purpose and keeps the code closed.

When the user enters into Kinect's field of vision, OpenNI detects the user's presence and starts the calibration process. After the calibration is successful, the data of the user's body joints are available. The x, y and z coordinate of the tracked joint, i.e. the head is constantly stored in 3D vectors. These values are then further used to create a view-dependent 3D projection. The next stage describes the transformation of head position vector from the Kinect camera coordinates to projection space coordinates.

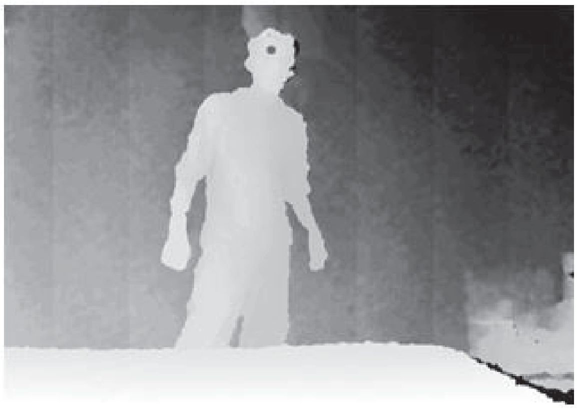
Fig.4. Head is being tracked from depth images by using OpenNI and NITE Middleware

III. KINECT CAMERA COORDINATES TO WORLD SPACE COORDINATES

After the estimation of a user's head position, the second stage of the proposed system is the mapping of Kinect camera coordinates to world space coordinates of the projection or the calibration of the projector. The issue here is that Kinect use different coordinate system. In Kinect, the +y points downwards, +z points from the Kinect towards the user and the origin lies at the top left corner of the frame as shown in Fig.5. The coordinate system of the projection plane is also shown.

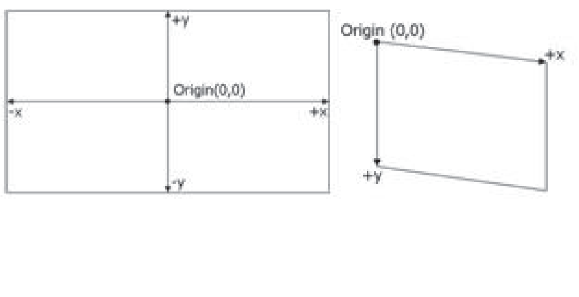

Fig.5. Kinect coordinates system (left) and Projection coordinate system (right)

In order to map the Kinect camera coordinates to the world coordinates of the projection plane, few things must be considered. First, the projection of the 3D scene can be projected on any surface, table, wall or floor. In our case, we are projecting it on the floor as shown in Fig.6.

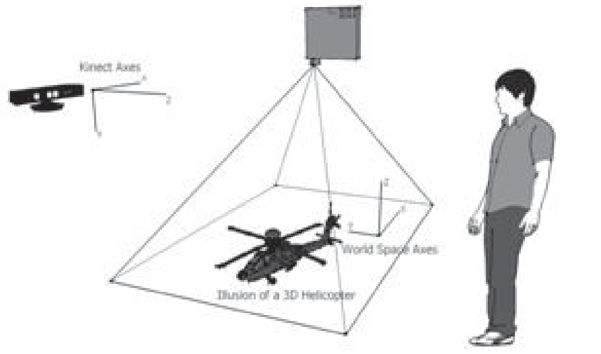

Fig.6. System configuration with Kinect, Projector, user and projection space. The 2D image gives the illusion of a 3D Helicopter

Now from Fig.6. by comparing the Kinect axes and world space axes, we can see that the direction of x-axis is same for both the system. But the y-axis of Kinect corresponds to the z-axis of projection plane and similarly, the z-axis of Kinect represents the direction of y-axis in the projection plane. The Kinect provides 640 x 480 resolution depth image with a depth precision of 11 bit or 211= 2048 values per pixel. Let us denote (xh,yh,zh) be the position of the user's head and (xm,ym,zm) be the world space coordinate of projection plane. Keeping the above things in mind, we can write formulas for this transformation,

$$x_m = (x_h - 320).k_x \quad (1)$$

$$y_m = (1800 - z_h).k_y \quad (2)$$

$$z_m = (y_h - 100).k_z \quad (3)$$

Where kx, ky and kz are the scaling constants which define the sensitivity of the head movement and should be adjusted manually depending on the size of the projection space dimensions. The numeric values (320, 1800 and 100) were evaluated by positioning the user's head exactly at the origin of the surface of projection and then noting the head coordinates from Kinect. The values were found to be,

$$\begin{pmatrix} x_h \\ y_h \\ z_h \end{pmatrix} = \begin{pmatrix} 320 \\ 100 \\ 1800 \end{pmatrix} \quad (4)$$

Therefore, in order to obtain the correspondence between Kinect coordinates and projection plane coordinates, these formulas are created. Based on these values, we will construct an accurate distorted perspective image that would give a real 3D illusion to the user.

Apply Off-Axis Projection or Skewed Frustum

The next stage is to project a 3D scene on a 2D surface (floor) in such a way that this distorted perspective gives the user a real three dimensional perspective. So, the process by which 3D models or objects are transformed to 2D images implies "Perspective projection". Since our head-coupled system consists of a simple 3D virtual scene, to render a three-dimensional scene on a two-dimensional display screen, we need to determine where on the screen each 3D point should be drawn. Let us denote P(x,y,z) be the coordinate of any 3D virtual point and P'(x',y',z') be the coordinate of the point projected on the 2D image plane as shown in Fig.7.

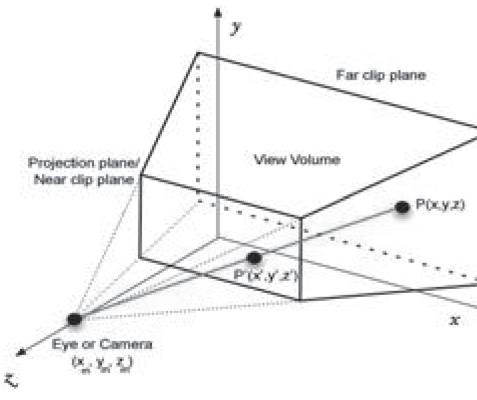

Fig.7. Projection of a point in 3D space to the projection plane

Also the coordinates of the viewpoint or camera in 3D scene are the same as were determined in the second stage, which was (xm, ym, zm). That means that the camera movement in the 3D virtual scene is directly coupled with the user's head.

Now, how to determine the correct perspective projection for every 3D virtual point P? Usually, the standard camera most often used for viewing in computer graphics uses a perspective projection matrix to perform a perspective transform. The perspective projection matrix is a 4x4 matrix that has very importance in the computer graphics pipeline as many 3D renders including the most common 3D rendering engine "OpenGL" uses it. To implement this operation, we will multiply each point P(x,y,z) in the scene with the perspective projection matrix M frustum which is defined in equ. (5).

$$P' = M_{frustum}.P = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} \quad (5)$$

Where r, l, t, b, n and f are right, left, top, bottom, near and far clip planes respectively. Fig.8. shows the view frustum of a 3D scene which is bounded by six planes, four of which correspond to the edges of the screen and are called right, left, top and bottom frustum planes while the remaining two planes define the minimum and maximum distances at which objects in a scene are visible to the camera and are called near and far frustum planes.

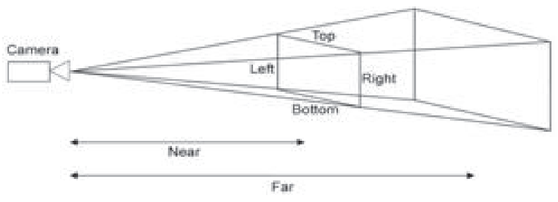

Fig.8. Left, right, top, bottom, near and far frustum planes

Consider the camera is centered with the image plane parallel to the x-y plane. For a standard symmetric viewing frustum, the values of the clipping planes have the following relation, left = -right and top = -bottom. For the near and far clip planes, it is better to keep them at a fixed position in the world coordinate system. In our case, we choose 0.3 for near and 1000 for the far clip plane.

Skew Camera

We have described the transformation of a standard perspective projection for the camera. The above relations between frustum values are only true when the camera is centered at the origin. Now as the user's head moves, camera also changes its position. If a camera moves towards the right, the whole scene will appear to be shifted towards the left in the camera view. In that case, the left boundary of the image plane is surely not equal to the absolute value of the right boundary. Also, the top and bottom clip planes do not follow the standard relationship. This is because the edges of the view plane are not symmetric with the viewing direction. Consider Fig.9. and note that the red points are the screen borders. It is needed to consider the position of the 3D model within the frustum. The key insight for achieving the virtual reality is that these red points or borders remain constant in the camera view despite the camera has changed its position. The screen is something like a window into the virtual reality, and to align the real world with the virtual reality, we need to align the frustum with that window [16].

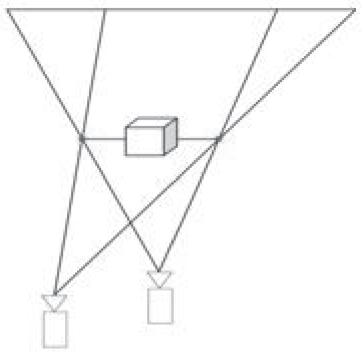

Fig.9. The red points remain constant in the camera despite the changes in camera position

This is where an off-axis perspective projection or skew camera comes into play. In order to align the frustum with the screen borders, we have to modify the left, right, top and bottom clip plane values in such a way that when these values are put into a projection matrix, the screen borders with each point lies within the view frustum. This method centers the scene in the middle of the window and stretches them to fit. The two terms in the projection matrix, $\frac{r+l}{r-l}$ and $\frac{t+b}{t-b}$ are responsible for skewing or distorting the scene.

To evaluate the values for the left, right, top and bottom planes, first step is to compute the boundaries of the 3D scene. This can be done by calculating the bounding box for all the objects in the scene. The width and height of the bounding box are then used to define the values of the clipping planes which is as follows,

$$l = Scale.\left(-\frac{width}{2} - x_m\right) \quad (6)$$

$$r = Scale.\left(\frac{width}{2} - x_m\right) \quad (7)$$

$$b = Scale.\left(-\frac{height}{2} - y_m\right) \quad (8)$$

$$t = Scale.\left(\frac{height}{2} - y_m\right) \quad (9)$$

We will investigate about the "Scale" parameter in the above equations later. Firstly, we discuss how these formulas are created. Consider the movement of camera along the x-axis. To stretch the scene to fit in the camera view, the left and right planes are set by trial and error, which are as follows,

For Camera:	$-8.85 < xm < 8.85$
Left	: $0.9 < l < -1.5$
Right	: $1.5 < r < -0.9$

We have retrieved the correct left and right range values for any given xm. But we need a mapping function which should provide a mathematical relation between these values. So, in order to re-map a number from one range to another, a standard approach known as "Linear interpolation" can be used which is illustrated in equ. (9).

$$y = \left[(y_{\max} - y_{\min}) \cdot \frac{x - x_{\min}}{x_{\max} - x_{\min}} \right] + y_{\min} \tag{9}$$

Putting values in the above equation for left clip plane results,

$$l = \left[(-1.5 - 0.9) \cdot \frac{x_m - (-8.85)}{8.85 - (-8.85)} \right] + 0.9 \tag{10}$$

After simplifying, it gives,

$$l = 0.136(-2.21 - x_m) \tag{11}$$

The value 2.21 here is approximately equal to the half of the width of the bounding box for the 3D scene which is 4.4. The value 0.136 corresponds to the term "Scale" in the equ. (5). Similarly, we evaluated the equations for right, top and bottom clip planes.

$$r = 0.136(2.21 - x_m) \tag{12}$$

$$b = 0.136(-1.25 - y_m) \tag{13}$$

$$t = 0.136(1.25 - y_m) \tag{14}$$

Where, value '1.25' is half of the height of the scene.

The above relations for the left, right, top and bottom clip planes perfectly stretch image to fit inside the view frustum and skew it depending on the position of the camera. But these formulas only work when the camera is moved along x-y plane. Here the off-axis projection properly works when camera lies at z=0. But what if the camera is moved along z-axis also? Unfortunately, this leads to the boundaries no longer remain constant in the view and the frustum alignment gets destroyed. To overcome this problem, the "Scale" value must be set in such a way that the above relations for clipping planes come true for any zm value. We set the z-axis range to 0 <zm< 1.75 to measure the values.

For every 0.25 increase in zm, we put the proper "Scale"value by trial and error such that the off-axis projection is correct for all the movements in xy plane. The results are shown in the table.

zm	Scale
0	0.136
0.25	0.152
0.5	0.175
0.75	0.204
1	0.246
1.25	0.311
1.5	0.42
1.75	0.66

Table I: Relation between zm¬ and Scale

The graph between zm and Scale is plotted which is shown in Fig.10. It is noticed that these values have a non-linear relationship. The curve gets quite flat for the values of zm greater than 1.25. It means that there is very high precision at points closer to far plane and small change in zm can greatly affect the Scale value. To minimize this problem, the range of zm should be as small as possible so we choose the range from 0 <zm< 1.25. Now to map the values of zm to the Scale values, a fitting function is calculated for the set of data points. We find best approximate fitting function by using different regression techniques like Logarithmic, Exponential or Power regression methods. For the relation between above two variables, we found polynomial least square fittings as the best approximation function with least error which is illustrated in equ. (15).

$$\text{Scale} = 0.0506z_m^4 - 0.0647z_m^3 + 0.0727z_m^2 + 0.0513z_m + 0.136 \tag{15}$$

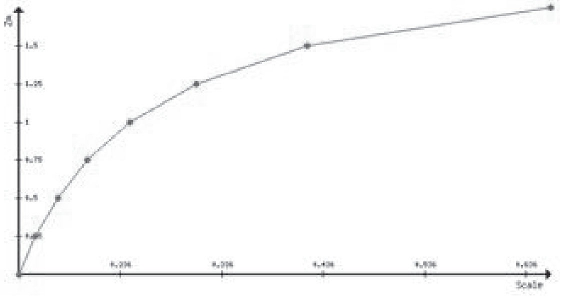

Fig.10. Graph between Scale and Zm

So in our case, the final values for the clipping planes are,

$$l = \text{Scale}(-2.21 - x_m) \tag{16}$$

$$r = \text{Scale}(2.21 - x_m) \tag{17}$$

$$b = \text{Scale}(-1.25 - y_m) \tag{18}$$

$$t = \text{Scale}(1.25 - y_m) \tag{19}$$

The above explained approach will result in a skewed projection according to the viewing person's head position. The deformation of the image projected on the surface when viewed from any direction gives a realistic illusion of a 3D immersive environment. In the next stage, we implement the proposed system in a 3D rendering engine and describe how the illusion can be enhanced by rendering shadows, shaders and using 3D stereoscopy.

Implementation in 3D Engine

The previous three stages have investigated the view-dependent 3D projection or head-coupled perspective approach in detail. This stage describes the implementation of the proposed system in OpenGL API and Unity3D Game Engine. We will also explore different techniques to enhance and improve the 3D perception by using stereoscopy, shadows and shader mapping.

OPENGL

OpenGL or Open Graphics Library is a cross-platform API for rendering 2D and 3D graphics and it consists of several methods and functions to deal with the operations and programming of high quality 3D graphical objects. To achieve hardware-accelerated rendering, OpenGL is implemented with Graphics processing unit (GPU) and as it is not platform-specific, it allows us to write an application which can run on many different graphic cards which increases the chance that the application will continue to work when new hardware becomes available [17].

In OpenGL, the camera does not move. It always stays fixed at origin (0,0,0) in the scene, looking in the same direction. To simulate the appearance of moving the camera in a particular direction, we have to move all the objects in the 3D world in the opposite direction. The values (xm, ym, zm) now control the transformation of the entire scene. Therefore, a slight modification is needed in equ. (5) to equ. (8) so that the effect can be implemented properly in OpenGL. The modified equations are:

$$\mathrm{l} = \mathrm{Scale}\,.\left(-\frac{\mathrm{width}}{2} + \mathrm{x_m}\right) \tag{20}$$

$$\mathrm{r} = \mathrm{Scale}\,.\left(\frac{\mathrm{width}}{2} + \mathrm{x_m}\right) \tag{21}$$

$$\mathrm{b} = \mathrm{Scale}\,.\left(-\frac{\mathrm{height}}{2} + \mathrm{y_m}\right) \tag{22}$$

$$\mathrm{t} = \mathrm{Scale}\,.\left(\frac{\mathrm{height}}{2} + \mathrm{y_m}\right) \tag{23}$$

The perspective projection matrix in equ. (5) is applied by glFrustum function in OpenGL. The function takes six parameters: left, right, bottom, top, bottom, near and far values respectively and then creates the projection matrix automatically.

To create a visually appealing demonstration of an immersive 3D experience, it is an essential requirement to animate the 3D scene and create an interactive graphical user interface. In OpenGL, only graphic rendering is concerned. It does not provide functions for 3D model animations, GUI designs and other amazing features. This is where Game engines come in.

IV. UNITY 3D

As previously discussed, OpenGL is a graphical API. There is a large difference between Graphical APIs and Game Engines. Graphical APIs provides programmers to modify the parameters of the 3D objects precisely at a basic level. In Game Engines, the designing and scripting of the 3D world is done at a higher level and the majority of the animation and rendering work is left in the software to be automated. The graphical APIs are therefore less useful for the application of this project as the animations, rendering and modeling of 3D objects is done by Game Engine quite easily and time-efficiently. This is an advantage for developers to spend more time on the algorithm. Unity 3D is a very popular development engine for the creation of 2D and 3D games for desktop platforms, consoles, mobile devices and web based applications. It is flexible, multi-platform and user-friendly. There is wide support available online and the features like 3D models, environments, animations, scripts, textures, shaders etc. are easily available at the Unity asset store. There are also projects like Zigfu which support Kinect directly in the Unity3D engine. That's why we designed our system in Unity3D.

We have created a simple virtual scene composed of a 3D apache helicopter model and a bricked floor. The helicopter rotor is animated to spin such that it gives a nice cool effect of taking-off. Unlike OpenGL, Unity3D allows to move and rotate the camera. The "Cam Projection Matrix" function applies the perspective projection to camera view. In Fig.11. a few pictures of our system are shown. The pictures are taken from different viewpoints by placing the camera directly in front of user's eyes. The deformations in the rendered scene of a helicopter give perfect 3D illusion. The full demonstration of our system can be viewed online on our website.

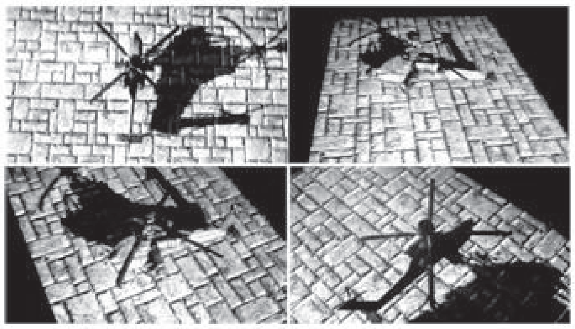
Fig.11. Pictures of our system taken from four different viewpoints by placing camera directly in front of user's eyes. The resulting impression gives a realistic illusion of 3D

V. SMOOTH CAMERA MOVEMENT

So far we have achieved an effective head coupled perspective. But there are some factors that limit the naturalness of the impression of 3D system. Among them, one issue is the precision and latency of the head tracking system. The inaccuracy and lack of precision of Kinect results in small rapid oscillations and jitters in the estimation of head position. Since the camera is directly coupled with the head position, it also fluctuates giving an unrealistic perception. To reduce this problem, we achieve smooth movement from point to point. Instead of updating camera position instantly, we add a smooth damp towards the target position. The position of camera is non-linearly interpolated between the current position and the target position by a damping factor of speed and Δt. The camera position is equal to the previous position of the lerp which is multiplied by the speed and Δt.

Where "Lerp" is a function to calculate linear interpolation between two points and "speed" is a damping parameter. The greater the speed, more quickly the camera snaps to the target location. The resulting motion of the camera is non-linear, i.e. the camera moves quickly at first, but slows down as it reaches the target position. The value of "speed" should be set precisely because if it is set too small, then the user will observe a delay and if it is set high, it will cause the projection to jitter. We have to make a trade-off between smoothness and delay.

VI. SHADOWS AND SURFACE MAPPING

To optimize the user experience and to enhance the illusion of 3D, we introduced shadows in the 3D scene. Shadows hold very much importance as they help us to understand the shape and characteristic of objects. Without shadows, our 3D scene does not look practically convincing and lacks realism. There are several methods for casting shadows of the 3D objects. These methods differ in quality and performance of the result. In OpenGL, there is no direct support for projecting shadows, but we can implement the techniques by writing our own functions. Luckily, Unity3D makes it possible to render real-time shadows easily and each object can cast either hard or soft shadows on any plane. So, we placed virtual directional lights in our virtual scene and projected hard shadows of the 3D model on the bricked floor plane.

In addition to projecting shadows, we used high quality textures with parallax mapping to greatly enhance the details and the appearance of the virtual scene. Texture mapping in computer graphics refers to adding detail, surface texture or color to a 3D model or surface while the parallax mapping is applied to the textures to look like as they have depth as the viewer's eye or the camera moves across the scene. Parallax mapping makes the textures more realistic without the need of adding complex geometry. The technique looks best on floors or walls, thus we implemented it on a bricked floor in our virtual scene.

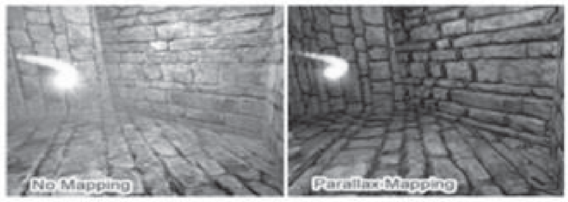

Fig.12. Comparison between no-mapped and parallax mapped floor and walls. Note the depth difference

One thing should be considered here is that when it comes to render shadows, mapping textures and advanced visual effects, there is a conflict between performance and quality of the system. These methods, although improving the quality of the 3D illusion, but due to the complexity of objects, they decrease the performance of the whole system.

VII. STEREOSCOPY

Using head tracking in virtual reality offers great and effective 3D perception. It is noticed that the illusion of virtual scene is more convincing when viewed with a single eye. This is because with two-eyed viewing, there is a depth cue conflict between the stereoscopically perceived surroundings and the monoscopic projection of the scene [18]. Therefore, the system can be further improved by adding stereoscopic 3D. Stereoscopy refers to presenting two slightly different, offset images, one for each eye. These two images are perceived by our brain to determine 3D depth of the view. Several methods like Anaglyphs, polarized filters, shutter glasses or head mounted displays exist to achieve the stereoscopic effect. In color anaglyph systems, the two images are encoded using filters of different chromatically opposite color pairs, among which red/cyan combination is the most common. The two differently filtered color images are then placed on top of each other to form a single image and when viewed through a color-coded anaglyph glasses, a 3D stereoscopic image is perceived. The anaglyph method is chosen for the project as it is cheap and requires no special hardware or display systems except a low cost anaglyph glasses. In Unity3D, the effect is implemented by rendering two camera views, one for each eye. Each camera is offset horizontally about the central position and then encoded using red/cyan color filters. Also, there are built-in assets available in the Unity asset store that provide full control of all stereo 3D parameters.

That pretty much covers everything we need to develop an effective head coupled perspective that gives the user an illusion of 3D immersive environment. It will be a more interesting and practical approach if the user is able to interact and manipulate objects in 3D scene with his body or hands. The next stage describes the implementation of interaction in virtual environments through our body.

VIII. INTERACTION WITH 3D VIRTUALSCENE

This stage investigates the technique to build a real world experience, in which a user can manipulate and interact with 3D objects in virtual environments. The user can control the position and orientation of 3D models, interact with GUI incorporated in the virtual scene by either his/her hands or can use pen or wand based interaction. In our proposed system, we focus on implementing full-body or hand based interaction.

3D Hot points

Coming back to the Kinect, so far we were accessing the depth image which consists of x and y position along with the depth of each pixel. Now, we construct a “Point cloud” or a 3D space of a set of points by using x, y and z (depth) values of each pixel. The point cloud is a 3D version of, pixels in which each point usually contains position information. The collection of all these 3D data points makes them look like a cloud of points floating in space. It is very useful way of visualizing or representing the transformation of each Kinect’s pixel into a three dimensional point.

In everyday life, we touch objects and it is the most common and natural way of making interaction. In our augmented reality system, 3D hot points are used to build an interactive touch or hover triggered application. This feature is inspired by the proposed work by Greg Borenstein in his book “Making Things See” [19]. Basically, Hotpoint is an interactive 3D floating element, particularly a cube or cuboid, which is drawn in a point cloud at such position that it corresponds with an area in the real world to which a user makes contact. To activate a 3D hot point, it is required that some part of point cloud or 3D data points lie inside the bounds of the element. When the number of points inside the hot point exceeds the defined threshold value, the hot point is activated. We added various hot points exactly slightly above the projection area such that when the user places his hands or any other solid object inside the specified bounds of hot point, then the touch or hover function is triggered.

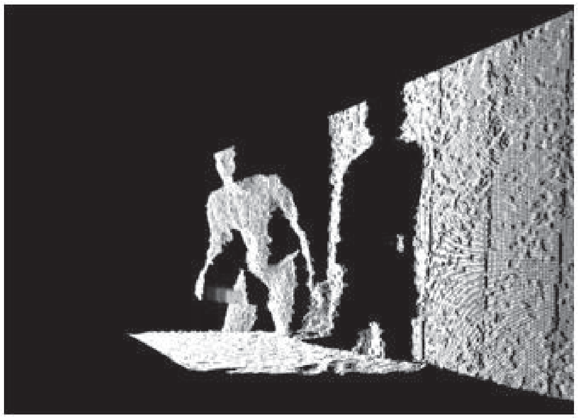

Fig.13. Viewing point cloud from an alternate point of view. Here the user is interacting with 3D Hot points (blue, red and green boxes) in point cloud which are placed above the projection area

In this way, the user can touch virtual objects in the air to perform different functions. In our case, the color and the animation of the spinning rotor of helicopter is triggered with hot points. Also, the user interacts with UI elements quite easily without even touching the projection surface or screen.

IX. TRACKING HANDS

As said earlier in the first stage, the OpenNI framework along with NITE (Natural Interface Technology for End-user) has the ability to track user’s skeleton accurately. We can easily access the position of any joint which is in our interest and use this data to interact with 3D objects. So, instead of loop checking each and every point in point cloud whether or not it lies inside the hot point’s bounds, a more time-efficient and practical approach is to check whether the position of the user’s hands lie inside the hot point’s bounds in point cloud. Besides touch or hover functions, we have implemented a more intuitive way for interacting with 3D virtual objects by controlling the position and orientation of 3D models by using both hands as if the user is holding the model in his hands and manipulating it like a real object, thus creating a realistic and richer user experience.

To implement a hand-based virtual interface, we need the relative position and orientation data of both hands. The position and angle between two hands is used to control the position and orientation of 3D models in the virtual scene. In 2D, only angle is needed because there is only one plane in which vectors are rotated. But in three dimensional rotations, an angle is needed along with the direction or axis about which the rotation is made. The representation is called Axis-angle method which calculates both the axis and the angle to represent two vector's relative rotation. Let $\vec{P}$ and $\vec{Q}$ be the 3D vectors representing left hand and right coordinates respectively. We calculate difference vector by subtracting $\vec{Q}$ from $\vec{P}$ which results in another vector D which represents the direction from right hand left hand as well as the distance between them.

$$\vec{D} = \vec{P} - \vec{Q} \qquad (24)$$

To apply the orientation of vector $\vec{D}$ to our 3D model, we have to define another vector that represents the orientation of 3D model. We define the orientation of the model as a unit vector $\vec{M}$ (1, 0, 0) pointing towards positive x-axis because we want to point the width of the model (the wider side of the model) in the direction of vector$\vec{D}$. The angle Θ between the vectors $\vec{M}$ and $\vec{D}$Is calculated by applying dot product of both vectors.

$$\Theta = \cos^{-1}(\vec{M}.\vec{D}) \qquad (25)$$

The axis $\vec{A}$ about which the angle works are given by the cross product of two vectors $\vec{M}$and$\vec{D}$.

$$\vec{A} = \vec{M} \times \vec{D} \qquad (26)$$

Now we can apply the angle and axis information to our 3D model. The rotation for the axis-angle method can be expressed in terms of a 3x3 rotation matrix R, defined in equ. (27).

$$R(\Theta) = \begin{bmatrix} A_x^2C + \cos\Theta & A_xA_yC - A_z\sin\Theta & A_xA_zC + A_y\sin\Theta \\ A_yA_xC + A_z\sin\Theta & A_y^2C + \cos\Theta & A_yA_zC - A_x\sin\Theta \\ A_zA_xC - A_y\sin\Theta & A_zA_yC + A_x\sin\Theta & A_z^2C + \cos\Theta \end{bmatrix} \qquad (27)$$

Where Ax, Ay and Az are the components of unit vector A, Θ is the angle of rotation and C = 1- cosΘ. In Unity3D, the function "Quaternion Angle Axis" rotates the 3D model based on provided axis and angle values. Also the model is translated and positioned based on the relative position of both hands inside the projection of the virtual scene. The above discussed methods provide impressive interaction with virtual objects as if they are real. The user is able to view, move, manipulate and observe the details of objects from any angle naturally by using his hands.

X. CONCLUSION

We introduced a cutting edge augmented reality system in which the computer generated projection provides users a powerful realistic illusion of 3D. It serves as a basis for direct interaction with 3D immersive environment. The projection cleverly adopts the viewpoint of the user in real-time. The system allows the user to walk freely in the immersive environment, making interactions with hovering menus and virtual objects by his body, thus pushing the limits of open interactivity. Although combining stereoscopy with head coupled perspective improves depth perception, the illusion also works great with only using head coupled perspective. The three dimensional head and hand tracking from Kinect is effective and does not need extraneous markers or sensors to wear, facilitating the goals of the project. The system is currently limited to providing the 3D illusion to a single user at a time, viewers other than the tracked one will not be able to perceive the effect. This restriction can be solved by using the active shutter display system in which each user sees its own perspective. Also, one of the limitations is the poor depth capture of Kinect in outdoor sunlight. Our approach proposed in the paper has proven to be the next generation technology, which has applications in several different fields. It can be used in military training purposes to simulate the real situations. Also it delivers a learning platform for medical learning and training. The technology can be easily integrated with mobile devices to build an interactive head coupled perspective. By integrating the view-dependent projection in computer games, it will make an entire game holographic thus providing more immersive and realistic gaming experience. The project can be enhanced by adding support for providing the illusion to multiple users. The algorithm can be improved by developing more sophisticated techniques for isolating the projection matrix. Also, multiple Kinects can be used in combination to cover large spatial tracking zone. As the technology develops, new possibilities will come forward, which replace the real world with the simulated one.

REFERENCES

[1] Sutherland, I. E. (1968, December). A head-mounted three dimensional display. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I* (pp. 757-764). ACM.

[2] MacKenzie, I. S., & Ware, C. (1993, May). Lag as a determinant of human performance in interactive systems. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems* (pp. 488-493). ACM.

[3] Rekimoto, J. (1995, March). A vision-based head tracker for fish tank virtual reality-VR without head gear. In *Virtual Reality Annual International Symposium, 1995. Proceedings.* (pp. 94-100). IEEE.

[4] Lee, J. (2007). Head tracking for desktop VR displays using the Wii remote. *Published online at http://www. cs. cmu. edu/johnny/projects/wii*.

[5] Cruz-Neira, C., Sandin, D. J., & DeFanti, T. A. (1993, September). Surround-screen projection-based virtual reality: the design and implementation of the CAVE. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (pp. 135-142). ACM.

[6] OpenNI Framework, Available online at http://www.openni.org.

[7] NITE Middleware, Available online at http://www.primesense.com/solutions/nitemiddleware.

[8] Official OpenGL website, http://www.opengl.org.

[9] Official Unity3D website, http://unity3d.com.

[10] Lee, Johnny Chung. "Hacking the nintendowii remote." IEEE pervasive computing 7, no. 3 (2008).

[11] TrackIR – Premium head tracking for gaming, official website http://www.naturalpoint.com/trackir.

[12] FaceAPI from Seeing Machines, Available online at http://www.seeingmachines.com/product/faceapi.

[13] Parvizi, E., & Wu, Q. J. (2007, October). Real-time 3d head tracking based on time-of-flight depth sensor. In *Tools with Artificial Intelligence, 2007. ICTAI 2007. 19th IEEE International Conference on* (Vol. 1, pp. 517-521). IEEE.

[14] Gokturk, S. B., & Tomasi, C. (2004, July). 3D head tracking based on recognition and interpolation using a time-of-flight depth sensor. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on* (Vol. 2, pp. II-II). IEEE.

[15] Moreno, F., Tarrida, A., Andrade-Cetto, J., & Sanfeliu, A. (2002). 3D real-time head tracking fusing color histograms and stereovision. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on* (Vol. 1, pp. 368-371). IEEE.

[16] Skewed frustum/off-axis projection for head tracking in OpenGL, Available at http://stackoverflow.com/questions/16723674/skewed-frustum-off-axis-projection-for-head-tracking-inopengl.

[17] OpenGL FAQ, Available online at http://www.opengl.org/wiki/FAQ.

[18] Djajadiningrat, J. P., & Gribnau, M. W. (1998). Desktop VR using QuickDraw 3D, Part I. *MacTech*, *14*(7), 32-43.

[19] Borenstein, G. (2012). *Making things see: 3D vision with kinect, processing, Arduino, and MakerBot*. " O'Reilly Media, Inc.