



SRM Institute of Science and Technology

Chennai Ramapuram

Faculty of Liberal Arts & Business Studies

(A Place for Transformation)

School of Applied Sciences



PG Department of Computer Applications (MCA & MCA GEN AI)

PRACTICAL RECORD

NAME :

REGISTER NUMBER :

COURSE : MCA

SEMESTER / YEAR : III / II

SECTION :

SUBJECT CODE : PCA25D08J

**SUBJECT NAME : AUGMENTED REALITY AND
VIRTUAL REALITY**

OCTOBER 2026



SRM Institute of Science and Technology
Chennai Ramapuram



Faculty of Liberal Arts & Business Studies

(A Place for Transformation)

School of Applied Sciences

PG Department of Computer Applications (MCA & MCA GEN AI)

REGISTER NUMBER:

BONAFIDE CERTIFICATE

This is to certify that the bonafide work done by _____
in the subject **AUGMENTED REALITY AND VIRTUAL REALITY [PCA25D08J]** at SRM
Institute of Science and Technology, Chennai Ramapuram in **OCTOBER 2026**.

STAFF IN-CHARGE

HEAD OF THE DEPARTMENT

Submitted for the University Practical Examination held at SRM Institute of Science and
Technology, Chennai Ramapuram on _____

INTERNAL EXAMINER 1

INTERNAL EXAMINER 2

INDEX

S.No	Date	Name of the Program	Page No	Faculty Signature
1		Study of different game engines		
2		Understand the use of AR Software and Toolkit		
3		Implementation on Video/ Feature viewing		
4		Installation of Unity and Visual Studio, setting up Unity for VR development, understanding documentation of the same.		
5		Demonstration of the working of HTC Vive, Google Cardboard, Google Daydream and Samsung gear VR		
6		Study the use of Virtual Reality at NASA		
7		Implementation on Virtual tour		
8		Sweeping coverage of eye movements		
9		Implementation on material animation		
10		Understand the Human Physiology and perception,		
11		Automatic stitching of panoramas in Virtual Reality		
12		Develop a simple UI (User Interface) menu with images, canvas, sprites and button.		
13		Create an immersive environment (living room/ battlefield/ tennis court) with only static game objects		
14		Implementation to show portal planets		
15		Demonstration of Digital Entertainment by VR.		

Ex. No: 01

Name:

Date:

Reg. No:

Study of Different Game Engines

Aim

To study various game engines used for game development, Virtual Reality (VR), and Augmented Reality (AR) applications, and understand their features, advantages, and applications.

Procedure

Step 1: Identify Popular Game Engines

1. Collect information about commonly used game engines.
2. Study their architecture and development environment.
3. Analyze supported platforms and programming languages.

Step 2: Study Unity Game Engine

1. Explore the Unity Editor interface.
2. Understand Game Objects, Components, and Assets.
3. Study VR and AR development support in Unity.

Step 3: Study Unreal Engine

1. Explore the Unreal Engine interface.
2. Understand Blueprint Visual Scripting.
3. Study high-quality graphics rendering features.

Step 4: Study Godot Engine

1. Examine the Godot development environment.
2. Study scene-based architecture.
3. Explore GDScript programming.

Step 5: Study CryEngine

1. Understand CryEngine's rendering capabilities.
2. Analyze its use in high-end gaming applications.
3. Study real-time environment creation features.

Step 6: Compare Game Engines

1. Compare ease of use, performance, and graphical quality.
2. Analyze VR and AR support.

- Record observations in a comparison table.

Program

This experiment is a study-oriented exercise; therefore, no programming code is required.

Comparison of Game Engines

Game Engine	Programming Language	VR/AR Support	Platform Support	Key Features
Unity	C#	Excellent	Windows, Android, iOS, Web, VR	Easy to learn, large community
Unreal Engine	C++, Blueprints	Excellent	Windows, Console, VR	High-quality graphics
Godot	GDScript, C#	Good	Windows, Linux, Android, iOS	Open source and lightweight
CryEngine	C++	Moderate	Windows, Console	Advanced rendering
GameMaker Studio	GML	Limited	Desktop, Mobile	Suitable for 2D games

Output

- Unity is widely used for VR and AR development due to its flexibility and extensive asset store.
- Unreal Engine provides superior graphical quality and realistic rendering.
- Godot is open-source and suitable for beginners and independent developers.
- CryEngine offers advanced graphics capabilities for high-end applications.
- Different game engines are selected based on project requirements, performance needs, and development complexity.

Comparative Analysis

Feature	Unity	Unreal	Godot	CryEngine
Ease of Learning	High	Medium	High	Medium
Graphics Quality	Good	Excellent	Good	Excellent
VR Support	Excellent	Excellent	Good	Moderate

Feature	Unity	Unreal	Godot	CryEngine
Community Support	Excellent	Excellent	Good	Moderate

Result

Thus, the study of different game engines was successfully completed. The features, advantages, limitations, and applications of Unity, Unreal Engine, Godot, and CryEngine were analyzed and compared. It was observed that Unity and Unreal Engine are the most widely used platforms for developing modern VR and AR applications.

Ex. No: 02

Name:

Date:

Reg. No:

Understand the Use of AR Software and Toolkit

Aim

To study and understand the features, components, and applications of Augmented Reality (AR) software and toolkits used for developing AR-based applications.

Procedure

Step 1: Study the Basics of Augmented Reality

1. Understand the concept of Augmented Reality.
2. Learn the differences between AR, VR, and Mixed Reality.
3. Identify real-world applications of AR technology.

Step 2: Explore AR Development Software

1. Install Unity 3D.
2. Explore the Unity interface and project structure.
3. Understand how Unity supports AR application development.

Step 3: Study Popular AR Toolkits

1. Examine Vuforia Engine.
2. Study ARCore by Google.
3. Study ARKit by Apple.
4. Explore Wikitude and EasyAR toolkits.
5. Analyze the features of each toolkit.

Step 4: Understand AR Components

1. Study AR Camera functionality.
2. Learn about Image Targets and Marker Tracking.
3. Understand Object Tracking and Plane Detection.
4. Explore Environmental Understanding and Motion Tracking.

Step 5: Create a Basic AR Setup

1. Open Unity and create a new project.
2. Import an AR toolkit such as Vuforia.

3. Add an AR Camera and Image Target.
4. Place a 3D object on the target image.
5. Run the application and observe the AR content.

Step 6: Analyze AR Toolkit Features

1. Compare tracking accuracy.
2. Evaluate platform compatibility.
3. Analyze ease of development and deployment.
4. Document observations.

Program

This experiment is primarily a study and setup exercise. However, a simple AR object activation script can be used.

ARObjectDisplay.cs

using UnityEngine;

```
public class ARObjectDisplay : MonoBehaviour
{
    public GameObject arObject;

    void Start()
    {
        arObject.SetActive(true);
    }
}
```

Steps to Execute

1. Create a 3D object in Unity.
2. Attach the script to the Image Target.
3. Assign the 3D object in the Inspector.
4. Run the AR application.
5. Observe the object appearing when the target is detected.

Output

Observations

- Unity provides an effective environment for AR application development.
- Vuforia supports image target and object tracking.
- ARCore provides motion tracking and environmental understanding for Android devices.

- ARKit offers advanced AR capabilities for iOS devices.
- AR toolkits enable real-time interaction between virtual and physical environments.

Comparison of AR Toolkits

Toolkit	Platform	Tracking Features	Key Applications
Vuforia	Android, iOS	Image & Object Tracking	Education, Marketing
ARCore	Android	Motion Tracking, Plane Detection	Mobile AR Apps
ARKit	iOS	Face Tracking, Plane Detection	iOS AR Applications
Wikitude	Android, iOS	Image Recognition	Enterprise AR
EasyAR	Android, iOS	Image Tracking	AR Learning Applications

Result

Thus, the use of various **AR Software and Toolkits** was successfully studied. The functionalities of Unity, Vuforia, ARCore, ARKit, Wikitude, and EasyAR were analyzed, and their roles in developing interactive Augmented Reality applications were understood.

Ex. No: 03

Name:

Date:

Reg. No:

Implementation of Video / Feature Viewing in AR

Aim

To develop an Augmented Reality (AR) application that displays and plays video content or multimedia features when a predefined image target is detected using Unity and Vuforia.

Procedure

Step 1: Create a New Unity Project

1. Open Unity Hub.
2. Create a new 3D project.
3. Save the project with an appropriate name.

Step 2: Import Vuforia Engine

1. Enable Vuforia Engine in Unity.
2. Configure the Vuforia License Key.

Step 3: Add AR Camera

1. Remove the default Main Camera.
2. Add an AR Camera from the Vuforia menu.

Step 4: Create Image Target

1. Add an Image Target object.
2. Import the target image database.
3. Select the desired marker image.

Step 5: Add Video Content

1. Import an MP4 video into the Assets folder.
2. Create a Plane as a child of the Image Target.
3. Add a Video Player component to the Plane.

Step 6: Configure Video Playback

1. Assign the imported video to the Video Player.
2. Enable looping if required.
3. Set the video to play when the image target is detected.

Step 7: Build and Deploy

1. Switch the platform to Android.
2. Build the APK file.
3. Install the application on a mobile device.

Step 8: Test the Application

1. Launch the application.
2. Point the camera toward the target image.
3. Observe the video playback on the detected image target.

Program

PlayVideo.cs

```
using UnityEngine;
using UnityEngine.Video;

public class PlayVideo : MonoBehaviour
{
    public VideoPlayer videoPlayer;

    void Start()
    {
        videoPlayer.Play();
    }
}
```

Steps to Attach the Script

1. Create a C# script named **PlayVideo.cs**.
2. Copy and paste the above code.
3. Attach the script to the Plane object.
4. Assign the Video Player component in the Inspector.

Output

Before Detection

- The camera scans the environment.
- No video content is displayed.

After Detection

- The image target is recognized.
- The video automatically plays on the target image.
- Users can view multimedia content in the AR environment.

Result

Thus, the implementation of **Video/Feature Viewing in Augmented Reality** was successfully completed using Unity and Vuforia. The application accurately detected the image target and displayed the associated video content in real time.

Ex. No: 04

Name:

Date:

Reg. No:

Installation of Unity and Visual Studio, Setting Up Unity for VR Development, and Understanding Documentation

Aim

To install Unity and Visual Studio, configure Unity for Virtual Reality (VR) development, and understand the documentation and tools required for VR application development.

Procedure

Step 1: Install Unity Hub

1. Download Unity Hub from the official Unity website.
2. Run the installer and complete the installation.
3. Launch Unity Hub and sign in with a Unity account.

Step 2: Install Unity Editor

1. Open Unity Hub.
2. Navigate to **Installs** → **Install Editor**.
3. Select the latest Long-Term Support (LTS) version.
4. Include the following modules:
 - Android Build Support
 - Android SDK & NDK Tools
 - OpenJDK
 - Windows Build Support (optional)

Step 3: Install Visual Studio

1. Download Visual Studio Community Edition.
2. Run the installer.
3. Select the **Game Development with Unity** workload.
4. Complete the installation.

Step 4: Create a New Unity Project

1. Open Unity Hub.
2. Click **New Project**.
3. Select the 3D template.

4. Enter a project name and location.
5. Click **Create Project**.

Step 5: Configure Unity for VR Development

1. Open the Unity project.
2. Go to **Edit** → **Project Settings** → **XR Plug-in Management**.
3. Install XR Plug-in Management if prompted.
4. Enable the required VR platform:
 - OpenXR
 - Oculus XR Plugin
 - Windows XR Plugin (if applicable)

Step 6: Install XR Interaction Toolkit

1. Open **Window** → **Package Manager**.
2. Search for **XR Interaction Toolkit**.
3. Install the package.
4. Import starter assets if required.

Step 7: Set Up a Basic VR Scene

1. Add an XR Origin to the scene.
2. Add a Floor (Plane).
3. Add simple 3D objects (Cube, Sphere).
4. Configure interaction settings.

Step 8: Understand Unity Documentation

1. Open Unity Documentation.
2. Study:
 - Unity Interface
 - Scene View
 - Game View
 - Inspector Window
 - Hierarchy Window
 - XR Interaction Toolkit Documentation
3. Review VR development workflow and best practices.

Step 9: Test the VR Environment

1. Connect a compatible VR headset (if available).
2. Run the scene.
3. Verify movement and interaction functionality.

Program

Simple Object Rotation Script

using UnityEngine;

```
public class RotateObject : MonoBehaviour
{
    void Update()
    {
        transform.Rotate(0, 50 * Time.deltaTime, 0);
    }
}
```

Steps

1. Create a C# script named **RotateObject.cs**.
2. Attach it to a Cube or Sphere.
3. Run the scene to observe continuous rotation.

Output

- Unity Hub and Unity Editor are successfully installed.
- Visual Studio is integrated with Unity.
- XR Plug-in Management and XR Interaction Toolkit are configured.
- A basic VR scene is created and executed.
- The 3D object rotates continuously within the scene.
- Documentation resources are explored and understood.

Result

Thus, Unity and Visual Studio were successfully installed and configured for VR development. A basic VR environment was created, essential VR packages were integrated, and the relevant Unity documentation was studied to understand the VR development workflow.

Ex. No: 05

Name:

Date:

Reg. No:

Demonstration of the Working of HTC Vive, Google Cardboard, Google Daydream, and Samsung Gear VR

Aim

To study and demonstrate the working principles, setup procedures, features, and user interactions of HTC Vive, Google Cardboard, Google Daydream, and Samsung Gear VR virtual reality devices.

Procedure

HTC Vive

1. Connect the HTC Vive headset to a VR-ready computer.
2. Install SteamVR software.
3. Set up the base stations for room-scale tracking.
4. Pair the controllers with the headset.
5. Launch a VR application.
6. Observe head tracking, hand tracking, and movement within the virtual environment.

Google Cardboard

1. Install a Google Cardboard-compatible VR application on a smartphone.
2. Insert the smartphone into the Cardboard viewer.
3. Launch the VR application.
4. Observe stereoscopic 3D visualization.
5. Navigate using head movements and the Cardboard trigger button.

Google Daydream

1. Install a Daydream-supported application on a compatible smartphone.
2. Insert the smartphone into the Daydream headset.
3. Pair the Daydream controller.
4. Launch the VR application.
5. Observe controller-based interaction and motion tracking.

Samsung Gear VR

1. Install the Oculus application on a compatible Samsung smartphone.

2. Connect the smartphone to the Gear VR headset.
3. Complete the device setup process.
4. Launch a VR application.
5. Observe immersive virtual reality experiences using head tracking and touchpad controls.

Comparative Observation

1. Compare tracking accuracy among the devices.
2. Analyze user comfort and ease of use.
3. Observe differences in interaction methods and display quality.
4. Record observations regarding performance and immersion.

Program

Since this experiment is a hardware demonstration and study exercise, no programming code is required.

Sample Observation Table

VR Device	Tracking Type	Controller Support	Immersion Level
HTC Vive	Room-scale Tracking	Yes	High
Google Cardboard	Head Tracking	No	Basic
Google Daydream	Motion Tracking	Yes	Moderate
Samsung Gear VR	Head Tracking	Touchpad/Controller	Moderate

Output

- HTC Vive provides immersive room-scale virtual reality with precise tracking.
- Google Cardboard offers a low-cost VR experience using a smartphone.
- Google Daydream supports controller-based interaction and improved user experience.
- Samsung Gear VR delivers immersive VR through smartphone integration and Oculus support.
- Differences in tracking, interaction, display quality, and immersion are observed and documented.

Result

Thus, the working of **HTC Vive, Google Cardboard, Google Daydream, and Samsung Gear VR** was successfully demonstrated and studied. The features, tracking mechanisms, user interactions, and immersive capabilities of each VR device were analyzed and compared.

Ex. No: 06

Name:

Date:

Reg. No:

Study the Use of Virtual Reality at NASA

Aim

To study documented applications of virtual reality in NASA-related training, visualization, and mission preparation.

Procedure

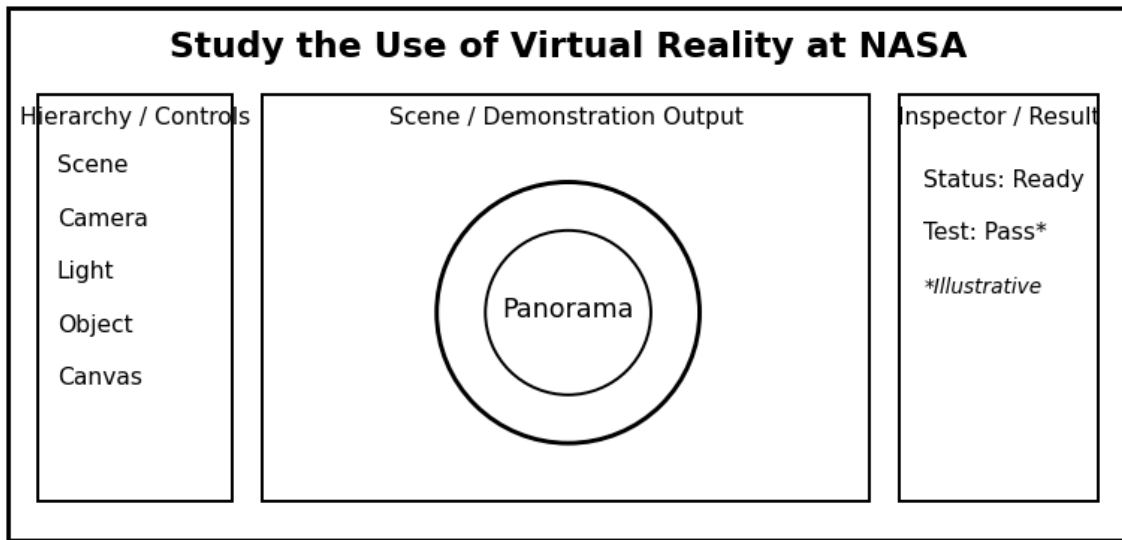
1. Review NASA educational or technical material describing VR applications.
2. Identify the task supported by VR, such as simulation, visualization, or training.
3. Note the user interaction, virtual environment, and expected benefit.
4. Prepare a short case-study summary with source details.

Code

```
// Case-study notes template  
string application = "NASA VR case study";  
string task = "Describe the task supported by VR";  
string interaction = "Describe user interaction";  
Debug.Log(application + " | " + task + " | " + interaction);
```

Output

Expected output: a one-page case-study summary identifying the application, users, interaction, and cited source.



Result

VR can support simulation and visualization in aerospace contexts. Specific claims should be tied to the NASA material reviewed.

Ex. No: 07

Name:

Date:

Reg. No:

Implementation of a Virtual Tour

Aim

To develop a Virtual Reality (VR)-based Virtual Tour application that allows users to navigate and explore a virtual environment interactively using Unity.

Procedure

Step 1: Create a New Unity Project

1. Open Unity Hub.
2. Create a new 3D project named **Virtual Tour**.
3. Save the project.

Step 2: Create the Environment

1. Add a Plane as the ground.
2. Add 3D objects such as buildings, trees, chairs, and pathways.
3. Arrange the objects to represent a campus, museum, or tourist location.

Step 3: Add First Person Controller

1. Open Package Manager and install Starter Assets (if required).
2. Add a First Person Controller to the scene.
3. Position the controller at the starting point.

Step 4: Create Tour Points

1. Create multiple empty GameObjects named:
 - TourPoint1
 - TourPoint2
 - TourPoint3
2. Place them at different locations in the virtual environment.

Step 5: Create Navigation Script

1. Create a C# script named **VirtualTour.cs**.
2. Attach the script to the Main Camera or Player object.
3. Assign tour point positions.

Step 6: Run the Application

1. Save the scene.
2. Click Play.
3. Navigate through the virtual environment using keyboard controls.

Program Code

VirtualTour.cs

using UnityEngine;

```
public class VirtualTour : MonoBehaviour
{
    public Transform[] tourPoints;
    private int currentPoint = 0;

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space))
        {
            currentPoint++;

            if (currentPoint >= tourPoints.Length)
                currentPoint = 0;

            transform.position = tourPoints[currentPoint].position;
            transform.rotation = tourPoints[currentPoint].rotation;
        }
    }
}
```

Steps to Execute

1. Create three or more empty GameObjects as tour points.
2. Attach the script to the Main Camera.
3. Drag all tour points into the **Tour Points** array in the Inspector.
4. Press **Play**.
5. Press **Spacebar** to move between tour locations.

Output

Initial View

- User starts at the first location of the virtual environment.

During Navigation

- Pressing the Spacebar moves the user to the next tour point.
- Different areas of the virtual environment are displayed.
- Users can explore the environment interactively.

Sample Output

Key Press	Action
Start	Displays first location
Spacebar	Moves to Tour Point 2
Spacebar	Moves to Tour Point 3
Spacebar	Returns to Tour Point 1

Result

Thus, a **Virtual Tour Application** was successfully implemented using Unity. The application enabled users to navigate between different locations within a virtual environment, providing an immersive and interactive tour experience.

Ex. No: 08

Name:

Date:

Reg. No:

Sweeping Coverage of Eye Movements

Aim

To study and implement the tracking and visualization of eye movements in a Virtual Reality (VR) environment to understand user attention, gaze direction, and interaction behavior.

Procedure

Step 1: Create a New Unity Project

1. Open Unity Hub and create a new 3D project.
2. Save the project as **Eye Movement Tracking**.

Step 2: Set Up the VR Environment

1. Add a Camera to represent the user's viewpoint.
2. Create several objects (Cube, Sphere, Cylinder) in different positions.
3. Arrange objects within the user's field of view.

Step 3: Simulate Eye/Gaze Tracking

1. Use the center of the camera as the gaze direction.
2. Perform a Raycast from the camera forward direction.
3. Detect which object the user is looking at.

Step 4: Create a Gaze Detection Script

1. Create a C# script named **EyeTracking.cs**.
2. Attach the script to the Main Camera.
3. Configure gaze detection using Raycasting.

Step 5: Run the Application

1. Enter Play Mode.
2. Move the camera toward different objects.
3. Observe the detected gaze target in the Console.

Step 6: Analyze Eye Movements

1. Record the sequence of viewed objects.
2. Measure gaze duration on each object.
3. Study user attention patterns.

Program Code

EyeTracking.cs

```
using UnityEngine;

public class EyeTracking : MonoBehaviour
{
    void Update()
    {
        Ray ray = new Ray(transform.position, transform.forward);
        RaycastHit hit;

        if (Physics.Raycast(ray, out hit))
        {
            Debug.Log("Looking at: " + hit.collider.gameObject.name);
        }
    }
}
```

Steps to Execute

1. Create multiple 3D objects in the scene.
2. Attach the script to the Main Camera.
3. Ensure objects have Collider components.
4. Run the scene.
5. Observe gaze detection messages in the Console window.

Output

Sample Console Output

```
Looking at: Cube
Looking at: Sphere
Looking at: Cylinder
Looking at: Cube
```

Observations

- The system detects the object currently being viewed.
- Eye/gaze movement changes are tracked in real time.
- User attention can be analyzed based on viewed objects.

- Sweeping eye movement patterns across multiple objects can be recorded.

Result

Thus, the **Sweeping Coverage of Eye Movements** experiment was successfully implemented using Unity. The gaze direction of the user was tracked using raycasting, enabling analysis of visual attention and eye movement behavior within a virtual environment.

Ex. No: 09

Name:

Date:

Reg. No:

Implementation of Material Animation

Aim

To implement material animation in Unity by dynamically changing the material properties of a 3D object and creating visual effects in a Virtual Reality environment.

Procedure

Step 1: Create a New Unity Project

1. Open Unity Hub and create a new 3D project.
2. Save the project as **Material Animation**.

Step 2: Create a Scene

1. Add a Plane to act as the ground.
2. Add a Cube or Sphere to the scene.
3. Position the object appropriately.

Step 3: Create a Material

1. In the Assets folder, select **Create → Material**.
2. Name the material **AnimatedMaterial**.
3. Assign the material to the Cube.

Step 4: Create Animation Script

1. Create a new C# script named **MaterialAnimation.cs**.
2. Write code to change the material color continuously.
3. Attach the script to the Cube.

Step 5: Configure the Object

1. Select the Cube.
2. Assign the material in the Inspector.
3. Attach the script component.

Step 6: Run the Application

1. Save the scene.
2. Click the **Play** button.
3. Observe the material color changing continuously.

Program Code

MaterialAnimation.cs

```
using UnityEngine;

public class MaterialAnimation : MonoBehaviour
{
    private Renderer objectRenderer;

    void Start()
    {
        objectRenderer = GetComponent<Renderer>();
    }

    void Update()
    {
        float value = Mathf.PingPong(Time.time, 1);
        objectRenderer.material.color = new Color(value, 0, 1 - value);
    }
}
```

Steps to Execute

1. Create a Cube in the scene.
2. Create and assign a material to the Cube.
3. Attach the **MaterialAnimation.cs** script to the Cube.
4. Run the scene.
5. Observe the material color smoothly changing over time.

Output

Initial State

- The Cube appears with the assigned material color.

During Execution

- The Cube's material color gradually changes between different shades.
- The animation runs continuously in real time.

- The visual effect enhances realism and interactivity.

Sample Observation

Time (Seconds)	Material Color
0	Blue
2	Purple
4	Red
6	Purple
8	Blue

Result

Thus, the **Material Animation** experiment was successfully implemented using Unity. The material properties of a 3D object were dynamically modified, creating a smooth animation effect and demonstrating the use of materials in interactive virtual environments.

Ex. No: 10

Name:

Date:

Reg. No:

Understand Human Physiology and Perception

Aim

To study the principles of human physiology and perception relevant to Virtual Reality (VR) and Augmented Reality (AR) systems, including vision, hearing, motion perception, and sensory integration.

Procedure

Step 1: Study Human Visual System

1. Understand the structure of the human eye.
2. Study binocular vision and depth perception.
3. Learn about field of view (FOV) and peripheral vision.
4. Analyze color perception and visual acuity.

Step 2: Study Auditory Perception

1. Understand the anatomy of the human ear.
2. Learn how humans perceive sound direction and distance.
3. Study the role of spatial audio in immersive environments.

Step 3: Study Vestibular System

1. Understand the role of the vestibular system in balance and motion sensing.
2. Examine how the brain processes movement information.
3. Analyze the causes of motion sickness in VR environments.

Step 4: Study Sensory Integration

1. Learn how visual, auditory, and motion cues are combined.
2. Understand human perception of virtual environments.
3. Study the relationship between sensory feedback and immersion.

Step 5: Observe Perception in VR

1. Wear a VR headset (if available).
2. Observe depth, distance, and object movement.
3. Identify factors affecting user comfort and immersion.
4. Record observations.

Step 6: Analyze Findings

1. Compare real-world perception with virtual perception.
2. Identify physiological limitations in VR systems.
3. Document observations and conclusions.

Program

This experiment is a theoretical and observational study; therefore, no programming code is required.

Sample Observation Table

Physiological Aspect	Function	Importance in VR/AR
Vision	Object recognition and depth perception	Creates realistic 3D environments
Hearing	Sound localization	Enhances immersion
Vestibular System	Balance and motion sensing	Supports movement perception
Touch (Haptics)	Physical interaction feedback	Improves realism
Cognitive Processing	Interpretation of sensory inputs	Determines user experience

Output

Observations

- Human vision plays a critical role in depth perception and spatial awareness.
- Spatial audio enhances immersion and realism in VR applications.
- The vestibular system contributes to balance and motion perception.
- Sensory conflicts between visual and vestibular cues may cause VR motion sickness.
- Proper integration of sensory information improves user comfort and presence.

Factors Affecting VR Perception

- Field of View (FOV)
- Refresh Rate
- Latency
- Resolution
- Spatial Audio Quality and Head Tracking Accuracy

Result

Thus, the study of **Human Physiology and Perception** was successfully completed. The functions of the visual, auditory, vestibular, and sensory systems were analyzed, and their significance in designing effective and immersive Virtual Reality and Augmented Reality applications was understood.

Ex. No: 11

Name:

Date:

Reg. No:

Automatic Stitching of Panoramas in Virtual Reality

Aim

To implement automatic panorama image stitching and visualize the resulting 360-degree panoramic image in a Virtual Reality environment.

Procedure

Step 1: Collect Images

1. Capture multiple overlapping photographs of a scene.
2. Ensure at least 30–50% overlap between adjacent images.
3. Store all images in a project folder.

Step 2: Create a Unity Project

1. Open Unity Hub.
2. Create a new 3D project named **Panorama Stitching VR**.
3. Save the project.

Step 3: Generate Panorama

1. Use panorama stitching software (such as OpenCV or Microsoft ICE).
2. Load the overlapping images.
3. Detect common feature points.
4. Match features between images.
5. Automatically stitch the images into a panoramic image.
6. Save the output panorama.

Step 4: Import Panorama into Unity

1. Import the stitched panoramic image into the Assets folder.
2. Create a Sphere in the scene.
3. Scale the Sphere to **(-1, 1, 1)** to view the texture from inside.

Step 5: Create Panorama Material

1. Create a new Material.
2. Set Shader to **Skybox/Panoramic** or apply the panorama texture.
3. Assign the panorama image to the material.

4. Apply the material to the inside of the sphere.

Step 6: Add VR Camera

1. Add XR Origin or Main Camera inside the sphere.
2. Position the camera at the center.
3. Configure VR settings.

Step 7: Run the Application

1. Click Play.
2. Move the camera or VR headset.
3. Observe the 360-degree panoramic environment.

Program Code

PanoramaViewer.cs

using UnityEngine;

```
public class PanoramaViewer : MonoBehaviour
{
    public float rotationSpeed = 10f;

    void Update()
    {
        float horizontal = Input.GetAxis("Horizontal");
        transform.Rotate(0, horizontal * rotationSpeed * Time.deltaTime, 0);
    }
}
```

Steps to Execute

1. Create a Sphere and apply the panorama texture.
2. Place the Camera inside the Sphere.
3. Attach the script to the Camera.
4. Run the application.
5. Use arrow keys to explore the panoramic scene.

Output

Before Stitching

- Multiple overlapping images of a scene are available.

After Stitching

- A single panoramic image is generated.
- The panorama covers a wide field of view.

In Virtual Reality

- Users can view the environment in 360 degrees.
- Head movement or camera rotation allows exploration of the entire panorama.
- An immersive virtual tour experience is achieved.

Sample Observation Table

Activity	Observation
Feature Detection	Successful
Image Matching	Accurate
Panorama Generation	Completed
VR Visualization	360° View Obtained
User Experience	Immersive

Result

Thus, the **Automatic Stitching of Panoramas in Virtual Reality** experiment was successfully implemented. Multiple overlapping images were stitched into a panoramic image and visualized in a VR environment, providing an immersive 360-degree viewing experience.

Ex. No: 12

Name:

Date:

Reg. No:

Develop a Simple UI (User Interface) Menu with Images, Canvas, Sprites, and Buttons

Aim:

To design and develop a simple User Interface (UI) menu in Unity using Canvas, Images, Sprites, and Buttons for interactive Virtual Reality (VR) and Augmented Reality (AR) applications.

Procedure:

Step 1: Create a New Unity Project

1. Open Unity Hub.
2. Create a new 3D project named **UI Menu Design**.
3. Save the project.

Step 2: Create a Canvas

1. Right-click in the Hierarchy window.
2. Select **UI → Canvas**.
3. Ensure the Canvas contains an Event System automatically.

Step 3: Add Background Image

1. Import an image into the Assets folder.
2. Right-click Canvas → **UI → Image**.
3. Assign the imported image as the Source Image.
4. Resize the image to fit the screen.

Step 4: Add UI Text

1. Right-click Canvas → **UI → Text (TextMeshPro)**.
2. Enter the title "VR Application Menu".
3. Adjust font size and position.

Step 5: Add Buttons

1. Right-click Canvas → **UI → Button**.
2. Rename buttons as:
 - Start
 - Settings

- Exit
- 3. Edit button text accordingly.
- 4. Arrange buttons vertically.

Step 6: Add Sprites

1. Import sprite images.
2. Assign sprites to buttons or decorative UI elements.
3. Adjust sprite size and alignment.

Step 7: Create Button Script

1. Create a C# script named **MenuController.cs**.
2. Attach the script to an empty GameObject.
3. Connect button events through the Inspector.

Step 8: Run the Application

1. Save the scene.
2. Click Play.
3. Test button interactions.

Program Code

MenuController.cs

```
using UnityEngine;

public class MenuController : MonoBehaviour
{
    public void StartApplication()
    {
        Debug.Log("Start Button Clicked");
    }

    public void OpenSettings()
    {
        Debug.Log("Settings Button Clicked");
    }

    public void ExitApplication()
    {
        Debug.Log("Exit Button Clicked");
        Application.Quit();
    }
}
```

$\left. \begin{array}{l} \\ \end{array} \right\}$

Steps to Execute

1. Create a **GameObject** named **MenuManager**.
2. Attach **MenuController.cs** to it.
3. Select each button.
4. Add an **OnClick()** event.
5. Assign the appropriate function:
 - Start → **StartApplication()**
 - Settings → **OpenSettings()**
 - Exit → **ExitApplication()**

Output

User Interface Layout

VR APPLICATION MENU	
[Start]	
[Settings]	
[Exit]	

Console Output

```
Start Button Clicked
Settings Button Clicked
Exit Button Clicked
```

Observations

UI Component	Function
Canvas	Holds all UI elements

UI Component	Function
Image	Displays background
Sprite	Enhances visual appearance
Button	User interaction
Text	Displays menu information

Result

Thus, a simple **User Interface Menu** was successfully developed using **Canvas, Images, Sprites, and Buttons** in Unity. The menu provided interactive navigation through button clicks and demonstrated the basic principles of UI design for VR and AR applications.

Ex. No: 13

Name:

Date:

Reg. No:

Create an Immersive Environment with Static Game Objects

Aim

To construct a simple immersive environment using static objects such as a living room, battlefield, or tennis court.

Procedure

1. Create a new 3D scene and select an environment theme.
2. Add floor, walls, lighting, and static props from primitives or approved assets.
3. Arrange objects to establish scale, composition, and navigable space.
4. Add a first-person or XR camera and inspect the scene.

Code:

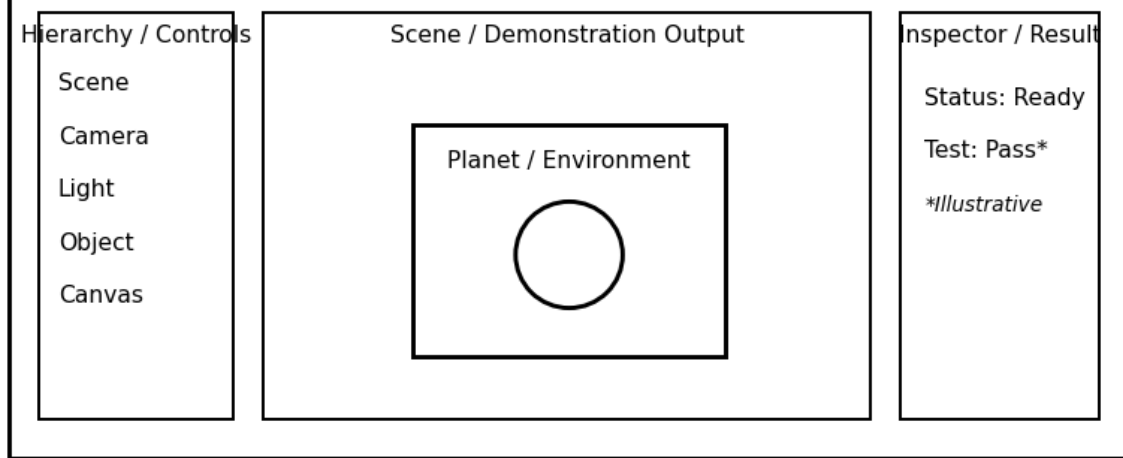
// Unity C#: place a static environment object at a fixed location
using UnityEngine;

```
public class EnvironmentSetup : MonoBehaviour {  
    public GameObject propPrefab;  
    public Vector3 placement = new Vector3(2, 0, 3);  
  
    void Start() {  
        if (propPrefab != null)  
            Instantiate(propPrefab, placement, Quaternion.identity);  
    }  
}
```

Output

Expected output: a themed 3D environment populated with static objects and visible from the scene camera.

Create an Immersive Environment with Static Game Object



Result

Static objects establish the environment layout. For performance, mark suitable non-moving geometry as static and configure lighting appropriately.

Ex. No: 14

Name:

Date:

Reg. No:

Implementation to Show Portal Planets

Aim

To create a simple visual portal effect that reveals a planet-like scene.

Procedure

5. Create a planet sphere with a material and lighting.
6. Create a portal frame using a ring or other mesh.
7. Place a camera-facing panel or planet view inside the frame.
8. Add a simple animation or interaction to reveal the planet scene.

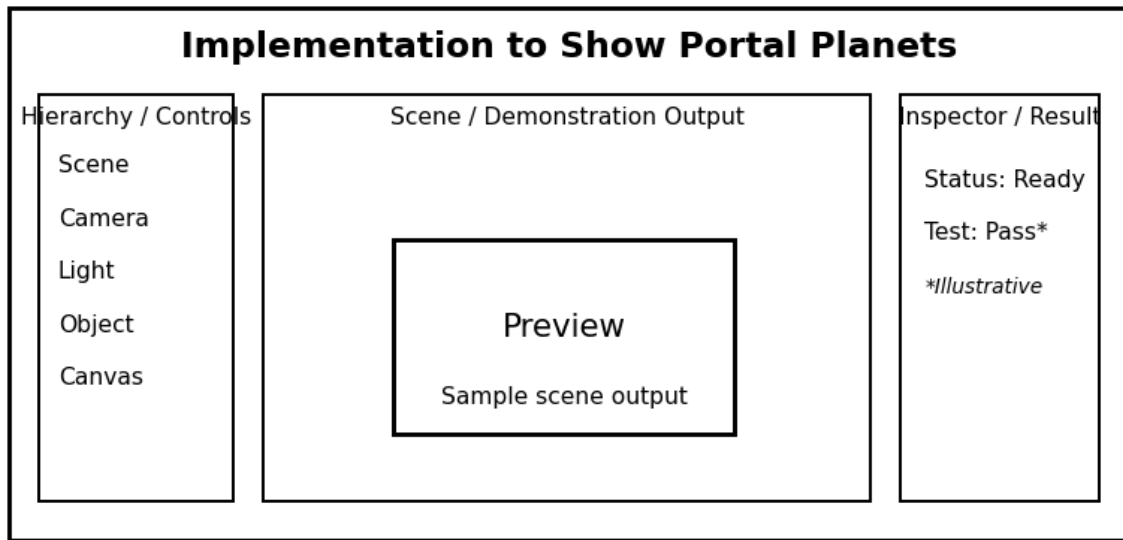
Code

```
// Unity C#: rotate a planet slowly
using UnityEngine;

public class PlanetSpin : MonoBehaviour {
    public float degreesPerSecond = 12f;
    void Update() {
        transform.Rotate(Vector3.up,
                        degreesPerSecond * Time.deltaTime,
                        Space.World);
    }
}
```

Output

Expected output: a planet is visible within or beyond a portal frame and rotates slowly.



Result

The exercise demonstrates a visual portal concept. A true camera-through-portal effect requires additional rendering and camera setup.

Ex. No: 15

Name:

Date:

Reg. No:

Demonstration of Digital Entertainment by VR

Aim

To explore how VR can be used for digital entertainment and demonstrate an interactive scene.

Procedure

1. Review examples of VR entertainment such as interactive environments, games, and immersive media.
2. Open a prepared VR scene or simulator and identify its interaction controls.
3. Perform a short interaction task and record user feedback on navigation and immersion.
4. Summarize the experience and note comfort and accessibility considerations.

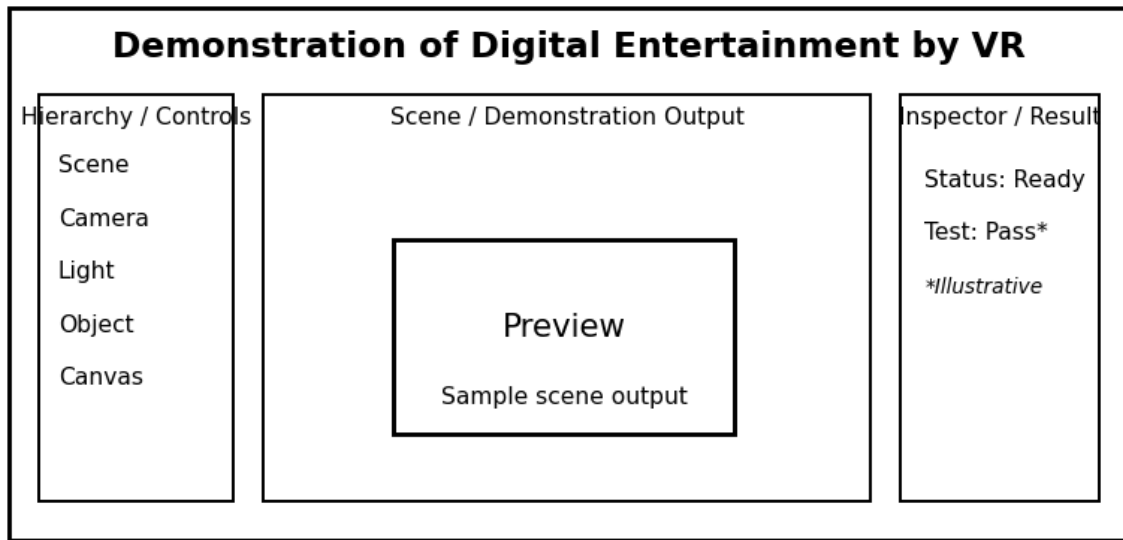
Code

```
// Unity C#: simple interaction feedback  
using UnityEngine;
```

```
public class InteractionFeedback : MonoBehaviour {  
    public void OnSelect() {  
        Debug.Log("Interactive object selected.");  
    }  
}
```

Output

Expected output: an interactive VR entertainment scene or simulator demonstration, with a brief observation record.



Result

VR entertainment combines immersive presentation and user interaction. The demonstration should use suitable hardware or a clearly identified simulator.