Ex no 1 Setting Up Unity for AR/VR Development

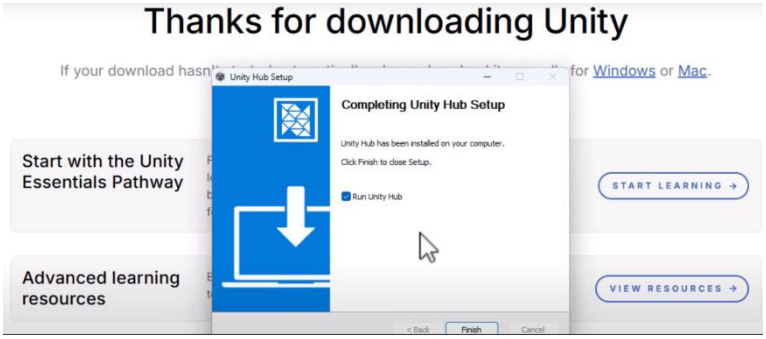
Aim:

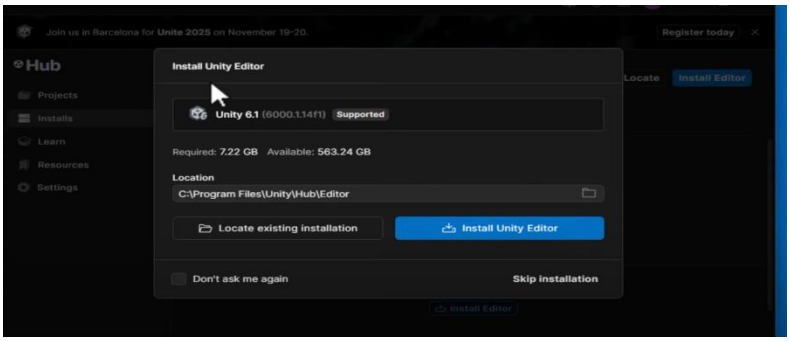
To install Unity and set up a basic AR/VR development environment with necessary packages and configurations.

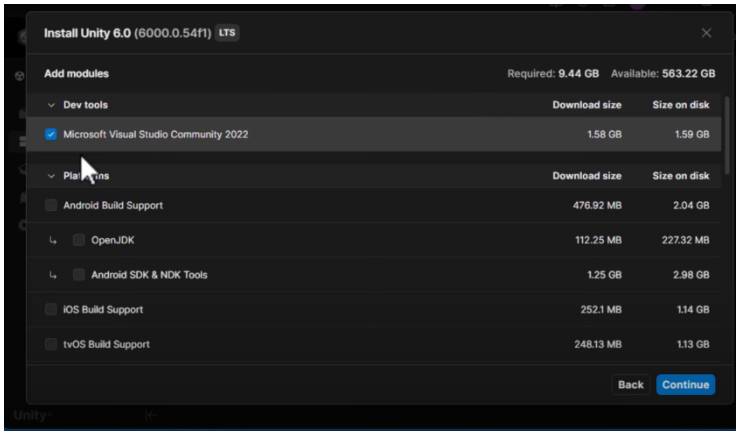
Procedure:

1. Download and install **Unity Hub** from the official Unity website.
2. In Unity Hub, install a Unity Editor version that supports AR/VR (e.g., Unity 2022 LTS or newer).
3. Open Unity and create a **New 3D Project**.
4. Go to **Window > Package Manager**, install:
    - AR Foundation
    - ARCore XR Plugin / ARKit XR Plugin
    - XR Interaction Toolkit
    - Vuforia Engine (if using image targets)
5. Configure **Build Settings**:
    - Switch platform to **Android** or **Windows**.
    - Enable **XR Plugin Management** under Project Settings.
6. Verify successful setup by running a simple scene.

OUTPUT:


Thanks for downloading Unity
Unity Hub Setup
Completing Unity Hub Setup
Unity Hub has been installed on your computer.
Click Finish to close Setup.
Run Unity Hub
Start with the Unity Essentials Pathway
START LEARNING
Advanced learning resources
VIEW RESOURCES
< Back
Finish
Cancel


Hub
Projects
Installs
Learn
Resources
Settings
Install Unity Editor
Unity 6.1 (6000.1.14f1)
Supported
Required: 7.22 GB Available: 563.24 GB
Location
C:\Program Files\Unity\Hub\Editor
Locate existing installation
Install Unity Editor
Don't ask me again
Skip installation


Install Unity 6.0 (6000.0.54f1) LTS
Add modules
Required: 9.44 GB Available: 563.22 GB
Dev tools
Download size
Size on disk
Microsoft Visual Studio Community 2022
1.58 GB
1.59 GB
Android Build Support
476.92 MB
2.04 GB
OpenJDK
112.25 MB
227.32 MB
Android SDK & NDK Tools
1.25 GB
2.98 GB
iOS Build Support
252.1 MB
1.14 GB
tvOS Build Support
248.13 MB
1.13 GB
Back
Continue

Result:

Unity environment is successfully configured for AR/VR development.

Ex no: 2 Design and Develop a 3D AR/VR Prototype

Aim:

To design and develop a simple AR/VR scene demonstrating spatial interaction and environment rendering.

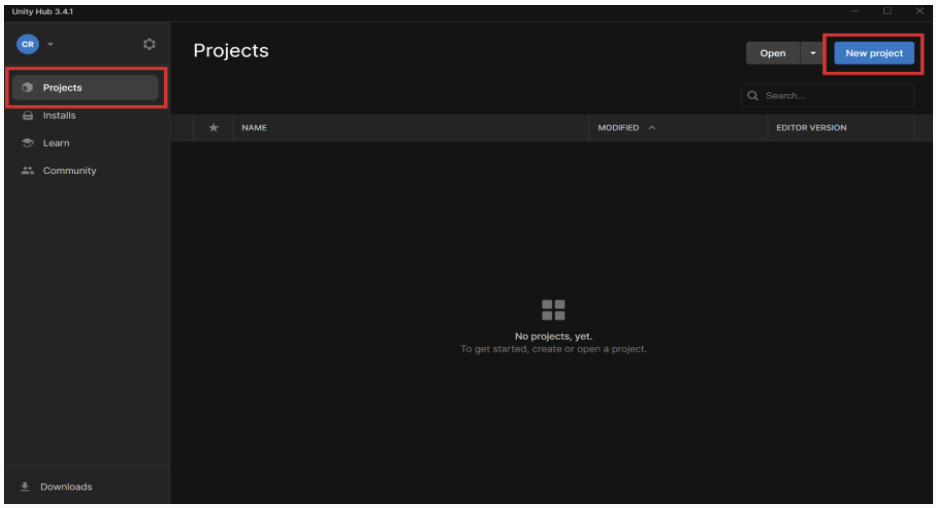
Procedure:

1. Create a New Unity Project:
    a. Open Unity Hub and click "New Project."
    b. Select a 3D template and give your project a name.
2. Create a Cube:
    a. In the Hierarchy window, right-click and choose "3D Object" > "Cube." This creates a default cube in your scene.
3. Position and Scale (Optional):
    a. Select the cube in the Hierarchy.
    b. In the Inspector window, adjust the "Transform" component's "Position" and "Scale" values to place and size the cube as desired.
4. Add a Material (Optional):
    a. In the Project window, right-click and choose "Create" > "Material."
    b. Select the new material and choose a color or assign a texture in the Inspector.
    c. Drag the material from the Project window onto the cube in the Scene or Hierarchy.
5. Add a Camera and Light:
    a. Unity projects automatically include a Main Camera and a Directional Light. You can adjust their positions and properties in the Inspector if needed.
6. Create a C# Script (Optional for Interaction):
    a. In the Project window, right-click and choose "Create" > "C# Script."
    b. Give it a name (e.g., CubeController).
    c. Double-click the script to open it in your IDE (e.g., Visual Studio).
    d. Add C# code within the Start() or Update() methods to control the cube's behavior (e.g., rotation, movement).
    e. Drag the script from the Project window onto your cube in the Hierarchy to attach it.

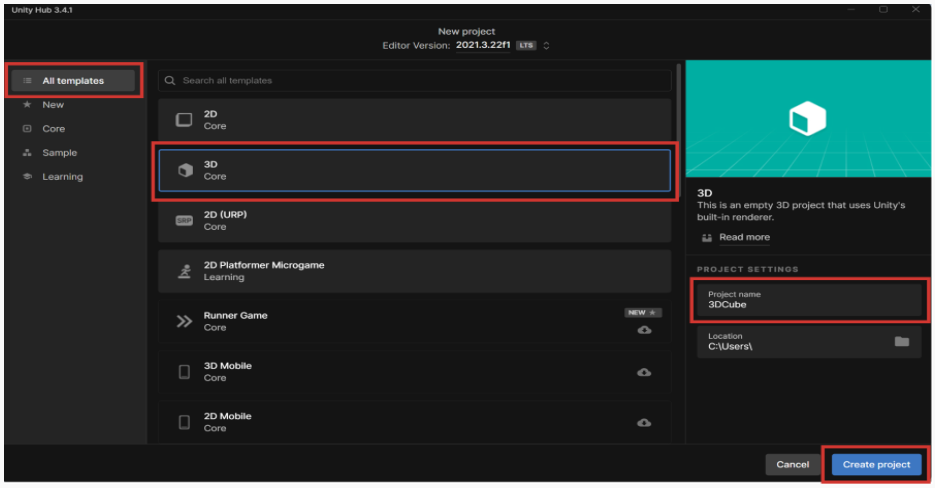
PROGRAM:

```
using UnityEngine;

public class CubeController : MonoBehaviour
{
    public float rotationSpeed = 100f;
    public float moveSpeed = 5f;

    // Called once at the beginning when the script starts
    void Start()
    {
        Debug.Log("CubeController started!");
    }

    // Called every frame
    void Update()
    {
        // Rotate the cube continuously
        transform.Rotate(Vector3.up * rotationSpeed * Time.deltaTime);

        // Move the cube using keyboard input
        float horizontal = Input.GetAxis("Horizontal"); // A/D or Left/Right
        float vertical = Input.GetAxis("Vertical");     // W/S or Up/Down

        Vector3 movement = new Vector3(horizontal, 0f, vertical);
        transform.Translate(movement * moveSpeed * Time.deltaTime);
    }
}
```

OUTPUT:

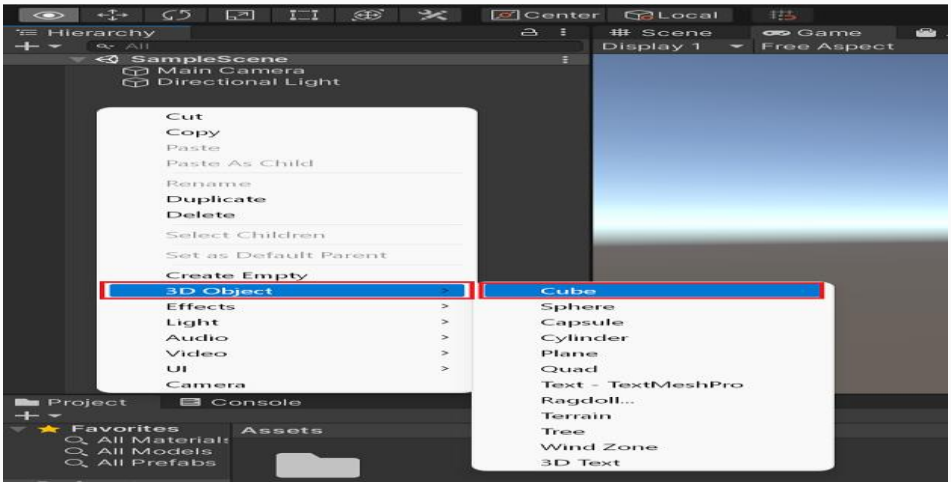


The **New project** dialog appears.



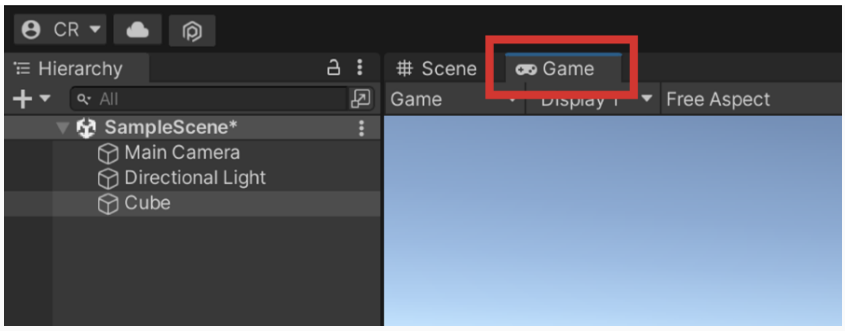
A new project is created, and Unity opens when the project finishes loading.

Right click on the **Hierarchy** window and choose **3D Object** > **Cube**.

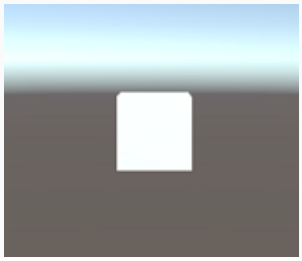


A cube object gets added to the **Hierarchy** window and the **Scene** view.

- Select the **Game** tab.



You should see a cube in the Game view, like the following:



Result:

A basic 3D AR/VR prototype scene with spatial awareness is created.

Ex no 3 Color Change on Click using C# Script

Aim:

To create a C# script that changes the color of a 3D object when clicked in AR/VR.

Procedure:

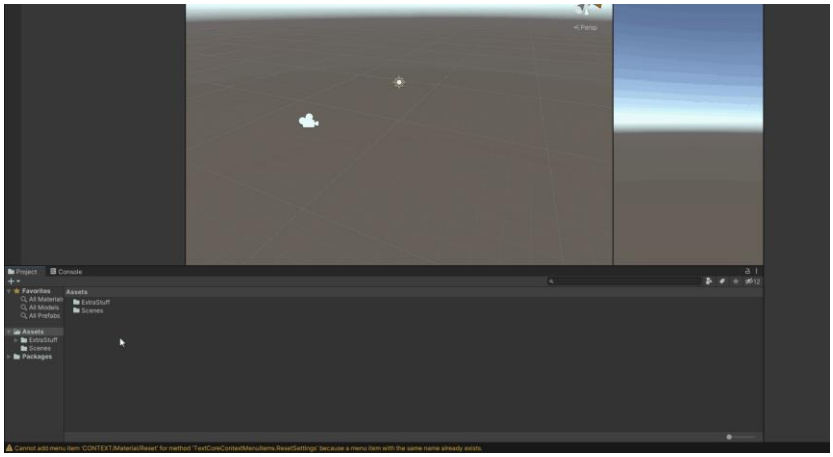
1. Create a **3D Cube** in your Unity scene.
2. Right-click in **Assets > Create > C# Script** and name it ColorChange.
3. Attach the script to the Cube object.
4. Add the C# code
5. Create material name and rename it as color
6. Click the Right click->Assets->material drag and drop on the cube the color will be changed
7. Save the file

PROGRAM:

```
using UnityEngine;

public class ColorChange : MonoBehaviour
{
    private Renderer objRenderer;
    void Start()
    {
        objRenderer = GetComponent<Renderer>();
    }

    void OnMouseDown()
    {
        objRenderer.material.color = new Color(Random.value, Random.value, Random.value);
    }
}
```

OUTPUT:

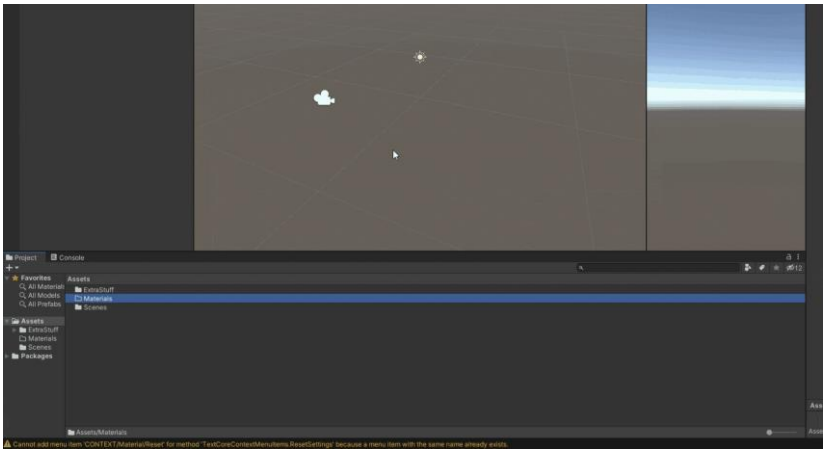
Step 1: Create a folder that contains all the colors



Right-click in Assets Panel > Create > Folder > Name it “Materials”

Step 2: Create a material

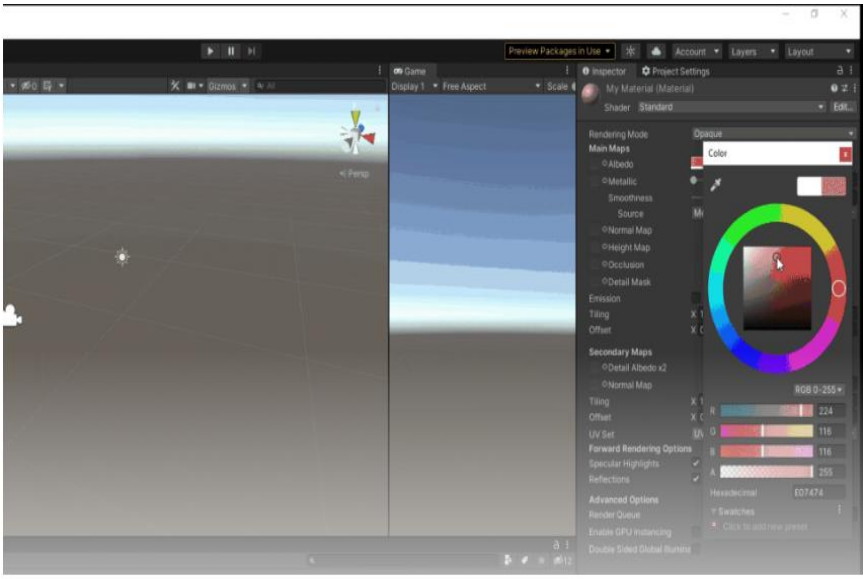
The next step is to create a color (or material).

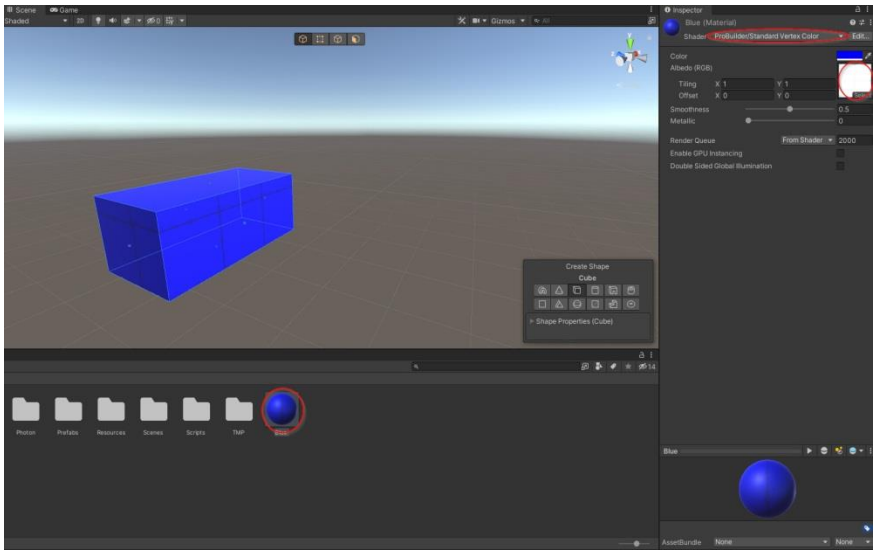


Go inside Materials Folder > Right-click > Create > Material > Name it “MyColor”(or whatever you want)

Step 3: Change the albedo property of the Material

Then select the created material and check out its properties in the Inspector Panel on the right-hand side.





Result:

The 3D object changes color every time the user clicks on it.

Ex no 4: Optimized Real-time C# Code for AR/VR

Aim:

To develop optimized, real-time interactive C# code for performance-tuned AR/VR applications.

Procedure:

Optimizing C# code for real-time AR/VR applications in Unity requires a focus on memory management, event-driven updates, and performance tuning.

Create a new C# script named OptimizedManager.

1. Memory Management:

- **Object Pooling:** Minimize new allocations during runtime by pre-instantiating frequently used objects (e.g., bullets, particles) and reusing them from a pool. This reduces garbage collection overhead.

2. Event-Driven Updates:

- **Delegates and Events:** Utilize C# delegates and events for communication between different parts of your application, promoting loose coupling and avoiding tight dependencies that can lead to performance issues.
- **Unity Events:**

  Leverage Unity's built-in UnityEvent system for inspector-configurable event handling, especially for UI interactions or simple game logic.

- **Coroutines for Asynchronous Tasks:**
  Use coroutines for tasks that don't need to complete within a single frame, such as loading assets or complex calculations, to avoid blocking the main thread.

3. Performance Tuning:

- Profile Regularly:

  Use Unity's Profiler to identify performance bottlenecks (CPU spikes, memory allocations, rendering issues).

- Batching and Instancing:

  Optimize rendering performance by using static or dynamic batching, and GPU instancing for drawing many identical objects.

- LOD (Level of Detail):

  Implement LOD for complex 3D models to reduce polygon count at greater distances.

- **Occlusion Culling:**

  Utilize occlusion culling to prevent rendering objects that are hidden behind other objects.

- **FixedUpdate for Physics:**

  Perform physics-related operations in FixedUpdate for consistent and accurate physics simulations, independent of frame rate.

- **Minimize Update Calls:**

  Only perform necessary calculations in Update. Consider using LateUpdate for camera follow logic or OnEnable/OnDisable for one-time setup/cleanup.

- **Early Exit Conditions:**

  Add early exit conditions in performance-critical methods to skip unnecessary computations when conditions are not met.

- **Burst Compiler and Jobs System:**
  For highly parallelizable computations, consider using Unity's Burst Compiler and Jobs System for significant performance gains by leveraging multi-core processors.

PROGRAM:

OptimizedManager:

```
using UnityEngine;
using System.Collections;

public class OptimizedManager : MonoBehaviour
{
   public GameObject prefab;
   private GameObject[] pool;
   private int poolSize = 5;

   void Start()
   {
      pool = new GameObject[poolSize];
      for (int i = 0; i < poolSize; i++)
      {
         pool[i] = Instantiate(prefab);
         pool[i].SetActive(false);
      }
   }

   public void Spawn(Vector3 position)
   {
      foreach (GameObject obj in pool)
      {
         if (!obj.activeInHierarchy)
         {
            obj.transform.position = position;
            obj.SetActive(true);
            break;
         }
      }
   }
}
```

OUTPUT:

GameObject prefab instances appear at given positions)
Example in Console (if you log positions):

```
Spawned object at (0,0,0)
Spawned object at (2,0,0)
Spawned object at (4,0,0)
```

Object pooling:

```
   public class ObjectPool
   {
      private Queue<GameObject> _pool = new Queue<GameObject>();
```

```
    public GameObject GetObject(GameObject prefab)
    {
        if (_pool.Count > 0)
        {
            GameObject obj = _pool.Dequeue();
            obj.SetActive(true);
            return obj;
        }
        return GameObject.Instantiate(prefab);
    }

    public void ReturnObject(GameObject obj)
    {
        obj.SetActive(false);
        _pool.Enqueue(obj);
    }
}
```

Delegates and Events:

```
public class GameEvents
{
    public static event Action OnPlayerDeath;

    public static void PlayerDied()
    {
        OnPlayerDeath?.Invoke();
    }
}
```

OUTPUT:

When pressing keys during runtime:Pgsql

Spawned Cube: Cube(Clone)
Spawned Cube: Cube(Clone)
Returned all cubes to pool
Spawned Cube: Cube(Clone)
Returned all cubes to pool

OUTPUT:

If you press **Space**, the console output will be:

Player died!
Game Over! Restart or Quit?

Result:

Optimized AR/VR code improves performance and reduces memory leaks.

Ex no 5 Level Design and Real-time Rendering Concepts

Aim:

To design an AR/VR level with optimized lighting and real-time rendering considerations.

Procedure:

1. Create multiple 3D models and arrange them as a level.
2. Apply **Lighting Settings**: Realtime Global Illumination, Reflection Probes.
3. Add **Occlusion Culling** for performance.
4. Bake lighting maps for static objects.
5. Test level performance in **Game Mode**.

Step 1: Create and Arrange 3D Models

1. Open Unity and create a **New 3D Scene**.
2. In the **Hierarchy**, go to **GameObject → 3D Object → Cube/Plane** to build the base ground.
3. Import or create multiple 3D models (e.g., trees, buildings, walls, characters).
4. Arrange these objects logically to form a **level layout** such as a room, street, or environment.
5. Set the **Transform (Position, Rotation, Scale)** of each object appropriately for realism.

Step 2: Apply Lighting Settings

1. Go to **Window → Rendering → Lighting** to open the **Lighting Settings** panel.
2. Enable **Realtime Global Illumination** and **Baked Global Illumination** for dynamic and static lighting.
3. Add a **Directional Light** (sunlight) to simulate outdoor lighting.
4. Adjust **Light Intensity**, **Shadow Type**, and **Color Temperature** for desired ambience.
5. Add **Reflection Probes** (GameObject → Light → Reflection Probe) to improve reflections on metallic surfaces.
6. Place probes strategically where environment reflections are needed.

Step 3: Add Occlusion Culling

1. Go to **Window → Rendering → Occlusion Culling**.
2. Click **Bake** to generate occlusion data.
3. Ensure large static objects (like walls or buildings) are marked as **Static** in the Inspector.
4. Test by moving the Scene camera — hidden objects should not render behind obstacles.
    - This improves frame rate and reduces GPU load.

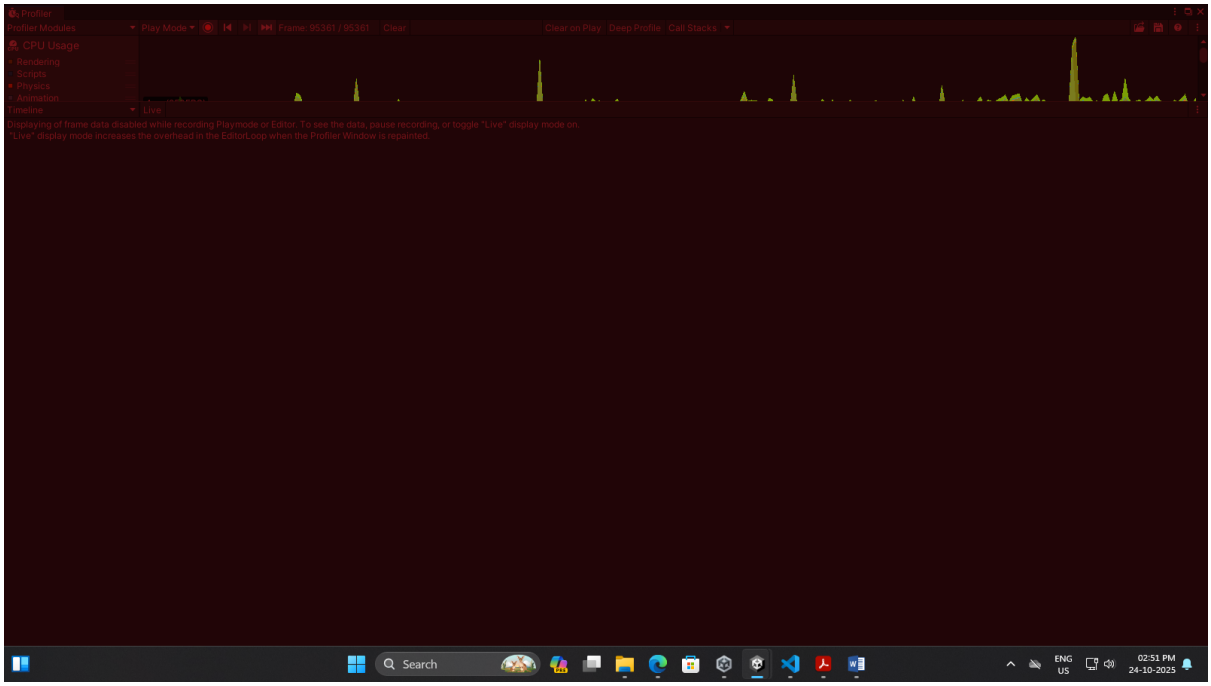
Step 4: Bake Lighting Maps

1. Select all environment objects that do not move and mark them as **Static → Lightmap Static**.
2. In **Lighting Settings**, scroll down to the **Lightmapping Settings** section.
3. Choose the **Lightmapper** (e.g., Progressive GPU or CPU).
4. Click **Generate Lighting** or **Bake**.
5. Wait for Unity to compute the lightmaps — static surfaces will now have baked lighting and shadows.

Step 5: Test Level Performance

1. Enter **Play Mode** to view the rendered level.
2. Use **Window → Analysis → Profiler** to check CPU/GPU usage, frame rate, and rendering stats.
3. Observe the lighting, reflections, and overall smoothness.
4. Optimize if needed by:
    - Reducing real-time lights
    - Adjusting lightmap resolution
    - Using occlusion culling for hidden areas

OUTPUT:


Profiler
Profiler Modules
Play Mode
Frame: 95361 / 95361
Clear
Clear on Play
Deep Profile
Call Stacks
CPU Usage
Rendering
Scripts
Physics
Animation
Timeline
Live
Displaying of frame data disabled while recording Playmode or Editor. To see the data, pause recording, or toggle "Live" display mode on.
"Live" display mode increases the overhead in the EditorLoop when the Profiler Window is repainted.
Search
ENG
US
02:51 PM
24-10-2025

Result:

Level renders in real-time with smooth performance and dynamic lighting.

Ex no 6: Build an AR App using Vuforia SDK

Aim:

To create an AR app displaying a 3D model on a printed image marker using Vuforia.

Procedure:

1. In Unity, enable **Vuforia Engine** (Project Settings > XR).
2. Open **Vuforia Configuration** from **Window > Vuforia Engine > Configuration**.
3. Add your **Vuforia License Key**.
4. Add an **AR Camera** and **Image Target** to the scene.
5. Import a 3D model and place it as a child of the Image Target.
6. Build and deploy to Android.
7. Point the camera at the printed marker to view 3D model overlay.

1. Project Setup and Vuforia Integration:

- **Create a new Unity Project:**

  Open Unity Hub and create a new 3D project.

- **Import Vuforia Engine:**

  Go to Window > Package Manager, search for "Vuforia Engine", and install or update it.

- **Obtain a Vuforia License Key:**

  Visit the Vuforia Developer Portal, log in, navigate to the License Manager, and create a new license key for your application. Copy this key.

- **Configure Vuforia in Unity:**
  Go to Window > Vuforia Configuration, paste your license key into the "App License Key" field.

2. Image Target Database Creation:

- **Create a Target Database:**

  On the Vuforia Developer Portal, go to the Target Manager and create a new database (e.g., "MyImageTargets").

- **Add an Image Target:**

  Within your database, click "Add Target", select "Single Image", upload your desired image marker, and set its width. Ensure the image has good contrast and features for reliable tracking.

- **Download Database:**

  Download the database "For Unity Editor" as a .unitypackage.

- **Import into Unity:**
  In Unity, go to Assets > Import Package > Custom Package and import the downloaded .unitypackage.

3. Setting up the Scene:

- **Delete Main Camera:** In the Hierarchy, delete the default "Main Camera" object.
- **Add AR Camera:** Right-click in the Hierarchy, select Vuforia Engine > AR Camera.
- **Add Image Target:** Right-click in the Hierarchy, select Vuforia Engine > Image Target.
- **Configure Image Target:** Select the "Image Target" GameObject in the Hierarchy. In the Inspector, under the "Image Target Behaviour" component, set the "Type" to "From Database" and choose your created database and the specific image target from the dropdowns.

4. Adding the 3D Object:

- **Create or Import 3D Object:**
- **Cube:** Right-click on your "Image Target" in the Hierarchy, select 3D Object > Cube.
- **Custom Model:** Import your 3D model (e.g., FBX, OBJ) into your Unity project, then drag it as a child of the "Image Target" in the Hierarchy.
- **Position and Scale:**
  Adjust the position, rotation, and scale of your 3D object as a child of the "Image Target" so it appears correctly on the marker when tracked. You may need to scale it down significantly.

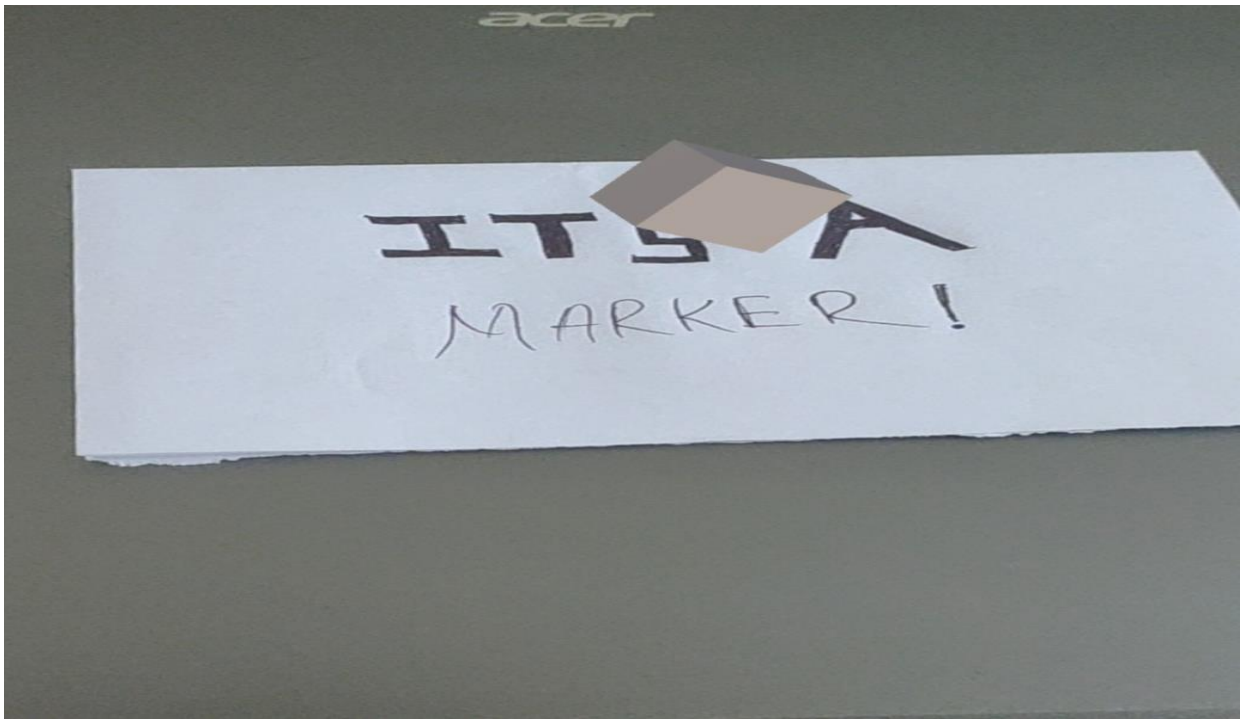
5. Testing and Building:

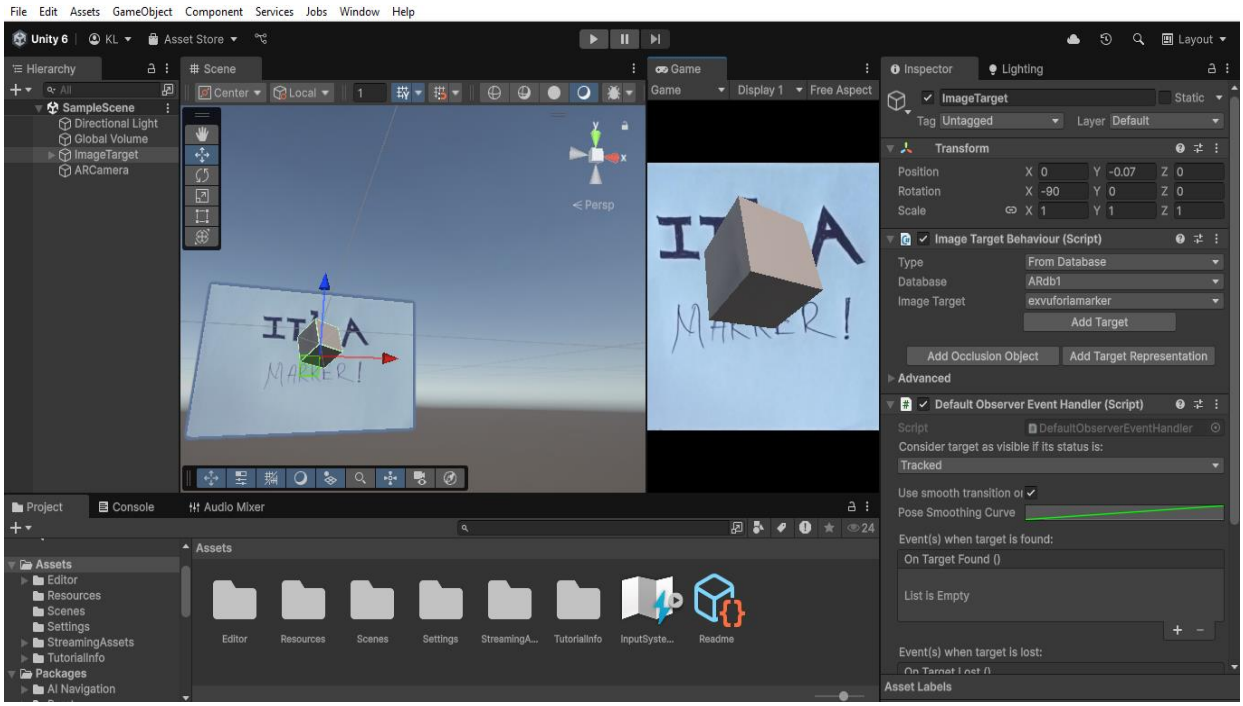
- **Test in Editor:**

  Click the "Play" button in Unity. Point your webcam (selected in the AR Camera's "Play Mode Type" in Vuforia Configuration) at the printed image marker. The 3D object should appear on the marker.

- **Build for Device:**
- Go to File > Build Settings.
- Select your target platform (e.g., Android, iOS).
- Click "Player Settings", navigate to "XR Plug-in Management", and enable "Vuforia Engine" for your chosen platform.
- Click "Build" and deploy the application to your device.
  When running the application on a device, pointing the device's camera at the printed image marker will cause the 3D object to appear augmented on top of the physical marker.

Output:


acer
MARKER!


File Edit Assets GameObject Component Services Jobs Window Help
Unity 6
Asset Store
Layout
Hierarchy
SampleScene
Directional Light
Global Volume
ImageTarget
ARCamera
Scene
Persp
Game
Display 1
Free Aspect
Inspector
Lighting
ImageTarget
Static
Tag Untagged
Layer Default
Transform
Position
Rotation
Scale
Image Target Behaviour (Script)
Type From Database
Database ARdb1
Image Target exvuforiamarker
Add Target
Add Occlusion Object
Add Target Representation
Advanced
Default Observer Event Handler (Script)
Script DefaultObserverEventHandler
Consider target as visible if its status is:
Tracked
Use smooth transition of
Pose Smoothing Curve
Event(s) when target is found:
On Target Found ()
List is Empty
Event(s) when target is lost:
Asset Labels
Project
Console
Audio Mixer
Assets
Editor
Resources
Scenes
Settings
StreamingAssets
TutorialInfo
Packages
AI Navigation
Editor
Resources
Scenes
Settings
StreamingA...
TutorialInfo
InputSyste...
Readme

Result:

The 3D model appears when the printed image marker is detected.

Ex no 7: Design, Develop, and Deploy AR/VR Application

Aim:

To design, develop, and deploy a full AR or VR application reflecting user experience flow.

Procedure:

1. Plan **User Interaction Flow** (e.g., scene transitions, gestures).
2. Design UI with **Canvas** and **Event System**.
3. Add **XR Rig or AR Session Origin** depending on app type.
4. Integrate scripts for navigation, interaction, and object control.
5. Test using Unity Play Mode and on-device build.
6. Deploy to device (Android APK or VR headset build).

Step 1: Plan User Interaction Flow

1. Define the purpose of your AR/VR application (e.g., product visualization, interactive demo, navigation).
2. Create a simple **flowchart** showing how users will interact with the app:
    - Start Scene → Main Menu → AR/VR View → Interaction → Exit/Quit.
3. Identify possible interactions such as:
    - **AR:** Tap to place, pinch to scale, drag to rotate.
    - **VR:** Gaze-based selection, controller input, teleportation.

Step 2: Design the User Interface (UI)

1. In Unity, go to **Hierarchy → UI → Canvas** to create a user interface.
2. Add UI elements such as **Buttons**, **Text**, or **Panels** for navigation.
3. Ensure **Canvas** has a proper **Render Mode: World Space (for AR/VR)**.
4. Add an **Event System** (automatically created with Canvas).
5. Customize UI layout for your application flow (e.g., Start, Quit, Restart buttons).

Step 3: Add XR Rig or AR Session Origin

1. **For VR Application:**
    - Go to **GameObject → XR → Device-based → XR Origin (VR)**.
    - This adds a VR-ready camera rig to handle headset movement and controller tracking.
2. **For AR Application:**
    - Go to **GameObject → XR → AR Session Origin** and **AR Session**.
    - The AR Session Origin manages camera tracking and real-world alignment.
3. Position the rig/camera properly and ensure the main camera is linked.

Step 4: Integrate Scripts for Interaction and Control

1. Create and attach **C# scripts** to enable user interaction.

   For AR:

```
using UnityEngine;
using UnityEngine.XR.ARFoundation;
using UnityEngine.XR.ARSubsystems;

public class ARObjectPlacer : MonoBehaviour
{
   public GameObject objectToPlace;
   private ARRaycastManager raycastManager;
   private List<ARRaycastHit> hits = new List<ARRaycastHit>();

   void Start()
   {
      raycastManager = GetComponent<ARRaycastManager>();
   }

   void Update()
   {
      if (Input.touchCount > 0)
      {
         Touch touch = Input.GetTouch(0);
         if (raycastManager.Raycast(touch.position, hits, TrackableType.Planes))
         {
            Pose hitPose = hits[0].pose;
            Instantiate(objectToPlace, hitPose.position, hitPose.rotation);
         }
      }
   }
}
```

   For VR:

   Use XR Interaction Toolkit components (grab interactables, teleportation area).

2. Link scripts to corresponding GameObjects (e.g., AR Session Origin or Interactable Object).

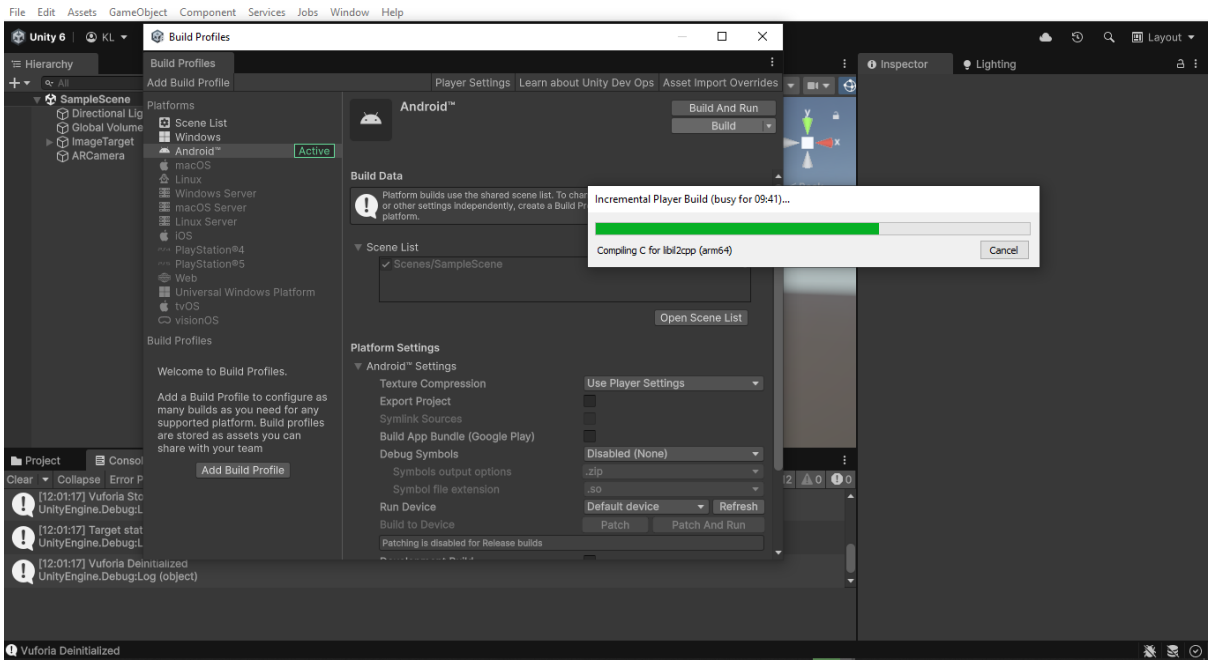
Step 5: Test in Unity Editor and On Device

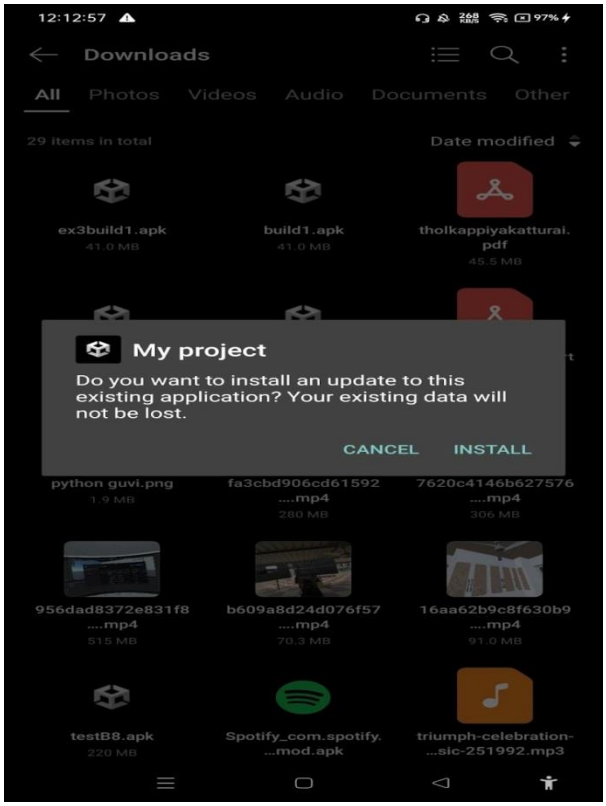
1. Save the scene (File → Save Scene As → MainScene).
2. Enter **Play Mode** in Unity to test basic UI and interactions.
3. Connect your device (AR-supported phone or VR headset).
4. Open **File → Build Settings** → select the correct platform (Android/Windows).
5. Enable **XR Plugin Management** under Project Settings → XR Plug-in Management.

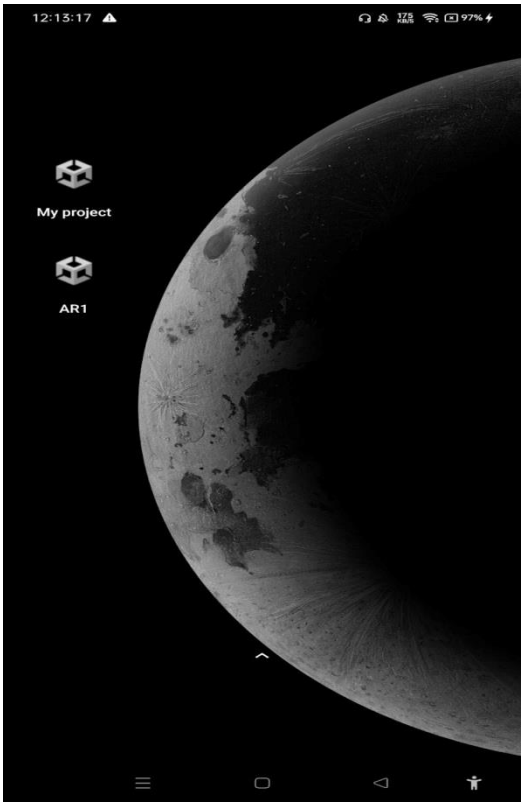
Step 6: Build and Deploy the Application

1. For **Android (AR App):**
    - Go to **Build Settings → Android → Switch Platform**.
    - Set **Minimum API Level (Android 10 or above)**.
    - Enable **ARCore Supported** in Player Settings.
    - Click **Build and Run** to generate and install .apk.
2. For **VR (Oculus or PCVR):**
    - Switch to **PC or Android** platform based on headset type.
    - Enable **Oculus / OpenXR Plugin**.
    - Click **Build and Run**.
3. Once installed, open the app on the device and test real-world performance.

OUTPUT:


Build Profiles
Android™
Build And Run
Build
Incremental Player Build (busy for 09:41)...
Compiling C for libil2cpp (arm64)
Cancel
Open Scene List
Vuforia Deinitialized


Downloads
My project
Do you want to install an update to this existing application? Your existing data will not be lost.
CANCEL
INSTALL


My project
AR1

Result:

A functional AR/VR application is deployed and interacts smoothly with user input.