AUGMENTED REALITY DRAWING APP

ABYAN KUMAR R[1], SANDHIYADEVI P[2]

abyankumar.cs20@bitsathy.ac.in, sandhiyadevip@bitsathy.ac.in

1. Student, Computer Science and Engineering, Bannari Amman Institute of Technology, Tamil Nadu, India

2. Faculty, Electronics and Communication Engineering, Bannari Amman Institute of Technology, Tamil Nadu, India

ABSTRACT

Augmented Reality (AR) technology has ushered in a new era of interactive digital experiences, offering users the ability to blend virtual content seamlessly with the physical world. In this report, we present the findings and recommendations resulting from the testing and evaluation of an AR Drawing App. The app aims to provide users with a creative platform for digital art creation, leveraging AR technology to enable users to draw and paint directly onto their surroundings.

Through a comprehensive testing process, including functionality testing, AR tracking evaluation, user interaction analysis, and real-world usability assessment, we identified both strengths and areas for improvement within the app. Notably, the app demonstrates intuitive drawing features, responsive touch input, and effective surface detection under optimal conditions. However, challenges such as performance degradation on older devices, limitations in low-light environments, and user interface complexities were also observed.

Based on these findings, we propose actionable recommendations to enhance the app's performance, usability, and overall user experience. These recommendations include optimizing performance for a broader range of devices, refining surface detection algorithms for challenging lighting conditions, and simplifying the user interface to improve accessibility and user engagement. Additionally, addressing privacy concerns related to data collection practices will be crucial in fostering user trust and compliance with privacy regulations.

By implementing these recommendations and continuing to iterate based on user feedback and emerging technologies, the AR Drawing App can realize its full potential as a transformative tool for digital creativity in augmented reality. This report serves as a guide for developers, designers, and stakeholders involved in the ongoing development and refinement of AR drawing applications, highlighting the importance of user-centred design principles, technical optimization, and continuous improvement in delivering compelling AR experiences.

KEYWORDS: *Augmented Reality, Interactive experience, usability-assessment, user experience, transformative tool, digital creativity, surface detection, data collection, creative platform.*

---

1. INTRODUCTION:

In the evolving landscape of digital artistry, the emergence of Augmented Reality (AR) drawing applications marks a significant leap forward in how creators interact with their environment and their art. These applications, harnessing the power of AR technology, have transformed traditional artistic expressions by blending the digital and physical worlds in unprecedented ways. This report delves into the core of AR drawing apps, exploring their technological foundations, user experiences, artistic possibilities, and the broader implications for both the art world and technology sectors.

1. AR drawing apps leverage sophisticated algorithms and the capabilities of modern smartphones and tablets to allow users to draw, design, and visualize art overlaid onto the real world through their device's screen. This revolutionary approach does not merely digitize the act of drawing but expands it into three-dimensional space, enabling artists and hobbyists alike to explore their creativity without the confines of paper or canvas.
2. As we explore the nuances of AR drawing applications, we consider several key areas. Firstly, the technological underpinnings that make such apps not just possible but practical and accessible. This

includes advancements in AR frameworks, such as ARKit for iOS and ARCore for Android, and the hardware developments that support them.

3. Secondly, the user experience (UX) design of these applications plays a critical role in their adoption and utility. The intuitive interfaces, responsiveness to real-world conditions, and the seamless integration of digital and physical views are crucial factors that define the success of any AR drawing app.
4. Furthermore, we examine the impact of AR drawing apps on artistic expression and education. These platforms offer novel ways to visualize and create art, making art more interactive and immersive. They hold the potential to democratize art creation, making it more accessible to individuals without traditional art training and providing a new medium for experienced artists to explore.
5. Finally, this report reflects on the future trajectory of AR drawing applications. As technology advances, what new features and capabilities can we anticipate? How might these tools further change the landscape of art, education, and entertainment?
6. In sum, AR drawing apps represent a vibrant intersection of technology and creativity, offering new dimensions of expression and engagement. This report aims to provide a comprehensive overview of this innovative field, underscoring the significance of AR technology in expanding the horizons of digital artistry.

1.2 SIGNIFICANCE OF THE PROPOSED PROJECT:

The advent and evolution of Augmented Reality (AR) Drawing apps have marked a transformative period in the realms of digital technology, art, and education. These innovative tools enable users to interact with their surroundings in novel ways, merging the virtual with the physical, and in the process, unlocking a myriad of opportunities for creativity, learning, and professional development. This report delves into the multifaceted significance of AR Drawing apps, exploring their impact across various domains.

1. **Enhancing Creative Expression**

   AR Drawing apps offer an unparalleled platform for artists and enthusiasts alike to experiment with 3D compositions in real-time environments. This technology not only expands the canvas for creators beyond traditional boundaries but also introduces a new dimension to art, allowing for the creation of immersive experiences that can be shared and explored by a wider audience.

2. **Revolutionizing Education**
   In educational settings, AR Drawing apps have the potential to revolutionize teaching methodologies. By facilitating interactive learning experiences that can overlay information directly onto the physical world, these applications make abstract concepts more tangible and engaging for students, thereby improving comprehension and retention rates. From visualizing geometric shapes in mathematics to simulating historical events in situ, the possibilities for enhancing education with AR technology are vast and varied.
3. **Democratizing Art and Design**
   The accessibility of AR Drawing apps means that anyone with a smartphone or tablet can explore the realms of 3D drawing and design, regardless of their background or expertise in traditional art. This democratization of creative tools lowers the barriers to entry for aspiring artists and designers, fostering a more inclusive community of creators and encouraging a wider spectrum of people to engage in artistic endeavors.
4. **Transforming Professional Practices**
   For professionals in architecture, engineering, and design, AR Drawing apps serve as powerful visualization and collaboration tools. These applications enable the overlaying of proposed designs onto physical spaces, facilitating a more intuitive understanding of how projects will interact with their environments. This capability not only aids in the design process but also enhances communication with clients and stakeholders, leading to more efficient decision-making and project approval processes.
5. **Encouraging Social Interaction and Collaboration**
   AR Drawing apps often incorporate social features that enable users to share their creations with a global community or collaborate on projects in real-time, despite geographical distances. This fosters a sense of community among users, promoting collaborative creativity and cultural exchange. In an increasingly connected world, these social interactions and collaborations through AR apps play a crucial role in inspiring collective artistic projects and innovations.

6. **Driving Technological Advancement**
   The development and refinement of AR Drawing apps push the boundaries of what is possible with AR technology. This drives innovation not only in software development, including AR frameworks and algorithms, but also in hardware, as devices must evolve to support the advanced capabilities required by these applications. As such, AR Drawing apps contribute to the overall advancement of digital technologies, setting the stage for future innovations.

1.3 PROJECT WORKFLOW IN UNITY:

Creating an AR Drawing app using Unity involves a structured workflow that combines the capabilities of Unity's game development environment with AR technologies, such as ARFoundation (a Unity package that supports both ARKit and ARCore). This workflow can be broadly categorized into initial setup, development of AR functionality, implementation of drawing features, and testing & optimization. Here's a step-by-step guide to navigate through this process:

1. Initial Setup

Install Unity: Ensured that the latest version of Unity installed that supports ARFoundation.

Create a New Project: A new Unity project have been started and chosen a 3D template.

ARFoundation and Dependencies: Through the Unity Package Manager, ARFoundation along with ARCore XR Plugin for Android has been installed.

2. AR Setup and Environment Preparation

Configure AR Session: An AR Session and AR Session Origin has been added to the scene. The AR Session Origin will contain an AR Camera.

Setting Up Plane Detection: To enable the drawing on surfaces, plane detection has been set up by adding an AR Plane Manager to the AR Session Origin and has been configured it to detect horizontal or vertical planes as required.

Visualize Planes: An AR Plane Visualizer has been added to see the detected planes during development.

3. Implementing Drawing Functionality

Create Drawing Controls: A control system has been added for when and where drawings will be placed in the AR environment. This involves touch controls or UI elements that enable users to start or stop the drawing process.

Developing Drawing Mechanics: Usinf Unity's Line Renderer or a custom shader to render lines in 3D space. When the user touches a point on the screen, raycast from the AR Camera to the detected plane and start rendering a line from that point. As the user moves their finger, continue adding points to the line renderer to create the drawing.

Optimize Performance: Ensured that the drawing mechanics are optimized for performance, particularly by limiting the number of line segments or implementing a pooling system for line objects.

4. User Interface and Experience

Add UI Elements: Created a user-friendly interface with buttons and sliders for different functionalities, such as changing the color and selecting different brushes.

Implement Feedback Systems: Provided a visual or haptic feedback when users start drawing or when their drawing is projected onto a plane.

5. Testing and Optimization

Test on Devices: The app has been tested regularly on actual AR-supported devices to get a real sense of the user experience and performance.

Optimize: Based on testing feedback, the app has been optimized for better performance and user experience. This might include adjustments to the AR settings, improving the drawing algorithm efficiency, or streamlining the UI.

6. Deployment

Build Settings: Configured the build settings for the target platform (iOS or Android), ensuring all AR and app permissions are correctly set.

Deploy: Build and deployed the application to the Android device for final testing.

1.4 CHALLENGES AND SOLUTIONS OF THE PROPOSED APPLICATION:

Technical Challenges:

Hardware Requirements: AR drawing apps often require advanced hardware with powerful processors and high-quality cameras to function smoothly. Users with older or less capable devices may experience lag, low-quality AR overlays, or even incompatibility issues.

Software Optimization: Ensuring the app runs smoothly across a variety of devices with different operating systems and specifications can be challenging. The app needs to be constantly updated and optimized for new devices and OS updates.

Battery Consumption: AR apps tend to consume a significant amount of battery life due to the intensive use of the camera, processor, and display. This can limit the duration of use.

User Experience (UX) Challenges:

User Interface (UI) Complexity: Crafting a user-friendly interface that balances the complexity of drawing tools with the simplicity needed for broader user adoption is challenging. The UI must be intuitive yet offer advanced features for more skilled artists.

Spatial Awareness: The app must accurately interpret and map the physical space through the camera. Any inaccuracies can disrupt the user experience, making drawings appear disjointed from the real world.

Learning Curve: For new users, particularly those not familiar with AR technology, there can be a significant learning curve. Educating users on how to best utilize these apps for optimal results is necessary.

Content Moderation and Privacy Concerns:

Inappropriate Content: As with any platform that allows user-generated content, there's a risk of inappropriate or offensive material being created and shared. Implementing effective moderation tools and policies is crucial.

Privacy Issues: Since these apps can capture and potentially store images or videos of private spaces, they must navigate privacy concerns and regulations carefully to protect user data.

2. LITERATURE SURVEY:

1. "Augmented Reality: Principles and Practice" by Dieter Schmalstieg and Tobias Hollerer

Overview: This book provides a comprehensive guide to the fundamentals of AR, offering insights into the technology's underlying principles. It covers the hardware, software, and general principles essential for developing AR applications, which can be applied to AR Drawing Apps.

2. "Understanding Augmented Reality: Concepts and Applications" by Alan B. Craig

Overview: Alan B. Craig's work is an accessible introduction to the concepts behind Augmented Reality. While not focused on drawing apps specifically, the book lays down the groundwork for understanding AR's potential applications, including art and design.

3. "Augmented Reality for Developers: Build practical augmented reality applications with Unity, ARCore, ARKit, and Vuforia" by Jonathan Linowes and Krystian Babilinski

Overview: A hands-on guide for developers interested in creating AR applications using popular platforms and tools. While it focuses on a broad range of applications, the principles and techniques can be directly applied to developing AR drawing and creative tools.

4. "Prototyping Augmented Reality" by Tony Mullen

Overview: This book offered a project-based approach to learning AR development, with a focus on creating interactive prototypes. The skills learned here can be invaluable for anyone looking to design and prototype AR Drawing Apps.

5. "Designing Immersive Video Games Using 3DUI Technologies: Improving the Gamer's User Experience" by Arun Kulshreshth and Joseph J. LaViola Jr.

Overview: Although focused on video games, this book discusses the importance of user interface and user experience in immersive environments. These insights are directly applicable to the design of user-friendly AR drawing interfaces.

REFERRED JOURNALS:

Unity learning platform by Saba Sohail:

https://www.tekrevol.com/blogs/how-to-build-an-augmented-reality-apps/

"Augmented Reality: Principles and Practice" by Dieter Schmalstieg and Tobias Hollerer

This book provides a comprehensive introduction to the principles and practices of augmented reality, covering both technical and practical aspects.

"Programming Augmented Reality" by Tony Mullen

Tony Mullen's book focuses on practical aspects of AR development, providing hands-on guidance for creating AR applications using various platforms and tools.

"Augmented Human: How Technology Is Shaping the New Reality" by Helen Papagiannis

Dr. Helen Papagiannis explores the current state and future potential of augmented reality in this book, discussing its impact on human experience and society.

"Augmented Reality for Developers: Build practical augmented reality applications with Unity, ARCore, ARKit, and Vuforia" by Jonathan Linowes

This book is a practical guide for developers interested in building AR applications, using popular platforms such as Unity, ARCore, ARKit, and Vuforia

3. OBJECTIVE AND METHODOLOGY:

3.1 OBJECTIVES OF THE PROPOSED WORK:

There is a fundamental disconnect between the wealth of digital data available to us and the physical world in which we apply it. While reality is three-dimensional, the rich data we now have to inform our decisions and actions remains trapped on two-dimensional pages and screens. This gulf between the real and digital worlds limits our ability to take advantage of the torrent of information and insights produced by billions of smart, connected products (SCPs) worldwide.

1. Enhancing Creativity and Artistic Expression

Objective: To provide users with intuitive and innovative tools that expand the boundaries of traditional art and design, allowing for the creation of augmented reality content that interacts with the real environment.

2. Simplifying Complex Design and Visualization Tasks

Objective: To enable professionals, such as architects, engineers, and educators, to visualize and manipulate 3D models in real space, thereby simplifying complex design tasks and enhancing understanding and collaboration.

3. Improving Educational Experiences

Objective: To offer interactive and immersive learning opportunities that can transform education by making abstract concepts tangible and by providing a more engaging, hands-on learning experience.

4. Increasing Engagement and Interaction

Objective: To captivate users' attention by merging digital content with the real world, thereby creating more engaging and interactive experiences compared to traditional 2D drawing or painting apps.

5. Facilitating Collaborative Work

Objective: To provide a platform that supports real-time collaboration among users, regardless of their physical location, enabling shared creativity and collective design efforts in augmented reality.

6. Enabling Immersive Storytelling and Presentation

Objective: To allow storytellers, marketers, and educators to craft immersive narratives or presentations where the audience can interact with spatially-aware digital content, enhancing the impact and retention of the conveyed messages.

7. Promoting Accessibility and Inclusivity

Objective: To make art and design more accessible by providing tools that require minimal technical skills to use, thereby lowering the barrier to entry for users with diverse backgrounds and abilities.

8. Fostering Exploration and Play

Objective: To encourage exploration, play, and experimentation with AR technology, thereby fostering a deeper understanding and appreciation of augmented reality's potential applications.

9. Streamlining Prototyping and Iterative Design

Objective: To enable designers and developers to quickly prototype and iterate on 3D designs in a real-world context, speeding up the feedback and refinement process.

10. Bridging Physical and Digital Creative Spaces

Objective: To seamlessly integrate digital creativity into the physical world, allowing users to draw, design, or annotate spaces and objects around them, thereby blending physical and digital creative spaces.

3.1.1MEDICAL OBJECTIVES:

1. This project brings out the inner imaginative ideas into somewhat of a real world implementation like a design on a table could later be implemented into an actual table cover.

2. Children suffering from ADHD and Autism can use this app to learn better as writing in a physical environment improves the attention and makes their concentration treatments easier.

3.2 METHODOLOGY:

The proposed Drawing app involves a multi-disciplinary approach, blending software development, user interface design, and an understanding of AR technology's capabilities and limitations. Below is a high-level methodology for developing an AR Drawing app, which can be adapted based on specific project requirements, target platforms (iOS, Android, etc.), and user needs.

1. Conceptualization and Planning

Identify Objectives: Defined the core objectives and unique value proposition of your AR Drawing app. Consider the target audience, use cases, and what sets your app apart from existing solutions.

Market Research: Conducted thorough market research to understand the competitive landscape, user expectations, and potential gaps in the market that the app could fill.

Define Features: Listed the key features of the proposed app, such as 3D drawing, AR object placement, collaboration tools, or educational content.

2. Technology Selection

Choose a Development Platform: the platforms are decided for which the app will support (iOS, Android, or cross-platform). In this case, it is developed for android

Select AR Toolkit: An AR development toolkit that suits the platform have been chosen and feature needs, such as ARKit for iOS, ARCore for Android, or Unity with Vuforia for cross-platform development.

3. Design Phase

UX/UI Design: Designed the user experience (UX) and user interface (UI) with a focus on simplicity and intuitiveness. AR apps require careful consideration of how users interact with both digital elements and the real world.

Prototyping: Developed an early interactive prototypes to test and refine the app’s design and user flow. Tools like Adobe XD, Sketch, or Figma can be used for non-AR prototyping, while Unity can be used for AR-specific scenes.

4. Development Phase

Environment Setup: The development environment has been set up, including the installation of necessary SDKs, IDEs (Integrated Development Environments), and AR toolkits.

Core Functionality Development: Started coding the core functionalities of your app, such as AR scene recognition, drawing capabilities, and user input handling.

Iterative Testing and Refinement: Regularly testing have been done on the app on actual devices to ensure performance and usability. Used feedback to make iterative improvements.

5. User Testing

Alpha/Beta Testing: Conducted alpha testing with internal stakeholders and beta testing with a select group of external users. feedback is collected and adjusted to identify bugs, usability issues, and areas for improvement.

Usability Testing: Performed focused usability testing to refine the app’s interface, interactions, and overall user experience.

6. Launch Preparation

Optimization and Final Testing: Optimized the proposed app for performance and battery life. Conducted final testing across different devices and operating systems to ensure compatibility and smooth operation.

3.2.1 U.I DESIGN:

Design Principles for AR UI

Minimize Clutter: Keep the UI unobtrusive to not distract from the AR experience. Use transparent elements and hide menus when not in use.

Spatial Awareness: Designed the UI elements that are aware of the user's environment, adjusting for obstacles and surfaces.

Ergonomics: interactions ensurance require minimal physical strain. Considered the natural range of motion and avoid requiring users to hold awkward poses.

Intuitive Interaction: Used gestures and interactions that feel natural and intuitive. Incorporate familiar gestures (e.g., pinch to zoom, swipe to change tools) and consider physical interactions with the AR elements.

4. Sketch and Wireframe Interfaces

Created sketches and wireframes for your app’s interface, focusing on placement and functionality of key elements like menus, toolbars, and buttons. Considered how these elements will appear in different environments.

5. Prototype and Test

Interactive Prototypes: Developed interactive prototypes using tools that support AR design, like Unity with mockups or specialized AR prototyping tools.

3.2.2 KEY ELEMENTS FOR U.I DESIGN:

Tool Selector: A menu for selecting drawing tools and brushes. Consider using icons with a preview or a short animation of the tool in action.

Color Picker: An intuitive and easily accessible color selection tool. Options include color wheels, swatches, or sliders for more precise control.

Undo/Redo: Easy-to-reach buttons for undoing and redoing actions, essential for a smooth drawing experience.

Environmental Interaction Indicators: Visual cues that help users understand how their drawings will interact with the physical environment, such as shadows or highlights on surfaces. In the proposed app, certain smaller digital dots indicate the spatial area.

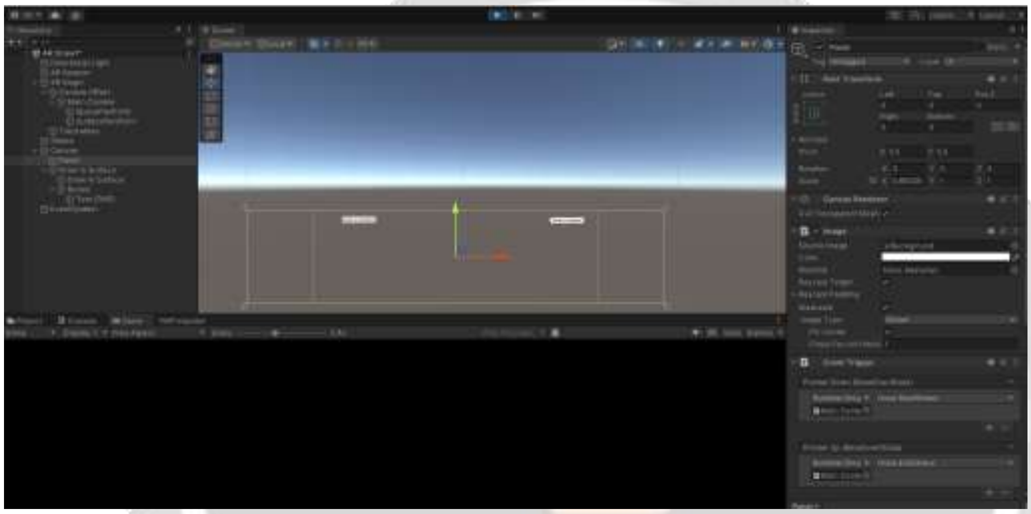


Figure 3.1

Figure 3.1 illustrates the general layout of the application where the user can choose two buttons which enables different pen points on the app.

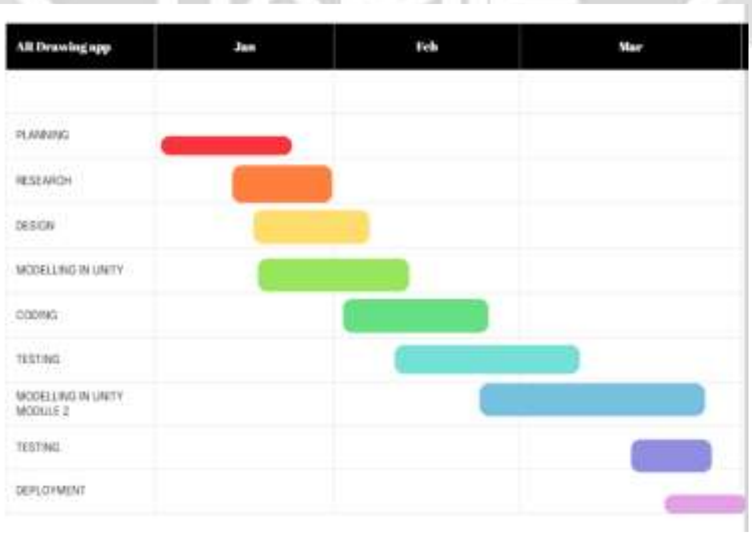


Figure 3.2

illustares the Gantt chart for the project - Figure 3.2

3.2.3 DETAILED METHODOLOGY:

The first thing that is imported is a module called an “AR Session.” An empty game object called “AR Session” and an “AR Session” component has been added. An AR Session is what starts and stops the Augmented Reality

session on the device. With this, we can start, stop, or pause any AR activities on the device. Next, added an input manager component (search “AR Input Manager”) to this game object.

AR Foundation:

The idea that can be started on augmented reality on the device but we need a way to take the information that’s being gathered by the device and turn it into usable 3D information that can translate over to Unity. To do this, we use another component called an “AR Session Origin.” This takes the tracked features in the real world and converts it to 3D coordinates that Unity can use. We’re going to add this into the scene in a different way than when we added the

AR Session. Plus XR Origin is added to enable the AR Camera which enables the application to access the mobile application.

3.3 KEY COMPONENTS OF AR FOUNDATION:

AR Session: The AR Session is a central component in AR Foundation. It manages the AR system's state and coordinates the interaction between the device and the AR experience. It is responsible for starting and stopping the AR experience, tracking the device's position and orientation in the real world, and managing session configuration changes (like switching between different types of AR experiences).

AR Session Origin: This component is used to define the coordinate system for the AR content. It typically contains the camera representing the user's viewpoint and is used to scale and position the AR content relative to the real world. Adjusting the AR Session Origin allows developers to control the placement and scale of AR objects in the scene.

Trackables: AR Foundation supports various types of trackables, such as planes (horizontal or vertical surfaces), images, faces, and feature points. These trackables represent real-world features detected by the AR system, and AR Foundation provides events and interfaces to interact with these elements, place content relative to them, or trigger actions based on their state.

AR Camera: The AR Camera in Unity acts as the user's viewpoint into the AR world. It's configured to match the physical properties of the device's camera, such as field of view, and it's used to render the AR content overlaid on the live camera feed. The AR Camera is typically a child of the AR Session Origin to ensure that the virtual content and real world are correctly aligned.

Key Features of AR Foundation:

Cross-platform support: Write your application once and deploy it on both iOS and Android devices, leveraging the best features of ARKit and ARCore.

Rich AR features: Support for a wide range of AR experiences, including plane detection, image and face tracking, environmental understanding (like occlusion and depth), and light estimation to make virtual content more realistic.

Scalable and flexible: AR Foundation is designed to be extensible, allowing developers to add new features or third-party plugins to enhance their AR experiences. It also scales from simple to complex AR projects.

3.3.1 INITIALIZING THE APPLICATION:

Install AR Foundation Package: Use the Unity Package Manager to install AR Foundation along with the ARCore and ARKit plugins for Android and iOS support, respectively.

Set Up Your Project: Create a new project, configure your project's settings for AR development, and set up the AR Session and AR Session Origin in your scene.

Start Building: Begin by adding AR features to your project, experimenting with plane detection, image tracking, or any other supported AR capabilities.

Unity's official documentation and the Unity Asset Store offer extensive resources, examples, and tutorials to help get started with AR Foundation and to advance your AR development skills.

3.3.2 U.I DESIGN:

Step 1: Identifying the needs and functions

Identify the core features your app will offer and what controls the user will need. Common features for an AR drawing app might include:

Color selection: To change paint colors.

Brush size adjustment: To alter the thickness of the strokes.

Tool selection: Different drawing tools or brushes.

Undo/Redo: For correcting mistakes.

Save/Export: To save or share the artwork.

Step 2: Implementing the UI in Unity

Using Unity's UI system, you can start bringing your design to life.

Creating a Canvas:

Methods:

GameObject > UI > Canvas. This will be the parent for your UI elements. Set the Render Mode to Screen Space - Overlay or Screen Space - Camera depending on your needs. Overlay is simpler and renders on top of everything, while Camera can integrate UI depth into the scene.

Add UI Elements:

Added UI components such as Buttons, Sliders, and Toggles for your tools. These can be found under GameObject > UI.

For a color picker, the features are chosen to use a custom UI or integrate a pre-existing solution.

Customize the UI:

Used the Inspector to customize the UI elements with colors, icons, and fonts that match your app's style.

Step 4: Script the UI Interactions

Connected the UI with the AR drawing functionality.

Certain scripts are needed such that:

1.Change the drawing color when a new color is selected.

2.Adjust the brush size based on slider value.

3.Change drawing tools based on user selection.

4. Saving the drawing or take a screenshot of the AR scene.

Step 3: Testing and Iteration

AR apps can be particularly challenging to get right due to the variety of real-world environments they can be used in. Test your app under different conditions and on multiple devices.

3.4 PREFABS IN UNITY:

In Unity, a Prefab is a type of asset that allows you to store a preconfigured game object complete with all its components, property values, and child game objects as a reusable template. Using Prefabs in an AR drawing app can significantly streamline development, ensuring consistency, reusability, and efficiency, especially for elements that are instantiated multiple times or require uniform functionality, such as drawn strokes, UI elements, or interactive objects within the AR environment.

How Prefabs Can Be Utilized in an AR Drawing App:

Drawing Strokes or Lines:

Each stroke made by the user can be instantiated from a Prefab. This Prefab would typically include a LineRenderer or a MeshRenderer component that represents the visual aspect of the stroke. By adjusting the properties of these components, you can change the appearance of the strokes, such as their color, width, and material.

3.4.1 UI Elements:

Common UI elements like buttons for changing tools, sliders for brush size, or color pickers can be made into Prefabs. This allows for easy duplication across different parts of the app while maintaining a consistent look and functionality.

Interactive Objects:

In more complex AR drawing apps, you might include interactive 3D objects that users can place in the world. These objects could be anything from simple geometric shapes to complex animated characters. Creating each of these objects as a Prefab allows for easy instantiation and management within the app.

Markers or Anchors:

For apps that allow drawing on detected surfaces or at specific points in space, Prefabs can be used for visual markers or anchors that highlight where the user can draw.

Benefits of Using Prefabs:

Efficiency: Prefabs allow for quick instantiation of objects during runtime, which is especially useful for apps that need to dynamically create content based on user input.

Consistency: By defining the properties and components of an object in a Prefab, you ensure that every instance of the Prefab behaves and appears the same way, maintaining consistency across your app.

Reusability: Once you've created a Prefab, it can be reused in different parts of your app or even in other projects. This saves time and effort as you don't need to recreate common elements from scratch.

Easy Updates: If you need to change an object's design or behavior, you can simply modify the Prefab, and all instances of the Prefab in your scenes will automatically update to reflect these changes.

3.4.2 Creating and Using Prefabs in Unity for the proposed App:

Create the GameObject: Start by creating a GameObject in your scene that represents the object you want to turn into a Prefab (e.g., a drawing stroke). Add and configure all necessary components such as MeshRenderer, LineRenderer, Collider, or custom scripts.

Create the Prefab: Drag the configured GameObject from the Hierarchy window into the Project window. This action creates the Prefab asset. The GameObject in the scene will automatically become an instance of this new Prefab.

Instantiate Prefabs at Runtime: Use Unity scripting to instantiate Prefabs during runtime based on user interactions. For drawing actions, you would typically listen for input events (e.g., touch or pointer down) and instantiate stroke Prefabs at the appropriate location within the AR scene, adjusting their properties as necessary to match the user's input.

Modify Prefabs: If you need to make changes, you can either directly edit the Prefab in the Project window or make changes to an instance in the scene and apply those changes back to the Prefab.

Using Prefabs efficiently in your AR drawing app development process can greatly enhance productivity, ensure a cohesive user experience, and provide flexibility to experiment with and iterate on your app's design and functionality.

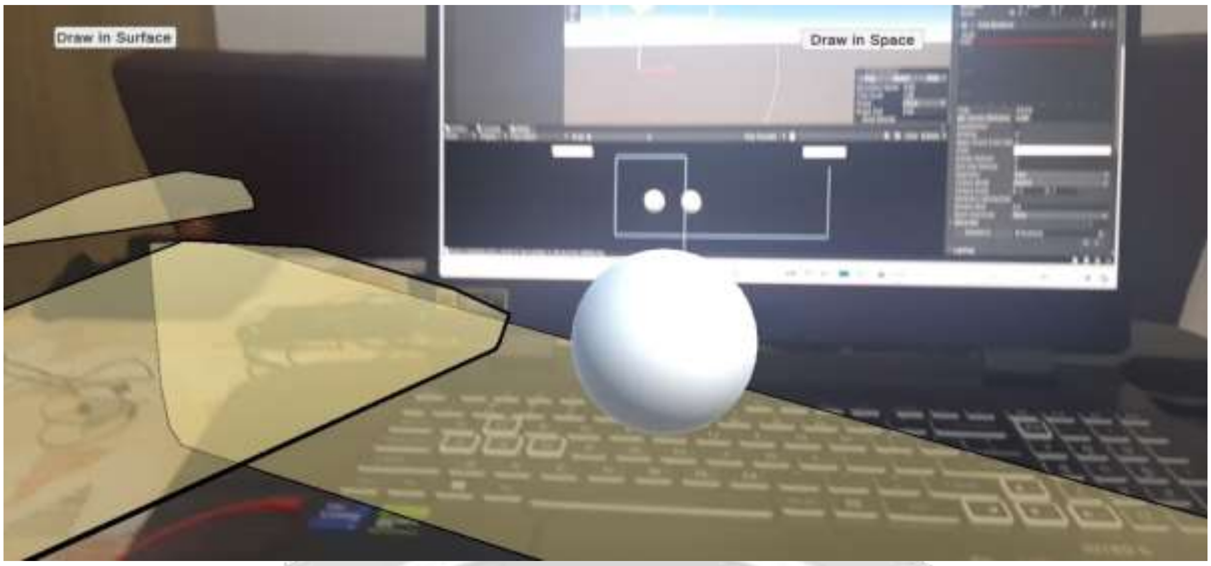


Figure 3.3

Figure 3.3 illustrates the pen point which acts inaccordance with the surface which has been detected by the application

3.5 SCRIPTING:

Unity primarily uses C# for scripting, leveraging the .NET framework to provide a comprehensive environment for game and application development. While C++ is widely used in game development and underpins many game engines, including Unity itself, Unity does not directly support C++ for scripting game logic and user interactions in the same way it does for C#.

However, there are scenarios where C++ might still play a role in a Unity project, especially when dealing with native plugins or when performance-critical sections of code need to be optimized beyond what's achievable with C#. Here's how C++ can be involved in Unity development:

Using C++ with Native Plugins

Native Plugin Creation:

Performance-critical code are developed in C++ and compiled it into a native plugin (.dll on Windows, .so on Linux, .dylib on macOS, or .a/.so on iOS/Android). This native plugin can then be called from C# scripts in Unity. This is particularly useful for heavy computational tasks, accessing third-party libraries, or features not available through Unity's API.

Integration with Unity:

After creating the native plugin, The Unity project has been integrated by placing the compiled files in the Assets/Plugins folder. Unity will automatically detect these plugins and make them available for use in your project.

Calling Native Code from C#:

Invoked functions from your native plugin by using Platform Invoke (P/Invoke) in your C# scripts. P/Invoke is a feature of the .NET framework that allows C# code to call native functions from DLLs.

3.5.1 CONSIDERATIONS FOR C++ IN UNITY:

Complexity: Using C++ in Unity projects adds complexity, especially in terms of project setup, building, and debugging. It's generally recommended only when necessary for performance reasons or when accessing specific hardware or software features not exposed by Unity.

Cross-Platform Development: If your Unity project targets multiple platforms, you'll need to compile your C++ code for each platform you intend to support. This can significantly increase development and testing efforts.

Performance: While native plugins can offer performance benefits, they should be used judiciously. Often, well-optimized C# code can achieve performance sufficient for most applications, and Unity's ongoing improvements continue to reduce the performance gap between managed and unmanaged code.

Compatibility and Maintenance: Maintaining compatibility with future versions of Unity and across different platforms can be more challenging with native plugins, as changes to Unity's architecture or platform-specific requirements might necessitate updates to your C++ code.

In summary, while Unity does not support C++ for direct scripting in the same way it supports C#, C++ can be used through native plugins to extend Unity's capabilities where necessary. This approach is best reserved for specific use cases where the benefits outweigh the added complexity and maintenance overhead.

3.6 PAINT STROKE EFFECT IN UNITY:

Creating a paint stroke effect in Unity can be approached in various ways depending on the Environment, in this case, to work on a 3D space, with accordance with the augmented reality (AR) context. The LineRenderer component is used dynamically to draw lines that follow the user's input, simulating a painting or drawing action. Here's a basic outline of how you might implement such a feature:

1. Setting Up the LineRenderer

An object called Stroke is created to hold the LineRenderer component. GameObject which is a stroke is added such as LineRenderer from the Components menu (Component -> Effects -> LineRenderer).

Configured the LineRenderer in the Inspector to suit the application's needs.

Important properties include:

Materials: Assign a material that fits your desired aesthetic for the stroke.

Width: Set the start and end width to determine how thick the line is.

Colors: Define the start and end colors of your line.

2. Scripting the Paint Stroke

Strokes are scripted to manage the drawing. This script will listen for user input, such as mouse movement or touch, and update the LineRenderer to create the stroke. Here's a basic example in C#:

```
using UnityEngine;

[RequireComponent(typeof(LineRenderer))]

public class PaintStroke : MonoBehaviour

{

    private LineRenderer lineRenderer;

    private Vector3 startPosition;

    private bool isDrawing = false;

void Awake()

    {

        lineRenderer = GetComponent<LineRenderer>();

    }

    void Update()

    {

        Draw();

    }
    void Draw()
    {
        if (Input.GetMouseButtonDown(0))
        {
            startPosition = Camera.main.ScreenToWorldPoint(Input.mousePosition);
            startPosition.z = 0; // Ensure the z position is consistent
            lineRenderer.positionCount = 1;
            lineRenderer.SetPosition(0, startPosition);
            isDrawing = true;
        }

        if (Input.GetMouseButton(0) && isDrawing)
        {
            Vector3 currentPos = Camera.main.ScreenToWorldPoint(Input.mousePosition);
            currentPos.z = 0;
            if (Vector3.Distance(currentPos, startPosition) > 0.1f) // Update to control the density of the line points
            {
                lineRenderer.positionCount++;
                lineRenderer.SetPosition(lineRenderer.positionCount - 1, currentPos);
                startPosition = currentPos;
            }
        }

        if (Input.GetMouseButtonUp(0))
        {
            isDrawing = false;
        }
    }
}
```

Notes on the Script:

Input Handling: This script uses Input.GetMouseButtonDown, Input.GetMouseButton, and Input.GetMouseButtonUp to manage drawing. For touch input, The equivalent touch events are used.

Line Creation: When the user starts drawing, the script initializes the LineRenderer with a single point. As the user moves the mouse (or finger), new points are added to the line.

Performance Considerations: This basic implementation adds a new point for every frame of input. Depending on your application's needs, you might want to optimize this by spacing out points or simplifying the line after it's drawn.

3. Optimizing and Expanding

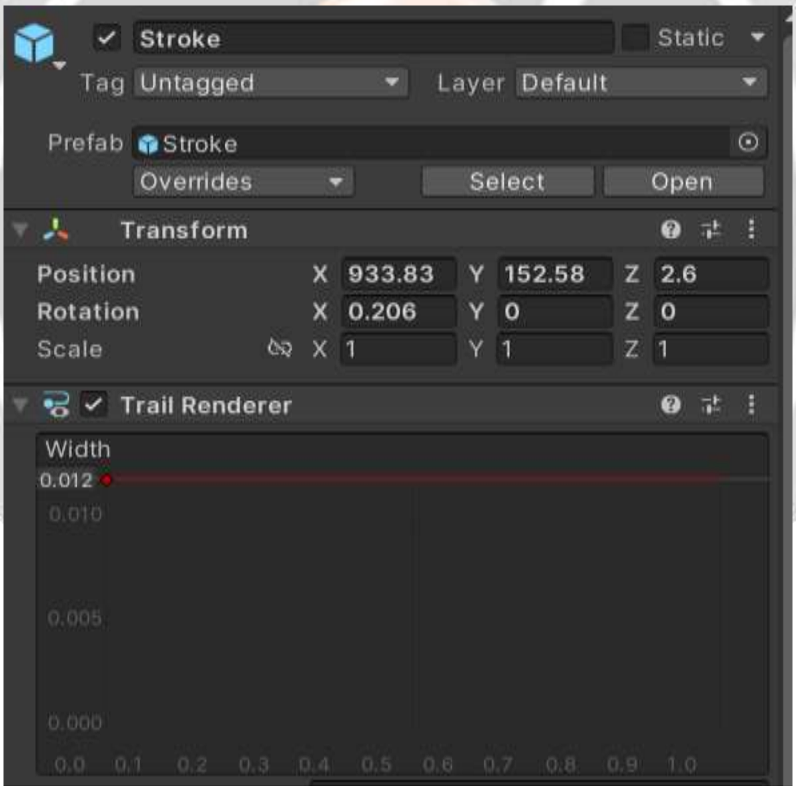
For more complex applications, especially those requiring high performance or supporting extensive drawing functionality, consider the following enhancements:

Line Simplification: Use algorithms like the Ramer-Douglas-Peucker algorithm to reduce the number of points in a stroke without significantly altering its appearance.

Pooling: If your application creates and destroys many lines, implement object pooling to reuse LineRenderer components and reduce garbage collection.

Brush Effects: To create different brush effects, you can experiment with various materials and shaders, or even dynamically change the LineRenderer's properties based on drawing speed or pressure (if using a pressure-sensitive input device).

Implementing paint strokes in Unity allows for a broad range of creative applications, from simple drawing programs to complex painting simulations. Experimentation and optimization will be key to achieving the best results for your specific project's needs.


Stroke
Static
Tag Untagged
Layer Default
Prefab Stroke
Overrides
Select
Open

Transform

	X	Y	Z
Position	933.83	152.58	2.6
Rotation	0.206	0	0
Scale	1	1	1

Trail Renderer
Width
0.012

Figure 3.4

Figure 3.4 illustrates the brush stroke prefab’s positions and trail size

3.7 PERFORMANCE METRICS:

Frame Rate (FPS): Targeted a stable frame rate, typically 30fps for basic AR experiences and 60fps for more interactive or high-end devices.

Memory Usage: Monitored the app memory usage to avoid crashes or slowdowns, especially on devices with limited resources.

CPU/GPU Usage: Optimized the usage to prevent overheating and battery drain.

Latency: Ensure minimal delay between user actions and visual feedback, crucial for drawing applications.

2. Testing Environments

Multiple Devices: Tested on a range of devices with varying hardware capabilities to ensure broad compatibility.

Lighting Conditions: The proposed app has been tested under different lighting conditions to ensure consistent performance and user experience.

Background Complexity: Tested against backgrounds with varying complexities to ensure the AR tracking remains stable.

3. Profiling and Optimization Tools

Unity Profiler: Used Unity's built-in profiler to track your app's performance in real-time, identifying bottlenecks in frame rate, memory usage, and CPU/GPU usage.

Unity Frame Debugger: Useful for debugging rendering issues and optimizing the rendering pipeline for better performance.

Third-party Tools: Considered tools namely ARFoundation (for cross-platform AR), EasyAR for additional profiling and optimization capabilities tailored to AR development.

4. Performance Testing Techniques

Stress Testing: Continuously used the drawing feature to test the app’s limits in terms of rendering and tracking stability.

Battery Usage: Monitored how long the app can run before significantly draining the device's battery.

Heat Management: Identified if extended usage leads to excessive device heating, which could throttle performance.

5. User Experience Testing

Responsiveness: Ensured the app responds quickly to user inputs without noticeable lag.

Accuracy: Tested the accuracy of drawing placement in the AR space, ensuring drawings align correctly with the real world.

User Interface: Ensured the UI remains responsive and informative under various conditions and use cases.

6. Optimization Strategies

Asset Optimization: Used compressed textures, LOD (Level of Detail) models, and efficient material settings.

Code Optimization: Noted inefficient algorithms or functions, especially those running every frame or requiring heavy computations.

AR Session Optimization: Optimize how your app initializes and manages AR sessions, including efficient tracking and light estimation.

3.7.1 PRECISION:

Precision in any Augmented Reality application is crucial for creating detailed and accurate artworks or designs directly onto the real-world environment. The proposed app leverage AR technology to overlay digital content onto the physical world, viewed through a device's screen. To enhance precision in such applications, developers and users can focus on several key areas:

Hardware Capabilities:

Device Sensors such as High-quality cameras, gyroscopes, accelerometers, and depth sensors can significantly enhance the AR experience by more accurately tracking movements and understanding the physical space. Processing Power of the appropriate devices with powerful CPUs and GPUs can better handle the complex calculations required for real-time AR rendering and tracking, leading to smoother and more precise drawing experiences. Need to examine the proposed device which must support ARCore compatibility.

Software Algorithms

Tracking and Scene Understanding: Advanced algorithms for motion tracking, plane detection, and spatial understanding help the app to precisely map the virtual drawings onto the real world. Technologies like SLAM (Simultaneous Localization and Mapping) are fundamental in these applications. Implemented unique software techniques that can predict and correct user input can help in achieving finer control over drawing. This can include smoothing algorithms to stabilize shaky hand movements.

3.7.2 BENEFITS OF THE PROPOSED APP:

The proposed AR (Augmented Reality) drawing app brings a fascinating blend of the digital and physical worlds to the forefront of creative expression and interaction. These apps allow users to draw and visualize their creations in real-time within their immediate environment, offering a unique and immersive experience. The benefits of using an AR drawing app are multifaceted, touching on educational, professional, and personal levels of engagement. Here are some key benefits:

- Enhanced Creativity and Imagination: the proposed AR drawing app push the boundaries of traditional art, allowing users to conceptualize and visualize their creations in 3D space. This not only enhances creativity but also encourages users to think outside the box and experiment with new forms of expression.

- Interactive Learning: For educational purposes, AR drawing can transform learning experiences by making them more interactive and engaging. This technology can be particularly effective in subjects such as art, design, and engineering, where visualizing concepts in three dimensions can significantly improve understanding and retention.

- Collaborative Work: the app with certain enhancements allows for multiple users to view and interact with the AR content simultaneously, making it an excellent tool for collaborative work in professional settings. Designers, architects, and educators can work together in real-time, regardless of their physical location, fostering teamwork and improving project outcomes.

- Accessible Art Creation: The proposed app democratize the process of art creation, making it accessible to people without traditional art skills. This can open up new avenues for creative expression for individuals who may not have engaged with art due to perceived skill barriers.

- Visualization and Prototyping: For professionals in design, architecture, and engineering, AR drawing apps serve as powerful tools for quick visualization and prototyping. Being able to draw and visualize changes in real-time within the actual environment can significantly speed up the design process and improve the accuracy of the final product.

4. PROPOSED WORK MODULES:

4.1 IMPORTING NDK AND SDK:

Importing and using the Android Native Development Kit (NDK) and Software Development Kit (SDK) in Unity is essential for developing Android applications with more complex, performance-critical parts that require native-level control. Unity uses these tools to compile and deploy games and apps to Android devices.

Install the Android SDK: During the Android Studio setup, the Android SDK has been installed. the NDK is installed via the SDK Manager. This is found under Tools > SDK Manager > SDK Tools tab. Check the box for "NDK (Side by side)" and click "OK" to install.

Step 1: Configuring Unity

the SDK and NDK has been installed via Android Studio, Configuring to find the location to help Unity navigate these tools.

Set the SDK and NDK paths:

For the Android SDK Location, navigate to the SDK location. To get the default location in Android Studio, it should be in a path similar to

C:\Users\[Your Username]\AppData\Local\Android\Sdk on Windows or ~/Library/Android/sdk on macOS.

For the Android NDK Location, the NDK is installed through Android Studio, it will be located within the SDK directory, under the ndk folder or a versioned folder within ndk. Select the appropriate NDK folder.

Set the JDK Path: Unity comes with an embedded version of OpenJDK, need not change this unless the specific need for a different version.

Step 2: Install Build Tools

Unity Hub: When creating a new project, you can also install Android Build Support through Unity Hub by selecting it as a module. This includes the necessary tools to build and run your game on Android. Android Studio's SDK Manager has been ensured whether the latest build tools installed. Unity requires them to compile the Android version of your game. We can manage these through Android Studio's SDK Manager.

Step 3: Build and Run

With your project open in Unity, went to File > Build Settings, select Android, and click Switch Platform if you haven't already.

Click Build and Run to compile your game and run it on a connected Android device.

4.2 XR ORIGIN:

In Unity, "XR Origin" refers to a component that acts as a foundational element in setting up an XR (Extended Reality) project, which includes Virtual Reality (VR) and Augmented Reality (AR) experiences. The XR Origin essentially serves as the root object in your scene that defines the starting point or "origin" for your XR experience, including the position and orientation of the user in the virtual world. It's part of the XR Interaction Toolkit, a Unity package that provides a set of tools and components for building immersive cross-platform XR applications.

Key Components of XR Origin

The XR Origin typically includes several key components to manage the XR environment effectively:

Camera Rig: This includes a main camera that represents the user's viewpoint. In a VR setup, this camera moves according to the user's head movements. In an AR setup, it might represent the view from a device's camera or a virtual camera moving within an augmented environment.

Tracking Space: This defines the coordinate system for the XR environment. The XR Origin's position and orientation are critical for tracking and mapping the user's movements accurately in the virtual or augmented world.

Input Handling: The XR Origin setup also includes mechanisms for handling input from various sources, like hand tracking, controllers, or other XR-compatible input devices. This allows for interaction with the virtual environment in a way that feels natural and intuitive.

Setting Up XR Origin in Unity

To set up an XR Origin in Unity, The following steps needs to be followed, assuming that the XR Interaction Toolkit has been installed:

Open Your Project: Start with a Unity project where you intend to implement XR features.

Installing XR Interaction Toolkit: installing the XR Interaction Toolkit via the Unity Package Manager. This toolkit provides the necessary components for building XR applications, including the XR Origin component.

Creating XR Origin: In the scene, By adding an XR Origin by creating a new GameObject and adding the XR Origin component to it. Alternatively, the XR Interaction Toolkit offers preconfigured prefabs like Room-Scale XR Rig or Stationary XR Rig that you can drag into your scene. These prefabs are set up with an XR Origin and include a Camera Rig and controllers setup.

Configuring Camera and Controllers: Ensuring the main camera is properly set up as a child of the XR Origin, and configured any controllers or input devices you plan to use. The toolkit provides components to easily map controller inputs to actions within the application.

Adjust Settings: Depending on the project's requirements, Certain settings needs to be adjusted related to tracking, input, or interaction. This could involve fine-tuning the camera settings, input actions, or adding additional components for specific XR interactions.

Test Your Setup: It's important to test your XR Origin setup within the Unity Editor using an XR Simulator (if available) or on an actual XR device to ensure everything is working as expected.

The XR Origin component and its associated setup play a crucial role in ensuring that your XR application behaves as intended, providing a stable and accurate foundation for user interaction and immersion in the virtual or augmented world. Unity's XR Interaction Toolkit simplifies this process, making it more accessible for developers to create compelling XR experiences.

4.3 C ++ SCRIPTS:

In Unity, the primary languages for scripting are C# and, to a lesser extent, JavaScript (UnityScript, which is now deprecated) and Boo (also deprecated). Unity does not directly support C++ for scripting purposes in the same way it supports C#. However, C++ can still be utilized within Unity projects through the use of native plugins. This allows developers to write performance-critical code in C++, which can then be called from C# scripts. This approach is particularly useful for tasks that require heavy lifting and could benefit from the performance optimizations C++ offers.

The following application used certain scripts such as Draw, DrawOnSurface, PenManager and Stroke.

4.3.1 Draw Script:

The way the proposed app allows to draw in 3D space is by simply spawning the “Stroke” prefab at the position of the pen point (the sphere that’s parented to the camera). the “Draw” script has been opened which allows to gain access to the pen point and our Stroke prefab by creating two public variables of type “Gameobject”.

```
using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Draw : MonoBehaviour

{

   public bool mouseLookTesting;

   public GameObject stroke;

   public GameObject spacePenPoint;

   public GameObject surfacePenPoint;

   public static bool drawing = false;

   private float pitch = 0;

   private float yaw = 0;

   // Start is called before the first frame update

   void Start()

   {

   }

   void Update()

   {
        if (mouseLookTesting)
        {
            yaw += 2 * Input.GetAxis("Mouse X");
            pitch -= 2 * Input.GetAxis("Mouse Y");
            transform.eulerAngles = new Vector3(pitch, yaw, 0.0f);
        }
        if (PenManager.drawingOnSurface)
        {
            spacePenPoint.GetComponent<MeshRenderer>().enabled = false;
            surfacePenPoint.GetComponent<MeshRenderer>().enabled = true;
        }
        else
        {
            surfacePenPoint.GetComponent<MeshRenderer>().enabled = false;
            spacePenPoint.GetComponent<MeshRenderer>().enabled = true;
        }
    }
    public void StartStroke()
    {
        GameObject currentStroke;
        drawing = true;
        currentStroke = Instantiate(stroke, spacePenPoint.transform.position, spacePenPoint.transform.rotation) as
GameObject;
    }
    public void EndStroke()
    {
        drawing = false;
    }
}
```

4.3.2 PENMANAGER SCRIPT:

Two buttons are created to enable two different penpoints to get trigged which need a way to make these buttons actually call certain methods that will switch between our drawing types.

The new script called “PenManager” needs to be created and needs to be attached to the AR Camera.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class PenManager : MonoBehaviour
```

```
{
   public static bool drawingOnSurface = false;
   public void DrawOnSurface()
   {
      drawingOnSurface = true;
   }
   public void DrawInSpace()
   {
      drawingOnSurface = false;
   }
}
```

4.3.3 STROKE SCRIPT:

To add the logic that will make the stroke track with the transform of the pen point which, in turn, the problem has been fixed by not being able to see it. Here is the script:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Script : MonoBehaviour
{
   private GameObject penPoint;
   void Start()
   {
     penPoint = GameObject.Find("PenPoint");
   }

   // Update is called once per frame
   void Update()
   {
     if (Draw.drawing)
     {
        this.transform.position = penPoint.transform.position;
        this.transform.rotation = penPoint.transform.rotation;
     }
     else
```

```
        {
            this.enabled = false;
        }
    }
}
```

4.3.4 TESTING THE APPLICATION:

AR Tracking and Performance

Surface Detection and Stability has been tested to ensure how well the app detects and tracks surfaces, including walls, floors, and objects. Checked for jitter, drift, or loss of tracking under different conditions (e.g., varying lighting, textures, and colors).

Performance has been Monitored to estimate the app's performance, especially its frame rate, under various conditions. High-quality AR experiences require a smooth frame rate to prevent user discomfort.

Multi-Anchor Support has been ensure to make sure the proposed app supports drawing on multiple surfaces or from different angles and further testing has been conducted to check the stability and reliability of these features.

Compatibility Test:

Device Compatibility: Tested the app on a wide range of devices with different hardware capabilities to ensure consistent performance. This includes testing on various smartphone models and AR glasses.

Environmental Variability: Tested the app in various real-world environments to ensure it can handle different lighting conditions, indoor and outdoor settings, and diverse surface textures.

5. RESULTS AND DISCUSSION

The proposed application gives us a final program that allows the user to draw on Surface or Draw in space. Based upon the user's intention, The user has to press the option they need and based upon the user's selection, The Different Pen point has been triggered which caused the particular sphere to pop out. As the application is started, the program detects certain points in the 3d space and triggers a surface area to indicate that the space has been detected and the space is available for the user to draw in. Figure 5.1 illustrates the plane spawning method.

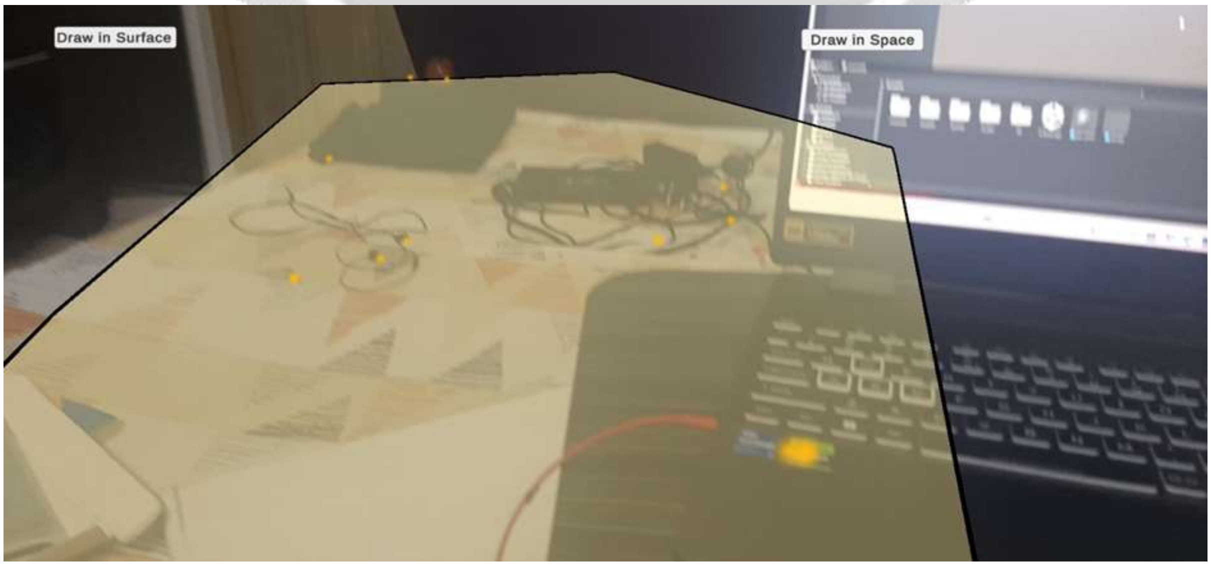


Figure 5.1

The app then demands the user to choose between DrawOnSurface or DrawInSpace buttons which causes the application to spawn the different Penpoint spheres which further lets the application to get the desired outlook. Figure 5.2 illustrates the spawing of Penpoint in accordance with the option chosen by the user.

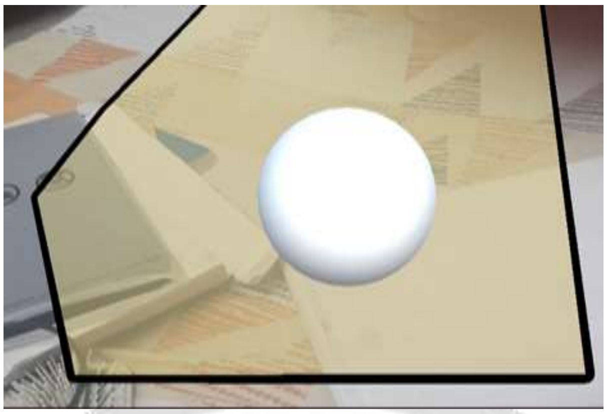


Figure 5.2

Then the user begins to draw on the 3d space according to his wish. The option to change the color, and intensity are also added to enable a more optimizable U.I for the user to experiment on. The color and intensity of the pen stroke can be changed by adjusting the slider in the application.

The AR drawing app demonstrates a high level of innovation and user engagement potential. However, testing has identified areas for improvement in UI/UX design, performance under certain conditions, and cross-device functionality.

Functionality Testing

Drawing Features

Success: All basic drawing functions (e.g., brush size, color selection) perform as expected across tested devices.

Issue: The "undo" feature is not implemented in the current stage of the app.

UI/UX

Success: Users found the app's interface intuitive for basic drawing tasks.

Improvement Needed: Several users noted difficulties in accessing advanced features, suggesting a need for a more streamlined navigation.

AR Tracking and Performance

Surface Detection

Success: The app effectively detects and tracks flat surfaces in well-lit conditions.

Limitation: Surface detection struggles in low-light conditions and with surfaces that lack distinct textures.

Performance Metrics

Findings: The app maintains a stable 30 FPS on high-end devices but drops to 15-20 FPS on older models, affecting the smoothness of AR effects.

User Input and Interaction

Touch Input

Success: Touch input is highly responsive on screens of all sizes.

Issue: Stylus input isn’t implemented on this version of the app

Integration and Compatibility

Device Compatibility

Findings: The app performs well on devices less than 3 years old. Performance issues and crashes were noted on older devices, particularly those not meeting the recommended RAM specifications.

Operating System Versions

Success: No significant issues across the latest versions of Android.

Observation: Some features exhibit inconsistent behavior on Android versions older than 9.0.

Real-World Usability

Environmental Factors

Success: Users report that outdoor use during daylight presents no significant issues.

Challenge: Indoor use in artificial lighting introduces inconsistencies in tracking performance.

User Feedback

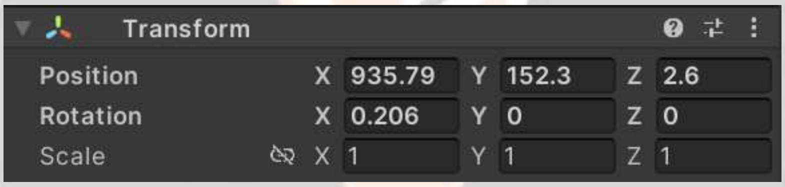
Positive: High praise for the immersive drawing experience and ease of starting with basic features.

Critique: Requests for more detailed tutorials for advanced features and better feedback on incorrect actions.

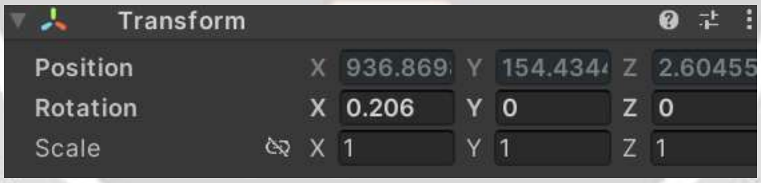
Security and Privacy

Finding: No major security vulnerabilities were identified. However, the app collects location data without explicit user confirmation, raising privacy concerns.

STROKE CHANGES:

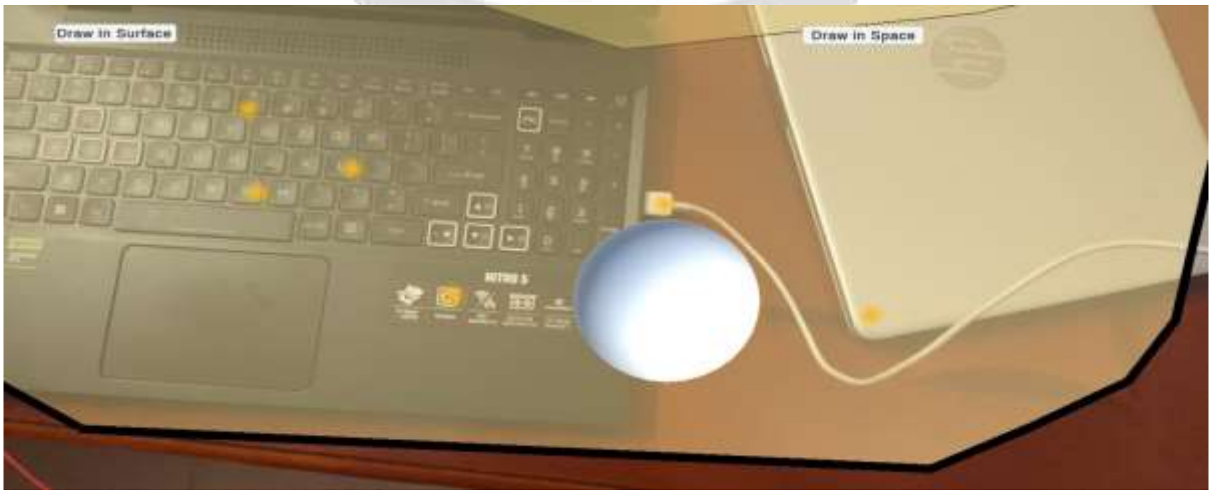


```
Transform
Position      X 935.79    Y 152.3    Z 2.6
Rotation      X 0.206     Y 0        Z 0
Scale         X 1         Y 1        Z 1
```



```
Transform
Position      X 936.869   Y 154.434  Z 2.60455
Rotation      X 0.206     Y 0        Z 0
Scale         X 1         Y 1        Z 1
```

5.1 FINAL OUTPUTS:



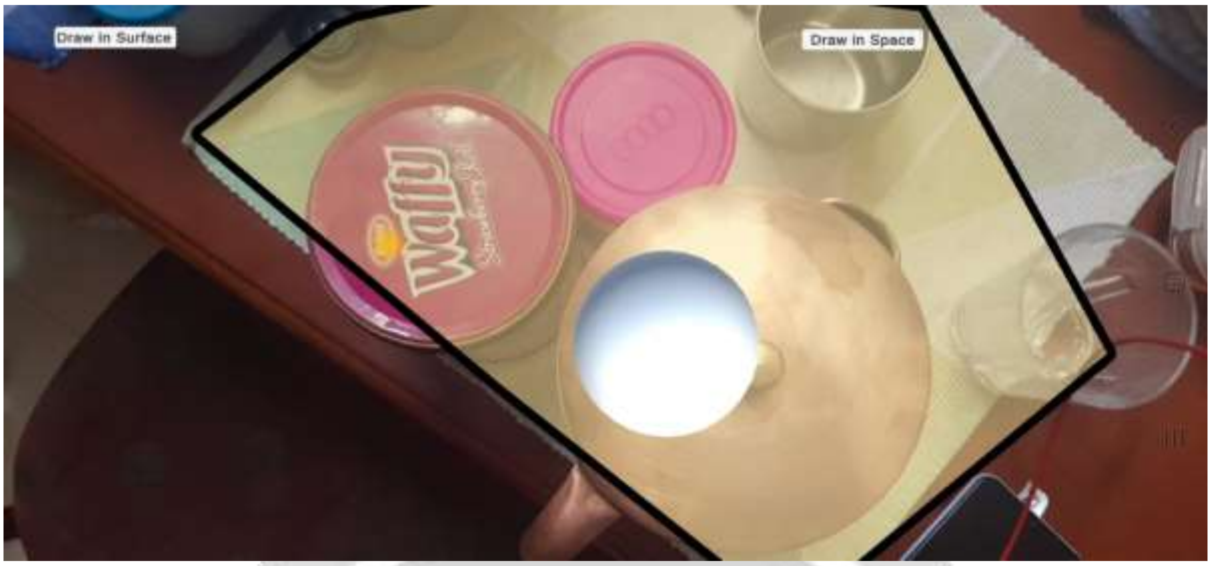


Figure 5.4 and 5.5

The figures 5.4 and 5.5 illustrates the detection of the plain area and the triggering of the penpoint is in the application.

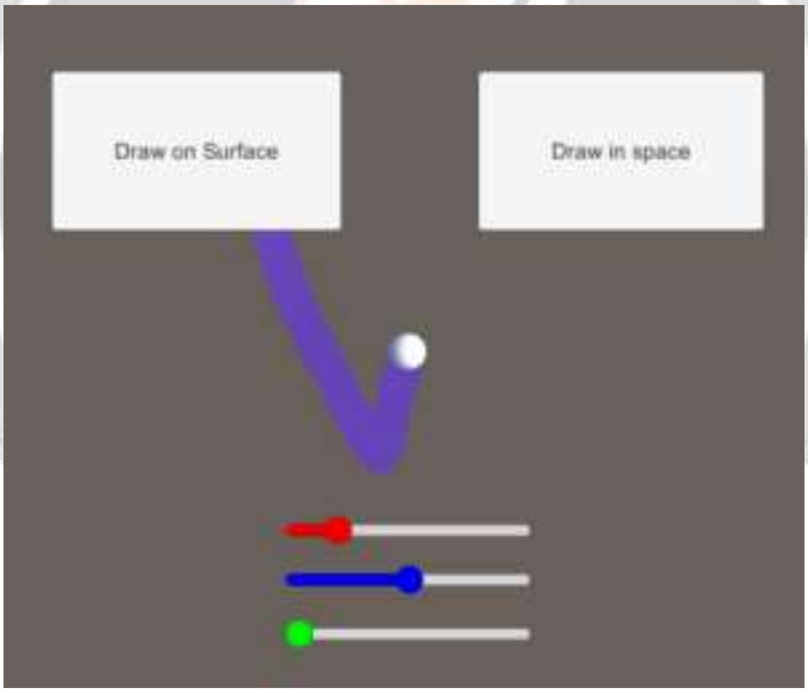


Figure 5.6 illustrates that the application is creating pen strokes.

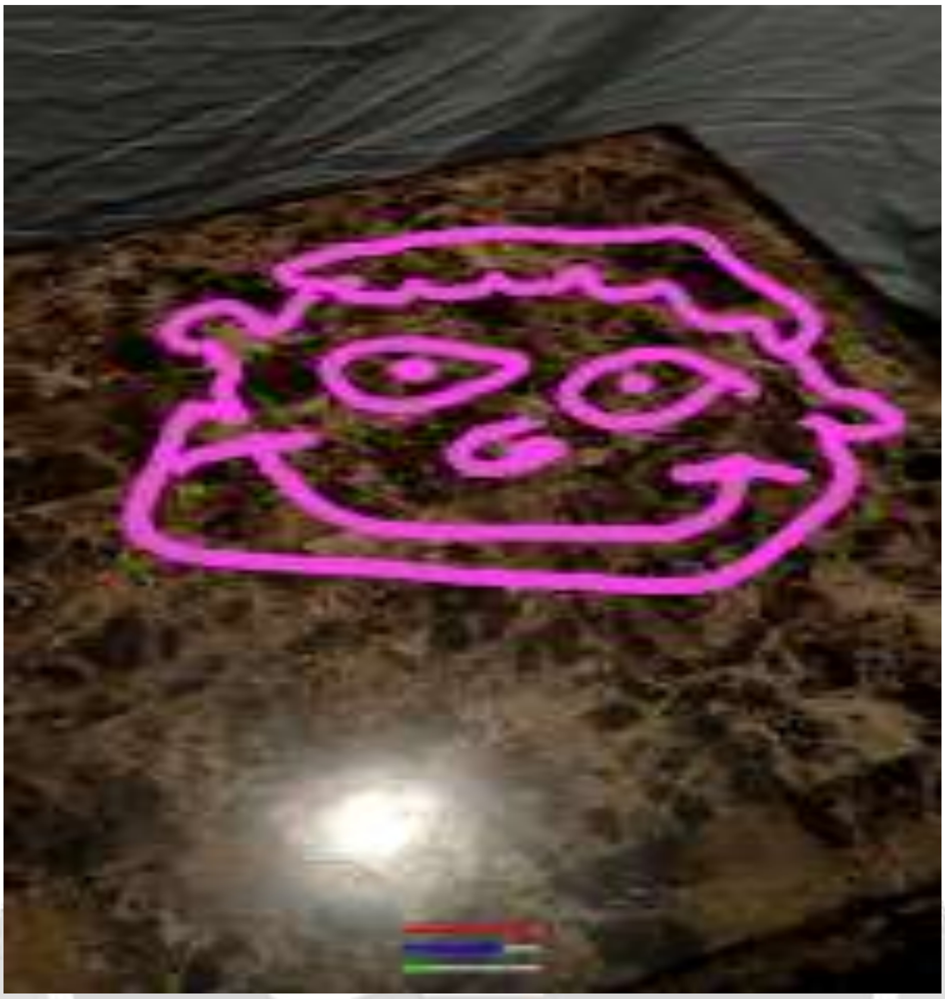


Figure 5.7 illustrates the application allows drawing in 3D space on the adetected plane.

6. CONCLUSION:

The testing and evaluation of the Augmented Reality (AR) Drawing App have provided valuable insights into its current performance, usability, and functionality across various dimensions. While the app demonstrates significant potential in offering a creative and immersive drawing experience in augmented reality, our findings indicate several areas requiring attention to enhance its overall user experience and performance.

Key strengths of the app include its intuitive user interface, effective surface detection in optimal lighting conditions, and responsive touch inputs. These features form a solid foundation for a compelling AR drawing experience, enabling users to easily engage with the app’s core functionalities. The positive feedback on the app's innovative approach to drawing in AR highlights its potential to captivate a broad user base. However, the evaluation also uncovered challenges that could hinder the app's adoption and enjoyment. Notably, issues such as performance drops on older devices, limited usability in low-light conditions, and inconsistencies across different operating systems need addressing. Furthermore, user feedback has pointed out areas for improvement in UI/UX design, particularly regarding the accessibility of advanced features and the need for clearer tutorials.

The recommendations provided aim to address these challenges, focusing on optimization for lower-end devices, enhancing the surface detection algorithm, and refining the UI/UX to make the app more accessible and enjoyable for all users. Additionally, addressing privacy concerns related to data collection practices will be crucial in building trust with users. Further research has also been conducted to include stylus input aswell.

In conclusion, the proposed app holds promise for revolutionizing how we interact with digital art and the environment around us. By taking proactive steps to address the identified issues, the development team can significantly improve the app's performance and user satisfaction. Implementing the recommended enhancements will not only elevate the user experience but also strengthen the app's market position as a whole. While benefiting creativity among users, The app also helps treat mental illnesses such as ADHD and Autism.

7. REFERENCES:

7.1 ACADEMIC RESEARCH:

AR Technology and User Experience

[1] Azuma, R. T. (1997). "A Survey of Augmented Reality." Presence: Teleoperators and Virtual Environments 6, 4 (August 1997), 355-385. This seminal paper provides foundational knowledge on AR technologies.

Billinghurst, M., Clark, A., & Lee, G. (2015). "A Survey of Augmented Reality." Foundations and Trends® in Human–Computer Interaction, 8(2-3), 73-272. Offers an overview of AR technologies with a focus on user interaction techniques.

AR in Art and Design

[2] Schmalstieg, D., & Hollerer, T. (2019). "Augmented Reality: Principles and Practice." Addison-Wesley. This book includes insights into AR's application in various fields, including art and design, providing a comprehensive background on technical and design principles.

[3] Hu, H., Gao, S., & Dai, G. (2021). "Exploring Augmented Reality for Digital Art Creation." International Journal of Creative Interfaces and Computer Graphics (IJCICG), 12(1), 1-15. Discusses the potential and challenges of using AR in creative processes.

7.2 Technical Documentation and Developer Guides

Unity Documentation

[4] Unity Technologies. "AR Foundation Documentation." Official documentation for Unity's AR Foundation, a tool for building AR experiences in Unity, offering guidelines on implementing features similar to those in an AR drawing app.

Android and iOS AR Development

[5] Google Developers. "ARCore Documentation." Provides technical guides and API references for developing AR applications on Android.

[6] Apple Developer. "ARKit Documentation." Offers comprehensive resources for developing AR apps on iOS devices, including user interface design and performance optimization tips.

User Feedback and Market Analysis

7.3 User Experience Design for AR

[7] Nielsen, J., & Norman, D. (2020). "Augmented Reality User Interface Design." Nielsen Norman Group. This article discusses best practices in AR UI/UX design, emphasizing the importance of user-centered design principles.

Market Reports

[8] MarketsandMarkets. (2023). "Augmented Reality Market Global Forecast." A market research report that provides insights into the AR market size, growth trends, and key players, helping contextualize the significance and market potential of AR drawing apps.

Software and AR Platforms

Vuforia Engine Documentation

[9] PTC Inc. "Vuforia Developer Portal." Offers resources for developers using Vuforia AR development platform, providing insights into creating interactive and engaging AR experiences.