Design Notes for AR/VR Development

Projects March 15, 2016

Cursor:

- Toggle
  - Switch out current reticle with particle generator
- Select different versions
  - Tail – decreasing radius with count
  - Combine – hex quad with paint

Sphere Rotation:

Rotate Sphere

Axis – controlled by tilt angle of device

Cursor

development notes

Now on to the animated cursor...

Note on moving between – Dev Build and Tango Build

in

```
TangoARPoseController
```

```
    /// public class TangoARPoseController : MonoBehaviour, ITangoLifecycle
    /// <summary>
    /// Update is called every frame.
    /// </summary>
    public void Update()
    {
        if (m_tangoARScreen.m_screenUpdateTime != m_poseTimestamp)
        {
            //_UpdateTransformation(m_tangoARScreen.m_screenUpdateTime);
            //userMovement.OnPoseUpdated(transform);
        }
    }
```

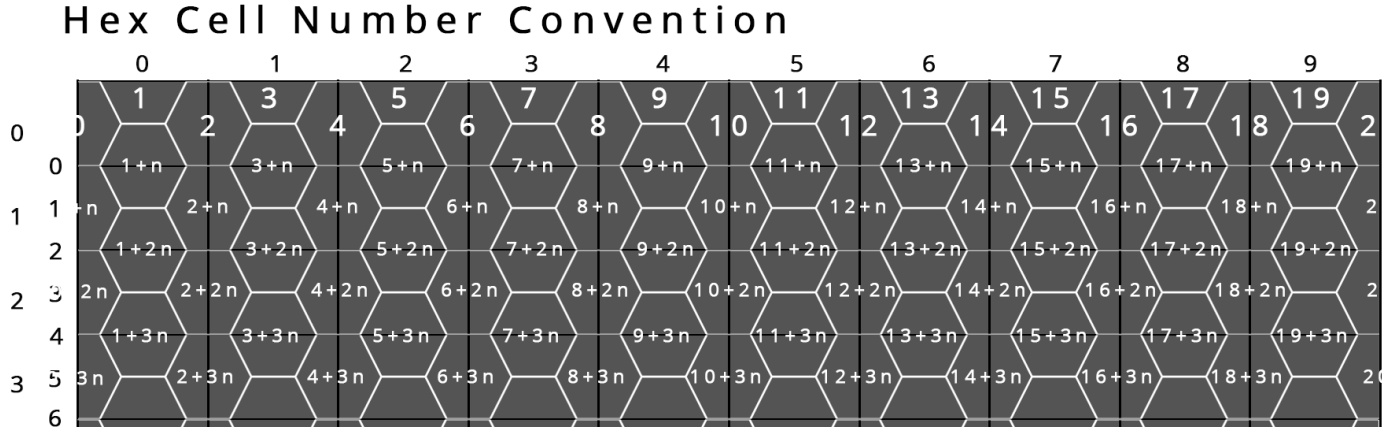
uncomment

```
            //_UpdateTransformation(m_tangoARScreen.m_screenUpdateTime);
            //userMovement.OnPoseUpdated(transform);
```

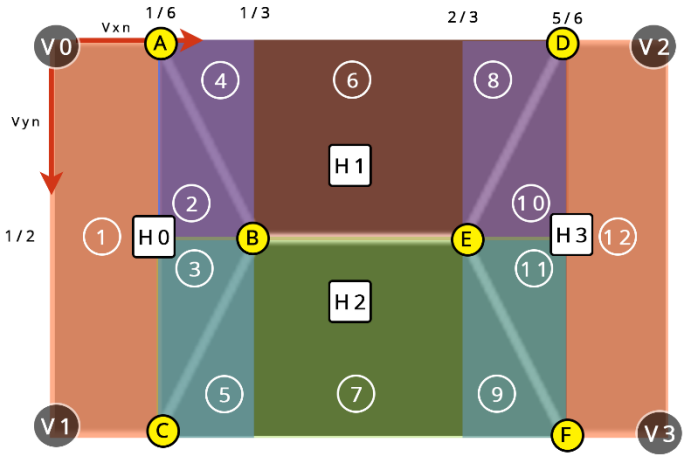
and return the Tango_AR_Camera to position = (0,0,0)

Complete next calculations on seam
Design Cursor animation possibilities

Alternate Numbering Convention

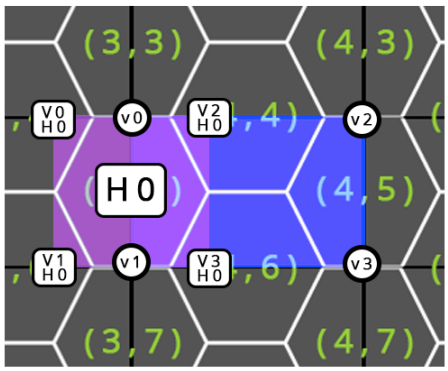


Hexagonal Cell Determination




```
Vxn = Norm(V2-V0)  Vyn = Norm(V1-V0)
Vp = Pos = V0;

If(dot(Vxn,Vp)<1/6) H = 0;
if(dot(Vxn,Vp)<5/6) H = 3;

if(dot(Vxn,Vp)>1/3 && dot(Vxn<2/3)
{
   if(dot(Vyn,Vp<1/2) H = 1;
   else H=2;
}

A = V0 + 1/6*Vxn;
B = V0 + 1/3*Vxn + 1/2*Vyn;
C = V1 + 1/6*Vxn;
D = V2 - 1/6*Vxn;
E = V2 - 1/6*Vxn + 1/2*Vyn;
F = V3 - 1/6*Vxn;

if(dot(Vxn,Vp)>1/6 && dot(Vxn,Vp)<1/3)
{
   if(dot(Vyn,Vp)<1/2)
   {
       if(dot(Cross(APos,AB),Fwd)>0) H = 0;
       else H = 1;
   }
   else
   {
       if(dot(Cross(CPos,CB),Fwd)<0) H=0;
       else H = 2;
   }
}

if(dot(Vxn,Vp)>2/3 && dot(Vxn,Vp)<5/6)
{
   if(dot(Vyn,Vp)<1/2)
   {
       if(dot(Cross(DPos,DE),Fwd)>0) H = 1;
       else H = 3;
   }
   else
   {
       if(dot(Cross(FPos,FE),Fwd)<0) H=2;
       else H = 3;
   }
}
```

Hex Quad Vertex Definitions

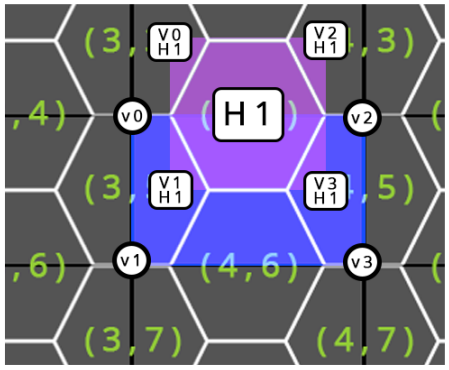


V0 H0 = v0 + 1/3*norm(v0-v2)*mag(v0-v1)

V1 H0 = v1 + 1/3*norm(v0-v2)*mag(v0-v1)

V2 H0 = v0 - 1/3*norm(v0-v2)*mag(v0-v1)

V3 H0 = v1 - 1/3*norm(v0-v2)*mag(v0-v1)



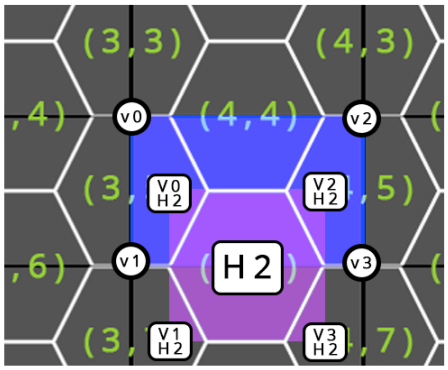
Vxn = norm(v0-v2) Vyn = norm(v0-v1)
Vxmag = mag(v0-v2) Vymag = mag(v0-v1)

V0 H1 = v0 - 1/6*Vxmag*Vxn + 1/2*Vymag*Vyn

V1 H1 = v0 - 1/6*Vxmag*Vxn - 1/2*Vymag*Vyn

V2 H1 = v2 + 1/6*Vxmag*Vxn + 1/2*Vymag*Vyn

V3 H1 = v2 + 1/6*Vxmag*Vxn - 1/2*Vymag*Vyn



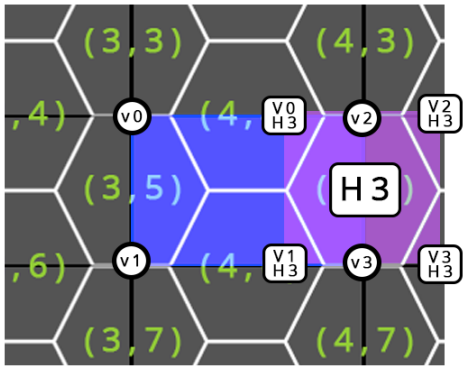
Vxn = norm(v1-v3) Vyn = norm(v1-v3)
Vxmag = mag(v1-v3) Vymag = mag(v1-v3)

V0 H2 = v1 - 1/6*Vxmag*Vxn + 1/2*Vymag*Vyn

V1 H2 = v1 - 1/6*Vxmag*Vxn - 1/2*Vymag*Vyn

V2 H2 = v3 + 1/6*Vxmag*Vxn + 1/2*Vymag*Vyn

V3 H2 = v3 + 1/6*Vxmag*Vxn - 1/2*Vymag*Vyn



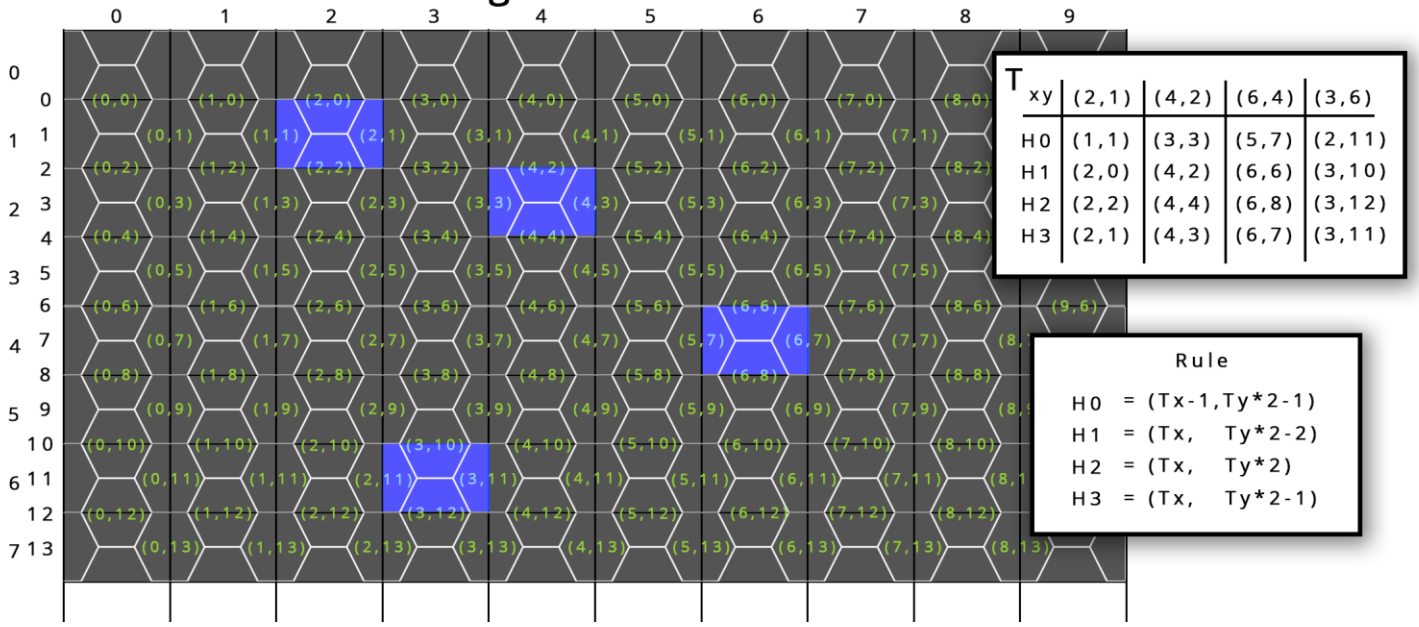
V0 H3 = v2 + 1/3*norm(v0-v2)*mag(v0-v1)

V1 H3 = v3 + 1/3*norm(v0-v2)*mag(v0-v1)

V2 H3 = v2 - 1/3*norm(v0-v2)*mag(v0-v1)

V3 H3 = v3 - 1/3*norm(v0-v2)*mag(v0-v1)

Hex Tile Indexing

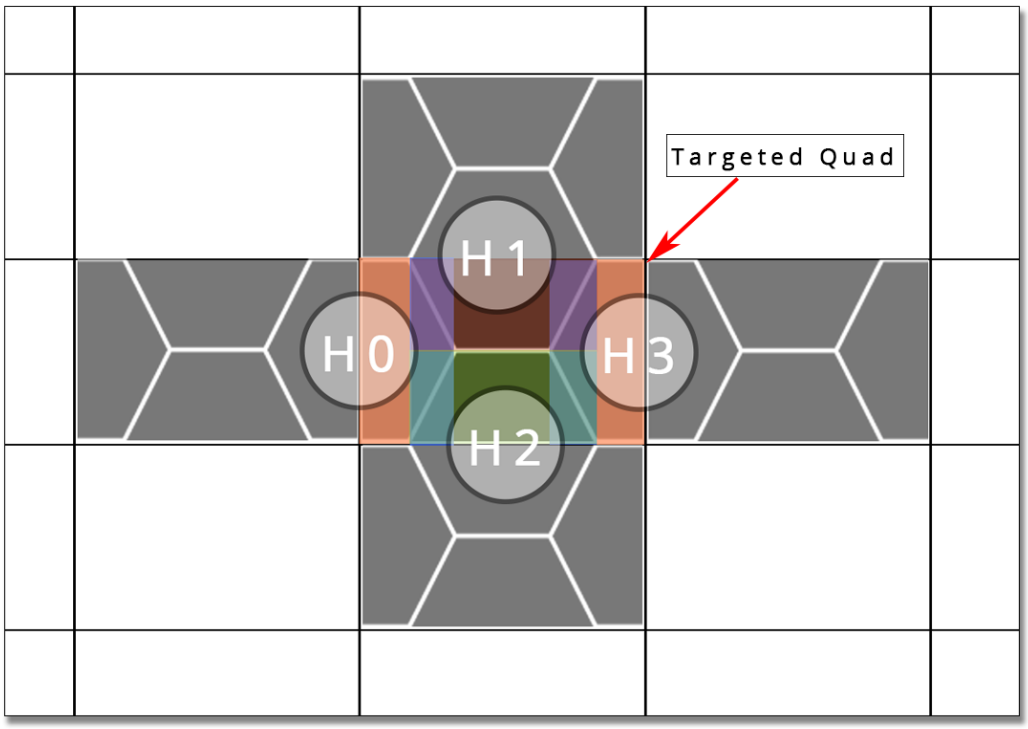


T xy	(2,1)	(4,2)	(6,4)	(3,6)
H0	(1,1)	(3,3)	(5,7)	(2,11)
H1	(2,0)	(4,2)	(6,6)	(3,10)
H2	(2,2)	(4,4)	(6,8)	(3,12)
H3	(2,1)	(4,3)	(6,7)	(3,11)

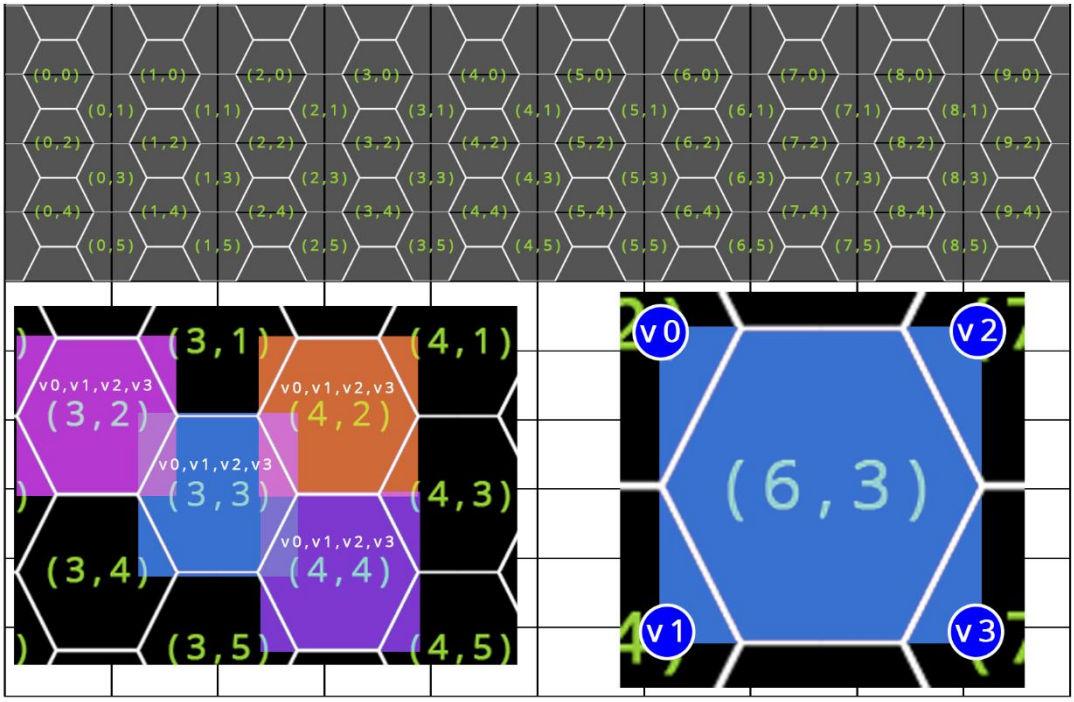
Rule

H0 = (Tx-1,Ty*2-1)
H1 = (Tx, Ty*2-2)
H2 = (Tx, Ty*2)
H3 = (Tx, Ty*2-1)

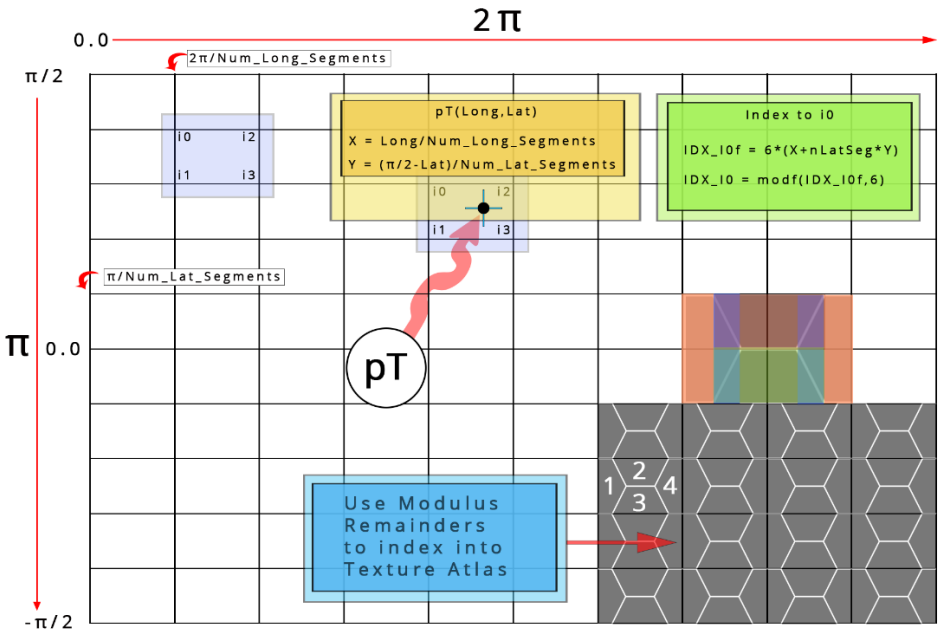
Cell Selection



Hex Tile Indexing



Quad Lookup Method given Point on Sphere



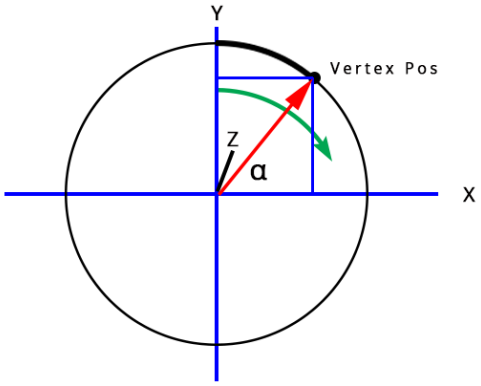
Important Note - ENTIRE MESH Must be written every frame – too much of a hit

α

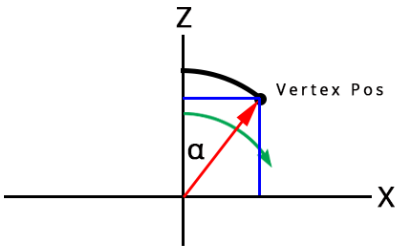
Sphere_Argos Mesh Indexing

Determine:
1. Latitude and Longitude Angle
2. Divide by Segment angles to get indices

```
public static MeshDraft Sphere_Argos(float radius, int longitudeSegments, int latitudeSegments)
    {
        var draft = new MeshDraft {name = "Sphere_Argos"};

        float longitudeSegmentAngle = Mathf.PI*2/longitudeSegments;
        float latitudeSegmentAngle = Mathf.PI/latitudeSegments;

        float currentLatitude = Mathf.PI/2;
        for (var ring = 0; ring <= latitudeSegments; ring++)
        {
            var currentLongitude = 0f;
            for (int i = 0; i < longitudeSegments; i++)
            {
                var point = PTUtils.PointOnSphere(radius, currentLongitude, currentLatitude);
                draft.vertices.Add(point);
                draft.normals.Add(point.normalized);
                //draft.uv.Add(new Vector2((float) i/longitudeSegments, (float) ring/latitudeSegments));
                draft.uv.Add(new Vector2((float)(i % 2), (float)(ring%2)));
                currentLongitude += longitudeSegmentAngle;
            }
            currentLatitude -= latitudeSegmentAngle;
        }

        int i0, i1, i2, i3;
        for (int ring = 0; ring < latitudeSegments; ring++)
        {
            for (int i = 0; i < longitudeSegments - 1; i++)
            {
                i0 = ring*longitudeSegments + i;
                i1 = (ring + 1)*longitudeSegments + i;
                i2 = ring*longitudeSegments + i + 1;
                i3 = (ring + 1)*longitudeSegments + i + 1;
                draft.triangles.AddRange(new[] {i0, i1, i2});
                draft.triangles.AddRange(new[] {i2, i1, i3});
            }

            i0 = (ring + 1)*longitudeSegments - 1;
            i1 = (ring + 2)*longitudeSegments - 1;
            i2 = ring*longitudeSegments;
            i3 = (ring + 1)*longitudeSegments;
            draft.triangles.AddRange(new[] {i0, i1, i2});
            draft.triangles.AddRange(new[] {i2, i1, i3});
        }

        return draft;
    }
}
```



Latitudeangle
Atan2(y,sqrt(x^2 + z^2))



Longitudeangle
Atan2(x/z)

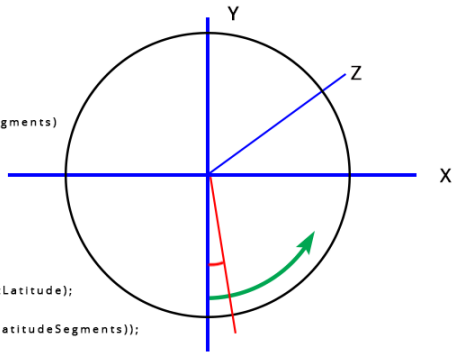
```
fCheckLong = Mathf.Atan2(point.x, point.z);

if(fCheckLong < 0.0f)
{
   fCheckLong = 2f * Mathf.PI + fCheckLong;
}
fCheckLat = Mathf.Atan2(point.y, Mathf.Sqrt(point.x*point.x+point.z*point.z));
```

(The Hive)

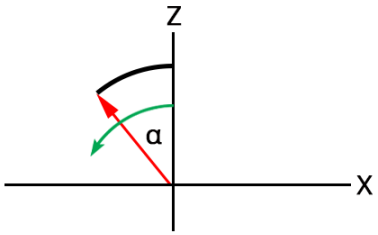
Sphere Mesh Indexing

Determine:
1. Latitude and Longitude Angle
2. Divide by Segment angles to get indices

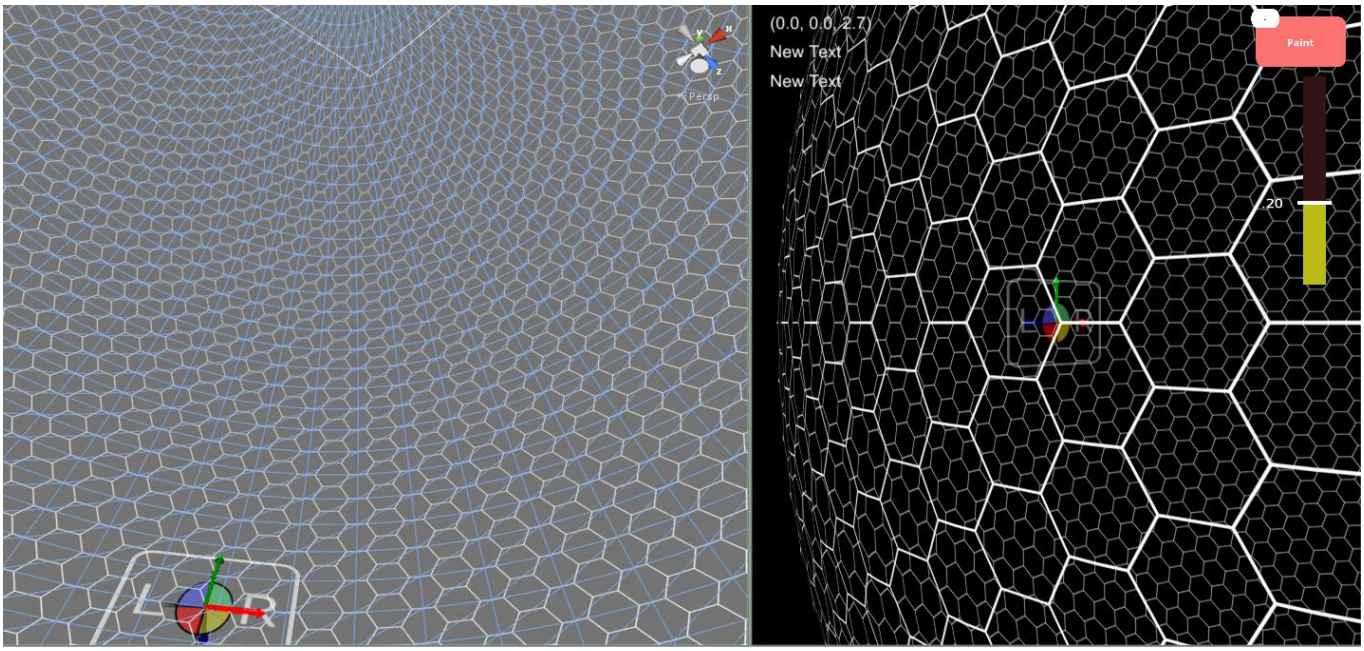
```
public static MeshDraft Sphere(float radius, int longitudeSegments, int latitudeSegments)
    {
        var draft = new MeshDraft {name = "Sphere"};

        float longitudeSegmentAngle = Mathf.PI*2/longitudeSegments;
        float latitudeSegmentAngle = Mathf.PI/latitudeSegments;

        float currentLatitude = -Mathf.PI/2;
        for (var ring = 0; ring <= latitudeSegments; ring++)
        {
            var currentLongitude = 0f;
            for (int i = 0; i < longitudeSegments; i++)
            {
                var point = PTUtils.PointOnSphere(radius, currentLongitude, currentLatitude);
                draft.vertices.Add(point);
                draft.normals.Add(point.normalized);
                //draft.uv.Add(new Vector2((float) i/longitudeSegments, (float) ring/latitudeSegments));
                draft.uv.Add(new Vector2((float)(i % 2), (float)(ring%2)));
                currentLongitude -= longitudeSegmentAngle;
            }
            currentLatitude += latitudeSegmentAngle;
        }

        int i0, i1, i2, i3;
        for (int ring = 0; ring < latitudeSegments; ring++)
        {
            for (int i = 0; i < longitudeSegments - 1; i++)
            {
                i0 = ring*longitudeSegments + i;
                i1 = (ring + 1)*longitudeSegments + i;
                i2 = ring*longitudeSegments + i + 1;
                i3 = (ring + 1)*longitudeSegments + i + 1;
                draft.triangles.AddRange(new[] {i0, i1, i2});
                draft.triangles.AddRange(new[] {i2, i1, i3});
            }

            i0 = (ring + 1)*longitudeSegments - 1;
            i1 = (ring + 2)*longitudeSegments - 1;
            i2 = ring*longitudeSegments;
            i3 = (ring + 1)*longitudeSegments;
            draft.triangles.AddRange(new[] {i0, i1, i2});
            draft.triangles.AddRange(new[] {i2, i1, i3});
        }

        return draft;
    }
}
```

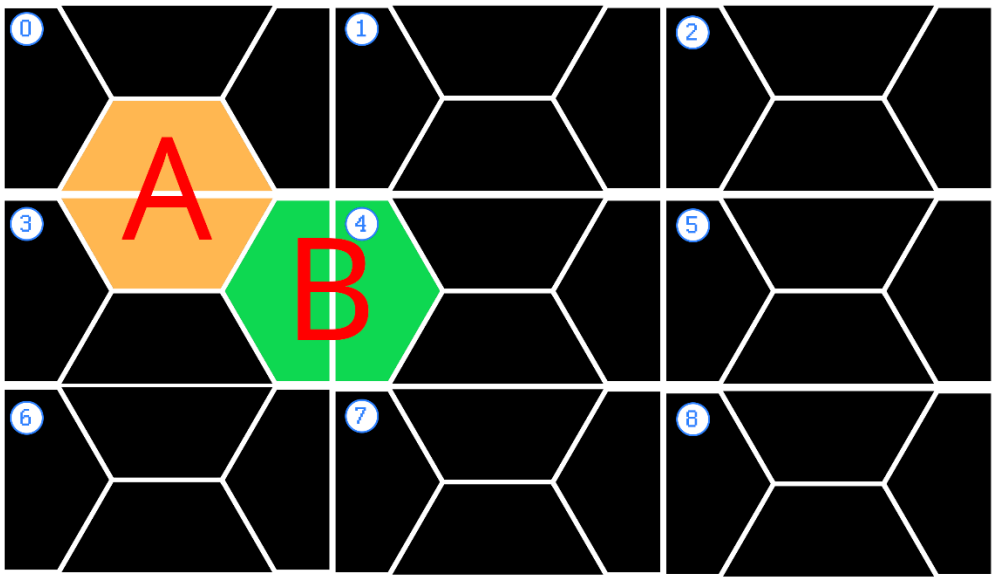


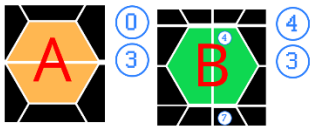
Latitudeangle
Atan2(sqrt(x^2 + z^2),y)



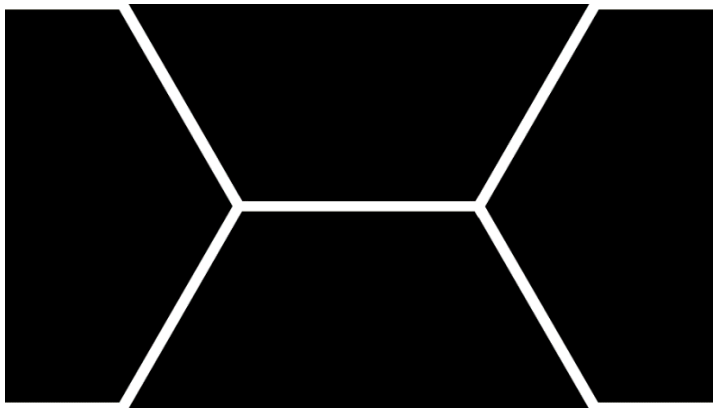
Longitudeangle
Atan2(x/z)







Etc..



Line width
Line Color

Callback Timer in Unity:
https://gist.github.com/Valryon/9915075

Tie into TangoARPose Controller in Update – Good Performance

```
using System;
using System.Collections;
using Tango;
using UnityEngine;

/// <summary>
/// This is a movement controller based on the poses returned from the Tango service, using the correct
timing needed
/// for augmented reality.
/// </summary>
[RequireComponent(typeof(TangoARScreen))]
public class TangoARPoseController : MonoBehaviour, ITangoLifecycle
{
    public UserMovement userMovement;

    /// <summary>
    /// If set, this controller will use the Device with respect Area Description frame pose.
    /// </summary>
    public bool m_useAreaDescriptionPose = false;

    /// <summary>
    /// Total number of poses ever applied by this controller.
    /// </summary>
    [HideInInspector]
    public int m_poseCount;

    /// <summary>
    /// The most recent pose status applied.
    /// </summary>
    [HideInInspector]
    public TangoEnums.TangoPoseStatusType m_poseStatus;

    /// <summary>
    /// The most recent pose timestamp applied.
    /// </summary>
    [HideInInspector]
    public double m_poseTimestamp;

    /// <summary>
    /// The most recent Tango rotation.
    /// </summary>
```

```
    [HideInInspector]
    public Vector3 m_tangoPosition;

    /// <summary>
    /// The most recent Tango position.
    /// </summary>
    [HideInInspector]
    public Quaternion m_tangoRotation;

    /// <summary>
    /// Matrix that transforms from the Unity Camera to Device.
    /// </summary>
    [HideInInspector]
    public Matrix4x4 m_dTuc;

    // We use couple of matrix transformation to convert the pose from Tango coordinate
    // frame to Unity coordinate frame.
    // The full equation is:
    //     Matrix4x4 uwTuc = uwTss * ssTd * dTuc;
    //
    // uwTuc: Unity camera with respect to Unity world, this is the desired matrix.
    // uwTss: Constant matrix converting start of service frame to Unity world frame.
    // ssTd: Device frame with repect to start of service frame, this matrix denotes the
    //       pose transform we get from pose callback.
    // dTuc: Constant matrix converting Unity world frame frame to device frame.
    //
    // Please see the coordinate system section online for more information:
    //     https://developers.google.com/project-tango/overview/coordinate-systems

    /// <summary>
    /// Matrix that transforms from Start of Service to the Unity World.
    /// </summary>
    [HideInInspector]
    public Matrix4x4 m_uwTss;

    /// <summary>
    /// The TangoARScreen that is being updated.
    /// </summary>
    private TangoARScreen m_tangoARScreen;

    /// @cond
    /// <summary>
    /// Awake is called when the script instance is being loaded.
    /// </summary>
    public void Awake()
    {
        // Constant matrix converting start of service frame to Unity world frame.
        m_uwTss = new Matrix4x4();
        m_uwTss.SetColumn(0, new Vector4(1.0f, 0.0f, 0.0f, 0.0f));
        m_uwTss.SetColumn(1, new Vector4(0.0f, 0.0f, 1.0f, 0.0f));
        m_uwTss.SetColumn(2, new Vector4(0.0f, 1.0f, 0.0f, 0.0f));
        m_uwTss.SetColumn(3, new Vector4(0.0f, 0.0f, 0.0f, 1.0f));

        m_poseTimestamp = -1.0f;
        m_poseCount = -1;
        m_poseStatus = TangoEnums.TangoPoseStatusType.NA;
        m_tangoRotation = Quaternion.identity;
```

```
        m_tangoPosition = Vector3.zero;
    }

    /// <summary>
    /// Start is called on the frame when a script is enabled.
    /// </summary>
    public void Start()
    {
        m_tangoARScreen = GetComponent<TangoARScreen>();

        TangoApplication tangoApplication = FindObjectOfType<TangoApplication>();
        if (tangoApplication != null)
        {
            tangoApplication.Register(this);

            // If already connected to a service, then do initialization now.
            if (tangoApplication.IsServiceConnected)
            {
                OnTangoServiceConnected();
            }
        }
        else
        {
            Debug.Log("No Tango Manager found in scene.");
        }
    }

    /// <summary>
    /// Update is called every frame.
    /// </summary>
    public void Update()
    {
        if (m_tangoARScreen.m_screenUpdateTime != m_poseTimestamp)
        {
            _UpdateTransformation(m_tangoARScreen.m_screenUpdateTime);
            userMovement.OnPoseUpdated(transform);    <--------------Add Here
        }
    }
```

Procedural ToolKit

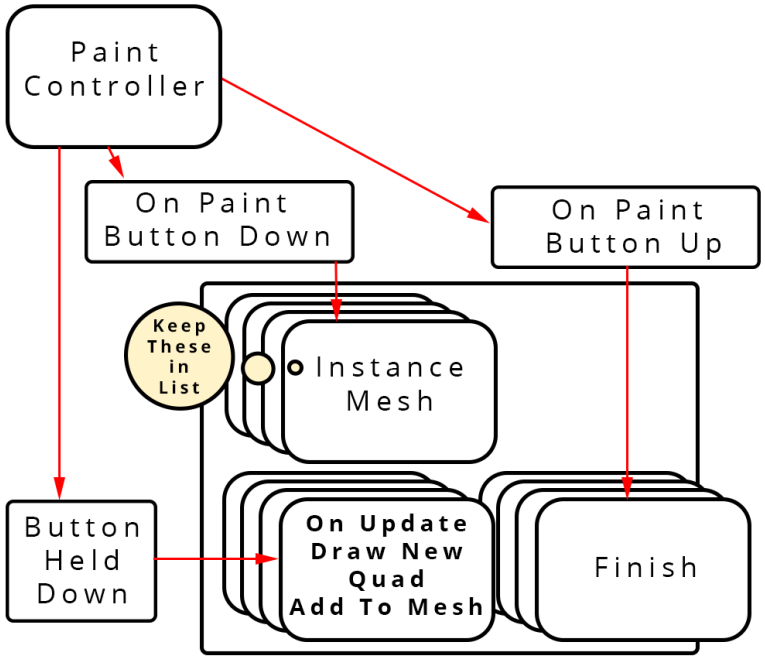
```
using System.Collections.Generic;
using UnityEngine;

namespace ProceduralToolkit
{
    /// <summary>
    /// Helper class for procedural mesh generation
    /// </summary>
    public partial class MeshDraft
    {
        public string name = "";
        public List<Vector3> vertices = new List<Vector3>();
        public List<int> triangles = new List<int>();
        public List<Vector3> normals = new List<Vector3>();
```

```
    public List<Vector2> uv = new List<Vector2>();
    public List<Color> colors = new List<Color>();

    public MeshDraft()
    {
    }

    public MeshDraft(Mesh mesh)
    {
        name = mesh.name;
        vertices.AddRange(mesh.vertices);
        triangles.AddRange(mesh.triangles);
        normals.AddRange(mesh.normals);
        uv.AddRange(mesh.uv);
        colors.AddRange(mesh.colors);
    }

    /// <summary>
    /// Adds mesh information from another draft
    /// </summary>
    public void Add(MeshDraft draft)
    {
        foreach (var triangle in draft.triangles)
        {
            triangles.Add(triangle + vertices.Count);
        }
        vertices.AddRange(draft.vertices);
        normals.AddRange(draft.normals);
        uv.AddRange(draft.uv);
        colors.AddRange(draft.colors);
    }

    /// <summary>
    /// Moves draft vertices by <paramref name="vector"/>
    /// </summary>
    public void Move(Vector3 vector)
    {
        for (int i = 0; i < vertices.Count; i++)
        {
            vertices[i] += vector;
        }
    }

    /// <summary>
    /// Rotates draft vertices by <paramref name="rotation"/>
    /// </summary>
    public void Rotate(Quaternion rotation)
    {
        for (int i = 0; i < vertices.Count; i++)
        {
            vertices[i] = rotation*vertices[i];
            normals[i] = rotation*normals[i];
        }
    }

    /// <summary>
    /// Scales draft vertices uniformly by <paramref name="scale"/>
```

```
        /// </summary>
        public void Scale(float scale)
        {
            for (int i = 0; i < vertices.Count; i++)
            {
                vertices[i] *= scale;
            }
        }

        /// <summary>
        /// Scales draft vertices non-uniformly by <paramref name="scale"/>
        /// </summary>
        public void Scale(Vector3 scale)
        {
            for (int i = 0; i < vertices.Count; i++)
            {
                var v = vertices[i];
                vertices[i] = new Vector3(v.x*scale.x, v.y*scale.y, v.z*scale.z);
                var n = normals[i];
                normals[i] = new Vector3(n.x*scale.x, n.y*scale.y, n.z*scale.z).normalized;
            }
        }

        /// <summary>
        /// Paints draft vertices with <paramref name="color"/>
        /// </summary>
        public void Paint(Color color)
        {
            colors.Clear();
            for (int i = 0; i < vertices.Count; i++)
            {
                colors.Add(color);
            }
        }

        /// <summary>
        /// Flips draft faces
        /// </summary>
        public void FlipFaces()
        {
            FlipTriangles();
            FlipNormals();
        }

        /// <summary>
        /// Reverses winding order of draft triangles
        /// </summary>
        public void FlipTriangles()
        {
            for (int i = 0; i < triangles.Count; i += 3)
            {
                var temp = triangles[i];
                triangles[i] = triangles[i + 1];
                triangles[i + 1] = temp;
            }
        }
```

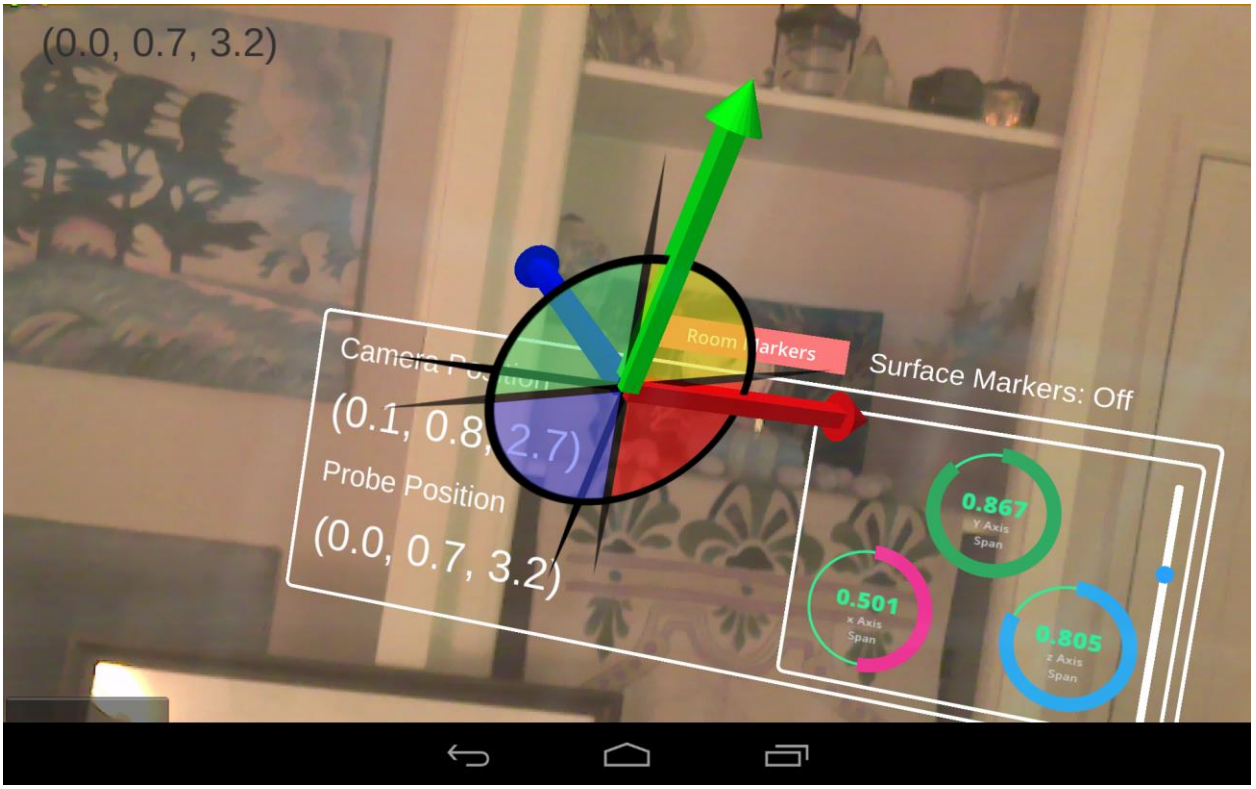
```
        /// <summary>
        /// Reverses direction of draft normals
        /// </summary>
        public void FlipNormals()
        {
            for (int i = 0; i < normals.Count; i++)
            {
                normals[i] = -normals[i];
            }
        }

        /// <summary>
        /// Creates new mesh from information in draft
        /// </summary>
        public Mesh ToMesh()
        {
            return new Mesh
            {
                name = name,
                vertices = vertices.ToArray(),
                triangles = triangles.ToArray(),
                normals = normals.ToArray(),
                uv = uv.ToArray(),
                colors = colors.ToArray()
            };
        }
    }
}
```

Paint Controller - Sphere Tango 3



Create a prefab that will be used to draw lines

```
MeshFilter mf = GetComponent<MeshFilter>();
if( mf.sharedMesh == null )
     mf.sharedMesh = new Mesh();

Mesh mesh = mf.sharedMesh;
Vector3[] vertices = new Vector3[]{ … };
Vector3[] normals = new Vector3[]{ … };
Vector2[] uvs = new Vector2[]{ … };

mesh.Clear();
mesh.vertices = vertices;
mesh.normals = normals;
mesh.uv = uvs;
mesh.triangles = triangleIndices;
```

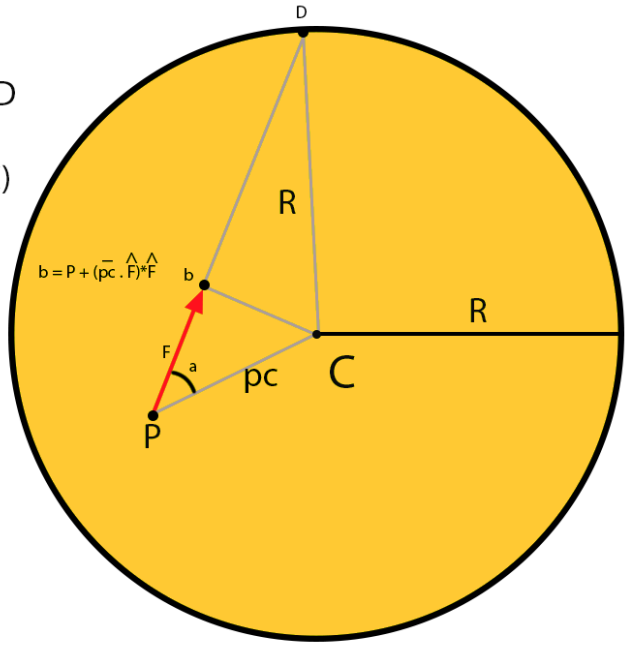


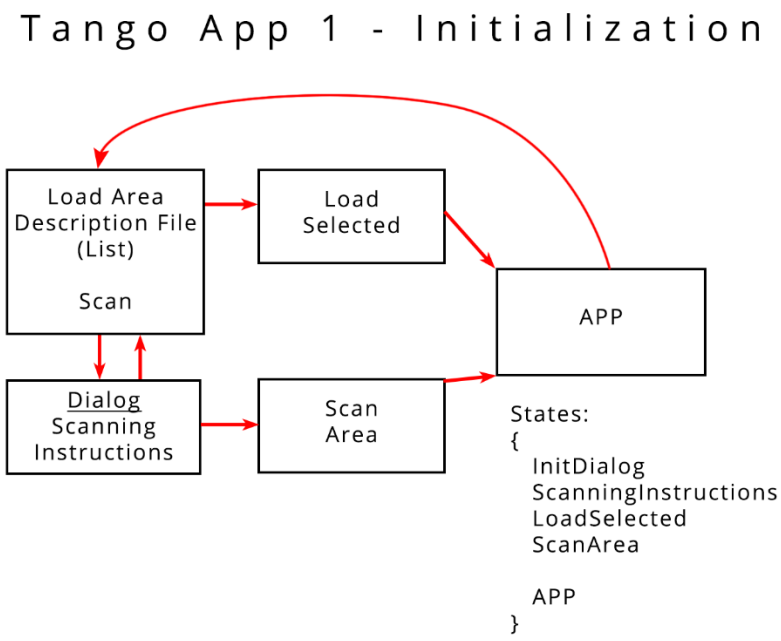
Ray Intersection with Sphere

R*R=bC*bC + bD*bD

bD=sqrt(R*R-bC*bC)

$D = b + \hat{F} * bD$





Project: 2/26/26

UI: Load Area Description

Place Objects In Specific Locations

UI: Create Area Description

UI: Save Area Description

UI: Save Objects

Reload Objects Placed

√ **Mesh Example**

√ **Calibrate Unit**

Project Tango Unity SDK Reference

These are the reference pages for the Project Tango Unity SDK.

Classes	
AndroidHelper	Misc Android related utilities provided by the Tango CoreSDK.
AreaDescription	C API wrapper for the Tango area description interface.
Metadata	Easy access to the metadata fields for an Area Description.
OrientationManager	Manages the orientation of the screen.
PoseListener	Marshals Tango pose data between the C callbacks in one thread and the main Unity thread.
PoseProvider	C API wrapper for the Tango pose interface.
TangoApplication	Main entry point for the Tango Service.
TangoCameraIntrinsics	The TangoCameraIntrinsics struct contains intrinsic parameters for a camera.
TangoEnums	Enumerations used by the Tango Service.
TangoEvent	The TangoEvent structure signals important sensor and tracking events.
TangoImageBuffer	The TangoImageBuffer contains information about a byte buffer holding image data.
TangoPoseData	The TangoPoseData struct contains 6DOF pose information.
TangoUnityDepth	Like TangoXYZij , but more Unity friendly.
TangoUnityImageData	The TangoUnityImageData contains information about a byte buffer holding image data.
TangoXYZij	The TangoXYZij struct contains information returned from the depth sensor.
VideoOverlayProvider	C API wrapper for the Tango video overlay interface.

YUVTexture	Wraps separate textures for Y, U, and V planes.

Interfaces

IExperimentalTangoVideoOverlay	Experimental API only, subject to change.
ITangoAreaDescriptionEvent	Tango Area Descrrption event interface.
ITangoDepth	Tango depth interface.
ITangoEvent	Event notification interface.
ITangoEventMultithreaded	Tango event interface where the handler will be invoked from multiple threads.
ITangoLifecycle	Tango lifecycle interface.
ITangoPose	Pose interface.
ITangoVideoOverlay	Tango video overlay interface.

Structs

Common	This struct holds common global functionality used by this SDK.
ErrorType	Codes returned by Tango API functions.
MetaDataKeyType	Metadata keys supported by Tango APIs.
TangoCoordinateFramePair	The TangoCoordinateFramePair struct contains a pair of coordinate frames of reference.
TangoDeviceOrientation	Holds the current and default orientation of the device.

Coordinate System Conventions

The Project Tango API returns pose data as a rotation and translation between two frames of reference with X, Y, and Z values. It's important to understand where the X, Y, and Z axes are with relation to the device and its environment. This page describes some of our common coordinate systems.

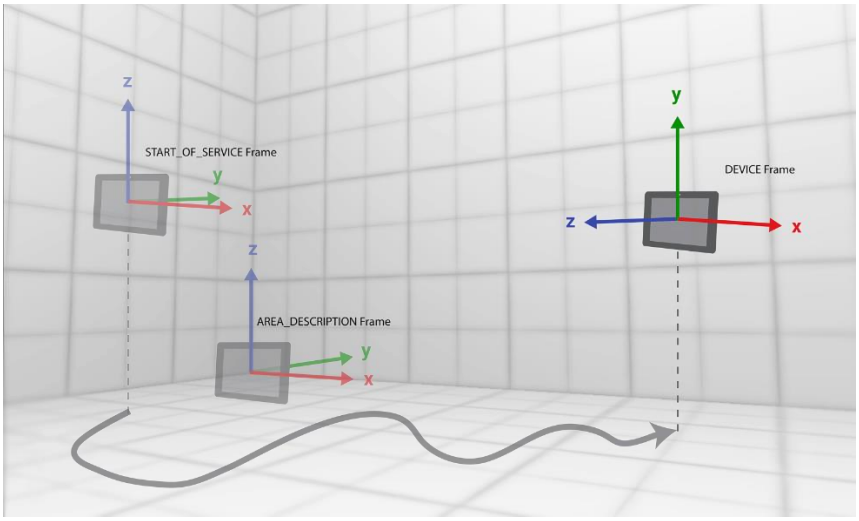
Coordinate Frames

In all the figures, arrows denote the positive sense of the axis. The circle with an "x" inside it denotes an arrow pointing into the page away from you; The circle with a dot inside it denotes an arrow pointing out of the page toward you. The coordinate systems are shown with the device held in its default orientation, screen facing the user, back cameras facing away from you into the page.

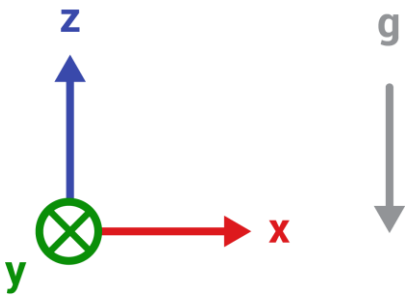
In this document, we use the following notation to represent a transformation from a Base coordinate frame to a Target coordinate frame.

$$\text{Base}\ T\ \text{Target}$$

Project Tango Coordinate Frames

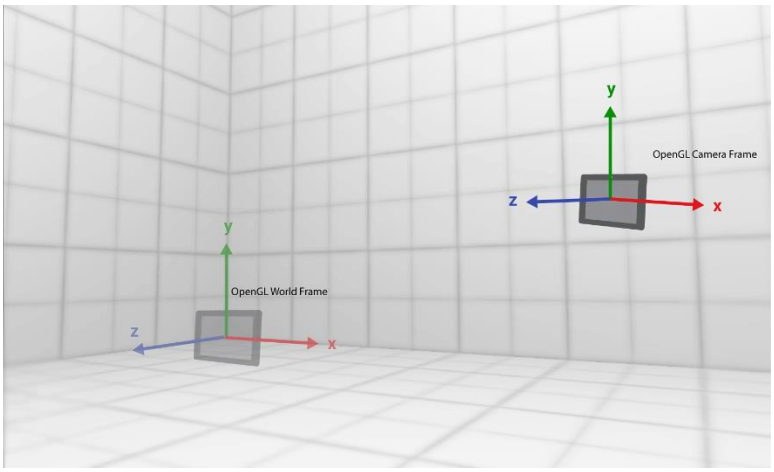


Project Tango uses a right-handed, local-level frame for the `START_OF_SERVICE` and `AREA_DESCRIPTION` coordinate frames. This convention sets the Z-axis aligned with gravity, with Z+ pointed upwards, and the X-Y plane is perpendicular to gravity and locally level with the ground plane. This local-level convention is based on the local east-north-up (ENU) earth-based coordinate system. Instead of true north, Project Tango uses the direction the back of the device is pointed when the service started as the Y axis, and the X axis is pointed to the right.
The `START_OF_SERVICE`and `AREA_DESCRIPTION` base coordinate frames of the API will use this local-level frame convention.



Project Tango's `DEVICE` coordinate frame is the same as the Android Sensor Coordinate System. When a device is held in its default orientation, the X axis is horizontal and points to the right, the Y axis is vertical and points up, and the Z axis points toward the outside of the screen face. In this system, the back of the device faces the negative Z axis.

OpenGL Coordinate Frames

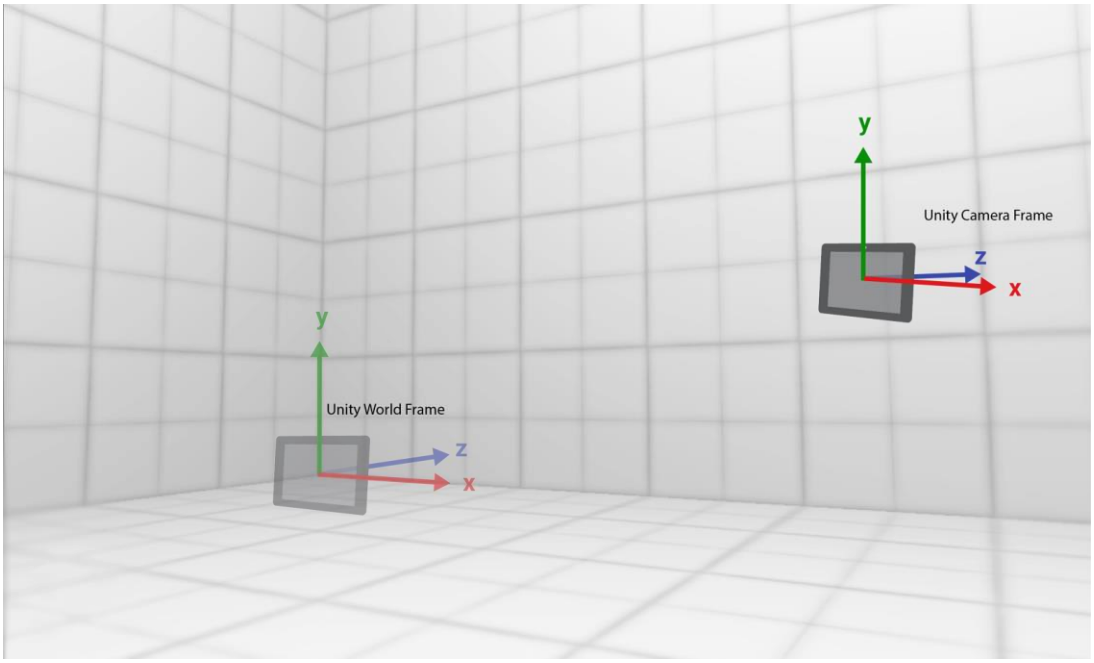


The OpenGL World Frame uses a right-handed coordinate system, with the Y axis aligned with gravity and pointed upwards, and the Z axis pointed towards the user.

The OpenGL Camera Frame also uses a right-handed coordinate system. The X axis is horizontal and points to the right, the Y axis is vertical and points up, and the Z axis points towards the camera. In this system, the camera looks down the negative Z axis.

Note: If an Android device in its default orientation is being used to control a camera, the device and OpenGL camera are using the same coordinate system conventions.

Unity Coordinate Frames

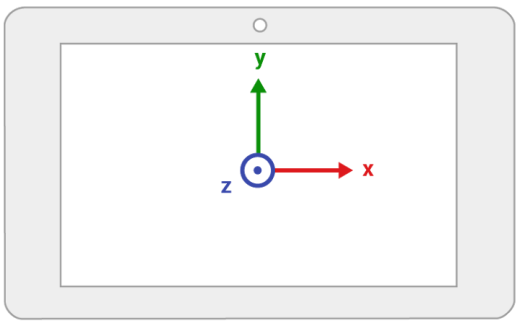


The Unity World Frame uses a left-handed coordinate system, with the Y axis aligned with gravity and pointed upwards, and the Z axis pointed away from the user.

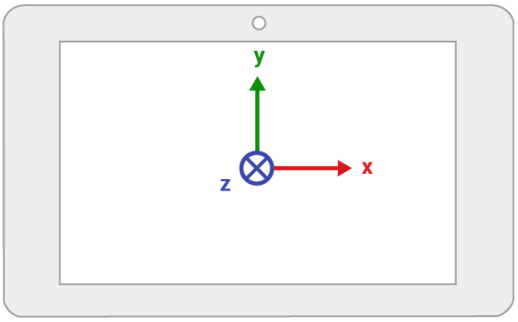
The Unity Camera Frame also uses a left-handed coordinate system. The X axis is horizontal and points to the right, the Y axis is vertical and points up, and the Z axis points towards the camera's view. In this system, the camera looks down the positive Z axis.

Coordinate Frame Conversions

In this section, we will describe how to convert the pose data returned by the Project Tango APIs into common alternate frames, such as the OpenGL and the Unity camera coordinate frames as illustrated below:



OpenGL Camera Frame



Unity3D Camera Frame

Both OpenGL and Unity use a world coordinate frame which defines the origin of the scene, and a camera coordinate frame which is attached to the viewport camera.

For these examples, we will look at the pose of the device using `DEVICE` (D) as the target coordinate frame and`START_OF_SERVICE` (SS) as the base coordinate frame, represented by this notation:

$$ {}^{SS}_{D}T $$

We will also convert the device's position to match the target platform's camera coordinate frame, as for an application that matches the camera's movement to the device movement.

Converting Project Tango pose data into the OpenGL Coordinate System

We use two transformations to convert Project Tango pose data into the OpenGL coordinate system (with the Y axis pointed up and aligned with gravity). The first transformation converts the from OpenGL World Frame (OW) to the Project Tango START_OF_SERVICE (SS) coordinate frame:

$$ {}^{OW}_{SS}T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

The second transformation converts from the Project Tango DEVICE(D) coordinate frame to the OpenGL Camera (OC) coordinate frame. Because the from the OpenGL Camera (OC) Frame and the Project Tango DEVICE (D) coordinate frame are the same, this is the identity matrix.

$$ {}^{D}_{OC}T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

By multiplying the transformations together in this order, we are able to obtain a single transformation matrix representing the pose returned by the API converted to the OpenGL Camera (OC) target coordinate frame with respect to the OpenGL World (OW) base coordinate frame.

You just need to left multiply the Tango Pose with ${}^{OW}_{SS}T$ and right multiply with ${}^{D}_{OC}T$ as shown here:

$$ {}^{OW}_{OC}T = {}^{OW}_{SS}T * {}^{SS}_{D}T * {}^{D}_{OC}T $$

This is provided in the API Sample Code using quaternions. Note that ${}^{D}_{OC}T$ is an identity transformation matrix and can be omitted, but is shown here for completeness.

Converting Project Tango Pose data into the Unity Coordinate System

We can handle Unity Coordinate System conversion the same way as OpenGL—the first transformation converts from the Unity World Frame(UW) to the Project Tango START_OF_SERVICE(SS):

$$ {}^{UW}_{SS}T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

The second transformation converts from the Project Tango DEVICE(D) coordinate frame to the Unity Camera (UC) coordinate frame. You'll notice that unlike in OpenGL, this is not the identity matrix because Unity uses a left-handed coordinate system.

$$ {}^{D}_{UC}T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

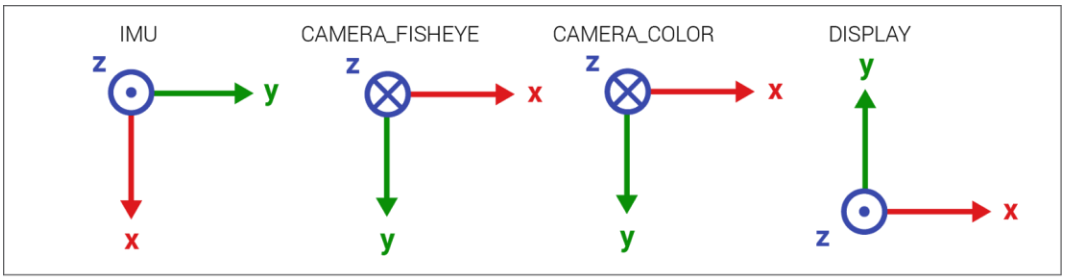
Left multiply the first conversion and right multiply the second conversion to get the pose data converted to the Unity Camera (UC) coordinate frame with respect to the Unity World (UW) coordinate frame.

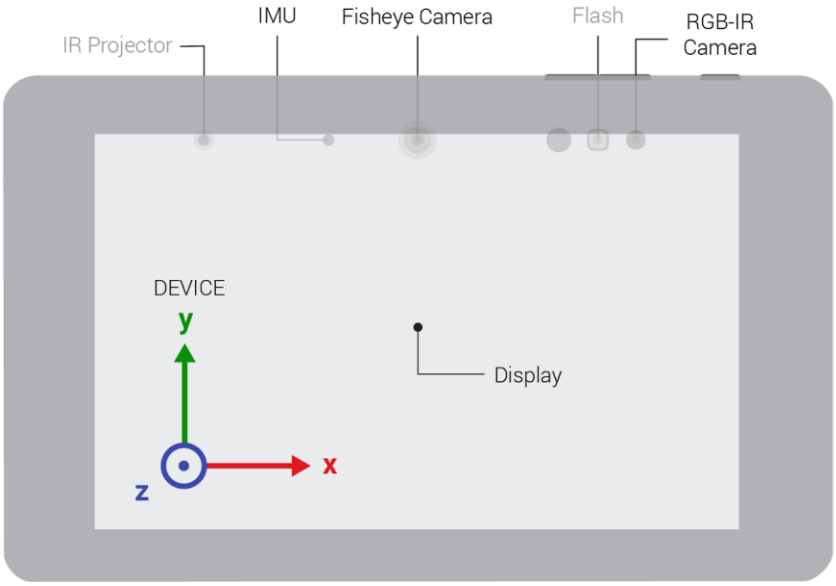
$$ {}^{UW}_{UC}T = {}^{UW}_{SS}T * {}^{SS}_{D}T * {}^{D}_{UC}T $$

On-Device Coordinate Frames

Each component on a Project Tango device, like the IMU, the cameras, and the display, has its own coordinate frame. The device itself also has a coordinate frame which aligns with the default Android Device coordinate frame. You can get the position and orientation offset between each component (extrinsics) using the Project Tango APIs.

The coordinate frames for the Project Tango development kit are illustrated here. You should not hardcode these values, since other devices may be different.





VIO - Visual-Inertial Odometry

Project Tango implements motion tracking using **visual-inertial odometry**, or VIO, to estimate where a device is relative to where it started. Unlike GPS, motion tracking using VIO works indoors and can provide higher accuracy.

Standard visual odometry uses camera images to determine a change in position by looking at the relative position of different features in those images. For example, if you took a photo of a building from far away and then took another photo from closer up, it would be possible to calculate the distance the camera moved based on the change in size and position of the building in the photos.

Visual-inertial odometry supplements visual odometry with inertial motion sensors capable of tracking a device's rotation and acceleration. This allows a Project Tango device to estimate both its orientation and movement within a 3D space with even greater accuracy.

Motion tracking is great by itself if you need to know a device's orientation and relative position, but there are some limitations:

- Over long distances and periods of time the accumulation of small errors can cause measurements to "drift," leading to larger errors in absolute position.
- Motion tracking does not understand the actual area around it. Every time you start a new motion tracking session, the tracking starts over and reports its position relative to its most recent starting position.

Both of these limitations are addressed by the next core technology, area learning.

Area Learning and SLAM

Human beings learn to recognize their environment by noticing the features around them: a doorway, a staircase, the way to the nearest restroom. Walk around a building with a Project Tango device and it can do something similar.

Project Tango's technology notes the key visual features of a physical space—the edges, corners, other unique features—so it can recognize that area again later. Areas that are more visually interesting, with furniture and doors and patterns on the floor or walls, are easier to recognize than empty rooms with white walls. This process of learning an area while keeping track of a user's current position within it is known as **Simultaneous Localization and Mapping, or SLAM**.

A Project Tango device can use what it learned about an area in the past to help it understand what it is seeing now. To do this it stores a mathematical description of the visual features it has identified inside a searchable index on the device. This allows the device to quickly match what it currently sees against what it has seen before without any cloud services.

When a Tango device has learned an area, there are some key things it can do to improve upon the information provided by motion tracking alone:

- Perform **localization** to orient and position itself within a previously learned area
- Perform **drift correction** to fix errors that occur during prolonged motion tracking by recognizing its position within a known physical area

For information on how to write applications that can learn an area, or that use previously-learned area descriptions, see the developer overview on Area Learning.

Area Learning

Project Tango allows devices to use visual cues to navigate and understand the world around them. This page describes Project Tango's area learning functionality for both improved motion tracking and localization.

What is area learning?

Using area learning, a Project Tango device can remember the visual features of the area it is moving through and recognize when it sees those features again. These features can be saved in an Area Description File (ADF) to use again later. With an ADF loaded, Project Tango devices gain two new features: improved motion tracking and localization.

Improving motion tracking

While moving through an area, a Project Tango device estimates its location and the path it has travelled. As mentioned in the concepts overview, motion estimates become less accurate over time. Although the device corrects for some errors by orienting itself to gravity, errors in other aspects of its pose cannot be detected in motion tracking alone.

Using area learning, a Project Tango device can remember the visual features of the area it has visited and use them to correct errors in its understanding of its position, orientation, and movement. This differs from motion tracking alone, which has no memory of the environment. This memory allows the system to perform **drift corrections**, also called **loop closures**. When the device comes back to a place it has already visited, it realizes it has travelled in a loop and adjusts its path to be consistent with its previous observations. These corrections can be used to adjust the device's position and trajectory within your application.

Figure 1 shows an example of drift correction. The green line is the real trajectory that the device is travelling, and the red line is the trajectory that the device estimates. After a little while the device starts drifting, but when it returns to the origin point it recognizes that point and corrects the trajectory.

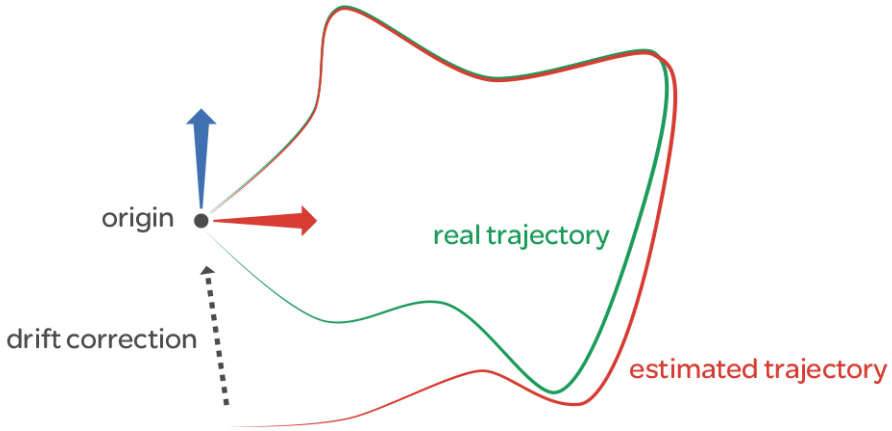


Figure 1: Correcting the Estimated Trajectory

Without drift correction, a game or application using a virtual 3D space aligned with the real world may encounter inaccuracies in motion tracking after extended use. For example, if a door in a game world corresponds with a door frame in the real world, after accumulating drift, the game door may appear in the middle of the real world wall instead of in the door frame.

Localization using area descriptions

In addition to drift correction, a device with a loaded area description knows where it is within the learned area relative to where the original learning started. This process is called **localization**. Without localizing to an area description, a device's starting point is lost every time you end the session. This ability to learn an area and localize against that area is required if you want to create a consistent experience within the same mapped space, such as having virtual objects appear in the same location as the last time the user visited an area.

Figure 2 shows what happens when a device starts from a new origin and moves back into an area it has learned before.

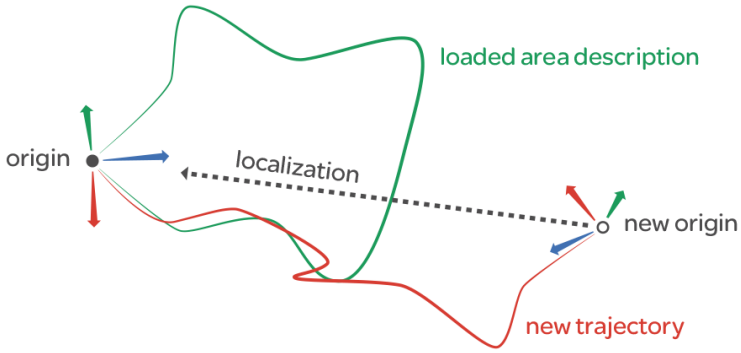


Figure 2: Localizing to a Saved Area Description

Project Tango devices depend on the visual diversity of the area to localize. If you are in an area with many identical rooms or in a completely empty room with blank walls, it will be difficult to localize.

Creating and saving an Area Description

There are two ways to create an Area Description File. The simplest is to use an application like Project Tango Explorerto create the ADF, and have your application load it. The second is to use the Project Tango APIs to handle the learning, saving, and loading all within your application. Depending on your settings, you can learn a new area or append to an existing ADF.

A room may look quite different when standing in different positions, if furniture is moved around, or during the day than it does at night. If current conditions are similar to when you create the area description, localization is more likely to succeed. If localization is not occuring, the area descriptions may be too different from the current conditions.

Because there are many reasons why an area may change in appearance, you might create multiple area description files for a single physical location under different conditions and select the correct one that will be most similar to the conditions the user will have during their session. Alternatively, you could append multiple area learning sessions onto the same ADF to capture visual descriptions of the environment from every position and angle, under every variation of lighting or environmental change within one file.

Our UX Best Practices has additional tips on creating ADFs and using area learning.

Possible use cases

As mentioned, learning and loading area descriptions has many advantages. You can now intentionally align the device's coordinate frame with a pre-existing coordinate frame so that content in a game or app always appears in the same physical location.

Another advantage is for multi-player experiences, where two users can localize to the same coordinate frame, shared through a cloud service. This allows multiple people to interact in the same physical space where all of their relative positions are known. Although the Project Tango APIs do not natively support data sharing in the cloud, it is possible to use Google Cloud Storage and the Google Play Games API to implement this.

Some use cases may include a space owner making available an area description of their environment, such as a retail store or public space. These area descriptions may have been created by an expert user, such as a store manager. This would allow end users to localize precisely within that physical space, for example with location-aware shopping experience.

Important: Saved area descriptions do not directly record images or video of the location, but rather contain descriptions of images of the environment in a very compressed form. While those descriptions can’t be directly viewed as images, it is in principle possible to write an algorithm that can reconstruct a viewable image. Therefore, you must ask the user for permission before saving any of their learned areas to the cloud or sharing areas between users to protect the user's privacy, just as you would treat images and video.

Area learning and using area descriptions are powerful features and we’re excited to see what developers come up with for offering new user experiences.

Using Learning Mode and loaded Area Description Files

Depending on your settings for learning mode or whether you loaded an ADF, the behavior of some aspects of the Project Tango APIs will vary.

In this table, the left two columns denote whether you have learning mode on, and whether you have loaded a previously-stored area description. Depending on the status of those two things, you may or may not be able to save an area description. For example, if you don't have learning mode on, you cannot save an area description file. If you have learning mode on, and have loaded an area description file, you can only save again once you have localized against the loaded ADF.

Also, if you aren't in learning mode, and don't have an ADF loaded, you cannot get pose data using the`TANGO_COORDINATE_FRAME_AREA_DESCRIPTION` frame of reference. If you have an ADF loaded, you can get pose data from that frame of reference once your device localizes to the loaded ADF.

Is learning mode on?	Is there an ADF loaded?	Is pose data available for this frame of reference pair?			Can you save an area description file?
		Start of service to device	Area description to device	Area description to start of service	
False	False	Available at start	Not available	Not available	Cannot save area description.
True	False	Available at start	Available at start*	Available at start*	Current area description saved with new UUID.
False	True	Available at start	Available after localized	Available after localized	Cannot save area description.
True	True	Available at start	Available after localized	Available after localized	You cannot save the area description until after you have localized against the loaded ADF. When you save, it will create a new file with a new UUID.

*If tracking is lost, these frame of reference pairs will no longer be available. After service reset, the session functions as if learning mode is `True` and an ADF was loaded, where the area descriptions are those that were learned up to the loss of tracking. To continue using the area description frame of reference, you must localize against what you learned before the loss of tracking. You must also localize to include what you learned before tracking was lost when saving an ADF.

Release Notes

February 3, 2016: Gemma (Project Tango Core 1.31)

The Gemma release consists of software updates only, there is no associated OTA update.

Unity SDK

- The names of the Area Description options in Tango Manager have changed. Instead of **Load ADF** and **Area Learning**, there is the option **Enable Area Descriptions** with a **Learning Mode** option under it. Existing applications will continue to work, but if you are using area descriptions and update the SDK, you will need to update your project to adjust.
- The UI for handling video overlays has been changed. If **Enable Video Overlay** is selected, you'll see a drop-down menu to select the video overlay method. Selecting **TextureID** enables the`OnExperimentalTangoImageAvailable` callback of the `IExperimentalTangoVideoOverlay` interface. This provides the same functionality as the **Experimental Video Overlay** did before. Selecting **Raw Bytes** enables the `OnTangoImageAvailableEventHandler` callback of the `ITangoVideoOverlay` interface. This is the same behavior as having only **Enable Video Overlay** enabled before. You also have the option to enable both methods. If you are using the AR Camera prefab, the **TextureID** method or **Both** must be selected.

Support Library

- `upsampleImageNearestNeighbor` has been added to the Java support library.
- `getPoseInEngineFrame` has been added to the transformation helper in C and Java.
- Fixed the naming of the function TangoSupport_upsampleImageNearestNeighbor.

Unity Tutorial : Motion Tracking with Unity

These instructions will get you up and running with a simple application in Unity, intended for Unity developers of any experience level. You'll build a simple scene in Unity, import the Project Tango functionality, and attach a prefab camera that lets you use the Project Tango device's motion in the real world to move the camera around in the Unity scene.

For more experienced Unity developers, you may also want to review the code samples in GitHub and the SDK Reference Documentation.

If you haven't set up your computer to develop with Project Tango in Unity yet, see Getting Started With Unity.

Create a new project in Unity

1. Go to **File > New Project**, which opens up the New project window.
2. Enter a name for your new project in the **Project Name** text box.
3. Make sure that the project is set to 3D.
4. Click **Create Project**.

Import the Tango SDK

If you haven't already downloaded the Tango SDK .unitypackage file in the Getting Started section, get it from ourDownloads page.

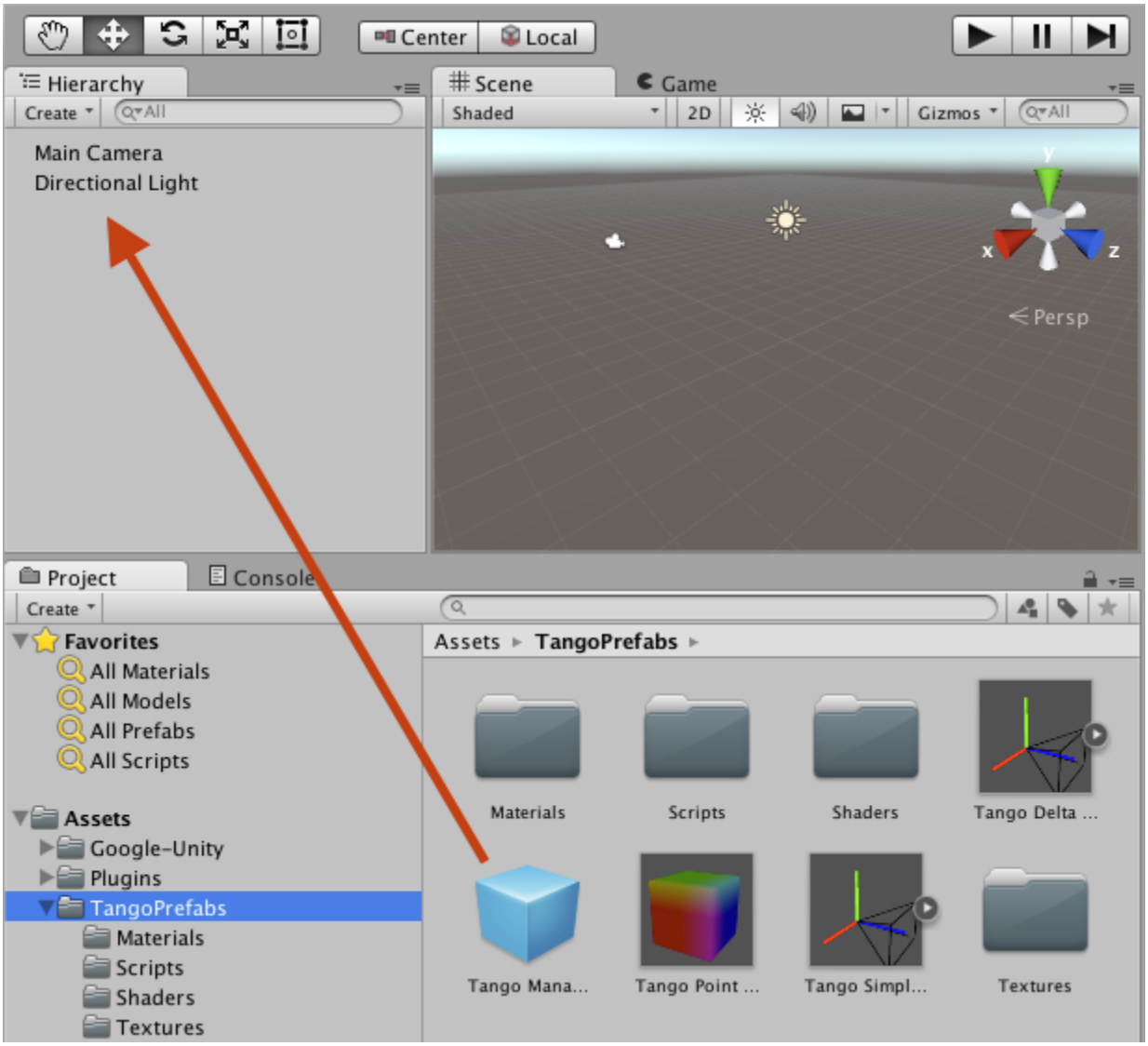
1. Go to **Assets > Import Package > Custom Package**.
2. Find the Tango SDK .unitypackage file that you downloaded and select it (either by double-clicking or by highlighting it and clicking Open).
3. Make sure all the boxes are checked (they should be by default) and click **Import** at the bottom right.

The Project Tango assets should now be displayed in the **Assets** file dropdown in your Project panel at the lower left.

Add the Tango Manager prefab to your Hierarchy

If you're not familiar with Prefabs, see the Unity documentation on them.

1. Open the **Assets** folder in the Project panel on the lower left.
2. Click on **Assets > TangoPrefabs**.
3. Drag the Tango Manager prefab (box icon) into the Hierarchy panel.



Add the Tango Delta Camera prefab to your Hierarchy

This is similar to the last step.

1. Find **Assets > TangoPrefabs** if you don't still have it open from the last step. This is the same location you found the Tango Manager prefab.
2. Drag the Tango Delta Camera prefab (frustum icon) into the Hierarchy panel.

Create a simple Unity scene

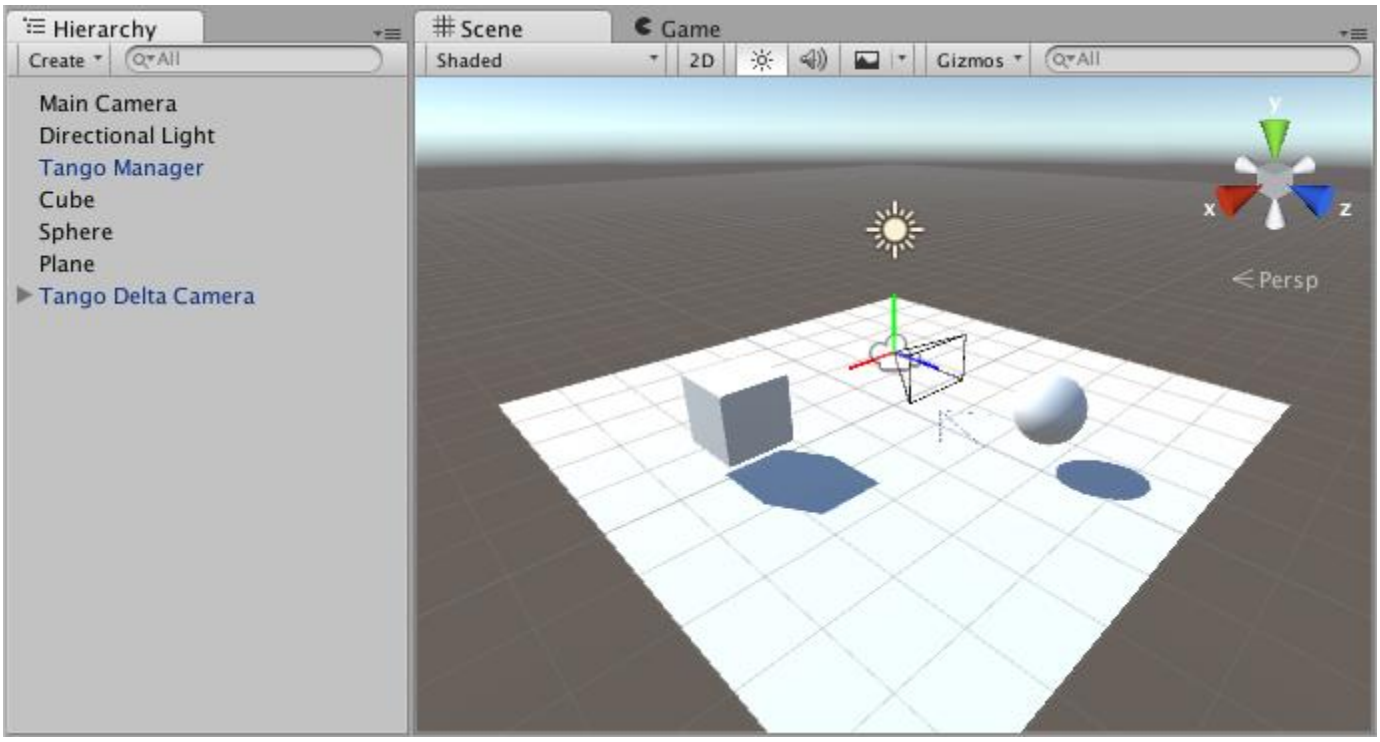
For the purposes of this tutorial, we recommend creating a plane and a few shape objects above or below it.

You can create 3D objects by going to **GameObject > 3D Object** and selecting your objects from that menu, such as Plane, Sphere, or Cube. To place them in the scene, either drag them to the desired location, or use the Transform section of the Inspector on the right to place them by typing values in for X, Y, and Z.

For example, in the following scene, we have the following objects and positions:

- Cube (3,1,0)
- Directional light (0,0,0), with intensity 1
- Main Camera (0,1,0)
- Plane (0,0,0)
- Sphere (0,1,3)
- Tango Delta Camera (0,1,0)

If you use the same objects we did, your scene will look like this:



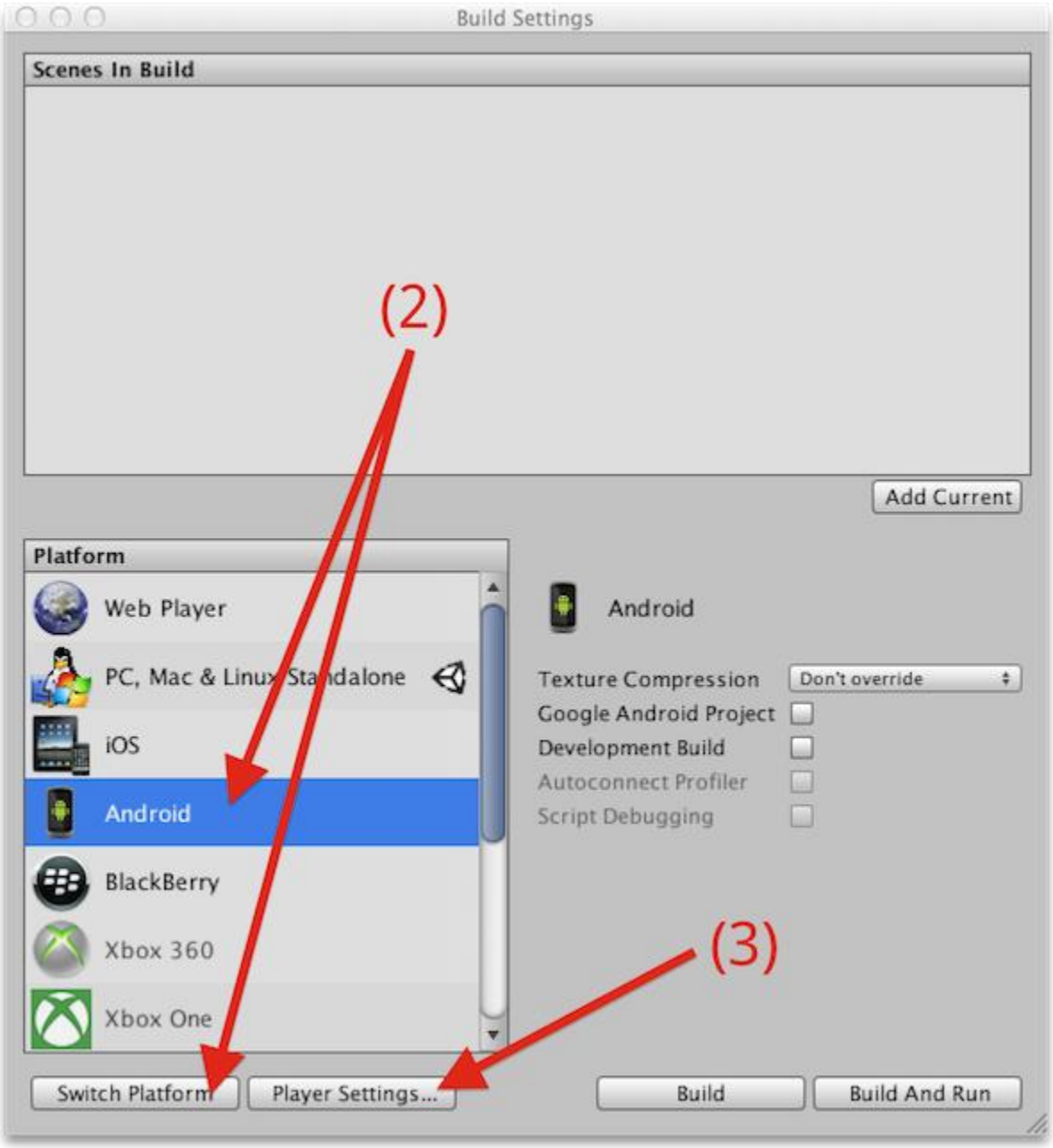
The Game view may seem a little strange, but that's basically the camera's viewpoint: it sees the sphere hovering slightly above the plane. When you view this with your Project Tango device, you'll be able to move the camera around and see the cube as well.

Change the Build Settings

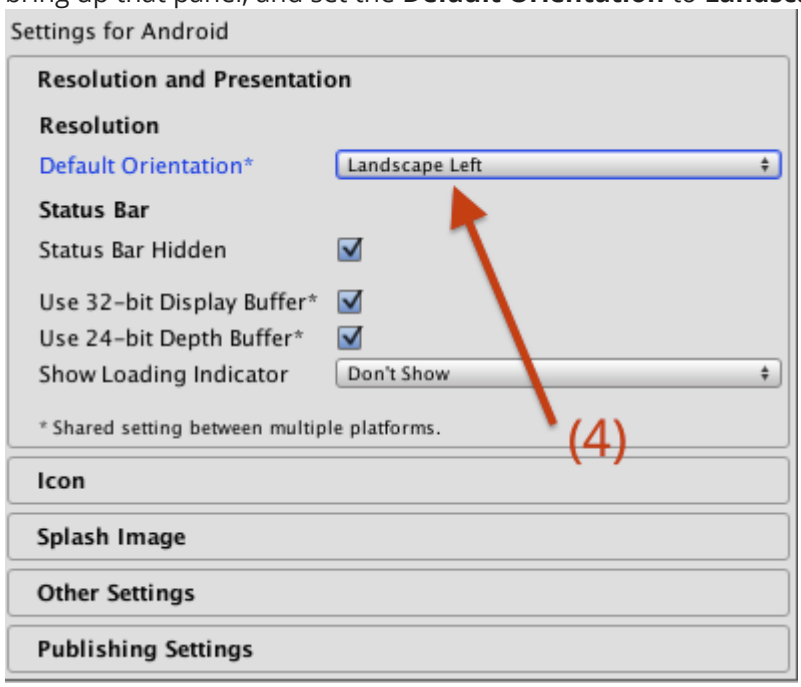
There are a few things you'll need to set up before the first time you build, but after that you'll be able to just click **Build and Run**.

1. Open up the Build Settings by going to **File > Build Settings**.
2. Select **Android** under **Platform**. After selecting Android, click the **Switch Platform** below it.

3. Go into Player Settings by clicking the button in the bottom row (see screenshot).

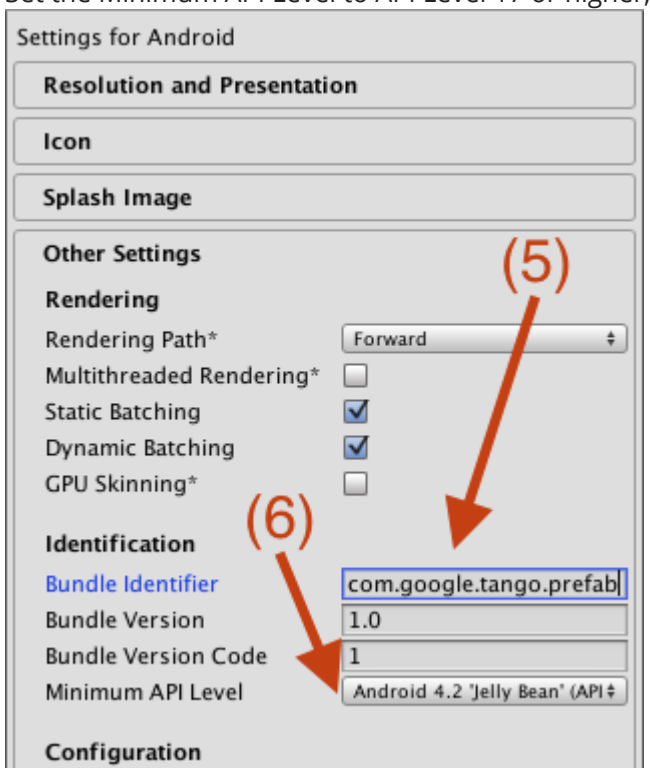


4. In the PlayerSettings Inspector, at the bottom under Settings for Android, click **Resolution and Presentation** to bring up that panel, and set the **Default Orientation** to **Landscape Left**.



5. Click open the **Other Settings** panel, and enter your desired package name into **Bundle Identifier**. In our example we use com.google.atap; for more information on Android application package names, see App Manifests.

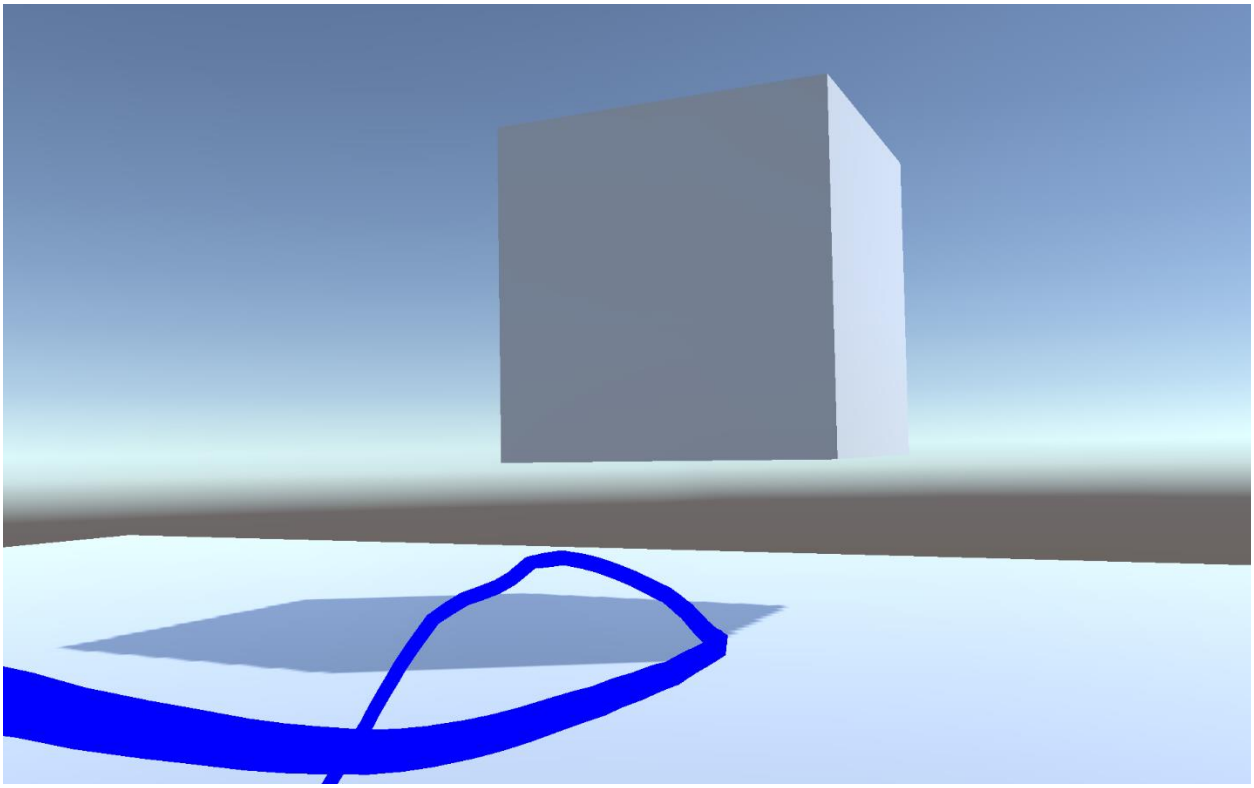
6. Set the Minimum API Level to API Level 17 or higher, if you haven't already.



Make sure that your Project Tango device is plugged into your computer, then click **Build and Run** in the Build Settings window or from **File > Build&Run**.

After the build bar clears, Unity launches your application on the Project Tango device.

You should see the sphere when it first launches. Tilt the device back and forth, move it around a bit, and go find the cube!



Disable the Trail Renderer (optional)

As you move around the device, you might notice a blue line being drawn following the path the device takes in 3D space. The Trail game object and its Trail Renderer, which is a subcomponent of the Tango Delta Camera prefab, draws this line. If you're planning to use the prefab to build new experiences in Unity and don't want to show the line, here's how to disable it:

1. In the **Hierarchy**, expand the **Tango Delta Camera** prefab's foldout arrow to show its children, including **Trail**, and select **Trail** by clicking on it.
2. In the **Inspector**, uncheck the box next to the overall **Trail** game object.
3. Build and run.

Using Area Learning in Unity

In this tutorial we will explain how to use area learning features in Unity. This tutorial assumes you're already familiar with using the Delta Camera prefab to integrate motion tracking into a Unity project. See our tutorial on using motion tracking if you haven't done this before. This tutorial also assumes you're familiar with area learning concepts.

Prerequisites

- Existing project integrated with motion tracking using a recent version of the Project Tango SDK (Ancha or newer). If you don't have one, you can complete the motion tracking tutorial.
- Familiarity with Unity.

Tango Manager settings related to area learning

Within the Tango Manager prefab, there are a few settings relevant to area learning. We'll go over them here, but you don't need to do anything yet.

In the Tango Application script, the **Auto-connect to Service** option controls the startup flow of Project Tango. It should not be checked if using area learning features. Area learning requires some application-specific settings during start-up and prevents us from using the Project Tango Unity SDK's built-in auto-connect option. Instead, we will create code to handle start-up and initialization using the Project Tango APIs.

Load ADF requests the area learning permission, which is required to load an area description or get the list of area descriptions on a device.

Area Learning requests the area learning permission and enables learning mode. With learning mode enabled, the area learning system will create new area descriptions based on what it sees. You can also save the area description, as described below, to create or extend an area description file.

Using Load ADF mode

In this section, we'll show you how to use previously created area descriptions in your application. Specifically, you'll be able to:

- Get a list of area descriptions
- Load a specific area description
- Set the Tango Delta Camera to use the area description
- Show the user instructions for relocalization

List and load area descriptions

To load an area description file, you must specify the area description using`TangoApplication.Startup(AreaDescription)` when connecting to the Tango Service. A common way to do this is by showing a list of all area descriptions on the device for the user to choose from. However, we can only query the list of area descriptions after we have requested the area learning permission. Therefore, the code must manually control the TangoApplication startup process by explicitly calling `TangoApplication.RequestPermissions()` and `TangoApplication.Startup()`.

First, make sure that **Auto-connect to Service** is unchecked and **Load ADF** is checked in the **Tango Manager** prefab. This will allow us to implement our own connection code, and add the area learning permission to our permissions request.

Next, we will add a new script to request permissions, get the list of area descriptions, and connect to a specified area description. For simplicity, we will use the most recent Area Description.

First, create a new, empty GameObject. Next, create a new C# script, name it "AreaLearningStartup", and drag it onto the GameObject you created. Finally, open the script and copy/paste this code into it:

```
using System.Collections;
using UnityEngine;
using Tango;

public class AreaLearningStartup : MonoBehaviour, ITangoLifecycle
{
    private TangoApplication m_tangoApplication;

    public void Start()
    {
        m_tangoApplication = FindObjectOfType<TangoApplication>();
        if (m_tangoApplication != null)
        {
            m_tangoApplication.Register(this);
            m_tangoApplication.RequestPermissions();
        }
    }

    public void OnTangoPermissions(bool permissionsGranted)
    {
        if (permissionsGranted)
        {
            AreaDescription[] list = AreaDescription.GetList();
            AreaDescription mostRecent = null;
            AreaDescription.Metadata mostRecentMetadata = null;
            if (list.Length > 0)
            {
                // Find and load the most recent Area Description
                mostRecent = list[0];
                mostRecentMetadata = mostRecent.GetMetadata();
                foreach (AreaDescription areaDescription in list)
                {
                    AreaDescription.Metadata metadata = areaDescription.GetMetadata();
                    if (metadata.m_dateTime > mostRecentMetadata.m_dateTime)
                    {
                        mostRecent = areaDescription;
                        mostRecentMetadata = metadata;
                    }
                }

                m_tangoApplication.Startup(mostRecent);
            }
```

```
            else
            {
                // No Area Descriptions available.
                Debug.Log("No area descriptions available.");
            }
        }
    }

    public void OnTangoServiceConnected()
    {
    }

    public void OnTangoServiceDisconnected()
    {
    }
}
```

For your application, you will need to implement more functionality in the `OnTangoPermissions` function. In this sample, we always pick the most recent area decription and only write a log if there are no visible area descriptions. for your application, you should implement your own logic for how to choose an area description and handle the case when there is no area description available.

Enable localized motion tracking in the Delta Controller

By default, the Tango Delta Camera does not use area descriptions. Enable the **Use Area Description Pose** option on the**Tango Delta Camera** prefab to use the localized coordinates from the area description.

Improve the UX before localization

Before localization happens, the application gets no motion updates. To improve the user experience, we'll show instructions to walk around until we have localized. To keep things simple, we'll use the same image and script used in the area learning example.

If you don't already have one, add a UI Canvas by right-clicking in the Hierarchy panel and selecting **UI > Canvas**.

Next, add a UI Image to the Canvas. Right click on the Canvas in your Hierarchy panel and select **UI > Image**.

Next, define the Source Image for the Image we just added. If you're using the Project Tango SDK for Unity 5, it can be found under **Assets > TangoSDK > Examples > Common > Textures > relocalize_screen** in your Project panel. Otherwise, you can get it from our Unity examples. Drag the file from your Project panel onto the **Source Image** field of the Image in your Hierarchy, then click the **Set Native Size** button so it displays correctly.

Now we'll add a script to listen for localization, and show the image as appropriate. In the Project Tango SDK for Unity 5 it is located at **Assets > TangoSDK > Examples > AreaLearning > Scripts > RelocalizingOverlay**, or get it from our Unityexamples. Drag the script onto the Tango Manager in your Hierarchy to add it.

Finally, we'll associate the image to the script. With the Tango Manager selected, drag the Image containing the relocalize_screen image onto the **Relocalization Overlay** field of the Relocalization Overlay script.

You can now **Build & Run** your application. If you don't have an area description on your device or the most recent area description isn't of the area you're currently in, you'll need to create one to localize against for the localization screen to go away. You can use Project Tango Explorer or any other application that can save area descriptions to do this. We'll also describe saving area descriptions in your own application later in this tutorial.

Congratulations! If you've gotten to this point, your application should be able to load the most recent area description, show a screen with instructions until relocalization occurs, and use the area description's coordinate frame while handling camera movement.

Using Area Learning mode

In this section, we'll describe how to use Area Learning mode to create new area descriptions or extend existing ones. It builds on top of the "Using Load ADF mode" tutorial, so make sure to go through that section first.

Remember that learning mode requires much more computational power than loading an area description. You should restrict learning mode to a seperate setup mode in your application and use Load ADF mode during your main experience.

Handling startup in Area Learning mode

First, make sure that **Auto-connect to Service** is unchecked and **Area Learning** is checked in the **Tango Manager** prefab.

In Area Learning mode, we can use the same startup script as Load ADF mode with one key difference in the behavior of`m_tangoApplication.Startup()`. In Area Learning mode, calling `m_tangoApplication.Startup(null)` is valid, and triggers the creation of a new area description.

In the AreaLearningStartup script, replace the line:

```
Debug.Log("No area descriptions available.");
```

with:

```
m_tangoApplication.Startup(null);
```

Now, our application will load the most recent area description if present, or create a new area description if none exist on the device.

Note: When we load an existing area description in learning mode, we often call this "extending" an area description, since we're accumulating new area description features on top of the existing area description.

Save an area description

Note: The rest of this tutorial is not meant to be copy/pasted directly.

In Area Learning mode, you can save the area description that was learned by calling`AreaDescription.SaveCurrent()` in a place that makes sense for your own application. Saving an area description can take a few minutes, so be sure to save in a background thread.

While the save is running, Tango Events will be sent that describe the save progress. You can listen for the Tango Events by implementing the `ITangoEvent` interface and listening to `AreaDescriptionSaveProgress` events. In the snippet below, we update a GUIText `m_savingText` with the save progress as a percentage. You'll need to create your own implementation to match your UI.

```
public void OnTangoEventAvailableEventHandler(Tango.TangoEvent tangoEvent)
{
    if (tangoEvent.type == TangoEnums.TangoEventType.TANGO_EVENT_AREA_LEARNING
        && tangoEvent.event_key == "AreaDescriptionSaveProgress")
    {
        m_savingText.text = "Saving. " + (float.Parse(tangoEvent.event_value) * 100) +
"%";
```

```
    }
}
```

You can also see saving progress implemented in our area learning sample.

Note: A new file with a new UUID will be created when saving both new ADFs and extended ADFs.

Handling drift corrections

When a relocalization occurs, small errors that have been accumulated are corrected, including errors in the past. We can use this to correct not only the device's location, but also objects that were placed relative to the device, even back in time.

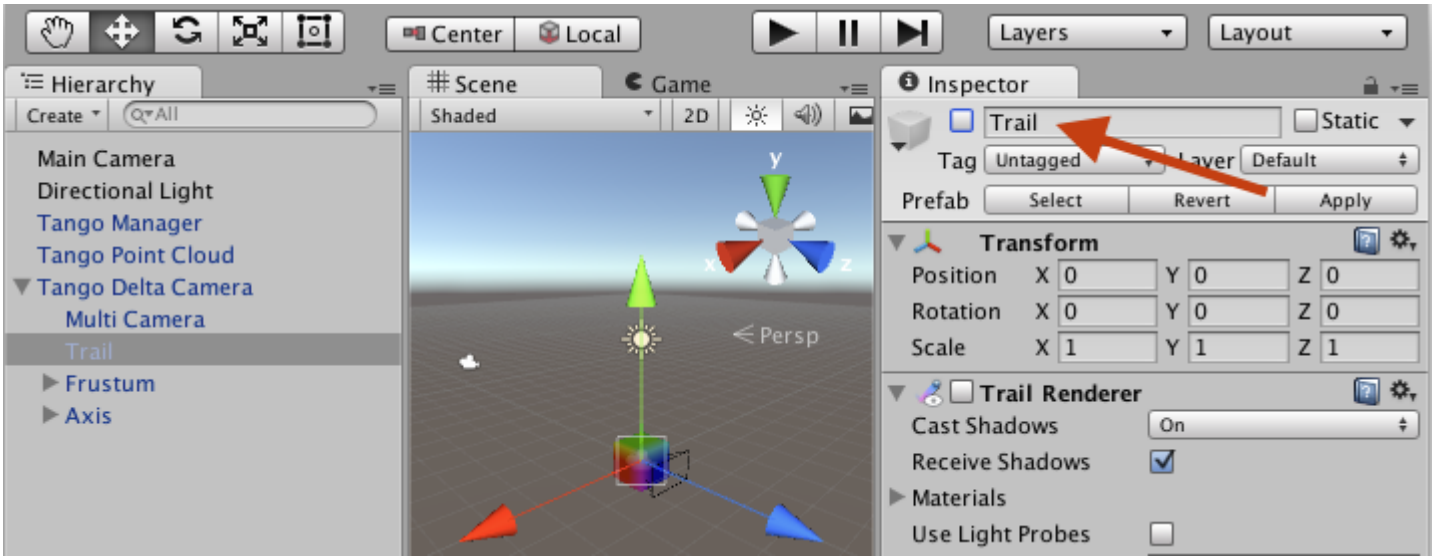
Our augmented reality sample does this. The sample lets you drop virtual markers into the real world. Every time amarker is dropped, we also store the timestamp when we placed the marker and the transformation from the device to the placed marker. When we relocalize, we go through the list of objects, ask for the (now corrected) device pose for each timestamp, and update the object's location using the new device pose and stored transformation.

The code snippet below from our augmented reality sample show's one implementation. Note that it relies on our`ARMarker` class's member variables to keep the timestamp and pose.

```
public void OnTangoPoseAvailable(Tango.TangoPoseData poseData)
{
    if (poseData.framePair.baseFrame ==
        TangoEnums.TangoCoordinateFrameType.TANGO_COORDINATE_FRAME_AREA_DESCRIPTION &&
        poseData.framePair.targetFrame ==
        TangoEnums.TangoCoordinateFrameType.TANGO_COORDINATE_FRAME_START_OF_SERVICE &&
        poseData.status_code == TangoEnums.TangoPoseStatusType.TANGO_POSE_VALID)
    {
        // Adjust mark's position each time we have a loop closure detected.
        foreach (GameObject obj in m_markerList)
        {
            ARMarker tempMarker = obj.GetComponent<ARMarker>();
            if (tempMarker.m_timestamp != -1.0f)
            {
                TangoCoordinateFramePair pair;
                TangoPoseData relocalizedPose = new TangoPoseData();

                pair.baseFrame =
TangoEnums.TangoCoordinateFrameType.TANGO_COORDINATE_FRAME_AREA_DESCRIPTION;
```

```
                pair.targetFrame =
TangoEnums.TangoCoordinateFrameType.TANGO_COORDINATE_FRAME_DEVICE;
                PoseProvider.GetPoseAtTime(relocalizedPose, tempMarker.m_timestamp, pair);
                Vector3 pDevice = new Vector3((float)relocalizedPose.translation[0],
                                              (float)relocalizedPose.translation[1],
                                              (float)relocalizedPose.translation[2]);
                Quaternion qDevice = new Quaternion((float)relocalizedPose.orientation[0],
                                                    (float)relocalizedPose.orientation[1],
                                                    (float)relocalizedPose.orientation[2],
                                                    (float)relocalizedPose.orientation[3])
;

                Matrix4x4 uwTDevice = m_uwTss * Matrix4x4.TRS(pDevice, qDevice,
Vector3.one) * m_dTuc;
                Matrix4x4 uwTMarker = uwTDevice * tempMarker.m_deviceTMarker;

                obj.transform.position = uwTMarker.GetColumn(3);
                obj.transform.rotation = Quaternion.LookRotation(uwTMarker.GetColumn(2),
uwTMarker.GetColumn(1));
            }
        }
    }
}
```


Last updated January 13, 2016.



Last updated January 26, 2016.

Dependency injection

In software engineering, **dependency injection** is a software design pattern that implements inversion of control for resolving dependencies. A dependency is an objectthat can be used (a service). An injection is the passing of a dependency to a dependent object (a client) that would use it. The service is made part of the client'sstate.[1] Passing the service to the client, rather than allowing a client to build or find the service, is the fundamental requirement of the pattern.

Dependency injection allows a program design to follow the dependency inversion principle. The client delegates to external code (the injector) the responsibility of providing its dependencies. The client is not allowed to call the injector code.[2] It is the injecting code that constructs the services and calls the client to inject them. This means the client code does not need to know about the injecting code. The client does not need to know how to construct the services. The client does not need to know which actual services it is using. The client only needs to know about the intrinsic interfaces of the services because these define how the client may use the services. This separates the responsibilities of use and construction.

There are three common means for a client to accept a dependency injection: setter-, interface- and constructor-based injection. Setter and constructor injection differ mainly by when they can be used. Interface injection differs in that the dependency is given a chance to control its own injection. All require that separate construction code (the injector) take responsibility for introducing a client and its dependencies to each other.

Editor Test Runner

The Editor Tests Runner is an implementation of the open source NUnit library - a well-known unit testing library for .Net languages. More information about NUnit can be found on http://www.nunit.org/. The implementaion is based on version 2.6.4.

Getting Started with NUnit

If you are new to NUnit please visit NUnit's Quick Start guide to get started. This article demonstrates the development process with NUnit in the context of a C# banking application.

To start your experience with testing in Unity, open the Editor Test Runner window. If no tests are present in your project, you will be given an option to create a simple test as an example. You can create an editor test file also from the menu (Assets / Create / Editor Test).

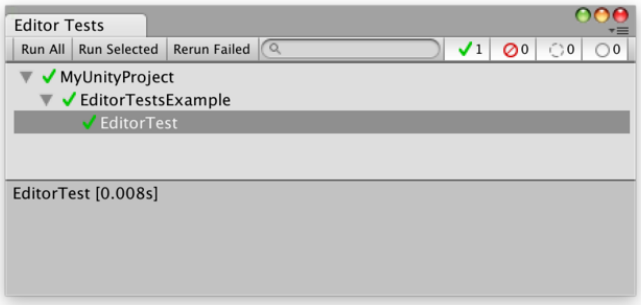
How Editor Tests Runner works

The Editor Test Runner uses NUnit library for authoring and running the tests. Your editor tests should be place under Editor folder (see Special Folders for more information). The runner will scan the assemblies (including external assemblies included in the project that reference *nunit.framework* library) and generate a list with tests that were located.

The advantage of the Editor Tests Runner integrated with the editor over a test runner from an IDE (like MonoDevelop or Visual Studio) is the possibility to invoke certain Unity API. Some Unity APIs will not be accessible if accessed outside of the editor (e.g. instantiating GameObjects). The tests are executed in the editor, in non-play mode, all in one frame. It is not possible to skip a frame and/or execute an API that requires skipping frames.

The code you execute via the test runner will directly affect your currently opened scene. For example, if you instantiate a GameObject, it will be persisted on the scene. Most of the times such behaviour is undesired. The test runner is integrated with the Undo system and will try to undo the changes after the run. In order for that to happen, you need to register every undoable change in the Undo system. Note: Undoing changes may take extra time to execute. As an option, the runner can execute tests on a new scene. When this option selected, before each run the tests runner will open a new scene, execute tests and reopen the original scene. This may prompt the user to save the scene (unless Autosave scene option is selected).

Editor Tests Runner window overview



- Run All — Run all tests.
- Run Selected — Run selected tests.
- Rerun Failed — Runs tests that has previously failed.
- Search — Filters the test list by the gives test name.
- Category filter — Show available categories and allows to filter the tests categories. Visible only if any test has a category assigned.
- Filter by test result — Show or hide tests with certain result. Allow the user to show or hide tests that has: succeeded, failed, are ignored or has not been run.
  At the bottom of the test list (or on the side, depending on the preferences) a test result preview is available. It shows the execution time of the test, the error and exception that has been thrown during the run of the test and the messages that has been logged.

  Additional options are available in the window's menu (top right corner of the window):

- Run on recompilation — Will automatically run all test on a successful recompilation.
- Run tests on a new scene — New scene will be opened before the tests are run. The original scene will be reopened once the run is finished.

- Autosave scene — Automatically save the scene before running tests (available only if the “Run tests on a new scene” option is selected).
- Vertical/Horizontal layout — Select your prefered window layout.

Headless runs (batch mode)

It is possible to run the editor tests in headless mode (using the command-line). In order to do that, run unity with following arguments:

- **runEditorTests** — Run editor tests. This argument is required.
- **editorTestsResultFile** — Path where the result file should be placed. If the path is a folder, a default file name will be used. If not specified, the results will be places in project’s root folder.
- **editorTestsFilter** — Filter tests by names. Can be applied multiple times.
- **editorTestsCategories** — Filter tests by categories. Can be applied multiple times.
- **editorTestsVerboseLog** — Print more to the log. Use “-editorTestsVerboseLog teamcity” for output formated for TeamCity.
  Example:

Run all editor tests from the project:

```
>Unity.exe

-projectPath PATH_TO_YOUR_PROJECT

-runEditorTests
```

Run all editor tests from the project containing “Player” in the name and place the result file in *C:\temp\results.xml*:

```
>Unity.exe -runEditorTests

-projectPath PATH_TO_YOUR_PROJECT

-editorTestsResultFile C:\temp\results.xml

-editorTestsFilter Player
```

The editor will exit with a return following code according to the result:

- **0** — Run succeeded, no failures occurred
- **2** — Run succeeded, some tests failed
- **3** — Run failure (other failure)

TIP

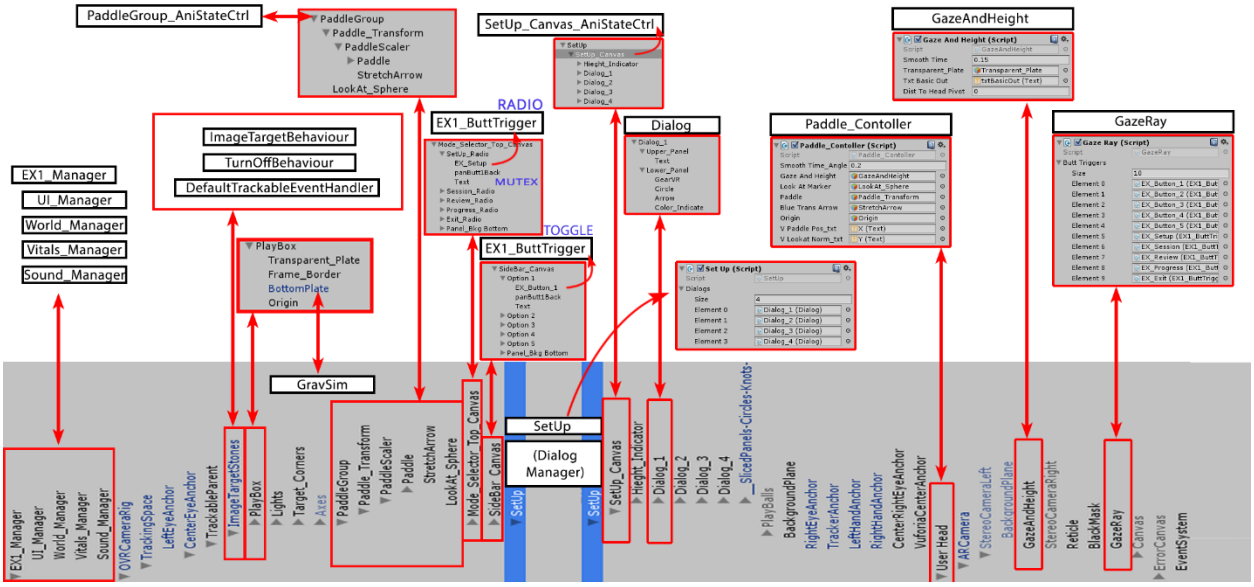
On Windows, in order to read the result code of the executed command, run following:

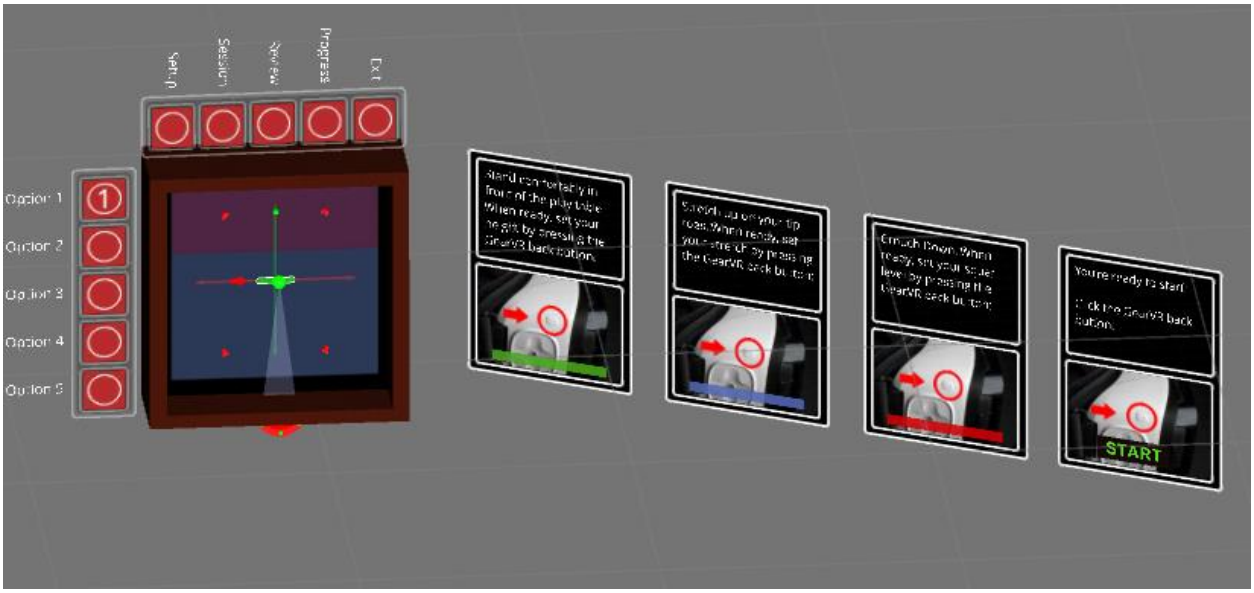
```
start /WAIT Unity.exe ARGUMENT_LIST
```

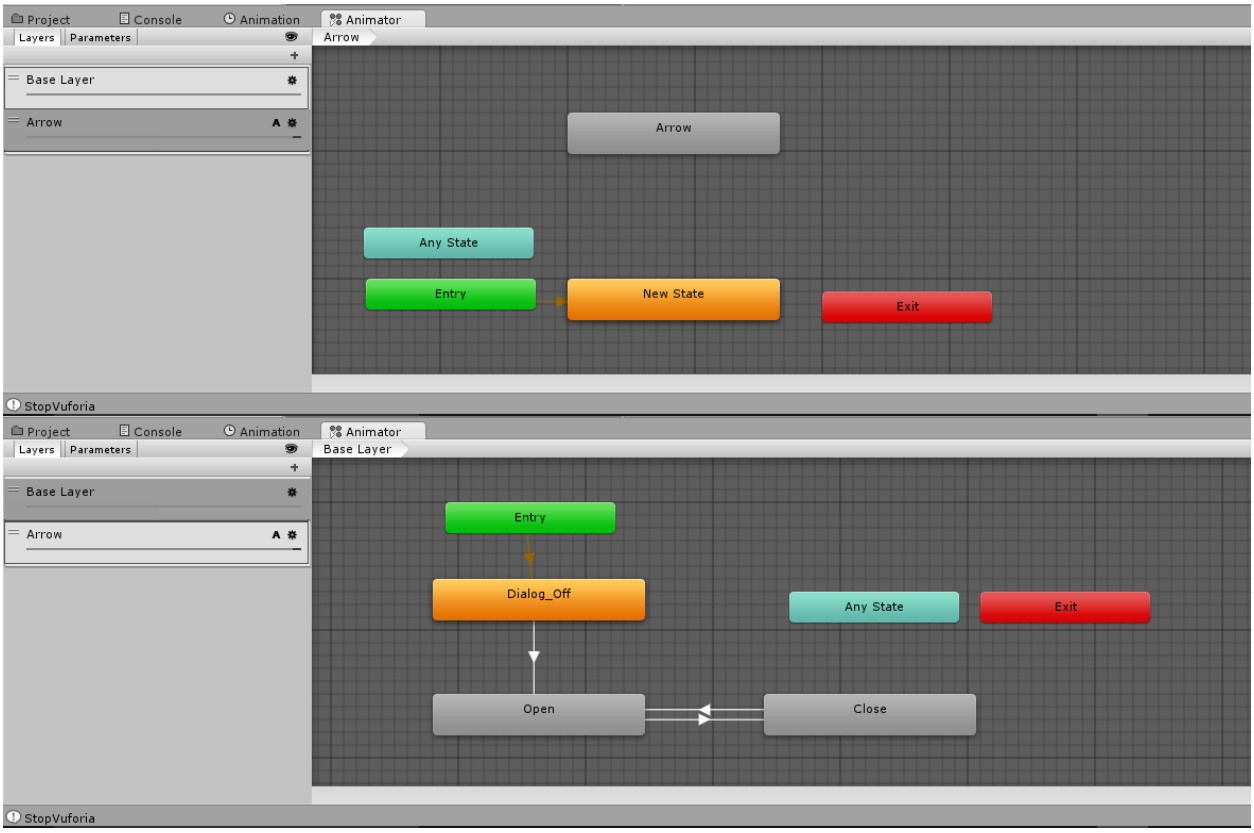
This will wait for the execution of the command to finish and allow you to read the return code from ERRORLEVEL variable.

```
   CoRoutine to indicate Animation complete.

1. IEnumerator onComplete() {
2.
3. while (animator.GetBool("kickedInHead")) {
4. Debug.Log("Animation in progress!");
5. yield return null;
6. }
7.
8. // this will get here when kickedInHead is false
9. Debug.Log("Animation complete!");
10. }
```

EX1 CLASS LOCATION IN UNITY EDITOR HIERARCHY








```
public void ShowDialog(Dialog dialog)
    {
        if (currDialog != null)
            currDialog.IsOpen = false;
        currDialog = dialog;
        currDialog.IsOpen = true;
        StartCoroutine(currDialog.WaitAndPlayArrow());
    }


    public IEnumerator WaitAndPlayArrow()
    {
        // suspend execution for 2.5 seconds
        yield return new WaitForSeconds(2.5f);
        _animator.Play("Arrow", _animator.GetLayerIndex("Arrow"));
    }
```

On Setup Select:

Deactivate Paddle

Activate Setup Canvas

Fade In Menu 1

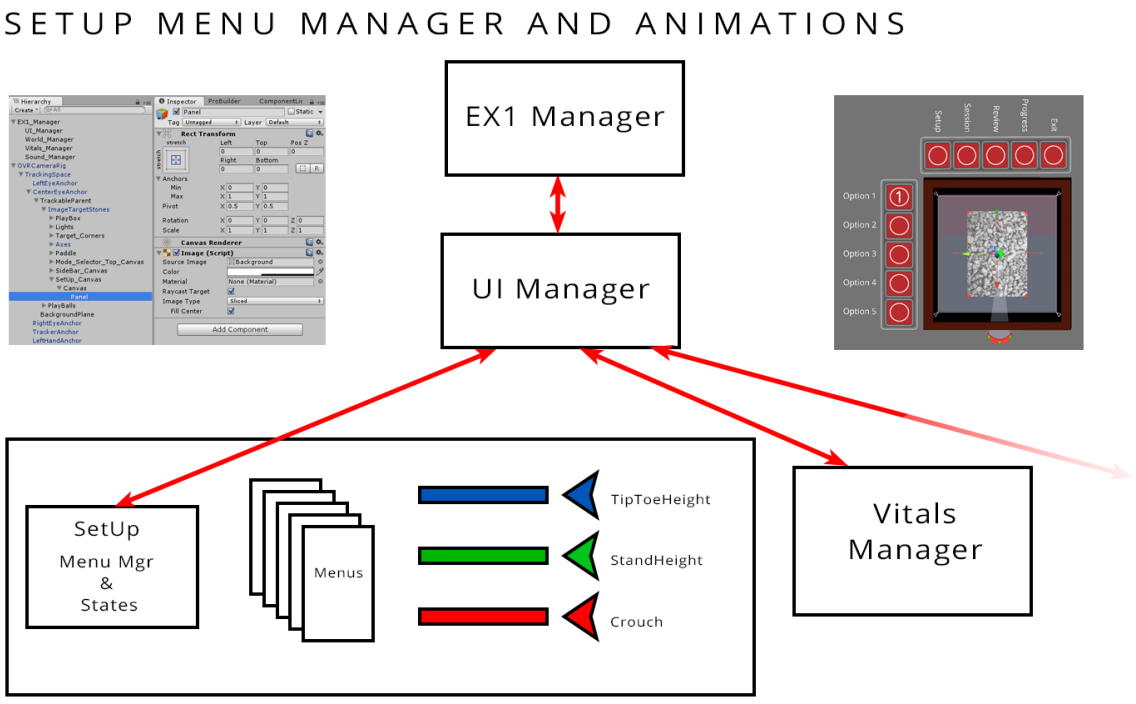
Wait 1.5 seconds

Fade up Arrow and Circle

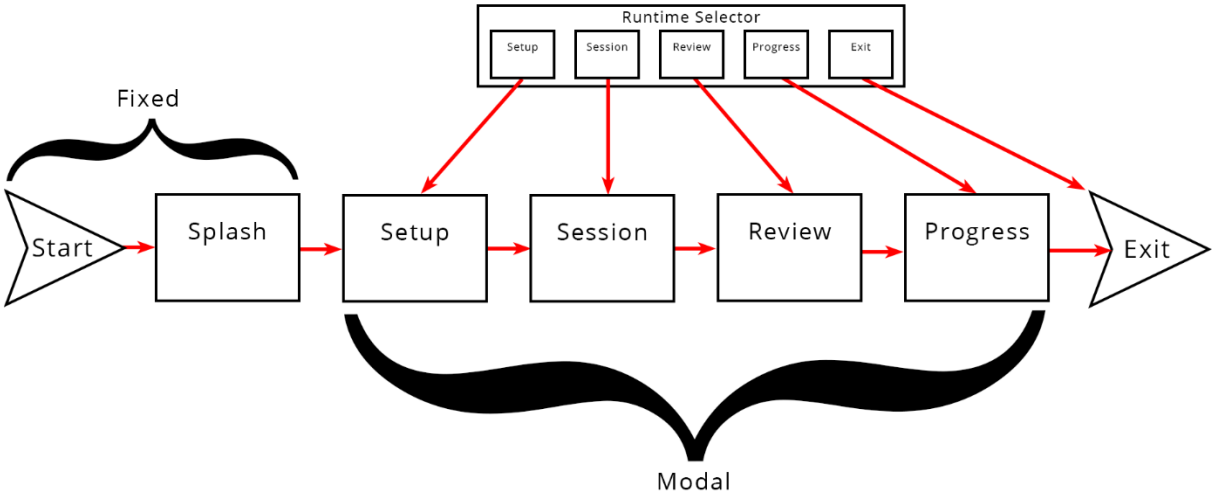
On Click – Cross Fade Next Menu

SetUp

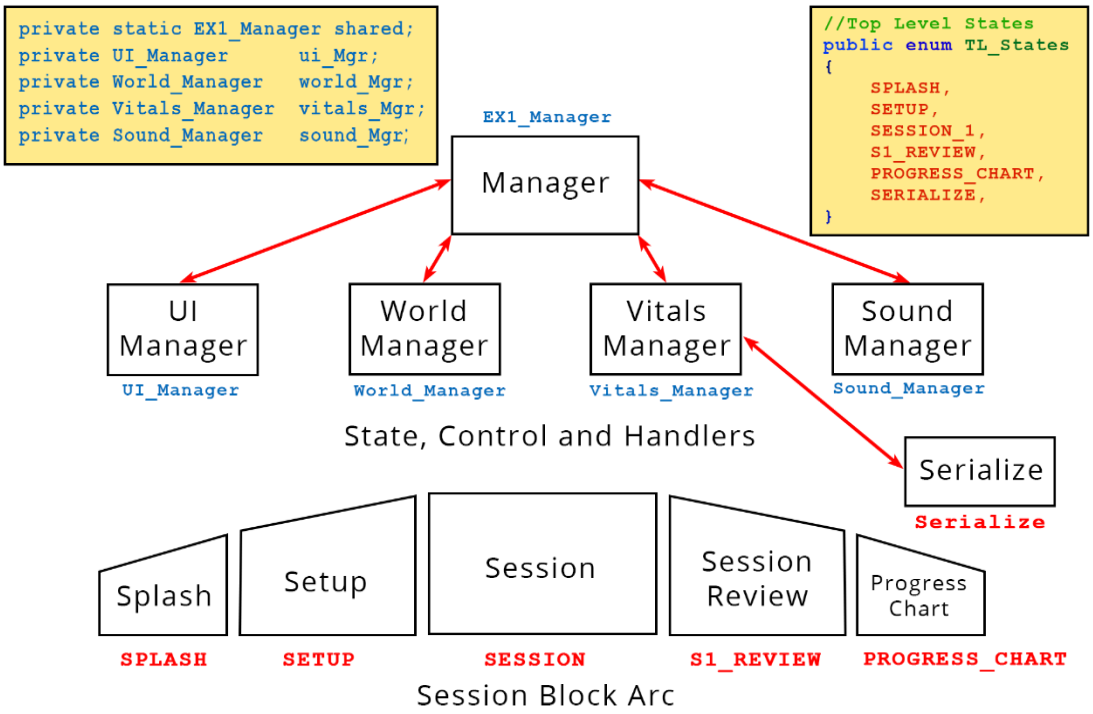
Dialog



RUNTIME MODAL STATE SELECTOR

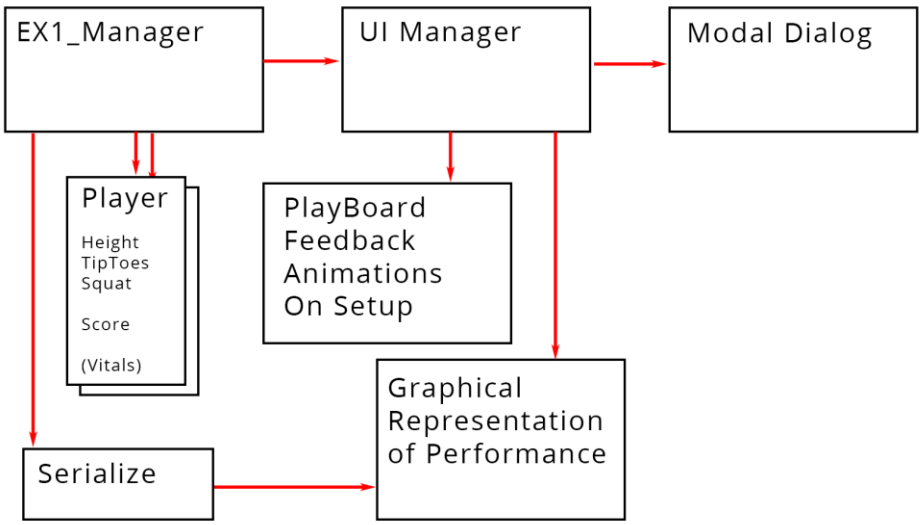


GENERIC AR EXERCISE COMPONENTS

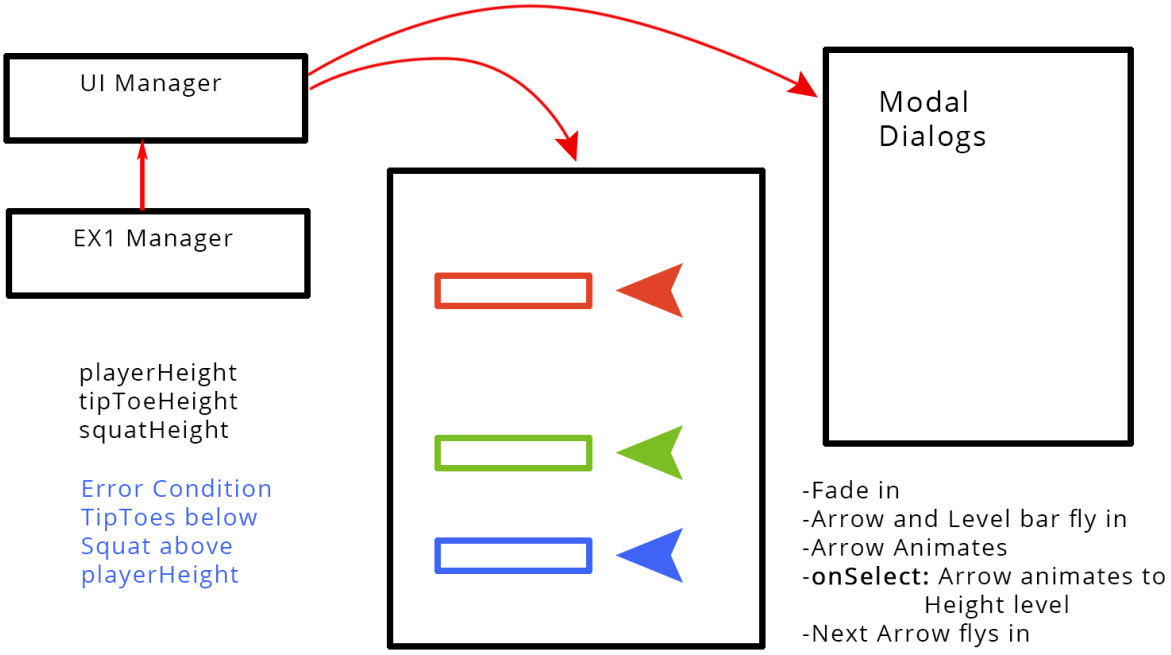


The manager tracks and directs the top level functions/states of the system. Consideration of unit testing and isolation needs to be taken into account. Testing will be conducted from the Manager/Manager-Tester which can be switched back and forth in the Unity Editor.

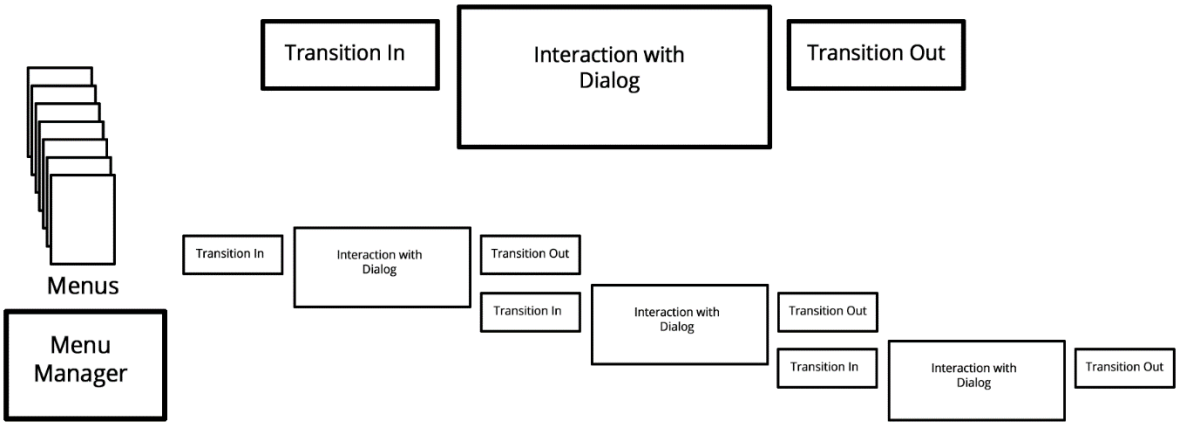
FUNCTIONAL BLOCKS UI CONTROL

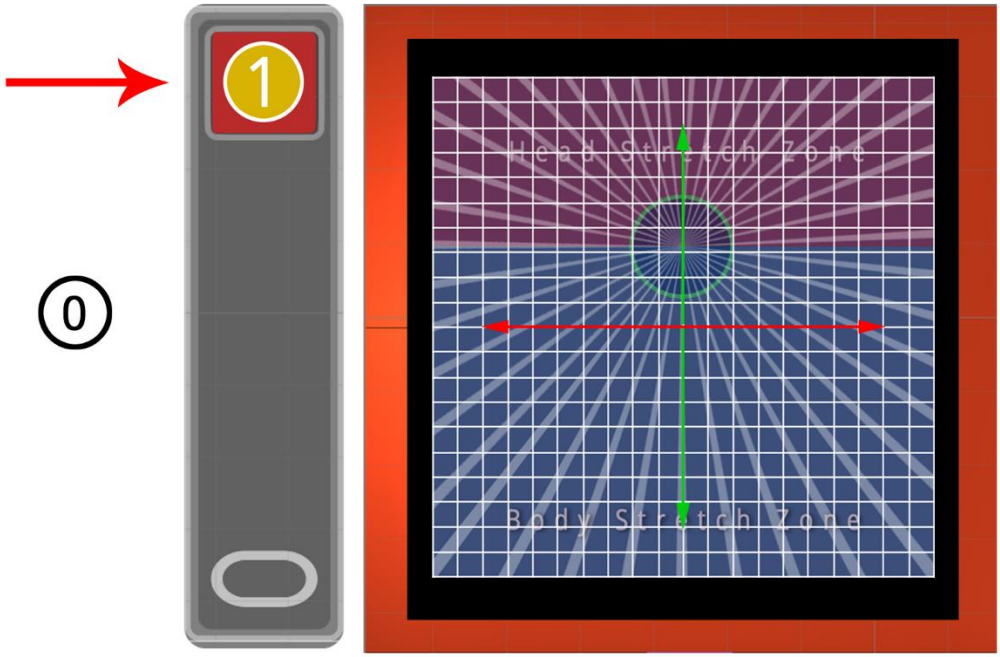


STATE CONTROL - MENU CONTROL

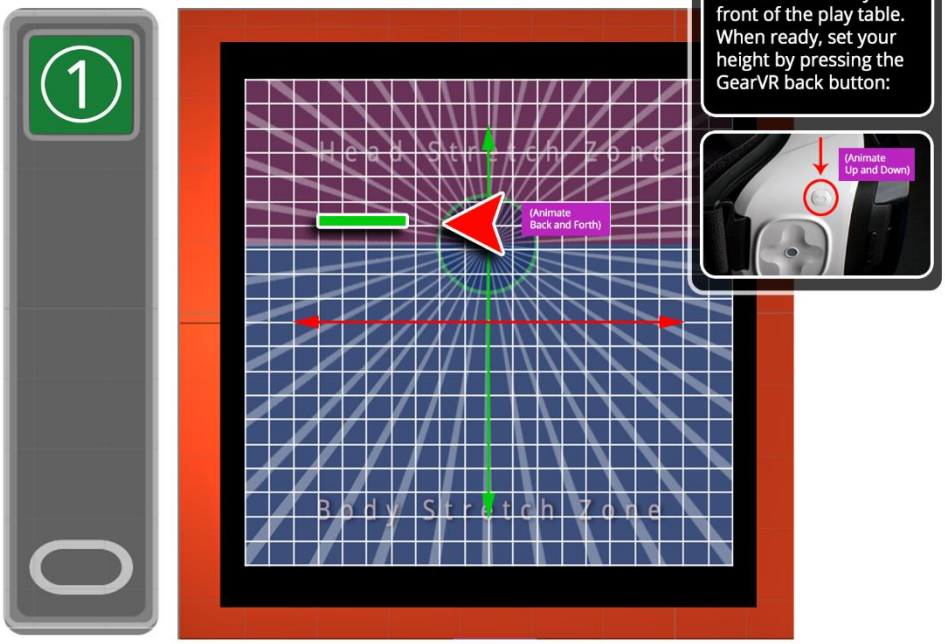


MENU MANAGER and TRANSITION ANIMATION



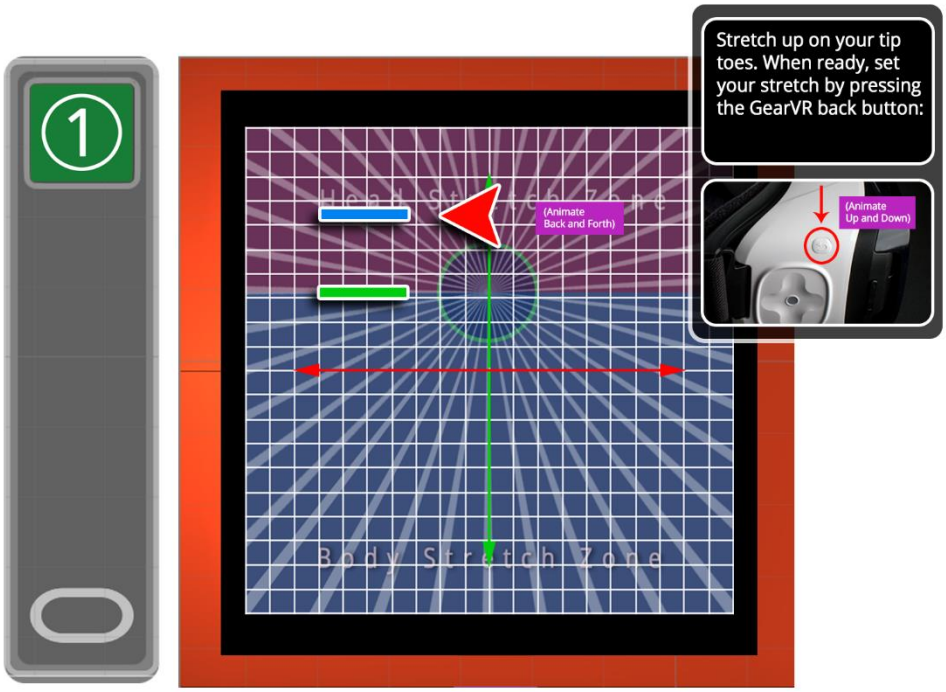

1
Head Stretch Zone
Body Stretch Zone

0

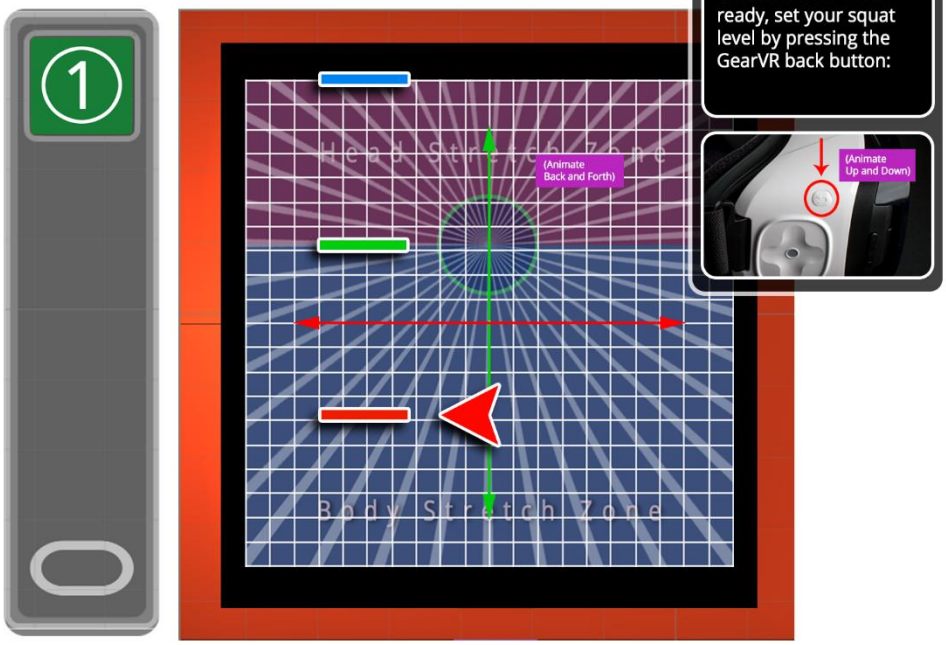

1
Head Stretch Zone
(Animate
Back and Forth)
Body Stretch Zone
Stand comfortably in front of the play table. When ready, set your height by pressing the GearVR back button:
(Animate
Up and Down)

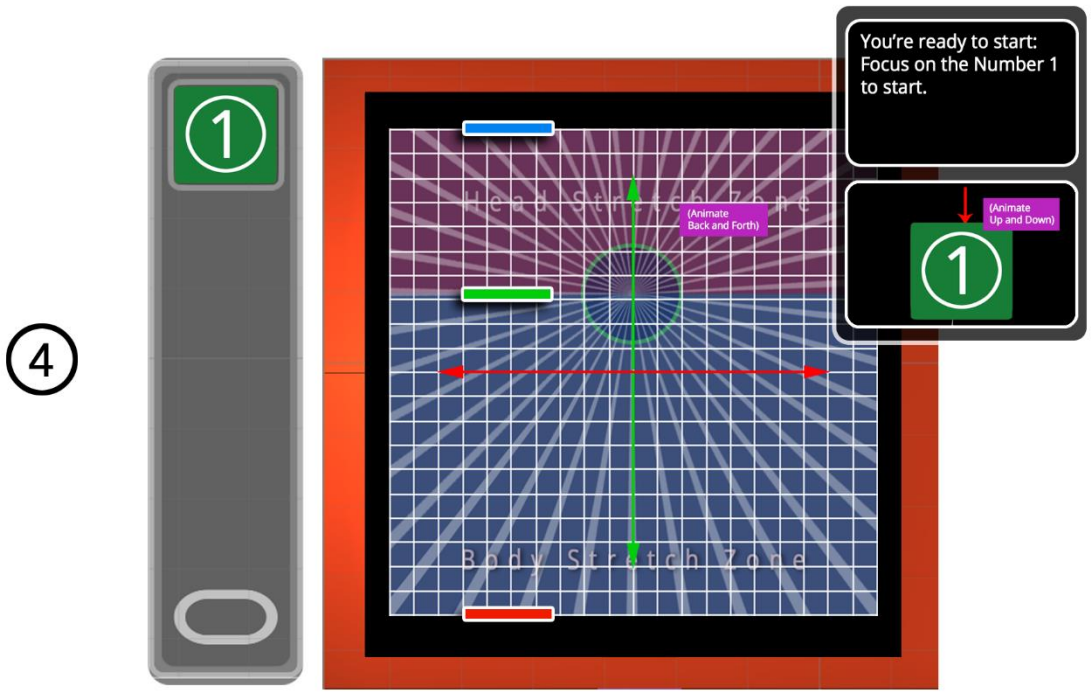
1

2


1
Head Stretch Zone
(Animate Back and Forth)
Body Stretch Zone
Stretch up on your tip toes. When ready, set your stretch by pressing the GearVR back button:
(Animate Up and Down)

3


1
Head Stretch Zone
(Animate Back and Forth)
Body Stretch Zone
Crouch Down. When ready, set your squat level by pressing the GearVR back button:
(Animate Up and Down)

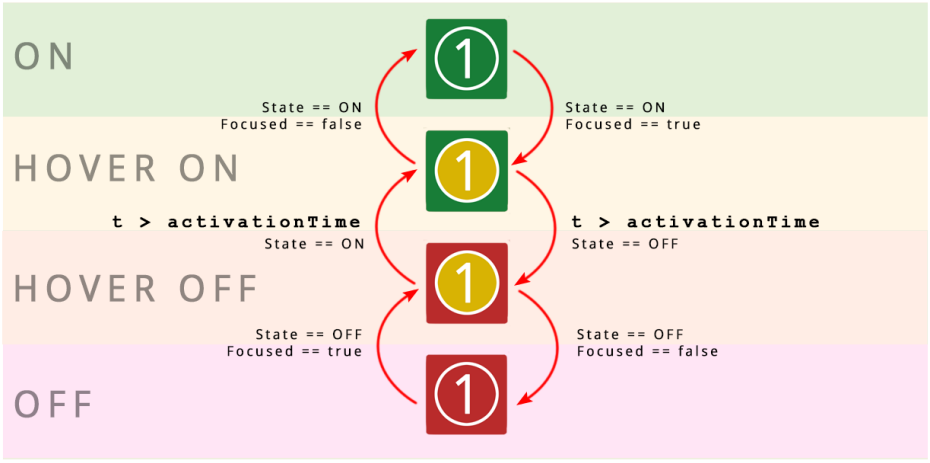


Stand comfortably in front of the play table. When your ready, set your height by pressing the GearVR back button:

BUTTON WORK METHOD 2

States:

ON
HOVER ON
HOVER OFF
OFF

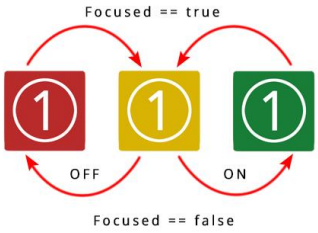


Exercise 1

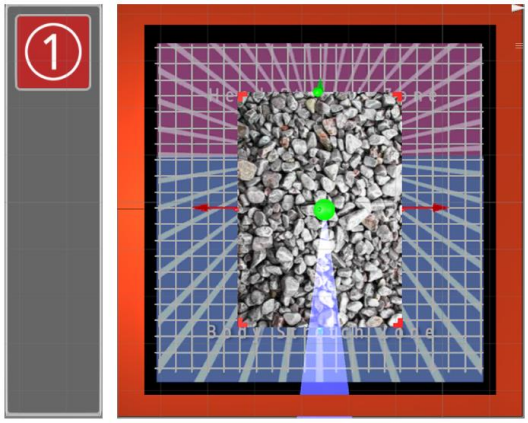
BUTTON WORK

States:

OFF
HOVER
ON

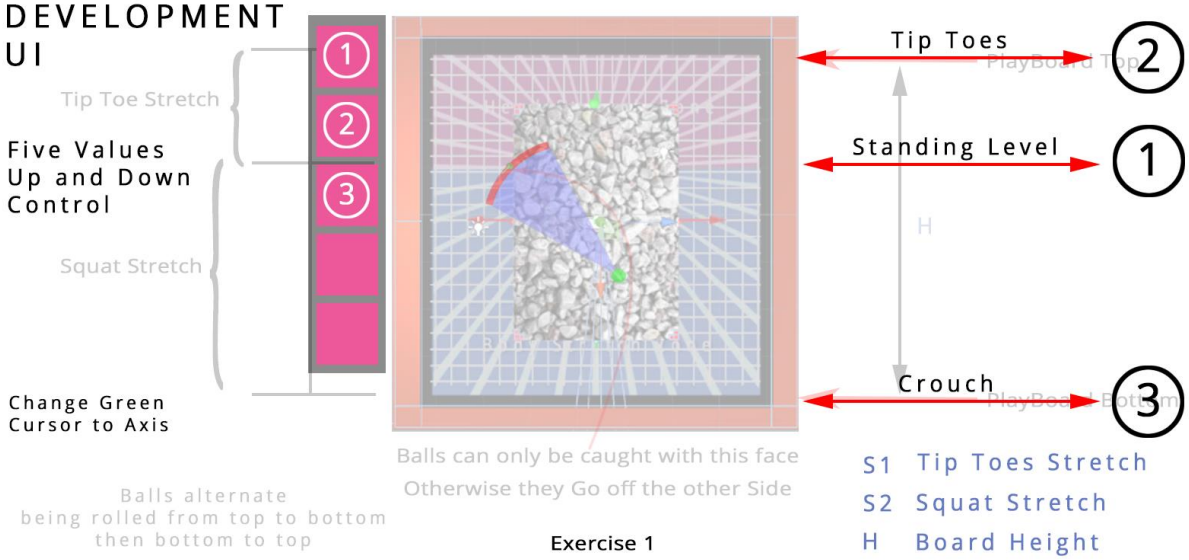


OFF HOVER ON



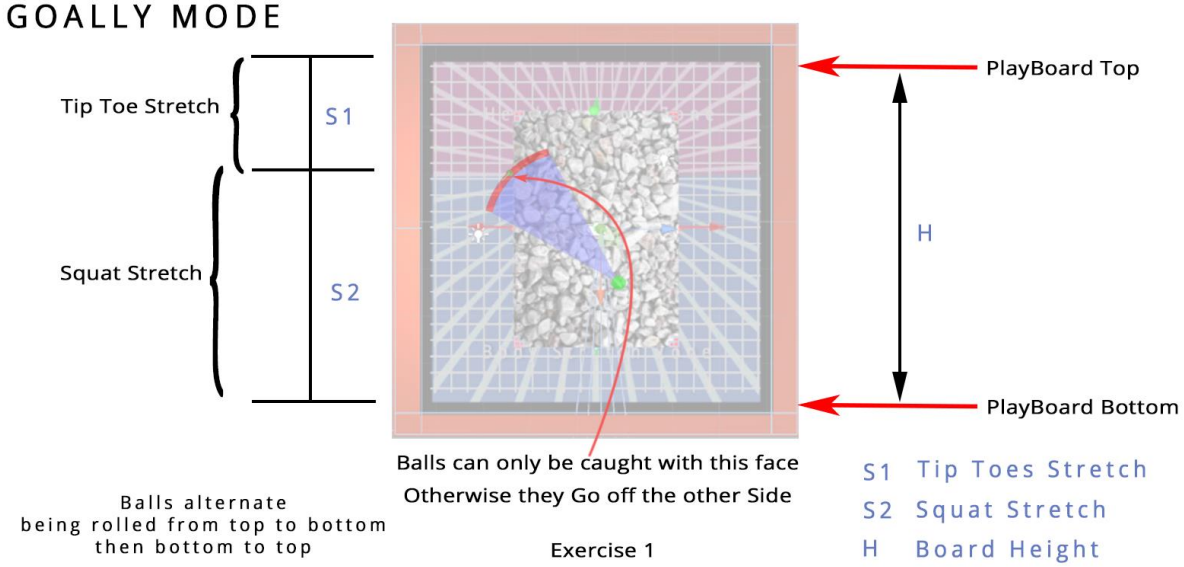
Exercise 1

DEVELOPMENT UI



Exercise 1

GOALLY MODE



Exercise 1

On Sun, Feb 7, 2016 – 10:20pm Told Yez about project. /dj

On Sun, Feb 7, 2016 at 6:35 AM, Paladin Luboff <paladinluboff@gmail.com> wrote:

ARDojo.life ?

Which might be translated to mean 'the place of the augmented reality way'

Unfortunately the .com is not available, but just about everything else looks like it is, including .net, .org, .co and .biz. ARDojo.com appears to have been purchased on February 4 through GoDaddy...they might be willing to sell for cheap.

Or for something longer...arvrdojo.com, or augmenteddojo.com (or with one D).

On Sat, Feb 6, 2016 at 12:12 PM, <feelzy@hushmail.com> wrote:

so we have AR / VR ..... reorder to VR AR and we can make the two R's gogles or a kinect camera looking thing... and the A in between the nose. A little guy with goggles on.... whatever... the base can turn into a smile if we want to animate it. granted, i don't like the sound of VeeeRahrrrr.

so throw an E on it --- and you get VRARE. (v RARE) vrare.com is free. make up whatever you wan the E to mean --- **vi**rtual **r**eality **a**ugmented**r**eality **e**ngine/excellence/extraordinare.

happy ---

what about NR? (NEW REALITY). That's what this is.. a NEW REALITY that can be virtual or augmented.

So --- something simple? NR FITNESS (new reality fitness)? (nrfitness.com is available)

3dfun.fit

3dnewreality.com

funandfit3d.com

funfitnr.com

(easy work with an FF logo)

newrealityfg.com (new reality fitness gaming)

arvrfitness.com

arvrfit.com

nuvizfit.com

nuviz.fit

tizuz.com

newreality.fit

makeu.fit

a blog at arvr.expert ?

futurefitness.guru

newreality.fit

augment ---- augmentia ---- becomes ogmentia.com (ahg-men-chia)

augmenz.com

augitfitness.com (augment it fitness)

ogreality.com (a gangster real estate company???) LOL

cnewbnew.com (see new, be new)

AR_2.unity AR_4 - SCRIBBLE

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

public class Manager : MonoBehaviour
{
    public MenuManager menuManager;
    public ButtonSelector buttonSelector;
    public GameObject Arm;
    public GameObject Sphere1;
    public GameObject Sphere2;
    public GameObject Sphere3;
    public GameObject Stylus;
    public Text Pivot1_Pos;
    public Text RPM0_Val;
    public Text RPM1_Val;
    public Text RPM2_Val;

    public GameObject IndicateDrawing;
    public GameObject IndicateBlue;
    public GameObject IndicateGreen;
    public GameObject IndicateRed;

    public Text DebugText;

    public enum ManagerState
    {
        PIVOT_1,
        RMP_0,
        RPM_1,
        RPM_2,

        DRAW
    }

    public enum AP
    {
        ARM_DRAWING = 0,
        LENGTH1,
        GO1,
        SPHERE2,
        LENGTH2,
        GO2,
        SPHERE3,
        LENGTH3,
        GO3,
    }

    public enum ButtonSel
    {
        DRAW,
        CLEAR,
```

```
    BLUE,
    GREEN,
    RED
}

public enum VBut
{
    Back,
    Up,
    Down,
    Next
}

private int MState = (int)ManagerState.PIVOT_1;

private float fScale1 = 1.0f;
private float P1_Pos_Low = 0.5f;
private float P1_Pos_High = 5.0f;
private float fTimeOn = 0.0f;  //used so initial increments are small for fine tuning
private float fTimeSinceDown = 0.0f;
private bool bUP_On = false;
private bool bDOWN_On = false;
private bool bBack_On = false;
private bool bNext_On = false;
private Transform[] ts;

private Vector3 vLen1Scale, vLen1Pos, vGO1Scale;
private Vector3 vLen2Scale, vLen2Pos, vGO2Scale;
private Vector3 vLen3Scale, vLen3Pos, vGO3Scale;
private float fGoLenRatio;
private bool bAdjustActive;//Values are being adjusted
private bool bDraw = false;
float IncDecTimer = 0.0f;

void Start ()
{

    //Pivot position
    ts = Arm.GetComponentsInChildren<Transform>();
    vLen1Scale = ts[(int)AP.LENGTH1].transform.localScale;
    vGO1Scale = ts[(int)AP.GO1].transform.localScale;
    vLen1Pos = ts[(int)AP.LENGTH1].transform.localPosition;

    vLen2Scale = ts[(int)AP.LENGTH2].transform.localScale;
    vGO2Scale = ts[(int)AP.GO2].transform.localScale;
    vLen2Pos = ts[(int)AP.LENGTH2].transform.localPosition;

    vLen3Scale = ts[(int)AP.LENGTH3].transform.localScale;
    vGO3Scale = ts[(int)AP.GO3].transform.localScale;
    vLen3Pos = ts[(int)AP.LENGTH3].transform.localPosition;

    Pivot1_Pos.text = vLen1Pos.x.ToString("F2");

    //Rotation 0
    RPM0_Val.text = Arm.GetComponent<Pivot_Rotation>().RPM.ToString("F0");

    //Rotation 1
```

```
        RPM1_Val.text = Sphere2.GetComponent<Pivot_Rotation_Z>().RPM.ToString("F0");

        //Rotation 2
        RPM2_Val.text = Sphere3.GetComponent<Pivot_Rotation_Z>().RPM.ToString("F0");

        IndicateDrawing.SetActive(false);//Drawing

        IndicateBlue.SetActive(true);//Drawing
        IndicateGreen.SetActive(false);//Drawing
        IndicateRed.SetActive(false);//Drawing
    }

       void Update ()
    {
        if(bAdjustActive)
        {
            fTimeOn += Time.deltaTime;
            if (fTimeOn > 2.5f) fTimeOn = 2.5f;
        }
        else
            fTimeOn = 0.0f;

        if (bBack_On || bNext_On)
            fTimeSinceDown = 0.0f;
        else
            fTimeSinceDown += Time.deltaTime;

        //DebugText.text = fTimeSinceDown.ToString("F2");

        StateAdjustments();
    }



    public void StateAdjustments()
    {
        ////////////////////  PIVOT 1  ///////////////////////

        if (MState == (int)ManagerState.PIVOT_1) // First State - Pivot 1 - Adjust Position
        {
            Sphere2.GetComponent<Pivot_Active>().onActive();//Blinking Sphere Pivot 1

            if (bUP_On)
            {
                if (vLen1Pos.x < P1_Pos_High - 1.0f)
                {
                    //single step for fine tuning for first 2.5 seconds
                    if (fTimeOn < 2.5f)
                    {
                        vLen1Pos.x += 0.1f * Time.deltaTime;
                    }
                    else//standard speed if not near boundary
                    {
                        vLen1Pos.x += (P1_Pos_High - P1_Pos_Low) * Time.deltaTime / 8.0f;
                    }
                }
                else//slow to boundary
```

```
                {
                    vLen1Pos.x += (P1_Pos_High - P1_Pos_Low) * (P1_Pos_High - vLen1Pos.x) *
Time.deltaTime / 8.0f;
                }
                RepositionPivot1(vLen1Pos.x);

                Pivot1_Pos.text = vLen1Pos.x.ToString("F2");
            }
            if (bDOWN_On)
            {
                //single step for fine tuning for first 2.5 seconds
                if (vLen1Pos.x > P1_Pos_Low + 1.0f)
                {
                    if (fTimeOn < 2.5f)// standard speed if not near boundary
                    {
                        vLen1Pos.x -= 0.1f * Time.deltaTime;
                    }
                    else// standard speed if not near boundary
                    {
                        vLen1Pos.x -= (P1_Pos_High - P1_Pos_Low) * Time.deltaTime /8.0f;
                    }
                }
                else//slow to boundary
                {
                    vLen1Pos.x -= (P1_Pos_High - P1_Pos_Low) * (vLen1Pos.x - P1_Pos_Low) *
Time.deltaTime / 8.0f;
                }
                RepositionPivot1(vLen1Pos.x);

                Pivot1_Pos.text = vLen1Pos.x.ToString("F2");
            }
        }
        else
        {
            //Sphere2.GetComponent<Pivot_Active>().onDeactivate();
        }

        ////////////////////  RPM_0  ////////////////////////

        if (MState == (int)ManagerState.RMP_0) // Second State - RPM 0 - Adjust Adjust RPM
        {
            Sphere1.GetComponent<Pivot_Active>().onActive();//Blinking Sphere Pivot 0
            float fRPM = Arm.GetComponent<Pivot_Rotation>().RPM;

            if (bUP_On)
            {
                if (fRPM < 70.0f)
                {
                    //single step for fine tuning for first 1.5 seconds
                    if (fTimeOn < 2.5f)
                    {
                        IncDecTimer += 1.50f * Time.deltaTime;
                        fRPM += Mathf.FloorToInt(IncDecTimer);
                    }
                    else//standard speed if not near boundary
                    {
                        IncDecTimer += 10.0f * Time.deltaTime;
```

```
                fRPM += Mathf.FloorToInt(IncDecTimer);
            }
        }
        else//slow to boundary
        {
            IncDecTimer += 10.0f * (80.0f - fRPM)/10.0f * Time.deltaTime;
            fRPM += Mathf.FloorToInt(IncDecTimer);
        }
        if (IncDecTimer > 1.0f) IncDecTimer = 0.0f;
        //Debug.Log(IncDecTimer);
        Arm.GetComponent<Pivot_Rotation>().RPM = fRPM;
        RPM0_Val.text = fRPM.ToString("F0");

    }
    if (bDOWN_On)
    {
        if (fRPM > -70.0f)
        {
            //single step for fine tuning for first 2.5 seconds
            if (fTimeOn < 2.5f)
            {
                IncDecTimer += 1.5f * Time.deltaTime;
                fRPM -= Mathf.FloorToInt(IncDecTimer);
            }
            else//standard speed if not near boundary
            {
                IncDecTimer += 10.0f * Time.deltaTime;
                fRPM -= Mathf.FloorToInt(IncDecTimer);
            }
        }
        else//slow to boundary
        {
            IncDecTimer += 10.0f * (80.0f + fRPM) / 10.0f * Time.deltaTime;
            fRPM -= Mathf.FloorToInt(IncDecTimer);
        }
        if (IncDecTimer > 1.0f) IncDecTimer = 0.0f;
        //Debug.Log(IncDecTimer);
        Arm.GetComponent<Pivot_Rotation>().RPM = fRPM;
        RPM0_Val.text = fRPM.ToString("F0");
    }
}
else if(MState != (int)ManagerState.DRAW)
{
    Sphere1.GetComponent<Pivot_Active>().onDeactivate();//Blinking Sphere Pivot 0
}

//////////////////  RPM_1  ///////////////////////

if (MState == (int)ManagerState.RPM_1) // Third State - RPM 1 - Adjust Adjust RPM
{
    Sphere2.GetComponent<Pivot_Active>().onActive();//Blinking Sphere Pivot 1
    float fRPM = Sphere2.GetComponent<Pivot_Rotation_Z>().RPM;

    if (bDOWN_On)
    {
        if (fRPM < 70.0f)
        {
```

```
                //single step for fine tuning for first 2.5 seconds
                if (fTimeOn < 2.5f)
                {
                    IncDecTimer += 1.5f * Time.deltaTime;
                    fRPM += Mathf.FloorToInt(IncDecTimer);
                }
                else//standard speed if not near boundary
                {
                    IncDecTimer += 10.0f * Time.deltaTime;
                    fRPM += Mathf.FloorToInt(IncDecTimer);
                }
            }
            else//slow to boundary
            {
                IncDecTimer += 10.0f * (80.0f - fRPM) / 10.0f * Time.deltaTime;
                fRPM += Mathf.FloorToInt(IncDecTimer);
            }
            if (IncDecTimer > 1.0f) IncDecTimer = 0.0f;
            //Debug.Log(IncDecTimer);
            Sphere2.GetComponent<Pivot_Rotation_Z>().RPM = fRPM;
            fRPM = -fRPM;
            RPM1_Val.text = fRPM.ToString("F0");

        }
        if (bUP_On)
        {
            if (fRPM > -70.0f)
            {
                //single step for fine tuning for first 2.5 seconds
                if (fTimeOn < 2.5f)
                {
                    IncDecTimer += 1.5f * Time.deltaTime;
                    fRPM -= Mathf.FloorToInt(IncDecTimer);
                }
                else//standard speed if not near boundary
                {
                    IncDecTimer += 10.0f * Time.deltaTime;
                    fRPM -= Mathf.FloorToInt(IncDecTimer);
                }
            }
            else//slow to boundary
            {
                IncDecTimer += 10.0f * (80.0f + fRPM) / 10.0f * Time.deltaTime;
                fRPM -= Mathf.FloorToInt(IncDecTimer);
            }
            if (IncDecTimer > 1.0f) IncDecTimer = 0.0f;
            //Debug.Log(IncDecTimer);
            Sphere2.GetComponent<Pivot_Rotation_Z>().RPM = fRPM;
            fRPM = -fRPM;
            RPM1_Val.text = fRPM.ToString("F0");
        }
    }
    else
    {
        //Sphere2.GetComponent<Pivot_Active>().onDeactivate();//Blinking Sphere Pivot 1
    }
```

```
        if(MState != (int)ManagerState.PIVOT_1 && MState != (int)ManagerState.RPM_1 && MState != (int)ManagerState.DRAW)
        {
            Sphere2.GetComponent<Pivot_Active>().onDeactivate();
        }

        ////////////////////  RPM_2  ////////////////////////

        if (MState == (int)ManagerState.RPM_2) // Forth State - RPM 2 - Adjust Adjust RPM
        {
            Sphere3.GetComponent<Pivot_Active>().onActive();//Blinking Sphere Pivot 3
            float fRPM = Sphere3.GetComponent<Pivot_Rotation_Z>().RPM;

            if (bDOWN_On)
            {
                if (fRPM < 70.0f)
                {
                    //single step for fine tuning for first 2.5 seconds
                    if (fTimeOn < 2.5f)
                    {
                        IncDecTimer += 2.0f * Time.deltaTime;
                        fRPM += Mathf.FloorToInt(IncDecTimer);
                    }
                    else//standard speed if not near boundary
                    {
                        IncDecTimer += 20.0f * Time.deltaTime;
                        fRPM += Mathf.FloorToInt(IncDecTimer);
                    }
                }
                else//slow to boundary
                {
                    IncDecTimer += 10.0f * (80.0f - fRPM) / 10.0f * Time.deltaTime;
                    fRPM += Mathf.FloorToInt(IncDecTimer);
                }
                if (IncDecTimer > 1.0f) IncDecTimer = 0.0f;
                //Debug.Log(IncDecTimer);
                Sphere3.GetComponent<Pivot_Rotation_Z>().RPM = fRPM;
                fRPM = -fRPM;
                RPM2_Val.text = fRPM.ToString("F0");

            }
            if (bUP_On)
            {
                if (fRPM > -70.0f)
                {
                    //single step for fine tuning for first 2.5 seconds
                    if (fTimeOn < 2.5f)
                    {
                        IncDecTimer += 2.0f * Time.deltaTime;
                        fRPM -= Mathf.FloorToInt(IncDecTimer);
                    }
                    else//standard speed if not near boundary
                    {
                        IncDecTimer += 20.0f * Time.deltaTime;
                        fRPM -= Mathf.FloorToInt(IncDecTimer);
                    }
                }
```

```
            else//slow to boundary
            {
                IncDecTimer += 10.0f * (80.0f + fRPM) / 10.0f * Time.deltaTime;
                fRPM -= Mathf.FloorToInt(IncDecTimer);
            }
            if (IncDecTimer > 1.0f) IncDecTimer = 0.0f;
            //Debug.Log(IncDecTimer);
            Sphere3.GetComponent<Pivot_Rotation_Z>().RPM = fRPM;
            fRPM = -fRPM;
            RPM2_Val.text = fRPM.ToString("F0");
        }
    }
    if (MState == (int)ManagerState.DRAW)
    {
        if (bUP_On)
        {
            if (fTimeOn > 0.8f)
            {
                int i = buttonSelector.Curr_Select_Idx;
                IncDecTimer += 2.0f * Time.deltaTime;//Decrement every 0.5 seconds

                if(IncDecTimer>1.0f)
                {
                    if (--i < 0)
                        i = 4;
                    buttonSelector.ButtonSelect(i);
                    IncDecTimer = 0.0f;
                }
            }
            else IncDecTimer = 0.0f;
        }
        if (bDOWN_On)
        {
            if (fTimeOn > 0.8f)
            {
                int i = buttonSelector.Curr_Select_Idx;
                IncDecTimer += 2.0f * Time.deltaTime;//Decrement every 0.5 seconds

                if (IncDecTimer > 1.0f)
                {
                    if (++i > 4)
                        i = 0;
                    buttonSelector.ButtonSelect(i);
                    IncDecTimer = 0.0f;
                }
            }
            else IncDecTimer = 0.0f;
        }
    }
    else if(MState != (int)ManagerState.RPM_2)
    {
        Sphere3.GetComponent<Pivot_Active>().onDeactivate();//Blinking Sphere Pivot 3
    }
}

public void RepositionPivot1(float val)
{
```

```
        ts[(int)AP.LENGTH1].transform.localScale = new Vector3(vLen1Scale.x, val, vLen1Scale.z);
        ts[(int)AP.GO1].transform.localScale = new Vector3(vGO1Scale.x, 0.4f / val, vGO1Scale.z);
        ts[(int)AP.LENGTH1].transform.localPosition = new Vector3(val, vLen1Pos.y, vLen1Pos.z);
        ts[(int)AP.GO1].transform.localPosition = new Vector3(0.0f, 1.0f, 0.0f);

        val = (6.0f - val) / 2.0f;

        ts[(int)AP.LENGTH2].transform.localScale = new Vector3(vLen2Scale.x, val, vLen2Scale.z);
        ts[(int)AP.GO2].transform.localScale = new Vector3(vGO2Scale.x, 0.4f / val, vGO2Scale.z);
        ts[(int)AP.LENGTH2].transform.localPosition = new Vector3(vLen2Pos.x, val, vLen2Pos.z);
        ts[(int)AP.GO2].transform.localPosition = new Vector3(0.0f, 1.0f, 0.0f);

        ts[(int)AP.LENGTH3].transform.localScale = new Vector3(vLen3Scale.x, val, vLen3Scale.z);
        ts[(int)AP.GO3].transform.localScale = new Vector3(vGO3Scale.x, 0.4f / val, vGO3Scale.z);
        ts[(int)AP.LENGTH3].transform.localPosition = new Vector3(vLen3Pos.x, val, vLen3Pos.z);
        ts[(int)AP.GO3].transform.localPosition = new Vector3(0.0f, 1.0f, 0.0f);
    }

    private void onButtonSelectDraw(int iButton)
    {
        if (MState == (int)ManagerState.DRAW)
        {
            if (iButton == (int)VBut.Back)
            {

            }
            if (iButton == (int)VBut.Up)
            {
                int i = buttonSelector.Curr_Select_Idx;
                if (--i < 0)
                    i = 4;
                buttonSelector.ButtonSelect(i);
            }
            if (iButton == (int)VBut.Down)
            {

                int i = buttonSelector.Curr_Select_Idx;
                if (++i > 4)
                    i = 0;
                buttonSelector.ButtonSelect(i);
            }
            if (iButton == (int)VBut.Next)
            {
                if (buttonSelector.Curr_Select_Idx == (int)ButtonSel.DRAW)
                {
                    bDraw = !bDraw;
                    if (bDraw)
                    {
                        StartRotation();
                        IndicateDrawing.SetActive(true);
                    }
                    else
                    {
                        Stop();
                        IndicateDrawing.SetActive(false);
                    }
                    Stylus.GetComponent<Draw_Spline>().Draw(bDraw);
```

```
            }
            if (buttonSelector.Curr_Select_Idx == (int)ButtonSel.CLEAR)
            {
                Stop();
                Reset();
                Stylus.GetComponent<Draw_Spline>().Clear();
                bDraw = false;
                IndicateDrawing.SetActive(bDraw);
            }
            if (buttonSelector.Curr_Select_Idx == (int)ButtonSel.BLUE)
            {
                IndicateBlue.SetActive(true);
                IndicateGreen.SetActive(false);
                IndicateRed.SetActive(false);

                Stylus.GetComponent<Draw_Spline>().setColor(0);
            }
            if (buttonSelector.Curr_Select_Idx == (int)ButtonSel.GREEN)
            {
                IndicateBlue.SetActive(false);
                IndicateGreen.SetActive(true);
                IndicateRed.SetActive(false);

                Stylus.GetComponent<Draw_Spline>().setColor(1);
            }
            if (buttonSelector.Curr_Select_Idx == (int)ButtonSel.RED)
            {
                IndicateBlue.SetActive(false);
                IndicateGreen.SetActive(false);
                IndicateRed.SetActive(true);

                Stylus.GetComponent<Draw_Spline>().setColor(2);
            }
        }
    }
}

private void Reset()
{
    Sphere1.GetComponent<Pivot_Rotation>().reset();
    Sphere2.GetComponent<Pivot_Rotation_Z>().reset();
    Sphere3.GetComponent<Pivot_Rotation_Z>().reset();
}

private void Stop()
{
    Sphere1.GetComponent<Pivot_Rotation>().stop();
    Sphere2.GetComponent<Pivot_Rotation_Z>().stop();
    Sphere3.GetComponent<Pivot_Rotation_Z>().stop();
}



private void StartRotation()
{
    Sphere1.GetComponent<Pivot_Rotation>().start();
    Sphere2.GetComponent<Pivot_Rotation_Z>().start();
```

```
        Sphere3.GetComponent<Pivot_Rotation_Z>().start();
    }
    private void SetBlinkersOn()
    {
        Sphere1.GetComponent<Pivot_Active>().onActive();
        Sphere2.GetComponent<Pivot_Active>().onActive();
        Sphere3.GetComponent<Pivot_Active>().onActive();
        Stylus.GetComponent<Pivot_Active>().onActive();
    }
    private void SetBlinkersOff()
    {
        Sphere1.GetComponent<Pivot_Active>().onDeactivate();
        Sphere2.GetComponent<Pivot_Active>().onDeactivate();
        Sphere3.GetComponent<Pivot_Active>().onDeactivate();
        Stylus.GetComponent<Pivot_Active>().onDeactivate();
    }

    public void OnButtonDown(string name)
    {
        if (name.Equals("Up"))
        {
            bUP_On = true;
            onButtonSelectDraw((int)VBut.Up);

            Debug.Log("UP Key Event received in Manager");
            Debug.Log(Time.time);
        }
        if (name.Equals("Down"))
        {
            bDOWN_On = true;
            onButtonSelectDraw((int)VBut.Down);
            Debug.Log("DOWN Key Event received in Manager");
            Debug.Log(Time.time);
        }
        if (name.Equals("Next"))
        {
            bNext_On = true;
            onButtonSelectDraw((int)VBut.Next);

            if(MState == (int)ManagerState.RPM_2)//going into DRAW
            {
                //Stop();
                //Reset();
                SetBlinkersOn();
            }

            if (MState != (int)ManagerState.DRAW && fTimeSinceDown > 0.9f)
            {
                MState++;
                menuManager.ShowMenu(MState);
            }
            //if(MState == (int)ManagerState.RPM_2)
            //{
            //    bDraw = !bDraw;
            //    Stylus.GetComponent<Draw_Spline>().Draw(bDraw);
            //}
            //Debug.Log(MState);
```

```
            //Debug.Log(Time.time);
        }
        if (name.Equals("Back"))
        {
            bBack_On = true;
            onButtonSelectDraw((int)VBut.Back);
            if (MState == (int)ManagerState.DRAW)
            {
                StartRotation();
                SetBlinkersOff();

                //Stylus.GetComponent<Draw_Spline>().Clear();
            }
            if (MState != (int)ManagerState.PIVOT_1 && fTimeSinceDown > 0.9f)
            {
                MState--;
                menuManager.ShowMenu(MState);
            }

            //Debug.Log(MState);
            //Debug.Log(Time.time);
        }
        IncDecTimer = 0.0f;
        bAdjustActive = true;
        fTimeOn = 0.0f;
        DebugText.text = MState.ToString();
    }

    public void OnButtonUp(string name)
    {
        bAdjustActive = false;
        fTimeOn = 0.0f;
        if (name.Equals("Up"))
        {
            bUP_On = false;
            Debug.Log("Up Key Released ----- In Manager");
            Debug.Log(Time.time);
        }
        if (name.Equals("Down"))
        {
            bDOWN_On = false;
            Debug.Log("DOWN Key Released ----- In Manager");
            Debug.Log(Time.time);
        }
        if (name.Equals("Next"))
        {
            bNext_On = false;
        }
        if (name.Equals("Back"))
        {
            bBack_On = false;
        }
    }
}


using UnityEngine;
```

```
using System.Collections;

public class MenuManager : MonoBehaviour
{
    public Menu Pivot_1;
    //public Menu Pivot_2;
    public Menu RPM_0;
    public Menu RPM_1;
    public Menu RPM_2;
    public Menu DRAW;

    private Menu[] Menus;
    private Menu CurrMenu;
    private int Curr_Menu_Idx;

    public void Start()
    {
        ShowMenu(Pivot_1);

        Curr_Menu_Idx = 0;

        CurrMenu = Pivot_1;

        Menus = new Menu[5];

        Menus[0] = Pivot_1;
        //Menus[1] = Pivot_2;
        Menus[1] = RPM_0;
        Menus[2] = RPM_1;
        Menus[3] = RPM_2;
        Menus[4] = DRAW;

        Menus[0].IsOpen = true;
        Menus[1].IsOpen = false;
        Menus[2].IsOpen = false;
        Menus[3].IsOpen = false;
        Menus[4].IsOpen = false;
    }

    public void ShowMenu(Menu menu)
    {
        if (CurrMenu != null)
            CurrMenu.IsOpen = false;
        CurrMenu = menu;
        CurrMenu.IsOpen = true;
    }

    public void ShowMenu(int Menu_Idx)
    {
        if (CurrMenu != null)
            CurrMenu.IsOpen = false;
        CurrMenu = Menus[Menu_Idx];
        CurrMenu.IsOpen = true;
    }
}
```

```
using UnityEngine;
using System.Collections;

public class Menu : MonoBehaviour
{
    private Animator _animator;
    private CanvasGroup _canvasGroup;

    public bool IsOpen
    {
        get { return _animator.GetBool("IsOpen"); }
        set { _animator.SetBool("IsOpen", value);}
    }

    public void Awake()
    {
        _animator = GetComponent<Animator>();
        _canvasGroup = GetComponent<CanvasGroup>();

        //var rect = GetComponent<RectTransform>();
        //rect.offsetMax = rect.offsetMin = new Vector2(0, 0);
    }
    public void Update()
    {
        if(!_animator.GetCurrentAnimatorStateInfo(0).IsName("Open"))
        {
            _canvasGroup.blocksRaycasts = _canvasGroup.interactable = false;
        }
        else
        {
            _canvasGroup.blocksRaycasts = _canvasGroup.interactable = true;
        }
    }
}
```

Design Patterns Reference:

Delegates Delegates, Events and Singletons with Unity3D – C#

http://www.indiedb.com/games/coco-blast/tutorials/delegates-events-and-singletons-with-unity3d-c

C# delegates, I love you ...
http://forum.unity3d.com/threads/150321-C-delegates-I-love-you

Delegates - Unity Official Tutorials
http://www.youtube.com/watch?v=RSN-A0NZTO0

Explaining Delegates and Events (TIP)
http://www.youtube.com/watch?v=N2zdwKIsXJs

Events

Events - Unity Official Tutorials

http://www.youtube.com/watch?v=6qyR73EO68w

Unity messaging

Messaging Systems
http://wiki.unity3d.com/index.php/Scripts/General#General_Concepts

Singleton

http://wiki.unity3d.com/index.php/Singleton

Coroutines

http://docs.unity3d.com/Documentation/Manual/Coroutines.html

Object pool

http://forum.unity3d.com/threads/76851-Simple-Reusable-Object-Pool-Help-limit-your-instantiations!

Model-View-Controller, State (State Machine) and Observer pattern

http://umu.diva-portal.org/smash/get/diva2:546698/FULLTEXT01.pdf

MVC : MVC in Unity: C# Component scripts and dependencies

http://unity-scripting.blogspot.nl/2013/05/mvc-in-unity-c-component-scripts-and.html
MVC : Why are MVC & TDD not employed more in game architecture?
http://gamedev.stackexchange.com/questions/3426/why-are-mvc-tdd-not-employed-more-in-game-architecture

Object-oriented programming

Polymorphism - Unity Official Tutorials
http://www.youtube.com/watch?v=vLFx4-i3zqM

Overriding - Unity Official Tutorials

http://www.youtube.com/watch?v=zmf6-Vlv31w

Game Managers

http://www.holoville.com/blog/?p=166

Unity.3.x Scripting (BOOK)

How to design your code architecture in unity?

http://answers.unity3d.com/questions/365770/how-to-design-your-code-architecture-in-unity.html

http://video.unity3d.com/video/4929872/luug-11-pt-1-of-4- Understandability, Flexibility and Extendability

http://video.unity3d.com/video/4929984/luug-11-pt-2-of-4- Performance, 2:12 Initialisation

http://video.unity3d.com/video/4930120/luug-11-pt-3-of-4-2:10 Messaging

http://video.unity3d.com/video/4930121/luug-11-pt4-of-4 - Scriptable Objects

C# Patterns

Design Patterns
http://en.wikipedia.org/wiki/Design_Patterns

.NET Design Patterns
http://www.dofactory.com/Patterns/Patterns.aspx

Singleton Objects in Unity

http://blog.hexdragonal.com/post/123650902421/manage-your-game-singleton-gameobjects-in-unity

1. MAKE IT STATIC

Once you've created a code file for your manager, **create a private static reference to your manager inside itself.**

```
DATA HOSTED WITH ♥ BY PASTEBIN.COM - DOWNLOAD RAW - SEE ORIGINAL
1. public class GameManager
2. {
3.     private static GameManager shared;
4. }
```

Static means there is only one instance of the variable ('shared') in memory, no matter how many instances of this class exist. Every variable called 'shared' in every GameManager object will point to the same address and value in memory.

We're making it private so that external sources can't mess with the shared variable. This class may be accessible to everything, but that doesn't mean giving full control to external classes!

2. SET IT

We need to set the shared variable. We'll do this on Awake() as follows:

```
DATA HOSTED WITH ♥ BY PASTEBIN.COM - DOWNLOAD RAW - SEE ORIGINAL
1. private void Awake()
2.   {
3.       // Set the 'shared' variable as this GameManager
4.       if (shared == null) {
5.           shared = this;
6.
7.
8.       } else {
9.           Destroy(this.gameobject);
10.      }
11. }
```

If shared != null, there is already a GameManager in the scene so we destroy 'this' instance of the GameManager. This is handy if your GameManager needs to exist across multiple scenes, you can

just put a GameManager prefab in every scene that needs it and add the 'DontDestroyOnLoad' function.

3. PLAY IT SAFE

Finally, **we need to reference our GameManager in other classes**! We'll add this to our GameManager.cs:

```
DATA HOSTED WITH ♥ BY PASTEBIN.COM - DOWNLOAD RAW - SEE ORIGINAL
1.  public static GameManager Shared()
2.  {
3.          return shared;
4.  }
```

This means that **any object in the Unity scene can gain access to the GameManager cleanly by calling GameManager.Shared()**; The Manager will handle setting itself up and cannot be overwritten externally as the actual static variable is private.

However, **I suggest null checking the function every time you call it**. For example:

```
DATA HOSTED WITH ♥ BY PASTEBIN.COM - DOWNLOAD RAW - SEE ORIGINAL
1.  public class Player
2.  {
3.      // If the game isn't paused, runs the gameplay loop
4.      private void Update()
5.      {
6.          if (GameManager.Shared() != null) {
7.              if (!GameManager.Shared().IsPaused()) {
8.                  DoStuff();
9.              }
10.         }
11.     }
12. }
```

This safe guards your class in case for some reason, there is no manager in the scene. I'd recommend adding an error message in such a scenario so it's easy to track.

The other danger is in god-classing your GameManager, but if you can **treat it as a router/container for information and restrict functionality to other classes**, you can keep your manager clean and under control.

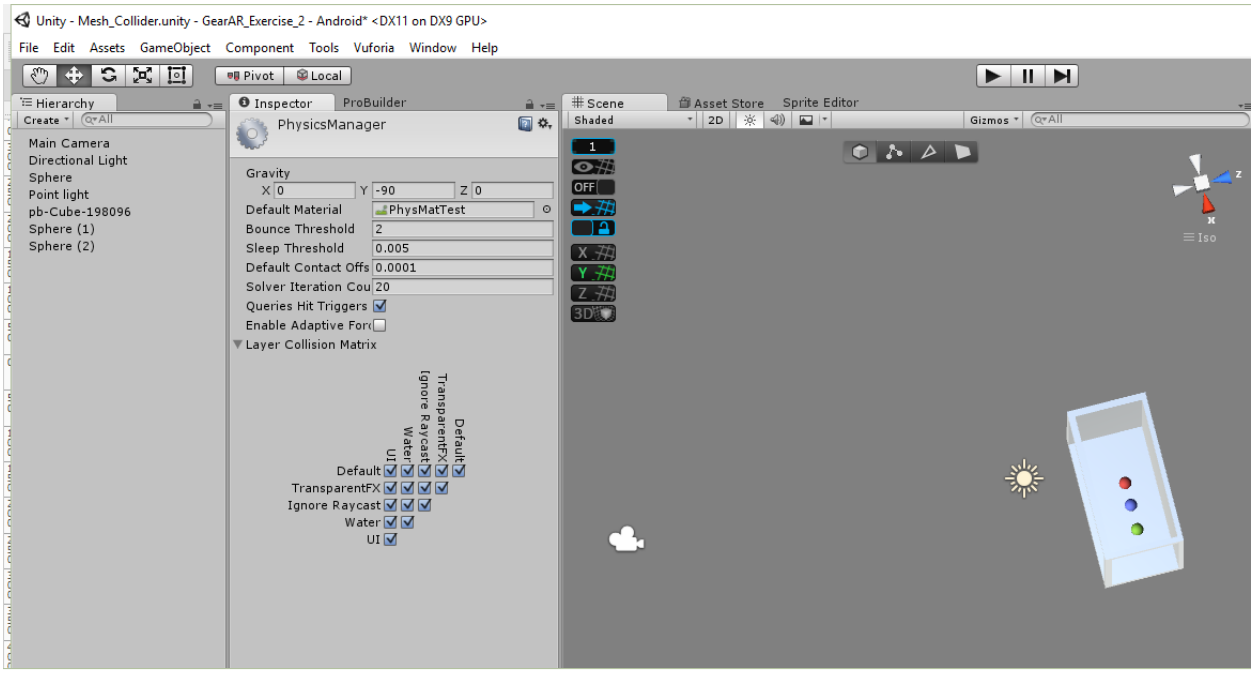
I hope this helps someone, if you have your own solution that adds to or is straight up better than the one I've presented, I'd love to learn from you! **Here's a full example of the classes we just wrote** for reference:

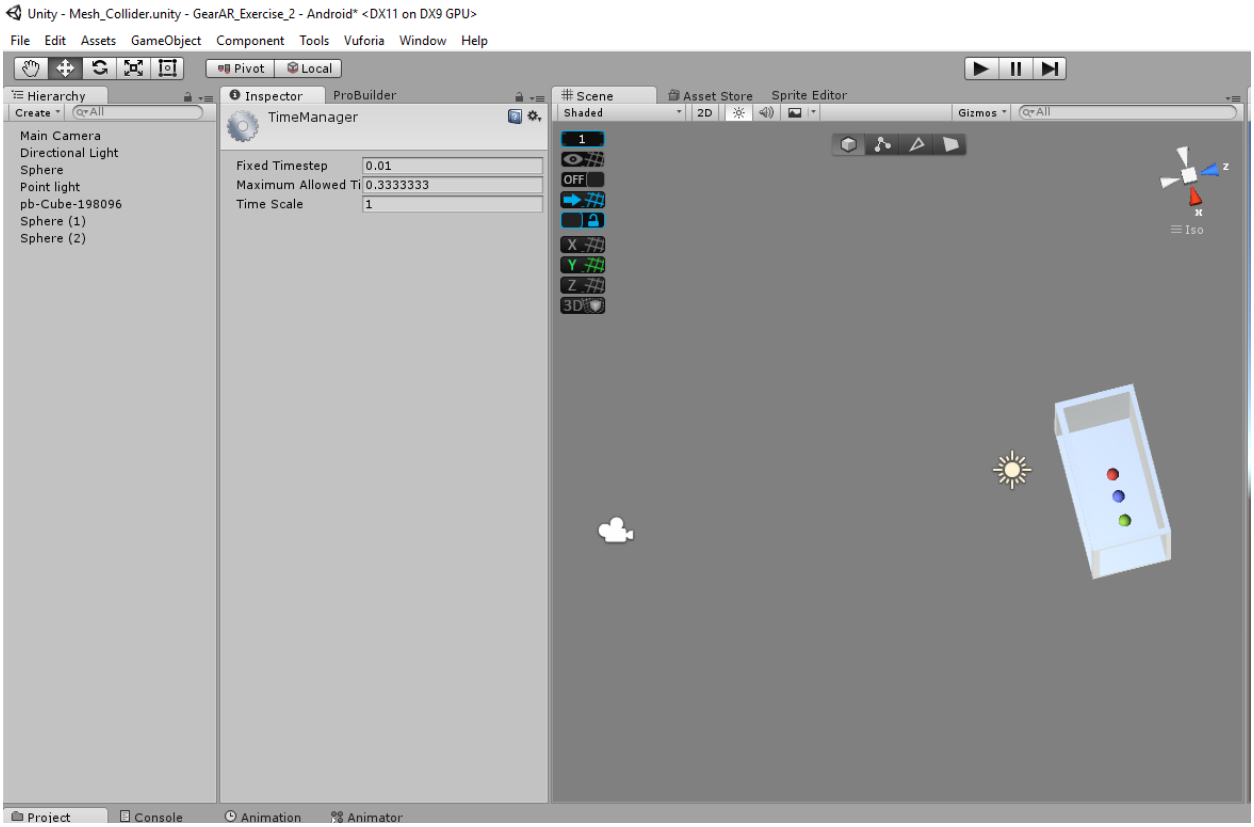

```
using UnityEngine;
using System;

public class GameManager
{
    private static GameManager shared;
    private bool bPaused = false;

    // Setup the static instance of this GameManager
    private void Awake()
    {
        // Set the 'shared' variable as this GameManager
        if (shared == null) {
            shared = this;

        // If shared != null, there is already a GameManager in the scene!
        } else {
            Destroy(this.gameobject);
        }
    }

    // Return the static instance of this GameManager
    public static GameManager Shared()
    {
        return shared;
    }

    // Returns true if the game is paused
    public bool IsPaused()
    {
```

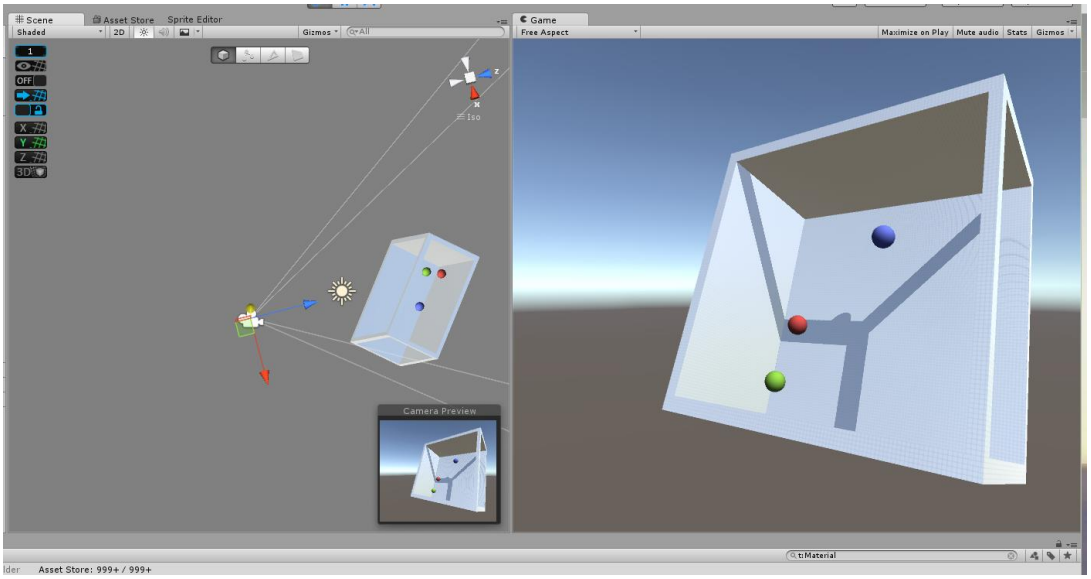
```
31.         return bPaused;
32.     }
33. }
34.
35. public class Player
36. {
37.     // If the game is'nt paused, runs the gameplay loop
38.     private void Update()
39.     {
40.         if (GameManager.Shared() != null) {
41.             if (!GameManager.Shared().IsPaused()) {
42.                 DoStuff();
43.             }
44.         }
45.     }
46.
47.     // Gameplay!
48.     private void DoStuff()
49.     {
50.         // Gameplay!
51.     }
52. }
```

TAGS: game development indie games indie dev Unity Unity3D Singletons Game managers code pattern programming SingletonArclight Cascade

Vimeo Upload at: https://vimeo.com/153605534


Unity - Mesh_Collider.unity - GearAR_Exercise_2 - Android* <DX11 on DX9 GPU>
File Edit Assets GameObject Component Tools Vuforia Window Help
Pivot Local
Hierarchy
Create
Main Camera
Directional Light
Sphere
Point light
pb-Cube-198096
Sphere (1)
Sphere (2)
Inspector ProBuilder
PhysicsManager
Gravity
X 0 Y -90 Z 0
Default Material PhysMatTest
Bounce Threshold 2
Sleep Threshold 0.005
Default Contact Offs 0.0001
Solver Iteration Cou 20
Queries Hit Triggers
Enable Adaptive For
Layer Collision Matrix
Default
TransparentFX
Ignore Raycast
Water
UI
Scene Asset Store Sprite Editor
Shaded 2D Gizmos
Iso


Unity - Mesh_Collider.unity - GearAR_Exercise_2 - Android* <DX11 on DX9 GPU>
File Edit Assets GameObject Component Tools Vuforia Window Help
Pivot Local
Hierarchy
Create
Main Camera
Directional Light
Sphere
Point light
pb-Cube-198096
Sphere (1)
Sphere (2)
Inspector ProBuilder
TimeManager
Fixed Timestep 0.01
Maximum Allowed Ti 0.3333333
Time Scale 1
Scene Asset Store Sprite Editor
Shaded 2D Gizmos
Iso
Project Console Animation Animator



1/30/2016

Physics boundarys for Balls in Container

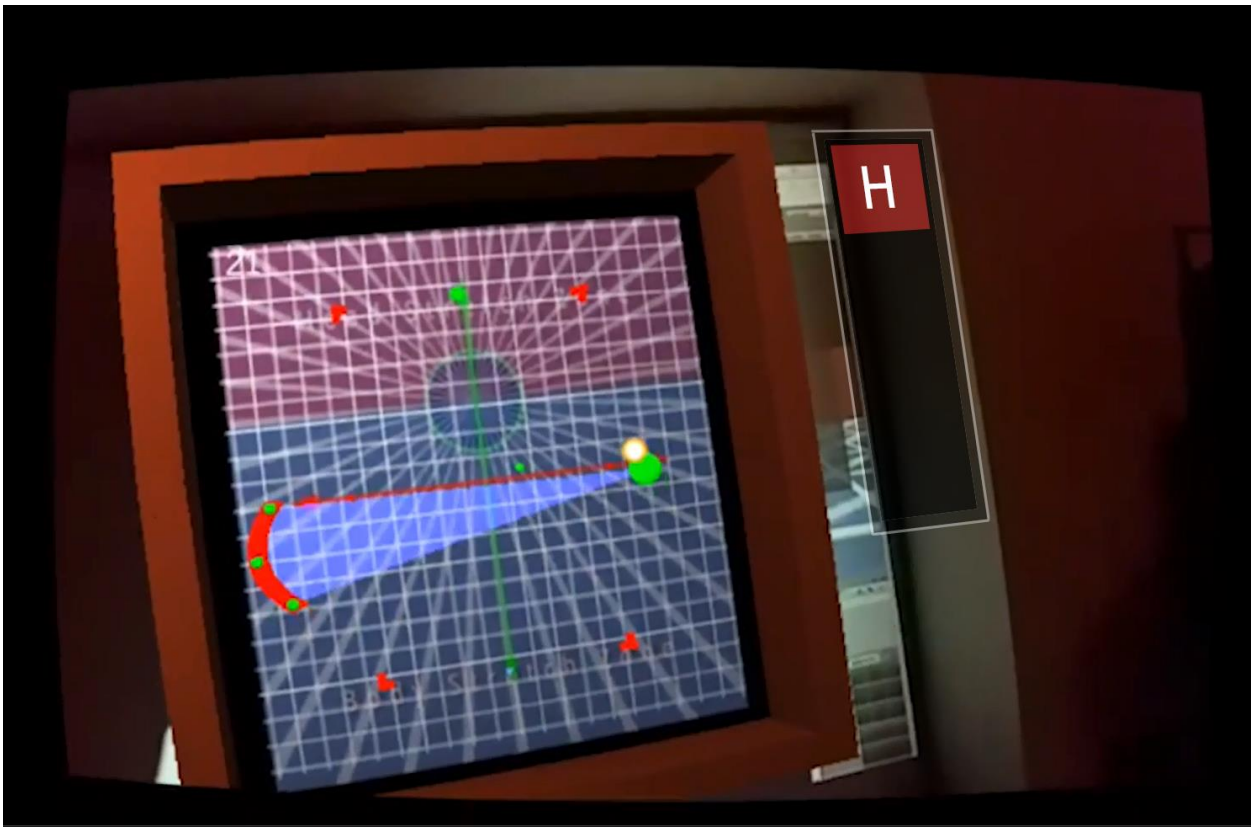
Physical Movement

Interface - Buttons and Sliders – Screen Animations

Sliding Dialogs into main Play area.

Saving Settings

Branding Idea 1/30/2016



WORLD COORDS AND QUATERNIONS

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class EX1_Manager : MonoBehaviour
{
    public float smoothTime_Angle = 0.2f;

    public GameObject gazeAndHeight;
    public GameObject lookAtMarker;
    public GameObject paddle;
    public GameObject BlueTransArrow;

    private Vector3 lookatPos;
    private Vector3 paddlePos;

    public Text vPaddlePos_txt;
    public Text vLookatNorm_txt;

    private float anglePaddle;
    private float angularVelocity;
```

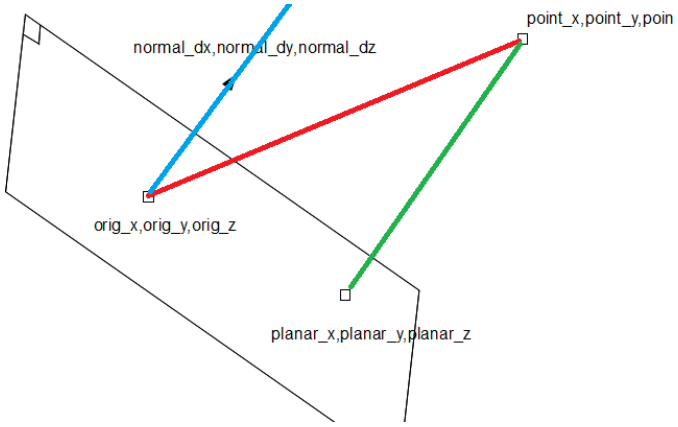
```
    // Use this for initialization
    void Start ()
    {


        }

        // Update is called once per frame
        void Update ()
    {
        lookatPos = gazeAndHeight.GetComponent<GazeAndHeight>().vGaze_Loc;
        paddlePos = gazeAndHeight.GetComponent<GazeAndHeight>().vHeight_Loc;

        //Debug.Log(lookatPos);

        Vector3 vLookat = new Vector3();

        vLookat = lookatPos - paddlePos;// + 5.3f* paddle.transform.up;

        float vDist = vLookat.magnitude;

        //Debug.Log(vLookat);

        lookAtMarker.transform.position = lookatPos;// + 5.3f * paddle.transform.up;
        paddle.transform.position = paddlePos;// + 5.3f * paddle.transform.up; ;

        Quaternion q = new Quaternion();

        vLookat = lookAtMarker.transform.position - paddle.transform.position;
        vLookat.Normalize();
        q = Quaternion.LookRotation(vLookat);
        //float alpha = Mathf.Atan2(-vLookat.y , -vLookat.x)*360.0f/(2.0f* Mathf.PI);


        //anglePaddle = Mathf.SmoothDampAngle(anglePaddle, alpha, ref angularVelocity, smoothTime_Angle);

        paddle.transform.rotation = q; //Quaternion.Euler(0.0f, 0.0f,alpha);

        BlueTransArrow.transform.localScale = new Vector3(vDist / 25, 1f, 1f);

        vPaddlePos_txt.text  = "Paddle Pos = " + paddle.transform.localPosition;
        vLookatNorm_txt.text = "vLookAt Norm = " + vLookat.ToString();

        //Input.GetKeyDown(KeyCode.Escape)
        float x_start = -100.0f;
        if (Input.GetMouseButton(0))
        {
            //Instantiate(prefab, new Vector3(x_start, 0.0f, 0.0f), Quaternion.Euler(new Vector3(90.0f, 0.0f, 0.0f)));
        }
    }
}
```

Drive Point to a Plane

1) Make a vector from your orig point to the point of interest:
v = point-orig (in each dimension);
2) Take the dot product of that vector with the unit normal vector n:
dist = vx*nx + vy*ny + vz*nz; dist = scalar distance from point to plane along the normal

3) Multiply the unit normal vector by the distance, and subtract that vector from your point.
projected_point = point - dist*normal;
Edit with picture: I've modified your picture a bit. Red is 'v'; 'v' dot 'normal' = length of blue and green (dist above). Blue is normal*dist. Green = blue * -1 : to find planar_xyz, start from point and add the green vector.

http://stackoverflow.com/questions/9605556/how-to-project-a-3d-point-to-a-3d-plane



Scale Radially

Adjust ScaleHeight value:

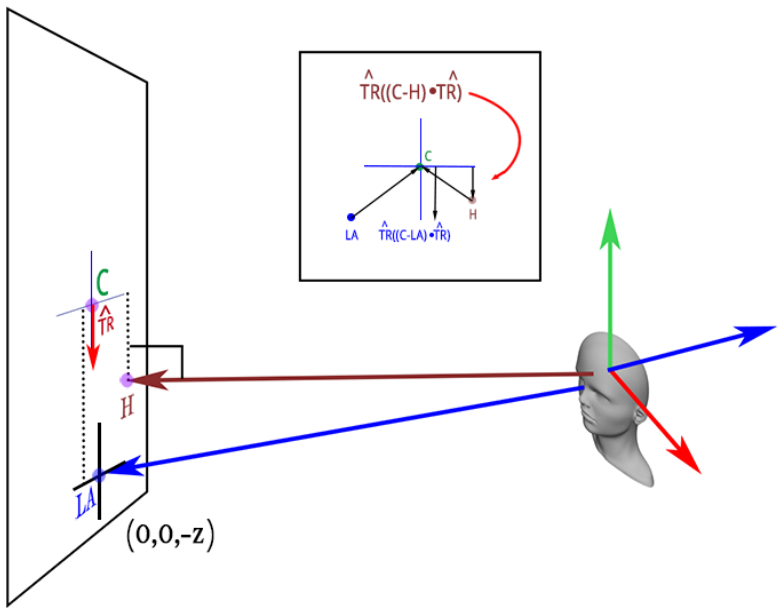
GearVR BackButton:
Input.GetKeyDown(KeyCode.Escape);

```
for (int i = 0; i < hits.Length; i++)
 {
     if (Transparent_Plate.gameObject == hits[i].collider.gameObject)
     {
```

```
            Ray heightPos = new Ray(this.transform.position, -hits[i].normal);

            RaycastHit[] hitsFindHeight;
            hitsFindHeight = Physics.RaycastAll(heightPos, Mathf.Infinity);

            for (int j = 0; j < hitsFindHeight.Length; j++)
            {
                if (Transparent_Plate.gameObject == hitsFindHeight[j].collider.gameObject)
                {
                    vScaled_H = hitsFindHeight[j].point;
                    vCH = vCanvasCenter - vScaled_H;
                    scale_dist = Vector3.Dot(vCH, vn_Down);

                    Debug.Log(scale_dist);

                    vScaled_H -= (scaleHeight - 1f) * scale_dist * vn_Down;
                    _vHeight_Loc = Vector3.SmoothDamp(_vHeight_Loc, vScaled_H, ref vVelHeight,
                     smoothTime);
                    _vGaze_Loc = Vector3.SmoothDamp(_vGaze_Loc, hits[i].point, ref vVelGaze, smoothTime);
                    return;//all done :)
                }
            }
        }
    }
```

SCALE HEIGHT OFFSET IN RELATION

TO CENTER OF CANVAS



SOLVED TASKBAR AUTO HIDE

*TASK BAR AUTO HIDE not working – restart windows explorer in taskmanager

Rigidbody.AddRelativeForce

public void **AddRelativeForce**(Vector3 **force**, ForceMode **mode** = ForceMode.Force);

Parameters

force	Force vector in local coordinates.

Description

Adds a force to the rigidbody relative to its coordinate system.
Force can be applied only to an active rigidbody. If a GameObject is inactive, AddRelativeForce has no effect.

Wakes up the Rigidbody by default. If the force size is zero then the Rigidbody will not be woken up.

See Also: AddForce, AddForceAtPosition, AddRelativeTorque.

```
using UnityEngine;
using System.Collections;

// Add a thrust force to push an object in its current forward
// direction (to simulate a rocket motor, say).
public class ExampleClass : MonoBehaviour {
   public float thrust;
   public Rigidbody rb;
   void Start() {
      rb = GetComponent<Rigidbody>();
   }
   void FixedUpdate() {
      rb.AddRelativeForce(Vector3.forward * thrust);
   }
}
```

Attempts to write vertices on the fly

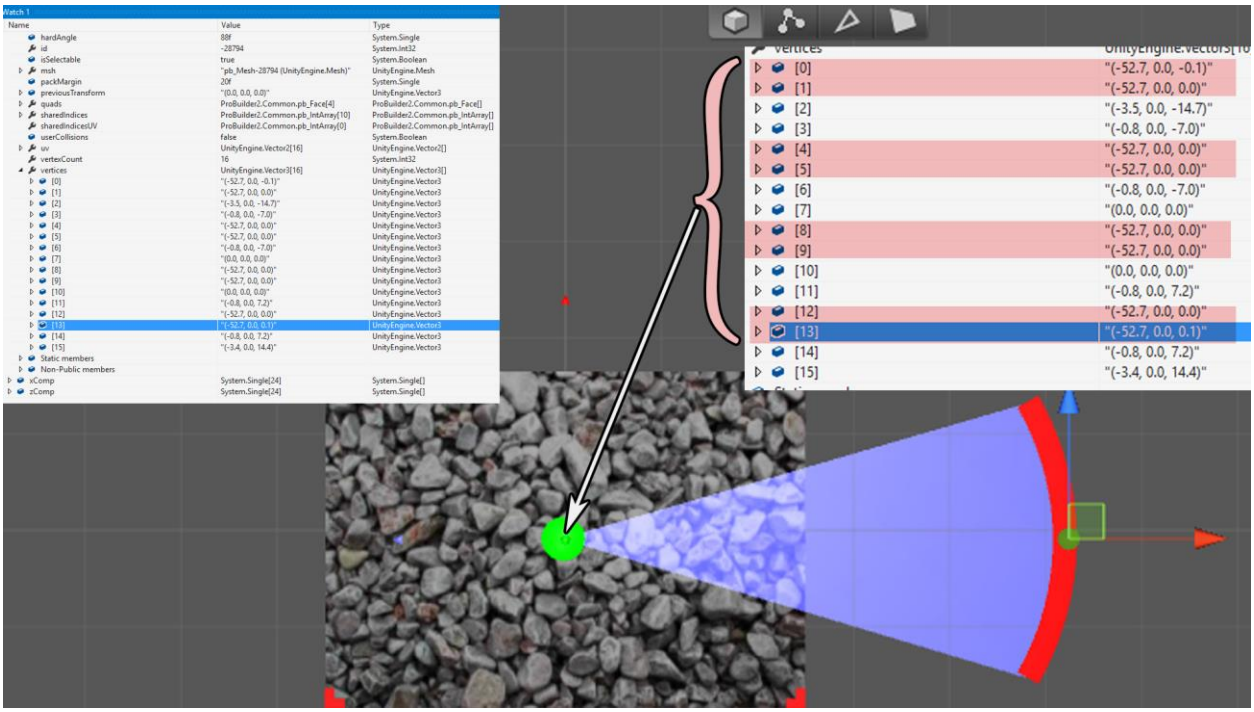
```
using UnityEngine;
using System.Collections;
using ProBuilder2.Common;   /// for pb_Object and pb_Face
using ProBuilder2.Math;

public class StetchPlane : MonoBehaviour
{
    pb_Object pb;
    pb_Face face;

    public struct IndicesPB
    {
        public int[] indices;
        public Vector3[] Verts;
    }

    IndicesPB[] idPB = new IndicesPB[4];
```

```
Vector3 nrm;

Vector3 vVert_0;
Vector3[] vVerts;

void Start()
{
    pb = GetComponent<pb_Object>();

    vVert_0 = pb.msh.vertices[0];

    pb_Face[] face = pb.faces;

    nrm = pb_Math.Normal(pb, face[0]);

    // only grab the unique indices - so (0,1,2,1,3,2) becomes (0,1,2,3)
    for(int i = 0; i< face.Length; i++)
    {
        idPB[i].indices = face[i].distinctIndices;
        for(int k = 0; k<4;k++)
        {
            idPB[i].Verts[k] = pb.msh.vertices[idPB[i].indices[k]];
        }
    }

    // store the origin points of each vertex we'll be moving
    //vertexOrigins = pbUtil.ValuesWithIndices(pb.vertices, indices);
    //vertexCenter = pb_Math.BoundsCenter(vertexOrigins);
}

float rotation = 0f;

//void Update()
//{
//    rotation = Mathf.Sin(Time.time) * 90f;

//    Quaternion faceRotation = Quaternion.Euler(nrm * rotation);

//    for (int i = 0; i < indices.Length; i++)
//    {
//        Vector3 v = vertexOrigins[i] - vertexCenter;

//        v = faceRotation * v;

//        v += vertexCenter;

//        // Using SetSharedVertexPosition guarantees that all vertices
//        // that are shared among the indices are also rotated.  It
//        // also applies the pb.vertices array to the mesh.
//        pb.SetSharedVertexPosition(indices[i], v);
//    }
//}


void Update()
{
    vVert_0 -= Vector3.right * Time.deltaTime * 5f;
```

```
        for (int i = 0; i < 4; i++)
        {
            pb.SetSharedVertexPosition(idPB[i].indices[0], vVert_0);
            pb.SetSharedVertexPosition(idPB[i].indices[1], vVert_0);
            pb.SetSharedVertexPosition(idPB[i].indices[2], vVert_0);
            pb.SetSharedVertexPosition(idPB[i].indices[3], vVert_0);

        }
    }
}
```

```
using UnityEngine;
using System.Collections;
using ProBuilder2.Common;     /// for pb_Object and pb_Face
using ProBuilder2.Math;

public class RotateFace : MonoBehaviour
{
        pb_Object pb;
        pb_Face face;

        int[] indices;
        Vector3[] vertexOrigins;
        Vector3 vertexCenter;
        Vector3 nrm;

        void Start()
        {
                pb = GetComponent<pb_Object>();
                pb_Face face = pb.faces[0];

                nrm = pb_Math.Normal(pb, face);

                // only grab the unique indices - so (0,1,2,1,3,2) becomes (0,1,2,3)
                indices = face.distinctIndices;

                // store the origin points of each vertex we'll be moving
                vertexOrigins = pbUtil.ValuesWithIndices(pb.vertices, indices);
                vertexCenter = pb_Math.BoundsCenter(vertexOrigins);
        }

        float rotation = 0f;

        void Update()
        {
                rotation = Mathf.Sin(Time.time) * 90f;

                Quaternion faceRotation = Quaternion.Euler( nrm * rotation );

                for(int i = 0; i < indices.Length; i++)
                {
                        Vector3 v = vertexOrigins[i] - vertexCenter;
```

```
                    v = faceRotation * v;

                    v += vertexCenter;

                    // Using SetSharedVertexPosition guarantees that all vertices
                    // that are shared among the indices are also rotated.  It
                    // also applies the pb.vertices array to the mesh.
                    pb.SetSharedVertexPosition(indices[i], v);
                }
            }
}
```



Mathf.SmoothDampAngle

public static float **SmoothDampAngle**(float **current**, float **target**, ref float **currentVelocity**, float **smoothTime**, float**maxSpeed** = Mathf.Infinity, float **deltaTime** = Time.deltaTime);

public static float **SmoothDampAngle**(float **current**, float **target**, ref float **currentVelocity**, float **smoothTime**, float**maxSpeed** = Mathf.Infinity, float **deltaTime** = Time.deltaTime);

public static float **SmoothDampAngle**(float **current**, float **target**, ref float **currentVelocity**, float **smoothTime**, float**maxSpeed** = Mathf.Infinity, float **deltaTime** = Time.deltaTime);

Parameters

current	The current position.
target	The position we are trying to reach.
currentVelocity	The current velocity, this value is modified by the function every time you call it.
smoothTime	Approximately the time it will take to reach the target. A smaller value will reach the target faster.
maxSpeed	Optionally allows you to clamp the maximum speed.
deltaTime	The time since the last call to this function. By default Time.deltaTime.

Quaternion.LookRotation

public static Quaternion **LookRotation**(Vector3 **forward**, Vector3 **upwards** = Vector3.up);

public static Quaternion **LookRotation**(Vector3 **forward**, Vector3 **upwards** = Vector3.up);

Parameters

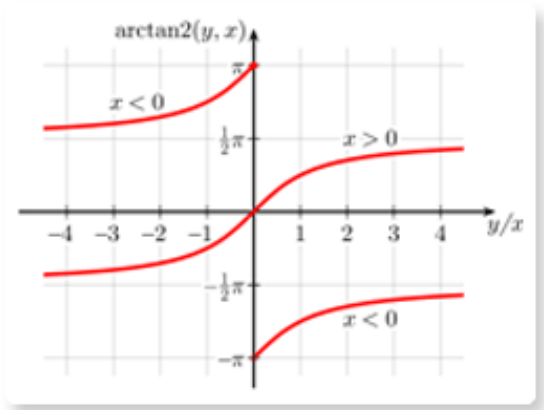
forward	The direction to look in.
upwards	The vector that defines in which direction up is.

Description

Creates a rotation with the specified forward and upwards directions.

Returns the computed quaternion. If used to orient a Transform, the Z axis will be aligned with forward/ and the Y axis withupwards if these vectors are orthogonal. Logs an error if the forward direction is zero.

ROTATION of PADDLE



Vector3.SmoothDamp

public static Vector3 SmoothDamp(Vector3 current, Vector3 target, ref Vector3 currentVelocity, float smoothTime, floatmaxSpeed = Mathf.Infinity, float deltaTime = Time.deltaTime);

public static Vector3 SmoothDamp(Vector3 current, Vector3 target, ref Vector3 currentVelocity, float smoothTime, floatmaxSpeed = Mathf.Infinity, float deltaTime = Time.deltaTime);

public static Vector3 SmoothDamp(Vector3 current, Vector3 target, ref Vector3 currentVelocity, float smoothTime, floatmaxSpeed = Mathf.Infinity, float deltaTime = Time.deltaTime);

Parameters

current	The current position.
target	The position we are trying to reach.
currentVelocity	The current velocity, this value is modified by the function every time you call it.
smoothTime	Approximately the time it will take to reach the target. A smaller value will reach the target faster.
maxSpeed	Optionally allows you to clamp the maximum speed.
deltaTime	The time since the last call to this function. By default Time.deltaTime.

Description

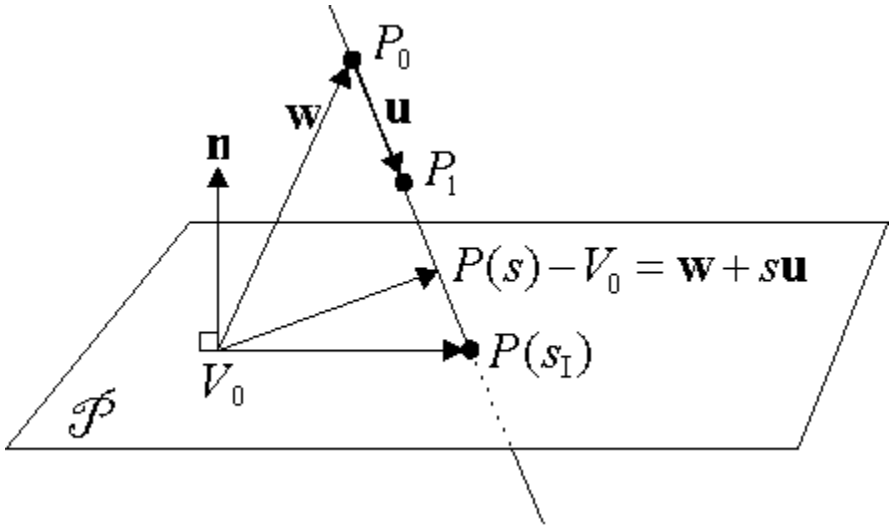
Gradually changes a vector towards a desired goal over time.

The vector is smoothed by some spring-damper like function, which will never overshoot. The most common use is for smoothing a follow camera.

Line-Plane Intersection

In 3D, a line L is either parallel to a plane P or intersects it in a single point. Let L be given by the parametric equation: $P(s) = P_0 + s(P_1 - P_0) = P_0 + s\mathbf{u}$, and the plane P be given by a point V_0 on it and a normal vector $\mathbf{n} = (a,b,c)$. We first check if L is parallel to P by testing if $\mathbf{n} \cdot \mathbf{u} = 0$, which means that the line direction vector u is perpendicular to the plane normal n. If this is true, then L and P are parallel and either never intersect or else L lies totally in the plane P . Disjointness or coincidence can be determined by testing whether any one specific point of L, say P_0, is contained in P , that is whether it satisfies the implicit line equation: $\mathbf{n} \cdot (P_0 - V_0) = 0$.

If the line and plane are not parallel, then L and P intersect in a unique point $P(s_I)$ which is computed using a method similar to the one for the intersection of two lines in 2D. Consider the diagram:



At the intersect point, the vector $P(s)-V_0=\mathbf{w}+s\mathbf{u}$ is perpendicular to n, where $\mathbf{w}=P_0-V_0$. This is equivalent to the dot product condition: $\mathbf{n}\bullet(\mathbf{w}+s\mathbf{u})=0$. Solving we get:

$$s_I=\frac{-\mathbf{n}\cdot\mathbf{w}}{\mathbf{n}\cdot\mathbf{u}}=\frac{\mathbf{n}\cdot(\mathrm{V}_0-\mathrm{P}_0)}{\mathbf{n}\cdot(\mathrm{P}_1-\mathrm{P}_0)}=\frac{-(ax_0+by_0+cz_0+d)}{\mathbf{n}\cdot\mathbf{u}}$$

If the line L is a finite segment from P_0 to P_1, then one just has to check that $0\le s_I\le 1$ to verify that there is an intersection between the segment and the plane. For a positive ray, there is an intersection with the plane when $s_I\ge 0$.

RaycastHit

struct in UnityEngine

Description

Structure used to get information back from a raycast.

See Also: Physics.Raycast, Physics.Linecast, Physics.RaycastAll.

Variables

barycentricCoordinate	The barycentric coordinate of the triangle we hit.
collider	The Collider that was hit.
distance	The distance from the ray's origin to the impact point.
lightmapCoord	The uv lightmap coordinate at the impact point.
normal	The normal of the surface the ray hit.
point	The impact point in world space where the ray hit the collider.

rigidbody	The Rigidbody of the collider that was hit. If the collider is not attached to a rigidbody then it is null.
textureCoord	The uv texture coordinate at the impact point.
textureCoord2	The secondary uv texture coordinate at the impact point.
transform	The Transform of the rigidbody or collider that was hit.
triangleIndex	The index of the triangle that was hit.

Ray

struct in UnityEngine

Description

Representation of rays.

A ray is an infinite line starting at origin and going in some direction.

Variables

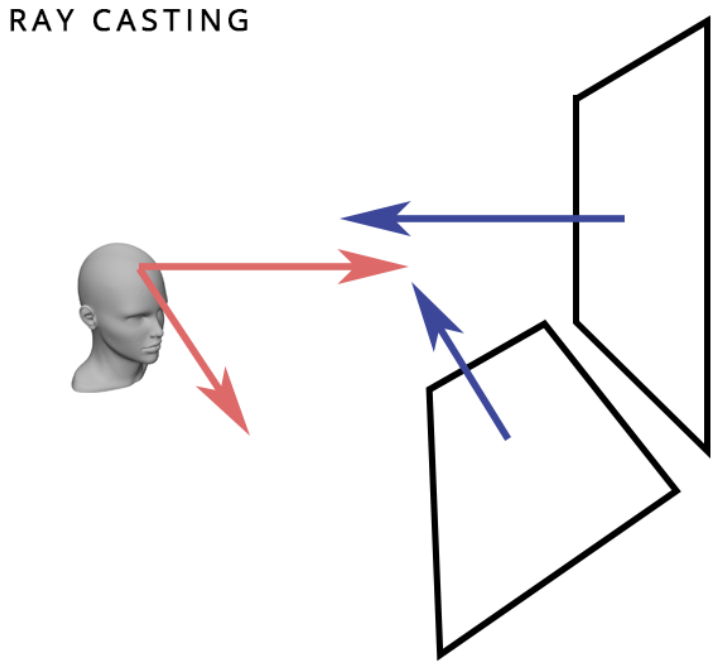
direction	The direction of the ray.
origin	The origin point of the ray.

Constructors

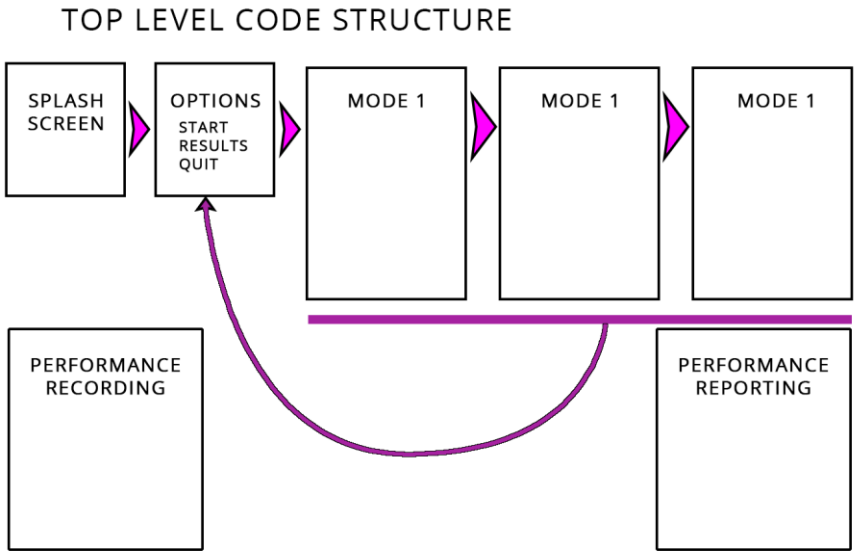
Ray	Creates a ray starting at origin along direction.

Public Functions

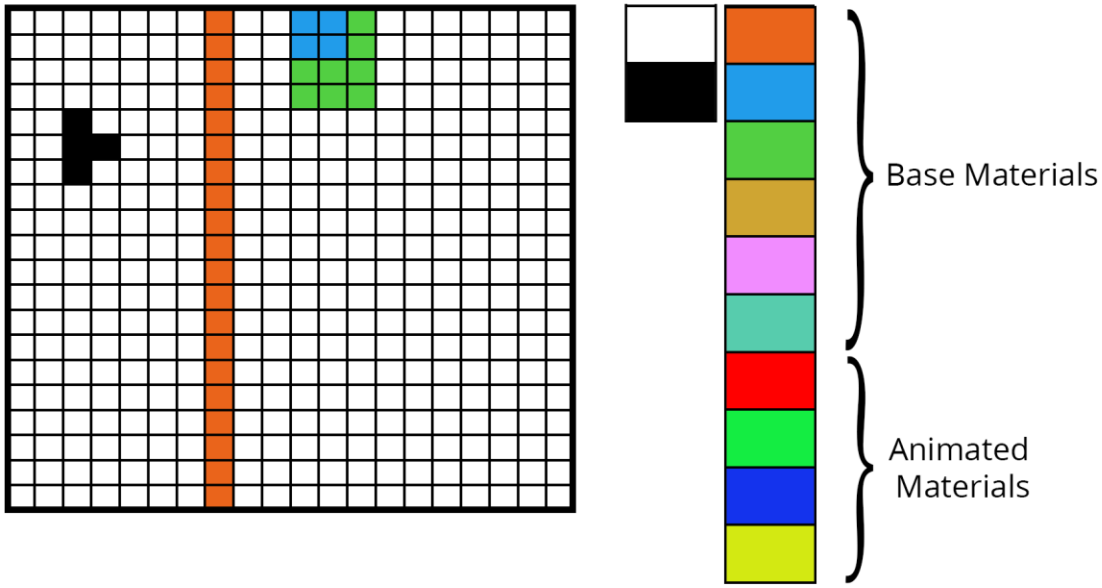
GetPoint	Returns a point at distance units along the ray.
ToString	Returns a nicely formatted string for this ray.



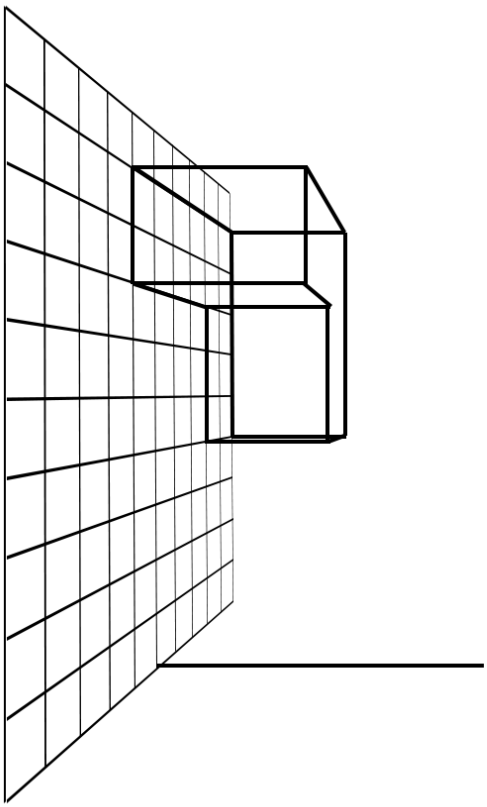
https://developer.vuforia.com/library/articles/Solution/Integrating-Gear-VR-and-the-AR-VR-Sample-in-Unity-5-3-and-above



MATERIAL ASSIGNMENT AT RUNTIME



1/22/16



1/21/16

How to ScreenCap Video on Android/GearVR

```
D:\ADB>adb shell screenrecord /sdcard/demo.mp4
```

`vidrec.bat`

https://developer.android.com/tools/help/shell.html#screencap

File can be found in Local Storage > Device Storage - on device

And in SAMSUNG-SM-N910A > Phone

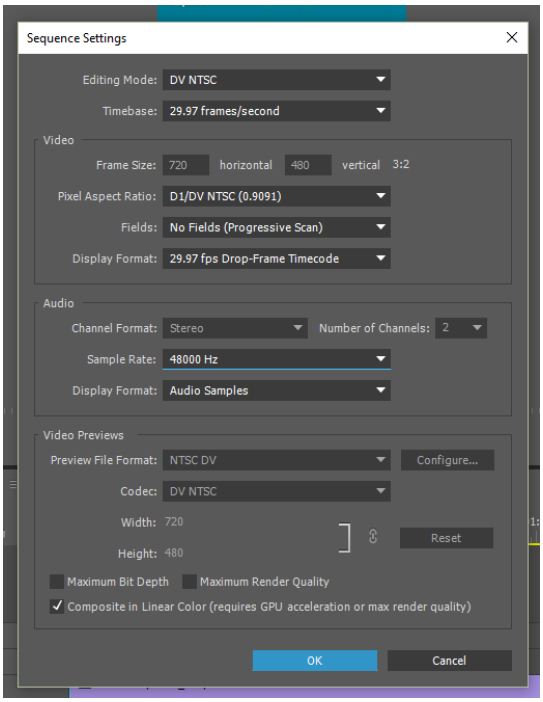
Copy to Video Work

Create New Project in Premier

Import Clip

Drop Clip on Time Line

Goto Sequence > Sequence Settings



Goto Effects search for transform – choose Crop and place on clip in time line.

Click the Motion Selection to get a sizable focus area.

In Effect Controls – scale the video so a single gear VR window is positioned on the output canvas.

File > Exoprt Media

Format H.264 preset HD 29.97

Best Youtube Export Settings With Adobe Premiere Pro CS6
https://youtu.be/4FbTsk_3QD4

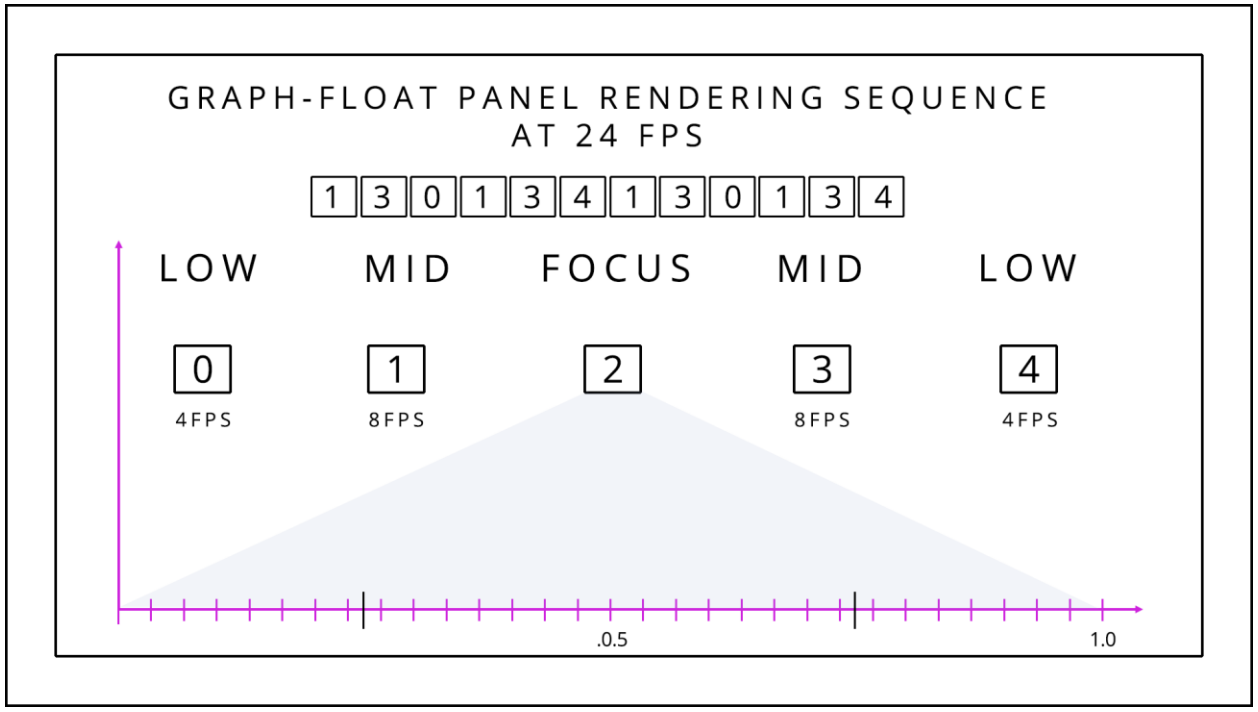
1/20/16

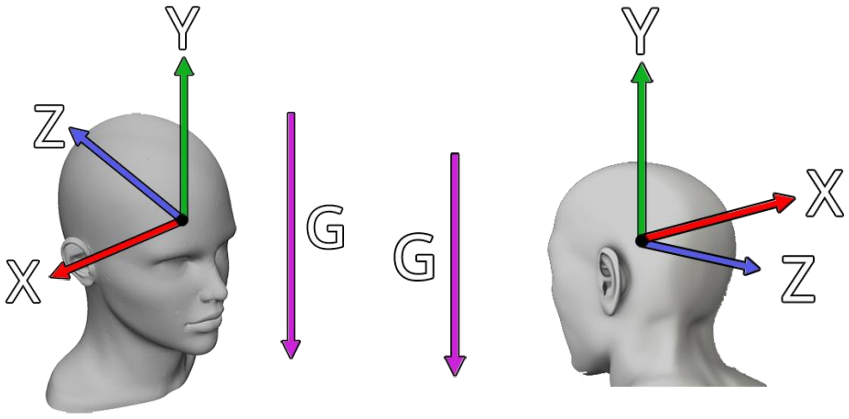
Exercise and Memory Game

1/17/2016

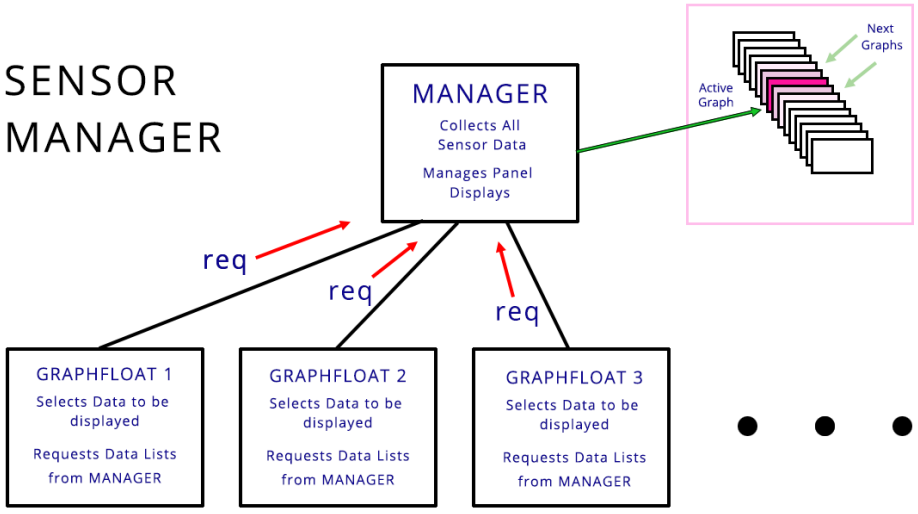
Vectrosity proves to be too much of a resource hog for what I was designing. Particularly with rotations. Efficiencies might include – reducing the points per trace, i.e. skipping elements/stride. Will continue with one screen rendering at a time.

Work on this can be found in: D:\0_AR_Dev\GearVR_Vuforia_Oculus_3

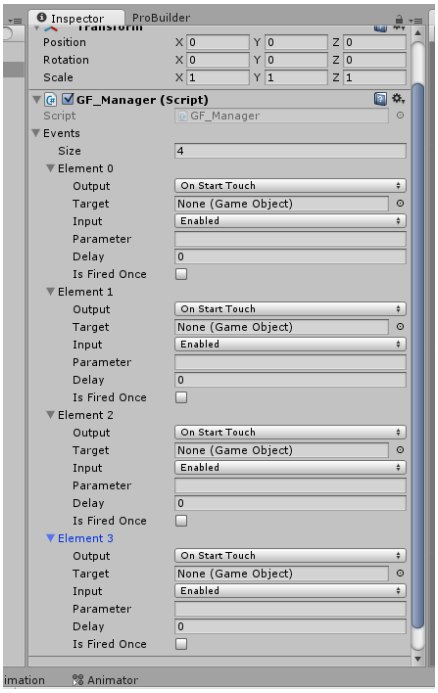




Accelerometer Coordinate System



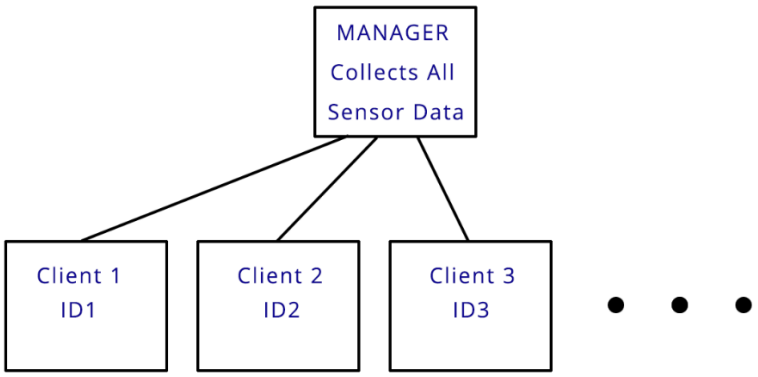

```
[System.Serializable]
public class Outputs
{
    public enum Output : byte { OnStartTouch, OnStartTouchAll, OnEndTouch, OnEndTouchAll, OnTrigger }
    public Output output;
    public GameObject Target;
    public enum Inputs : byte { Enabled, Disabled, SetParent, PlayAudio, EnableTrigger, DisableTrigger, SetCheckPoint }
    public Inputs input;
    public string parameter;
    public float delay;
    public bool isFiredOnce;
}
public Outputs[] events;
```



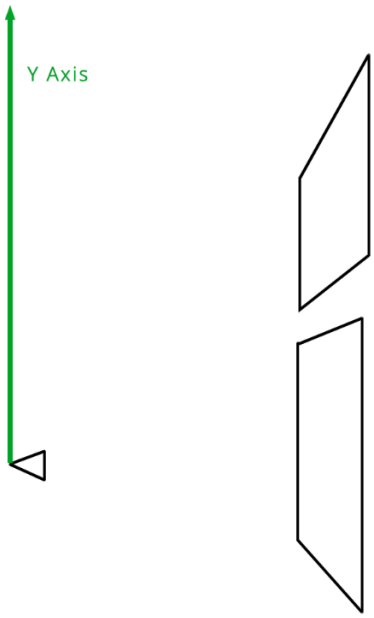
Sensor Manager:

Collects all sensor data, stores and distributes display information. Monitors Active displays.

Positions Displays through User Interaction. Collect and Store Data.



Window to track rotations around the head y axis and slide vertically parallel to the y axis.



Screens:

Input.acceleration

Input.acceleration	**3**	**(Vec3)**

Input.gyro

Gyroscope.attitude	**4**	**(Quat)**
Gyroscope.gravity	**3**	**(Vec3)**
Gyroscope.rotationRate	**3**	**(Vec3)**
Gyroscope.rotationRateUnbiased	**3**	**(Vec3)**
Gyroscope.userAcceleration	**3**	**(Vec3)**

Input.compass

Compass.headingAccuracy	**1**	**(float) ~**
Compass.magneticHeading	**1**	**(float)**
Compass.rawVector	**3**	**(Vec3)**
Compass.trueHeading	**1**	**(float)**

26 Screens

GearVR Input

The Gear Vr touchpad is mapped up like a mouse in unity. You can use Input.GetAxis("Mouse X") and Input.GetAxis("Mouse Y") to read the motion of the finger on the touchpad. Input.GetMouseButton(0) will detect if the finger touches the touchpad or not.

With this data, it's quite simple to build your own implementation of simple gesture detection.

Clean Up Presentation – Fluttering - Design

Icon for Each Sensor

Pivot/Move Between Screens – Damping - Intuitive

1/14/2016

1. Tick Marks and Time – Secs – with 0.5 hashes

2. Current value in box

3. Max value on Y axis

4. Red x
Green y
Blue z

5. Three screens Positioned to monitor comfortably

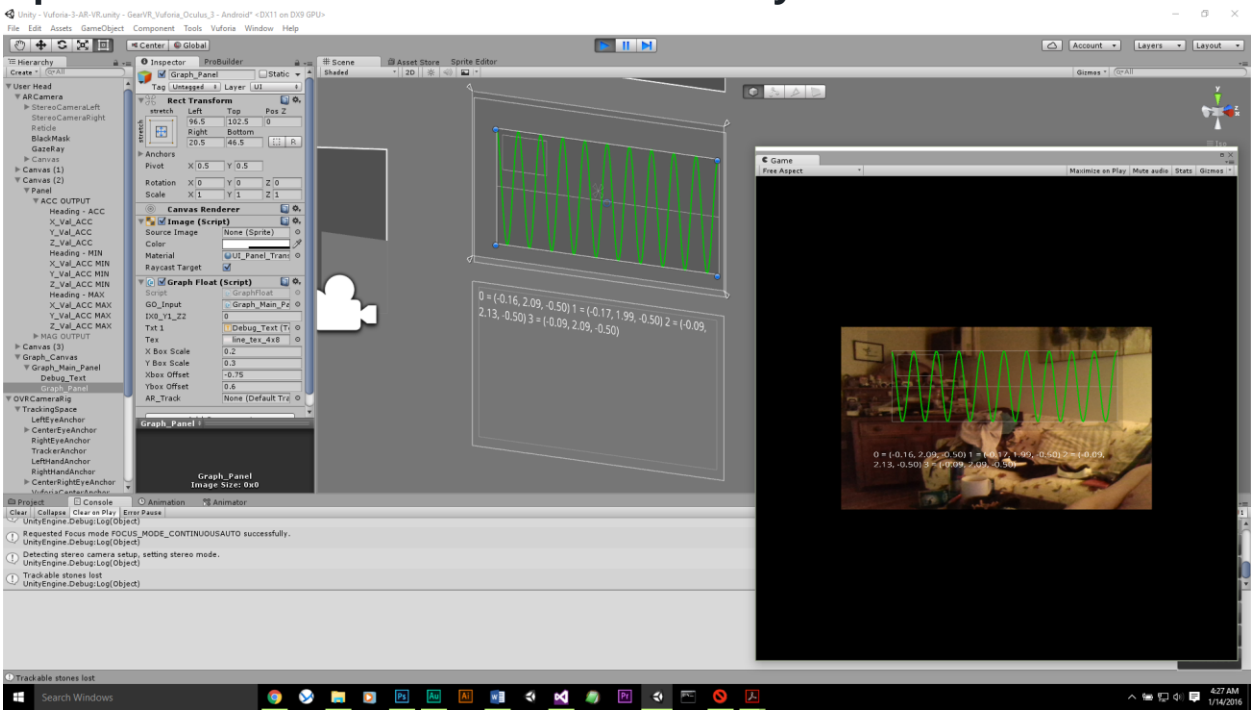
```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
using Vectrosity;
using System.Collections.Generic;
using Vuforia;

public class GraphFloat : MonoBehaviour
{
    public Sensor_Input GO_Input;
    public int iX0_Y1_Z2;
    public Text Txt1;
    public Texture tex;
    public float xBoxScale = 0.2f;
    public float yBoxScale = 0.3f;
    public float xboxOffset = 0.28f;
    public float yboxOffset = 0.26f;

    VectorLine Border_Points;
    VectorLine Box_Outline;
    VectorLine GraphAxis;

    VectorLine GraphIDL;

    List<InputData> InputDataList = new List<InputData>();

    public DefaultTrackableEventHandler AR_Track;
    private bool bTracking = false;
    private Vector3[] vCorners;
    private Vector3[] vBoxOutline;
    private Vector3 vCenter;
    private float Width;
    private float Height;
    private float pWidth; //past width
    private float pHeight;//past height
    private Vector3 vx;
    private Vector3 vy;
    private float fElapsedTime;

    void Start()
    {
        vCorners = new Vector3[4];
        vBoxOutline = new Vector3[4];
        vCenter = new Vector3();
        Vector3 vx = new Vector3();
        Vector3 vy = new Vector3();
```

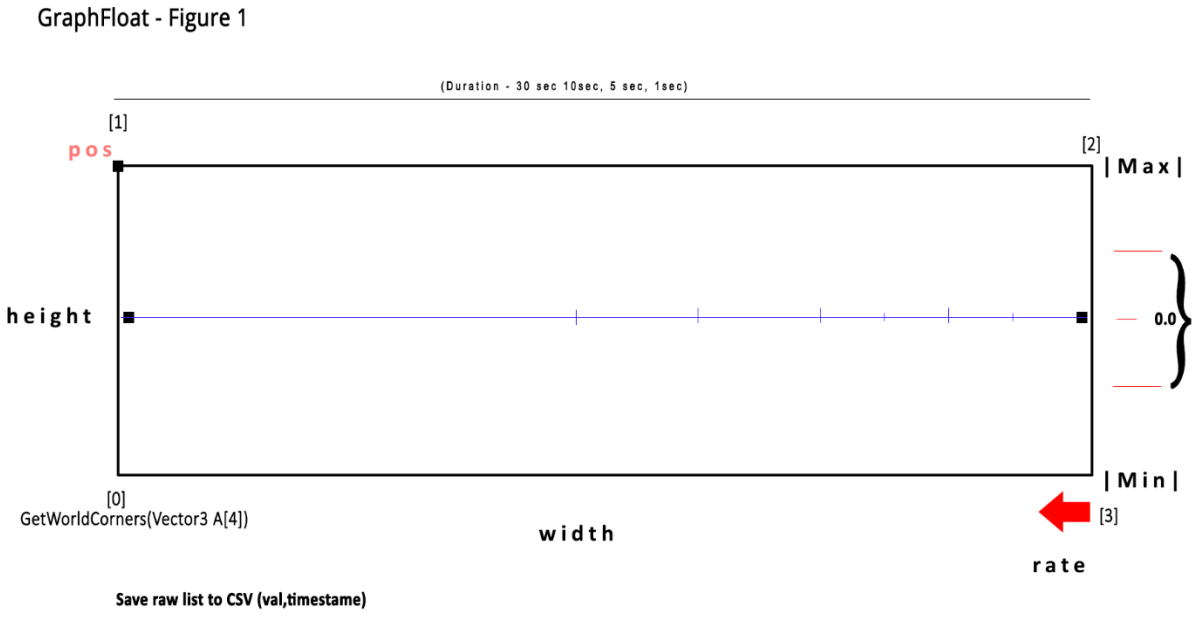
```
        fElapsedTime = 0.0f;

        vx = vCorners[3] - vCorners[0];
        vy = vCorners[1] - vCorners[0];
        pWidth = vx.magnitude;
        pHeight = vy.magnitude;

        vx.Normalize();
        vy.Normalize();

        GetComponent<RectTransform>().GetWorldCorners(vCorners);

        //Panel Border
        Border_Points = new VectorLine("3DLine", new List<Vector3>(), tex, 1.0f, LineType.Continuous);
        Border_Points.points3.Add(vCorners[0]);
        Border_Points.points3.Add(vCorners[1]);
        Border_Points.points3.Add(vCorners[2]);
        Border_Points.points3.Add(vCorners[3]);
        Border_Points.points3.Add(vCorners[0]);

        vCenter = Vector3.zero;

        for(int i = 0; i<4; i++)
        {
            vCenter += vCorners[i];
        }
        vCenter /= 4.0f;

        Border_Points.SetColor(Color.white);
        Border_Points.Draw3DAuto();

        //Axis
        GraphAxis = new VectorLine("Axis", new List<Vector3>(), tex, 1.0f, LineType.Discrete);
        GraphAxis.points3.Add(vCenter - vx * pWidth/2.0f);
        GraphAxis.points3.Add(vCenter + vx * pWidth/2.0f);
        GraphAxis.Draw3DAuto();

        //Graph Input Data List
        GraphIDL = new VectorLine("GraphIDL", new List<Vector3>(), 2.0f, LineType.Continuous, Joins.Fill);
        GraphIDL.color = Color.blue;
        GraphIDL.Draw3DAuto();

        //Box Outline
        Box_Outline = new VectorLine("GraphOutline", new List<Vector3>(), tex, 1.0f, LineType.Continuous);

        Vector3 vBoxCenter = new Vector3();

        vBoxCenter = vCenter + vx*xboxOffset*pWidth/2.0f + vy*yboxOffset*pHeight/2.0f;

        for (int i = 0; i<4;i++)
        {
            vBoxOutline[i] = vBoxCenter + xBoxScale * Vector3.Dot(vCorners[i]- vCenter, vx) * vx
                + yBoxScale * Vector3.Dot(vCorners[i] - vCenter, vy) * vy;
            Box_Outline.points3.Add(vBoxOutline[i]);
        }
        Box_Outline.points3.Add(vBoxOutline[0]);
        Box_Outline.Draw3DAuto();
    }

    private void AddInputToList(float fIn, float fTime)
    {
        InputData id = new InputData();
        id.fVal = fIn;
        id.fTime = fTime;
        InputDataList.Add(id);
    }

    private float FindMax()
    {
        float maxVal = 0.0f;
        foreach (InputData id in InputDataList)
        {
            float fAbs = Mathf.Abs(id.fVal);
            if (fAbs > maxVal)
            {
                maxVal = fAbs;
            }
```

```
        }
        return maxVal;
    }


    private void DrawList()
    {
        float max = FindMax();

        GraphIDL.points3.Clear();
        GraphIDL.color = Color.green;

        Vector3 vOrigin = vCorners[2] - vy * Height / 2.0f;
        float yC = (Height / 2.0f) / max;
        float xC = Width / 10.0f;

        int cnt = InputDataList.Count;

        if (cnt > 2)
        {
            float fBaseTime = InputDataList[cnt - 1].fTime;

            float fv;
            float delT;

            Vector3 vtimeX = new Vector3();
            Vector3 vvalY = new Vector3();

            for (int i = cnt - 1; i > 0; i--)
            {
                fv = InputDataList[i].fVal;
                delT = fBaseTime - InputDataList[i].fTime;

                if (delT > 10.0f) break;

                vtimeX = vx * xC * delT;
                vvalY = vy * yC * fv;

                GraphIDL.points3.Add(vOrigin - vtimeX + vvalY);
            }
        }
    }

      // Update is called once per frame
      void Update ()
    {
        //float fInput = GO_Input.GetVal(iX0_Y1_Z2);
        fElapsedTime += Time.deltaTime;
        float fInput = Mathf.Sin(fElapsedTime*2.0f*Mathf.PI);
        AddInputToList(fInput, fElapsedTime);

        GetComponent<RectTransform>().GetWorldCorners(vCorners);
        vx = vCorners[3] - vCorners[0];
        vy = vCorners[1] - vCorners[0];
        Width = vx.magnitude;
        Height = vy.magnitude;

        vx.Normalize();
        vy.Normalize();

        DrawList();

        if (pHeight != Height || pWidth != Width)
        {
            vx.Normalize();
            vy.Normalize();
            //Panel Border
            Border_Points.points3[0] = vCorners[0];
            Border_Points.points3[1] = vCorners[1];
            Border_Points.points3[2] = vCorners[2];
            Border_Points.points3[3] = vCorners[3];
            Border_Points.points3[4] = vCorners[0];

            vCenter = Vector3.zero;
            for (int i = 0; i < 4; i++)
            {
                vCenter += vCorners[i];
```

```
            }
            vCenter /= 4.0f;

            //Graph Axis
            GraphAxis.points3[0] = vCenter - vx * Width/2.0f;
            GraphAxis.points3[1] = vCenter + vx * Width/2.0f;

            //Box Outline
            Vector3 vBoxCenter = new Vector3();

            vBoxCenter = vCenter + vx * xboxOffset * Width/2.0f + vy * yboxOffset * Height/2.0f;

            for (int i = 0; i < 4; i++)
            {
                vBoxOutline[i] = vBoxCenter + xBoxScale * Vector3.Dot(vCorners[i] - vCenter, vx) * vx
                    + yBoxScale * Vector3.Dot(vCorners[i] - vCenter, vy) * vy;
                Box_Outline.points3[i] = vBoxOutline[i];
            }
            Box_Outline.points3[4] = vBoxOutline[0];

            pWidth = Width;
            pHeight = Height;
        }
        Txt1.text = "0 = " + vBoxOutline[0].ToString("F2") + " 1 = " + vCorners[0].ToString("F2") + " 2 = " + vBoxOutline[2].ToString("F2") +
" 3 = " + vBoxOutline[3].ToString("F2");
    }
}

//float x = GetComponent<RectTransform>().rect.x;
//float y = GetComponent<RectTransform>().rect.y;
//float xMax = GetComponent<RectTransform>().rect.xMax;
//float yMax = GetComponent<RectTransform>().rect.yMax;
//float xMin = GetComponent<RectTransform>().rect.xMin;
//float yMin = GetComponent<RectTransform>().rect.yMin;
//float width = GetComponent<RectTransform>().rect.width;
//float height = GetComponent<RectTransform>().rect.height;

public class InputData
{
    private float _fVal;
    private float _fTime;

    public float fVal
    {
        set
        {
            _fVal = value;
        }
        get
        {
            return _fVal;
        }
    }

    public float fTime
    {
        set
        {
            _fTime = value;
        }
        get
        {
            return _fTime;
        }
    }
}
```

Option to Record Data for Review and Playback





RectTransform.GetWorldCorners

public void **GetWorldCorners**(Vector3[] **fourCornersArray**);

Parameters

fourCornersArray	Array that corners should be filled into.

Description

Get the corners of the calculated rectangle in world space.

Rect

struct in UnityEngine

Description

A 2D Rectangle defined by X and Y position, width and height.

Unity uses a number of 2D coordinate spaces, most of which define X as increasing to the right, and Y increasing upwards. The one exception is in the GUI and GUILayout classes, where Y increases downwards.

The following examples are illustrated in GUI space, where (0,0) represents the top-left corner and Y increases downwards.

Rectangles can be specified in two different ways. The first is with an x and y position and a width and height:

The other way is with the X and Y coordinates of each of its edges. These are called xMin, xMax, yMin and yMax:

Note that although x and y have the same values as xMin and yMin, they behave differently when you set them.
Setting x or ychanges the position of the rectangle, but preserves its size:

Setting any of xMin, xMax, yMin and yMax will resize the rectangle, but preserve the position of the opposite edge:

See Also: GUI Scripting Guide, Camera.rect, Camera.pixelRect.

Variables

center	The position of the center of the rectangle.

height	The height of the rectangle, measured from the Y position.
max	The position of the maximum corner of the rectangle.
min	The position of the minimum corner of the rectangle.
position	The X and Y position of the rectangle.
size	The width and height of the rectangle.
width	The width of the rectangle, measured from the X position.
x	The X coordinate of the rectangle.
xMax	The maximum X coordinate of the rectangle.
xMin	The minimum X coordinate of the rectangle.
y	The Y coordinate of the rectangle.
yMax	The maximum Y coordinate of the rectangle.
yMin	The minimum Y coordinate of the rectangle.

Constructors

Rect	Creates a new rectangle.

Public Functions

Contains	Returns true if the x and y components of point is a point inside this rectangle. If allowInverse is present and true, the width and height of the Rect are allowed to take negative values (ie, the min value is greater than the max), and the test will still work.
Overlaps	Returns true if the other rectangle overlaps this one. If allowInverse is present and true, the widths and heights of the Rects are allowed to take negative values (ie, the min value is greater than the max), and the test will still work.
Set	Set components of an existing Rect.
ToString	Returns a nicely formatted string for this Rect.

Static Functions

MinMaxRect	Creates a rectangle from min/max coordinate values.
NormalizedToPoint	Returns a point inside a rectangle, given normalized coordinates.
PointToNormalized	Returns the normalized coordinates cooresponding the the point.

Operators

operator !=	Returns true if the rectangles are different.
operator ==	Returns true if the rectangles are the same.

RectTransform

Description

123

Position, size, anchor and pivot information for a rectangle.

RectTransforms are used for GUI but can also be used for other things. It's used to store and manipulate the position, size, and anchoring of a rectangle and supports various forms of scaling based on a parent RectTransform.

Variables

anchoredPosition	The position of the pivot of this RectTransform relative to the anchor reference point.
anchoredPosition3D	The 3D position of the pivot of this RectTransform relative to the anchor reference point.
anchorMax	The normalized position in the parent RectTransform that the upper right corner is anchored to.
anchorMin	The normalized position in the parent RectTransform that the lower left corner is anchored to.
offsetMax	The offset of the upper right corner of the rectangle relative to the upper right anchor.
offsetMin	The offset of the lower left corner of the rectangle relative to the lower left anchor.
pivot	The normalized position in this RectTransform that it rotates around.
rect	The calculated rectangle in the local space of the Transform.
sizeDelta	The size of this RectTransform relative to the distances between the anchors.

Public Functions

GetLocalCorners	Get the corners of the calculated rectangle in the local space of its Transform.

GetWorldCorners	Get the corners of the calculated rectangle in world space.
SetInsetAndSizeFromParentEdge	Set the distance of this rectangle relative to a specified edge of the parent rectangle, while also setting its size.
SetSizeWithCurrentAnchors	Makes the RectTransform calculated rect be a given size on the specified axis.

```
public static class RectTransformExtensions
{
    public static void SetDefaultScale(this RectTransform trans) {
        trans.localScale = new Vector3(1, 1, 1);
    }
    public static void SetPivotAndAnchors(this RectTransform trans, Vector2 aVec) {
        trans.pivot = aVec;
        trans.anchorMin = aVec;
        trans.anchorMax = aVec;
    }

    public static Vector2 GetSize(this RectTransform trans) {
        return trans.rect.size;
    }
    public static float GetWidth(this RectTransform trans) {
        return trans.rect.width;
    }
    public static float GetHeight(this RectTransform trans) {
        return trans.rect.height;
    }

    public static void SetPositionOfPivot(this RectTransform trans, Vector2 newPos) {
        trans.localPosition = new Vector3(newPos.x, newPos.y, trans.localPosition.z);
    }

    public static void SetLeftBottomPosition(this RectTransform trans, Vector2 newPos) {
        trans.localPosition = new Vector3(newPos.x + (trans.pivot.x * trans.rect.width), newPos.y + (trans.pivot.y *
trans.rect.height), trans.localPosition.z);
    }
    public static void SetLeftTopPosition(this RectTransform trans, Vector2 newPos) {
        trans.localPosition = new Vector3(newPos.x + (trans.pivot.x * trans.rect.width), newPos.y - ((1f - trans.pivot.y) *
trans.rect.height), trans.localPosition.z);
    }
    public static void SetRightBottomPosition(this RectTransform trans, Vector2 newPos) {
        trans.localPosition = new Vector3(newPos.x - ((1f - trans.pivot.x) * trans.rect.width), newPos.y + (trans.pivot.y *
trans.rect.height), trans.localPosition.z);
    }
    public static void SetRightTopPosition(this RectTransform trans, Vector2 newPos) {
        trans.localPosition = new Vector3(newPos.x - ((1f - trans.pivot.x) * trans.rect.width), newPos.y - ((1f -
trans.pivot.y) * trans.rect.height), trans.localPosition.z);
    }

    public static void SetSize(this RectTransform trans, Vector2 newSize) {
        Vector2 oldSize = trans.rect.size;
```

```
        Vector2 deltaSize = newSize - oldSize;
        trans.offsetMin = trans.offsetMin - new Vector2(deltaSize.x * trans.pivot.x, deltaSize.y * trans.pivot.y);
        trans.offsetMax = trans.offsetMax + new Vector2(deltaSize.x * (1f - trans.pivot.x), deltaSize.y * (1f -
trans.pivot.y));
    }
    public static void SetWidth(this RectTransform trans, float newSize) {
        SetSize(trans, new Vector2(newSize, trans.rect.size.y));
    }
    public static void SetHeight(this RectTransform trans, float newSize) {
        SetSize(trans, new Vector2(trans.rect.size.x, newSize));
    }
}
```

Accelerometer

Input.acceleration
public static Vector3 acceleration;
Description
Last measured linear acceleration of a device in three-dimensional space. (Read Only)

GetAccelerationEvent

Magnetometer

Input.compass
public static Compass **compass**;
Description
Property for accessing compass (handheld devices only). (Read Only)

The compass is actually a magnetometer that measures the magnetic field in the device's XYZ coordinates - in the absence of a stronger magnet, it will measure the Earth's field from which the compass heading can be found. This property can be used if you want to make non-standard use of the compass (eg, find the heading from the X or Z axis of the device).

```
Input.compass.rawVector.ToString()
```

Variables

enabled	**Used to enable or disable compass. Note, that if you want Input.compass.trueHeading property to contain a valid value, you must also enable location updates by calling Input.location.Start().**
headingAccuracy	**Accuracy of heading reading in degrees.**
magneticHeading	**The heading in degrees relative to the magnetic North Pole. (Read Only)**
rawVector	**The raw geomagnetic data measured in microteslas. (Read Only)**
timestamp	**Timestamp (in seconds since 1970) when the heading was last time updated. (Read Only)**
trueHeading	**The heading in degrees relative to the geographic North Pole. (Read Only)**

Gyro

Input.gyro
public static Gyroscope gyro;
Description
Returns default gyroscope.

Description
Interface into the Gyroscope.
Use this class to access gyroscope.

Variables

attitude	Returns the attitude (ie, orientation in space) of the device.
enabled	Sets or retrieves the enabled status of this gyroscope.
gravity	Returns the gravity acceleration vector expressed in the device's reference frame.
rotationRate	Returns rotation rate as measured by the device's gyroscope.
rotationRateUnbiased	Returns unbiased rotation rate as measured by the device's gyroscope.
updateInterval	Sets or retrieves gyroscope interval in seconds.
userAcceleration	Returns the acceleration that the user is giving to the device.

LocationService

Description
Interface into location functionality.
Variables

isEnabledByUser	Specifies whether location service is enabled in user settings.
lastData	Last measured device geographical location.
status	Returns location service status.

Public Functions

Start	Starts location service updates. Last location coordinates could be.
Stop	Stops location service updates. This could be useful for saving battery life.

“I can tell if it hits your head” /dj 9:36pm Jan 10, 2016

Preparation for VR/AR Conference

3 Strong Quick Demos
Interface Idea:
World Space Slide-able interface Panes

Scribble Interface
Update scribble with interface Buttons in world space
Send Drawing to other phone

Set Lengths

1 up/down
2 up/down
3 up/down
4 accept

Set Velocity

1 up/down
2 up/down
3 up/down

Set Pen Color

1 up/down

Start/stop

Do Not include /Library folder in Repository

Plastic SCM

Load new Workspace

https://www.plasticscm.com/branch-per-task-guide/gui/whats-a-workspace.html

```
cm mkrep myrepository
cm mkwk myworkspace .
cm switch /main@myrepository
cm add -R *
cm ci
```

to load a newly created workspace:
`cm update .`

Vuforia App

128

GearVR App

VuForia:

Scale on image target inversely scales it’s children

-- cc41e4fc

WIFI Command Debugging

```
adb shell screenrecord /sdcard/demo.mp4
```

adb_connect

Connect:
(Use whatever IP is listed in your Wifi settings, and the first time you do this you will need to be connected via USB)
Code: Select all

```
adb tcpip 5555
adb connect 192.168.1.etc
pause
```

List connected devices:
(To make sure it's connected over wifi - sometimes it won't show up immediately after connecting)
Code: Select all

```
adb devices
pause
```

adb_logcap

Start capturing log:
(Delete old log file, clear out old/pending log data on device, start capturing new log data to new log file. To stop capturing, just close the command window or Ctrl+break, etc. Then you can look at your new android_log.txt and see what's going on)
Code: Select all

```
del E:\AndroidLog\android_log.txt
adb logcat -c
adb logcat -s Unity:D > E:\AndroidLog\android_log.txt
```

Note that this only captures the Unity messages to avoid seeing a ton of stuff you're not interested in, but there are a bunch more that you can capture if you need additional detail from another module, like this:

adb_logcapfull

Code: Select all

```
adb logcat -s Unity:D UnityPlugin:V OVR:V VrApi:V VrLib:V ThermalEngine:V
VRManagerService:V TimeWarp:V > E:\AndroidLog\android_log.txt
```

adb_disconnect

Disconnect device from Wifi:
(You'll want to do this before you start pushing a new APK, otherwise it will try to push it wirelessly which can take a very very long time. If it's connected over wifi and connected via USB at the same time, it tends to choose the wifi connection.)
Code: Select all

```
adb disconnect 192.168.1.etc
adb devices
pause
```

adb_startserver

Starting ADB server:
(Sometimes something gets messed up and you'll need to start ADB)
Code: Select all

```
adb start-server
pause
```

Push your app to the device:
(This can be useful if you want to just Build instead of Build&Run in order to have more visibility into what you're doing. I find that uninstalling first also gives me peace of mind to ensure the old program is uninstalled and therefore if you run the new one, you know for sure it's the new version!)
Code: Select all

```
adb uninstall com.yourcompany.yourpackagename
adb install YourAPKName.apk
pause
```

My hack for quickly testing scenes is to comment out the following line from the manifest.xml file.

```
"<meta-data android:name="com.samsung.android.vr.application.mode" android:value="vr_only"/>"
```

This removes the requirement to put the phone in GearVR. Head tracking wont work but you can turn the OVRPlayerController by swiping the screen. This also allows me to view the unity profiler info at the same time.

Like I said, its a quick easy hack to make sure things actually run and to view some basic performance numbers.

https://forums.oculus.com/viewtopic.php?t=18021