2022 AR Software Research

Camilo Diaz

Studied AR Software

- ARCORE
- ARKIT
- Unity Mars
- Niantic Lightship
- IATK
- RagRug

Niantic Lightship

- OpenSource
- Works on IOS/Android
- Easy to integrate
- Outdated documentation
- Easy to complement knowledge of internal functionalities by using a C# debugger

Niantic Lightship

- ***AR Session Manager***: Manages the AR application lifecycle. The session can either be created and run automatically through Unity lifecycle events, or can be controlled programmatically. Any outstanding sessions are always cleaned up on destruction. Integrates with the capability checker to ensure the device supports AR
- ***AR Camera Position Helper:*** A helper class to automatically position an AR camera and transform its output
- ***AR Rendering Manager***: Assists with rendering our digital objects against our real-world space. Ensures that the rendering pipeline can render the camera feed to the screen and for setting up and managing how native images get rendered to the screen continuously, with every frame update.
- ***AR Depth Manager***: Simplify the process of enabling and accessing depth data,

AR Virtual Lights

Do basic directional light calculation using the information retrieved from the environment.

Research how to use the depth information to calculate distance of elements in the scene to the camera.

Place a “Virtual light” in the scene and do Phong light calculations in the shader.

How to get the color of the objects material?

How to define positions in “world space” using AR framework?

All the calculations must be done in the per fragment to avoid hitting performance or low fps.

Retrieving Depth Data

Subscribe a callback in order to retrieve the latest depth buffer:

```
_depthManager.DepthBufferUpdated += OnDepthBufferUpdated;
private void
OnDepthBufferUpdated(ContextAwarenessArgs<IDepthBuffer> args)
    {
        IDepthBuffer depthBuffer = args.Sender.AwarenessBuffer;
```

Retrieve inverse projection and view transformation matrices. Lightship updates this matrices on every frame update:

```
_BackProjectionTransform =  args.Sender.BackProjectionTransform;
        _CameraToWorldTransform =
args.Sender.CameraToWorldTransform;
```

Get last drew frame

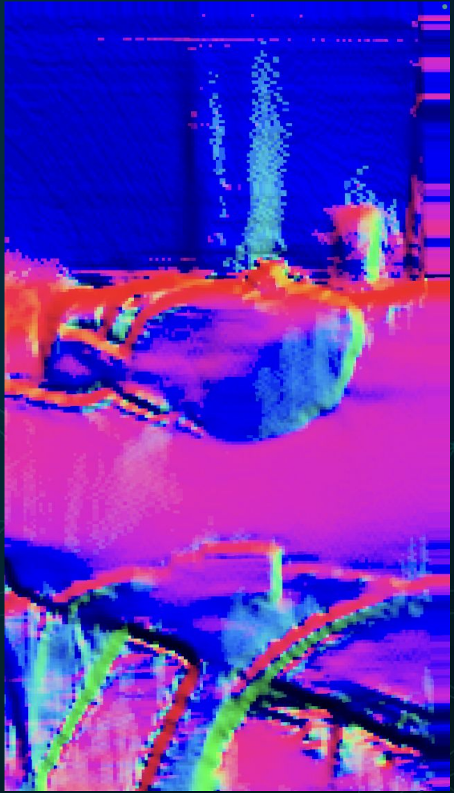
- Overwrite the GameObject method OnRenderImage and send data to the shader:

```
void OnRenderImage(RenderTexture source, RenderTexture destination)
    {
        _shaderMaterial.SetTexture("_DepthTex", _depthTexture);
      _shaderMaterial.SetMatrix("_depthTransform", _depthManager.DepthBufferProcessor.SamplerTransform);
        _shaderMaterial.SetVector("_kLightPos", _arCameraTransform);
_shaderMaterial.SetMatrix("_backProjectionTransform", _BackProjectionTransform);
        _shaderMaterial.SetMatrix("_cameraToWorldTransform", _CameraToWorldTransform );
```

Light Calculations in Shader (1/2)

- Compute Normal Map from depth data

```
float dx = 1.0/_bufferWidth;
                float dy = 1.0/_bufferHeight;
                float dzdx=(
tex2D(_DepthTex,depthUV+float2(dx,0.0)).x -
tex2D(_DepthTex,depthUV+float2(-dx,0.0)).x )/(2.0*dx);
                float dzdy=(
tex2D(_DepthTex,depthUV+float2(0.0,dy)).x -
tex2D(_DepthTex,depthUV+float2(0.0,-dy)).x )/(2.0*dy);
                float3 n=float3(-dzdx,-dzdy,1.0);
                float3 normal=normalize(n);
```



Light Calculations in Shader (2/2)

```
float3 lightDir = normalize( worldPos - _kLightPos);
                float diff = max(dot(normalWorldSpaceN,lightDir),0.0);
                float3 diffuse = diff * _lightColor.rgb;

                //float3 ambient = _lightColor.rgb  * 0.1;
                float4 result = float4((ambient+diffuse) * sourceColor,1.0);
```

Lessons learned

- It is not difficult to work with lightship. Support Forums are helpful, they reply in less than 48 hours,
- At the moment of rendering to texture (use depth buffer texture), semantic querying is not longer useful.
- Results look better on the unity editor than in phone.
- I learned more Unity rendering pipeline, shaderlab and AR implementations
-