

# Windows Holographic API와 C++을 이용한 AR앱 개발.

유영천

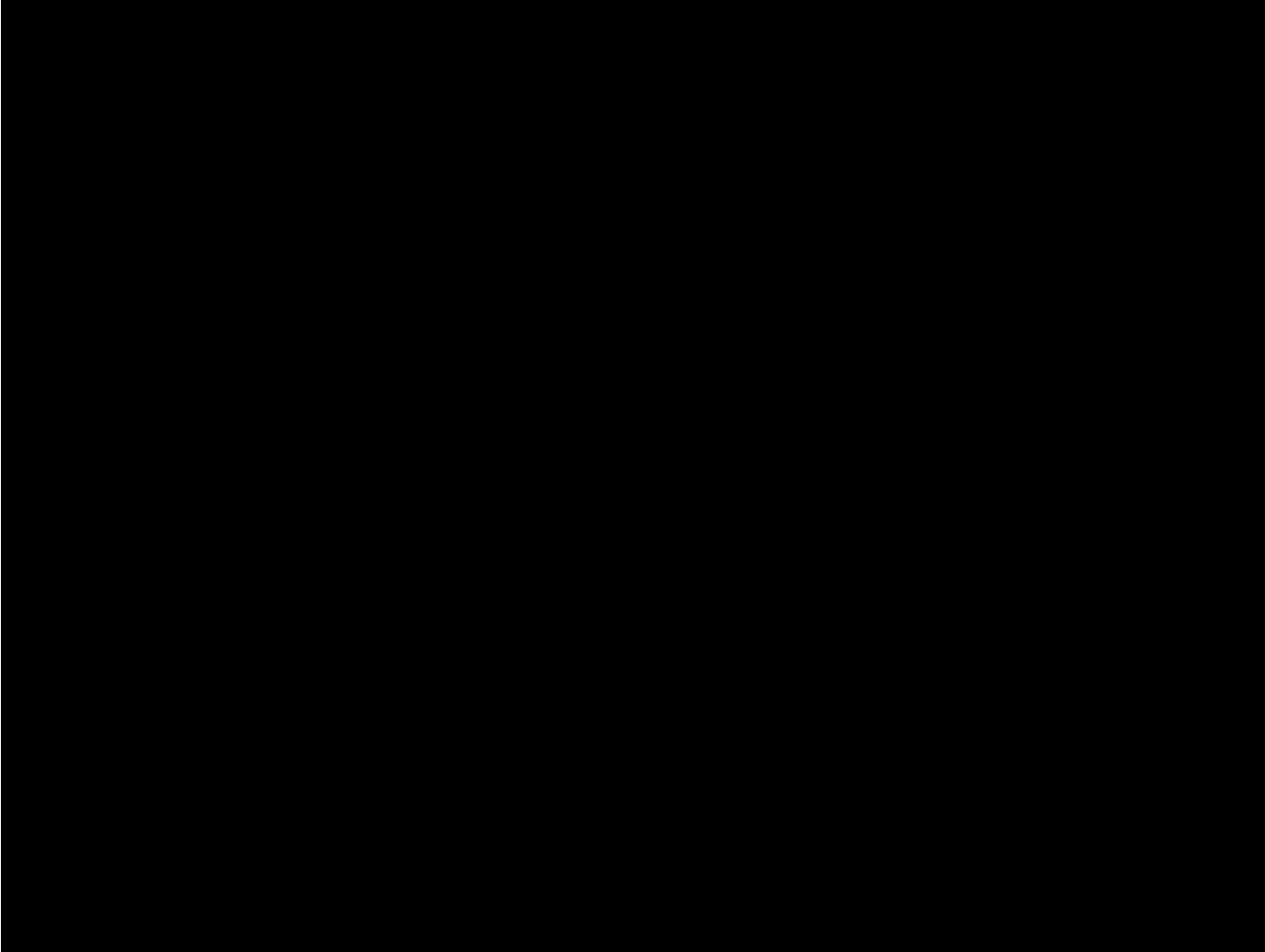
Microsoft Visual C++ MVP

Pearl Abyss

tw: @dgtman

<https://megayuchi.wordpress.com>

# AR? Holographic?



AR ?

실제 환경에 가상 사물이나  
정보를 합성하여 원래의 환경에  
존재하는 사물처럼 보이도록  
하는 컴퓨터 그래픽 기법

그리고 진짜 진짜처럼 합성합니다.

# Windows Holographic

- HoloLens등(아직은 HoloLens뿐이지만)의 VR/AR 디바이스들의 -
- VR/AR기능을 제어할 수 있게 하는 -
- Windows 10 UWP의 API
- 일단은 HoloLens를 위한 API

# HoloLens의 4가지 기능

- 반투명 스크린을 이용한 영상 출력 – 주변 사물을 볼 수 있음  
Oculus등과의 차별점
- 입체영상 – 양쪽 눈에 각각 다른 상을 보여줌으로서 입체감 부여
- 위치와 방향 판정 – 헤드셋의 센서로 내 위치와 방향을 감지
- Spatial Mapping – 주변 사물을 실시간 스캔하여 삼각형 데이터로 변환

# Holographic App 개발

사전지식

# 필요 SW & HW

- HW
  - HoloLens 디바이스 혹은 HoloLens Emulator 혹은 언젠가 나올 3<sup>rd</sup> party HMD?
- OS
  - Windows 10
- Language
  - C++/CX
  - C# - Holographic API 사용은 가능하지만 DirectX 직접 제어불가. 오늘의 주제 아님.
- API
  - UWP / DirectX 11 / Windows::Graphics::Holographic API

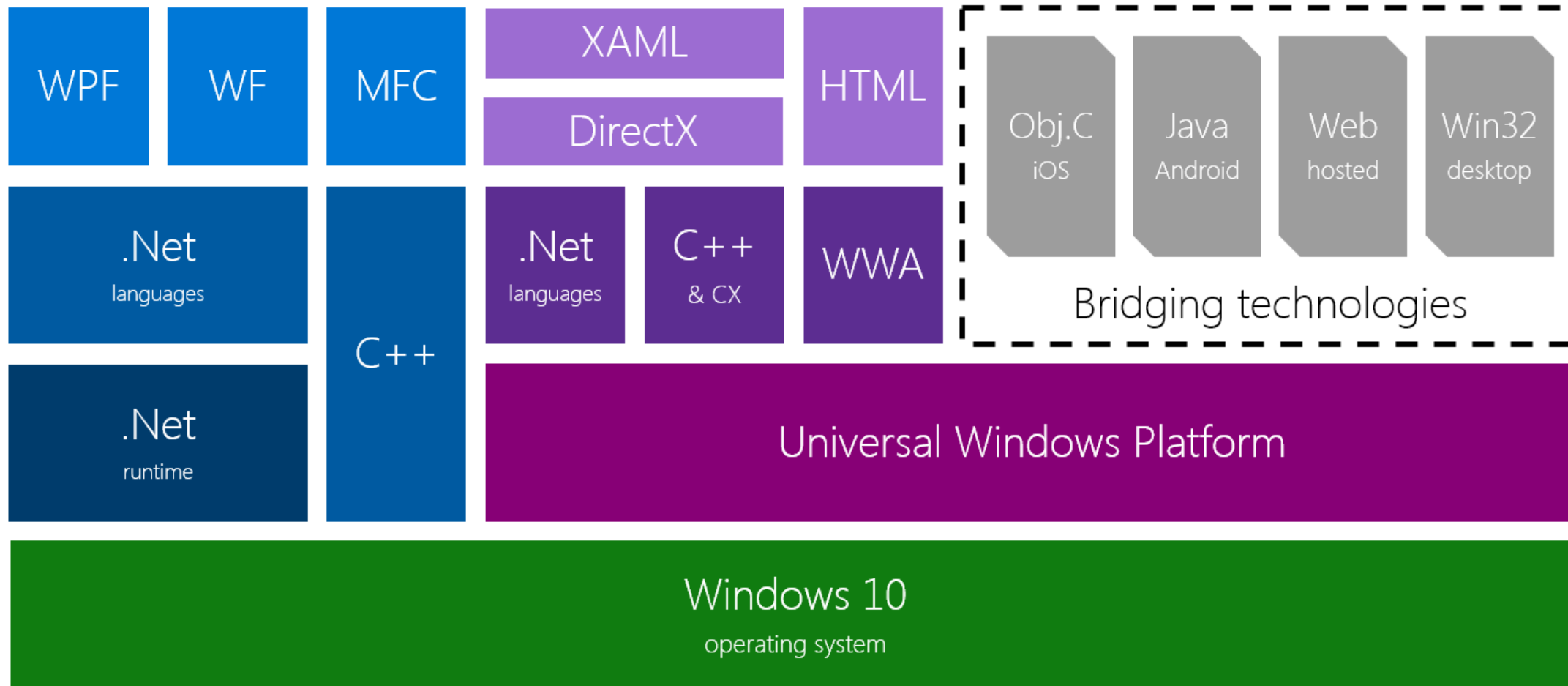
# UWP(Universal Windows Platform)

Windows 10 App = Windows Store App = UWP App

# UWP (Universal Windows Platform)

- Sandbox 시스템
- 앱의 명시적 종료가 따로 없음(iOS와 비슷)
- GDI사용할 수 없음. (WPF가 아니라면 UI는 새로 작성합니다.)
- UI는 XAML로 작성
- C++ / Java Script(HTML) , C#으로 개발가능
- C++로 개발할때
  - Win32일부 사용 가능.
  - Direct X 사용 가능.
  - 표준 C/C++ 라이브러리 어느 정도 사용 가능.





# C++/CX

Auto ref count 등, COM을 언어레벨에서 쉽게 사용하기 위한 C++ 확장.  
UWP app을 C++로 작성하고자 한다면 피해갈 수 없는 관문.

# C++/CX

- UWP API(Windows Runtime API)를 호출하기 위한 MS의 C++확장
- UWP의 모든 API는 객체지향(내부적으로 COM). C++/CX의 ref class로 구현되어 있다.
- Reference count기반 C++.
- ref class가 스마트 포인터를 내장하고 있다.
- I/O관련 모든 API는 비동기 방식. create\_task(), then() 사용.
- 당연히 기존 C/C++코드와는 공존하는데 아무 문제 없다.

# ref class 선언

```
ref class CSimpleObject sealed
{
    String^ _Name;
    ~CSimpleObject();
public: // 외부에 메타데이터를 노출함. 심지어 다른 언어에서도 이 클래스의 public 멤버 호출 가능
    property Platform::String^ Name // get(),set() 직접 코딩
    {
        Platform::String^ get()
        {
            return _Name;
        }
        void set(Platform::String^ name)
        {
            _Name = name;
        }
    }
    property int Value; // get(),set()자동 생성
    CSimpleObject();
    CSimpleObject(String^ name,int value)
    {
        _Name = name;
        Value = value;
    }
internal: // 이 모듈(빌드되는 바이너리) 내에서만 public.
    CSimpleObject^ operator+(CSimpleObject^ obj);
};
```

## ref class 구현

```
#include "pch.h"
#include "SimpleObject.h"
```

```
CSimpleObject::CSimpleObject()
{

}
```

```
CSimpleObject^ CSimpleObject::operator+(CSimpleObject^ obj)
{
    CSimpleObject^result = ref new CSimpleObject();
    result->Value = Value + obj->Value;
    result->Name = _Name + L" + " + obj->Name;

    return result;
}
```

```
CSimpleObject::~~CSimpleObject()
{
    String^Message = _Name + L" destroyed\n";
    OutputDebugString(Message->Data());
}
```

# ref class 사용

```
void MainPage::TestRefClass()
{
    CSimpleObject^ Obj0 = ref new CSimpleObject(L"Object 0",0);
    CSimpleObject^ Obj1 = ref new CSimpleObject(L"Object 1",1);

    CSimpleObject^ Obj2 = Obj0 + Obj1; // 새로운 객체 Obj2 생성

    CSimpleObject^ Obj3 = Obj2; // Obj3이 Obj2를 참조. ref count 1 증가

    Obj0 = nullptr; // ref count가 0이 되어 해제
    Obj1 = nullptr; // ref count가 0이 되어 해제
    Obj2 = nullptr; // ref count가 1 남아있으므로 해제되지 않음.
    Obj3 = nullptr; // ref count가 0이 되어 해제
}
```

<Output>

Object 0 destroyed

Object 1 destroyed

Object 0 + Object 1 destroyed

# Direct X

오랜 세월 MS플랫폼의 그래픽과 사운드를 담당해온 API

# Direct X

- MS의 Graphics API .
- Windows 95에서 Game SDK로 처음 소개
- 초기에는 비트맵을 빠르게 출력하기 위해 비디오 메모리를 액세스 할 수 있는 통로를 제공하는 것이 주 목적. Direct Draw가 주 기능.
- 게임을 위한 여러가지 기능들을 제공.DirectDraw, Direct3D, Direct Input, DirectSound, DirectMusic, DirectPlay..
- 현재는 3D Graphics API인 Direct3D와 Direct2D가 주 기능.



# DirectX 11

- XBOX, Windows Phone, HoloLens, PC 등 모든 Windows 디바이스에서 공통적으로 돌아가는 버전 -> 사실상의 표준.
- API가 추상화되어있고 드라이버에서 해주는게 많음. 따라서 GPU회사들의 지속적인 노력으로 최대한의 성능을 끌어내고 있다.
- HoloLens에서 DirectX 11을 사용함.

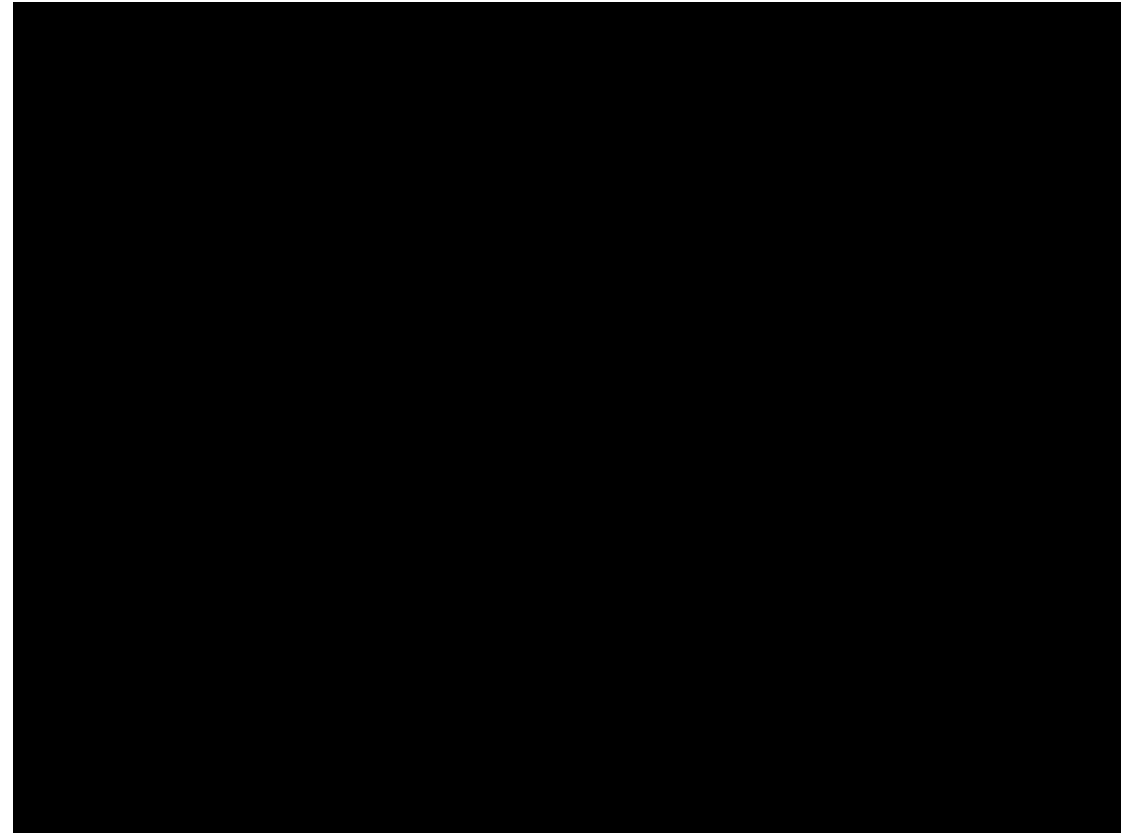
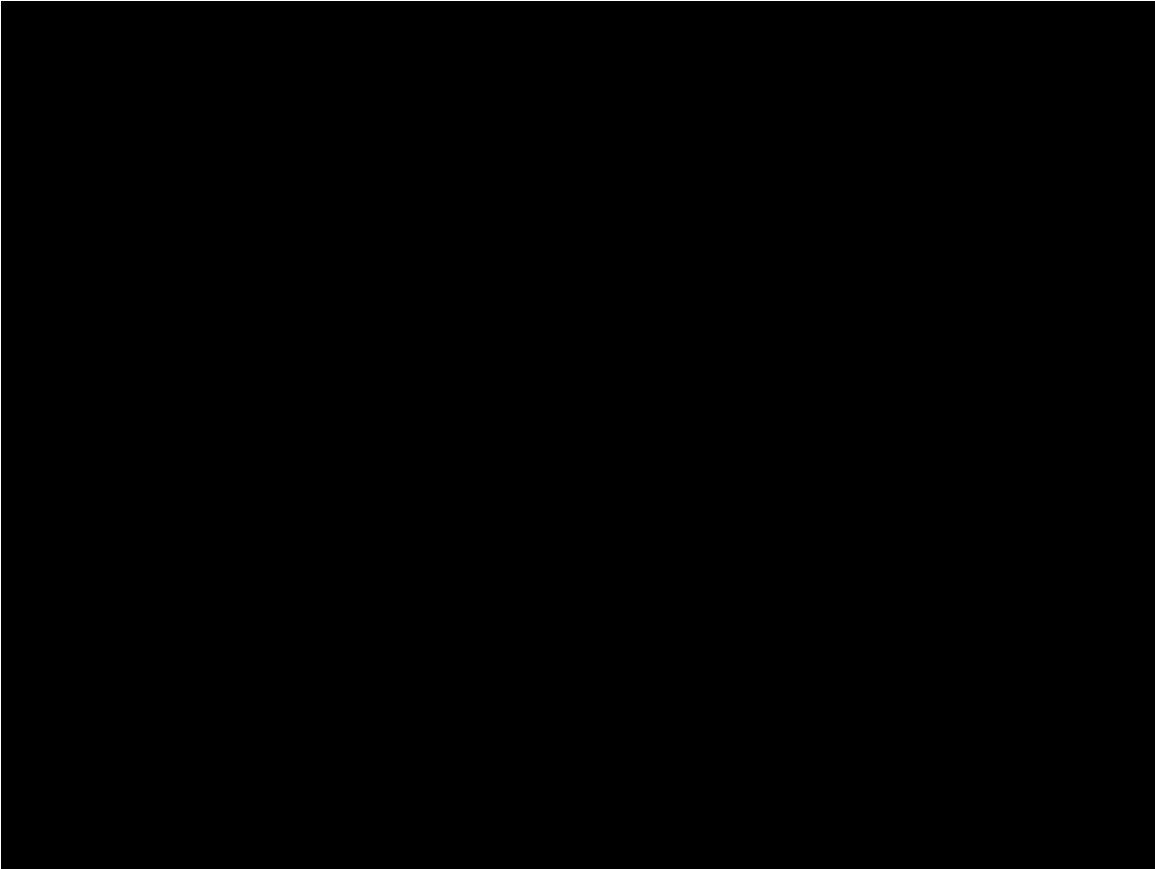
# DirectX 11 – 기본적인 사용 방법

- D3D Device, D3D Immediate Context 생성
- VertexBuffer, IndexBuffer, Texture, RenderTarget 등 resource 생성
- Shader 코드로부터 Shader 오브젝트 생성
- PSSet..(), VSSet..(), IASet..() 등으로 Resource Binding
- Draw...() 호출 -> 실제 렌더링
- Present() -> 화면 버퍼로 전송

# Hololens Emulator

- 처음 보는 순간 – 이게 뭣미? 허망하기 이룰데 없다.
- 그러나 Holographic 개발자의 친구.
- HoloLens를 한번 써보고 나면 에뮬레이터의 작동을 이해할 수 있다. 유감스럽게도...HoloLens를 한번 써봐야...

HoloLens



Emulator

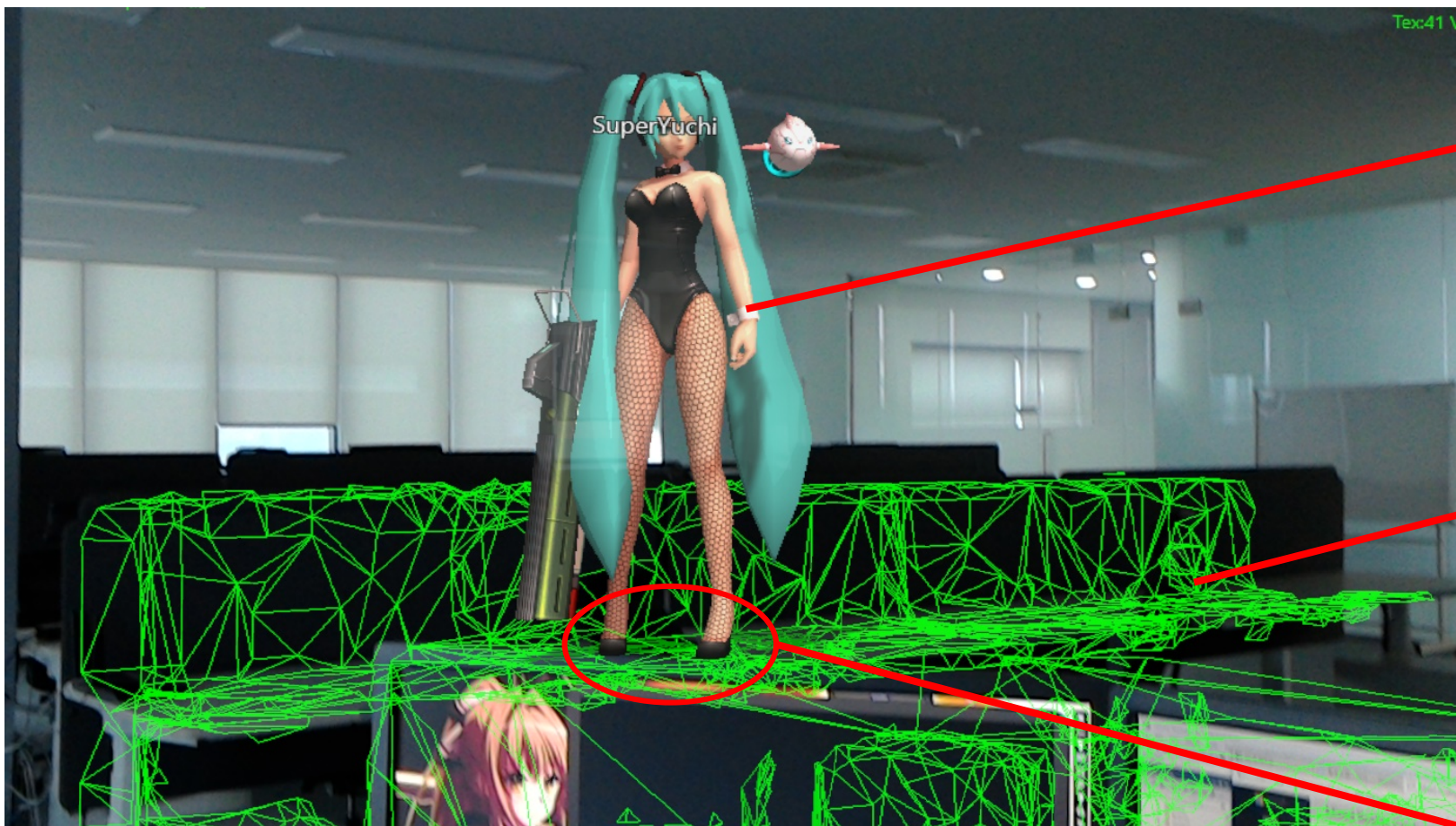
# 뭘 만들까?

게임 비슷한걸로...?

# 개발목표

- 3D 캐릭터를 HoloLens의 입체 영상으로 렌더링한다.
- 주변의 지형지물을 스캔하여 캐릭터와 상호작용하게 한다.
- HoloLens에 키보드 마우스를 붙이는것이 가능한 하지만 매우 번거로우므로 XBOX ONE 컨트롤러를 사용한다.
- Unity등의 상용엔진을 사용하지 않고 C++/CX를 이용하여 DirectX 와 Holograhpic API를 직접 제어한다.

# 이런거?



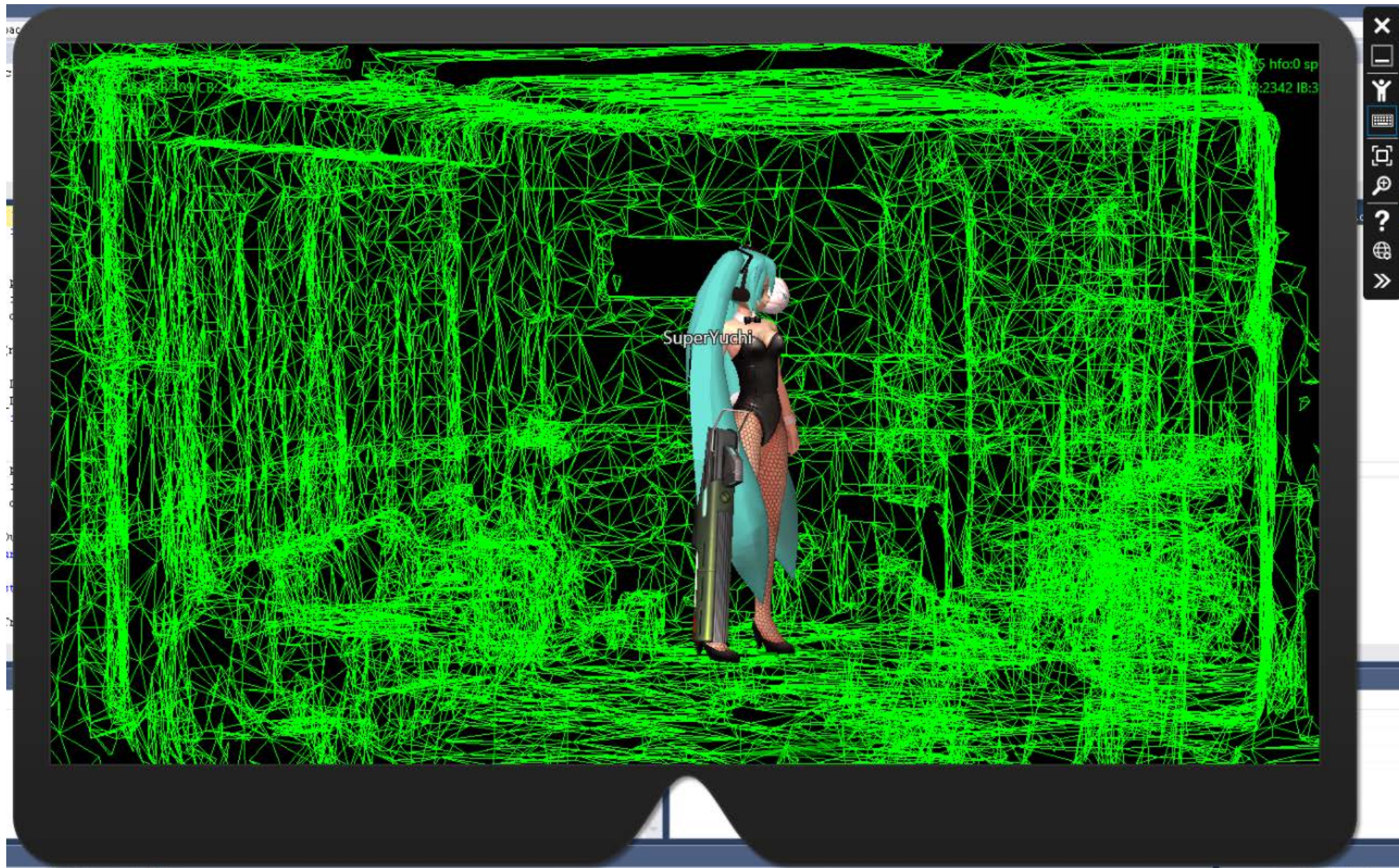
3D 모델링된  
캐릭터

지형지물을 스캔하여 얻은  
삼각형들

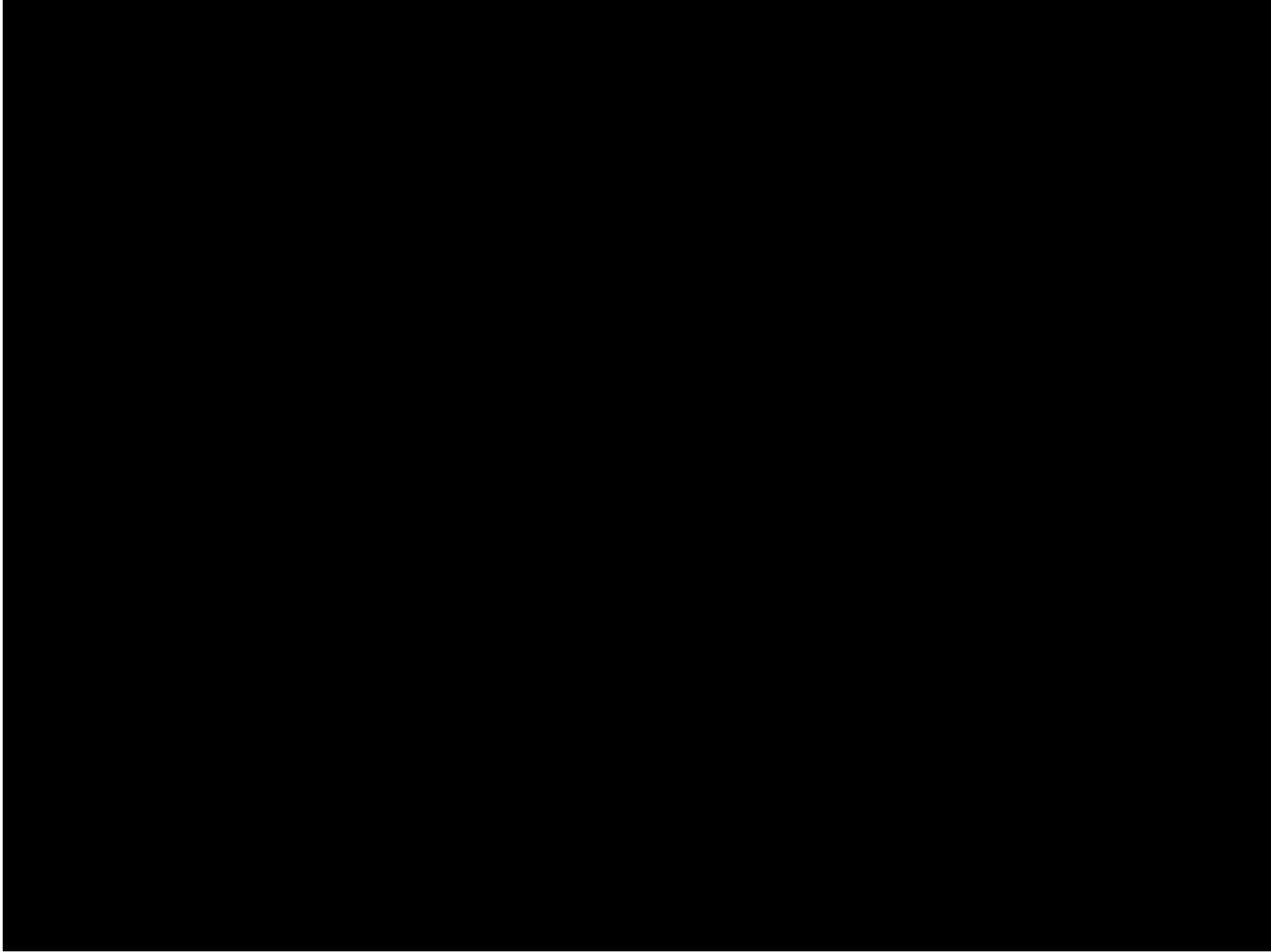
지형지물로부터 얻은  
삼각형들에  
대한 충돌처리



에뮬레이터에선 이렇게 나옵니다.ㅠㅠ







# 구현해야할 기능

- 입체영상을 위해 양쪽 눈에 대한 렌더링
  - 텍스트나 비트맵을 찍어보자. - <https://github.com/Microsoft/Windows-universal-samples/tree/master/Samples/HolographicTagAlong>
- 카메라 위치와 방향
  - 삼각형이나 3D모델을 렌더링한다. – Holographic DirectX 11 App (Universal Windows) in VS2015 project template
- Spatial Mapping
  - 주변 지형지물을 스캔해서 렌더링한다. - <https://github.com/Microsoft/Windows-universal-samples/tree/master/Samples/HolographicSpatialMapping>

# Holographic App 개발

Coding

# D3D초기화

- ID3D11Device4 객체 사용.
- ID3D11DeviceContext3 객체 사용
- 기본적으로 ID3D11Device와 ID3D11DeviceContext 객체 생성 방법은 PC의 DX11과 같다.
- Frame Buffer로 사용할 RenderTargetView를 SwapChain으로부터 얻어오지 않는다.

Windows::Graphics::Holographic::HolographicFrame  
객체로부터 얻어온다. 이 점이 다르다.

```
D3D_FEATURE_LEVEL featureLevels[] =
{
    D3D_FEATURE_LEVEL_11_1,
    D3D_FEATURE_LEVEL_11_0,
    D3D_FEATURE_LEVEL_10_1,
    D3D_FEATURE_LEVEL_10_0,
    D3D_FEATURE_LEVEL_9_3,
    D3D_FEATURE_LEVEL_9_1
};
```

```
ID3D11DeviceContext*pDeviceContext = nullptr;
ID3D11Device*pDevice = nullptr;
hr = D3D11CreateDevice(
    nullptr,
    D3D_DRIVER_TYPE_HARDWARE,
    nullptr,
    creationFlags,
    featureLevels,
    ARRAYSIZE(featureLevels),
    D3D11_SDK_VERSION,
    &pDevice,
    &m_FeatureLevel,
    &pDeviceContext
);
```

```
void CreateCameraResources(
    Windows::Graphics::Holographic::HolographicFrame^ frame,
    HolographicFramePrediction^ prediction)
{
    auto camPoses = prediction->CameraPoses;

    HolographicCameraPose^ pose = camPoses->GetAt(0);
    auto cameraParameters = frame->GetRenderingParameters(pose);

    IDirect3DSurface^ surface = cameraParameters->Direct3D11BackBuffer;

    ComPtr<ID3D11Resource> resource;
    GetDXGIInterfaceFromObject(surface, IID_PPV_ARGS(&resource));

    ComPtr<ID3D11Texture2D> cameraBackBuffer;
    resource.As(&cameraBackBuffer);

    ID3D11Texture2D* pCameraTexResource = nullptr;
    pCameraTexResource = cameraBackBuffer.Get();

    ID3D11RenderTargetView* pCameraRTV = nullptr;
    m_pD3DDevice->CreateRenderTargetView(pCameraTexResource, nullptr, &pCameraRTV);
}
```

Render Target 얻어오기

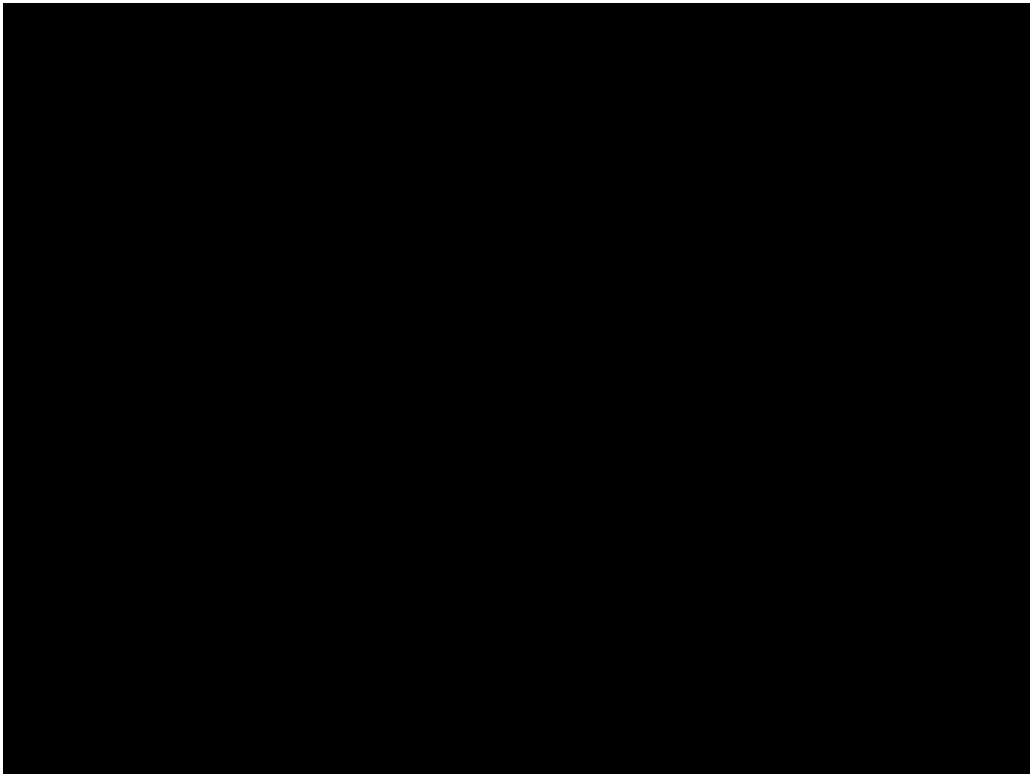


# 홀로그램을 월드 공간에 위치시키는 방법

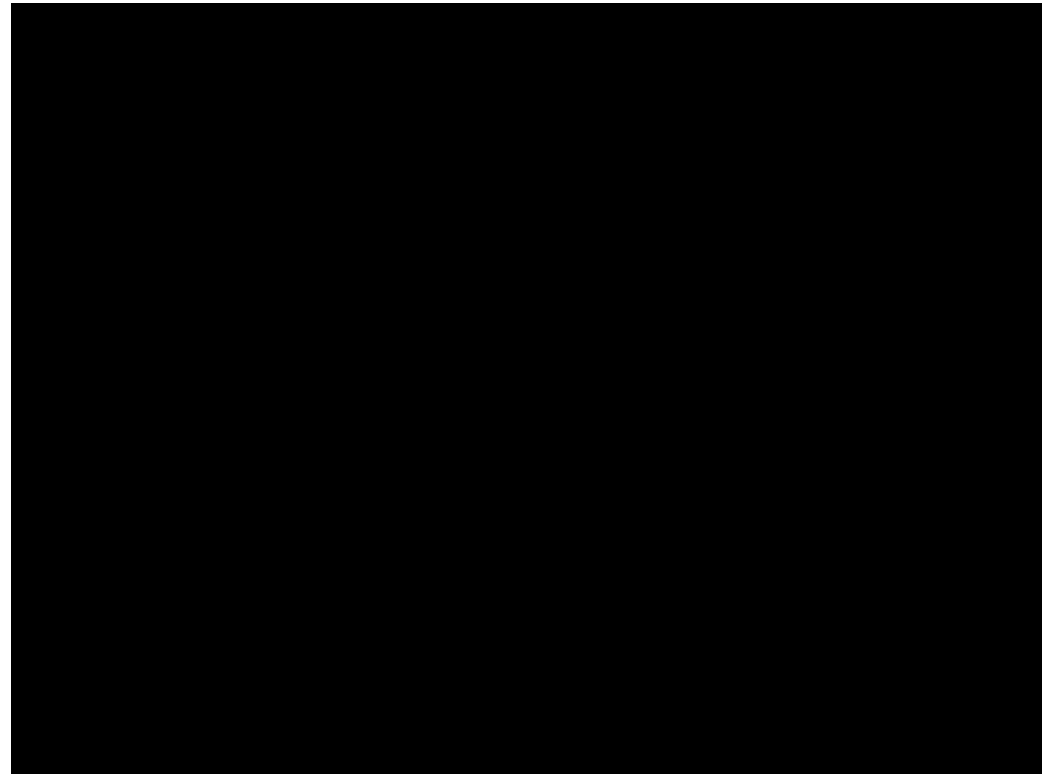
- Stationary frame of reference
  - 장치가 초기화될때 HMD가 바라보고 있는 방향이  $z=-1$ , 위치는  $0,0,0$
- Spatial anchors
  - 리얼공간상의 일종의 기준 위치
  - 원하는 위치와 방향으로 SpatialAnchor를 배치할 수 있다.
  - 각각의 SpatialAnchor로의 변환행렬을 얻을 수 있다.
  - SpatialAnchor를 이용해서 특정 위치에 홀로그램을 배치할 수 있다.
  - 저장 가능.

# 홀로그램을 월드 공간에 위치시키는 방법

**Stationary frame of reference**



**Spatial anchors**





## Stationary frame of reference 생성 및 초기화

```
m_locator = SpatialLocator::GetDefault();
```

```
m_locatabilityChangedToken =  
    m_locator->LocatabilityChanged +=  
        ref new Windows::Foundation::TypedEventHandler<SpatialLocator^, Object^>(  
            std::bind(&ClientHLMMain::OnLocatabilityChanged, this, _1, _2));
```

```
m_cameraAddedToken =  
    m_holographicSpace->CameraAdded +=  
        ref new Windows::Foundation::TypedEventHandler<HolographicSpace^,  
        HolographicSpaceCameraAddedEventArgs^>(  
            std::bind(&ClientHLMMain::OnCameraAdded, this, _1, _2));
```

```
m_cameraRemovedToken =  
    m_holographicSpace->CameraRemoved +=  
        ref new Windows::Foundation::TypedEventHandler<HolographicSpace^,  
        HolographicSpaceCameraRemovedEventArgs^>(  
            std::bind(&ClientHLMMain::OnCameraRemoved, this, _1, _2));
```

```
m_positionalTrackingDeactivatingToken =  
    m_locator->PositionalTrackingDeactivating +=  
        ref new Windows::Foundation::TypedEventHandler<SpatialLocator^,  
        SpatialLocatorPositionalTrackingDeactivatingEventArgs^>(  
            std::bind(&ClientHLMMain::OnPositionalTrackingDeactivating, this, _1, _2));
```

```
m_referenceFrame = m_locator->CreateStationaryFrameOfReferenceAtCurrentLocation();
```

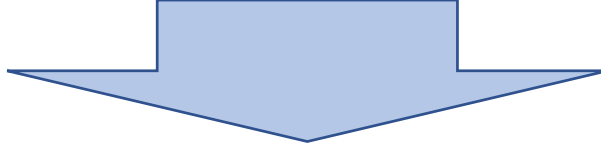
# 카메라의 위치와 방향 얻기

- 매 프레임의 업데이트 시기가 되면...
- `Windows::Graphics::Holographic::HolographicSpace::CreateNextFrame()` 호출로 다음 프레임을 준비
- `Windows::Graphics::Holographic::HolographicFrame::UpdateCurrentPrediction()` 호출로 카메라의 위치와 방향을 예측
- `Windows::UI::Input::Spatial::SpatialPointerPose::TryGetAtTimestamp()` 호출로 카메라의 위치와 방향을 얻는다.

```
HolographicFrame^ ClientHLMain::Update()  
{  
    HolographicFrame^ holographicFrame = m_holographicSpace->CreateNextFrame();  
    holographicFrame->UpdateCurrentPrediction();  
    HolographicFramePrediction^ prediction = holographicFrame->CurrentPrediction;  
  
    Windows::Perception::Spatial::SpatialCoordinateSystem^ coordinateSystem = m_referenceFrame->CoordinateSystem;  
  
    SpatialPointerPose^ pointerPose =  
        Windows::UI::Input::Spatial::SpatialPointerPose::TryGetAtTimestamp(coordinateSystem, prediction->Timestamp);  
    if (pointerPose)  
    {  
        Windows::Perception::People::HeadPose^ headPose = pointerPose->Head;  
        auto pos = headPose->Position;  
        auto up = headPose->UpDirection;  
        auto forward = headPose->ForwardDirection;  
    }  
}
```

Camera Position,  
Direction

Next Page



```
auto camPoses = prediction->CameraPoses;  
HolographicCameraPose^ pose = camPoses->GetAt(0);
```

```
HolographicStereoTransform cameraProjectionTransform = pose->ProjectionTransform;  
Platform::IBox<HolographicStereoTransform>^ viewTransformContainer = pose->  
>TryGetViewTransform(coordinateSystem);  
if (viewTransformContainer)  
{  
    HolographicStereoTransform viewCoordinateSystemTransform = viewTransformContainer->Value;
```

```
MATRIX4 matViewLeft, matProjLeft;  
matViewLeft = *(MATRIX4*)&viewCoordinateSystemTransform.Left;  
matProjLeft = *(MATRIX4*)&cameraProjectionTransform.Left;  
  
MATRIX4 matViewRight, matProjRight;  
matViewRight = *(MATRIX4*)&viewCoordinateSystemTransform.Right;  
matProjRight = *(MATRIX4*)&cameraProjectionTransform.Right;
```

View / Projection Matrix

카메라 위치와 방향, view/proj matrix 얻기

# 양안 렌더링

- 일반적인 렌더링과 크게 다르지는 않다.
- 왼쪽/오른쪽 눈에 대응하는 두개의 Render Target에 렌더링 한다.  
RenderTextureArray와 인스턴싱을 사용해서 한번에 두개의 Render Target View에 렌더링한다.
- Holographic API로부터 양안에 대한 view/projection matrix를 얻어온다. -> Constant Buffer로 전달
- HoloLens 디바이스에선 VertexShader 에서 SV\_RenderTargetArrayIndex 시맨틱을 사용할 수 있지만 에뮬레이터에선 사용할 수 없으므로 추가로 Geometry Shader가 필요하다

# CPU측 렌더링 코드

```
#ifdef STEREO_RENDER
    pDeviceContext->DrawIndexedInstanced(dwIndicesNum, 2, 0, 0, 0);
#else
    pDeviceContext->DrawIndexed(dwIndicesNum, 0, 0);
#endif
```

Geometry Shader 외의 Shader Stage에서 SV\_RenderTargetArrayIndex 시맨틱을 사용할 수 있는지 체크할것 (현재 HoloLens 디바이스에서만 지원)

```
D3D11_FEATURE_DATA_D3D11_OPTIONS3 options;  
m_pD3DDevice->CheckFeatureSupport(D3D11_FEATURE_D3D11_OPTIONS3, &options, sizeof(options));  
m_bRTArrayIndexFromAnyShader = options.VPAndRTArrayIndexFromAnyShaderFeedingRasterizer;
```

# Shader declaration

```
struct VS_INPUT_BBOARD
{
    float4  Pos          : POSITION;
    float2  TexCoord     : TEXCOORD0;
    uint    instId       : SV_InstanceID;
};

struct VS_OUTPUT_BBOARD
{
    float4  Pos          : SV_POSITION;
    float4  Color         : COLOR0;
    float4  NormalColor  : COLOR1;
    float2  TexCoord     : TEXCOORD0;
    uint    ArrayIndex   : BLENDINDICES;

    #if (1 == VS_RTV_ARRAY)
        uint    RTVIndex : SV_RenderTargetArrayIndex;
    #endif
};

struct GS_OUTPUT_BBOARD : VS_OUTPUT_BBOARD
{
    #if (1 != VS_RTV_ARRAY)
        uint RTVIndex : SV_RenderTargetArrayIndex;
    #endif
};

struct PS_INPUT_BBOARD : VS_OUTPUT_BBOARD
{
};
```

D3D11\_FEATURE\_D3D11\_OPTIONS3를 지원하는 경우

D3D11\_FEATURE\_D3D11\_OPTIONS3를 지원하지 않는  
경우. Emulator등



# Vertex Shader

```
PS_INPUT_BBOARD vsDefault(VS_INPUT_BBOARD input)
{
    PS_INPUT_BBOARD output = (PS_INPUT_BBOARD) 0;

    uint ArrayIndex = input.instId % 2;
    output.Pos = mul(input.Pos, matWorldViewProjArray[ArrayIndex]);
    output.NormalColor.rgb = float3(0, -1, 0);
    output.NormalColor.a = 1;
    output.Color = MtlDiffuse + MtlAmbient;
    output.TexCoord = input.TexCoord;
    output.ArrayIndex = ArrayIndex;
    #if (1 == VS_RTV_ARRAY)
        output.RTVIndex = ArrayIndex;
    #endif
    return output;
}
```

D3D11\_FEATURE\_D3D11\_OPTIONS3를 지원하는 경우

# Geometry Shader

(Hololens 디바이스에선 필요없음)

```
[maxvertexcount(3)]  
void gsDefault(triangle VS_OUTPUT_BBOARD input[3], inout TriangleStream<GS_OUTPUT_BBOARD> TriStream)  
{  
    GS_OUTPUT_BBOARD output[3];  
  
    for (uint i = 0; i < 3; i++)  
    {  
        output[i].Pos = input[i].Pos;  
        output[i].Color = input[i].Color;  
        output[i].NormalColor = input[i].NormalColor;  
        output[i].TexCoord = input[i].TexCoord;  
        output[i].Clip = input[i].Clip;  
        output[i].ArrayIndex = input[i].ArrayIndex;  
        output[i].RTVIndex = input[i].ArrayIndex;  
        TriStream.Append(output[i]);  
    }  
}
```

D3D11\_FEATURE\_D3D11\_OPTIONS3를 지원하지 않는  
경우. Emulator등

# Pixel Shader

(일반 렌더링과 똑같음)

```
PS_TARGET psDefault(PS_INPUT_BBOARD input)
{
    PS_TARGET output = (PS_TARGET) 0;

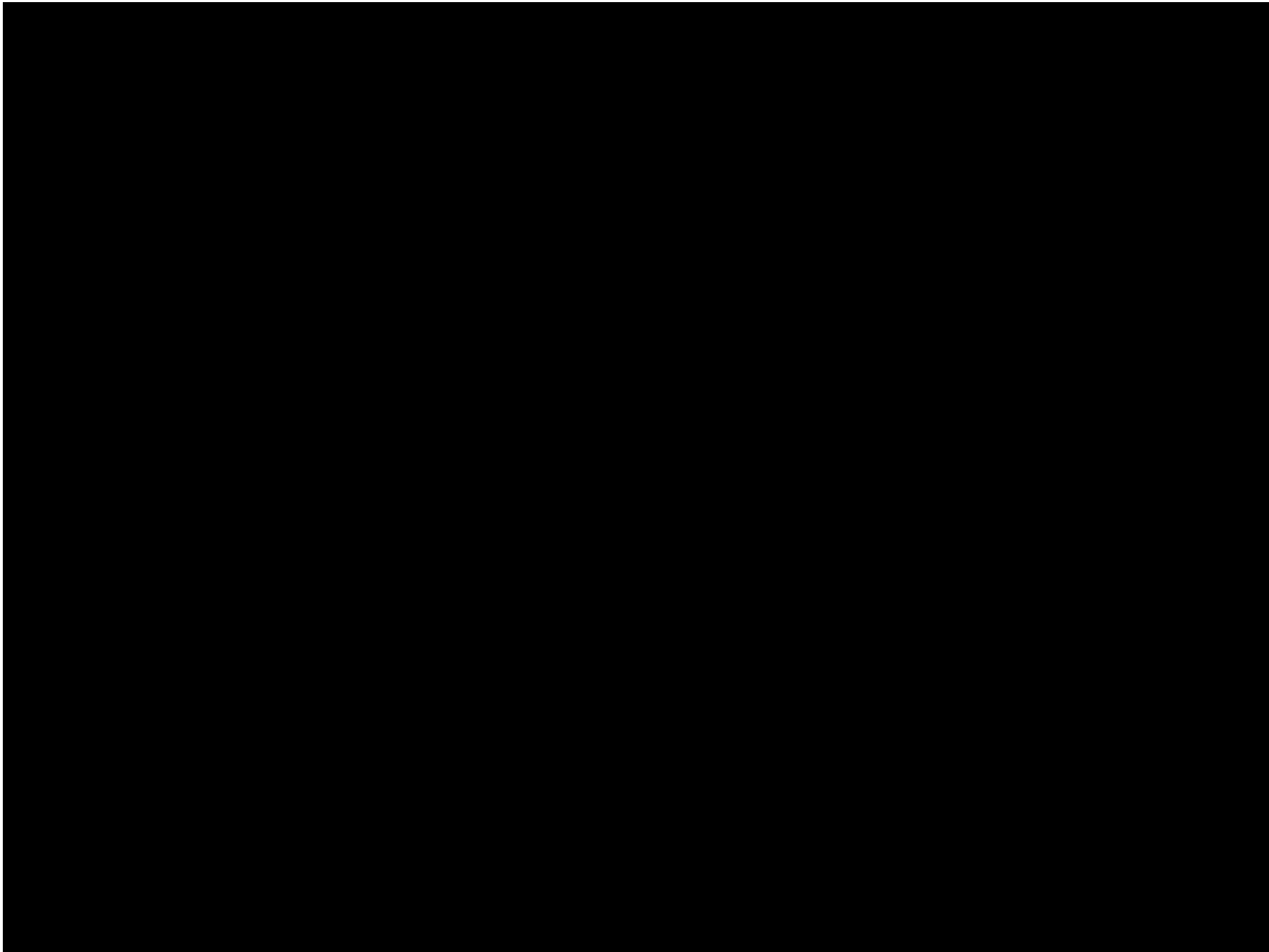
    float4 texColor = texDiffuse.Sample(samplerWrap, input.TextCoord);
    float4 outColor = texColor * input.Color;

    output.Color0 = outColor;

    return output;
}
```

# Spatial Mapping

- HoloLens가 주변을 스캔하여 삼각형 매시를 생성
- 삼각형 매시는 일정 단위로 쪼개져서 Surface Mesh라는 단위로 전달된다.
- 각 Surface Mesh는 고유의 GUID를 가지고 있다.
- Surface Mesh가 갱신되면 이벤트 핸들러에 해당 GUID가 전달된다.
- 이벤트 핸들러는 UI 스레드가 아닌 임의의 스레드에서 호출된다.

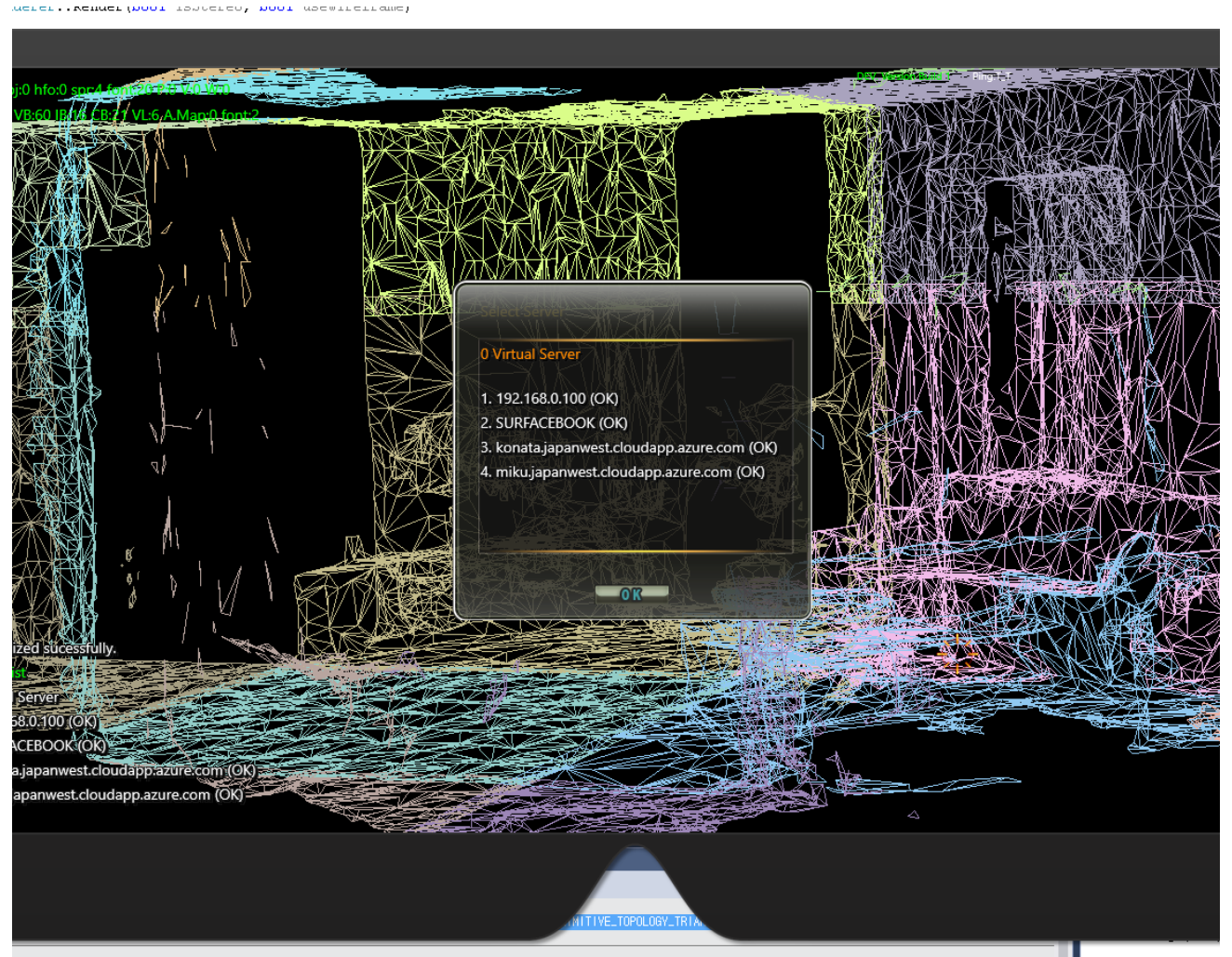


# Spatial Mapping – Surface Mesh

각각의 Surface Mesh를 다른  
색으로 렌더링 해보면...

대략 일정 크기로 나뉘어 있지만  
Surface Mesh의 AABB를 렌더링  
해보면 일부 겹치는 것을 확인할  
수 있다.

Surface Mesh 한 개당 삼각형  
개수가 제법 많기 때문에(500-  
3000) 그 자체만으로 공간 분할  
단위로 사용하긴 어렵다.



# Spatial Mapping – 구현

- Spatial Mapping API 권한 체크
- 삼각형 매시를 어떤 포맷으로 얻을지 설정
- `Windows::Perception::Spatial::Surfaces::SpatialSurfaceObserver` 객체 생성
- `Windows::Perception::Spatial::Surfaces::SpatialSurfaceObserver::GetObservedSurfaces()` 호출로 최초의 Surface Mesh 수집
- `Windows::Perception::Spatial::Surfaces::SpatialSurfaceObserver::ObservedSurfacesChanged()` 이벤트 핸들러 세팅으로 이후 Surface Mesh 갱신 이벤트 수신 -> Surface Mesh 갱신

Spatial mapping API가 사용자의 환경을 스캔할 권한이 있는지  
체크한다.

```
auto initSurfaceObserverTask = create_task(SpatialSurfaceObserver::RequestAccessAsync());
initSurfaceObserverTask.then([this,
currentCoordinateSystem](Windows::Perception::Spatial::SpatialPerceptionAccessStatus status)
{
    switch (status)
    {
        case SpatialPerceptionAccessStatus::Allowed:
            m_surfaceAccessAllowed = true;
            break;
        default:
            // Access was denied.
            m_surfaceAccessAllowed = false;
            break;
    }
});
```

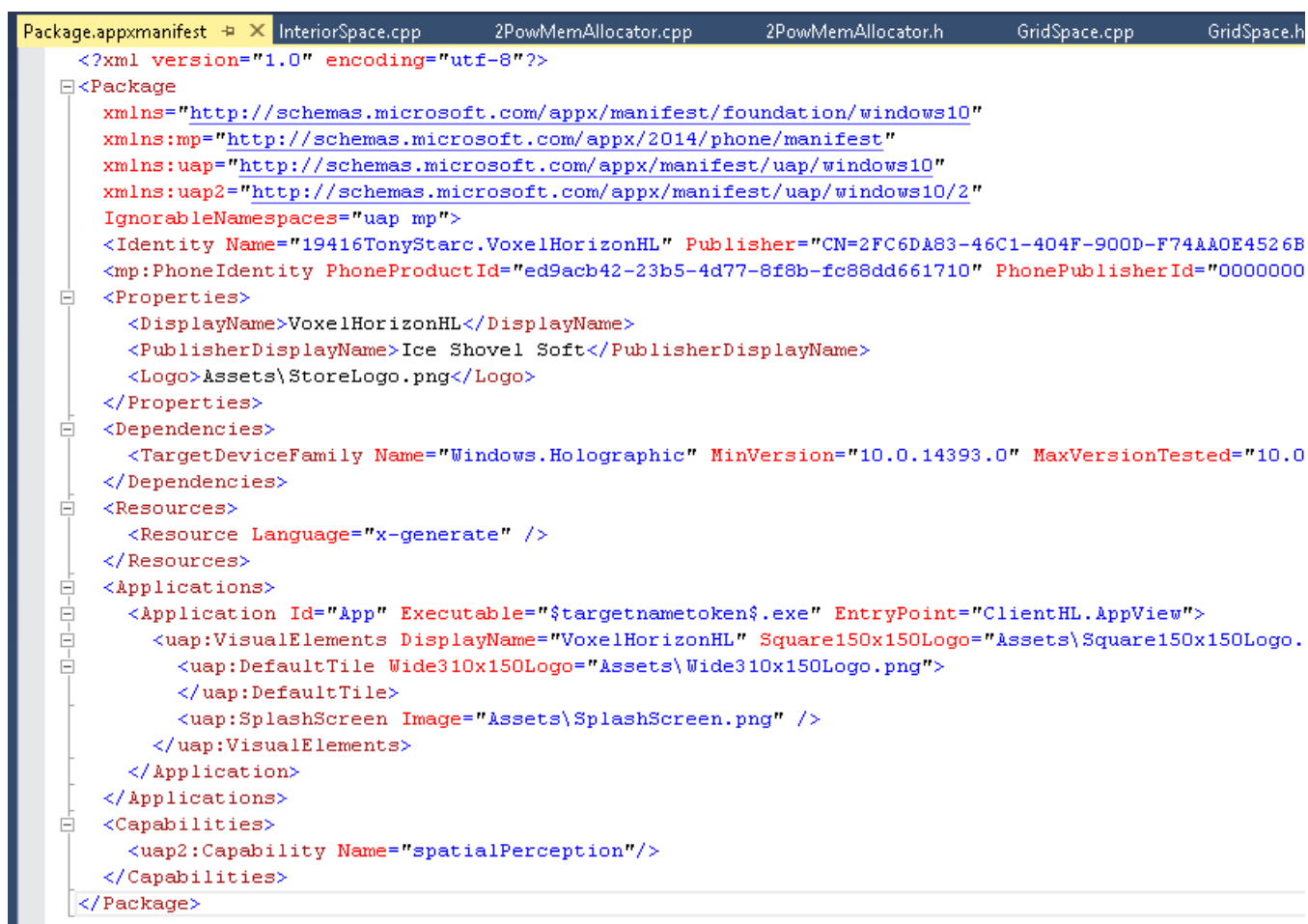


Package.appxmanifest파일에 SpatialPerception 항목을 수동으로 추가해야 한다.

<Capabilities>

<uap2:Capability Name="spatialPerception"/>

</Capabilities>



```
<?xml version="1.0" encoding="utf-8"?>
<Package
  xmlns="http://schemas.microsoft.com/appx/manifest/foundation/windows10"
  xmlns:mp="http://schemas.microsoft.com/appx/2014/phone/manifest"
  xmlns:uap="http://schemas.microsoft.com/appx/manifest/uap/windows10"
  xmlns:uap2="http://schemas.microsoft.com/appx/manifest/uap/windows10/2"
  IgnorableNamespaces="uap mp">
  <Identity Name="19416TonyStarc.VoxelHorizonHL" Publisher="CN=2FC6DA83-46C1-404F-900D-F74AA0E4526B"
    <mp:PhoneIdentity PhoneProductId="ed9acb42-23b5-4d77-8f8b-fc88dd661710" PhonePublisherId="00000000"
  <Properties>
    <DisplayName>VoxelHorizonHL</DisplayName>
    <PublisherDisplayName>Ice Shovel Soft</PublisherDisplayName>
    <Logo>Assets\StoreLogo.png</Logo>
  </Properties>
  <Dependencies>
    <TargetDeviceFamily Name="Windows.Holographic" MinVersion="10.0.14393.0" MaxVersionTested="10.0"
  </Dependencies>
  <Resources>
    <Resource Language="x-generate" />
  </Resources>
  <Applications>
    <Application Id="App" Executable="$targetnametoken$.exe" EntryPoint="ClientHL.AppView">
      <uap:VisualElements DisplayName="VoxelHorizonHL" Square150x150Logo="Assets\Square150x150Logo.
        <uap:DefaultTile Wide310x150Logo="Assets\Wide310x150Logo.png">
        </uap:DefaultTile>
        <uap:SplashScreen Image="Assets\SplashScreen.png" />
      </uap:VisualElements>
    </Application>
  </Applications>
  <Capabilities>
    <uap2:Capability Name="spatialPerception"/>
  </Capabilities>
</Package>
```

# Spatial Mapping API가 넘겨줄 삼각형 메시의 포맷을 설정

```
SpatialBoundingVolume^ bounds = SpatialBoundingVolume::FromBox(currentCoordinateSystem,
axisAlignedBoundingBox);

m_surfaceMeshOptions = ref new SpatialSurfaceMeshOptions();
IVectorView<DirectXPixelFormat>^ supportedVertexPositionFormats =
    m_surfaceMeshOptions->SupportedVertexPositionFormats;

unsigned int formatIndex = 0;

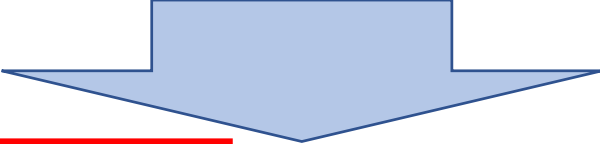
if (supportedVertexPositionFormats->IndexOf(DirectXPixelFormat::R32G32B32A32Float, &formatIndex))
{
    m_surfaceMeshOptions->VertexPositionFormat = DirectXPixelFormat::R32G32B32A32Float;
}

IVectorView<DirectXPixelFormat>^ supportedVertexNormalFormats =
    m_surfaceMeshOptions->SupportedVertexNormalFormats;

if (supportedVertexNormalFormats->IndexOf(DirectXPixelFormat::R8G8B8A8IntNormalized, &formatIndex))
{
    m_surfaceMeshOptions->VertexNormalFormat = DirectXPixelFormat::R8G8B8A8IntNormalized;
}
```



Next Page



```
SpatialBoundingBox axisAlignedBoundingBox =  
{  
    { 0.f, 0.f, 0.f },  
    { 20.f, 20.f, 10.f },  
};  
m_surfaceObserver = ref new SpatialSurfaceObserver();  
m_surfaceObserver->SetBoundingVolume(bounds);
```

SpatialSurfaceObserver를 생성

```
Windows::Perception::Spatial::SpatialCoordinateSystem^ coordinateSystem =  
    m_referenceFrame->CoordinateSystem;
```

```
auto mapContainingSurfaceCollection = m_surfaceObserver->GetObservedSurfaces();  
for (auto const& pair : mapContainingSurfaceCollection)  
{  
    auto const& id = pair->Key;  
    auto const& surfaceInfo = pair->Value;  
    m_pSurfaceCacheManager->AddSurface(id, surfaceInfo, m_surfaceMeshOptions, coordinateSystem);  
}
```

최초의 Surface Mesh를 수집

```
m_surfacesChangedToken = m_surfaceObserver->ObservedSurfacesChanged +=  
    ref new TypedEventHandler<SpatialSurfaceObserver^, Platform::Object^>(  
        bind(&ClientHLMMain::OnSurfacesChanged, this, _1, _2)  
    );
```

Surface Mesh의 변경 이벤트가  
발생했을때 호출될 핸들러 설정

```

void ClientHLMain::OnSurfacesChanged(SpatialSurfaceObserver^ sender, Object^ args)
{
    IMapView<Guid, SpatialSurfaceInfo^>^ surfaceCollection = sender->GetObservedSurfaces();
    auto dispatcher = Windows::ApplicationModel::Core::CoreApplication::MainView->CoreWindow->Dispatcher;
    dispatcher->RunAsync(Windows::UI::Core::CoreDispatcherPriority::Normal, ref new
Windows::UI::Core::DispatchedHandler([this, surfaceCollection]()
{
    Windows::Perception::Spatial::SpatialCoordinateSystem^ coordinateSystem = m_referenceFrame-
>CoordinateSystem;
    for (const auto& pair : surfaceCollection)
    {
        auto id = pair->Key;
        auto surfaceInfo = pair->Value;

        if (m_pSurfaceCacheManager->HasSurface(id))
        {
            m_pSurfaceCacheManager->UpdateSurface(id, surfaceInfo, m_surfaceMeshOptions,
coordinateSystem);
        }
        else
        {
            m_pSurfaceCacheManager->AddSurface(id, surfaceInfo, m_surfaceMeshOptions,
coordinateSystem);
        }
    }
}));
}

```

Surface Mesh의 업데이트 이벤트 처리

# Spatial Mapping으로 얻은 삼각형 관리

- 충돌처리/picking을 하기에 Surface Mesh는 단위가 너무 크다.
- 각각의 Surface Mesh간 기하학적 관계가 없다.
- 따라서 KD-Tree, Grid, BSP Tree 뭐든 간에 3차원 공간에 적합한 자료구조를 만들어서 삼각형들을 관리해야 한다.

# TIP

- 반드시 win32 버전부터 개발할 것.
- 거의 같은 코드를 유지하는 UWP버전을 개발할 것.
- 그 UWP버전으로 Holographic 버전을 개발할 것.
- 에뮬레이터를 적극 활용할 것.
- 에뮬레이터는 Dynamic Buffer를 사용해서 렌더링 할 때 문제가 생긴다. D3D11\_USAGE\_DEFAULT타입의 버퍼로 Debugging용 매시를 렌더링할 방법을 준비할 것.
- HoloLens의 GPU와 CPU는 생각보다 훨씬 느리다.
- 속도! 속도! 속도! 메모리! 메모리! 메모리!

# Reference

- <https://megayuchi.wordpress.com> 에서 HoloLens로 검색
- [https://developer.microsoft.com/ko-kr/windows/holographic/development overview](https://developer.microsoft.com/ko-kr/windows/holographic/development%20overview)

Q/A

