Relazione Hololens - Unity

HoloLens 2 rappresenta il vertice dell'innovazione nella realtà aumentata (AR) e nella realtà mista (MR). Questi dispositivi offrono un'esperienza visiva immersiva, consentendo agli utenti di interagire con il mondo digitale in modo senza precedenti.
Tuttavia, per garantire un'esperienza utente fluida e coinvolgente, è cruciale condurre analisi approfondite delle performance grafiche delle HoloLens 2.

Gli strumenti per l'Analisi delle Performance Grafiche
Per condurre analisi precise delle performance grafiche su HoloLens 2, è essenziale utilizzare gli strumenti giusti. Alcuni degli strumenti principali:

- Unity Profiler: Unity è una delle principali piattaforme di sviluppo per esperienze AR/MR. Il suo profiler offre una visione dettagliata delle performance, con informazioni sul tempo di rendering, l'utilizzo della CPU e della GPU e molto altro. Ad esempio, è possibile identificare specifici script o componenti che causano rallentamenti nell'applicazione Unity e apportare correzioni.

- Visual Studio Graphics Diagnostics: Questo strumento permette di condurre analisi approfondite delle performance grafiche direttamente da Visual Studio. Si può effettuare il debug in tempo reale, analizzare il rendering dei frame e individuare i problemi di texture o shader che possono compromettere le prestazioni. Ecco un esempio: un developer potrebbe utilizzare Visual Studio Graphics Diagnostics per individuare un eccessivo utilizzo della GPU durante la visualizzazione di un oggetto 3D complesso, causato da un shader inefficiente.

- Windows Performance Analyzer: Questo strumento consente di analizzare le performance del sistema operativo Windows e delle applicazioni. Può rivelarsi utile per identificare problemi di sistema che influiscono sulle performance grafiche. Ad esempio, si potrebbe scoprire che un servizio in background sta causando un consumo eccessivo di CPU durante l'esecuzione di un'app AR.

- HoloLens Device Portal: Questa interfaccia web fornisce accesso ai dati in tempo reale direttamente dal dispositivo HoloLens 2. Gli sviluppatori possono monitorare l'utilizzo della CPU, della GPU e della memoria durante l'esecuzione dell'applicazione. Un esempio concreto sarebbe l'utilizzo di Device Portal per rilevare un aumento repentino della temperatura del dispositivo, che potrebbe indicare un surriscaldamento dovuto a un'applicazione inefficiente.

Metriche Chiave per l'Analisi delle Performance Grafiche

Durante le analisi delle performance grafiche su HoloLens 2, è importante tenere sotto controllo alcune metriche fondamentali:

- Frame per secondo (FPS): Questa metrica indica quanti frame vengono visualizzati al secondo. Per un'esperienza fluida, si mira a mantenere un FPS stabile di 60 o superiore. Ad esempio, se l'app presenta picchi di FPS inferiori a 30 durante l'interazione con oggetti virtuali complessi, l'esperienza risulterà sgradevole per l'utente.

- Tempo di rendering: Questo rappresenta il tempo necessario per generare un frame completo. Un tempo di rendering superiore a 16,67 millisecondi (corrispondente a 60 FPS) può causare scatti nell'animazione. Un esempio concreto potrebbe essere un tempo di rendering di 25 millisecondi per un frame, che indica la necessità di ottimizzare gli elementi

visivi dell'applicazione.

- Utilizzo della CPU e della GPU: Monitorare l'utilizzo di CPU e GPU è essenziale per garantire un'efficienza energetica e una durata della batteria ottimale. Ad esempio, un'app che utilizza costantemente il 90% della CPU potrebbe esaurire rapidamente la batteria delle HoloLens 2, riducendo l'esperienza dell'utente.

- Latenza di input: Questa metrica misura il ritardo tra l'input dell'utente e la risposta del sistema. Un alto ritardo può causare un'esperienza utente deludente. Ad esempio, se c'è un ritardo significativo tra il momento in cui l'utente tocca un oggetto virtuale e la risposta del dispositivo, l'interazione risulterà poco naturale.

Concetti alla base del rendering di Hololens:
Il rendering su dispositivi olografici è fondamentalmente diverso rispetto alle tradizionali visualizzazioni bidimensionali. Invece di proiettare semplicemente immagini su uno schermo piatto, questi dispositivi devono interagire con il mondo reale sovrapponendo contenuti digitali nel campo visivo dell'utente.
Gli aspetti chiave del rendering olografico:

1. Spatial Awareness: I dispositivi olografici devono comprendere e rispondere all'ambiente fisico. Questo comporta il mappaggio spaziale in tempo reale e il tracciamento dei movimenti dell'utente e dell'ambiente circostante. È cruciale che il rendering si allinei con l'angolazione dell'utente in modo accurato.
2. Layering Digital Content: Il rendering olografico implica il posizionamento di oggetti digitali all'interno del mondo fisico. Questi oggetti digitali, spesso chiamati ologrammi, dovrebbero fondersi perfettamente con l'ambiente dell'utente, mantenendo correttezza nella sovrapposizione, nella profondità e nell'illuminazione.
3. Interactivity: il rendering olografico non riguarda solo la visualizzazione di ologrammi statici. Comprende anche le interazioni dell'utente con questi elementi digitali. Questa interattività deve risultare naturale ed intuitiva, sia essa basata su tocco, gesti o comandi vocali.

GPU
La GPU nelle HoloLens è progettata per eseguire calcoli in parallelo, il che significa che può elaborare una vasta quantità di dati grafici simultaneamente. Questa caratteristica è fondamentale per gestire in tempo reale gli oggetti virtuali e i dati di tracciamento necessari per la realtà mista.

Rendering 3D e Shader

HoloLens gestisce la GPU (Graphics Processing Unit) in modo ottimizzato per offrire un'esperienza di realtà mista (MR) fluida e coinvolgente.
La gestione della GPU in HoloLens coinvolge vari aspetti:

- Rendering 3D: La GPU è principalmente responsabile del rendering dei contenuti 3D e dei modelli virtuali all'interno dell'ambiente del mondo reale. Ciò include la generazione di ologrammi, oggetti virtuali e l'elaborazione grafica degli elementi visivi. La GPU deve essere in grado di generare immagini realistiche e garantire che gli oggetti virtuali si integrino in modo credibile con l'ambiente fisico dell'utente.

- Ottimizzazione delle Prestazioni: HoloLens è progettato per operare con prestazioni fluide. La GPU è gestita in modo da garantire un alto tasso di frame per secondo (FPS) per evitare lag e garantire un'esperienza utente fluida. Sono implementate tecniche di ottimizzazione come l'Occlusion Culling degli oggetti non visibili, la riduzione dei dettagli per oggetti lontani e il bilanciamento delle prestazioni in base al carico di lavoro.

- **Vertex Shader**: Il vertex shader è il primo passo nel processo di rendering. Esso elabora le informazioni relative ai vertici dei modelli 3D, trasformandoli dalle loro coordinate locali dell'oggetto alle coordinate globali nello spazio del mondo.
  Ad esempio, quando si interagisce con un oggetto virtuale in una stanza, il vertex shader si occupa di posizionarlo correttamente in base alla tua posizione e alla tua prospettiva. Inoltre, può essere utilizzato per applicare trasformazioni, deformazioni o animazioni ai vertici.

- **Fragment Shader**: Il fragment shader, noto anche come pixel shader, si concentra sull'elaborazione dei pixel. Ogni pixel sullo schermo è sottoposto al fragment shader, che determina il colore, la trasparenza e le proprietà visive di quel pixel in base a variabili come la luce, le texture e le ombre. Questo contribuisce a creare l'aspetto realistico degli oggetti virtuali e della scena MR nel complesso.

- Gestione Termica: Per evitare il surriscaldamento del dispositivo, HoloLens implementa sistemi di gestione termica che includono il raffreddamento passivo e, in alcuni casi, il raffreddamento attivo. La GPU è gestita in modo da evitare il surriscaldamento, incluso l'uso del thermal throttling per ridurre la frequenza della GPU in caso di necessità.

- Sincronizzazione con i Sensori: HoloLens si basa su sensori come telecamere di profondità e unità di misurazione inerziale (IMU) per comprendere l'ambiente reale e le interazioni dell'utente. La GPU deve lavorare in sincronia con questi sensori per garantire un tracciamento preciso e una resa grafica accurata.

- Ottimizzazione del Consumo Energetico: Data la capacità limitata della batteria di HoloLens, l'efficienza energetica è cruciale. La GPU è gestita in modo da consumare meno energia possibile quando viene visualizzato un contenuto meno impegnativo e di aumentare le prestazioni quando servono per gestire scene più complesse.

Hololens Lighting Methods

HoloLens utilizza diverse tecniche di illuminazione per rendere realistiche le esperienze di realtà mista (MR) e assicurare che gli oggetti virtuali siano ben integrati nell'ambiente fisico dell'utente. Ecco alcuni dei metodi di illuminazione utilizzati nelle HoloLens:

- **Ambient Lighting**:
  Ambient Lighting è una tecnica di illuminazione che rappresenta la luce ambientale presente nell'ambiente circostante senza una sorgente di luce direzionale specifica. Questa illuminazione uniforme contribuisce a evitare che gli oggetti sembrino completamente bui o privi di dettagli, fornendo una base di illuminazione per la scena.

  Unity Equivalent:
  Per gestire l'illuminazione ambientale in Unity per HoloLens, è possibile seguire questi passaggi:

  Impostazione dell'illuminazione: nell'Editor di Unity, è possibile configurare l'illuminazione ambientale nel pannello "Window > Rendering > Lighting" dove è possibile regolare il colore e l'intensità dell'illuminazione ambientale.

  Materiali e Shader: Assicurarsi che i materiali degli oggetti nella scena utilizzino uno shader che tiene conto dell'illuminazione ambientale, come lo shader standard di Unity.

  Hololens Runtime: quando viene eseguito il progetto su HoloLens, Unity si occupa di gestire l'illuminazione ambientale in base alle impostazioni configurate. L'illuminazione ambientale verrà riflessa sugli oggetti virtuali nella scena.

  GLSL Equivalent:

```
uniform vec3 ambientColor; // Colore ambientale
uniform float ambientIntensity; // Intensità ambientale

void main()
{
// Calcola il colore finale con l'illuminazione ambientale
vec3 finalColor = ambientColor * ambientIntensity;

// Altri calcoli come illuminazione diffusa, speculare, ecc...

// Imposta il colore del pixel
gl_FragColor = vec4(finalColor, 1.0);
}
```

  In questo esempio, ambientColor rappresenta il colore dell'illuminazione ambientale, e ambientIntensity rappresenta l'intensità dell'illuminazione ambientale. Il colore finale del pixel viene calcolato moltiplicando il colore ambientale per l'intensità ambientale. Questo contribuisce a fornire un'illuminazione uniforme su tutti gli oggetti nella scena.

- **Directional Lighting**:
  Directional Lighting è una tecnica di illuminazione che rappresenta la luce proveniente da una fonte di luce direzionale, come il sole, e si estende uniformemente in una particolare direzione all'interno della scena. Questo tipo di illuminazione è spesso utilizzato per simulare luce solare o altre fonti di luce direzionale.

  Unity Equivalent:
  Creazione di una Light: Per rappresentare l'illuminazione direzionale, è necessario creare una Light GameObject di tipo "Directional Light" nella tua scena. Questa luce simulerà la luce proveniente da una direzione specifica.

  Configurazione dell'illuminazione: Seleziona la Light GameObject e nel pannello Inspector, puoi configurare le sue proprietà, come la colore e l'intensità della luce. Puoi anche regolare la direzione da cui proviene la luce modificando la sua rotazione.

  Materiali e Shader: Bisogna assicurarsi che i materiali degli oggetti nella scena utilizzino uno shader che tiene conto dell'illuminazione direzionale, come lo shader standard di Unity.

  HoloLens runtime: Quando esegui il tuo progetto su HoloLens, Unity gestirà l'illuminazione direzionale in base alle impostazioni configurate. La luce direzionale influenzerà l'aspetto degli oggetti virtuali nella scena.

  GLSL Equivalent:

```
uniform vec3 lightDirection; // Direzione della luce
uniform vec3 lightColor; // Colore della luce
uniform float lightIntensity; // Intensità della luce

void main()
{
// Calcola il vettore della direzione della luce
vec3 lightDir = normalize(-lightDirection);

// Calcola il termine di illuminazione diffusa
float diff = max(dot(normal, lightDir), 0.0);

// Calcola il colore finale con l'illuminazione direzionale
vec3 finalColor = diff * lightColor * lightIntensity;

// Altri calcoli come illuminazione speculare, ecc...

// Imposta il colore del pixel
gl_FragColor = vec4(finalColor, 1.0);
}
```

  In questo esempio, lightDirection rappresenta la direzione della luce direzionale, lightColor rappresenta il colore della luce, e lightIntensity rappresenta l'intensità della luce. Il colore finale del pixel viene calcolato considerando la direzione della luce e la normale della

superficie dell'oggetto. Questo contribuisce a fornire un'illuminazione direzionale realistica sugli oggetti.

In Unity per HoloLens, puoi utilizzare shader simili per personalizzare l'aspetto dei materiali degli oggetti virtuali e applicare l'illuminazione direzionale in base alle impostazioni della scena Unity.

- **Shadows and Shading**:
  Shadows and Shading si riferisce ad una tecnica di illuminazione che consente di calcolare e proiettare ombre realistiche sugli oggetti virtuali in una scena tridimensionale. Questa tecnica è fondamentale per rendere credibili gli oggetti virtuali in un ambiente fisico e migliorare la percezione della profondità.

  Unity Equivalent:
  Unity offre diverse opzioni per gestire ombre e shading:
  Real-time Shadows: Unity supporta ombre dinamiche generate in tempo reale da luci puntiformi e direzionali. Puoi abilitare questa funzione nelle impostazioni delle luci all'interno del pannello Lighting.

  Shadow Casting: Per far sì che un oggetto getti ombre, devi assicurarti che il suo materiale sia configurato per gettare ombre e che la luce abbia l'opzione "Cast Shadows" abilitata.

  Shader Customization: Puoi personalizzare gli shader dei materiali per tener conto dell'illuminazione e delle ombre nella scena. Unity offre molte funzionalità per modificare il comportamento degli shader in modo da calcolare ombre e riflessi.

  GLSL Equivalent:

```
// Uniform per la mappa delle ombre
uniform sampler2D shadowMap;

void main()
{
// Calcola la coordinata di proiezione delle ombre
vec4 shadowCoord = ShadowMatrix * vec4(fragmentPosition, 1.0);
// Calcola la distanza dalla luce
float depth = texture(shadowMap, shadowCoord.xy).r;
// Applica l'illuminazione solo se il punto è illuminato
if (shadowCoord.z > depth + 0.001)
{
// Calcolo dell'illuminazione diffusa e speculare
// ...
// Imposta il colore finale del pixel
gl_FragColor = vec4(finalColor, 1.0);
```

```
}
else
{
// Il punto è ombreggiato
gl_FragColor = vec4(0.0, 0.0, 0.0, 1.0);
}
}
```

In questo esempio GLSL, il codice verifica se il punto è illuminato o ombreggiato confrontando la coordinata di proiezione delle ombre con i dati nella mappa delle ombre. Se il punto è illuminato, calcola l'illuminazione diffusa e speculare, altrimenti imposta il colore del pixel su nero per rappresentare l'ombra.

Questi sono esempi semplificati e nei progetti reali, la gestione delle ombre e dello shading può essere molto più complessa e coinvolgere tecniche avanzate come shadow mapping, shadow volumes o ray tracing.

- **Lighting Texture Mapping**:
  Lighting Texture Mapping è una tecnica utilizzata nell'illuminazione avanzata per applicare informazioni di illuminazione dettagliate agli oggetti virtuali all'interno di una scena 3D. Queste informazioni possono includere dati sull'illuminazione ambientale, direzionale e altri effetti luminosi, consentendo di ottenere un aspetto più realistico e dettagliato per gli oggetti virtuali.

  Unity Equivalent:
  In Unity, è possibile applicare l'illuminazione basata su texture in vari modi:

  Lightmaps: Unity consente di creare lightmaps, che sono texture precalcolate contenenti informazioni sull'illuminazione. Queste texture vengono applicate ai materiali degli oggetti e forniscono dettagli di illuminazione aggiuntivi.

  Custom Shaders: Puoi creare shader personalizzati in Unity che utilizzano mappe di illuminazione per influenzare l'aspetto degli oggetti. Ad esempio, puoi utilizzare una mappa di illuminazione speculare per controllare la riflessione speculare su una superficie.

  GLSL Equivalent:

```
uniform sampler2D specularMap; // Mappa di illuminazione speculare

void main()
{
// Esempio di calcolo della riflessione speculare
vec3 specular = texture(specularMap, texCoord).rgb;
// Calcolo della riflessione speculare effettiva
```

```
vec3 finalSpecular = specular * objectColor; // Moltiplica per il colore
dell'oggetto

// Altri calcoli come illuminazione diffusa, ecc...

// Imposta il colore finale del pixel
gl_FragColor = vec4(finalSpecular, 1.0);
}
```

In GLSL, è possibile implementare l'illuminazione basata su texture all'interno degli shader personalizzati. Supponiamo di avere una mappa di illuminazione speculare che determina la riflessione speculare su un oggetto.
In questo esempio GLSL, specularMap rappresenta la mappa di illuminazione speculare. La texture viene campionata in base alle coordinate della texture texCoord, e il valore risultante viene utilizzato per calcolare la riflessione speculare effettiva sull'oggetto.

Le mappe di illuminazione possono contenere molte altre informazioni, come mappe di normali, mappe di ambient occlusion e altro ancora, che possono essere utilizzate per migliorare l'aspetto degli oggetti virtuali nella scena.

- **Global Illumination**:
  Global Illumination (GI) è una tecnica avanzata di illuminazione utilizzata per simulare il comportamento della luce nell'ambiente 3D, compresa la riflessione della luce tra le superfici e gli effetti di luce indiretta. Questo approccio mira a creare scene virtuali più realistiche e naturali.

  Unity Equivalent:
  Unity offre una serie di soluzioni per Global Illumination, tra cui:

  Real-time Global Illumination: Unity ha il suo sistema di Global Illumination in tempo reale chiamato Enlighten. Con questa soluzione, è possibile calcolare in tempo reale l'illuminazione indiretta e i riflessi in una scena.

  Precomputed Global Illumination: Unity supporta il calcolo di Global Illumination in modalità precompilata. In questa modalità, l'illuminazione viene calcolata in anticipo e i dati vengono salvati in lightmaps o light probes.

  Rimbalzo di luce: Unity permette di configurare il rimbalzo di luce tra oggetti nella scena, creando un effetto di illuminazione indiretta.

  GLSL Equivalent:

```
void main()
{
```

```
// Calcolo dell'illuminazione diretta (diffusa e speculare)
vec3 directLighting = CalculateDirectLighting();

// Calcolo dell'illuminazione globale indiretta
vec3 indirectLighting = CalculateIndirectLighting();

// Combinazione dell'illuminazione diretta e globale
vec3 finalColor = directLighting + indirectLighting;

// Imposta il colore finale del pixel
gl_FragColor = vec4(finalColor, 1.0);
}
```

In GLSL, il supporto per il Global Illumination varia a seconda della piattaforma e dell'hardware. Tuttavia, è possibile simulare alcune delle caratteristiche del Global Illumination attraverso shader personalizzati.
Nell'esempio GLSL sopra, CalculateIndirectLighting() rappresenta il calcolo semplificato dell'illuminazione globale indiretta. Questo potrebbe coinvolgere la riflessione della luce da altre superfici o il calcolo di illuminazione indiretta da fonti di luce ambientale.
Tieni presente che la simulazione del Global Illumination in GLSL è molto più complessa rispetto all'uso di soluzioni integrate come Enlighten in Unity. I motori di gioco come Unity forniscono strumenti specializzati per gestire il Global Illumination in modo efficiente.

- **Dynamic Lighting**:
  Dynamic Lighting è una tecnica di illuminazione che coinvolge fonti di luce che possono muoversi o cambiare nel tempo, creando effetti di illuminazione realistici e dinamici. Questo è in contrasto con l'illuminazione statica, che è precalcolata e rimane costante durante l'esecuzione della scena.

  Unity Equivalent:
  In Unity, è possibile implementare Dynamic Lighting utilizzando luci che possono essere controllate e modificate durante l'esecuzione del gioco. Puoi utilizzare luci puntiformi o spot, e variare la loro posizione, intensità e colore in base alle necessità della scena. Ad esempio, puoi far lampeggiare una luce quando un evento specifico si verifica o farla seguire un oggetto in movimento.

  GLSL Equivalent:

```
uniform vec3 lightPosition; // Posizione della luce
uniform float lightIntensity; // Intensità della luce

void main()
{
// Calcola la direzione della luce
```

```
vec3 lightDirection = normalize(lightPosition - vertexPosition);

// Calcola l'illuminazione diffusa
float diff = max(dot(normal, lightDirection), 0.0);
vec3 diffuseColor = diff * lightIntensity * lightColor;

// Calcola il colore finale del pixel
vec3 finalColor = objectColor * diffuseColor;

// Imposta il colore del pixel
gl_FragColor = vec4(finalColor, 1.0);
}
```

In GLSL, puoi implementare l'illuminazione dinamica all'interno degli shader personalizzati. Ad esempio, puoi modificare la posizione o l'intensità di una luce direzionale in base a variabili o condizioni specifiche all'interno degli shader. Tuttavia, tieni presente che l'implementazione precisa può variare notevolmente a seconda delle esigenze specifiche della scena e degli shader.
In questo esempio, lightPosition rappresenta la posizione dinamica della luce, e lightIntensity rappresenta l'intensità che cambia nel tempo. Il colore finale del pixel viene calcolato in base a queste variabili, consentendo di ottenere effetti di illuminazione dinamica all'interno dello shader.

- **Real-World Lighting**:
  Real-World Lighting è una tecnica che mira a simulare l'illuminazione dell'ambiente fisico in cui si trova l'utente. Questo significa che l'illuminazione virtuale nell'app HoloLens si adatta e rispecchia l'illuminazione del mondo reale circostante. Questa tecnica contribuisce a creare esperienze di realtà aumentata più immersive e realistiche.

  Unity Equivalent:
  L'obiettivo principale di Real-World Lighting in Unity per HoloLens è quello di integrare l'illuminazione reale dell'ambiente con l'illuminazione virtuale. Per farlo, puoi seguire questi passaggi:

  Utilizzo di sensori di luce: HoloLens è dotato di sensori di luce ambientale che possono rilevare l'intensità luminosa dell'ambiente circostante. Puoi utilizzare questi dati per regolare dinamicamente l'illuminazione nella tua app.

  Applicazione dell'illuminazione dinamica: In base ai dati dei sensori di luce, puoi regolare l'intensità e il colore delle luci virtuali nella tua scena Unity. Ad esempio, se l'ambiente è più luminoso, puoi abbassare l'intensità delle luci virtuali per farle sembrare più realistiche.

  Ombre e riflessioni: Puoi anche calcolare ombre e riflessioni basate sull'illuminazione reale per gli oggetti virtuali nella tua app, in modo che si integrino meglio nell'ambiente.

  Rilevamento della superficie: Per ancorare oggetti virtuali in superfici reali, puoi utilizzare la funzione di rilevamento delle superfici di HoloLens. Questo consente di posizionare gli oggetti virtuali in modo coerente con l'ambiente reale e di adattare dinamicamente l'illuminazione.

GLSL Equivalent:

L'implementazione di Real-World Lighting in GLSL richiede l'uso di dati dell'illuminazione reale provenienti da sensori esterni o dall'ambiente circostante, e quindi l'adattamento dell'illuminazione virtuale in base a questi dati.