



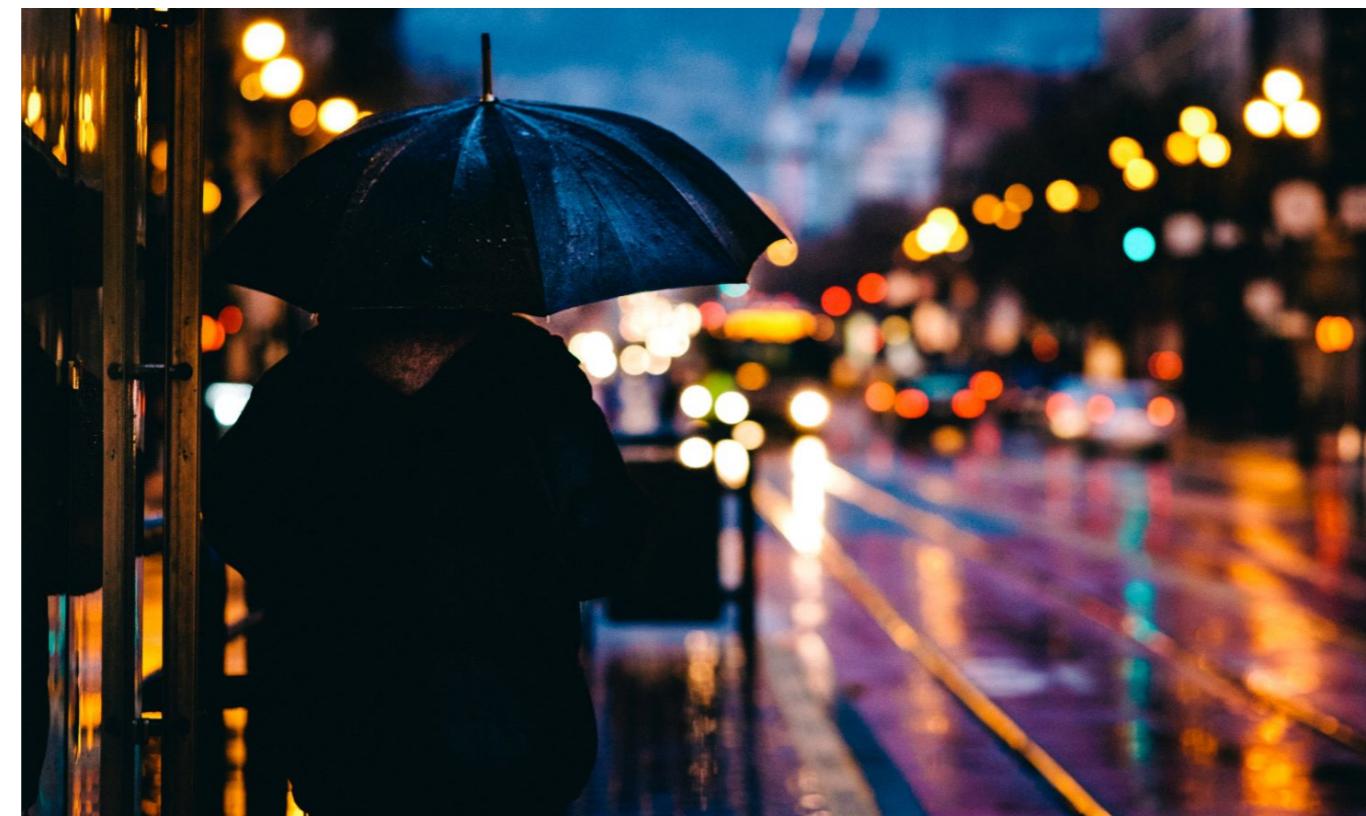
Depth of Field: implementazione con Three.js

Elaborato svolto per il corso Computer
Graphics & 3D
A cura di Alessandro Arezzo

IL PROGETTO

- **Obiettivo del progetto:** implementare l'effetto del Depth Of Field (Campo di Profondità) applicandolo ad una scena 3D.
 - **Depth of Field:** effetto in cui solo gli oggetti entro un certo intervallo di distanze dalla camera appaiono a fuoco mentre tutti gli altri risultano sfocati.

Nelle immagini reali l'effetto è dovuto a caratteristiche fisiche delle lenti, in computer graphics invece deve essere simulato.



IL PROGETTO: le sue fasi

1. Studio preliminare del Depth of Field

- Caratteristiche che generano l'effetto ed i suoi contesti applicativi
- Tecniche per simulare l'effetto in Computer Graphics

2. Implementazione: utilizzando WebGL e Three js

- Ideazione di una scena 3D
- Implementazione di una tecnica di simulazione del Depth of Field
- Aggiunta interfaccia utente per la gestione dei parametri dell'effetto e per muoversi sulla scena

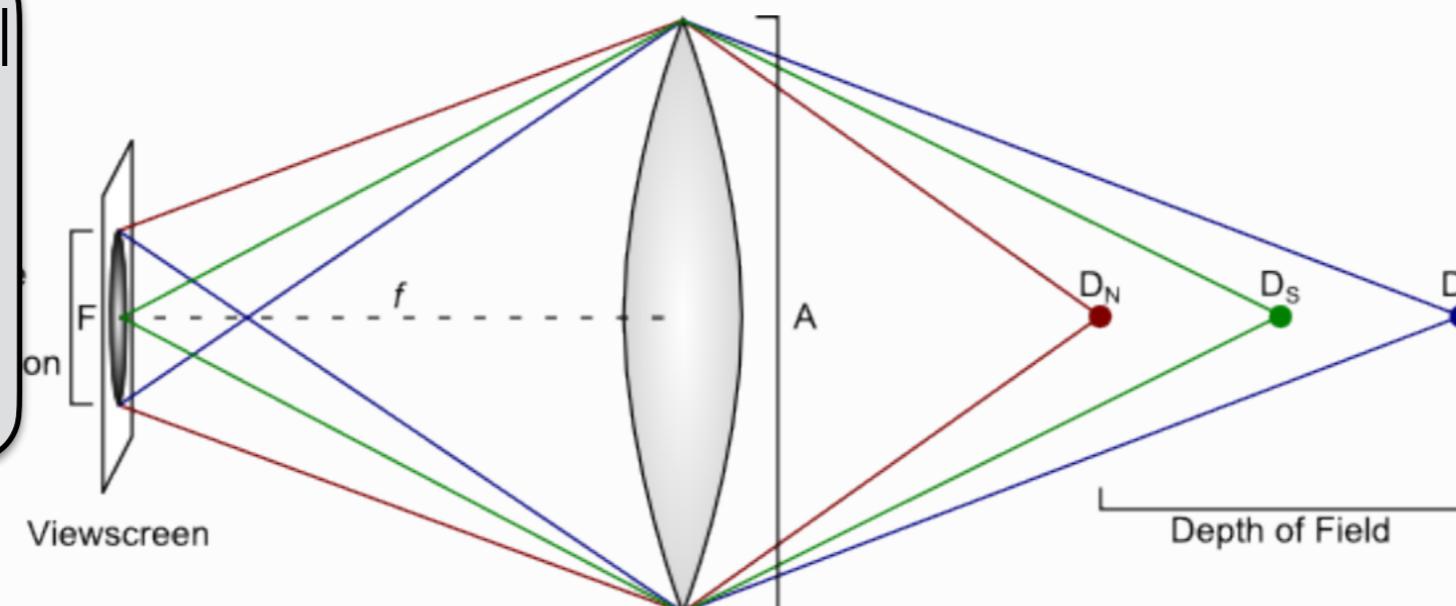


Depth of Field (DoF)

- Effetto in cui solo gli oggetti entro un certo intervallo di distanze dalla camera detto **piano a fuoco** appaiono nitidi mentre tutti gli altri risultano sfocati.
- La lente di una camera consente alla luce di passare attraverso la pellicola, ma solo i raggi di luce provenienti da una sorgente ad una certa distanza (piano a fuoco) convergono in un solo punto.
- Tutti gli altri punti sono mappati in una regione del film detta **Circle of Confusion (CoC)**.

$$CoC = \left| A \cdot \frac{F \cdot (P - D)}{D \cdot (P - F)} \right|$$

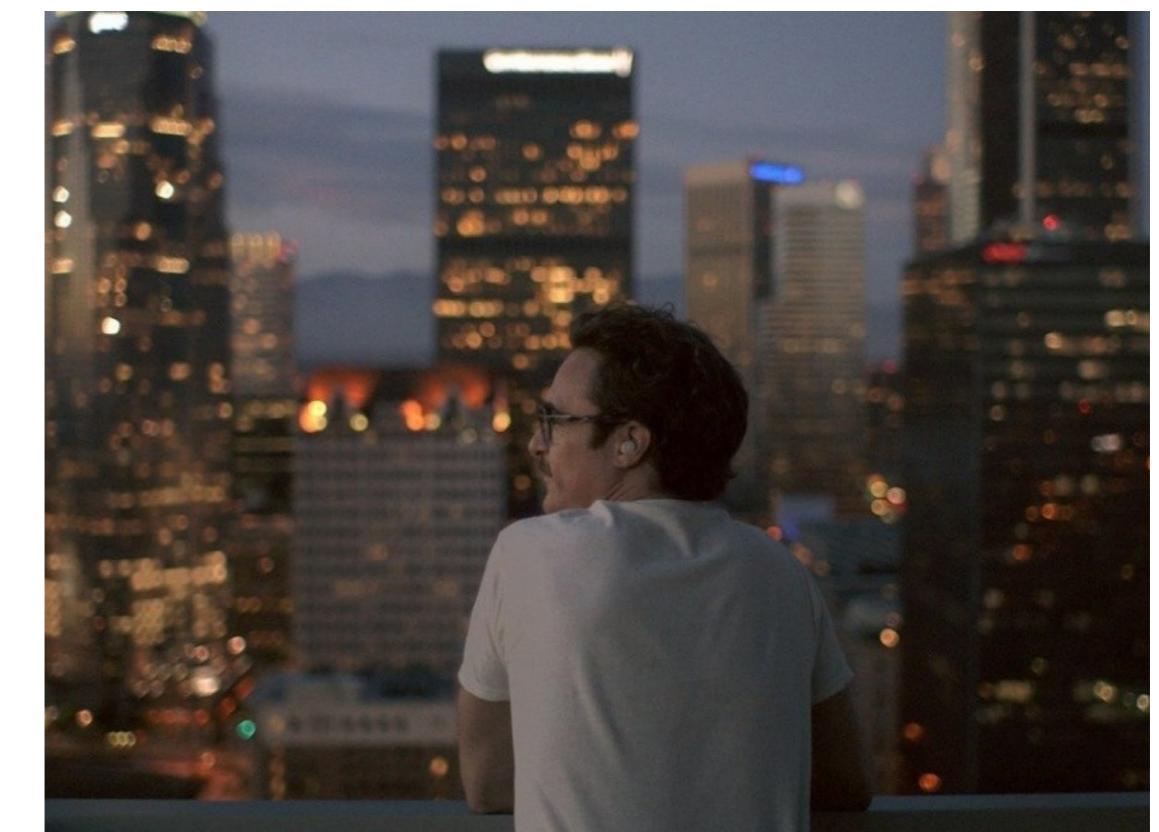
- A: apertura
- F: lunghezza focale
- D: distanza oggetto
- P: piano a fuoco





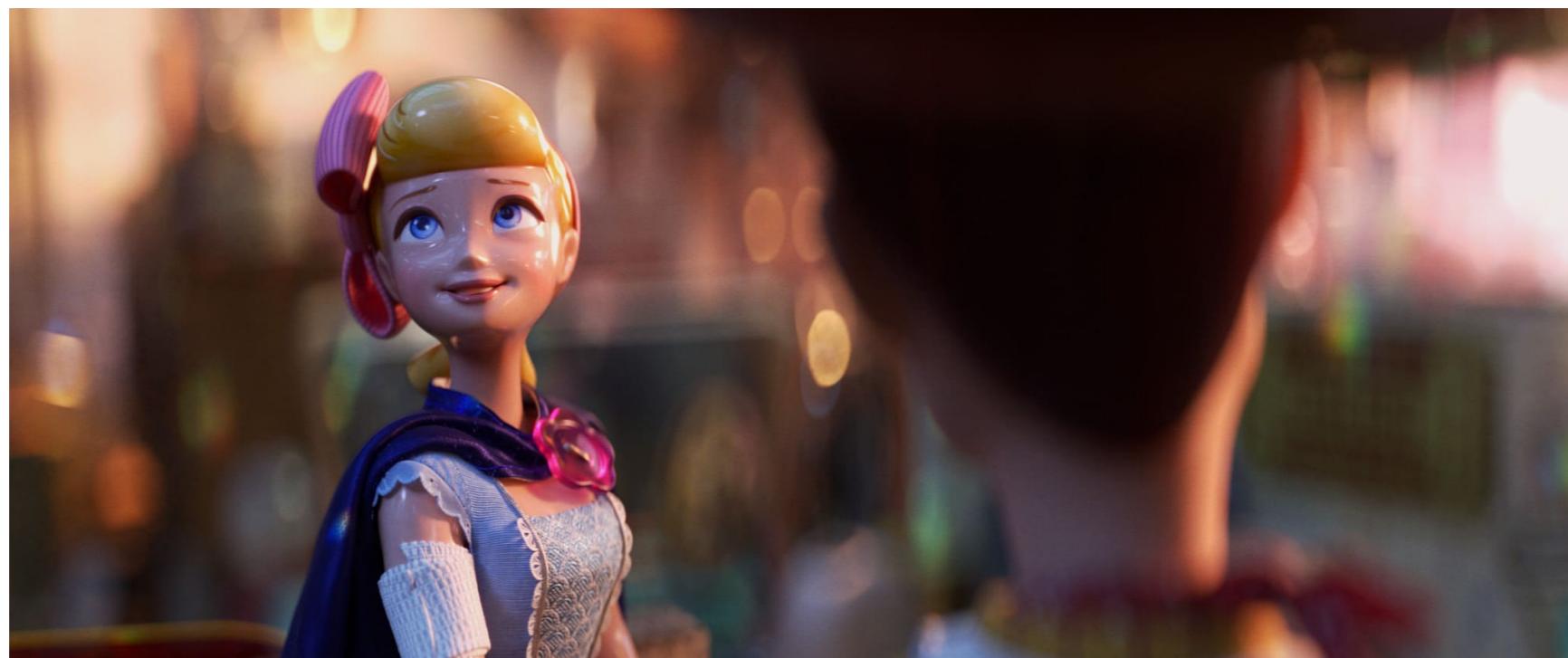
Depth of Field (DoF)

- L'effetto viene usato in fotografia e nell'industria cinematografica.
- Ha lo scopo di indirizzare l'attenzione dello spettatore su certi elementi della scena e dare un miglior senso di profondità.



Depth of Field (DoF)

- In Computer Graphics si utilizza un modello di camera pinhole ideale che proietta sul film virtuale usando una lente di dimensione zero: la luce viaggia attraverso un unico percorso dalla scena al film.
- L'effetto di sfocatura degli oggetti non a fuoco deve essere approssimato.





Tecnica Z-buffer

- Renderizzare la scena salvando sia il buffer di colore che quello di profondità durante il rendering.
- Applicare una fase di post processing: utilizzando il buffer di profondità (z-buffer) per calcolare il CoC di ciascun pixel.
 - Sia z la coordinata di profondità del pixel nello z-buffer:

$$D = \frac{-z_{far} \cdot z_{near}}{z \cdot (z_{far} - z_{near}) - z_{far}}$$

- Calcolo del CoC del pixel:

$$CoC = |A \cdot \frac{F \cdot (P - D)}{D \cdot (P - F)}|$$

Si può poi usare il CoC del pixel in modo da far sì che questo contribuisca al calcolo del colore dei suoi punti adiacenti





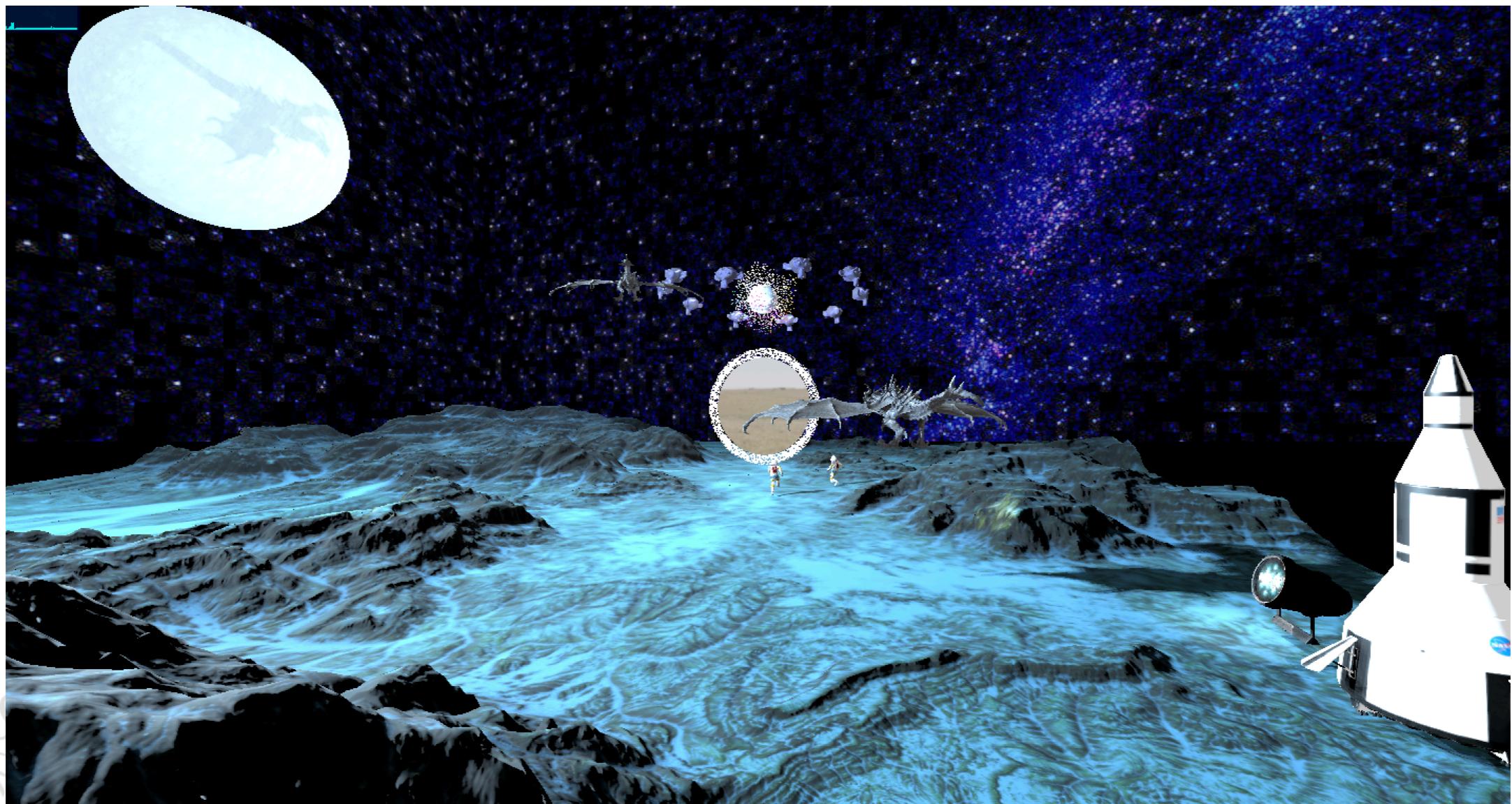
Implementazione

- Il progetto prevede un'implementazione dell'effetto DoF ad una scena di Computer Graphics che applica la tecnica z-buffer utilizzando il linguaggio WebGL e la libreria ThreeJS.
 - WebGL: libreria grafica per il web basata sul linguaggio OpenGL ES 2.0 .
 - ThreeJS: libreria javascript che consente di accedere con maggior semplicità alle risorse di WebGL.



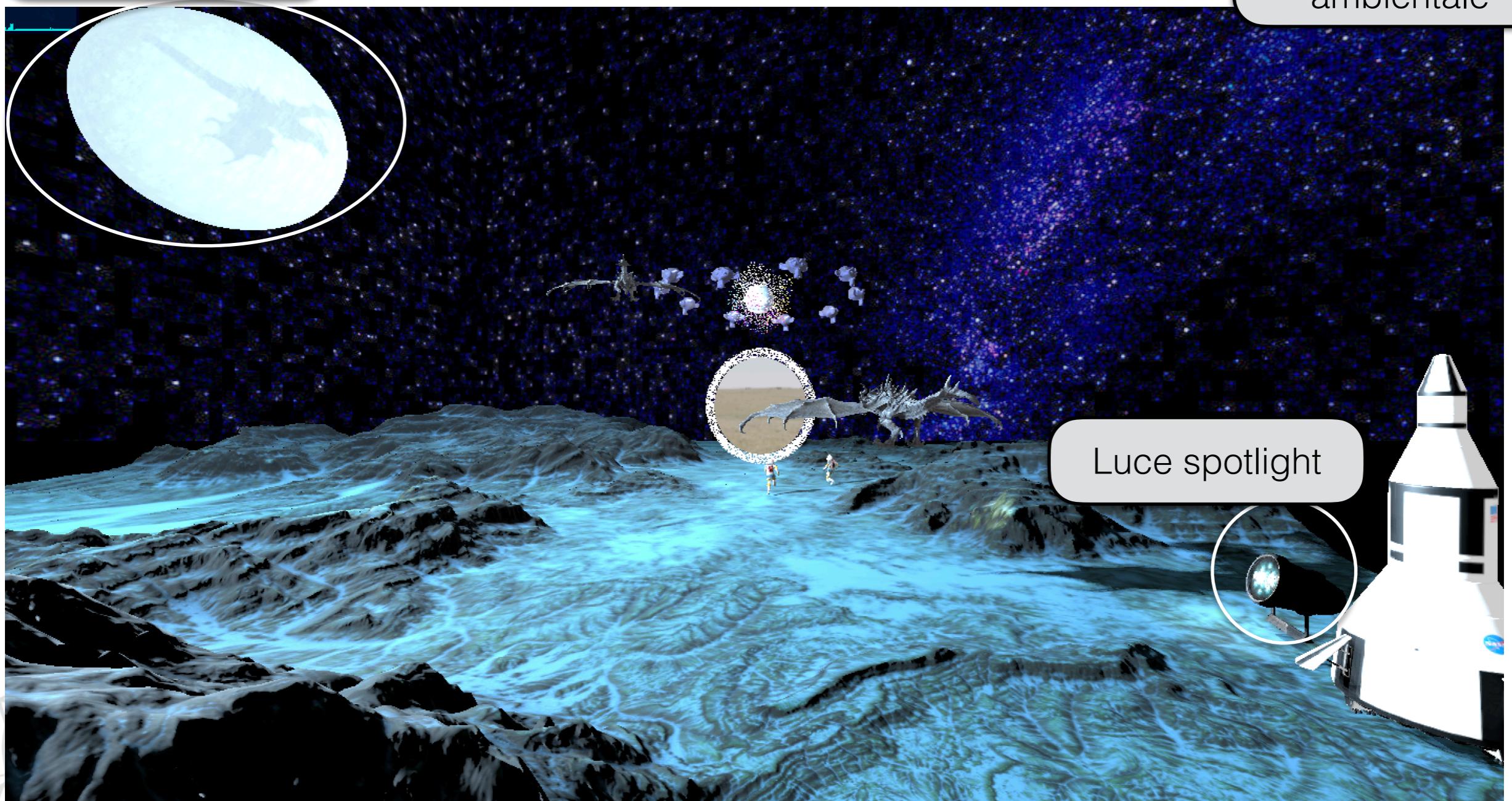
Scena 3D

- **Obiettivo:** simulare in computer graphics uno scenario che sembrasse simile a quello di una possibile sequenza cinematografica.



Scena 3D: luci

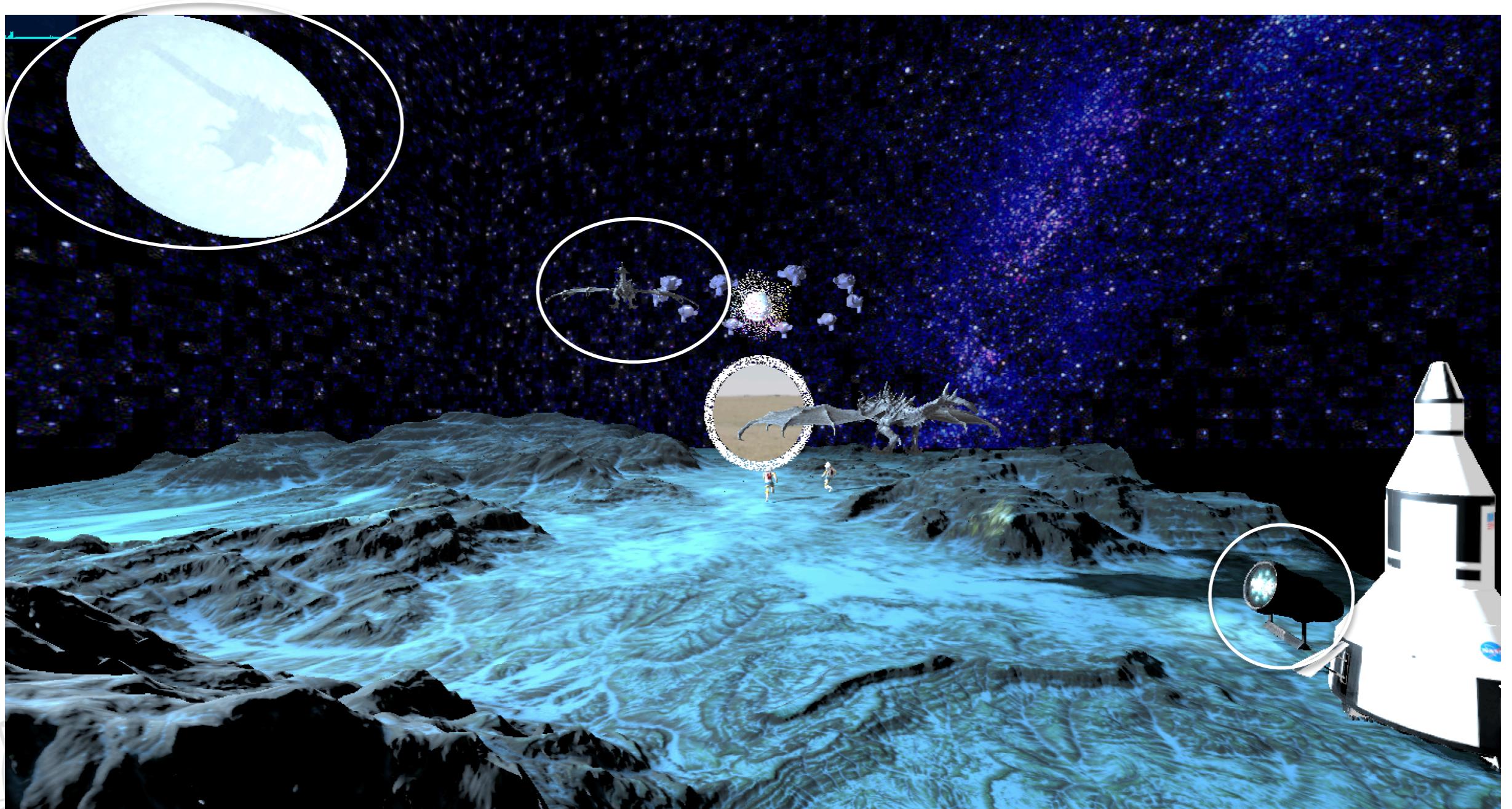
Luce puntuale



Luce
ambientale

Luce spotlight

Scena 3D: ombre





Effect Composer

- Nel progetto si utilizza la classe **EffectComposer** di Three JS che consente di applicare una sequenza di operazioni di post processing.
- Fasi:
 1. Rendering della scena
 2. Post Processing
 - Vertex shader
 - CoC fragment shader
 - DoF fragment shader



Vertex shader

```
1      varying vec2 vUv;  
2  
3  ↑  void main() {  
4      vUv = uv;  
5      gl_Position = projectionMatrix * modelViewMatrix * vec4(position, 1);  
6 }
```

- Uv contiene le coordinate delle texture per vertice provenienti dalla fase di rendering precedente
- Shader applica semplicemente le trasformazioni geometriche e di proiezione.



Fragment shader CoC

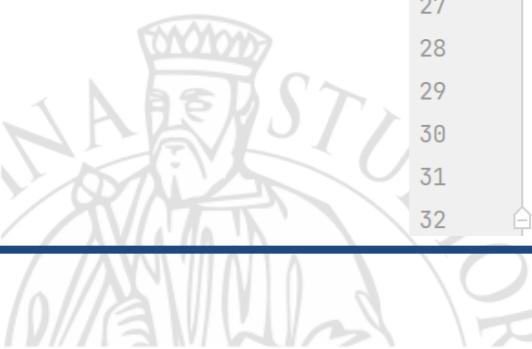
```

1  #include <packing>
2  uniform sampler2D tDepth;
3
4  // camera parameters
5  uniform float cameraNear;
6  uniform float cameraFar;
7  uniform float focalDepth;
8  uniform float focalLength;
9  uniform float aperture;
10
11 varying vec2 vUv;
12
13 ⚪ float readDepth( sampler2D depthSampler, vec2 coord ) {
14     float fragCoordZ = texture2D( depthSampler, coord ).x;
15     float viewZ = perspectiveDepthToViewZ( fragCoordZ, cameraNear, cameraFar );
16     return viewZToOrthographicDepth( viewZ, cameraNear, cameraFar );
17 }
18
19 ⚪ void main() {
20     // calculate distance of the object respect to the camera
21     float depth = readDepth( tDepth, vUv );
22     float distance = - cameraFar * cameraNear / (depth * (cameraFar - cameraNear) - cameraFar) * 1000.0;
23
24     // compute Circle of Confusion (CoC) for the current pixel
25     float focalDepth = focalDepth * 1000.0;
26     float CoC = aperture * (focalLength * (distance - focalDepth))
27     / (distance * (focalDepth - focalLength));
28     CoC = clamp(CoC, -1.0, 1.0);
29
30     /* assign CoC to blue component */
31     gl_FragColor.b = abs(CoC);
32 }
```

Definizione variabili uniformi

Calcolo distanza vertice dalla camera

Calcolo ed assegnazione del CoC del vertice



Fragment shader CoC

```
1 #include <packing>
2 uniform sampler2D tDepth;
3
4 // camera parameters
5 uniform float cameraNear;
6 uniform float cameraFar;
7 uniform float focalDepth;
8 uniform float focalLength;
9 uniform float aperture;
10
11 varying vec2 vUv;
...

```

- Definizione delle variabili uniformi:
 - Per ottenere informazioni sulla profondità delle texture nel rendering (tDepth)
 - Relative alla camera (cameraNear, cameraFar, focalLength, aperture)
 - Distanza del piano a fuoco dalla camera (focalDepth)



Fragment shader CoC

```
...
13  ① float readDepth( sampler2D depthSampler, vec2 coord ) {
14      float fragCoordZ = texture2D( depthSampler, coord ).x;
15      float viewZ = perspectiveDepthToViewZ( fragCoordZ, cameraNear, cameraFar );
16      return viewZToOrthographicDepth( viewZ, cameraNear, cameraFar );
17  }
18
19  ① void main() {
20      // calculate distance of the object respect to the camera
21      float depth = readDepth( tDepth, vUv );
22      float distance = - cameraFar * cameraNear / (depth * (cameraFar - cameraNear) - cameraFar) * 1000.0;
...

```

- Calcolo della distanza del vertice dalla camera
 - Calcolo della profondità del vertice dallo z-buffer (usando `tDepth` e coordinate del vertice)
 - Applicazione della formula: $distance = \frac{-z_{far} \cdot z_{near}}{z \cdot (z_{far} - z_{near}) - z_{far}}$



Fragment shader CoC

```
24
25     // compute Circle of Confusion (CoC) for the current pixel
26     float focalDepth = focalDepth * 1000.0;
27     float CoC = aperture * (focalLength * (distance - focalDepth))
28         / (distance * (focalDepth - focalLength));
29     CoC = clamp(CoC, -1.0, 1.0);
30
31     /* assign CoC to blue component */
32     gl_FragColor.b = abs(CoC);
```

- Calcolo del CoC del vertice
 - Applicazione della formula $CoC = |A \cdot \frac{F \cdot (P - D)}{D \cdot (P - F)}|$
 - Assegnazione del valore di CoC del vertice alla coordinata colore del blu.

Fragment shader DoF

```

3   uniform sampler2D tDiffuse;
4   uniform sampler2D tOriginal;
5   uniform float blurSize;
6   uniform float heightTex;
7   uniform float widthTex;
8   uniform bool applyEffect;
9   varying vec2 vUv;
10
11  float getWeight(float dist, float maxDist){
12      return 1.0 - dist/maxDist;
13  }
14
15  void main() {
16      vec3 sourceColor = texture2D(tOriginal, vUv).rgb;
17      float minCoC = 0.1;
18      const float max_iterations = 100.;
19      float blur = blurSize + 1.;
20      float CoC = texture2D(tDiffuse, vUv).b;
21      if(applyEffect){
22          if (CoC > minCoC){
23              vec3 colorBlurred = vec3(0.0);
24              float weightColorSum = 0.0;
25              for (float i=0.; i<max_iterations; i++){
26                  if (i > blur*2.) break;
27                  for (float j=0.; j<max_iterations; j++){
28                      if (j > blur*2.) break;
29                      vec2 dir = vec2(i-blurSize, j-blurSize) * vec2(widthTex, heightTex);
30                      float dist = length(dir);
31                      if (dist > blurSize){
32                          continue;
33                      }
34                      float curCoC = texture2D(tDiffuse, dir + vUv).b;
35                      if (curCoC > minCoC) {
36                          float currentWeight = getWeight(dist, blurSize);
37                          weightColorSum += currentWeight;
38                          colorBlurred += currentWeight * texture2D(tOriginal, dir + vUv).rgb;
39                      }
40                      colorBlurred /= weightColorSum;
41                      gl_FragColor.rgb = mix(sourceColor, colorBlurred, CoC);
42                      gl_FragColor.a = 1.0;
43                  }
44              }
45          } else {
46              gl_FragColor.rgb = sourceColor;
47          }
48      }
    }
```

} Definizione variabili uniformi

} Calcolo del colore sfumato per vertici non a fuoco

} Assegnazione del colore per vertici non a fuoco

} Assegnazione del colore per vertici sul piano a fuoco



Fragment shader DoF

```
3 uniform sampler2D tDiffuse;  
4 uniform sampler2D tOriginal;  
5 uniform float blurSize;  
6 uniform float heightTex;  
7 uniform float widthTex;  
8 uniform bool applyEffect;  
...
```

- Definizione delle variabili uniformi:
 - Per ottenere informazioni riguardo il colore originale ed il CoC del vertice (tDiffuse, tOriginal)
 - Relative all'effetto da applicare (blurSize, applyEffect)
 - Riguardanti le dimensioni della finestra (heightTex, widthTex)



Fragment shader DoF

```

14     vec3 sourceColor = texture2D(tOriginal, vUv).rgb;
15
16     float minCoC = 0.1;
17
18     float CoC = texture2D(tDiffuse, vUv).b;
19     if(applyEffect){
20         if (CoC > minCoC){
21             ...
22         }
23     }
24     else {
25         gl_FragColor.rgb = sourceColor;
26     }
27
28     else{
29         gl_FragColor.rgb = sourceColor;
30     }
31
32 }
```

- Si legge il CoC del vertice (dalla componente blu di colore)
- Al vertice viene assegnato il suo colore originale se:
 - L'effetto non deve essere applicato
 - Il CoC è sufficientemente piccolo da poter considerare il vertice come appartenente al piano a fuoco

Fragment shader DoF

```

10  ① ↑ ⚡ float getWeight(float dist, float maxDist){
11      return 1.0 - dist/maxDist;
12  }
...
20  if (CoC > minCoC){
21      vec3 colorBlurred = vec3(0.0);
22      float weightColorSum = 0.0;
23      for (float i=0.; i<max_iterations; i++){
24          if (i > blur*2.) break;
25          for (float j=0.; j<max_iterations; j++){
26              if (j > blur*2.) break;
27              vec2 dir = vec2(i-blurSize, j-blurSize) * vec2(widthTex, heightTex);
28              float dist = length(dir);
29              if (dist > blurSize){
30                  continue;
31              }
32              float curCoC = texture2D(tDiffuse, dir + vUv).b;
33              if (curCoC > minCoC) {
34                  float currentWeight = getWeight(dist, blurSize);
35                  weightColorSum += currentWeight;
36                  colorBlurred += currentWeight * texture2D(tOriginal, dir + vUv).rgb;
37              }
38      }
39  }
}

```

- Se l'effetto deve essere applicato ed il CoC del vertice è sufficientemente grande il colore viene sfumato.
- Viene calcolata una somma pesata del colore dei vertici appartenenti ad un intorno circolare.
- Peso è inversamente proporzionale alla distanza dal vertice.



Fragment shader DoF

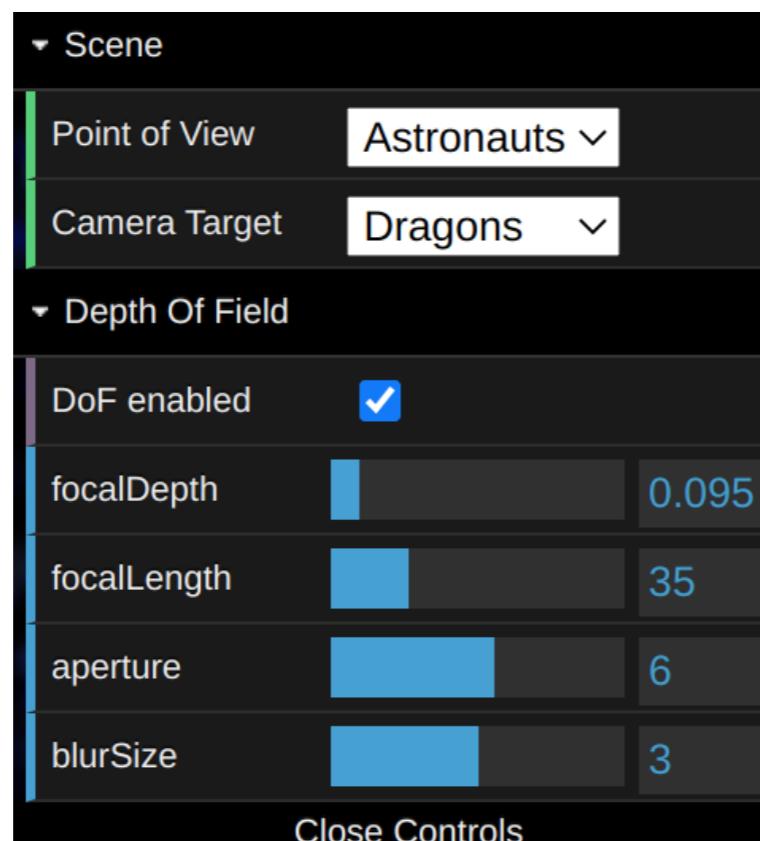
```
20 if (CoC > minCoC){  
    ...  
40     colorBlurred /= weightColorSum;  
41     gl_FragColor.rgb = mix(sourceColor, colorBlurred, CoC);  
42     gl_FragColor.a = 1.0;  
43 }  
44 }
```

- Dalla somma dei colori ne viene calcolata una media e questa viene unita al colore originale del vertice per originare il colore sfumato.



Interfaccia utente

- Utente può interagire con il progetto modificando:
 - Parametri relativi all'effetto (lunghezza focale, apertura, ampiezza della sfocatura, distanza del piano a fuoco)
 - Posizione della camera (punto di vista, direzione della camera)



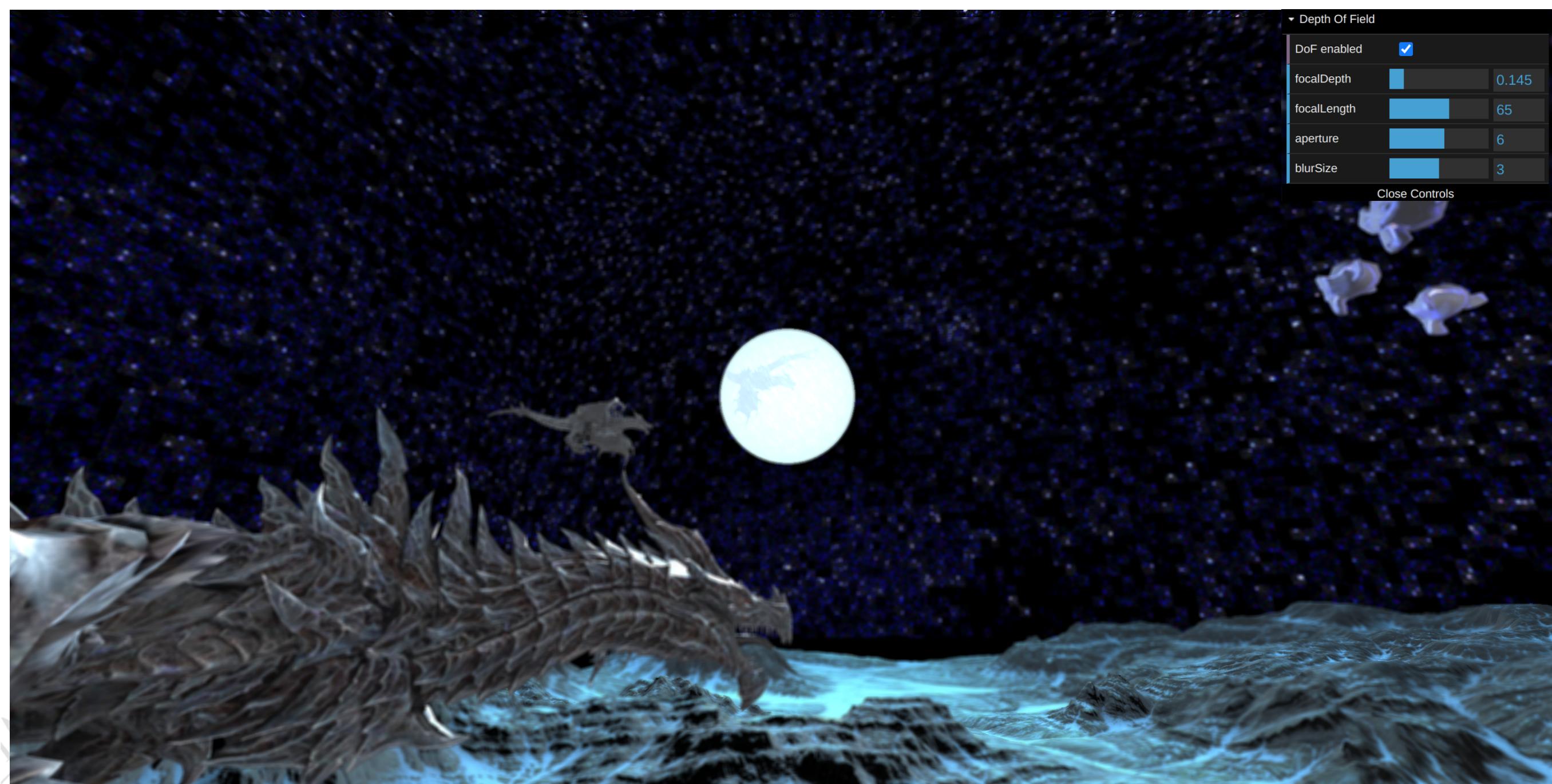


Esempi 1/3



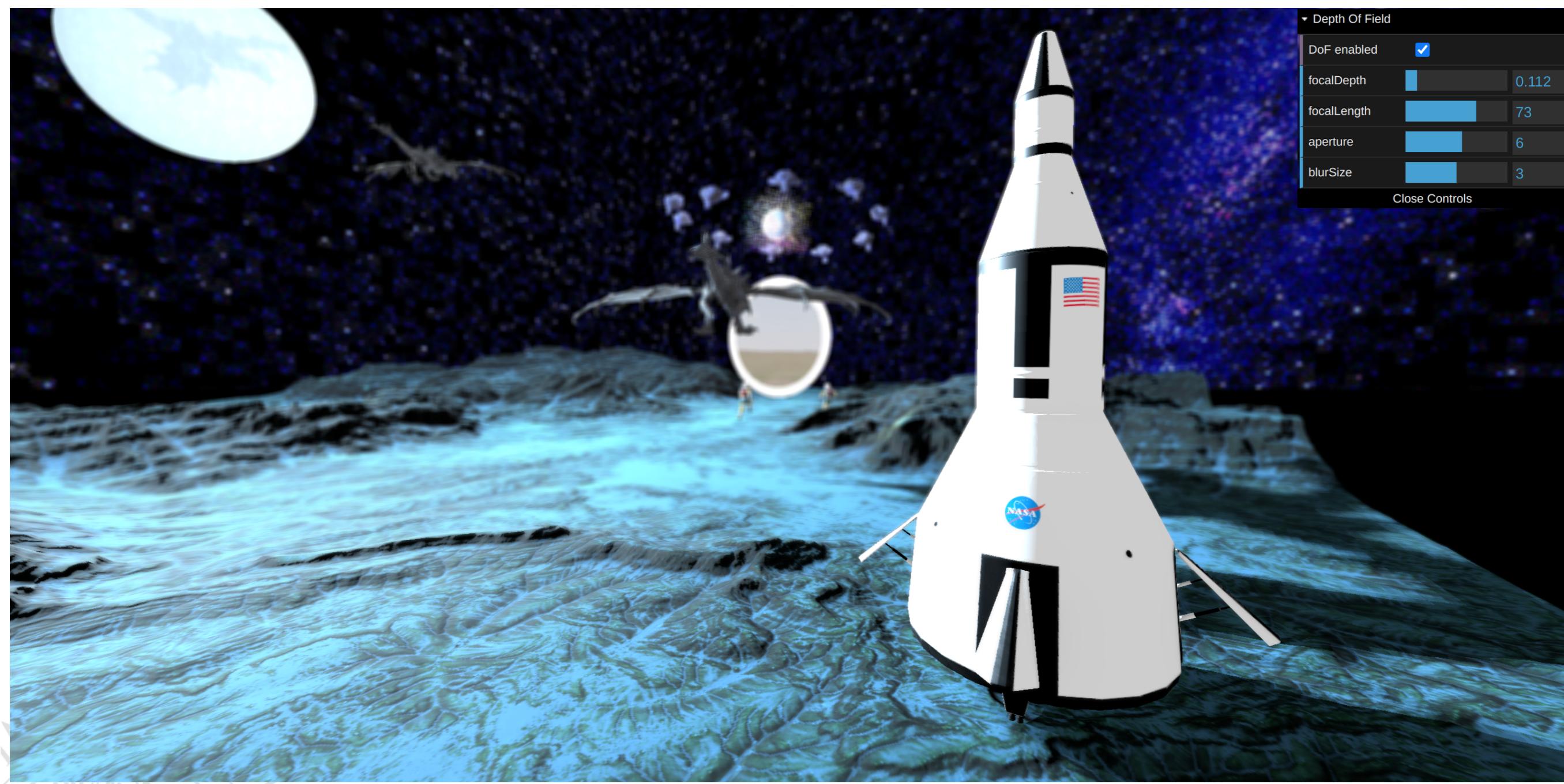


Esempi 2/3





Esempi 3/3



FINE PRESENTAZIONE

Presentazione realizzata da:
Alessandro Arezzo

