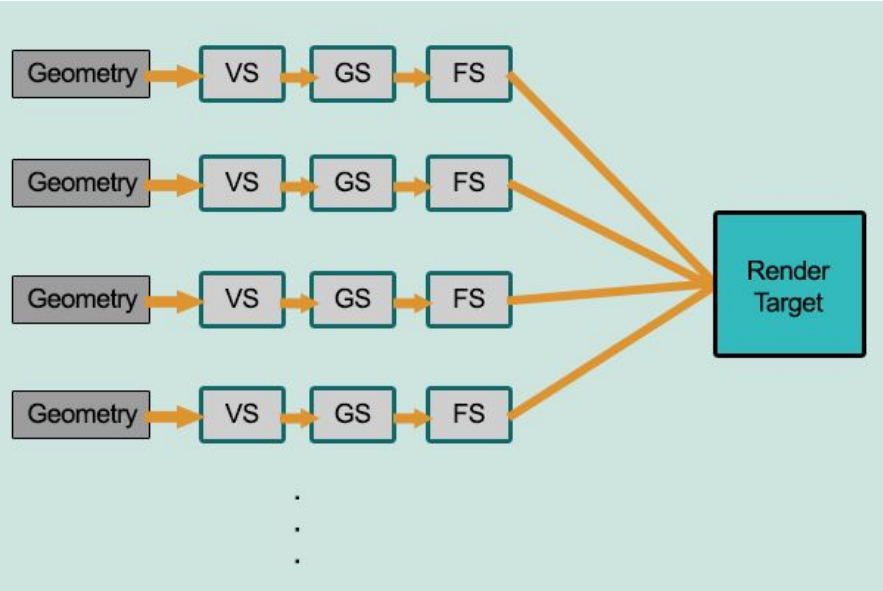
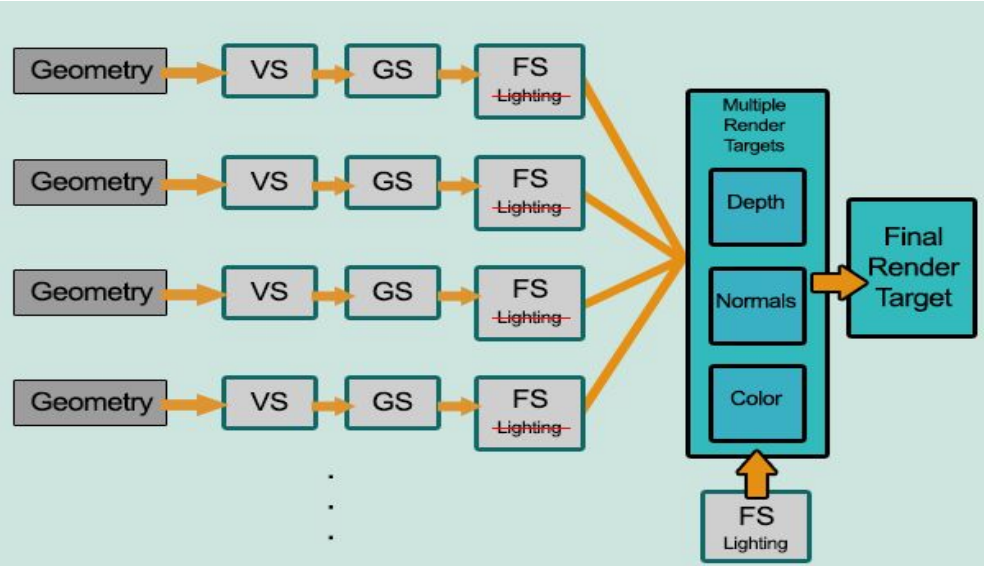
Deferred Shading

Alternativa al forward shading

Nel rendering classico i modelli vengono inviati alla GPU e attraverso la pipeline grafica questi vengono completamente renderizzati sullo schermo.



Nel deferred shading i modelli non vengono renderizzati immediatamente, ma in due fasi distinte, il **geometry-pass** e lo **shading-pass**.



Questa tecnica viene usata per diminuire la quantità di lavoro necessario a gestire grandi quantità di luci nella scena, permettendone anche a migliaia senza cali eccessivi di framerate.

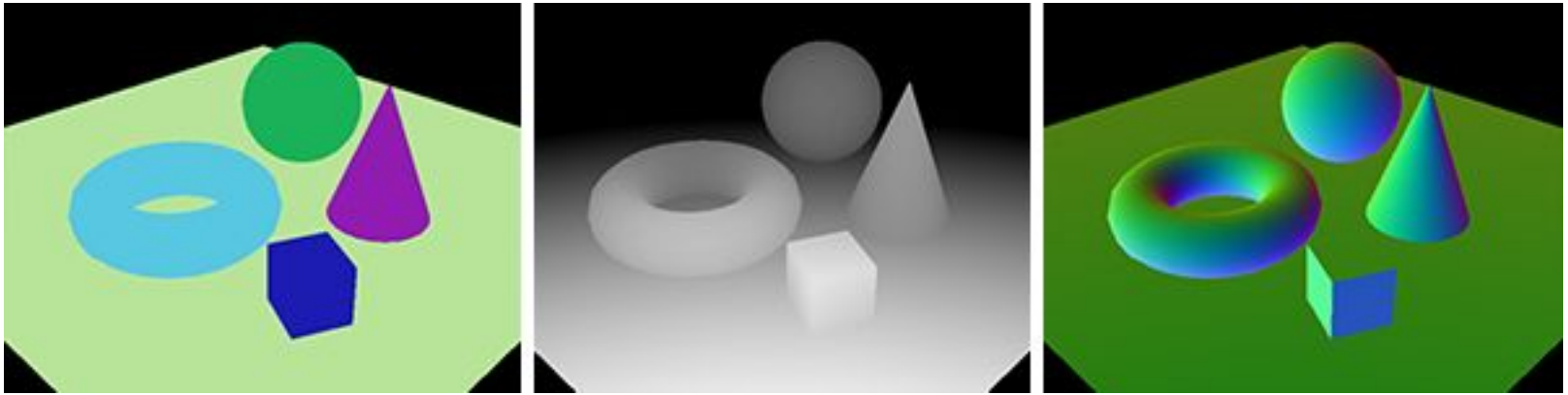
- Con il forward rendering per ogni oggetto della scena viene calcolato il contributo luminoso di tutte le luci per ogni fragment che questo va a coprire, incurante del fatto che un oggetto potrebbe essere nascosto da un'altro non ancora renderizzato.
  Una complessità di `O(num_geometry_fragments * num_lights)`.

- Il deferred shading posticipa lo shading ad una fase successiva, calcolando il contributo delle luci ad ogni pixel della scena una singola volta.
  Una complessità di `O(screen_resolution * num_lights)`.

- Utilizzando light-volumes per le luci durante la fase di shading è possibile ridurre ancora di più la complessità a `O(lights_volume_fragmens * num_lights)`.

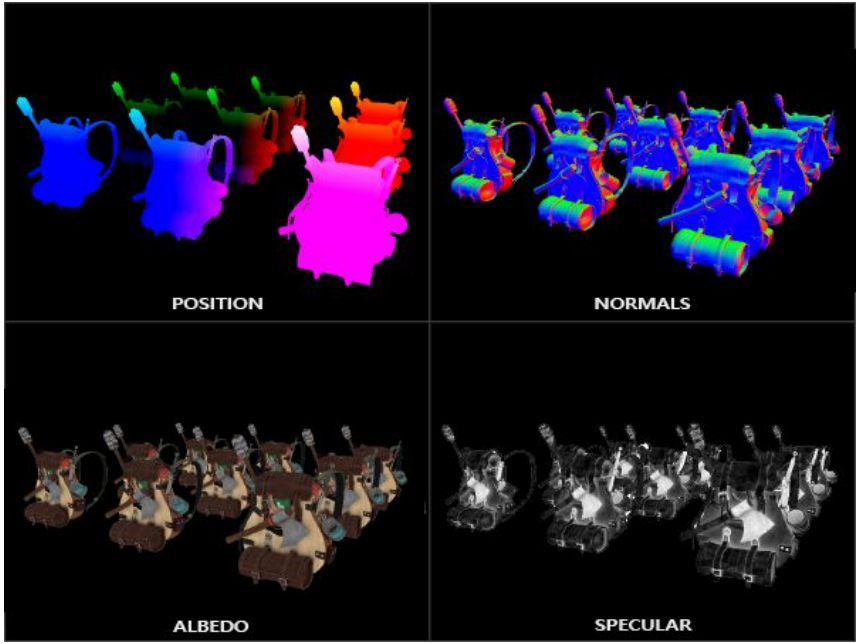
Vantaggi

Si occupa di catturare le informazioni intrinseche del modello e della sua posizione spaziale (normali, colore, profondità rispetto alla camera, etc) in un buffer temporaneo chiamato **gbuffer**.



Geometry Pass

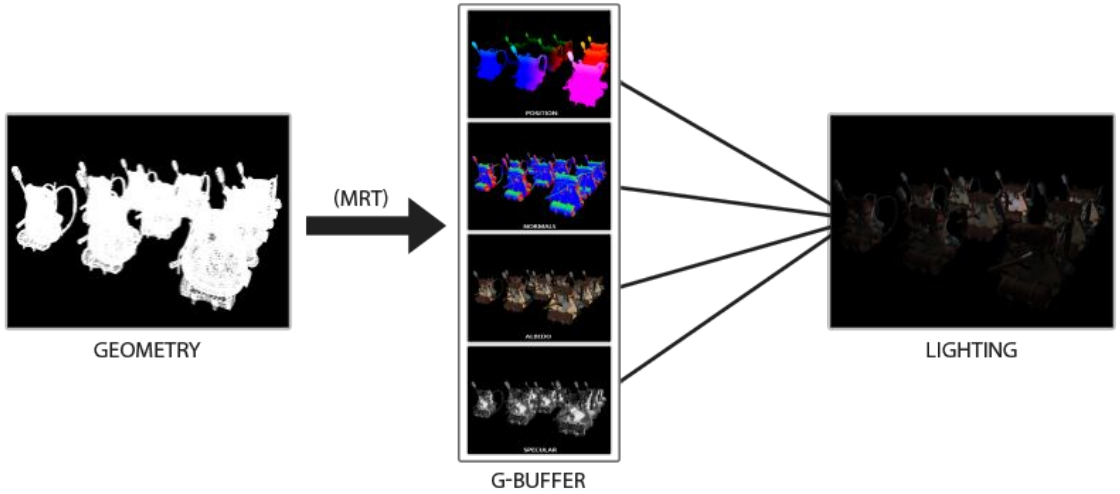
```
#version 330 core
layout (location = 0) out vec3 gPosition;
layout (location = 1) out vec3 gNormal;
layout (location = 2) out vec4 gAlbedoSpec;

in vec2 TexCoords;
in vec3 FragPos;
in vec3 Normal;

uniform sampler2D texture_diffuse;
uniform sampler2D texture_specular;

// Fragment shader
void main()
{
    gPosition = FragPos;
    gNormal = normalize(Normal);
    gAlbedoSpec.rgb = texture(texture_diffuse, TexCoords).rgb;
    gAlbedoSpec.a = texture(texture_specular, TexCoords).r;
}
```



Geometry Pass

Lo shading pass unisce il gbuffer e le informazioni sulle luci presenti nella scena per generare l'immagine finale.

```
#version 330 core
out vec4 FragColor;

in vec2 TexCoords;

uniform sampler2D gPosition;
uniform sampler2D gNormal;
uniform sampler2D gAlbedoSpec;

const int NR_LIGHTS = 32;
uniform Light lights[NR_LIGHTS];
uniform vec3 viewPos;

// …

// fragment shader
void main()
{
    vec3 FragPos = texture(gPosition, TexCoords).rgb;
    vec3 Normal = texture(gNormal, TexCoords).rgb;
    vec3 Albedo = texture(gAlbedoSpec, TexCoords).rgb;
    float Specular = texture(gAlbedoSpec, TexCoords).a;

    for(int i = 0; i < NR_LIGHTS; ++i) {
        FragColor += calculate_light(...)
    };
}
```



Shading pass

- Questa tecnica però impedisce l’uso del blending fra frammenti (ad esempio per effetti di trasparenza), in quanto al momento dello shading è presente il contributo del solo fragment vincente nella fase geometrica.

- Da sola forza tutte le geometrie ad essere renderizzate con lo stesso shader.

- Utilizza molta memoria per mantenere il gbuffer in VRAM.

- Per scene piccole il costo dell’overhead aggiuntivo supera i benefici.

Inoltre esistono tecniche di rendering ancora più performanti come quelle tile-based.

Svantaggi

Per permettere il rendering di oggetti con particolari shader e che richiedono blending è sufficiente copiare il depth-buffer dal gbuffer al framebuffer di base ed eseguire il rendering normalmente.