Fondamenti di Computer Graphics M (8 cfu)

Lab 03 - Navigazione interattiva in scena con modelli mesh 3D

Lorenzo Righi

8 marzo 2024

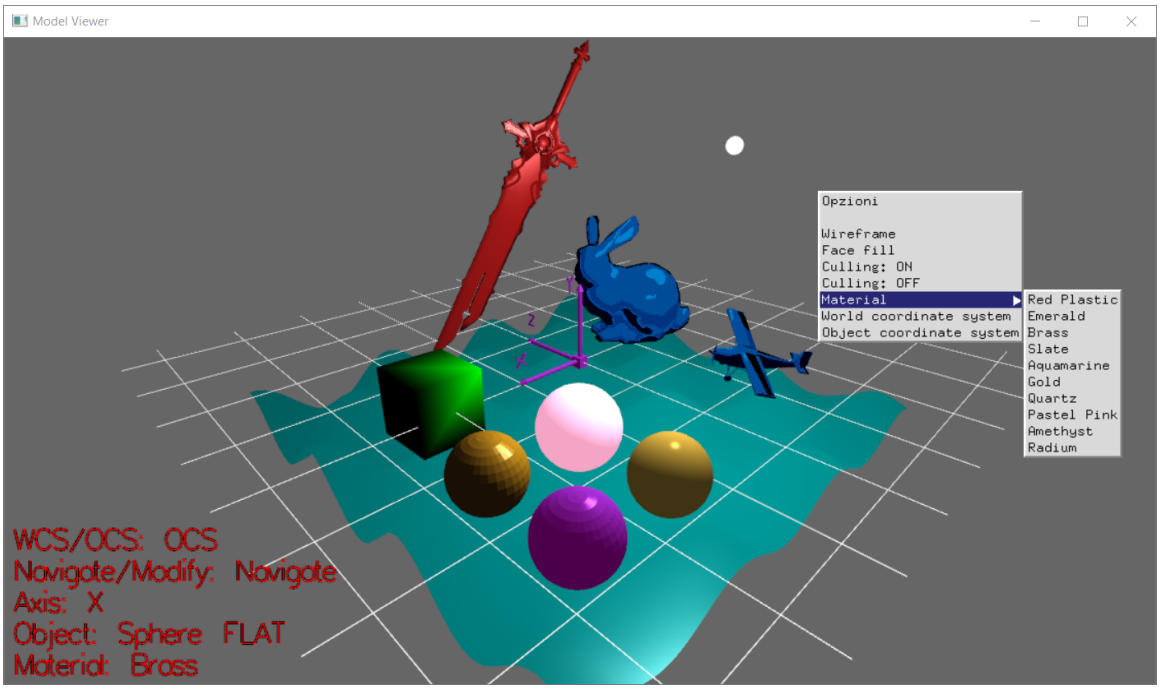


Figura 1: Demo Applicazione

Traccia

Estendere il programma secondo le seguenti specifiche:

1. **Caricamento e visualizzazione oggetti**

a) Calcolo e memorizzazione delle normali ai vertici per i modelli mesh poligonali. Visualizzazione in modalit'a normali alle facce (flat shading) e normali ai vertici (sia GOURAUD shading sia PHONG shading).

b) Provare a creare un materiale diverso da quelli forniti.

c) Prendete visione di come sono gestiti gli shader GOURAUD PHONG BLINN per l'illuminazione e shading.

d) **Wave motion:** Si crei l'animazione di un height field mesh (oggetto mesh GridPlane.obj contenuto nella directory Mesh) modificando la posizione dei vertici in un vertex shader v wave.glsl (effetto da ottenere mostrato in Fig. 2a). Utilizzare la variabile elapsed time t, passata da applicazione al vertex shader, per riprodurre il moto ondoso ottenuto con la sola modifica della coordinata y mediante la formula: vy = asin(ωt + 10vx)sin(ωt + 10vz), dove l'ampiezza dell'oscillazione a e la frequenza ω siano scelte a piacere, es. a = 0.1 e ω = 0.001.

e) **Toon shading:** Realizzare gli shaders v toon.glsl e f toon.glsl per la resa non-fotorealistica nota comunemente come "Toon shading". Un esempio 'e illustrato in Fig. 2b.

2. **Navigazione interattiva in scena**
   Implementare le seguenti funzionalità per il sistema interattivo:

3. **Trasformazione degli oggetti in scena**
   Si richiede di gestire opportunamente nella funzione modifyModelMatrix() le trasformazioni di traslazione, rotazione e scalatura dei singoli oggetti in scena rispetto al sistema di riferimento WCS o OCS (World e Object Coordinate Systems)

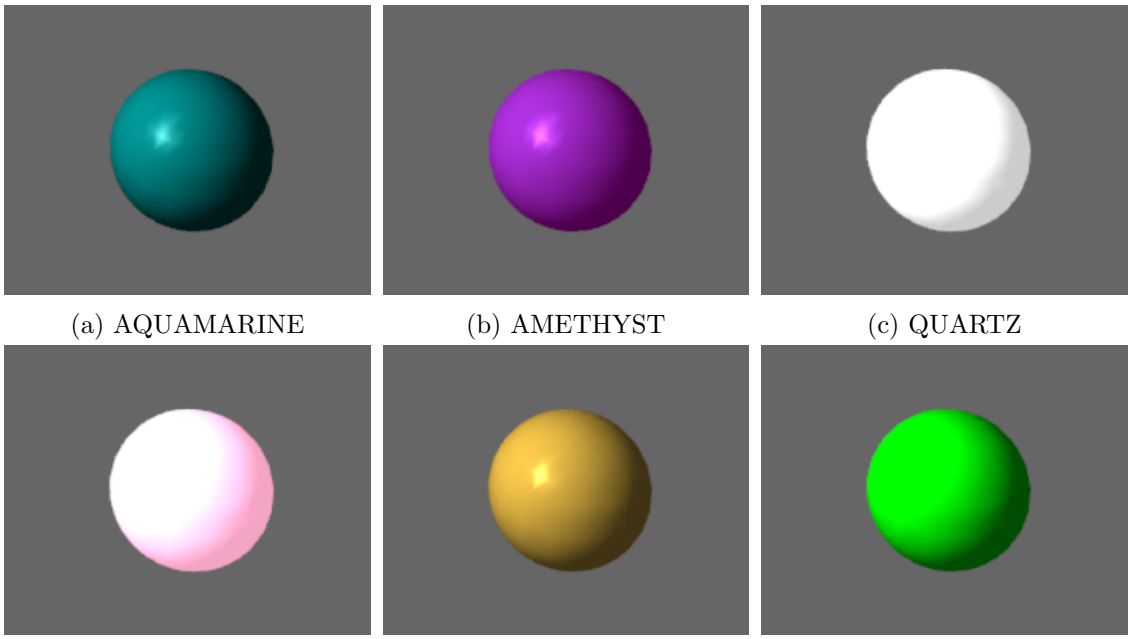
1 Introduzione

La soluzione è stata realizzata senza apportare modifiche alla struttura del codice già fornito, bensì aggiungendo le sole feature richieste e qualche extra.

2 Soluzione

2.1 Materiali

Oltre a quelli forniti dal pacchetto iniziale ne sono stati aggiunti cinque, nel dettaglio è stato realizzato il materiale Aquamarine per fornire alla mesh delle onde un colore più realistico, e un materiale generico per un semplice testing sui vari oggetti disponibili.



(a) AQUAMARINE (b) AMETHYST (c) QUARTZ

(d) PASTEL_PINK (e) GOLD (f) RADIUM

Figura 2: Materiali aggiunti con Goraud shading

2.2 Pan Orizzontale

La prima feature tra quelle richieste dalla traccia ad essere stata realizzata è lo spostamento della camera orizzontale. Essendo già implementata la traslazione verticale l'implementazione di quella sull'asse orizzontale della camera risulta molto immediata: è sufficiente sfruttare i vettor VUP (verticale) e CA (di profondità) della camera, facendone il cross product, per ottenere il vettore laterale. A seguire vi è il codice per lo spostamento a sinistra:

```
void moveCameraLeft() {
        glm::vec3 direction = ViewSetup.target - ViewSetup.position;
        glm::vec3 slide = glm::cross(direction, glm::vec3(ViewSetup.upVector)) *
            CAMERA_TRASLATION_SPEED;
        ViewSetup.position -= glm::vec4(slide, 0.0);
        ViewSetup.target -= glm::vec4(slide, 0.0);
}
```

"direction" rappresenta il vettore di profondità, che va dalla posizione della camera al centro della scena.

2.3 Shader

La traccia prevede la realizzazione di due shader, l'implementazione è presentata di seguito

2.3.1 Wave

Il vertex shader per la realizzazione dell'effetto a onda come da traccia sfrutta la funzione seno per alterare la componente y dei vertici a runtime, dipendentemente dalla loro posizione sul piano xz, il tempo trascorso, la frequenza e una height scale. Il codice del main è il seguente:

```
void main() {
    vec4 v = vec4(aPos, 1.0);
    v.y = a * sin(freq * time + 10 * v.x) * sin(freq * time + 10 * v.z);
    gl_Position = P * V * M * v;
}
```

L'applicazione dell'effetto onda alla mesh di un piano, usando v_waves.glsl, f_gouraud.glsl e il materiale AQUAMARINE è mostrata in Fig. 3.

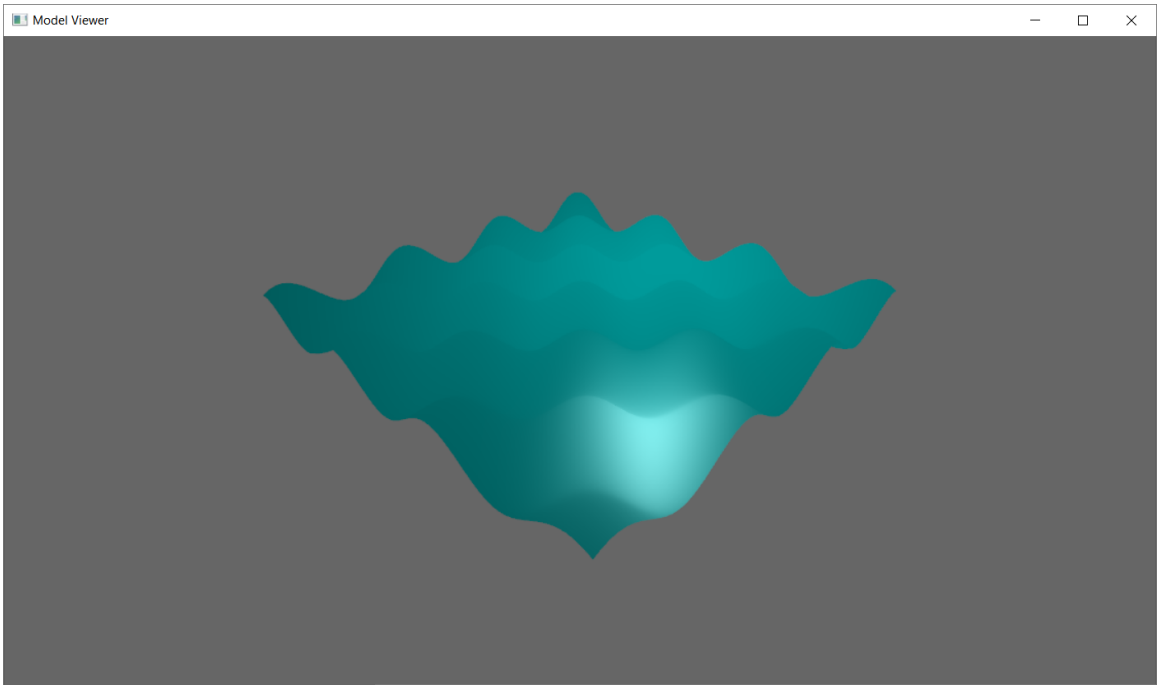


Figura 3: GridPlane con shader Wave

2.3.2 Toon

La toon shader è stata realizzata come riportato in seguito. Per i vertici:

```
void main() {
    gl_Position = P * V * M * vec4(aPos, 1.0);
    vec4 eyePosition = V * M * vec4(aPos, 1.0);
    vec4 eyeLightPos = V * vec4(lightPosition, 1.0);
    N = normalize(transpose(inverse(mat3(V * M))) * aNormal);
    L = normalize((eyeLightPos - eyePosition).xyz);
    E = normalize(-eyePosition.xyz);
}
```

Mentre per i frammenti:

```
void main() {
    vec4 a = vec4(0.0,0.5,0.8,1.0);
    vec4 b = vec4(0.0,0.3,0.6,1.0);
    vec4 c = vec4(0.0,0.2,0.5,1.0);
    vec4 d = vec4(0.0,0.3,0.6,1.0);
    vec4 e = vec4(0.0,0.0,0.1,1.0);
    vec4 color;
    float intensity = dot(normalize(L),normalize(N));
    if (intensity > 0.95)
        color = a;
    else if (intensity > 0.5)
        color = b;
    else if (intensity > 0.25)
        color = c;
    else if (intensity > 0.15)
        color = d;
    else
        color = e;
    float aa = dot(normalize(E), normalize(N));
    if(aa >= 0.0 && aa < 0.30)
        color = vec4(0.0, 0.0, 0.0, 1.0);
    FragColor = color;
}
```

È interessante sottolineare che, a differenza della wave shader, nella toon vengano passati anche i vettori con le informazioni sulla luce (rispettivamente in uscita nella vertex e in ingresso nella fragment). Questi servono infatti per discriminare la fascia d'intensità a cui

apparterrà una determinata zona della superficie della mesh, ovvero la tonalità di blu come si può notare in Fig. 4.

Figura 4: Bunny con shader TOON

2.4 Trasformazione degli Oggetti in Scena

La traccia prevedeva anche l'implementazione della possibilità di applicare le tre trasformazioni principali ai singoli oggetti presenti in scena, in riferimento alle coordinate WCS o OCS in base alla selezione corrente. L'implementazione è stata realizzata come segue:

```
void modifyModelMatrix(glm::vec3 translation_vector, glm::vec3 rotation_vector,
            GLfloat angle, GLfloat scale_factor) {
    glm::mat4 M = objects[selected_obj].M;
    //TRANSLATION
    M = glm::translate(M, translation_vector);
    //ROTATION
    if (angle != 0) {
        if (TransformMode == OCS) {
                M = glm::rotate(M, glm::radians(angle), rotation_vector);
        }
        else if (TransformMode == WCS) {
                M = glm::translate(M, -distances.at(selected_obj));
```

```
                M = glm::rotate(M, glm::radians(angle), rotation_vector);
                M = glm::translate(M, distances.at(selected_obj));
        }
    }
    //SCALE
    if (scale_factor != 1.0) {
        if (TransformMode == OCS) {
                M = glm::scale(M, glm::vec3(scale_factor));
        }
        else if (TransformMode == WCS) {
                M = glm::translate(M, -distances.at(selected_obj));
                M = glm::scale(M, glm::vec3(scale_factor));
                M = glm::translate(M, distances.at(selected_obj));
        }
    }
    objects[selected_obj].M = M;
}
```

3 Extra

In aggiunta alle richieste della traccia è stata implementata la possibilità di aggiungere un moltiplicatore alle trasformazioni degli oggetti in scena, che è possibile attivare tenendo premuta la barra spaziatrice, e due shader: GLITCH e RADIATION.

3.1 GLITCH shader

Per i vertici di questo shader non sono previste modifiche, quindi il vertex si comporta come un passthrough, con l'aggiunta di un vettore bidimensionale vUv che viene passato in uscita. Il fragment shader invece si occupa di realizzare l'effetto glitch: viene utilizzata una funzione per simulare un segnale di noise, il tempo e il vettore vUv passato in ingresso.

```
out vec4 fragColor;
in vec2 vUv;
uniform float time;
uniform vec2 resolution;
float noise(vec2 p) {
    return fract(sin(dot(p.xy, vec2(12.9898, 78.233))) * 43758.5453);
}
void main() {
    vec2 pos = vUv + vec2(noise(vUv + time * 0.1) * 0.005 - 0.0025, 0.0);
    float n = noise(pos * time);
```

```
    vec3 color = mix(vec3(0.9, 0.9, 0.9), vec3(0.1, 0.1, 0.1), n);
    fragColor = vec4(color, 1.0);
}
```

In Fig. 5 l'effetto del GLITCH shader è applicato a un modello di "Slenderman", per il quale questo effetto simile a un disturbo di segnale televisivo è sempre stato iconico.

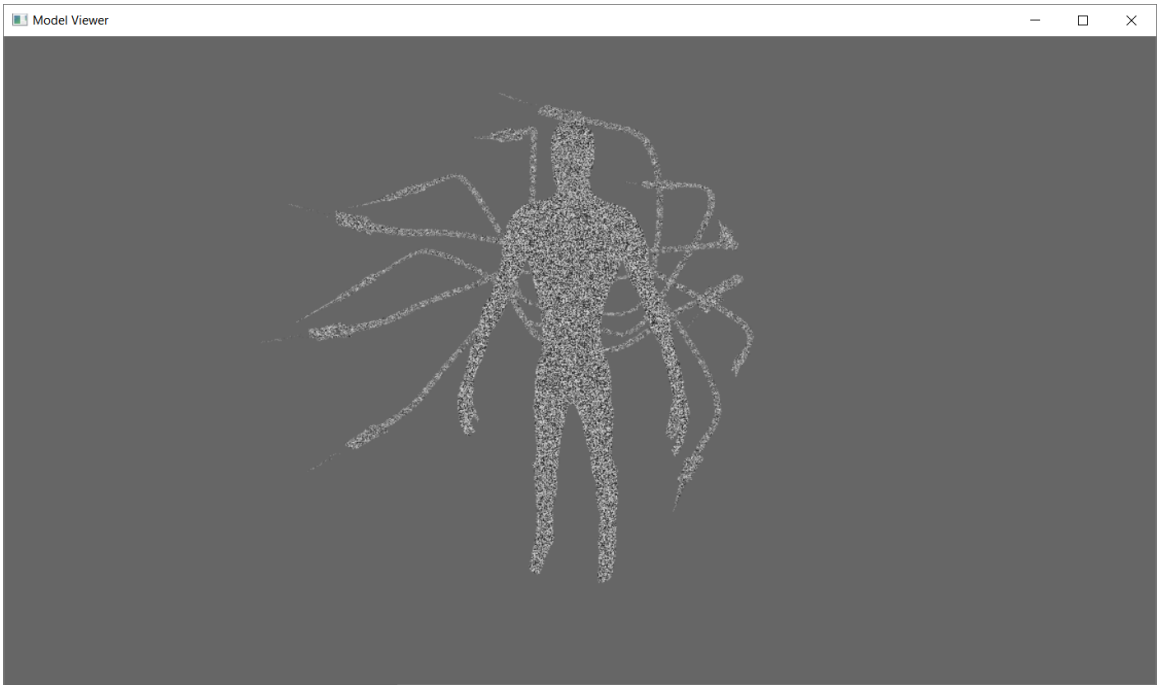


Figura 5: Slenderman con GLITCH shader

3.2 RADIATION shader

Per simulare delle radiazioni è stata realizzata una vertex shader al fine di creare un effetto pulsato sulla superficie degli oggetti a cui è applicata, il codice è il seguente:

```
void main() {
    vec4 worldPosition = M * vec4(aPos, 1.0);
        vec4 eyePosition = V * M * vec4(aPos, 1.0);
        vec4 eyeLightPos = V * vec4(light.position, 1.0);
        E = normalize(-eyePosition.xyz);
        L = normalize((eyeLightPos - eyePosition).xyz);
        N = normalize(transpose(inverse(mat3(V * M))) * aNormal);
    vec3 R = reflect(-L, N);
    vec3 ambient = light.power * material.ambient;
```

```
    float diff = max(dot(L,N), 0.0);
    vec3 diffuse = light.power * light.color * diff * material.diffuse;
    float spec = pow(max(dot(E, R), 0.0), material.shininess);
    vec3 specular =  light.power * light.color * spec * material.specular;
    Color = ambient + diffuse + specular;
    float distanceToCenter = length(worldPosition.xyz);based on the distance to the center
    float pulseIntensity = abs(sin(freq * time + distanceToCenter));
    vec3 pulsatingPosition = aPos + aNormal * pulseIntensity * 0.1;position to clip space
    gl_Position = P * V * M * vec4(pulsatingPosition, 1.0);
}
```

Questa viene combinata con la GOURAUD fragment shader e il risultato è mostrato in Fig. 6

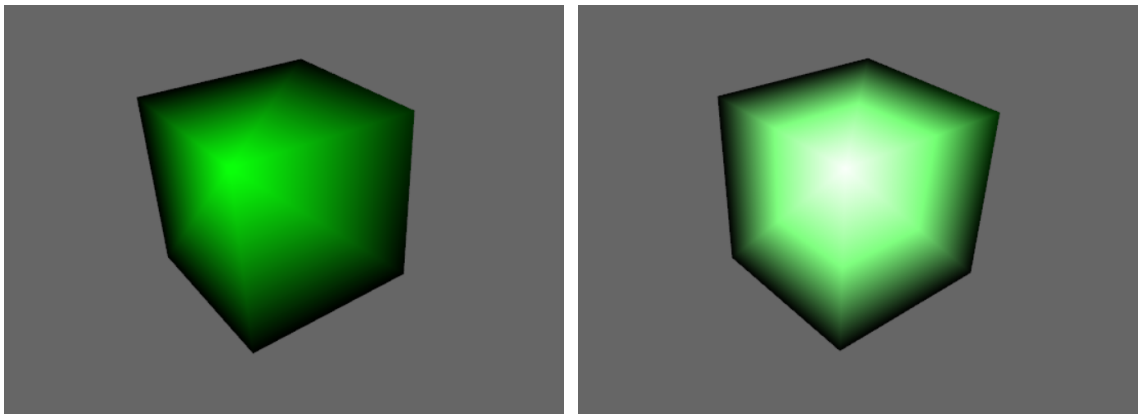


Figura 6: Cube con RADIATION shader

A seguire un'immagine della scena completa:

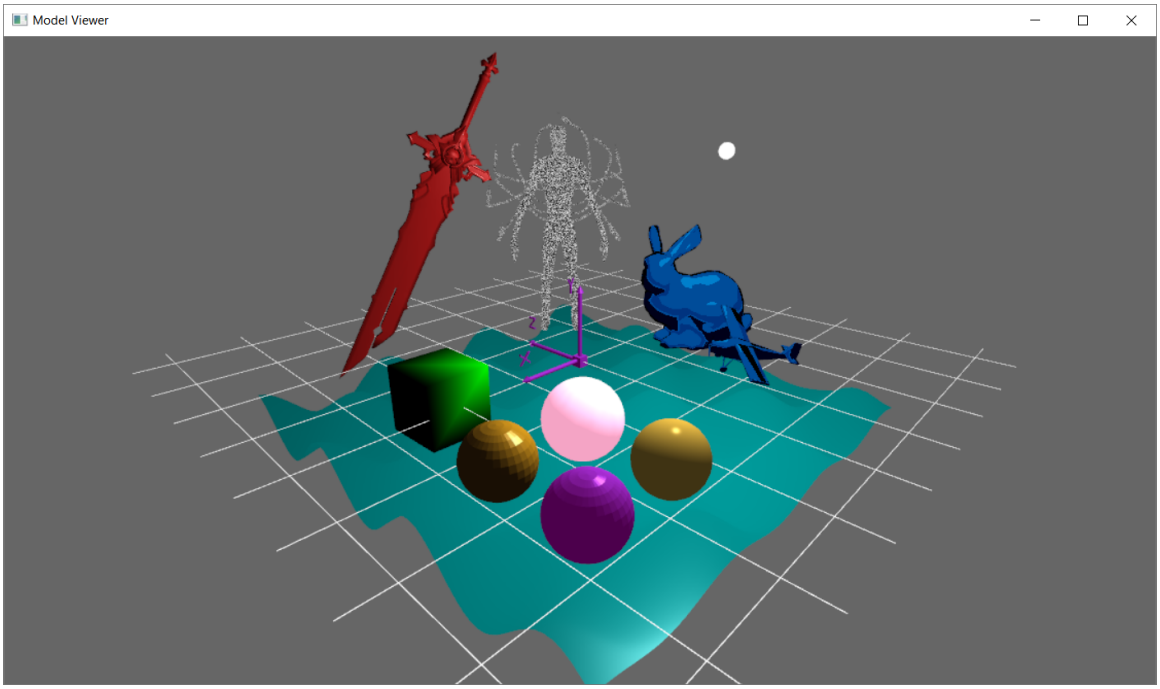


Figura 7: Scena completa