Videogame e rendering 3D

Enrico Colombini (Erix)

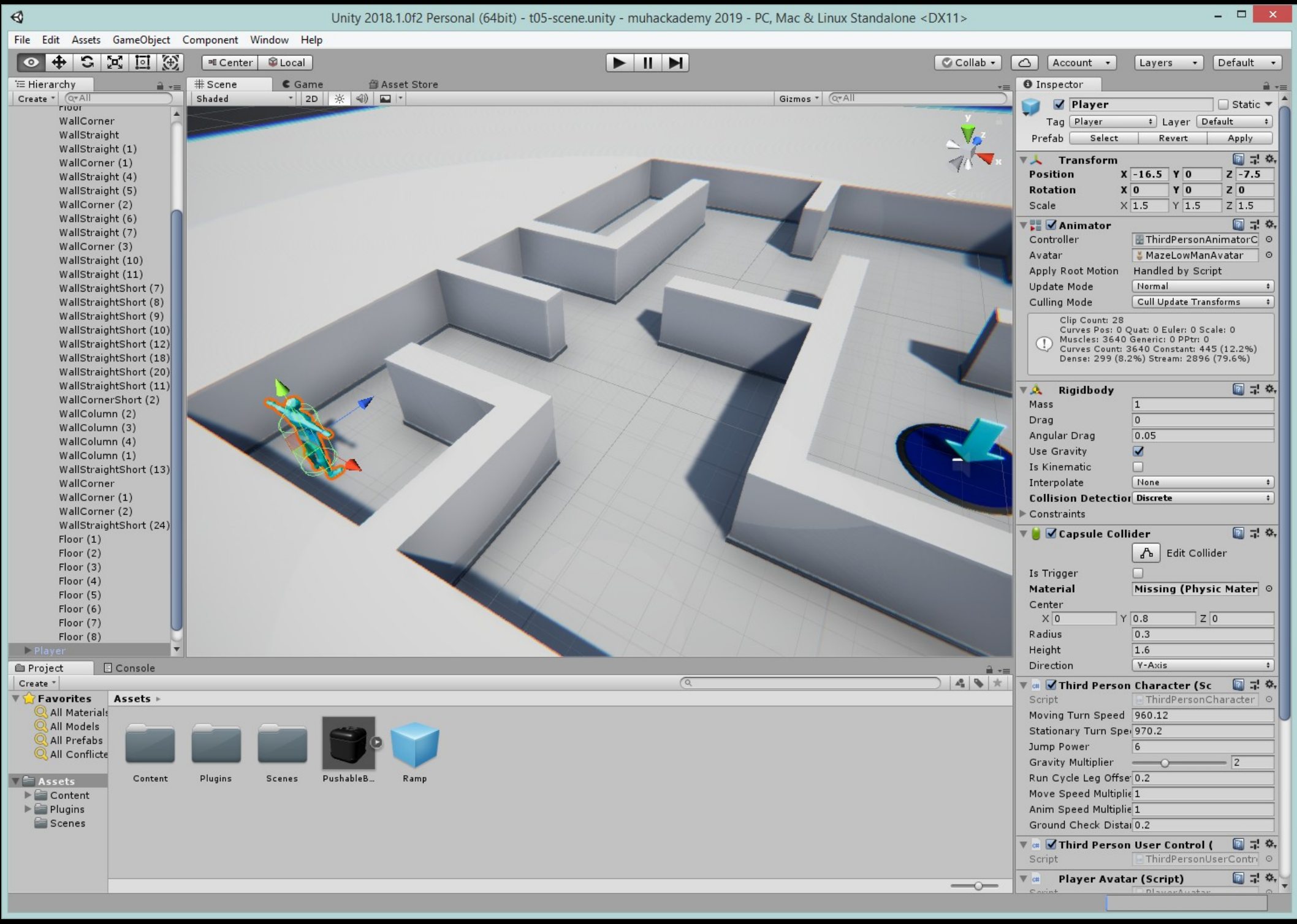
µhackademy 1 Marzo 2019

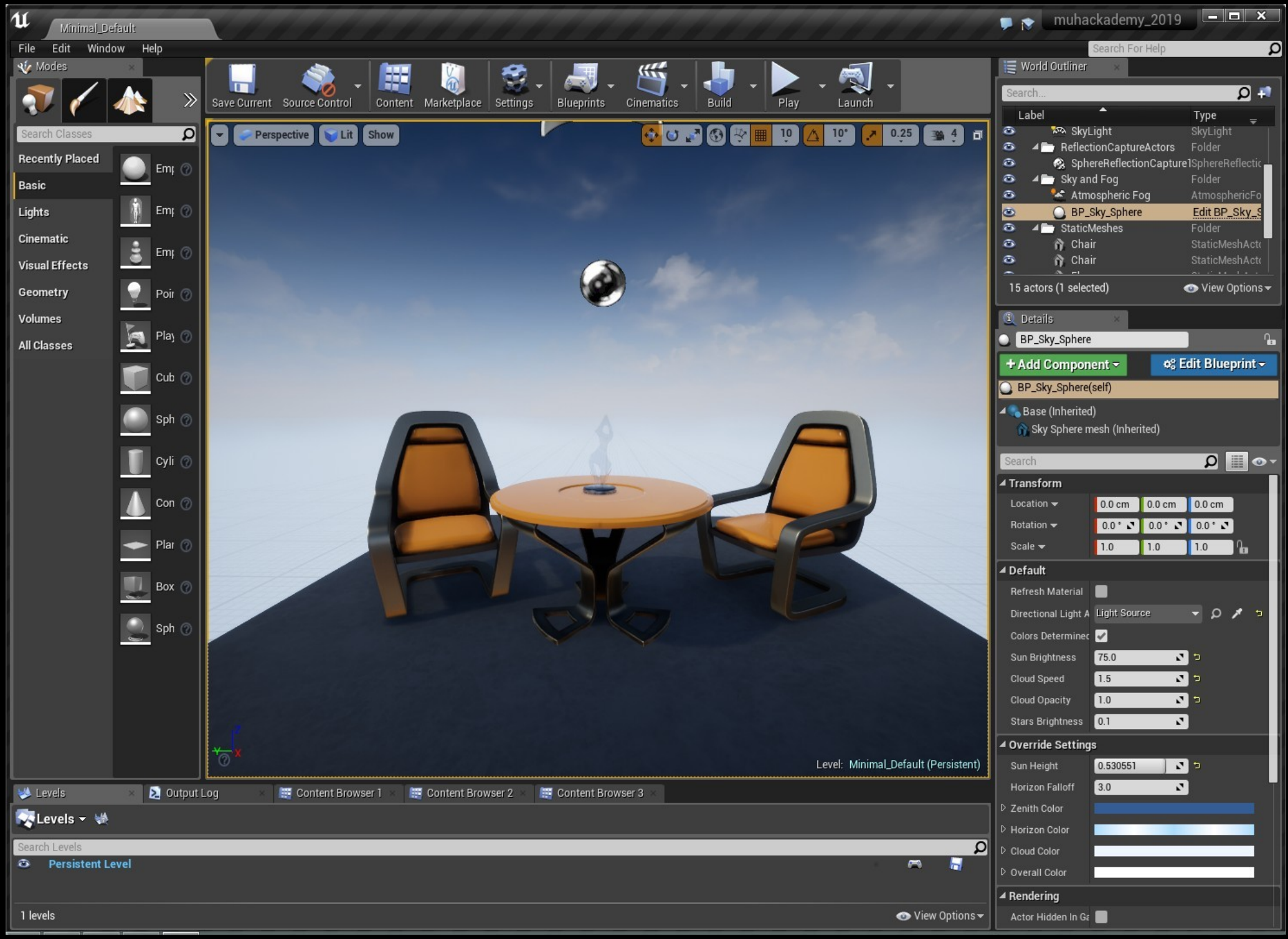
Volete creare un videogame?

Strumenti comodi: game engine

- Editor 3D, componenti
- Runtime engine (PC, console, mobile)
- Scripting, animazioni, effetti, ecc.

Tanta roba ma abbondanti tutorial


Unity 2018.1.0f2 Personal (64bit) - t05-scene.unity - muhackademy 2019 - PC, Mac & Linux Standalone <DX11>
File Edit Assets GameObject Component Window Help
Center Local
Collab Account Layers Default
Hierarchy
Create
WallCorner
WallStraight
WallStraight (1)
WallCorner (1)
WallStraight (4)
WallStraight (5)
WallCorner (2)
WallStraight (6)
WallStraight (7)
WallCorner (3)
WallStraight (10)
WallStraight (11)
WallStraightShort (7)
WallStraightShort (8)
WallStraightShort (9)
WallStraightShort (10)
WallStraightShort (12)
WallStraightShort (18)
WallStraightShort (20)
WallStraightShort (11)
WallCornerShort (2)
WallColumn (2)
WallColumn (3)
WallColumn (4)
WallColumn (1)
WallStraightShort (13)
WallCorner
WallCorner (1)
WallCorner (2)
WallStraightShort (24)
Floor (1)
Floor (2)
Floor (3)
Floor (4)
Floor (5)
Floor (6)
Floor (7)
Floor (8)
Player
Scene
Game
Asset Store
Shaded
2D
Gizmos
Inspector
Player
Static
Tag Player
Layer Default
Prefab Select Revert Apply
Transform
Position X -16.5 Y 0 Z -7.5
Rotation X 0 Y 0 Z 0
Scale X 1.5 Y 1.5 Z 1.5
Animator
Controller ThirdPersonAnimatorC
Avatar MazeLowManAvatar
Apply Root Motion Handled by Script
Update Mode Normal
Culling Mode Cull Update Transforms
Clip Count: 28
Curves Pos: 0 Quat: 0 Euler: 0 Scale: 0
Muscles: 3640 Generic: 0 PPtr: 0
Curves Count: 3640 Constant: 445 (12.2%)
Dense: 299 (8.2%) Stream: 2896 (79.6%)
Rigidbody
Mass 1
Drag 0
Angular Drag 0.05
Use Gravity
Is Kinematic
Interpolate None
Collision Detectior Discrete
Constraints
Capsule Collider
Edit Collider
Is Trigger
Material Missing (Physic Mater
Center
X 0 Y 0.8 Z 0
Radius 0.3
Height 1.6
Direction Y-Axis
Third Person Character (Sc
Script ThirdPersonCharacter
Moving Turn Speed 960.12
Stationary Turn Spe 970.2
Jump Power 6
Gravity Multiplier 2
Run Cycle Leg Offse 0.2
Move Speed Multiplie 1
Anim Speed Multiplie 1
Ground Check Distar 0.2
Third Person User Control (
Script ThirdPersonUserContr
Player Avatar (Script)
Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
All Conflicte
Assets
Content
Plugins
Scenes
Assets
Content
Plugins
Scenes
PushableB...
Ramp


Minimal_Default
muhackademy_2019
File
Edit
Window
Help
Search For Help
Modes
Save Current
Source Control
Content
Marketplace
Settings
Blueprints
Cinematics
Build
Play
Launch
Search Classes
Recently Placed
Basic
Lights
Cinematic
Visual Effects
Geometry
Volumes
All Classes
Perspective
Lit
Show
Level: Minimal_Default (Persistent)
World Outliner
Search...
Label
Type
SkyLight
ReflectionCaptureActors
SphereReflectionCapture1
Sky and Fog
Atmospheric Fog
BP_Sky_Sphere
Edit BP_Sky_S
StaticMeshes
Chair
Chair
15 actors (1 selected)
View Options
Details
BP_Sky_Sphere
+ Add Component
Edit Blueprint
BP_Sky_Sphere(self)
Base (Inherited)
Sky Sphere mesh (Inherited)
Search
Transform
Location
Rotation
Scale
Default
Refresh Material
Directional Light A
Light Source
Colors Determinec
Sun Brightness
75.0
Cloud Speed
1.5
Cloud Opacity
1.0
Stars Brightness
0.1
Override Settings
Sun Height
0.530551
Horizon Falloff
3.0
Zenith Color
Horizon Color
Cloud Color
Overall Color
Rendering
Actor Hidden In Ga
Levels
Output Log
Content Browser 1
Content Browser 2
Content Browser 3
Search Levels
Persistent Level
1 levels
View Options

Scripting: la logica del gioco

Unity:

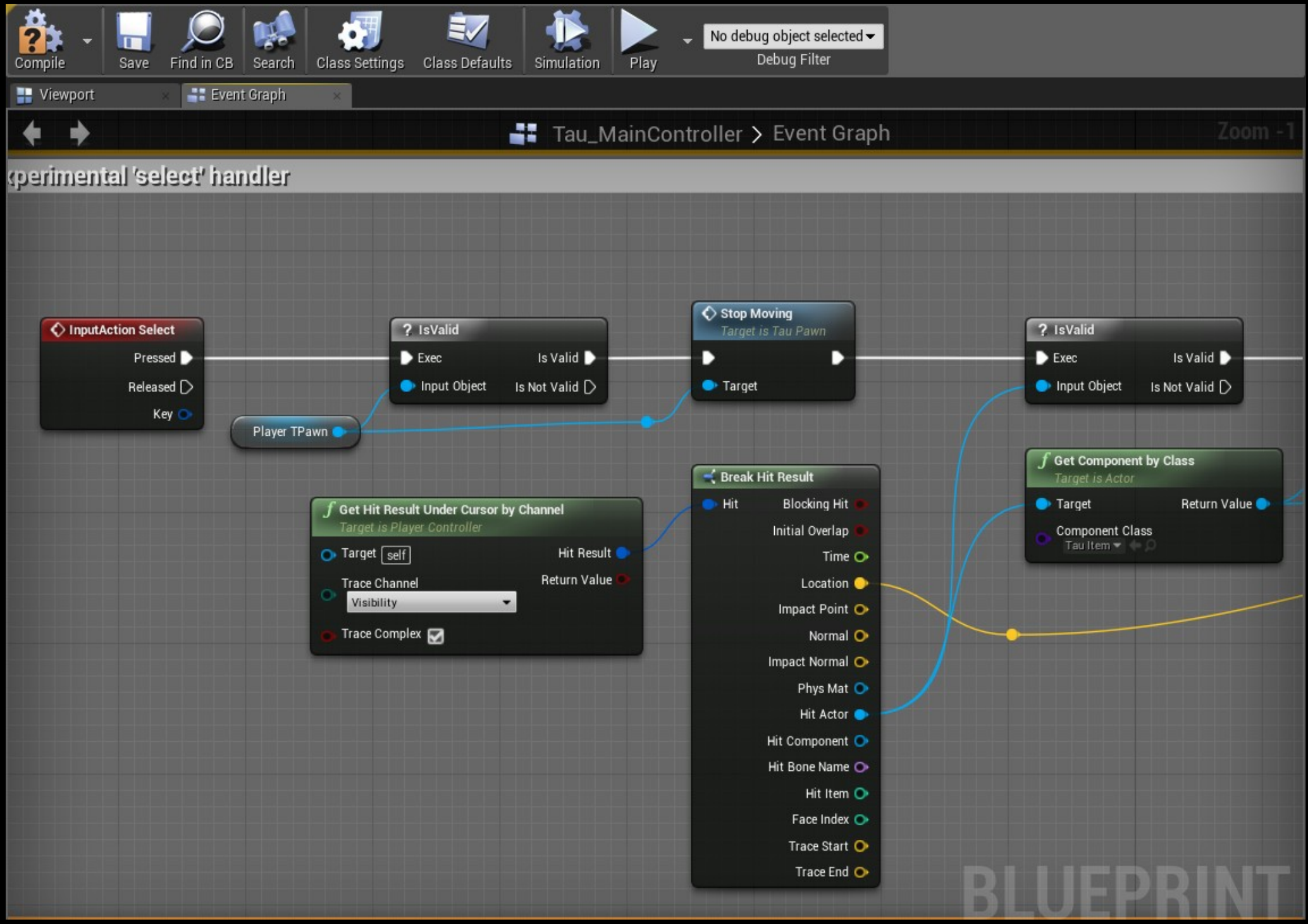
- C#, media difficoltà

Unreal Engine:

- Editor visuale (limitato) o C++ (hard)

UE ha una struttura di classi 'curiosa'

```
public class Door : MonoBehaviour
{
    public float timeUntilClose = 1f;
    private float timer = 0f;
    private bool isOpen = false;
    Animator animator;

    public void Start()
    {
        animator = GetComponent<Animator>();
    }

    public void Open()
    {
        isOpen = true;
        timer = timeUntilClose;

        if (animator)
        {
            animator.SetBool("DoorTriggered", true);
        }
        else
        {
            GetComponent<Collider>().enabled = false;
            GetComponent<Renderer>().enabled = false;
        }
```


Compile
Save
Find in CB
Search
Class Settings
Class Defaults
Simulation
Play
No debug object selected
Debug Filter
Viewport
Event Graph
Tau_MainController > Event Graph
Zoom -1
xperimental 'select' handler
InputAction Select
Pressed
Released
Key
IsValid
Exec
Input Object
Is Valid
Is Not Valid
Player TPawn
Stop Moving
Target is Tau Pawn
Target
IsValid
Exec
Input Object
Is Valid
Is Not Valid
Get Hit Result Under Cursor by Channel
Target is Player Controller
Target
self
Trace Channel
Visibility
Trace Complex
Hit Result
Return Value
Break Hit Result
Hit
Blocking Hit
Initial Overlap
Time
Location
Impact Point
Normal
Impact Normal
Phys Mat
Hit Actor
Hit Component
Hit Bone Name
Hit Item
Face Index
Trace Start
Trace End
Get Component by Class
Target is Actor
Target
Return Value
Component Class
Tau Item
BLUEPRINT

Serve capire come funziona?

- Scriversi un engine non è conveniente
- Capire come funziona è sempre utile, anche per usarlo meglio
- Gli engine fanno molte cose, ma il nucleo è la presentazione 3D

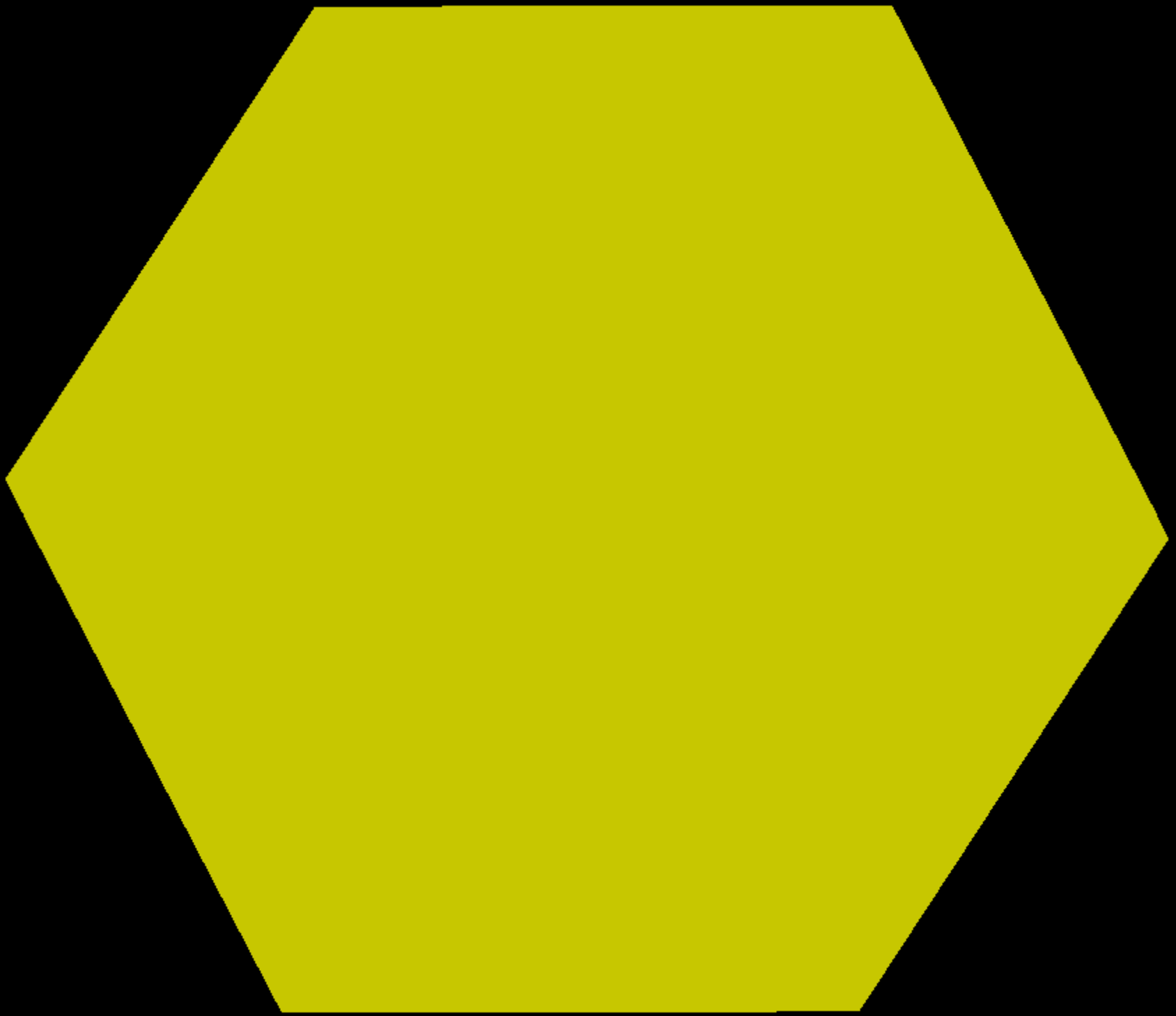
Dentro l'engine: come funziona il 3D?

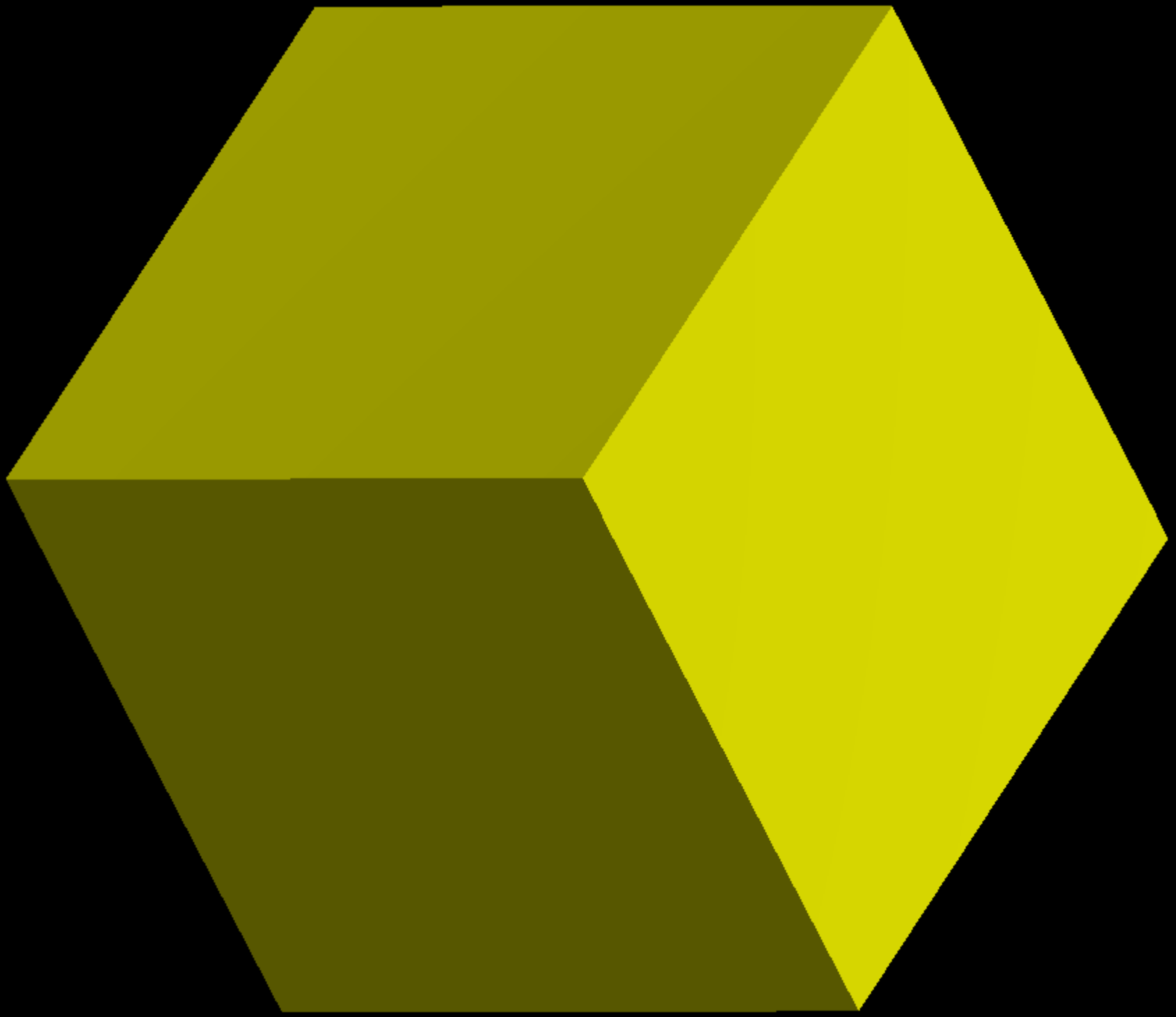
Questa non è una sedia

La grafica 3D...

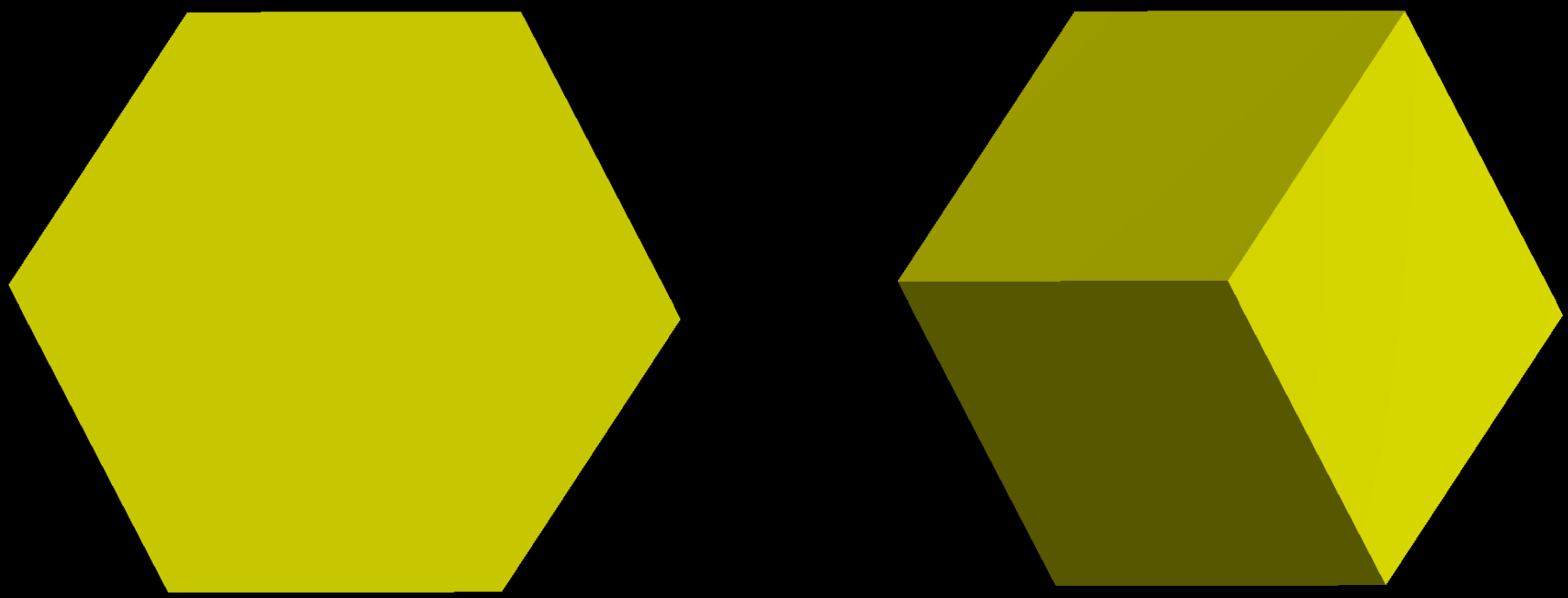
...è una illusione ottica

- Il cervello crede di vedere un oggetto solido su una superficie piana
- Lo stesso per foto, cinema, TV e VR: la retina dell'occhio è 2D!
- Non usiamo solo la visione binoculare





È lo stesso cubo, cambia la luce



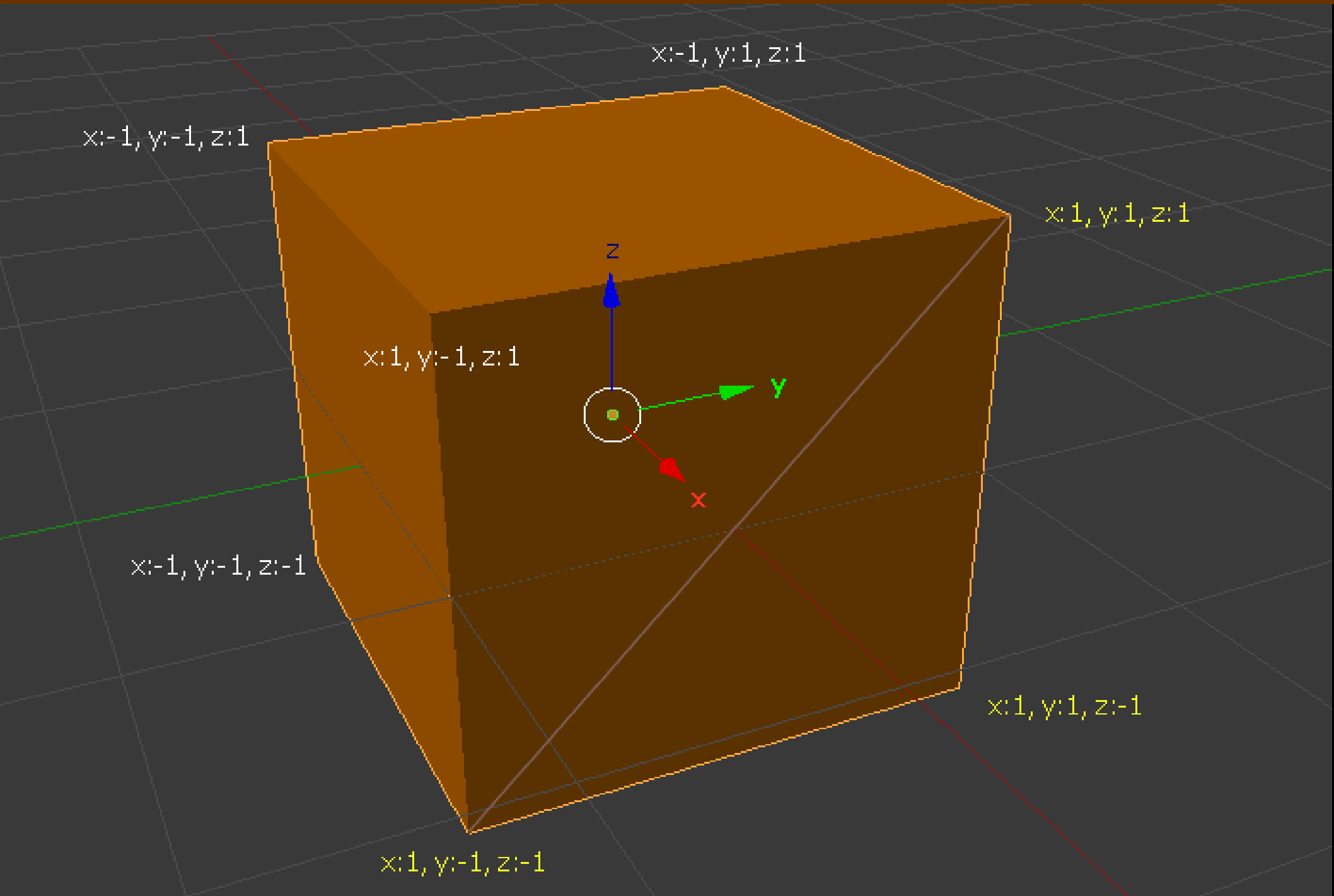
Il 3D calcola l'intensità di ciascun pixel

Per calcolare ogni pixel

Occorre sapere:

- Come sono orientate le superfici
- A quale punto corrisponde ogni pixel
- Quanta luce riceve ciascun pixel

Sistema di coordinate



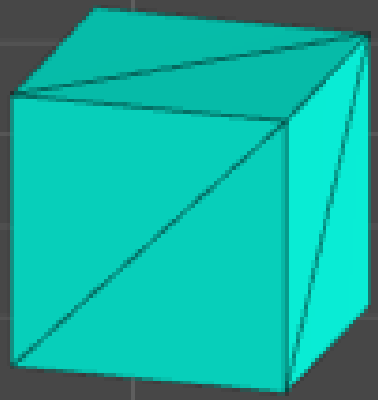
Trasformazioni

Moltiplicando il vettore di ciascun vertice (le sue coordinate locali) per una matrice 4x4 si ottengono:

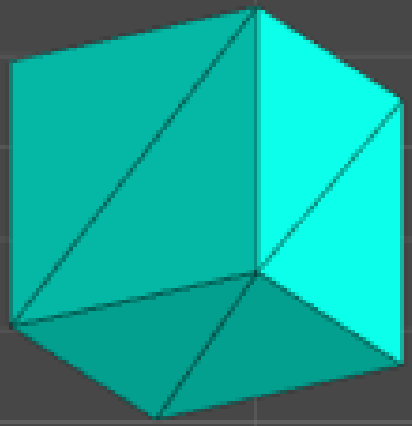
- Rotazioni
- Scaling
- Traslazioni

Se ne occupa la CPU

Originale

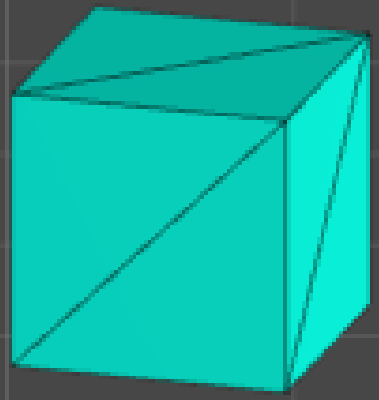


Rotazione

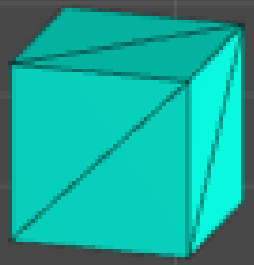


Traslazione

->

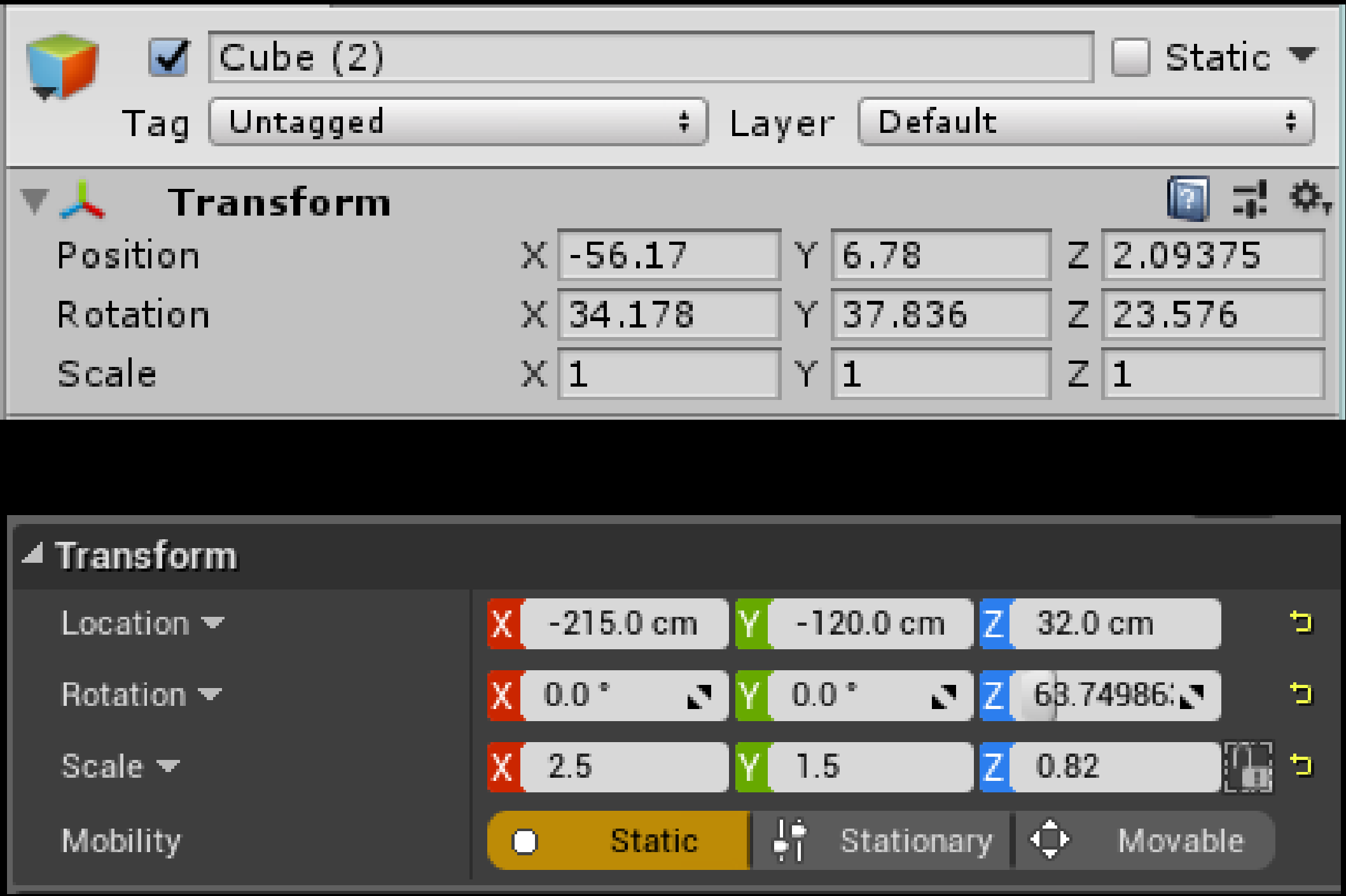


Scaling



Matrici di trasformazione

- View matrix: camera
- Model matrix: soggetto
- Modelview matrix = view * model
- Projection matrix: obiettivo e modalità
- Viewport trasformation: 3D -> 2D


Cube (2)
Static
Tag
Untagged
Layer
Default
Transform
Position
X -56.17
Y 6.78
Z 2.09375
Rotation
X 34.178
Y 37.836
Z 23.576
Scale
X 1
Y 1
Z 1
Transform
Location
X -215.0 cm
Y -120.0 cm
Z 32.0 cm
Rotation
X 0.0 °
Y 0.0 °
Z 63.74986
Scale
X 2.5
Y 1.5
Z 0.82
Mobility
Static
Stationary
Movable

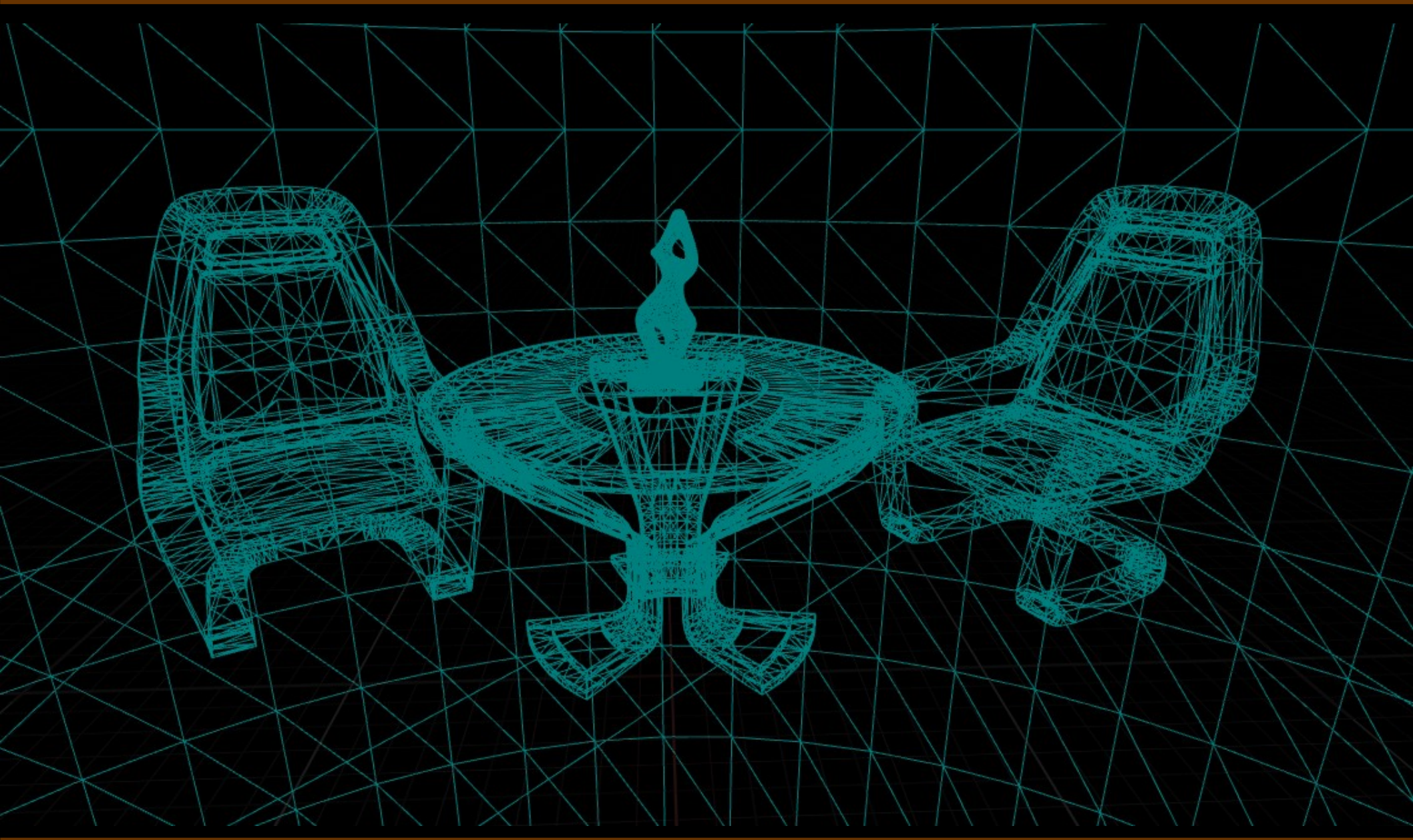
Librerie matematiche

Le trasformazioni vanno applicate dalla CPU a ciascun vertice:

- Gli engine lo fanno in automatico
- OpenGL offre le sue funzioni
- Per OpenGL ES si usano librerie esterne (host system, GLM, ...)

```
leftLight->enabled = true;
rightLight->enabled = true;
frontLight->enabled = true;
dirLights->UpdateAll(viewMatrix3x3);

modelMatrix = baseModelMatrix;
modelMatrix = glm::scale(modelMatrix, glm::vec3(0.0085f));
glm::vec3 rotAxis = glm::normalize(glm::vec3(0.0f, 1.0f, 0.0f));
modelMatrix = glm::translate(
   modelMatrix, glm::vec3(0.2f, -30.0f, 0.0f));
modelMatrix = glm::rotate(modelMatrix, carRotAngle, rotAxis);

//get a good orientation
modelMatrix = glm::rotate(modelMatrix, 70.0f,
   glm::normalize(glm::vec3(1.0f, 0.0f, 0.0f)));
modelMatrix = glm::rotate(modelMatrix, 90.0f,
   glm::normalize(glm::vec3(0.0f, 0.0f, 1.0f)));

modelViewMatrix = viewMatrix * modelMatrix;
mvpMatrix = projMatrix * modelViewMatrix;
normalMatrix = glm::inverseTranspose(glm::mat3(modelViewMatrix));

shaderProgram->SetMvpMatrix(mvpMatrix);
shaderProgram->SetNormalMatrix(normalMatrix);

daredevModel.RenderLightedOpaqueFirst(*shaderProgram);
```

Mesh



Un insieme di triangoli

La CPU comunica alla GPU:

- Coordinate, colore e normali dei vertici
- Coordinate e caratteristiche delle luci
- Texture da applicare alle superfici
- Shader (programmi GPU) e altro

La GPU esegue il rendering

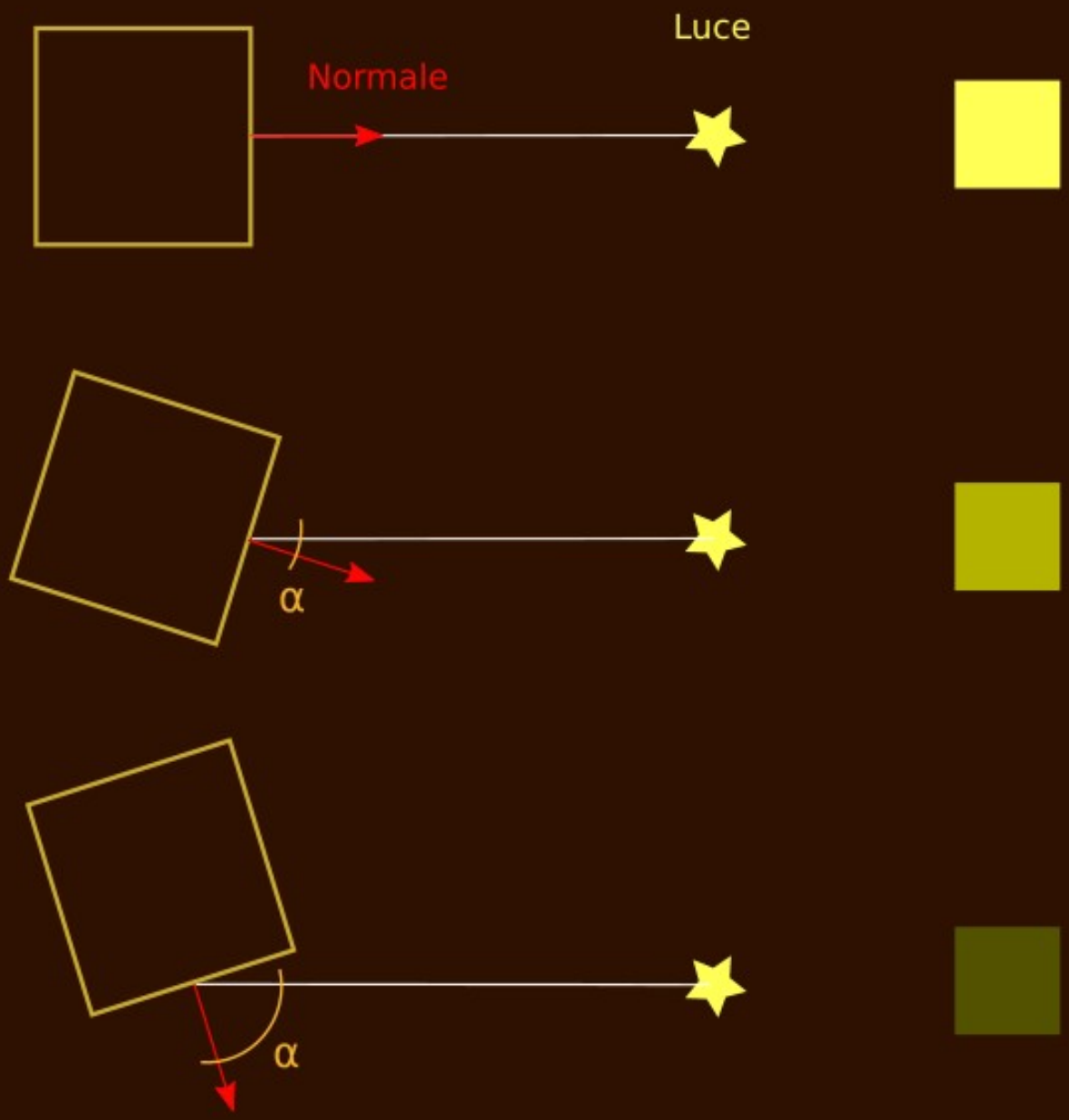
Rendering

Ricalcolato a ciascun frame

Illuminazione

- L'intensità dell'illuminazione dipende dall'angolo tra superficie e luce
- L'angolo si calcola rispetto al vettore normale alla superficie
- Il calcolo viene eseguito dalla GPU per ciascun frammento (pixel)

Per la luce diffusa dalla superficie


Luce
Normale
α
α

Chi fa il calcolo? Lo shader

Gli shader girano nella GPU:

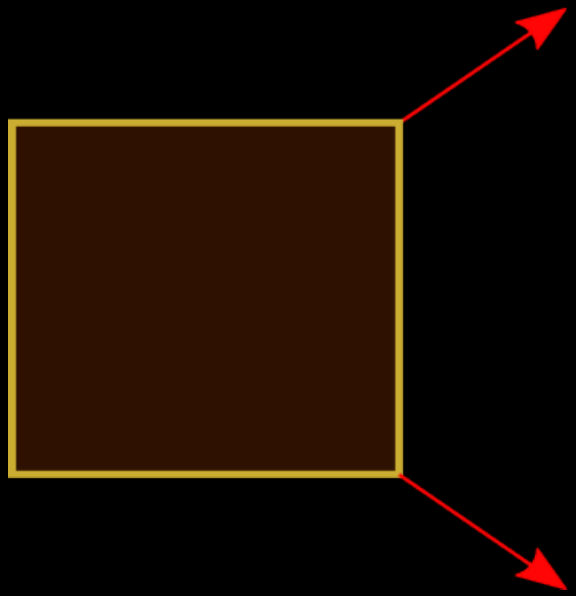
- Il vertex shader viene eseguito per ciascun vertice
- Il fragment shader viene eseguito per ciascun frammento (pixel)
- L'esecuzione è fortemente parallela

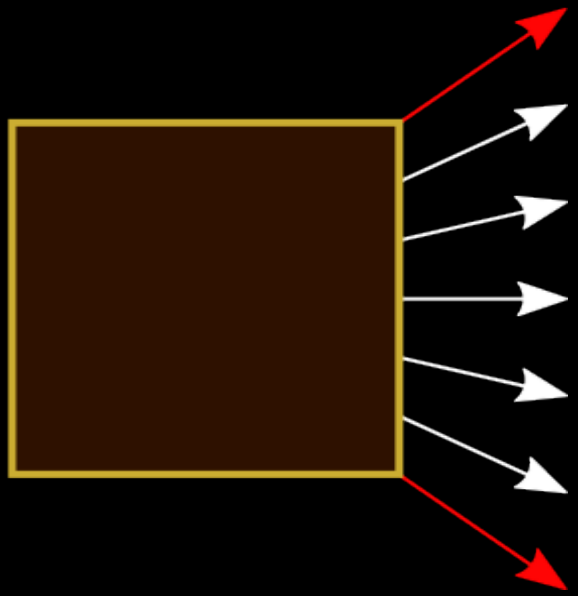
>100 GFLOPS (single precision FP)

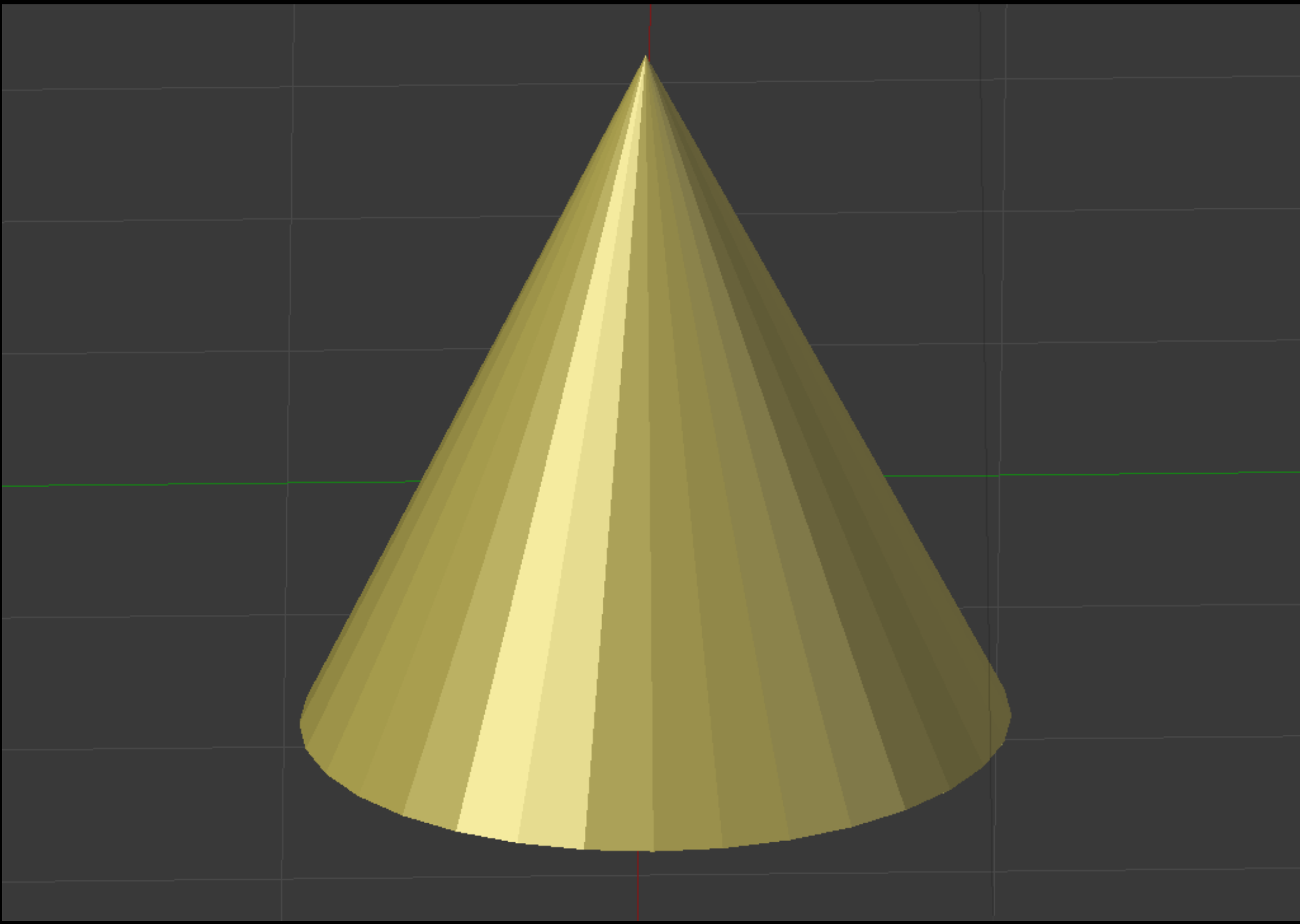
```
void main()
{
    vec3 normal = normalize(normalMatrix * vNormal) * normalSign;
    gl_Position = mvpMatrix * vPosition;
    texCoord = vTexcoord;

    float nDotL;
    float nDotH;
    color = material.emissive + (material.ambient * globalAmbient);

    for(int i = 0; i < NUM_DIR_LIGHTS; ++i)
    {
        if (dirLight[i].enabled)
        {
            nDotL = max(dot(normal, dirLight[i].direction), 0.0);
            nDotH = max(dot(normal, dirLight[i].hvector), 0.0);

            color += ((material.ambient * dirLight[i].ambient)
               + (material.diffuse * dirLight[i].diffuse * nDotL)
               + (material.specular * dirLight[i].specular
                  * pow(nDotH, material.shininess)));
        }
    }

    //alpha is in diffuse color
    color.a = material.diffuse.a;
}
```

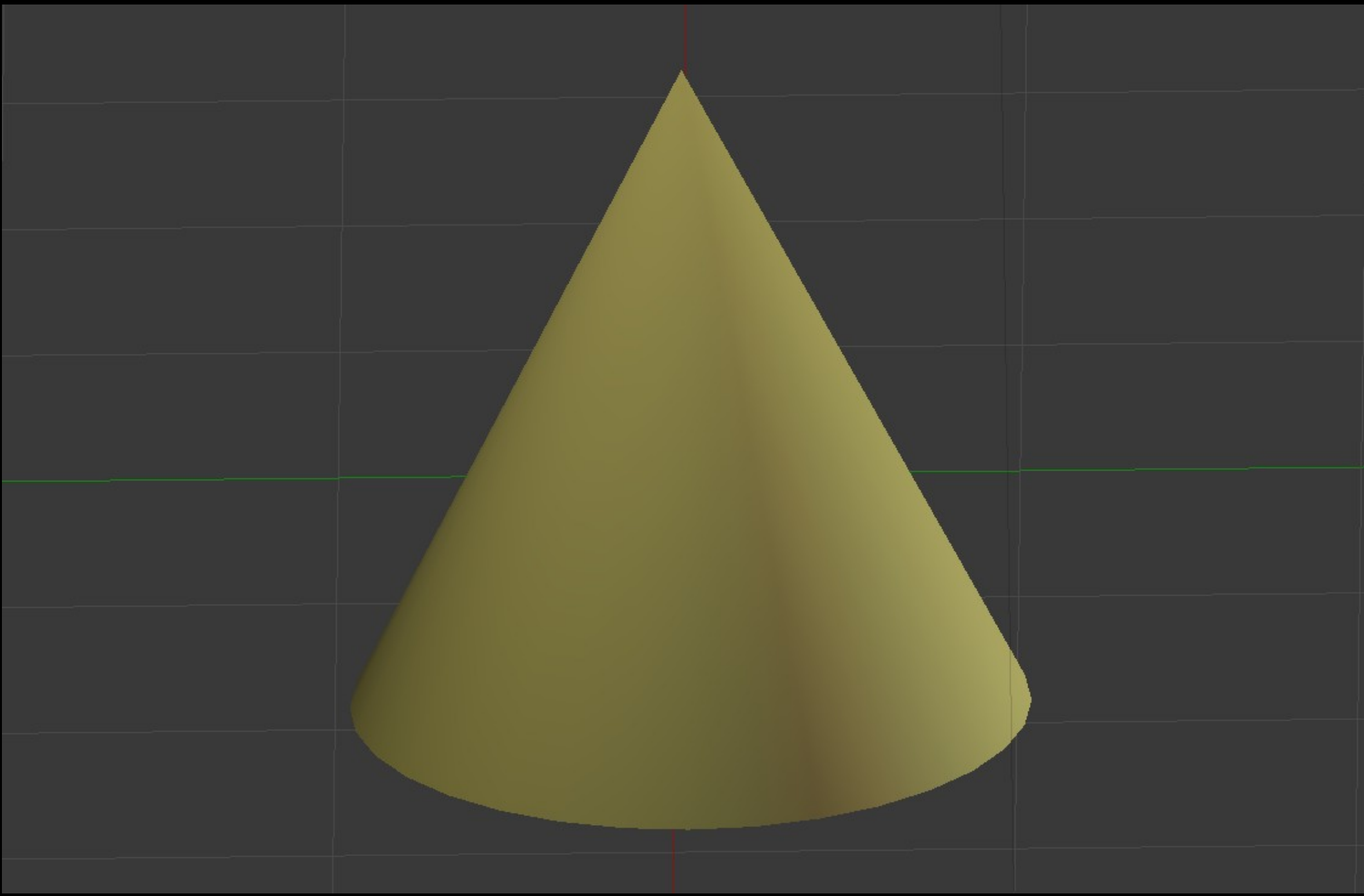
Interpolazione

- Colori e normali sono indicati nei vertici di ciascun triangolo
- La GPU interpola questi valori per ciascun frammento
- Se sono uguali nei vertici, sono costanti per tutto il triangolo





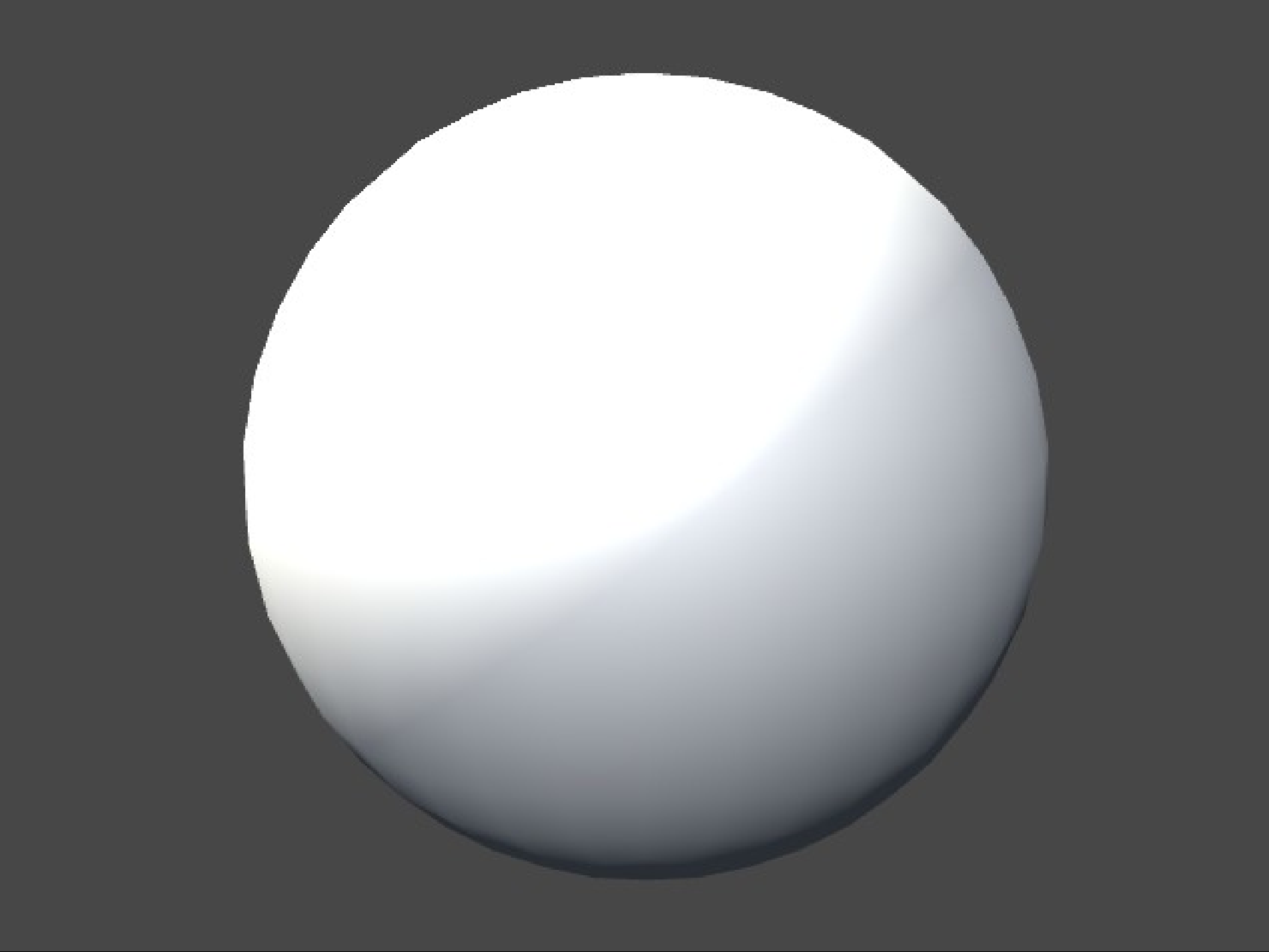




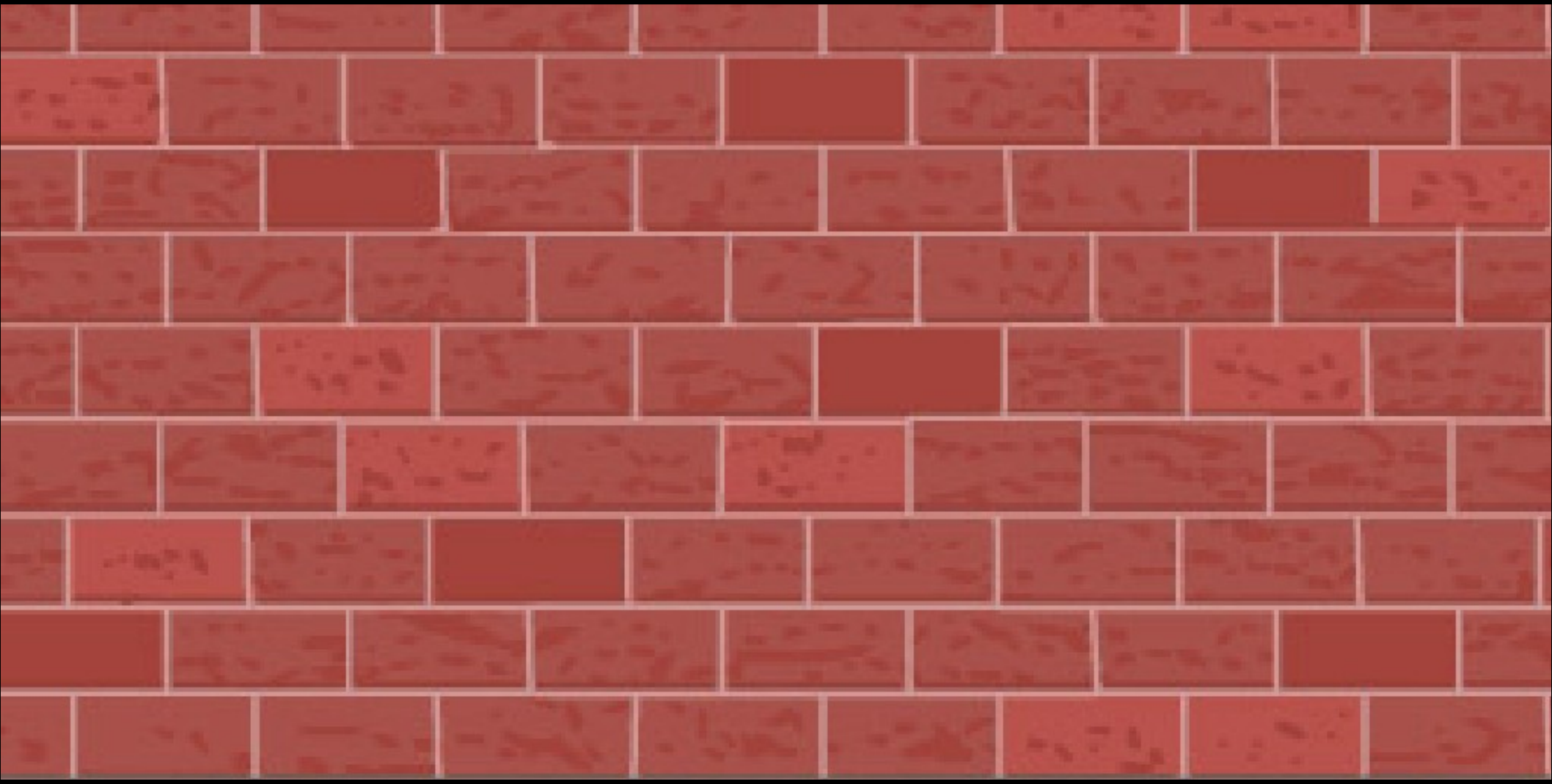
Texture

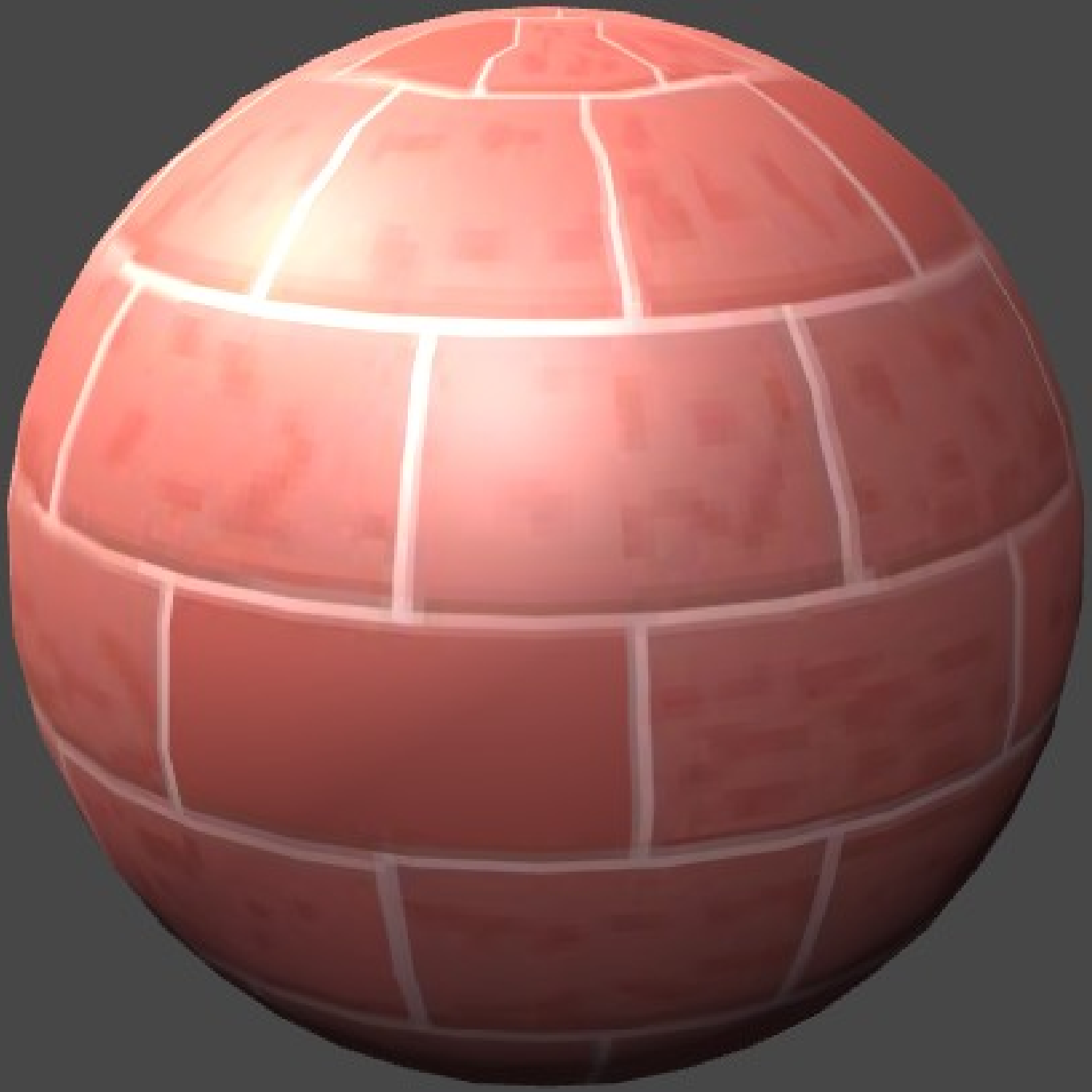
- Una texture è una immagine bitmap (convenzione: coordinate U,V)
- La texture viene mappata sulla mesh per darle un aspetto realistico
- La mappatura viene fatta dal fragment shader

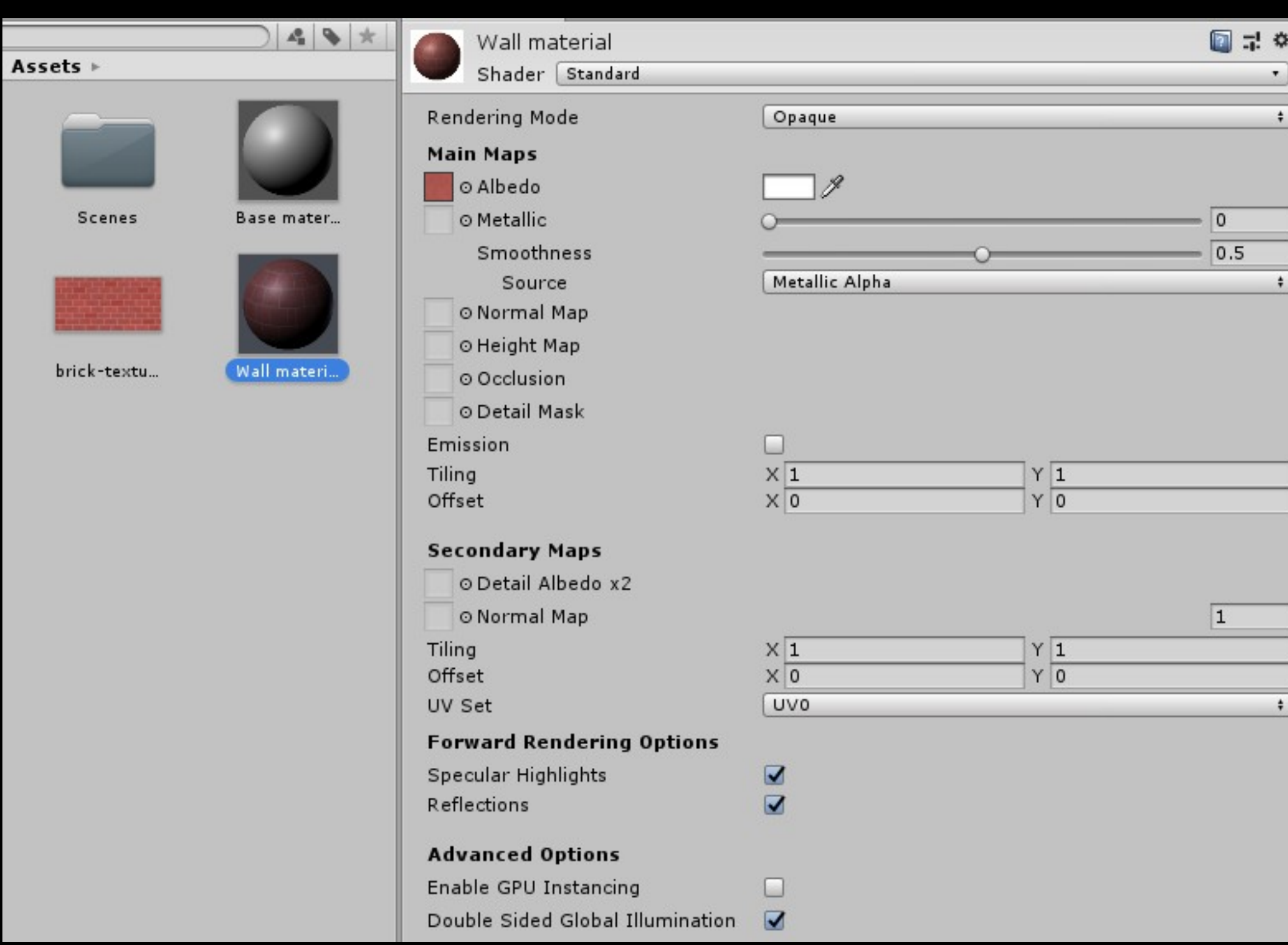
Si possono sovrapporre più texture



Texture image






Assets
Scenes
Base mater...
brick-textu...
Wall materi...
Wall material
Shader
Standard
Rendering Mode
Opaque
Main Maps
Albedo
Metallic
0
Smoothness
0.5
Source
Metallic Alpha
Normal Map
Height Map
Occlusion
Detail Mask
Emission
Tiling
X 1
Y 1
Offset
X 0
Y 0
Secondary Maps
Detail Albedo x2
Normal Map
1
Tiling
X 1
Y 1
Offset
X 0
Y 0
UV Set
UV0
Forward Rendering Options
Specular Highlights
Reflections
Advanced Options
Enable GPU Instancing
Double Sided Global Illumination

```
// adding texture color in fragment shader

precision mediump float;

uniform sampler2D   texture;

varying vec4 color;
varying vec2 texCoord;

void main()
{
    gl_FragColor = color * texture2D(texture, texCoord);
}
```


001 GAF

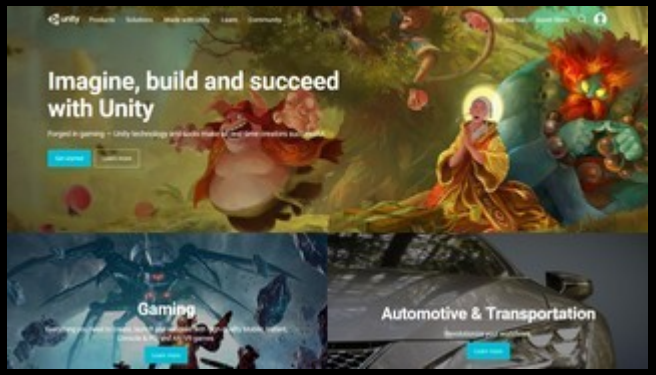
Normal map

- Una normal map è una forma particolare di texture
- Non contiene colori, ma le normali in ogni punto dell'immagine
- Serve per simulare irregolarità della superficie, coerenti con la luce

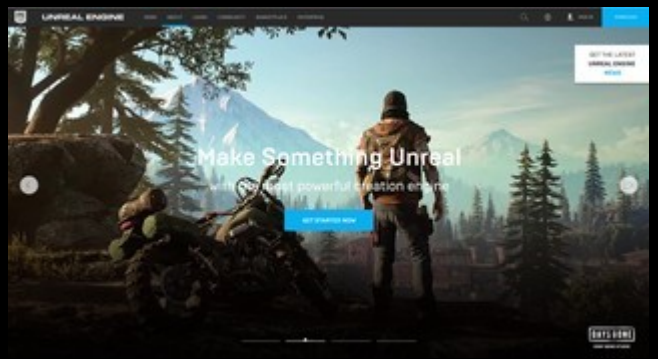
È più efficiente che complicare la mesh

Link: engine grafici

- **https://unity3d.com/**



- **https://www.unrealengine.com**



Link: OpenGL

- **https://www.khronos.org/**

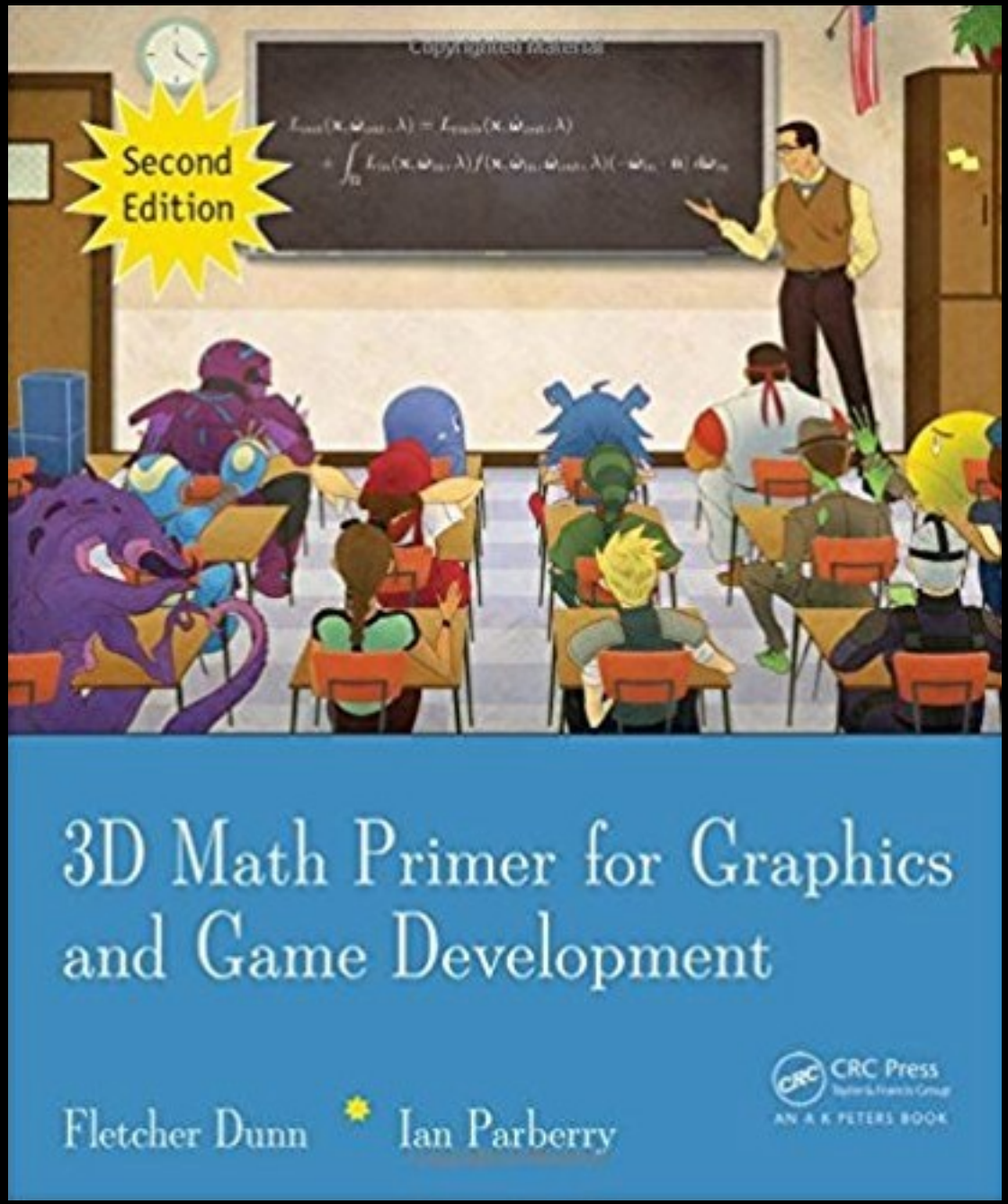
  OpenGL, OpenGL ES, GLM, WebGL & compagnia

  Link a vari tutorial

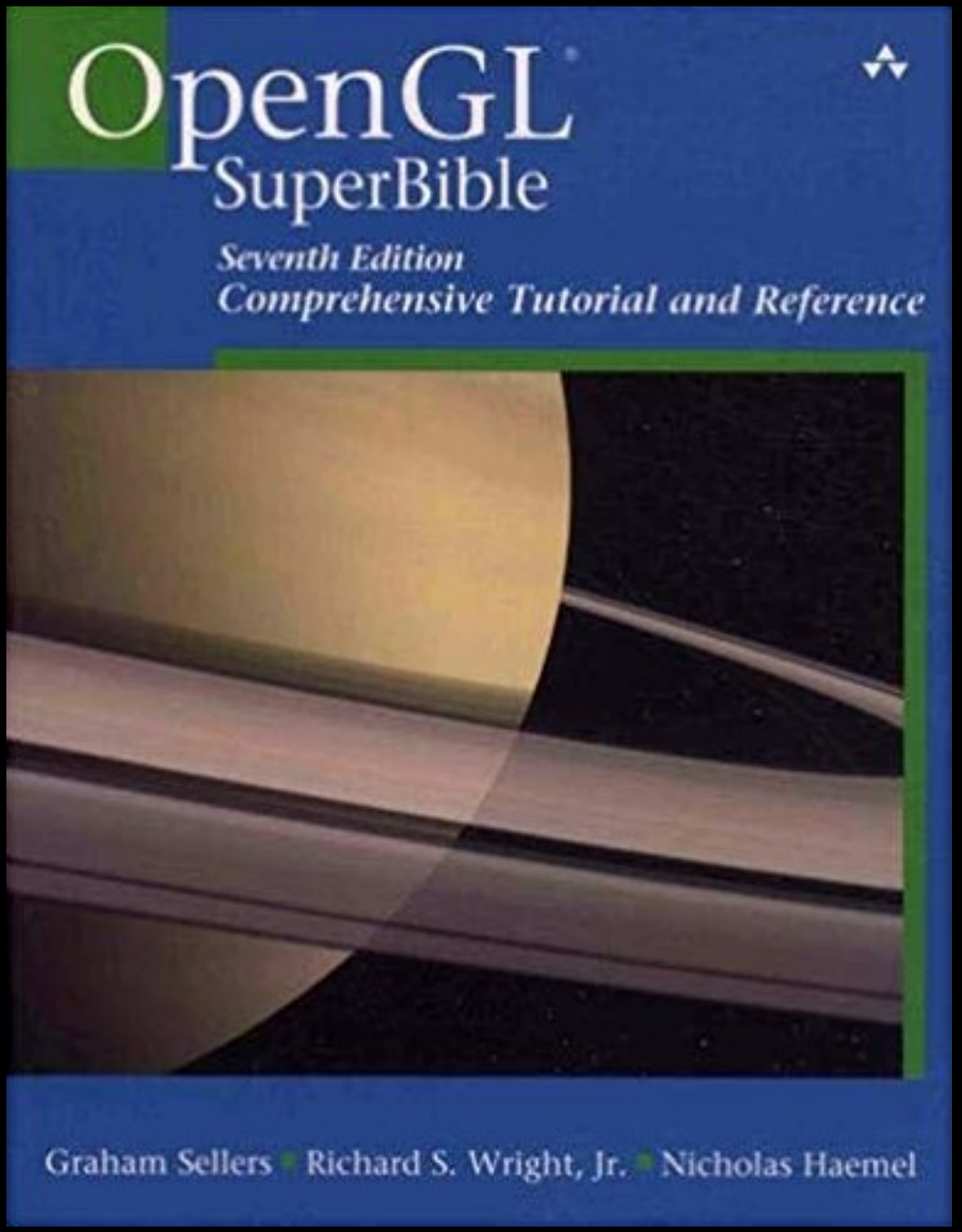
- **https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API**

  WebGL API (Mozilla)

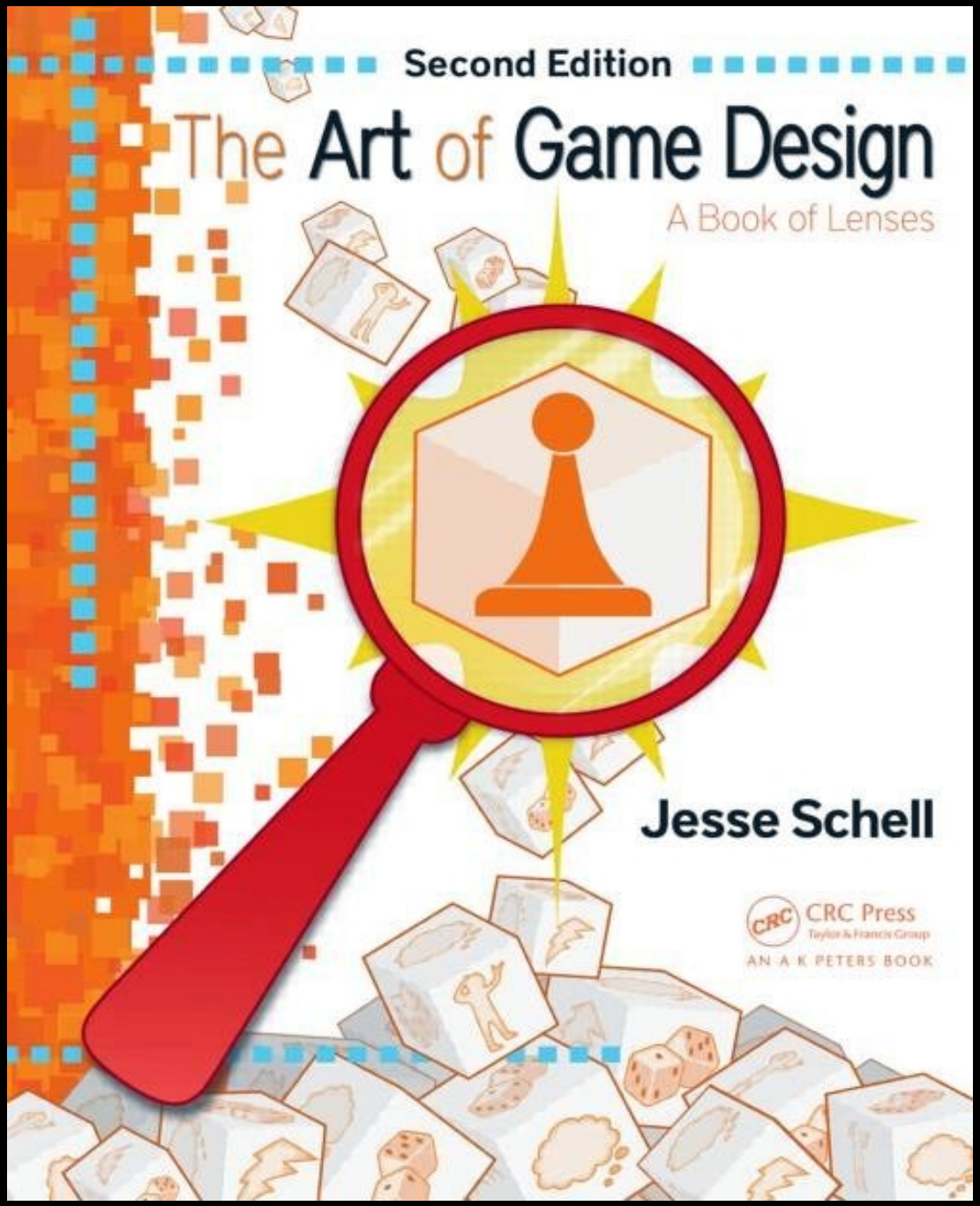
Per approfondire: geometria 3D



Per approfondire: OpenGL



La grafica non è tutto



That's all, folks!

µhackademy 1 Marzo 2019