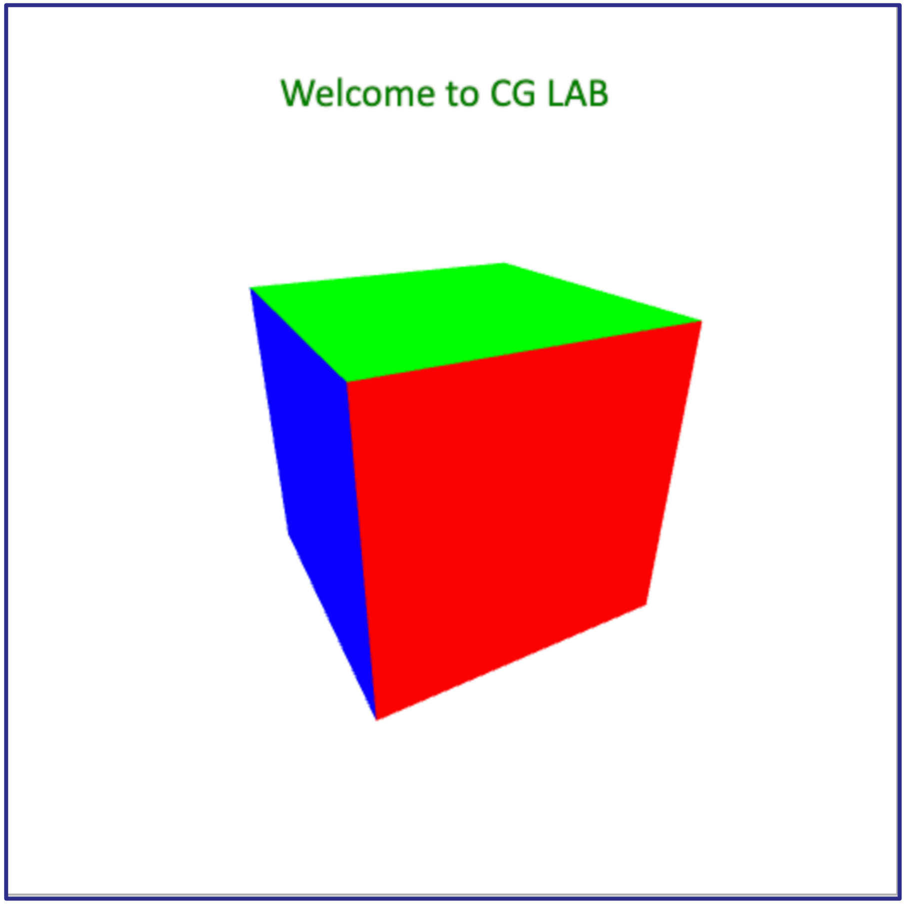
Trasformazione di Vista in WebGL

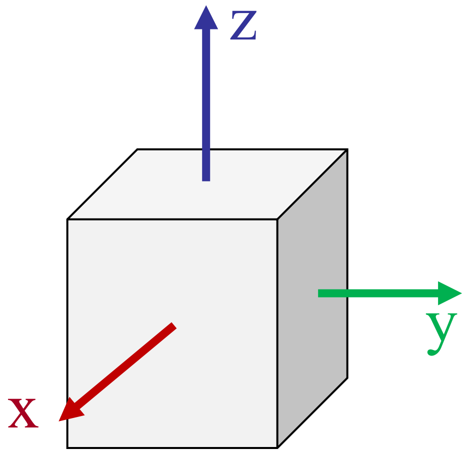


Definizione oggetto mesh 3D

In un sistema di riferimento Cartesiano xyzO destrorso, consideriamo un oggetto mesh 3D dando la lista dei suoi Vertici (coord. floating point) e la lista delle sue Facce (piane).

Vogliamo studiare cosa si deve fare per ottenere una sua rappresentazione grafica 2D (immagine 2D).

```
Coordinate (X,Y,Z) dei Vertices
1>  1.000000 -1.000000  1.000000
2> -1.000000 -1.000000  1.000000
3> -1.000000  1.000000  1.000000
4>  1.000000  1.000000  1.000000
5>  1.000000 -1.000000 -1.000000
6> -1.000000 -1.000000 -1.000000
7> -1.000000  1.000000 -1.000000
8>  1.000000  1.000000 -1.000000
Indici dei Vertices di ogni Face
1> 4 3 2 1
2> 8 7 3 4
3> 7 8 5 6
4> 5 1 2 6
5> 8 4 1 5
6> …
```



Geometria e Topologia

- definizione geometrica (dove sono posizionati nello spazio 3D i vertici)
- definizione topologica (come sono connessi i vertici da lati e facce)

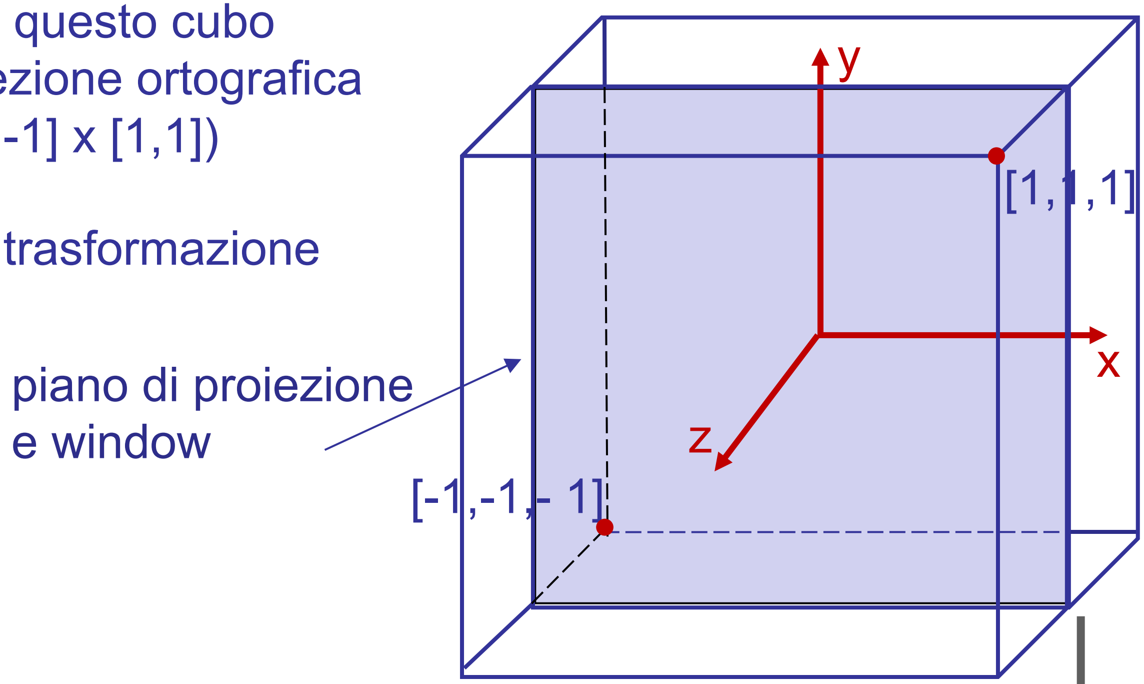
Sistema di Coordinate WebGL

In WebGL gli assi xyz sono quelli dell'Osservatore, ma destrorso, con l'asse z che indica la profondità.
Le coordinate sono limitate fra [-1,-1,-1] e [1,1,1]

WebGL non visualizzerà nulla che sia fuori da questo cubo.

Una geometria definita in questo cubo verrà visualizzata in proiezione ortografica sul piano xy (Window [-1,-1] x [1,1])

Quindi viene applicata la trasformazione Window $\longrightarrow$ Viewport

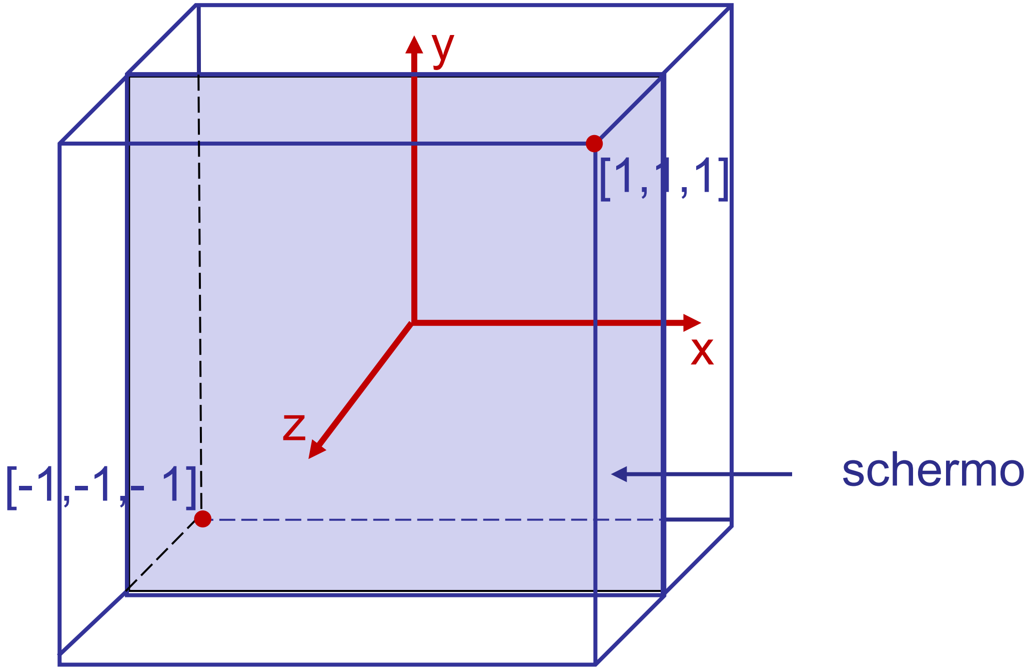


Trasformazione di Vista

Si devono definire:

1. la trasformazione del sistema di riferimento (x,y,z,O) → (xe,ye,ze,Vp)
2. la scala della piramide di vista $[xe',ye',ze',1]^T = S\ [xe,ye,ze,1]^T$
3. la prospettiva con profondità (xe,ye,ze,Vp) → (Xw,Yw,Zw,Ow)

   per portare la geometria nel cubo WebGL [-1,1]x[-1,1]x[-1,1].



Pipeline di Vista

Trasformazione Sistema di Riferimento
(x,y,z,O) → (xe,ye,ze,Vp)

↓

Scala della Piramide di Vista

↓

Clipping 3D rispetto al front e back plane

↓

Prospettiva con profondità
(xe,ye,ze,Vp) → (Xw,Yw,Zw,Ow)

↓

Clipping 3D rispetto alle pareti laterali del cubo

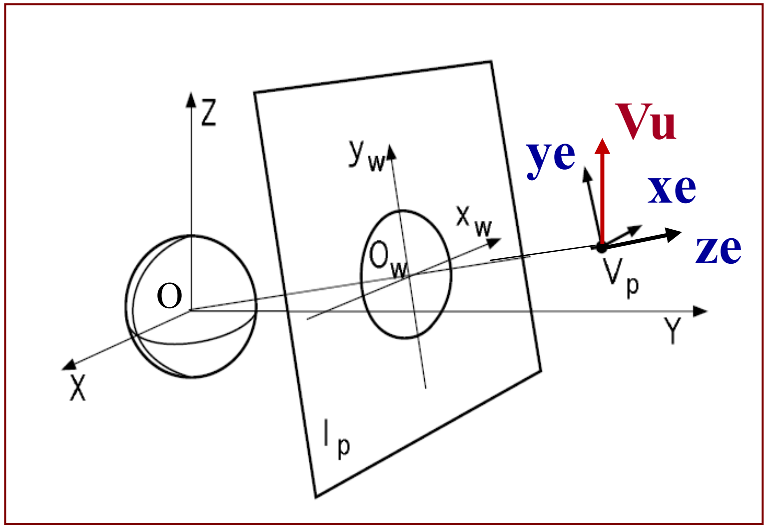
↓

Trasformazione Window-Viewport
(Xw,Yw,Ow) → (Xv,Yv,Ov)

Trasformazione Sistema di Riferimento

$$\mathbf{Vp} == [D \sin\varphi \cos\theta,\ D \sin\varphi \sin\theta,\ D \cos\varphi,\ 1]^T$$

$$\boxed{[xe,ye,ze,1]^T = VM\,[x,y,z,1]^T}$$



$$\mathbf{ze} = -(Vp-O) \,/\, \|Vp-O\|$$

$$\mathbf{xe} = (\mathbf{ze} \times \mathbf{Vu}) \,/\, \|\, \mathbf{ze} \times \mathbf{Vu} \,\|$$

$$\mathbf{ye} = -(\mathbf{ze} \times \mathbf{xe}) \,/\, \|\, \mathbf{ze} \times \mathbf{xe} \,\|$$

e poi: $\mathbf{ze} = -\,\mathbf{ze}$

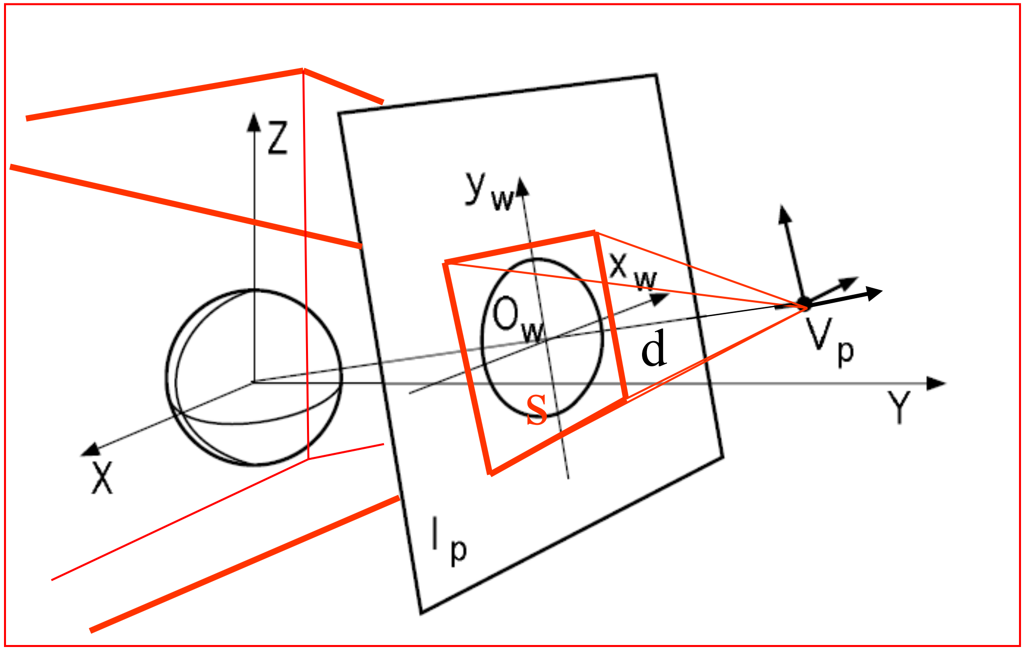
Trasformazione Sistema di Riferimento

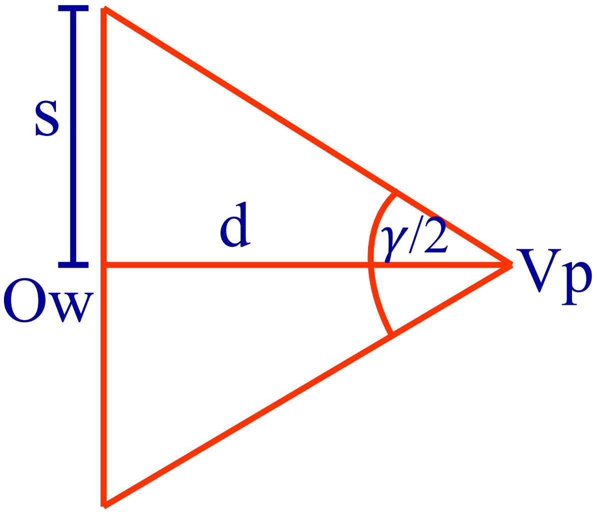
$$[xe,ye,ze,1]^T = VM\,[x,y,z,1]^T$$

$$B = \begin{pmatrix} \boxed{\mathbf{xe}} & \boxed{\mathbf{ye}} & \boxed{\mathbf{ze}} & \begin{matrix} D\sin\varphi\cos\theta \\ D\sin\varphi\sin\theta \\ D\cos\varphi \end{matrix} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$VM = B^{-1} = \begin{pmatrix} \boxed{\qquad\mathbf{xe}\qquad} & 0 \\ \boxed{\qquad\mathbf{ye}\qquad} & 0 \\ \boxed{\qquad\mathbf{ze}\qquad} & -D \\ 0 \quad 0 \quad 0 & 1 \end{pmatrix}$$

Piramide di Vista e Scala





$$s = d \cdot \tan(\gamma / 2)$$

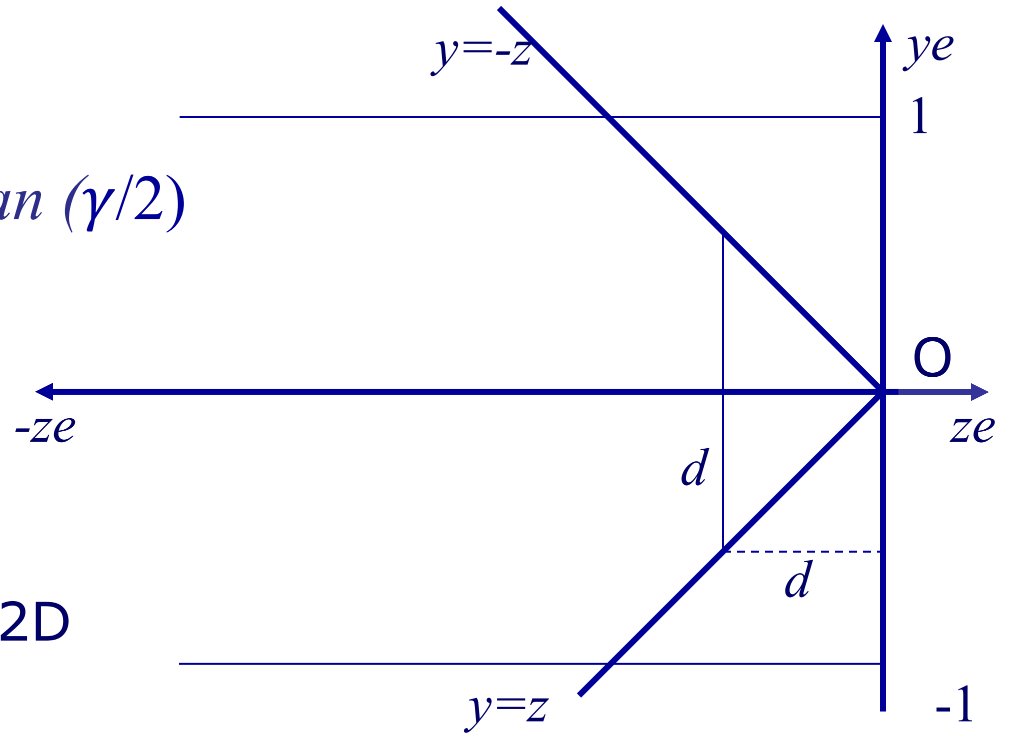
Prospettiva con Profondità

Si effettua la trasformazione di scala:

$[xe',ye',ze',1]^T = S\,[xe,ye,ze,1]^T$ con $S = \begin{pmatrix} d/s & 0 & 0 & 0 \\ 0 & d/s & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

ed

$s = d \tan(\gamma/2)$



Vista 2D

Prospettiva con Profondità

Ricordiamo il **Teorema** visto:
La proiezione

$$\begin{cases} xs = xe'/ze' \\ ys = ye'/ze' \\ zs = \alpha + \beta/ze' \end{cases}$$

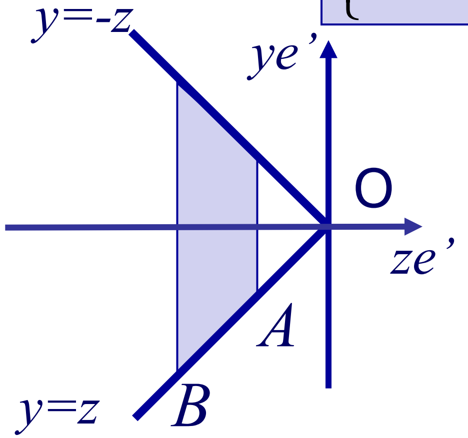
comporta una trasformazione dallo "spazio dell'osservatore" (xe',ye',ze',Oe) scalato allo "spazio del piano di proiezione" (xw,yw, zw,Ow) con la proprietà di trasformare rette in rette e piani in piani, per $\alpha, \beta \neq 0.$

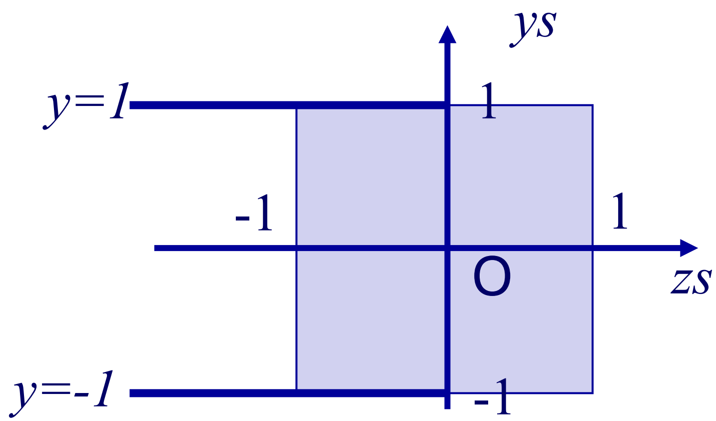
Prospettiva con Profondità

Determiniamo i valori α e β in modo che i valori ze' nell' intervallo $[B,A]$ con $A,B<0$ ($B<A$ *perché negativi*), siano trasformati in $zs \in [-1,1]$; sarà:

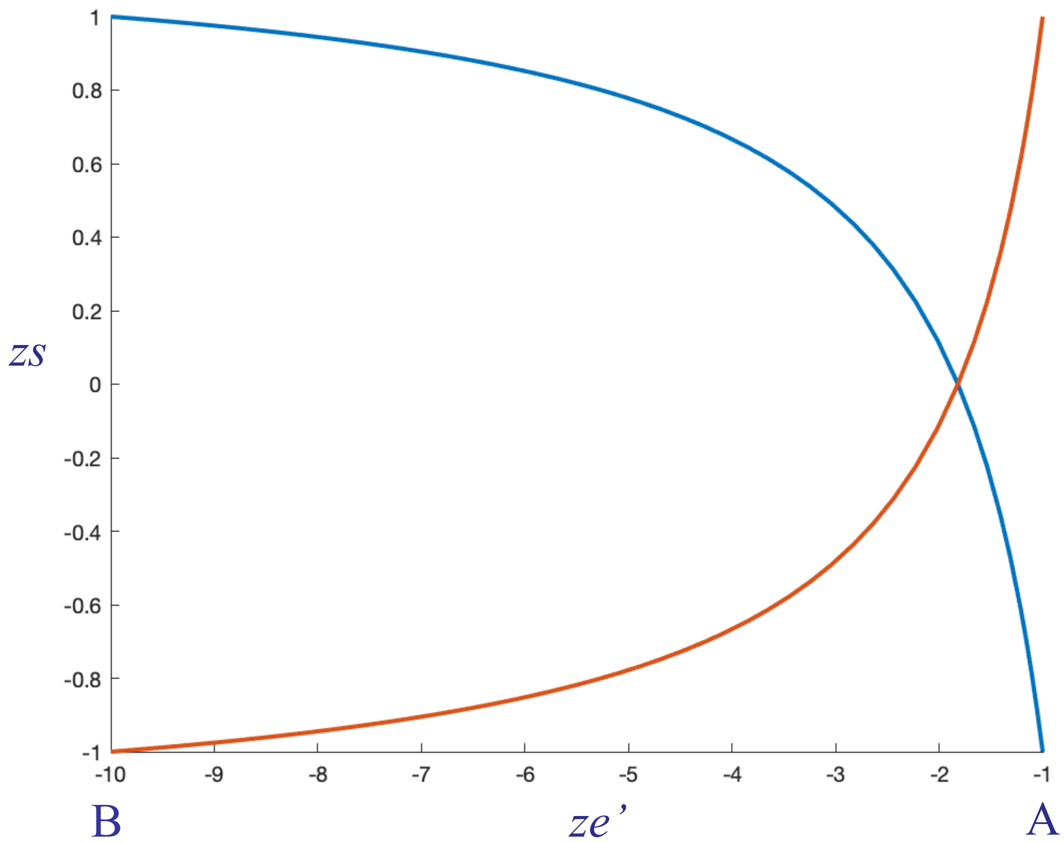
$$\begin{cases} 1 = \alpha + \beta / A \\ -1 = \alpha + \beta / B \end{cases} \qquad \begin{cases} A = A\alpha + \beta \\ B\alpha + \beta = -B \end{cases} \qquad \begin{cases} \beta = A(1-\alpha) \\ B\alpha + A(1-\alpha) = -B \end{cases}$$

$$\begin{cases} \beta = A(1-\alpha) \\ \alpha = -\dfrac{A+B}{B-A} \end{cases} \qquad \boxed{\begin{cases} \beta = \dfrac{2A \cdot B}{B-A} \\ \alpha = -\dfrac{A+B}{B-A} \end{cases}}$$





Immagini…amo



$zs = \alpha + \beta \,/\, ze'$

$[B,A] = [-10,-1]$

$ze' \in [B,A]$

$$\begin{cases} \beta = \dfrac{2A \cdot B}{B - A} \\[2ex] \alpha = -\dfrac{A + B}{B - A} \end{cases}$$

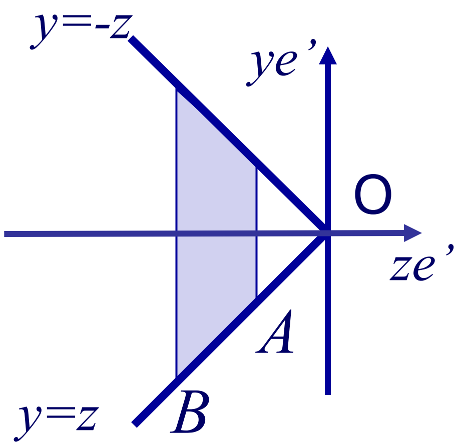
$zs = -\alpha - \beta \,/\, ze'$

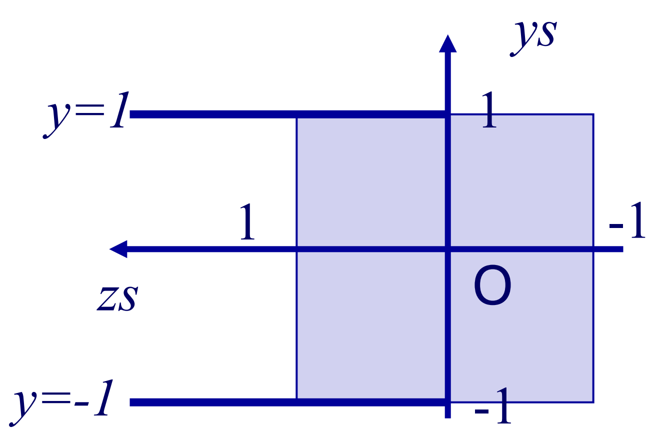
Prospettiva con Profondità

La trasformazione

$$zs = -\alpha - \beta \,/\, ze'$$

porta la piramide di vista nel cubo WebGL, ma mantenendo la relazione corretta di profondità ($-\beta > 0$), cioè i triangoli più lontani dall'osservatore hanno profondità (coordinata zs) maggiore.





Prospettiva con Profondità

per cui la proiezione in WebGL sarà:

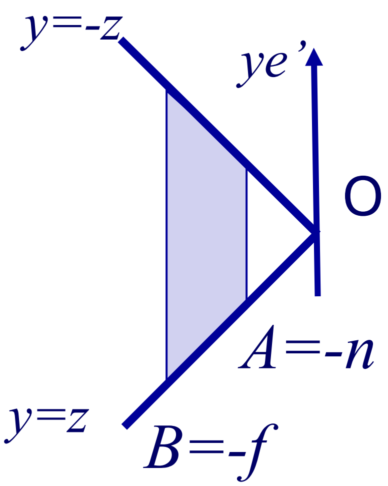
$$\begin{cases} xs = -\,xe' / ze' \\ ys = -\,ye' / ze' \\ zs = -\,\alpha - \beta / ze' \end{cases} \qquad \begin{cases} \beta = \dfrac{2A \cdot B}{B - A} \\ \alpha = -\dfrac{A + B}{B - A} \end{cases}$$

$$\begin{bmatrix} xs \\ ys \\ zs \end{bmatrix} = \begin{bmatrix} -\,xe\ d/s\ / ze' \\ -\,ye\ d/s\ / ze' \\ -\,\alpha - \beta / ze' \end{bmatrix}$$

ricordiamo che $ze' < 0$, per cui si correggono i segni in $-xe\ d/s\ / ze'$ e in $-ye\ d/s\ / ze'$ così da avere le prime due coordinate positive.

Osservazione

Essendo *A* e *B* negativi, spesso si indicano con –*n* (*near*) e –*f* (*far*), con *n* ed *f* positivi; in questo caso le formule per α e β diventano



$$\begin{cases} \beta = \dfrac{2A \cdot B}{B - A} = \dfrac{2 \cdot n \cdot f}{n - f} \\ \alpha = -\dfrac{A + B}{B - A} = \dfrac{f + n}{n - f} \end{cases}$$

Prospettiva con Profondità

Vediamo di scrivere queste trasformazioni in forma matriciale in coordinate omogenee:

$$[xs', ys', zs', ws']^T = P \cdot VM \cdot [x, y, z, 1]^T$$

quindi:

$$[xs, ys, zs, 1]^T = [xs'/ws', ys'/ws', zs'/ws', 1]^T$$

dove VM è quella data nelle slide precedenti e P è:

$$P = \begin{pmatrix} f/ar & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

con

$$f = d/s = 1/\tan(\gamma/2)$$

$$ar = \text{aspect-ratio (Viewport)} = width / height$$

Riassumendo

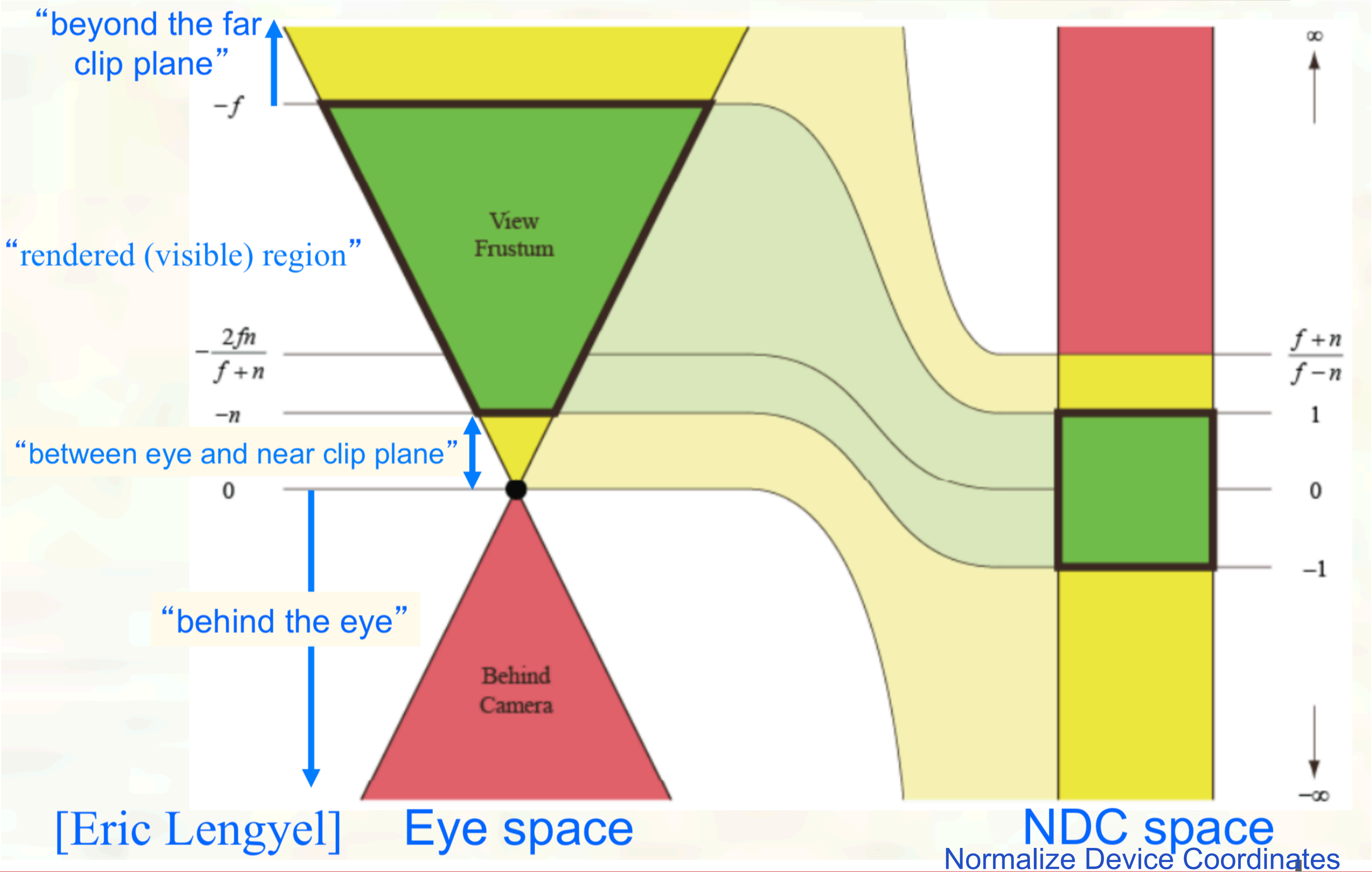
Infatti se consideriamo ar=1 possiamo ritrovare che:

$$\begin{bmatrix} xs' \\ ys' \\ zs' \\ ws' \end{bmatrix} = P \cdot VM \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = P \cdot \begin{bmatrix} xe \\ ye \\ ze \\ 1 \end{bmatrix} = \begin{bmatrix} xe\ d/s \\ ye\ d/s \\ \alpha\ ze + \beta \\ -ze \end{bmatrix}$$

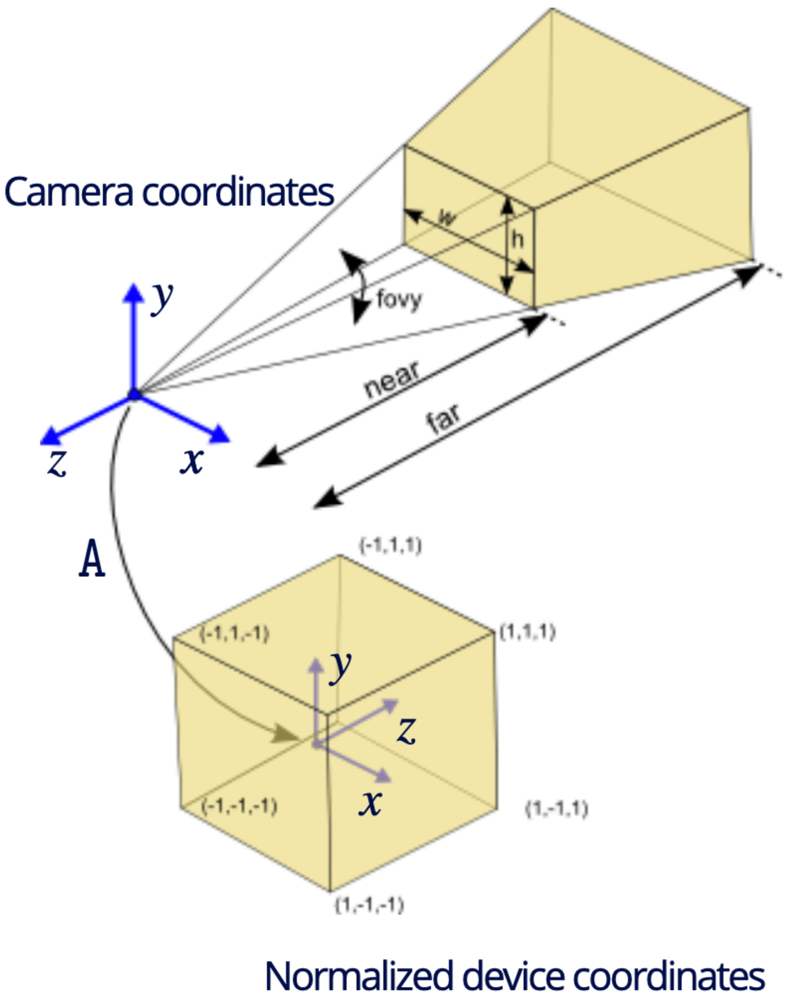
quindi, si procede con la divisione prospettica:

$$\begin{bmatrix} xs \\ ys \\ zs \\ 1 \end{bmatrix} = \begin{bmatrix} xs'/ws' \\ ys'/ws' \\ zs'/ws' \\ 1 \end{bmatrix} = \begin{bmatrix} -xe\ d/s\ /\ ze \\ -ye\ d/s\ /\ ze \\ -\alpha - \beta\ /\ ze \\ 1 \end{bmatrix}$$

Immagini…amo



Osservazione



Quando si usano le "normalize device coordinates" l'asse z viene rovesciato rispetto al sistema di coordinate della camera; si tratta quindi di un sistema di coordinate sinistrorso.

Osservazione

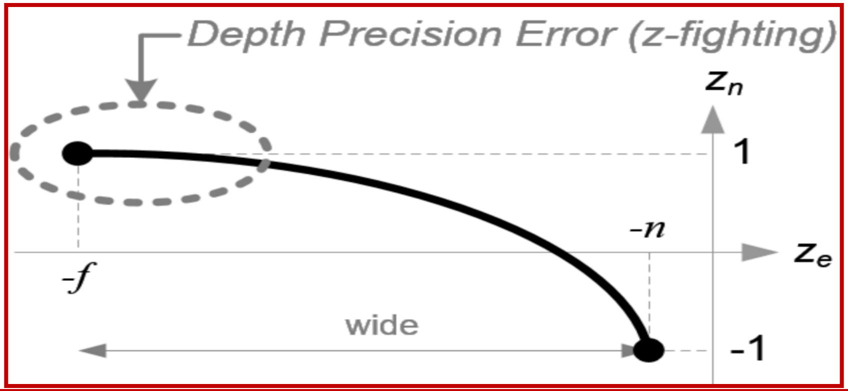
Prima di andare avanti, diamo ancora un'occhiata alla relazione

$$zs = -\alpha - \beta / ze'$$

Si nota che è una funzione razionale ed è una relazione non lineare tra zs e ze'.

Il suo andamento comporta una buona precisione sul piano *near*, ma bassa precisione sul piano *far*. Se l'intervallo [-*n*, -*f*] è grande si ha un problema di precisione sula profondità (z-fighting); un piccolo cambiamento di ze' vicino al piano *far* non influisce sul valore di zs.

La distanza tra *n* e *f* dovrebbe essere la più piccola possibile per ridurre al minimo il problema della precisione nel depth-buffer.



E il Clipping?

Il clipping viene effettuato dalla parte fissa della pipe-line, tra il vertex shader e il fragment shader ed è diviso in due parti:

1. Clipping 3D rispetto al front e back plane

Viene effettuato prima della divisione prospettica utilizzando le coordinate $[xs', ys', zs', ws']^T = [\ldots, \ldots, \ldots, -ze']^T$ infatti per essere rappresentati questi punti devono stare nella piramide di vista, per cui

$$B <= ze <= A$$

A questo punto, WebGL effettua la divisione prospettica, e tutto si trova in normalized device coordinate.

2. Clipping 3D rispetto alle pareti laterali del cubo

Viene effettuato utilizzando le coordinate $[xs, ys, zs, 1]^T$; per essere rappresentati questi punti devono essere tali che:

$$-1 <= xs <= 1 \qquad -1 <= ys <= 1$$

Demo

Vediamo alcuni differenti codici WebGL che visualizzano un cubo colorato:

Animazione: HTML5_webgl_1/cube_anim.html
Interazione: HTML5_webgl_1/cube_interact.html
Testo e Interazione: HTML5_webgl_1/cube_interact_and_text.html

Esempio in WebGL2: HTML5_webgl_1/cube_interact_webgl2.html

Esercizi 4

Modificare il codice cube_interact.html per:

1. gestire interattivamente i parametri di vista
2. sperimentare il clipping 3D rispetto al frustum
3. disegnare gli oggetti a linee
4. disegnare gli oggetti a facce e a linee (come nel logo)
5. sperimentare oltre la proiezione prospettica quella ortografica ed il frustum relativo
6. gestire più oggetti (mesh) mediante trasformazioni geometriche

Qualificatori Attribute e Varying

WebGL 2.0 si basa sulla versione OpenGL ES 3.0 (che usa lo shader preprocessor #version 300) e in questa i qualificatori attribute e varying sono stati sostituiti dai qualificatori in e out. Nel programma applicativo non cambia nulla.

Esempi di Vertex e Fragment Shader :

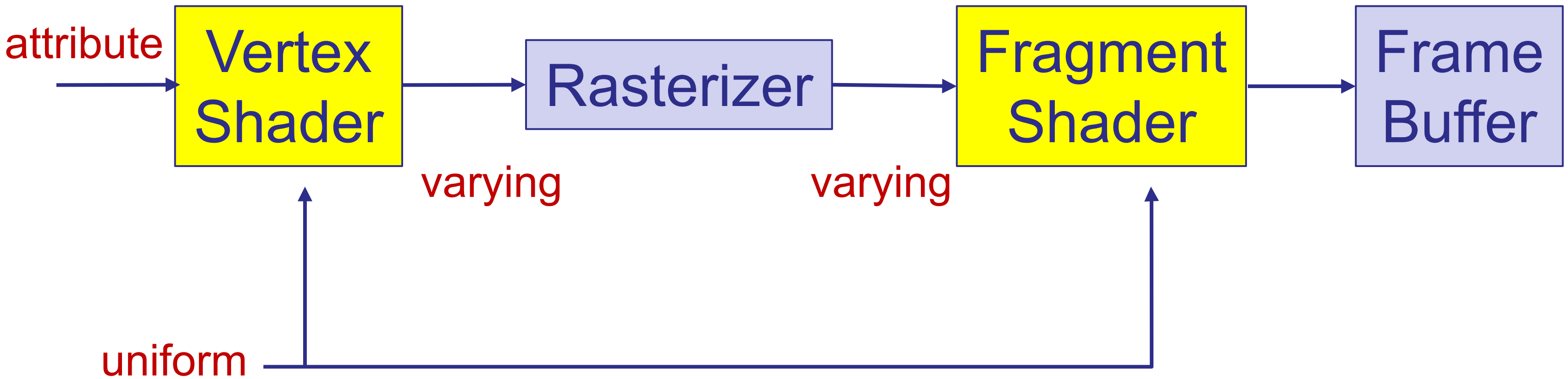
```
# version 100 es
attribute vec3 vPosition ;
varying vec3 color;
```

```
# version 300 es
in vec3 vPosition ;
out vec3 color;
```

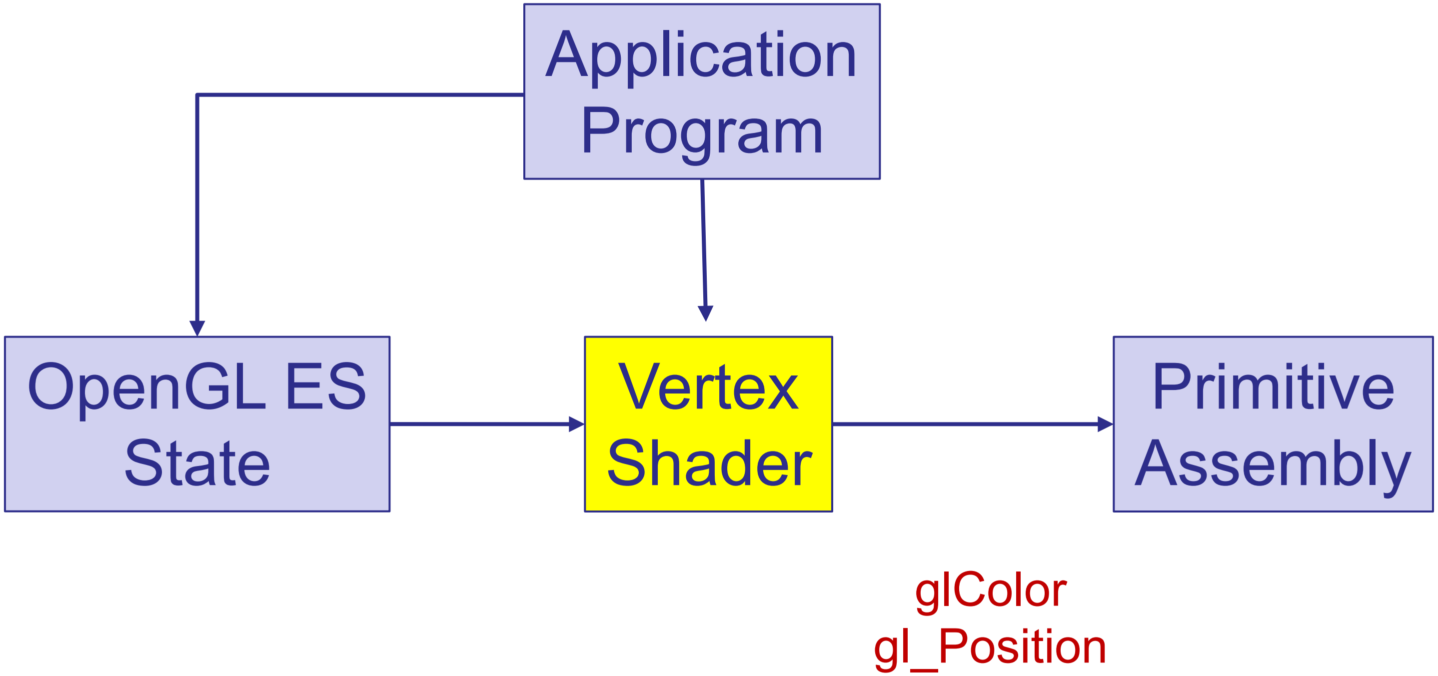
```
void main(void) {
    gl_FragColor = gl_Color ;
}
```

```
in vec4 color;
out vec4 FragColor ;
void main() {
    FragColor = color ;
   //gl_FragColor =color;
}
```

Qualificatori nella PipeLine

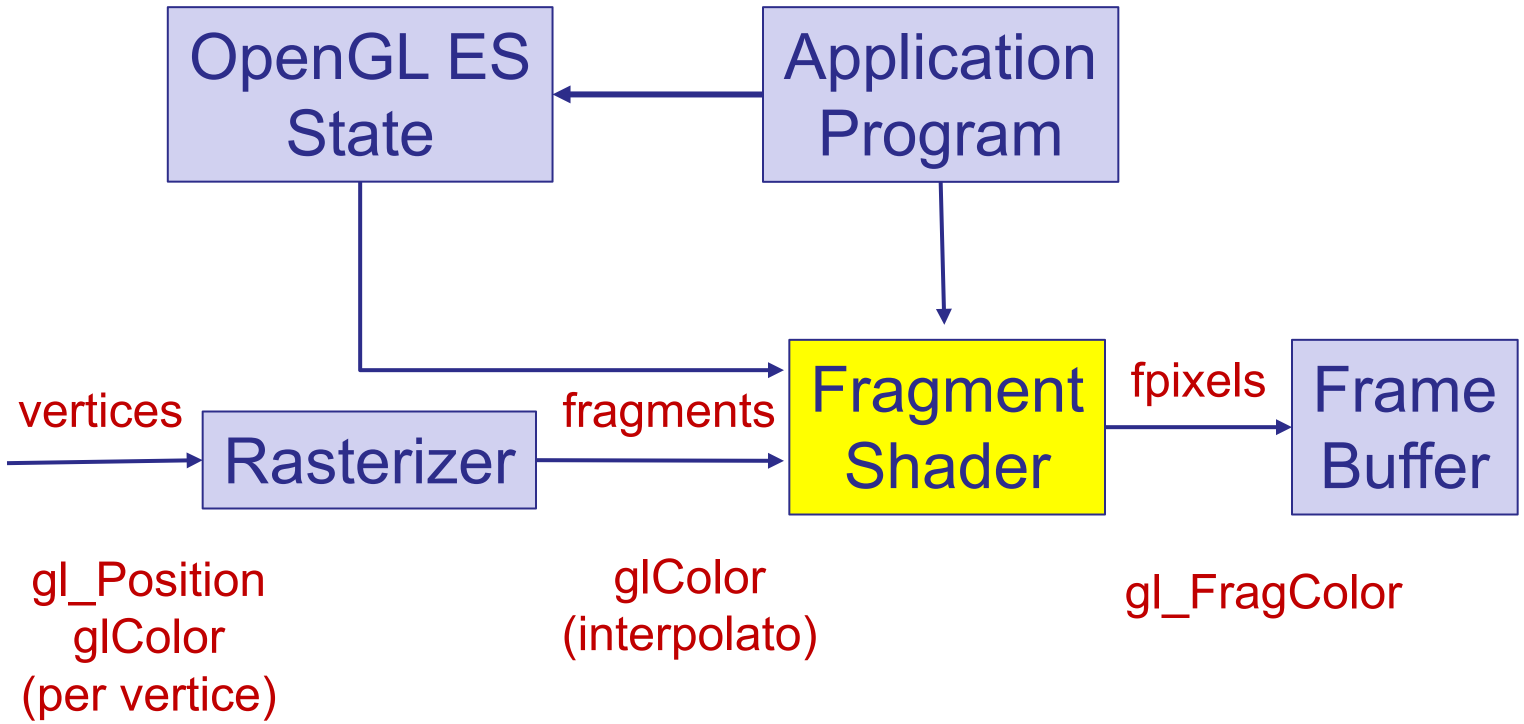


Architettura del Vertex Shader



La minima richiesta del Vertex Shader è di settare una position per il vertex assegnando un valore a gl_Position

Architettura del Fragment Shader



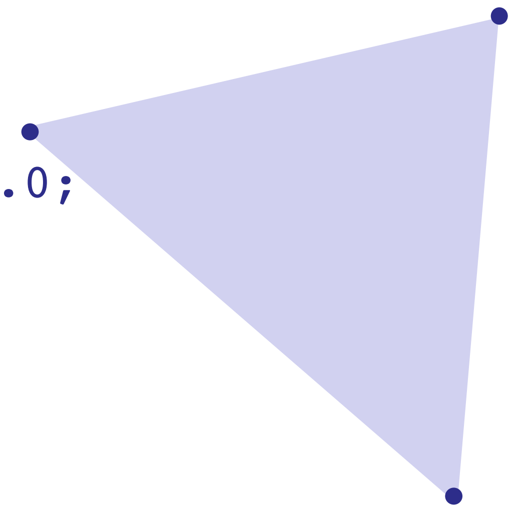
La minima richiesta del Fragment Shader è di assegnare un color al fragment settando gl_FragColor o scartandolo con 'discard'

Qualificatore Varying

Il vertex shader scrive i valori nelle variabili e il fragment shader legge i valori interpolati linearmente fra i vertici.

varying float intensity;

Deve essere definito nello stesso modo in entrambi gli shader;



L'interpolazione è sempre attiva e sempre 'perspective-correct'

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Giulio Casciola
Dip. di Matematica
giulio.casciola at unibo.it