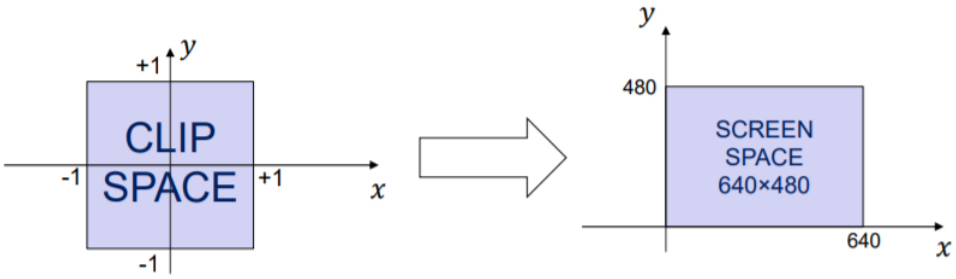
H1 Rasterization

H2 Introduzione

La **rasterizzazione** è il processo che prende in input i punti in spazio schermo e li rasterizza, cioè li converte in frammenti, che sono i pixel del monitor "coperti" dalla primitiva. Vi è quindi una fase preliminare che è il passaggio da spazio clip, che ha coordinate device independent, allo spazio schermo, che invece ha coordinate dipendenti dalla risoluzione del monitor.

H3 Da spazio clip a spazio schermo

Il rasterizzatore riceve i punti in spazio clip, ma per rasterizzare ha bisogno di considerare i pixel del monitor e quindi deve passare a spazio schermo.



Dobbiamo quindi portare:

- l'intervallo $[-1, +1]$ di x in $[0, resX - 1]$
- l'intervallo $[-1, +1]$ di y in $[0, resY - 1]$
- l'intervallo $[-1, +1]$ di z in $[0, 1]$

In spazio schermo, i pixel hanno quindi coordinate intere.

Per ottenere la trasformazione, ci basterà fare:

$$f_{toScreen} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \frac{x+1}{2} \cdot resX \\ \frac{y+1}{2} \cdot resY \\ \frac{z+1}{2} \end{pmatrix}$$

in questo modo prima gli intervalli sono tutti riportati in valori nell'intervallo $[0, 1]$, dopodiché x e y vengono moltiplicati per la risoluzione del monitor corrispondente. Nell'esempio della figura precedente avremo quindi i seguenti intervalli:

- l'intervallo $[-1, +1]$ di x in $[0, 640 - 1]$
- l'intervallo $[-1, +1]$ di y in $[0, 480 - 1]$
- l'intervallo $[-1, +1]$ di z in $[0, 1]$

H2 Rasterizzazione - descrizione

La rasterizzazione consiste nel prendere le primitive descritte dai punti in spazio schermo e individuare quali sono i pixel del monitor che vengono coperti dalle primitive in input. Tali pixel saranno i frammenti che saranno poi processati nell'ultima fase del rendering.
Un altro obiettivo del rasterizzatore è quello di prendere gli attributi memorizzati per vertice e interpolarli attraverso le coordinate baricentriche. Quindi in output oltre ai frammenti avremo anche gli attributi interpolati, che verranno calcolati per ogni frammento prodotto (ogni frammento avrà i suoi attributi)

Le primitive che siamo in grado di rasterizzare in modo efficiente sono principalmente 3:

- punti
- linee
- triangoli

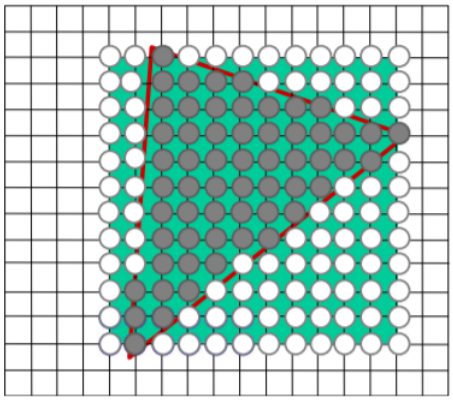
Proprio l'ultima primitiva è quella che ci interessa, in quanto ci rende possibile renderizzare mesh triangolari. Quindi analizzeremo la rasterizzazione di triangoli.

H3 Rasterizzazione di triangoli

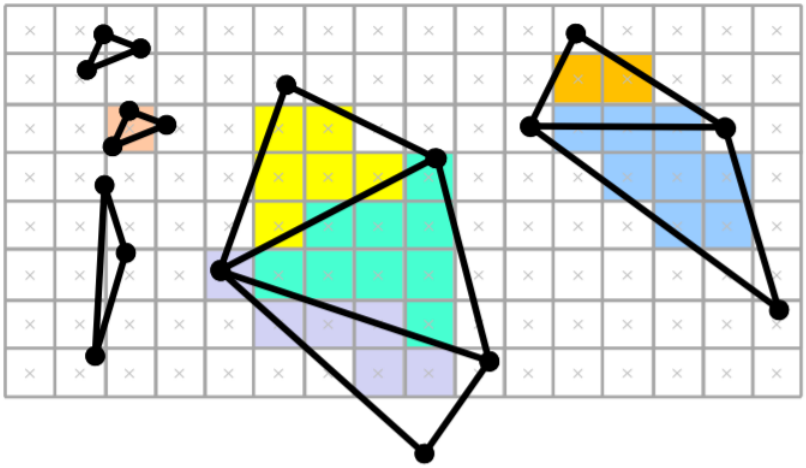
- *Input* : tre punti in 2D
- *Output* : frammenti

Il processo consiste nel trovare un rettangolo che contiene i tre vertici del triangolo. Dopodiché testare per ogni pixel in quel rettangolo se il pixel appartiene o meno al triangolo. Se sì quel pixel verrà mandato in output come frammento da processare, altrimenti no.

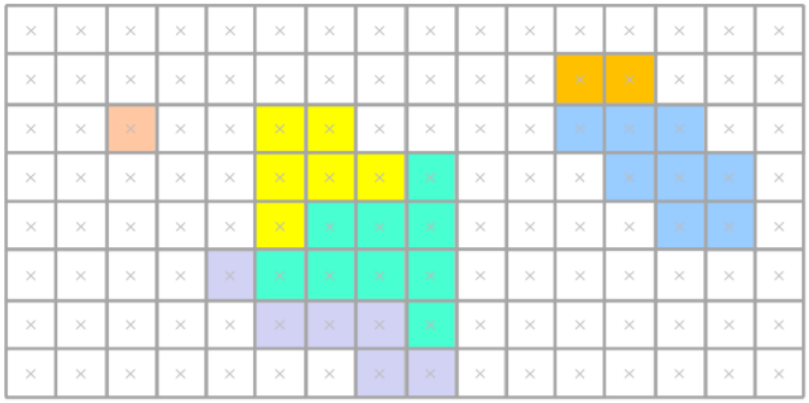
1. Confino il triangolo con un quadrato, all'interno del quale farò il processing



2. Faccio il test di appartenenza

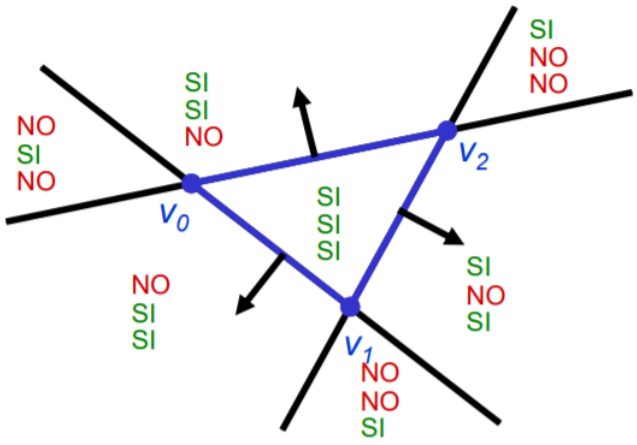


3. Mando in output i frammenti ottenuti

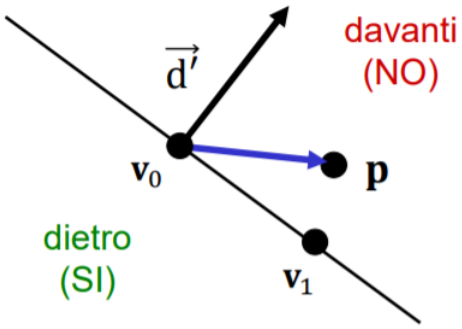


H4 Test di appartenenza

Si tratta di verificare se il centro del pixel appartiene al triangolo, cioè all'intersezione dei 3 semipiani generati dalle 3 rette:



Matematicamente, un modo per risolvere se un punto appartiene a un semipiano dato il segmento è il seguente:



Per capire se p appartiene o meno al semipiano "dietro", basta verificare se

$$(p - v_0) \cdot d' < 0$$

per le proprietà del prodotto dot. Per calcolare $\vec{d'}$, basterà considerarlo una rotazione di 90° del vettore $d = v_1 - v_0$:

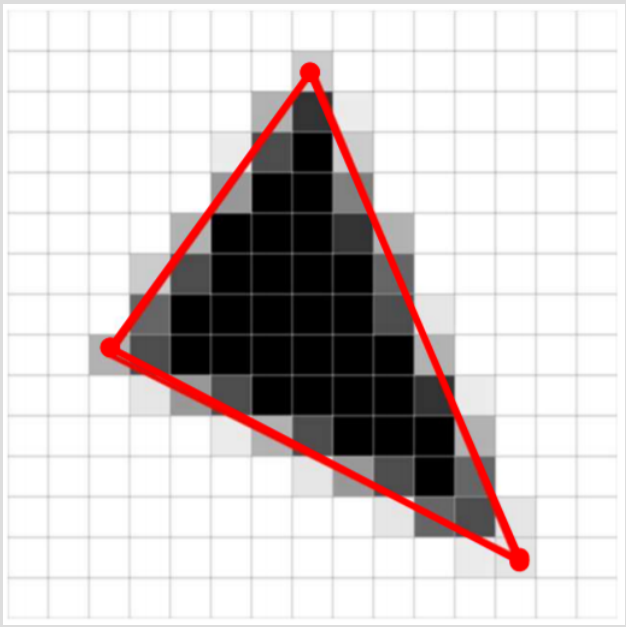
$$d = v_1 - v_0 = \begin{pmatrix} x \\ y \end{pmatrix} \Longrightarrow \vec{d'} = \begin{pmatrix} -y \\ x \end{pmatrix}$$

Questo test fa sì che è possibile che un triangolo troppo piccolo non generi frammenti.

H3 Considerazioni ulteriori

Se un triangolo è parzialmente fuori dal *viewport* (spazio schermo), verranno generati solo i frammenti interni al viewport (*clipping*).
Inoltre un edge condiviso fra due triangoli viene visualizzato a schermo senza buchi (il pixel appartiene per forza o a uno o all'altro triangolo) e senza sovrapposizioni (un frammento non appartiene ad entrambi i triangoli).

Quando la risoluzione è bassa la rasterizzazione approssima molto le forme dei triangoli. Per questo motivo sono stati sviluppate nel tempo delle tecniche per migliorare l'impatto estetico di tali approssimazioni, dette tecniche di ***antialiasing***.



esempio di antialising