H1 Strutture Dati Superficiali e Regolari

Ora vedremo tre nuove **strutture dati superficiali** che si basano su concetti molto simili:

- I campioni della superficie sono memorizzati su una **griglia regolare 2D**
- Hanno **connettività implicita** : ogni campione è *implicitamente* connesso ai propri vicini di griglia.

Esse sono:

- Campi di altezze
- Range Scans
- Geometry Images

H3 Vantaggi comuni di queste strutture

✔ La **connettività** è *implicita*, quindi non va memorizzata

✔ Le relazioni di **adiacenza** fra elementi sono *implicite*

✔ **Multirisoluzione** semplice da ottenere (allo stesso modo delle immagini)

✔ Il **geometry processing** su queste strutture è facile da parallelizzare

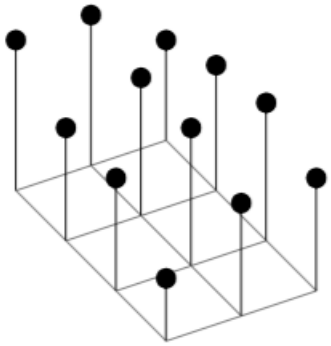
✔ Diverse *analogie* con le immagini 2D.

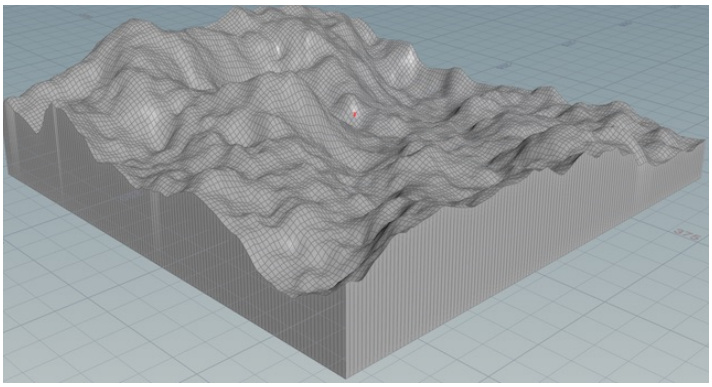
H3 Svantaggi comuni di queste strutture

✘ Risoluzione **non adattiva**

✘ Espressività *limitata*

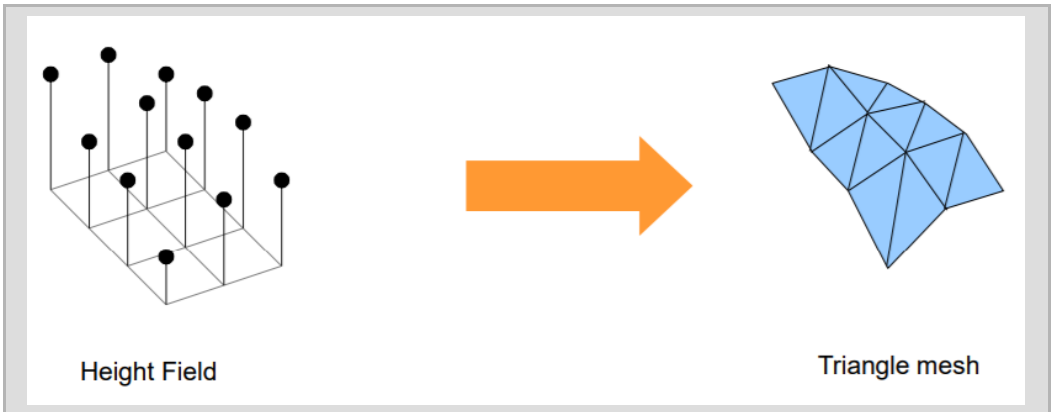
H2 Campi di altezze





```
float[][] height = new float[resX][resY];
```

Si tratta di un array bidimensionale di valori *scalari*, ciascuno dei quali rappresenta un'**altezza**. Ciò significa che il valore di coordinate `(x,y)` corrisponde al punto 3D `(x, y, height[x][y])`. Questo implica che 2 delle 3 coordinate sono implicite, poichè sono gli indici della griglia. Un'altra cosa implicita è la **connettività**, in quanto i vertici connessi ad un certo vertice sono quelli delle celle vicine. Ciò genera grosso risparmio di memoria.



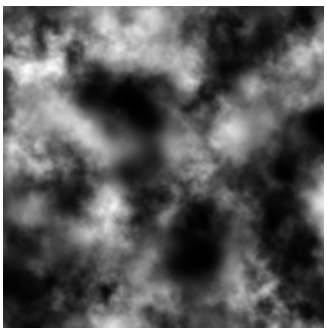
La connettività implicita ne fa una triangular mesh

In più questo tipo di rappresentazione ha i vantaggi delle immagini 2D riguardo la **multirisoluzione** e l'**editabilità**. Inoltre sarà **regolare** in quanto la *valenza* dei vertici è fissata dalla natura matriciale. Non è però una struttura dati adatta alla **risoluzione adattiva**, in quanto la risoluzione è fissa, e dipende dalla dimensione della griglia.

In realtà questo tipo di modello viene definito più propriamente **modello 2.5D**, in quanto non tutti i punti nello spazio sono rappresentabili, e non quindi tutti i tipi di modelli; ad esempio gli oggetti cavi, o con delle rientranze. Questo poiché i vertici hanno coordinate *x* e *y* 'univoche', ma la terza coordinata non lo è, in quanto dipende dall'unico valore presente dentro la rispettiva cella.

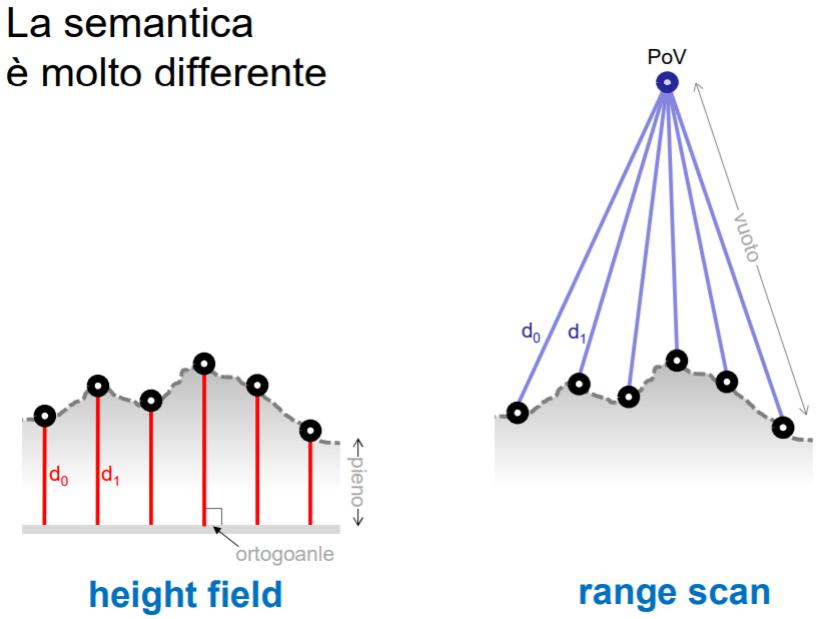
I campi di altezze sono però dei validissimi candidati per quanto riguarda la rappresentazione di terreni, proprio perchè questi difficilmente presentano rientranze.

Infine, spesso vengono rappresentati attraverso **scale di grigi**:



H2 Range scans

Le **range scan** sono una struttura dati interessante poiché sono ottenute spesso attraverso tecniche di acquisizione automatica. Essa è per certi versi simile ai campi di altezze, ma con una semantica opposta:



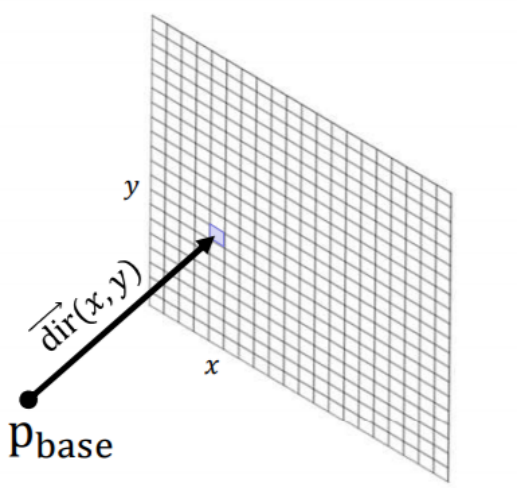
Una range scan è un array 2D di scalari, come i campi di altezze, ma tali scalari non rappresentano altezze di punti in una certa località *(x, y)*, bensì la distanza fra un punto *(x, y)* e l'osservatore.

```
float[][] rangeScan = new float[resX][resY]
```

d_0	d_1	d_2	d_3	d_4
d_5	d_6	d_7	d_8	d_9
d_{10}	d_{11}	d_{12}	d_{13}	d_{14}
d_{15}	d_{16}	d_{17}	d_{18}	d_{19}
d_{20}	d_{21}	d_{22}	d_{23}	d_{24}
d_{25}	d_{26}	d_{27}	d_{28}	d_{29}

`rangeScan[x][y]` può valere uno scalare d_k oppure un valore invalido (ad es. 0) nel caso in cui si rappresenti un punto 'vuoto'. Per calcolare le coordinate 'spaziali' di un punto usiamo i 3 valori `(x, y, rangeScan[x][y])`, in particolare avremo:

$$p_{base} + rangeScan[x][y] \cdot \vec{dir}(x, y) \tag{1}$$



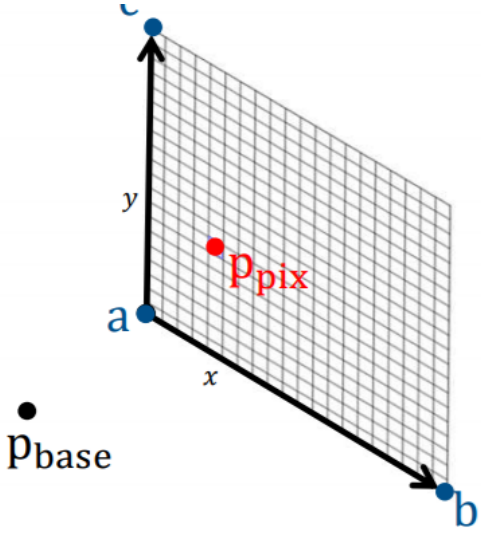
(1) funziona poiché noi stiamo scalando il vettore $\vec{dir}(x, y)$ con il valore memorizzato nella rispettiva cella, e sommiamo il vettore scalato al punto p_{base} per ottenere la profondità spaziale del punto.

È quindi necessario conoscere sia p_{base}, cioè il PoV, sia il vettore $\vec{dir}(x, y)$, il quale è calcolabile nel seguente modo:

$$p_{pix} - p_{base}$$

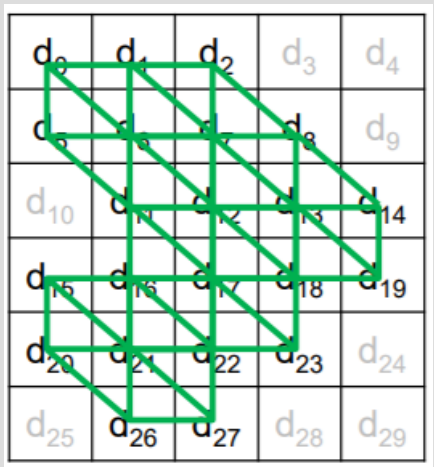
con

$$p_{pix} = a + (b - a)\frac{x}{resX} + (c - a)\frac{y}{resY} \tag{2}$$



L'espressione (2) funziona poiché noi stiamo scalando il vettore $b - a$ per il valore risultante da $\frac{x}{resX}$, e ciò che otteniamo sarà il vettore $x - a$. Allo stesso modo otteniamo $y - a$. Infine sommiamo al punto a i due vettori, e arriveremo al punto p_{pix}.

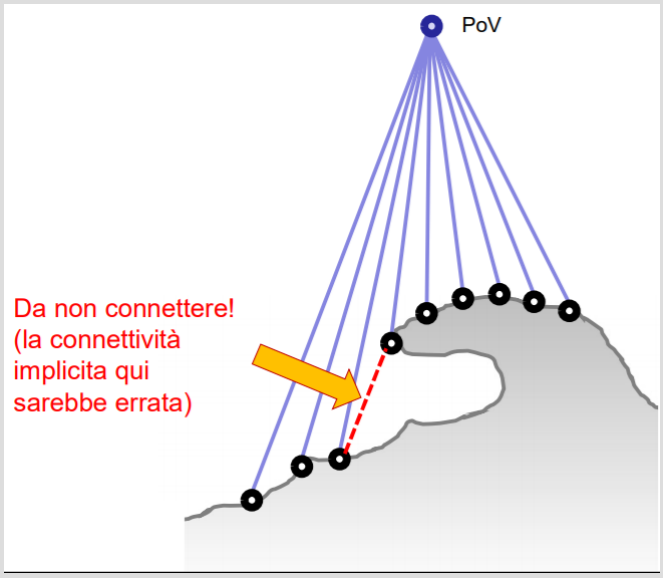
Anche se come abbiamo visto la semantica è diversa, le range scan condividono molti pregi e difetti dei campi di altezza. Hanno **connettività implicita**, memorizzano solo 1 coordinata su 3, sono **regolari**, sono adatti alla **multirisoluzione** e sono editabili come immagini 2D. Non sono adatte alla **risoluzione adattiva**. Sono anch'esse rappresentabili come **scala di grigi**.



La connettività implicita genera una triangular mesh

Scala di grigi per range scan

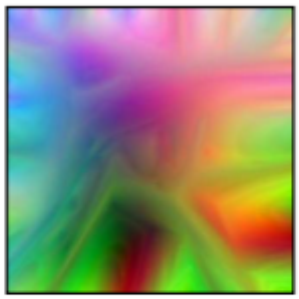
Anche le range scan sono **modelli 2.5D**, in quanto catturano 'solo un lato' dell'oggetto. Infatti potremmo avere dei **depth jumps**, cioè dei punti non memorizzati poiché coperti da qualcos'altro in primo piano. Non potendo accedere a quei punti della superficie dell'oggetto, non possiamo dire di trovarci di fronte a un modello 3D vero e proprio.



Depth jump

È però possibile ottenere un modello effettivamente 3D acquisendo un **set di range scan** dello stesso oggetto da angolazioni diverse. Spesso l'output di questo processo viene convertito facilmente in una **point cloud**, e usando quindi le sue tecniche di geometry processing, come il calcolo delle normali attraverso il *best fitting plane* e l'algoritmo ICP per l'allineamento di range scan diverse.

H2 Geometry Images



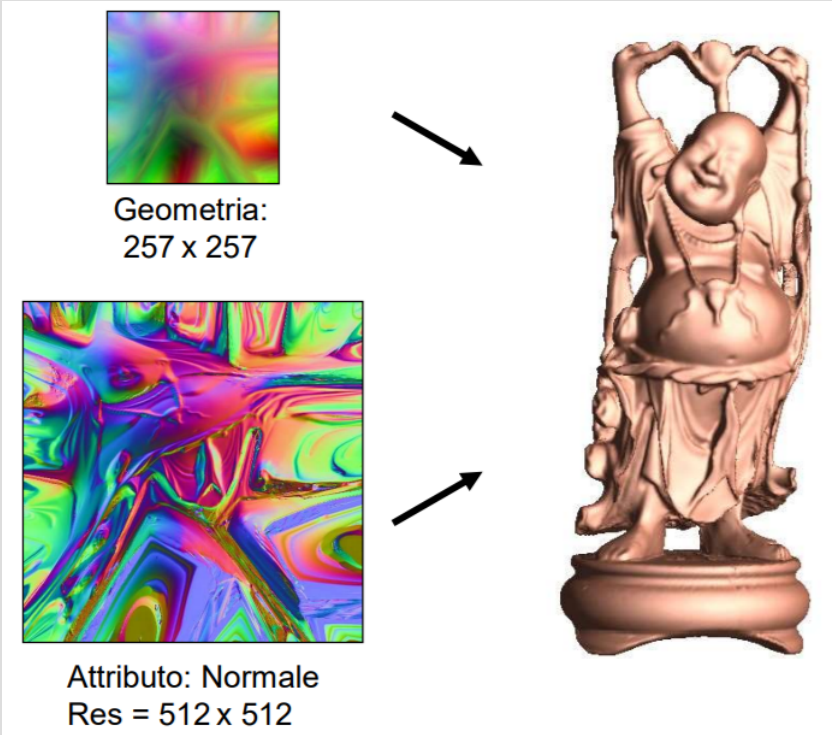
XZY per punto
(mostrato come RGB)

```
Point[][] geoImage = new Point[resX][resY]
```

Anche in questo caso abbiamo un array 2D per memorizzare il modello, ma stavolta i valori all'interno delle celle saranno le tre coordinate *(x, y, z)*. Ogni cella sarà quindi un punto 3D della mesh, mantenendo tutti i vantaggi delle strutture viste in precedenza riguardo la **regolarità**, la **connettività implicita**, il risparmio di memoria, la **multirisoluzione**, la capacità di compressione e l'**editabiltà** come immagine 2D etc.

Non solo, in questo caso ci troviamo di fronte alla geometria di un vero e proprio modello 3D, in quanto ciascun punto della superficie è identificato dalle 3 coordinate spaziali.

Anche gli attributi possono essere memorizzati, semplicemente usando ulteriori griglie regolari con risoluzione pari o superiore a quella della matrice della geometria.



La griglia che rappresenta la geometria ha risoluzione inferiore a quella che rappresenta le normali della mesh

Tutti questi vantaggi vengono a un costo molto oneroso. Infatti ciò che ci permette di convertire una normale mesh in una geometry image è un *UV-mapping* ancora più restrittivo di quello delle *texture*, in quanto deve essere completo e senza isole.