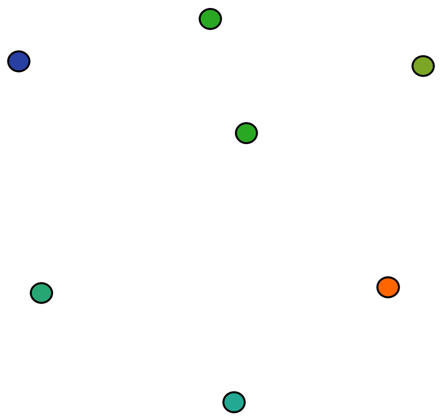
Advanced Data Representation

Visualisation – Lecture 8

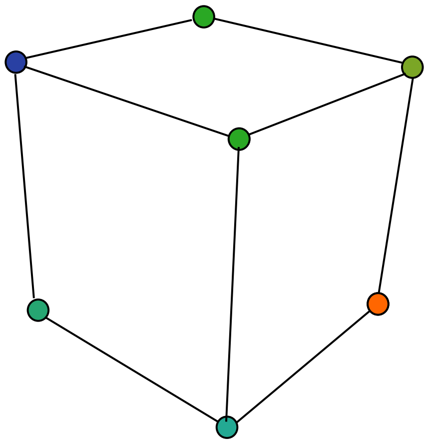
Taku Komura

Institute for Perception, Action & Behaviour
School of Informatics

Data Representation : Recap



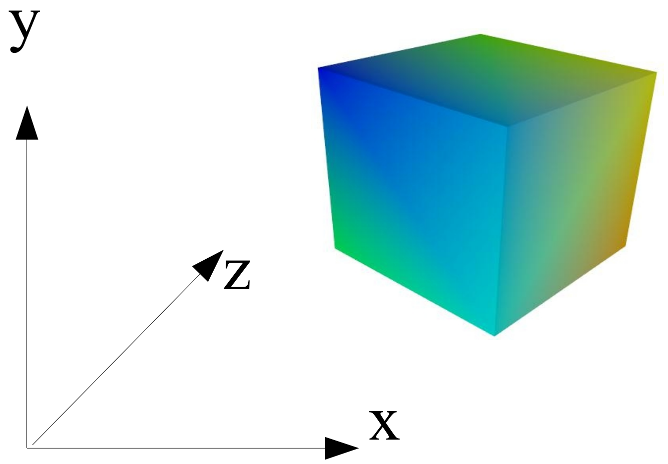
- Data is **discrete**
  - only know **value at points**
  - need to **interpolate** cells
- **Dataset**
  - **Structure : Topology & Geometry**
    - defined by **cells; regular** or **irregular**
  - **Value**
    - actual data itself
    - **attribute data** types : scalar, vector, tensor etc.



Overview

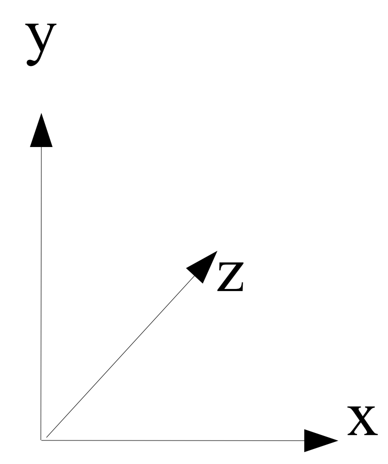
- **Scalar Algorithms** (last 2 lectures)
  - **contouring & colour mapping**
  - relied heavily on **spatial interpolation in cells**

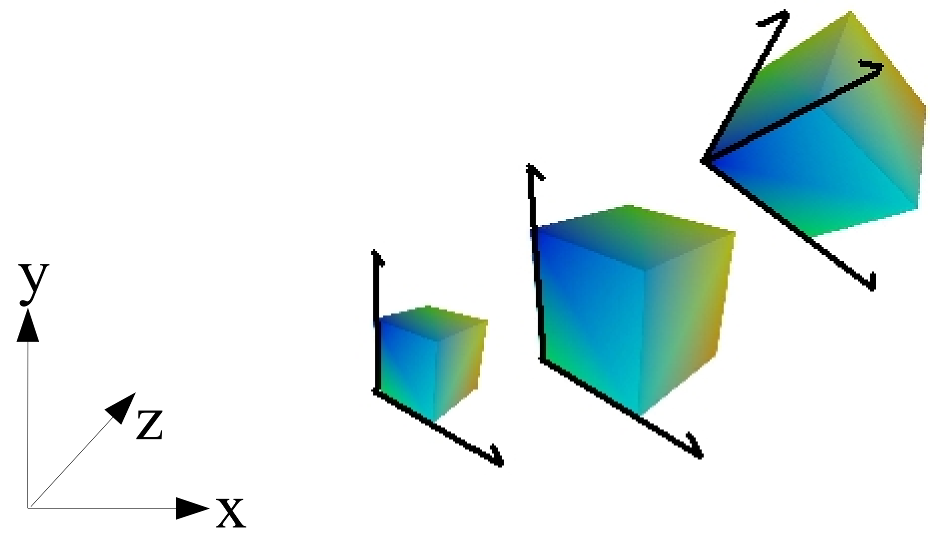
- **Today**
  - **co-ordinate systems**
  - **interpolation**
  - **search**



Global Co-ordinate Systems

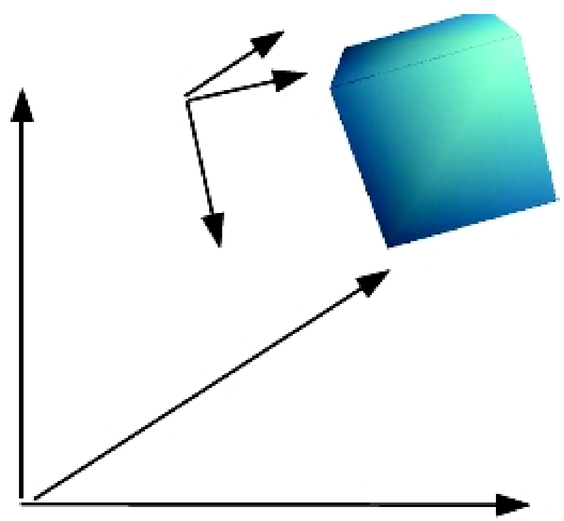
- **global coordinates ('World space')**
  - defined in Cartesian **3D space ($\mathbb{R}^3$)**
  - discrete point p is an triplet, *p = (x,y,z)*
  - space we render objects in
- **Local coordinates ('Object space')**
  - defined in the object' Cartesian **3D space ($\mathbb{R}^3$)**





Global Co-ordinates : Transformations 1

- **translation**
  - defined as translation vector, vector $\vec{t} = (t_x, t_y, t_z)'$
  - in x,y = position of dataset on image plane
  - in z = zoom (distance of dataset from image plane)
- **rotation**
  - **dataset viewpoint / view angle**
  - rotation around x, y and z axes
- **scaling**
  - in each axis direction $s = (s_x, s_y, s_z)'$



Global Co-ordinates : Transformations 2

- **Rotation matrix *R*** = $R_x(\theta_x)R_y(\theta_Y)R_z(\theta_Z)$
  - arbitrary angle order for x, y, z axis
  - generally use consistent system
  - Rotation parameters : $\{\theta_x , \theta_Y, \theta_Z\}$

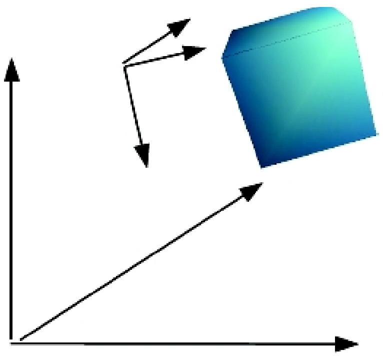
$$\mathrm{R}_x(\theta_x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta_x) & -sin(\theta_x) \\ 0 & sin(\theta_x) & cos(\theta_x) \end{bmatrix}$$

$$\mathrm{R}_y(\theta_y) = \begin{bmatrix} cos(\theta_y) & 0 & -sin(\theta_y) \\ 0 & 1 & 0 \\ sin(\theta_y) & 0 & cos(\theta_y) \end{bmatrix}$$

$$\mathrm{R}_z(\theta_z) = \begin{bmatrix} cos(\theta_z) & -sin(\theta_z) & 0 \\ sin(\theta_z) & cos(\theta_z) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

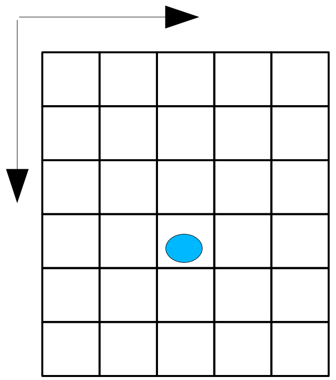
- **3D transformation :** 9 degrees of freedom
  - rotation (3); translation (3); scale (3)

Local Co-ordinate Systems

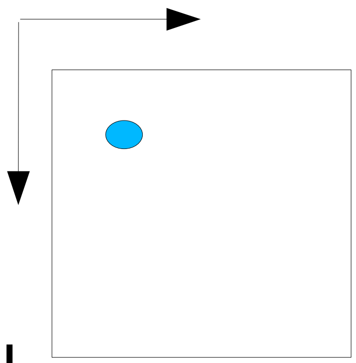


- Local **co-ordinate system of the dataset**
- Two main components
  - **topological co-ordinates** – the id of the cell
    - dependant on dataset structure
    - *Q: which cell within the dataset?*
  - **parametric co-ordinates** – location within the cell
    - internal to cell only
    - *Q: which location within the cell?*

Local Co-ordinate Systems

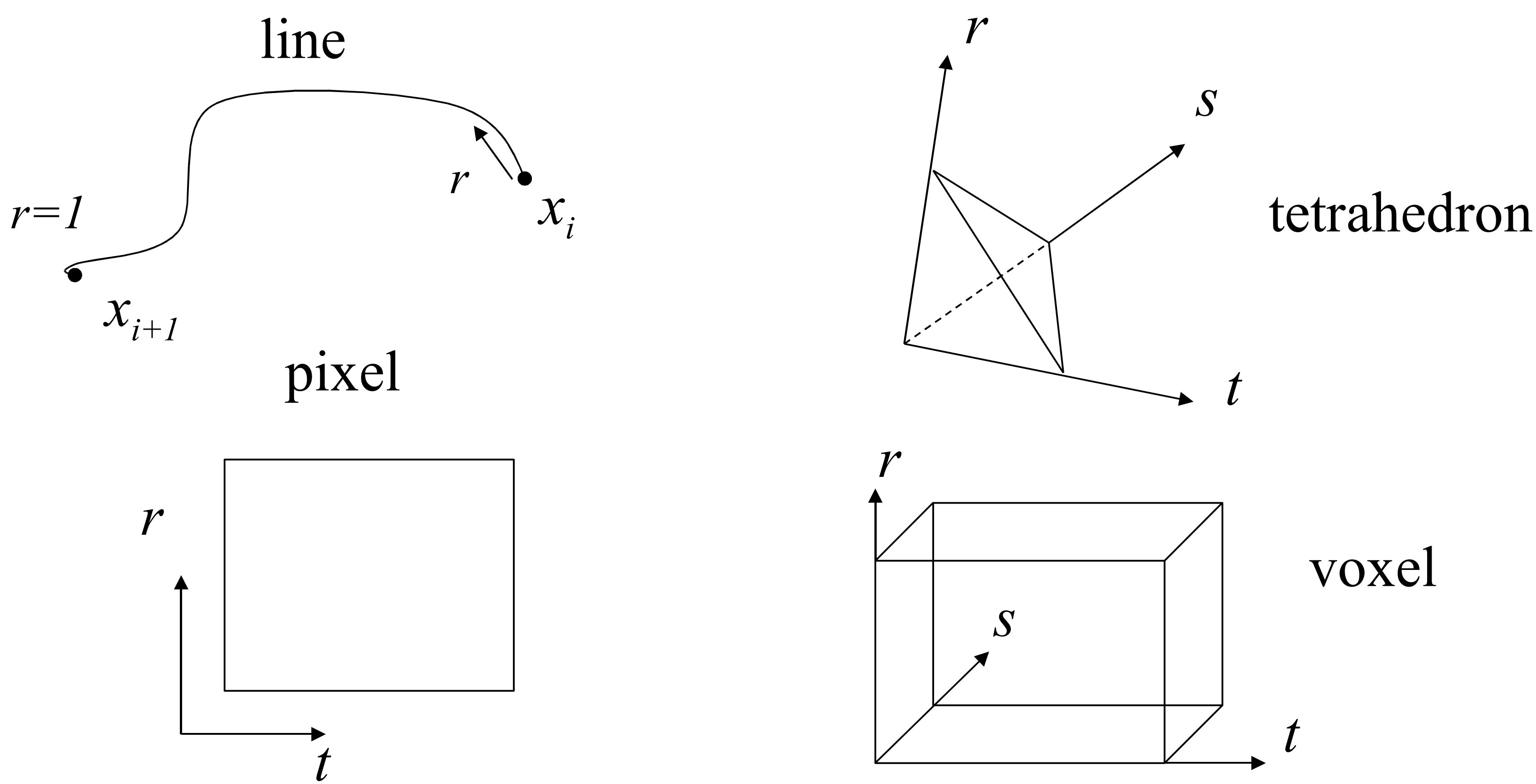


- **Topological** co-ordinates
  - identifies which **cell in the dataset**
    - i.e. within the **structure** of the dataset



- **Parametric** co-ordinates
  - location within the cell, within the dataset
  - **parametric co-ordinates** based on the topological dimension of the cell (e.g. 2D)

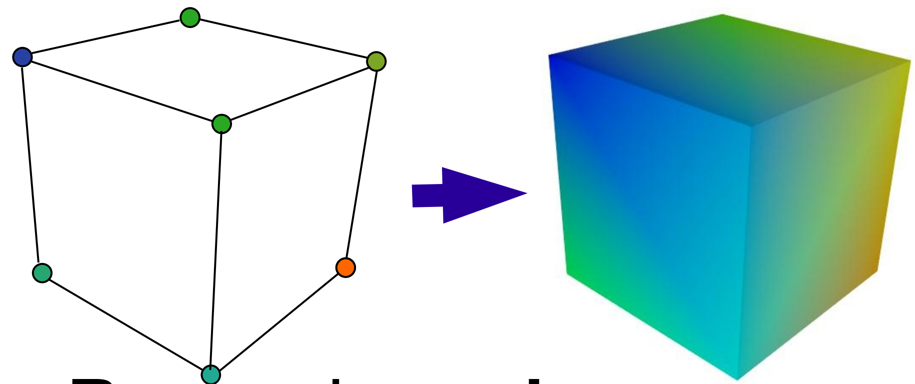
Parametric Co-ordinates

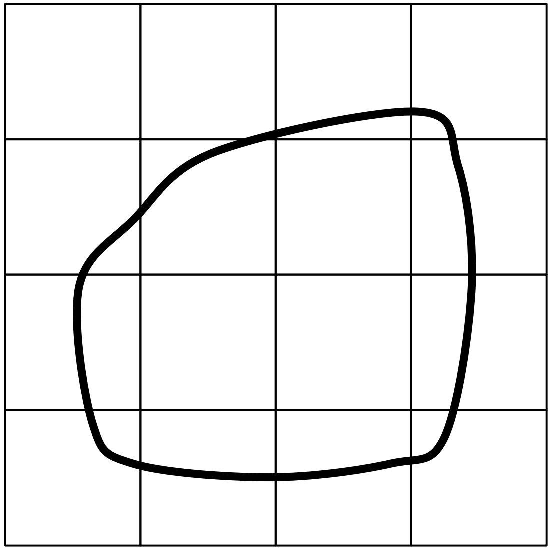


- Parameters r, t & s all in normalized range {0→1)
  - **topological dimension N requires N parameters**

Parametric Co-ordinates - Why?

- *Why do we need parametric co-ordinates?*
- *Why do we need to specify position inside a cell?*





- Remember : **data is discrete**

  ⇒ in visualization we require **interpolation** through cells

Interpolation

- Two types :
  - **Attribute interpolation** – derive the attribute value for a position defined in parametric cell space
  - **Geometric interpolation** – derive the global coordinates for a position in parametric cell space
    - local to global co-ordinate transformation

Interpolation : general form

- Performed in **local coordinate system**
  - internal to cell
  - **a weighted average of the point values from the cell**
  - weights in **inverse proportion to distance** from the points
    - nearer points more influence
    - further points less influence

$$d = \sum_{i=0}^{n-1} W_{i.} d_i$$

d is the **attribute value** at the position
d_i is the **attribute value** of the i^{th} point

$$\left(p = \sum_{i=0}^{n-1} W_{i.} p_i \right)$$

p is position value in **global coordinates**
p_i is the **global position** of the i^{th} point

n cell points
W_i is the weight for the i^{th} point

Interpolation weights

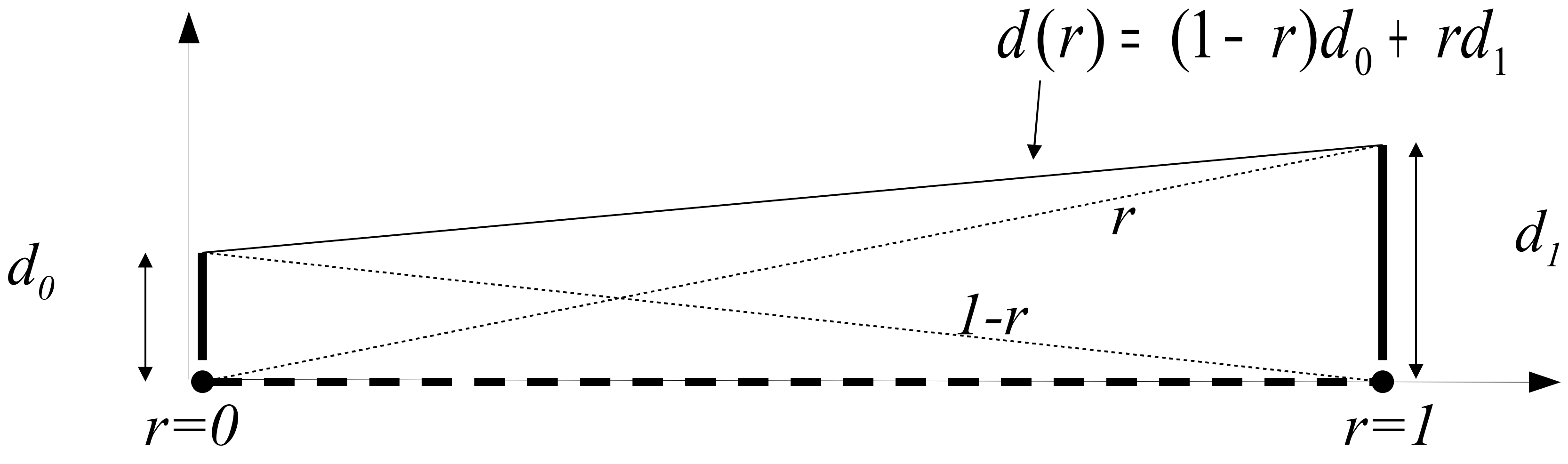
- **Requirements**
  - W_i =1 & W_j=0 when p = p_i and i != j
    - *when position is p_i all other weights must be 0 so that $d=d_i$*
  - interpolated value is no less than minimum $d_i = d_{min}$ and no greater than maximum $d_i = d_{max}$
    - i.e. $\mathbf{d_{min} \leq d_i \leq d_{max}}$
- **Weights bounded as follows:**

$$\sum W_i = 1, \ 0 \leq W_i \leq 1$$

Interpolation example : 1D line

- Data values d_0 and d_1
  - **line parametrised** by r from position of d_0 to d_1
  - **interpolate intermediate value** of d on 1D line
  - influence of d_1 increases as r→1 and d_0 decreases (& vice versa)

$$W_0 = (1-r)\ ,\ W_1 = r$$

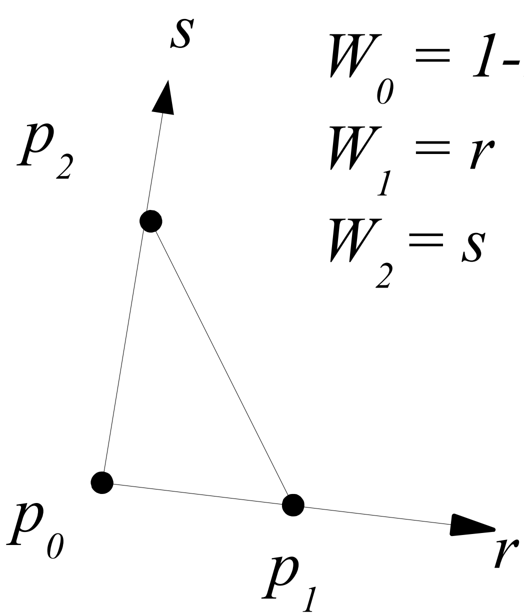


Interpolation in 2D/3D

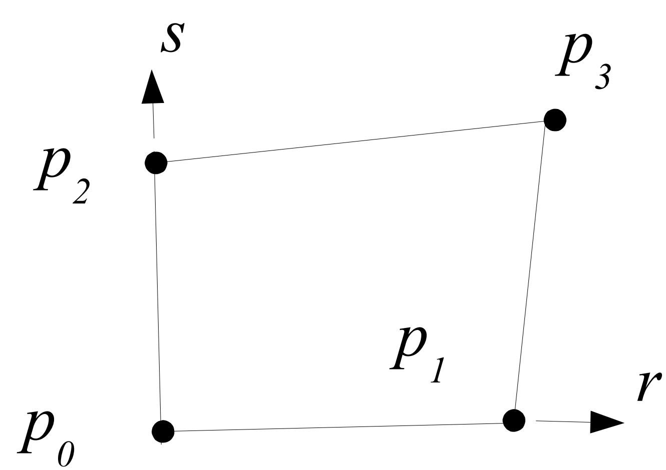
- 2D and 3D shapes have similar interpolation weights
  - N points, N weights
  - normalised to

$$\sum W_i = 1, \; 0 \le W_i \le 1$$

- 2D:

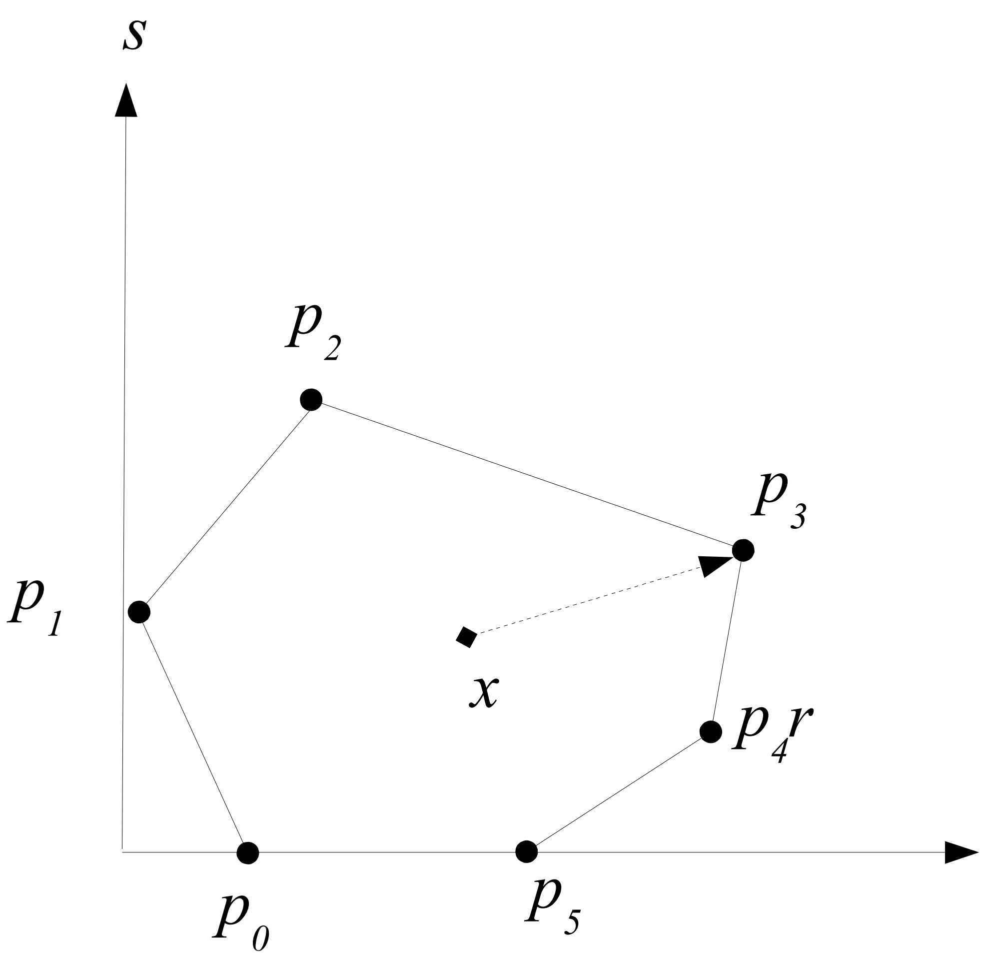


$$W_0 = 1-r-s$$
$$W_1 = r$$
$$W_2 = s$$



$$W_0 = (1-r)(1-s)$$
$$W_1 = r(1-s)$$
$$W_2 = (1-r)s$$
$$W_3 = rs$$

Interpolation : general polygon



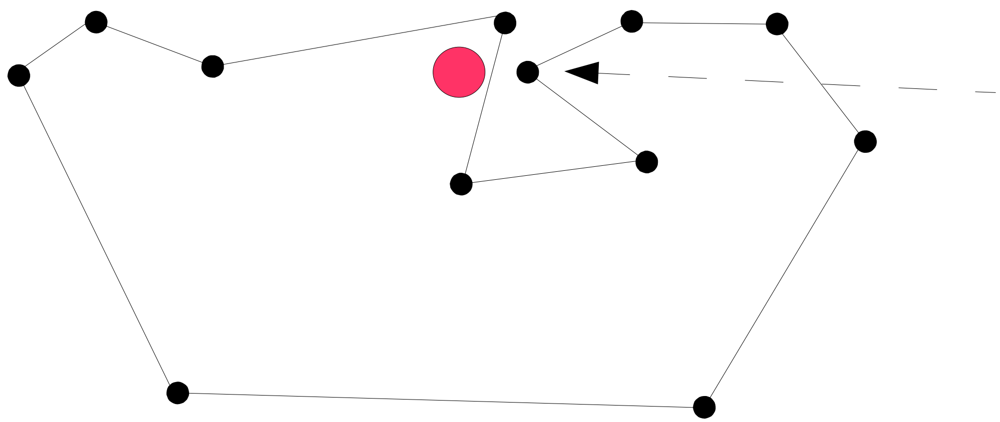
$$W_i = \frac{\left(\frac{1}{c_i}\right)^2}{\sum \left(\frac{1}{c_i}\right)^2}$$

$$c_i = |p_i - x|$$

Weighting functions are inversely proportional to distance squared.

c_i = distance from polygon point p_i

Interpolation : problem polygon



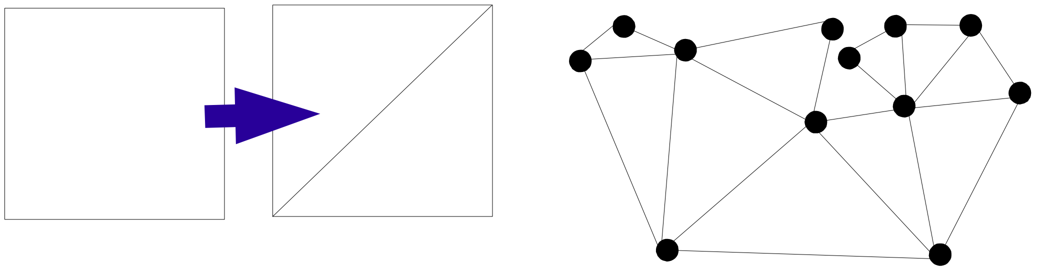
Problem: This vertex has too strong an influence in the red region

Solution: Split into simpler shapes

- **ill-formed polygon**
  - poor interpolation results based on point distance due to narrow concavity

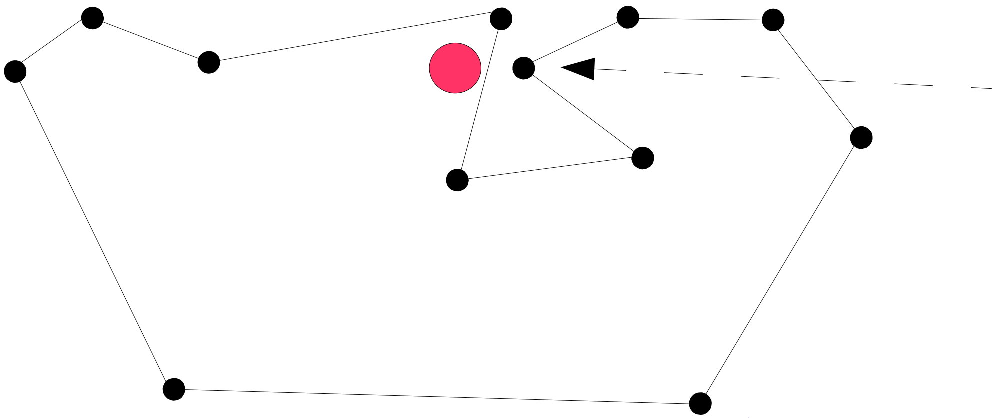
Simplices

- Shapes (in $\mathbb{R}^N$) can be decomposed into simpler shapes - **simplices**
  - 2D – triangles
  - 3D – tetrahedra
  - *ND – convex region of N+1 independent points*

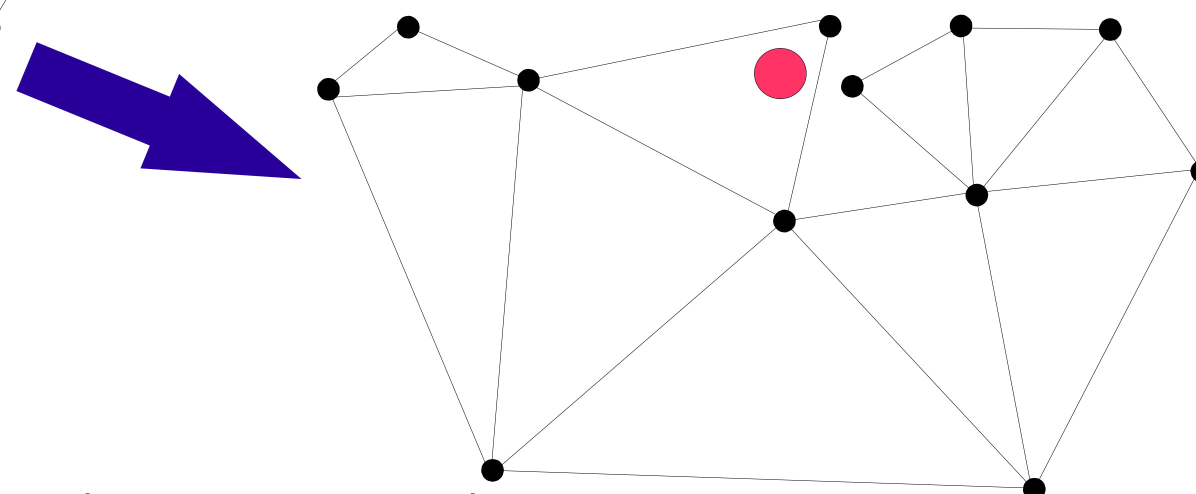


- **Principle :** implement visualisation algorithms for simplices
  - all other types can also be handled too
    - by simple reduction to simplices

Interpolation : via simplices



Problem: This vertex has too strong an influence in the red region

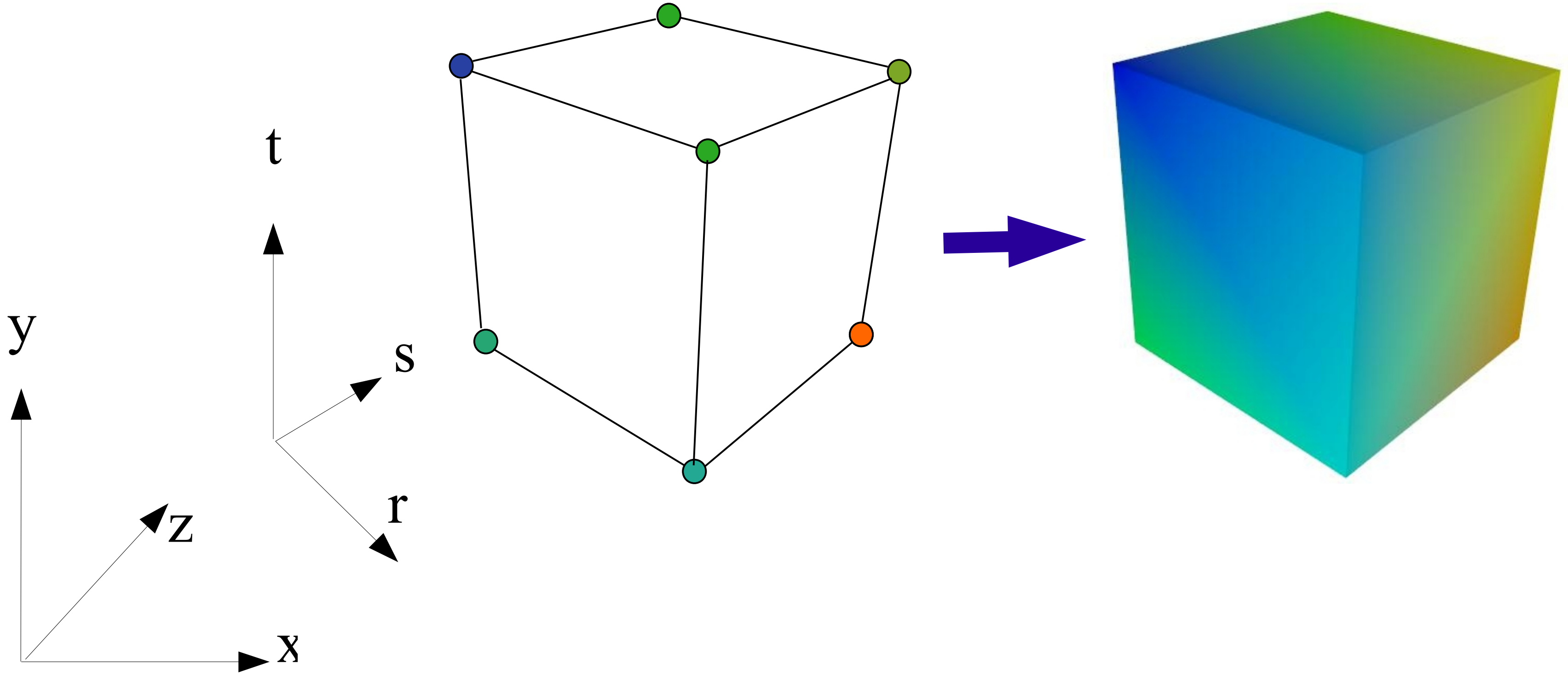


Solution : reduce to simplices (in 2D triangles)

- problem point no longer has interpolation influence in red region
- **change in topology** (can be used for other Vis. tech. too)

Interpolation : general principle

- **Interpolate** the value **attribute data over a cell** by **weighted sum** of **attribute data at discrete cell points**



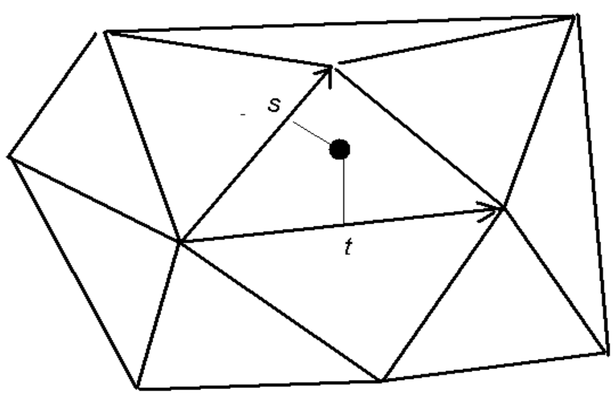
Co-ordinate Transformations 1

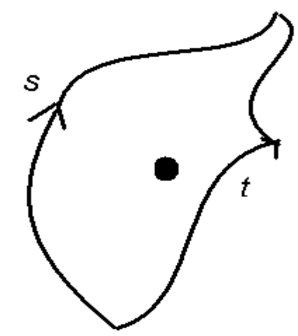
- **Geometric interpolation**
  - **local to global** co-ordinate transformation **via interpolation**
  - **Sometimes, we only know the local location of the mobile and the global location of the points**
    - **Mobile network : the distance from the stations and the location of the stations**
  - **Method:**
    - *sum the interpolation weight multiplied by the global positions of the cell points*
      - p = position of point in global coordinates
      - P_i = position of vertex
      - W_i is weighting function for vertex i at position p

$$p = \sum_{i=0}^{n-1} w_i p_i$$

Co-ordinate Transformations 2

- Inverse **global to local** interpolation
  - *Q: what is the interpolated value at p = (x,y,z) ?*
  - *A:* computationally expensive



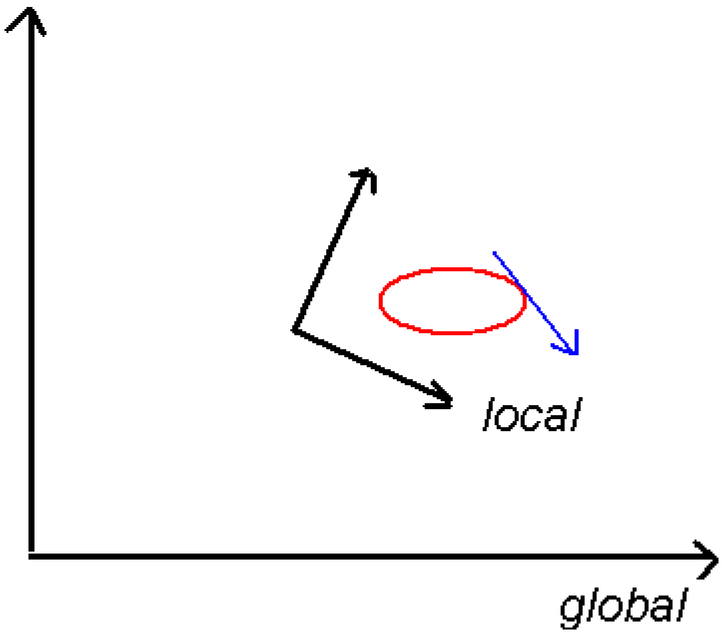


  - 2 stages
    - **search** : find cell containing globally specified point p
    - **parametric solution** : to resolve point position p inside cell
      - simple for linear weighting functions – exact solution
      - for non-linear – require iterative solution *(expensive)*

Computing Derivatives 1

- **Why ? :** interested in gradient of attribute data
  - rate of change of attribute data at arbitrary location in cell
  - e.g. stresses and strains from displacements

    direction of greatest temperature gradient



S

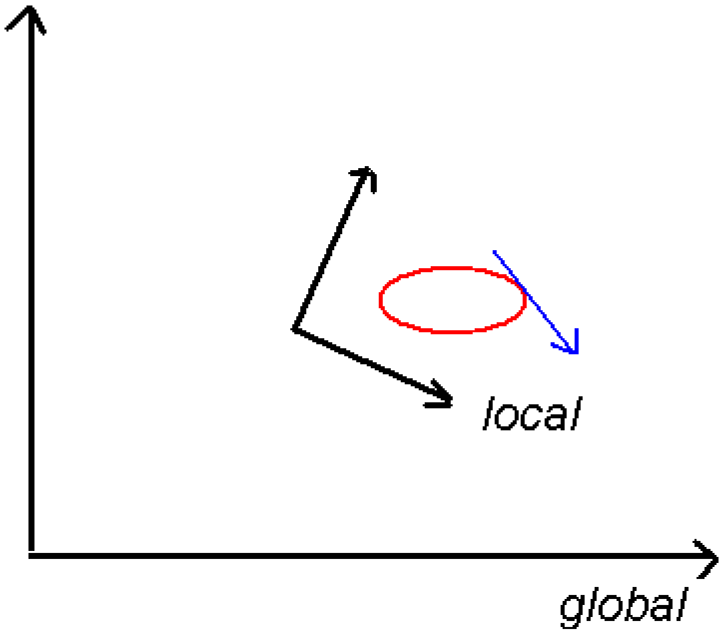
Computing Derivatives 1

- Differentiate interpolation functions in global coordinates

  1. Compute derivatives in local parametric coordinates
  2. Transform to global Cartesian space using chain rule

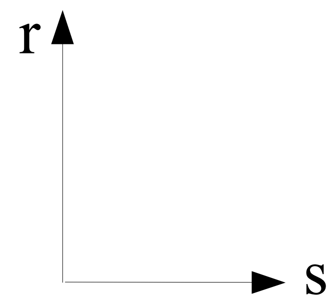
  2D:

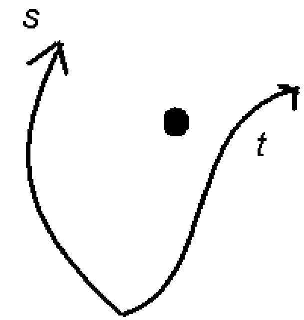
$$\begin{bmatrix} \frac{\delta D}{\delta r} \\ \frac{\delta D}{\delta s} \end{bmatrix} = \begin{bmatrix} \frac{\delta x}{\delta r} & \frac{\delta y}{\delta s} \\ \frac{\delta x}{\delta s} & \frac{\delta y}{\delta s} \end{bmatrix} . \begin{bmatrix} \frac{\delta D}{\delta x} \\ \frac{\delta D}{\delta y} \end{bmatrix}$$



Computing Derivatives 2

- If local space is not Cartesian *(we are assuming it is!)* *dx/dr* will vary with position in the cell
  - must re-evaluate for each point



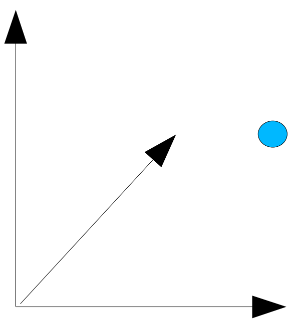


- Generalise to 3D for parameters (r,s,t)
  - 3x3 matrix J = **Jacobian**
    - **relates parametric (local) derivatives to global co-ordinate derivatives**
  - invert J to find global derivatives

$$\begin{bmatrix} \frac{\delta}{\delta r} \\ \frac{\delta}{\delta s} \\ \frac{\delta}{\delta t} \end{bmatrix} = \begin{bmatrix} \frac{\delta x}{\delta r} & \frac{\delta y}{\delta r} & \frac{\delta z}{\delta r} \\ \frac{\delta x}{\delta s} & \frac{\delta y}{\delta s} & \frac{\delta z}{\delta s} \\ \frac{\delta x}{\delta t} & \frac{\delta y}{\delta t} & \frac{\delta z}{\delta t} \end{bmatrix} . \begin{bmatrix} \frac{\delta}{\delta x} \\ \frac{\delta}{\delta y} \\ \frac{\delta}{\delta z} \end{bmatrix} = J. \begin{bmatrix} \frac{\delta}{\delta x} \\ \frac{\delta}{\delta y} \\ \frac{\delta}{\delta z} \end{bmatrix}$$

Cell Search

- For a given dataset
  - *Q: which cell contains point p = (x,y,z) ?*
  - ***point location query***



  - **Naive approach** – exhaustive search *O(n)* for n cells

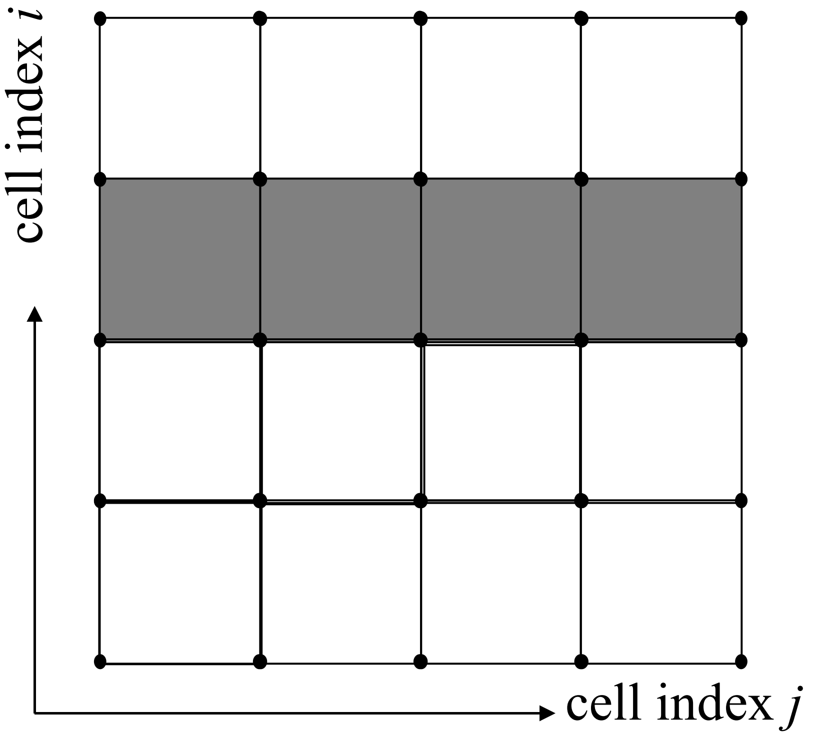
⇒**improvement possible**

Cell Search - Why?

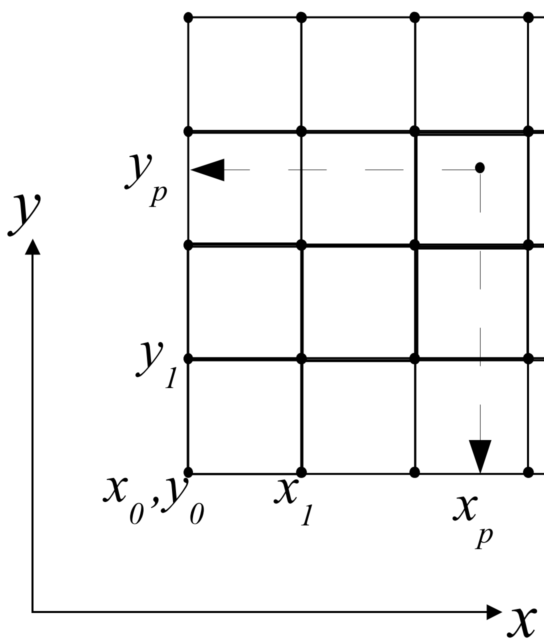
- **interaction** : picking a cell on the screen
- **nearest-neighbours :** need values of neighbouring cells/points
- 2D Cells – line intersection
  - perform line-plane intersection
    - find cell / perform parametric interpolation to get value/position in cell
- 3D Cells - line intersection
  - perform line-plane intersection with faces of cell
    - difficult with non-planar faces (e.g. hexahedron)

Cell Search – structured data

- **Regular Topology**
  - topological co-ordinates are sequential
- Can specify region of interest using range of indices
- Operations much simplified



Cell search – structured points



$$i = floor((x_p - x_0)/(x_1 - x_0))$$
$$j = floor((y_p - y_0)/(y_1 - y_0))$$

$$r = frac((x_p - x_0)/(x_1 - x_0))$$
$$s = frac((y_p - y_0)/(y_1 - y_0))$$

- **Searching**
  - easy to find cell containing point
  - *floor* gives topological coordinate, *fraction* gives parametric coordinates.
  - *constant time* O(1)

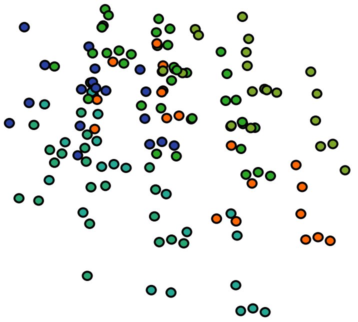
Unstructured Data : Examples

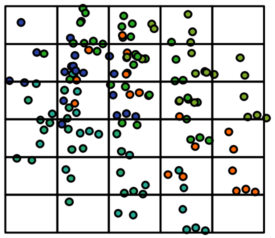
- Say you are given a list of cities and their positions
- You might need to know which city is close to which
- How do you search for such cities? How much is the cost?

- Think about a dictionary in random order
  - Requires O(n) time to search for each word

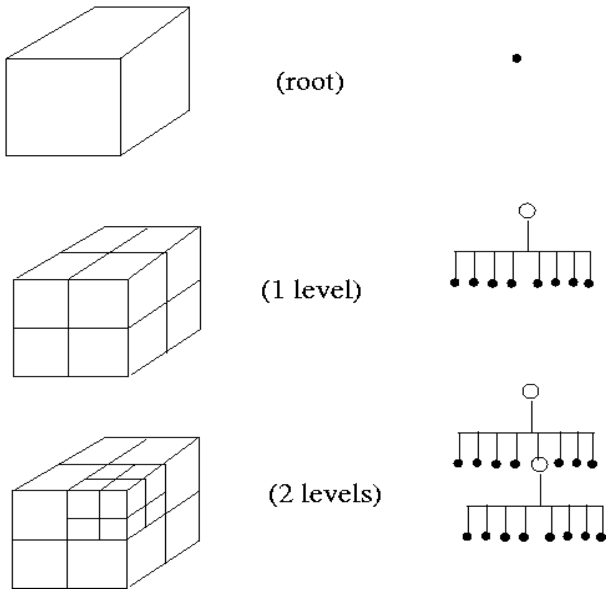
Cell search – unstructured points

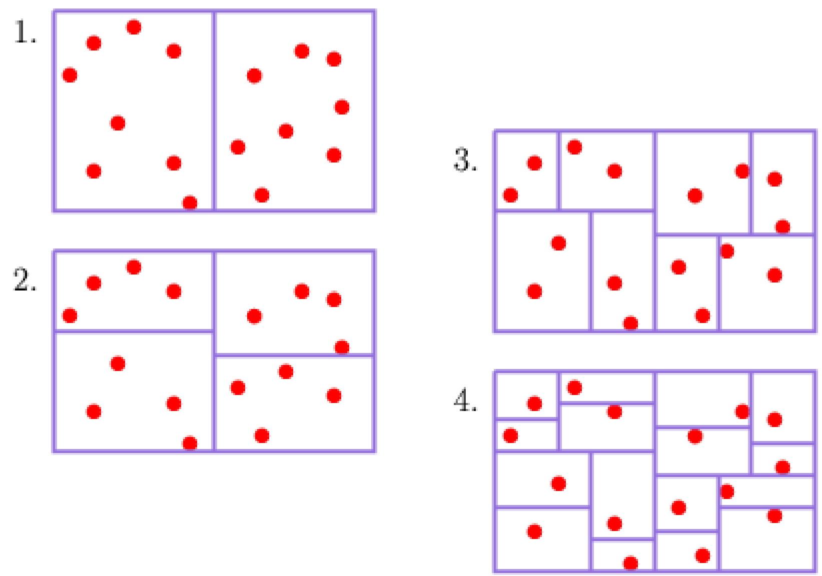
- Difficult to perform fast search on unstructured data
  - **solution:** introduce artificial structure
  - i.e. search structure / **introduce new topology**





Cell search – unstructured points





- Spatial searching data structures
  - **principle** : put points in ordered bins then find correct bin
    - > using hash functions
  - octrees : tree of recursively defined bounding boxes
  - kd-trees : extension of binary tree principle to k dimensions
    - construction *O(n log n); search query O(log n)*

Summary

- **Co-ordinate Systems**
  - global co-ordinates / parametric cell co-ordinates
- **Interpolation**
  - in parametric cell space
- **Derivatives**
  - in parametric cell space
- **Search**
  - point location queries in structured / unstructured data