CS 519: Scientific Visualization

Working with Data: Representation and Interpolation

Eric Shaffer

Today…

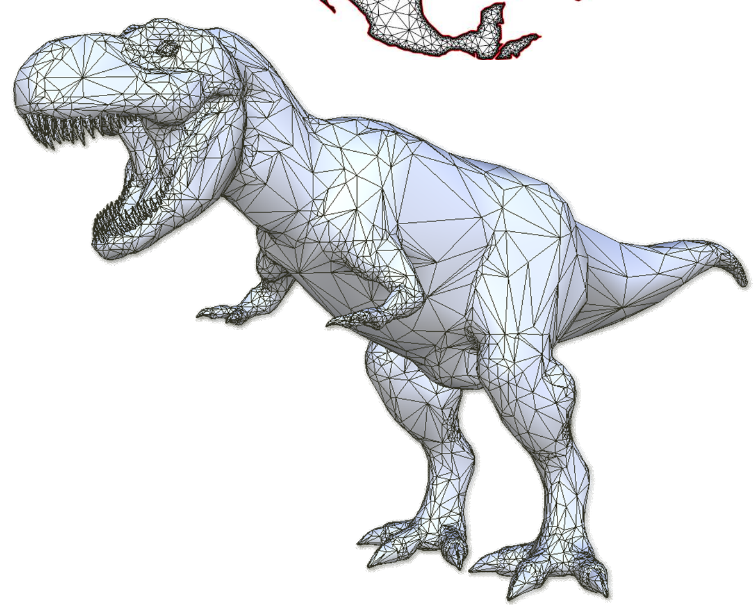
- Visualization of the day
  - https://www.youtube.com/watch?v=EgumU0Ns1YI
- Data representation
- (Linear) Interpolation

Things for later

- These are mentioned in Chapter 3…but we'll get to them later
  - Color
  - Advanced spatial data structures
  - Tensors
  - Calculating derivatives

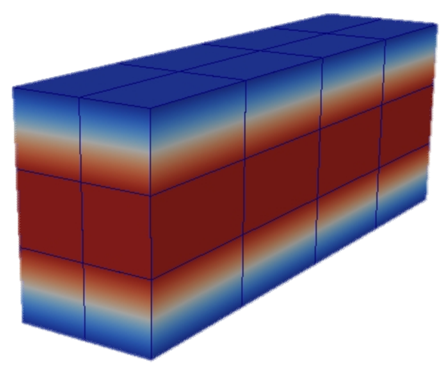
Data Representation

- Disclaimer: add “typically” before every sentence
- “Typically” Scientific Visualization data is discretely sampled
  - The original function is continuous
- The Domain over which the data is sampled is discretized
  - In 2D using a polygonal mesh of cells
    - This includes data sampled on a 2D surface embedded in 3-space
  - In 3D using a polyhedral mesh of cells
- Data values are either vertex or cell-centered



Interpolation

- Interpolation is a mathematical process for filling in missing data
- An interpolating function approximates some sampled function
  - Approximation matches original function value at the sample points
- Useful in visualization
  - We usually don't have original function values at each pixel.



How Can We Organize Data Sampled in Euclidean Space ?

- Chop the space up (discretize it) into **cells**
- A structure of cells is called a **grid** or **mesh**
- Lots of cell types are possible
  - 0D point
  - 1D line
  - 2D triangle, quad, rectangle
  - 3D tetrahedron, parallelepiped, box, pyramid, prism,

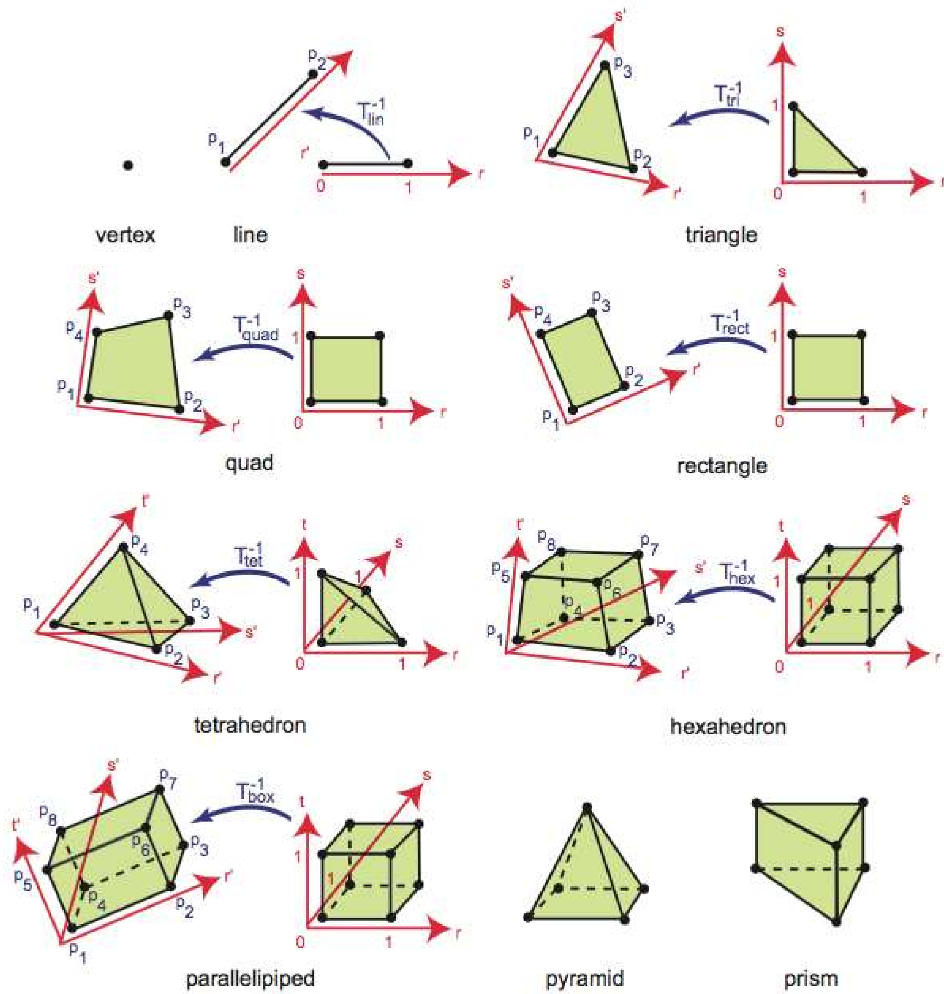


Figure 3.5. Cell types in world and reference coordinate systems.

Grids (or Meshes)

Cells

- provide interpolation over a small, simple-shaped spatial region

Grids (or meshes)

- partition our complex data domain D into cells
- allow applying per-cell interpolation (as described so far)

Given a domain D…

A grid G = {ci} is a set of cells such that

$c_i \cap c_j = \emptyset, \forall i \neq j$ no two cells overlap

$\bigcup_i c_i = D$ the cells cover all our domain

The dimension of the domain D constrains which cell types we can use

Uniform Grids

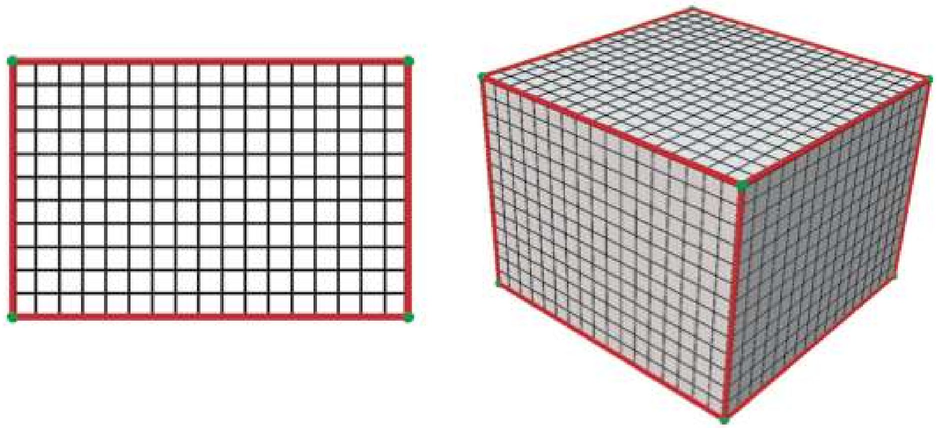
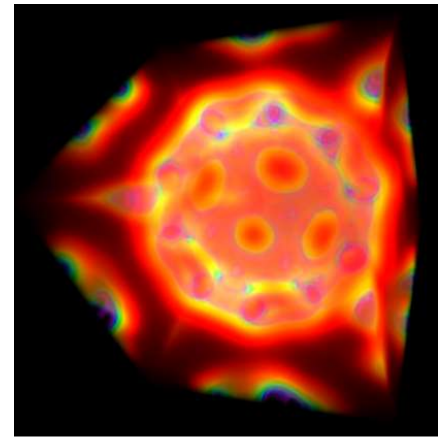


Figure 3.7. Uniform grids. 2D rectangular domain (left) and 3D box domain (right).

image



volume

- all cells have identical size and type (typically, square or cubic)

Storage requirements for the structure (not the data)

- m integers for the #cells along each of the m dimensions of D (e.g. m=2 or 3)
- two corner points

Rectilinear Grids

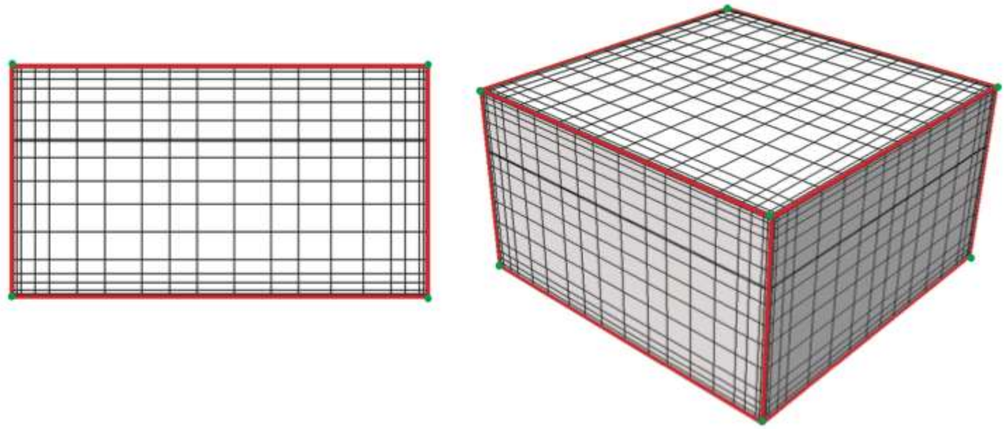


Figure 3.8. Rectilinear grids. 2D rectangular domain (left) and 3D box domain (right).

- all cells have same type
- cells can have different sizes but share them along axes
- Book says → "cannot model non-axis-aligned domains". Is that true?

Storage requirements for the structure

$\sum_{i=1}^{m} d_i$ floats (coordinates of vertices along each of the m axes of D)

And what else?

Structured Grids

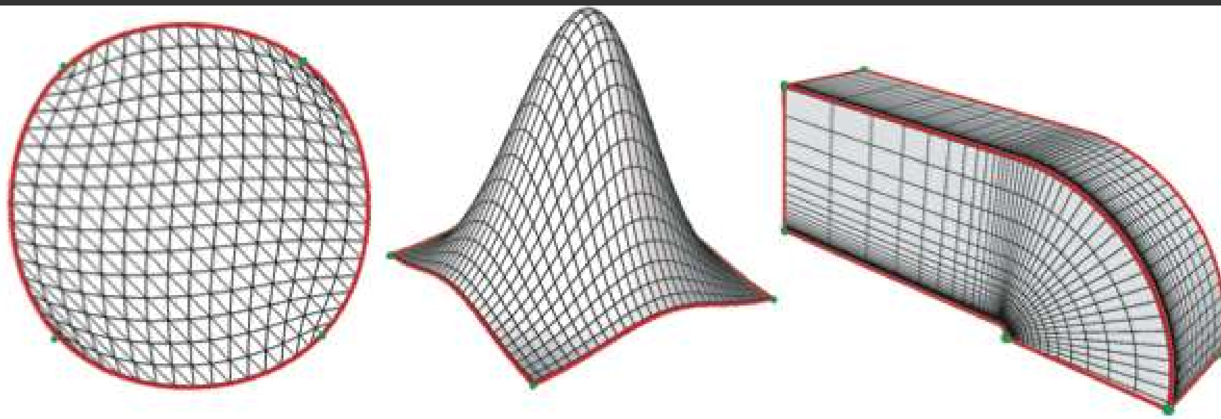


Figure 3.9. Structured grids. Circular domain (left), curved surface (middle), and 3D volume (right). Structured grid edges and corners are drawn in red and green, respectively.

- all cells have same type
- cell vertex coordinates are freely (explicitly) specifiable…
- …as long as cells assemble in a matrix-like structure
- can approximate more complex shapes than rectilinear/uniform grids

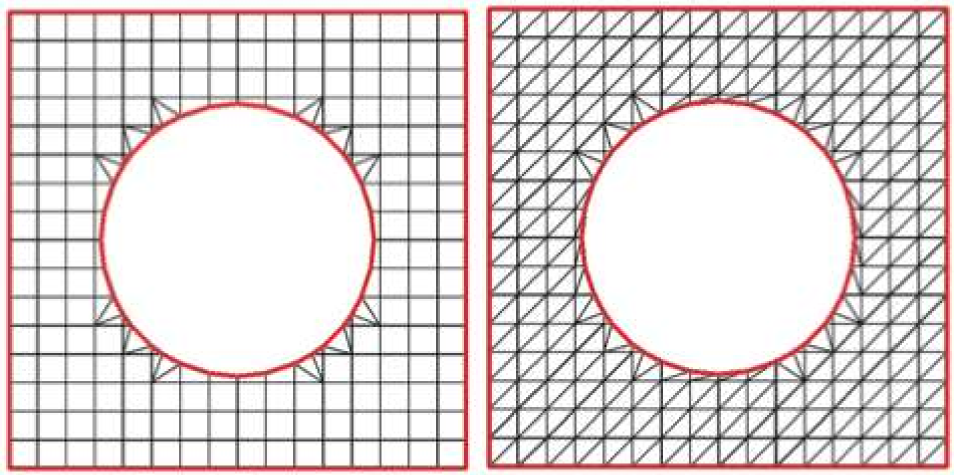
Storage requirements

$\prod_{i=1}^{m} d_i$ floats (coordinates of all vertices)

And what else?

Unstructured Grids

Consider the domain D: a square with a hole in the middle

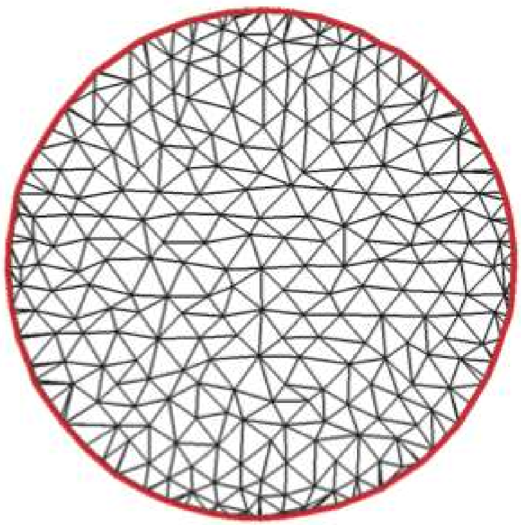


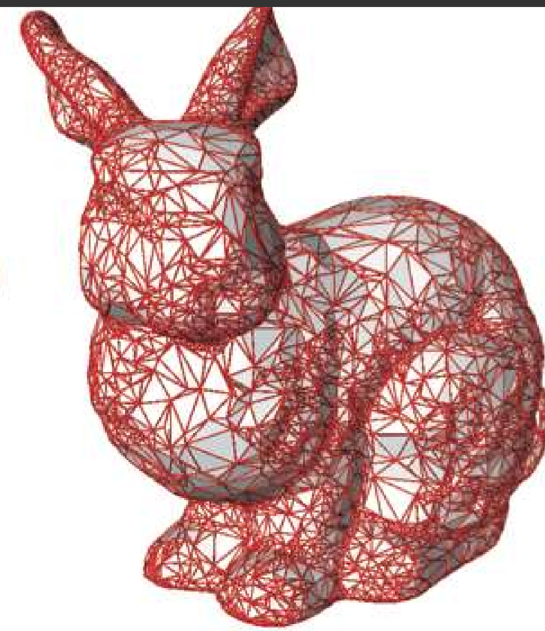
According to book: We cannot cover such a domain with a structured grid (why?)

- it's not of genus 0, so cannot be covered with a matrix-like distribution of cells
- Or could it?
- What about genus 2?
- BTW, genus means how many holes there are in the domain.

For more generality, we need unstructured grids

Unstructured Grids





- different cell types can be mixed
- both vertex coordinates and cell themselves are freely (explicitly) specifiable
- implementation
  vertex set $V = \{v_i\}$
  cell set $C = \{c_i = (indices\ of\ vertices\ in\ V)\}$
- most flexible, but most complex/expensive grid type

Storage requirements

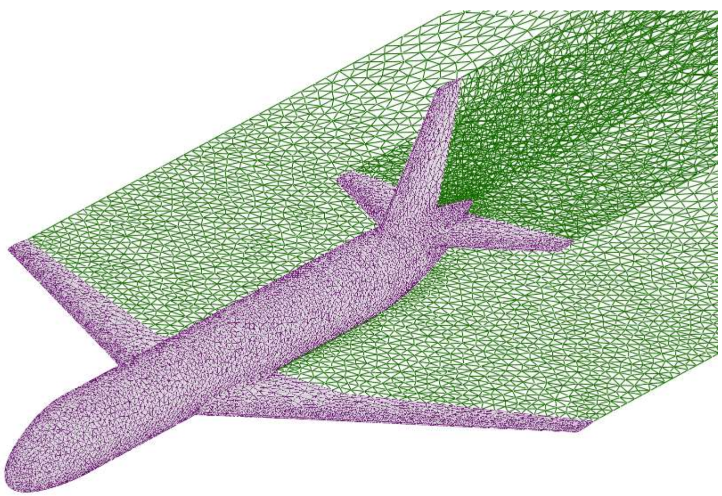
$m\|V\| + s\|C\|$ for a m-dimensional grid with cells having s vertices each
What operation is hard to do with just this information?

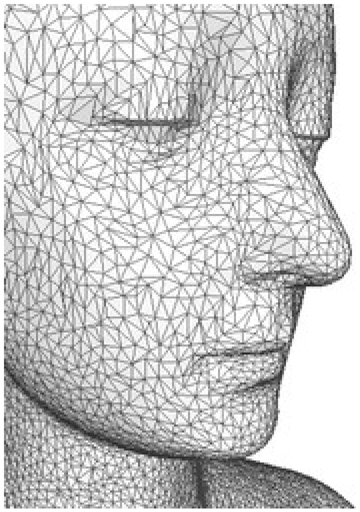
Example: Unstructured Surface Mesh

This Wavefront/OBJ file describes a single triangle.

Could add more vertices and faces for a bigger mesh

```
# Simple Wavefront file
v 0.0 0.0 0.0
v 0.0 1.0 0.0
v 1.0 0.0 0.0
f 1 2 3
```





Data Attributes

$f: \mathbf{R}^m \rightarrow \mathbf{R}^n$

- $n=0$ no attributes (we model a shape only e.g. a surface)
- $n=1$ scalars (e.g. temperature, pressure, curvature, density)
- $n=2$ 2D vectors
- $n=3$ 3D vectors (e.g. velocity, gradients, normals, colors)
- $n=6$ symmetric tensors (e.g. diffusion, stress/strain)
- $n=9$ asymmetric general tensors (not very common)

Remarks

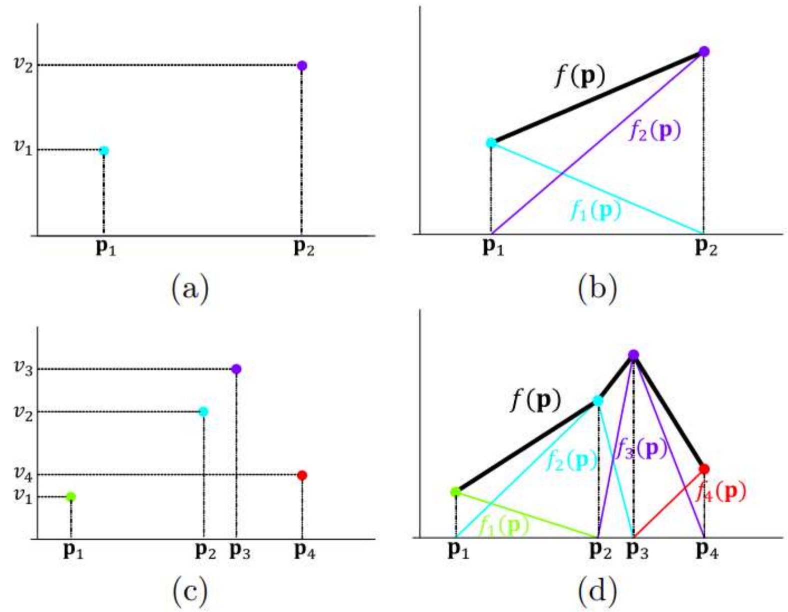
- an attribute is usually specified for **all** sample points in a dataset
- each attribute is interpolated separately
- different visualization methods for each n

A Simple Approach to Interpolation

- Suppose we have sampled values at two points P1 and P2 and want to guess at the function values on the line between the points.
- We can perform linear interpolation or LERP

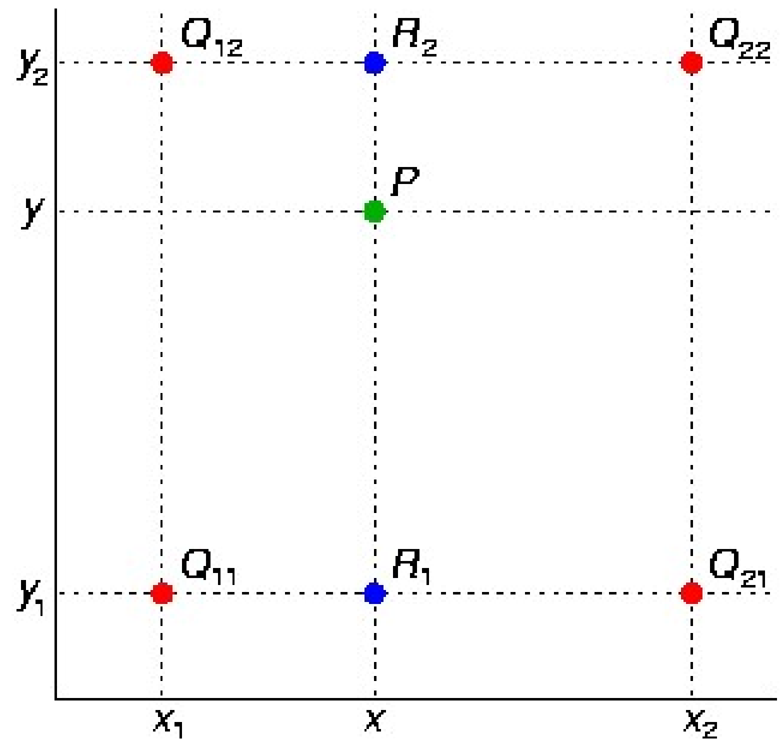
$$f(\mathbf{p}) = \frac{\mathbf{p}_2 - \mathbf{p}}{\mathbf{p}_2 - \mathbf{p}_1} v_1 + \frac{\mathbf{p} - \mathbf{p}_1}{\mathbf{p}_2 - \mathbf{p}_1} v_2$$

- When might this be a poor choice?
- Also, what dimension is this domain?



Bilinear Interpolation

- If we have a function defined on a 2D domain we need to do more
- You've already seen it …but repetition can be helpful
- Assume we know a function value at the four points
Q_{11} = (x1, y1), Q_{12} = (x1, y2),
Q_{21} = (x2, y1), Q_{22} = (x2, y2)
- We first do linear interpolation in the x-direction
- …and then in the y direction



Bilinear Interpolation

$$f(R_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21})$$

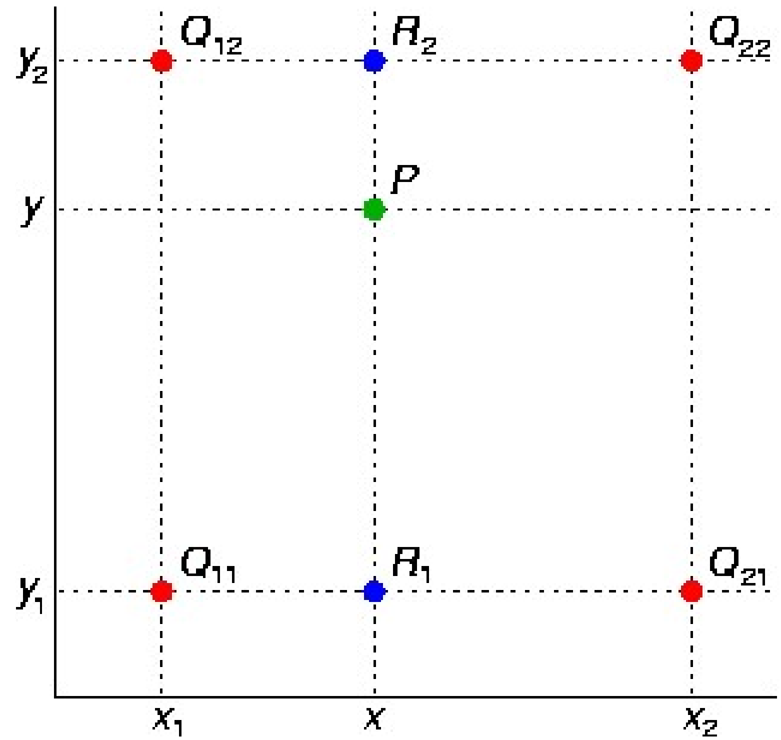
where $R_1 = (x, y_1)$,

$$f(R_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22})$$

where $R_2 = (x, y_2)$.

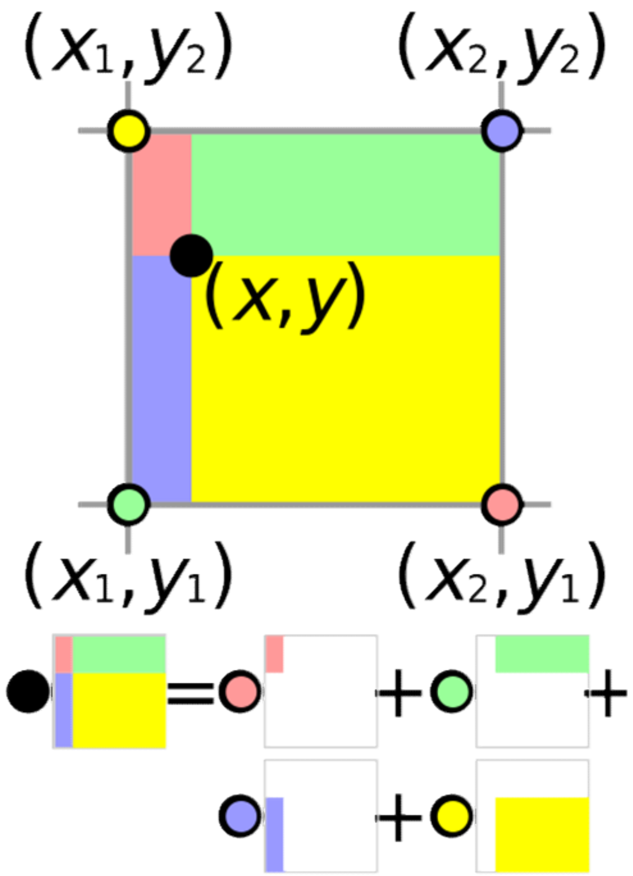
We proceed by interpolating in the *y*-direction.

$$f(P) \approx \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2).$$



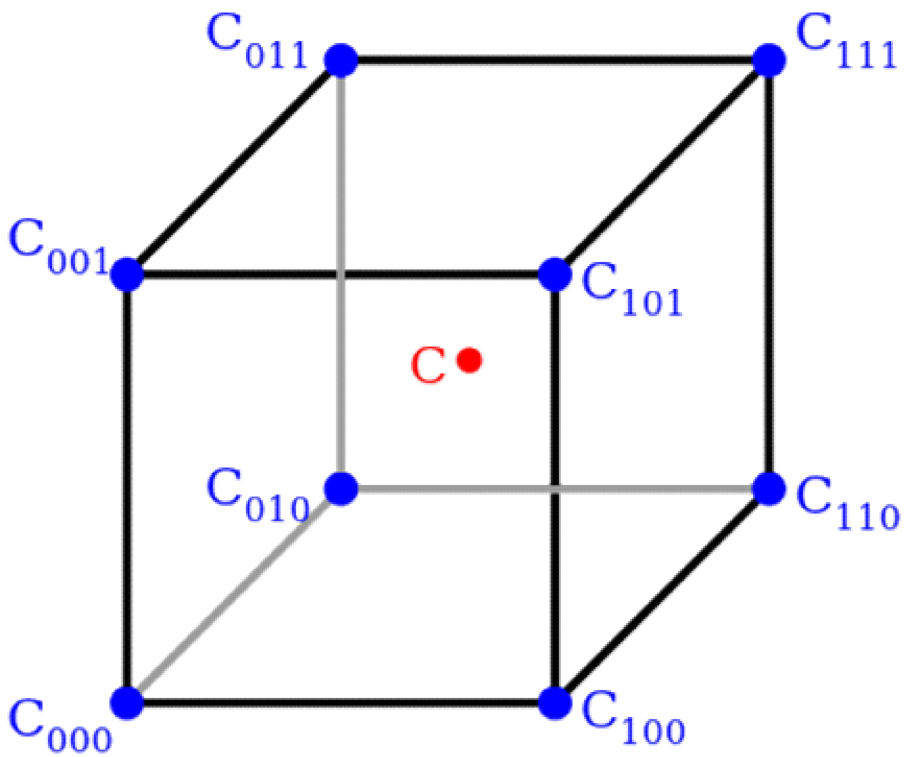
from Wikipedia

Bilinear Interpolation



What is the image telling us?

What about 3D?

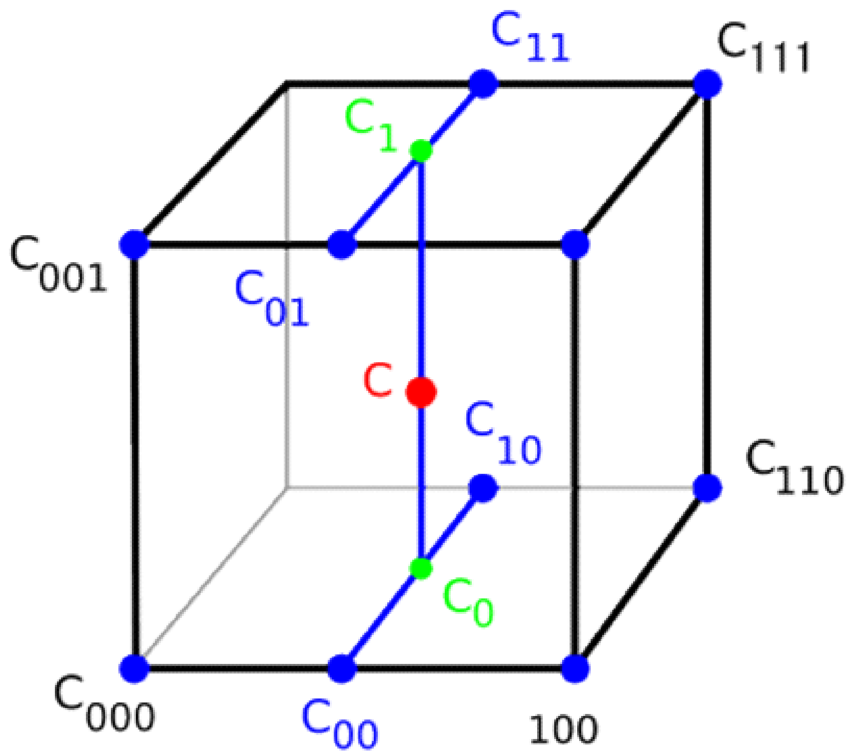


Trilinear Interpolation

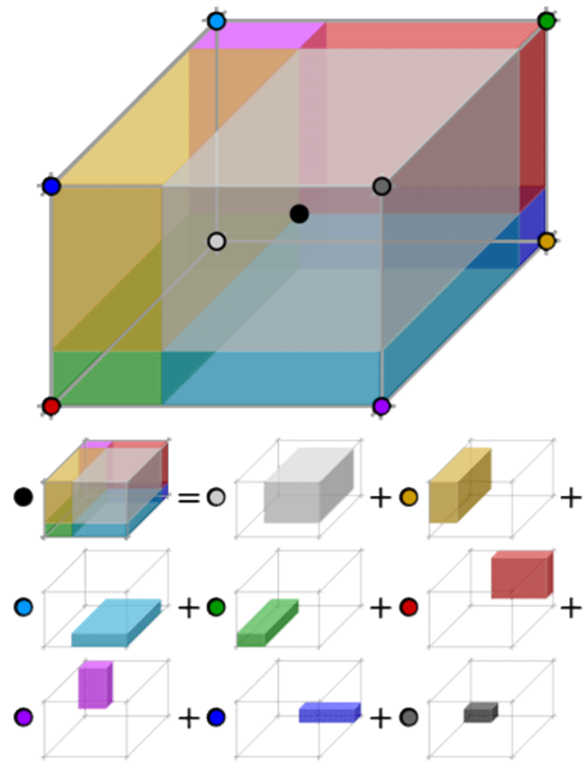
First interpolate in x to find c_{00}, c_{01}, c_{10}, and c_{11}

Then in y to find C_0 and C_1

And then in z to find C

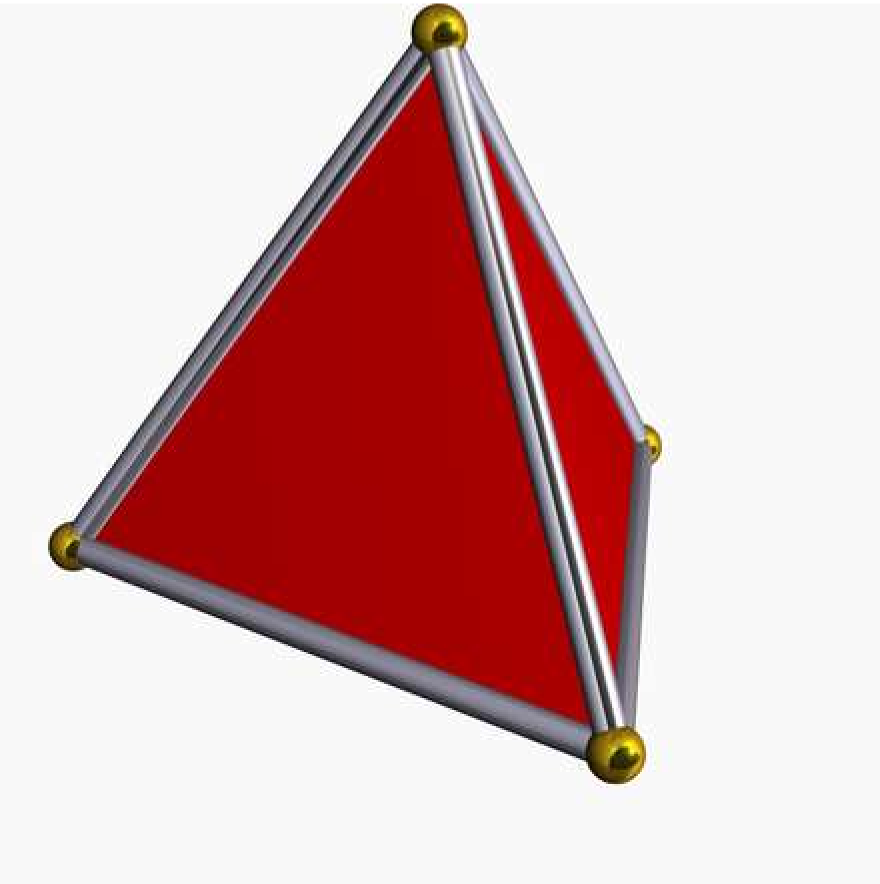


Trilinear Interpolation



Barycentric Coordinates

- What about non-quad-like things?
- A simplex is…
  - Convex hull of k+1 points in a k-dimensional space
  - Simplest convex “polygon” in a k-dimensional space
  - A 3-simplex is a….
- Barycentric coordinates provide a simple way to interpolate over simplices



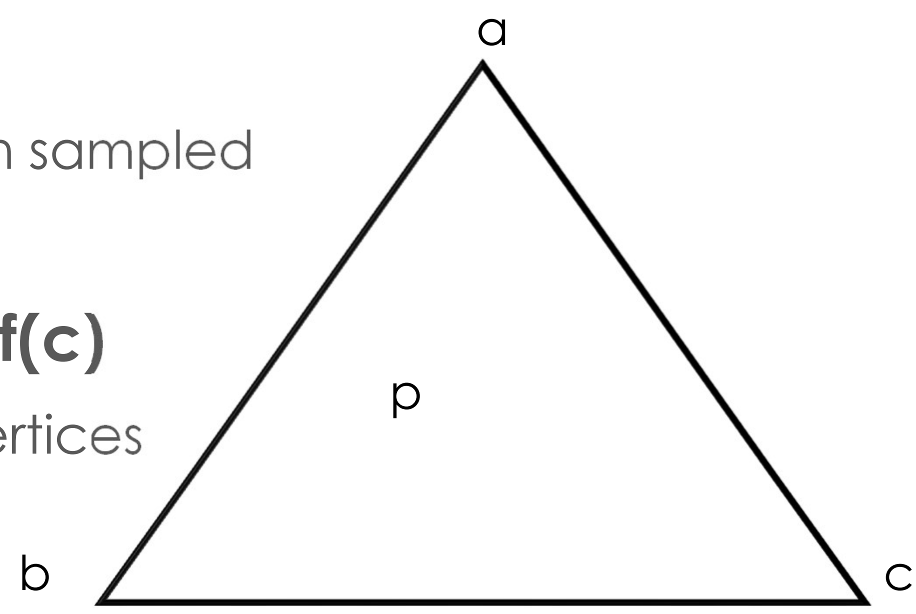
Barycentric Coordinates for Triangles

- Describe location of point in a triangle in relation to the vertices
- $p=(\lambda_1,\lambda_2,\lambda_3)$ where the following are true
  - $p=\lambda_1 a + \lambda_2 b + \lambda_3 c$
  - $\lambda_1+\lambda_2+\lambda_3= 1$
- To interpolate a function sampled

at the vertices we just do:

$$f(p)=\lambda_1 f(a) + \lambda_2 f(b) + \lambda_3 f(c)$$

- Does the order of the vertices need to specified?



Barycentric Coordinates for Triangles

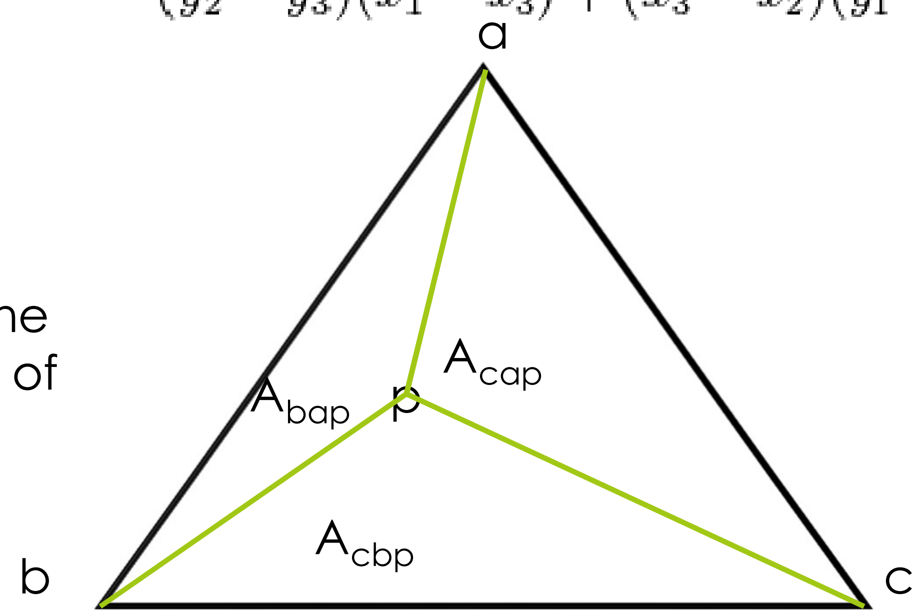
$$\lambda_1 = \frac{(y_2 - y_3)(x - x_3) + (x_3 - x_2)(y - y_3)}{\det(T)} = \frac{(y_2 - y_3)(x - x_3) + (x_3 - x_2)(y - y_3)}{(y_2 - y_3)(x_1 - x_3) + (x_3 - x_2)(y_1 - y_3)},$$

$$\lambda_2 = \frac{(y_3 - y_1)(x - x_3) + (x_1 - x_3)(y - y_3)}{\det(T)} = \frac{(y_3 - y_1)(x - x_3) + (x_1 - x_3)(y - y_3)}{(y_2 - y_3)(x_1 - x_3) + (x_3 - x_2)(y_1 - y_3)},$$

$$\lambda_3 = 1 - \lambda_1 - \lambda_2 .$$

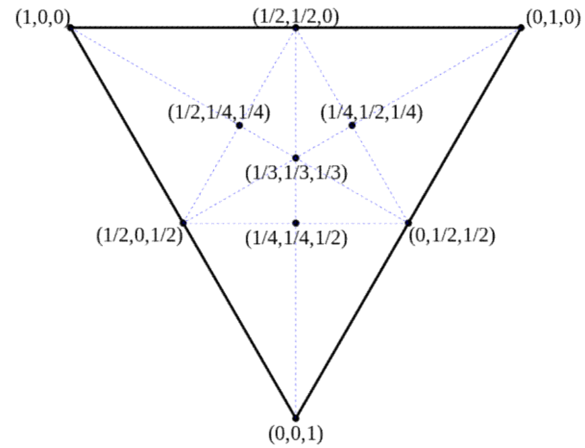
Coordinates are the signed area of the opposite subtriangle divided by area of the triangle

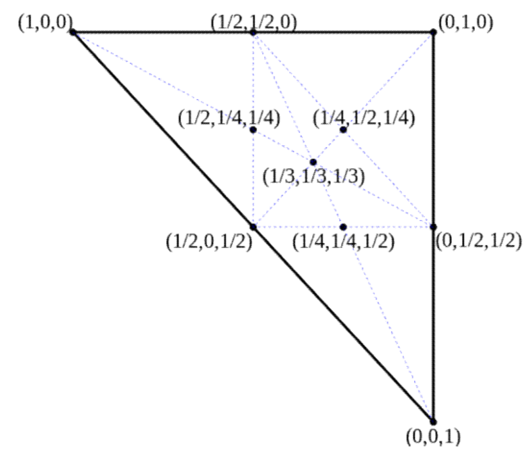
What about triangles in $\mathbf{R}^3$?



Barycentric Coordinates for Triangles

- Can barycentric coordinates be negative?
- What do you know about a point if it has a coordinate not in [0,1]?





Barycentric Coordinates for Tetrahedra

$$\begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{pmatrix} = \mathbf{T}^{-1}(\mathbf{r} - \mathbf{r}_4)$$

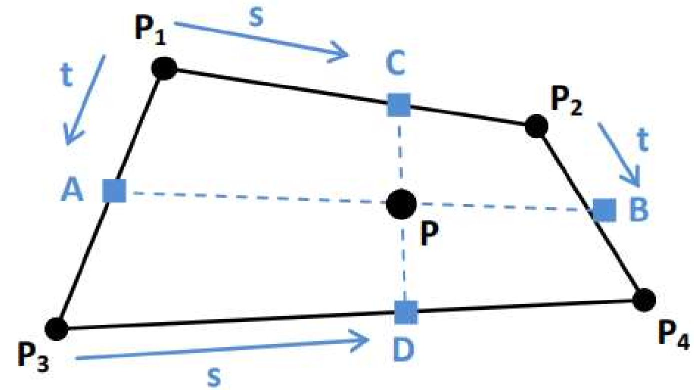
$$\mathbf{T} = \begin{pmatrix} x_1 - x_4 & x_2 - x_4 & x_3 - x_4 \\ y_1 - y_4 & y_2 - y_4 & y_3 - y_4 \\ z_1 - z_4 & z_2 - z_4 & z_3 - z_4 \end{pmatrix}$$

We can solve a linear system to find the coordinates.
Here, **r** is a point in $\mathbf{R}^3$ and $\mathbf{r}_4$ is the 4th corner of the tetrahedron.

How do we find λ_4?

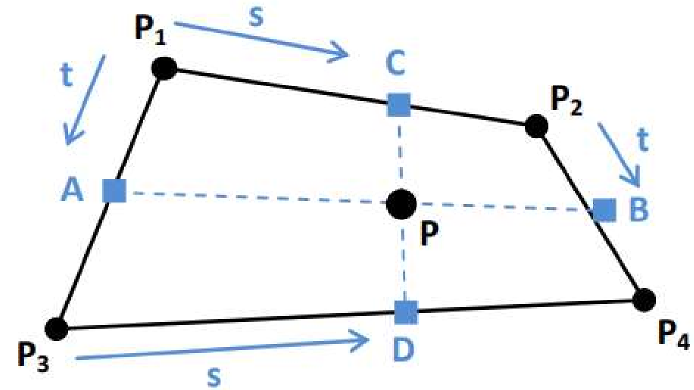
Scattered Data Interpolation

- What happens if you have data sampled irregularly?
  - Can you use bilinear interpolation?
- There are interpolation methods for unstructured data
  - i.e. for data sampled in any pattern
- Could we use barycentric coordinates?
  - How?
  - What would the main drawbacks be?



Scattered Data Interpolation

- What happens if you have data sampled irregularly? Bad things
  - Can you use bilinear interpolation? No..not by the method we have seen It can be done using non-linear interpolation
- There are interpolation methods for unstructured data
  - i.e. for data sampled in any pattern
- Could we use barycentric coordinates? Yes
  - How? Triangulate the domain using the sample points as vertices
  - What would the main drawbacks be? Possibly poor reconstruction of function, especially around the triangle boundaries



Scattered Data Interpolation

- There are interpolation methods for unstructured data
  - i.e. for data sampled in any pattern
- We will look at 2 types:
  - Shepard's interpolation
  - Radial Basis Functions
- Both allow us to interpolate without meshing
- Functions are defined in terms of distance from a point
  - Hence the the term *radial*

Shepard's Interpolation

$$f_i(\mathbf{p}) = \frac{(\|\mathbf{p} - \mathbf{p}_i\|^{-\alpha})}{\sum_{j=1}^{n} (\|\mathbf{p} - \mathbf{p}_j\|^{-\alpha})} v_i$$

$$f(\mathbf{p}) = \sum_{i=1}^{n} f_i(\mathbf{p})$$

- Point p is the location at which we are interpolating
  - f(p) is the interpolated value
- The pi are the sample points
- The vi are the function values at pi
- Alpha is a positive real number
  - What does it control?

Shepard's Interpolation

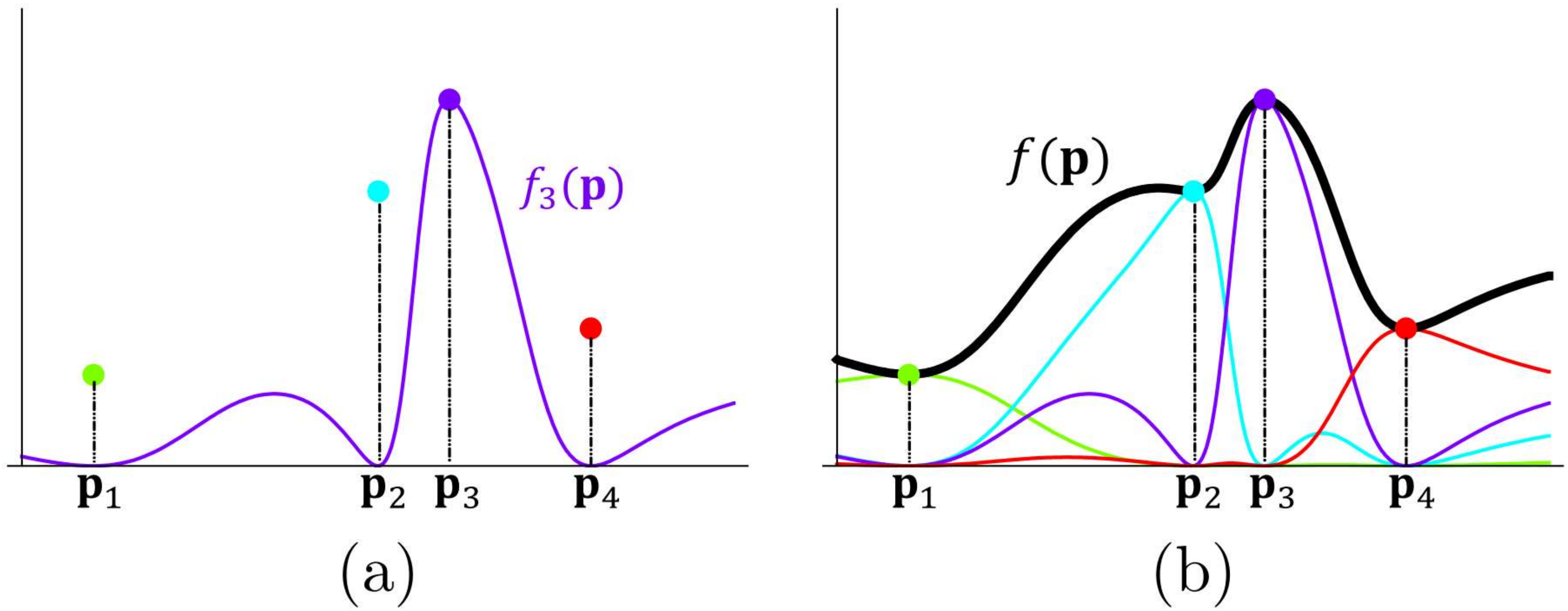
$$f_i(\mathbf{p}) = \frac{(\|\mathbf{p} - \mathbf{p}_i\|^{-\alpha})}{\sum_{j=1}^{n} (\|\mathbf{p} - \mathbf{p}_j\|^{-\alpha})} v_i$$

$$f(\mathbf{p}) = \sum_{i=1}^{n} f_i(\mathbf{p})$$

- When approaching pi, the bump function fi(p) has both numerator and denominator approach infinity
  - When approaching pj then fi(p) approaches 0
- Not confined to convex hull of points…can extrapolate
- Not ideal…flattens at pi and has more waviness than seems natural

Shepard's Interpolation: Example

- Using alpha=2



Radial Basis Functions (RBFs)

- Any function dependent on distance from a center is *radial*
- We can compute an approximate function as a weighted sum..

$$\phi(x, p) = \phi(\|x - p\|)$$

$$f(x) \approx \sum_{i=1}^{N} w_i \phi(x, p_i)$$

- Some popular RBFs include

$$\phi(r) = e^{-\lambda r^2}$$ Gaussian

$$\phi(r) = \frac{1}{1 + r^2}$$ Inverse distance

λ is a parameter you choose. What behavior does it control?

r is a distance

Radial Basis Functions (RBFs)

- We have the freedom to choose the weights w_i
- But we have to interpolate the data
- We can find the weights that allow us to do that by solving a system of equations

$$f(p_j) = \sum_{i=1}^{N} w_i \phi(p_j, p_i)$$

$$Aw = p$$

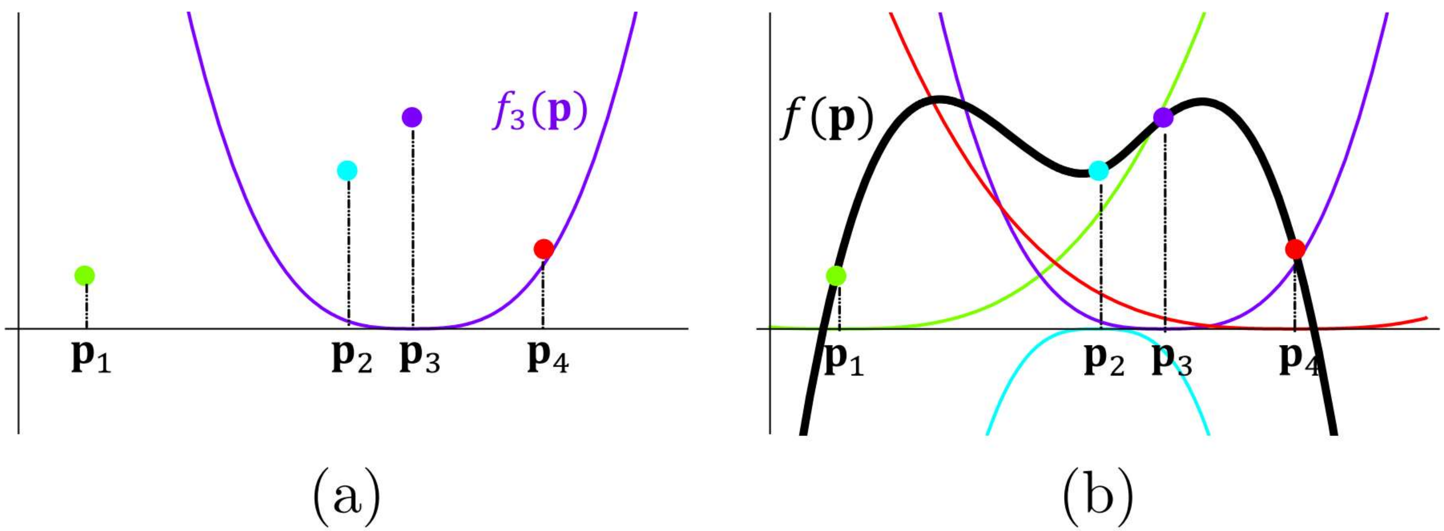
$$A = \begin{bmatrix} \phi(p_1, p_1) & \dots & \phi(p_1, p_N) \\ \dots & \dots & \dots \\ \phi(p_N, p_1) & \dots & \phi(p_N, p_N) \end{bmatrix}$$

$$w = \begin{bmatrix} w_1 \\ \dots \\ w_N \end{bmatrix}$$

$$p = \begin{bmatrix} f(p_1) \\ \dots \\ f(p_N) \end{bmatrix}$$

Example: RBF

- Using triharmonic radial functions

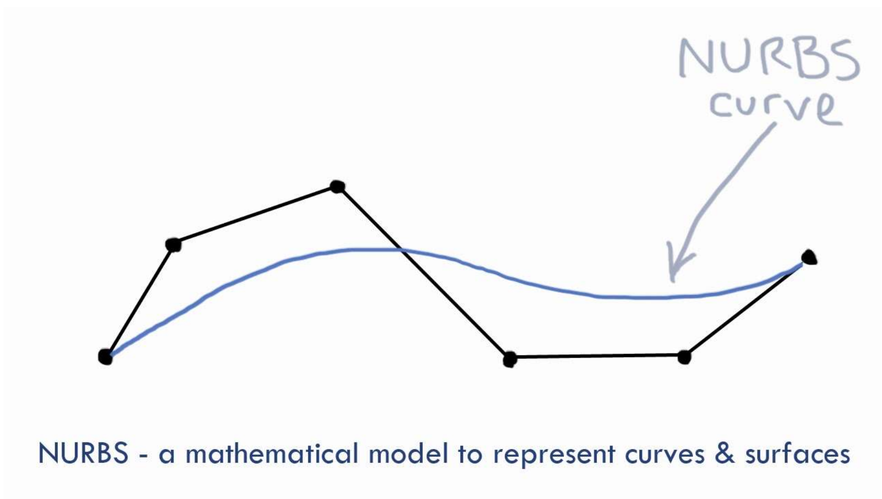


RBF Issues

- Doesn’t scale well
- For large number of points A becomes ill-conditioned
  - What does that mean?
- Also solving system takes time
  - How much for general Gaussian Elimination on an n x n?
- Just evaluating the interpolant takes time
- For large data sets, need to use RBFs with local support
  - Creates a sparse system more amenable to fast solvers

Filling in Missing Data

- There's lots of other options
- Especially polynomial interpolants
  - Especially piecewise polynomial interpolants
  - Splines (need extra control points)
- Could also use approximation methods
  - Least squares best fit….



Reviewing Assigned Reading: Continuous Data

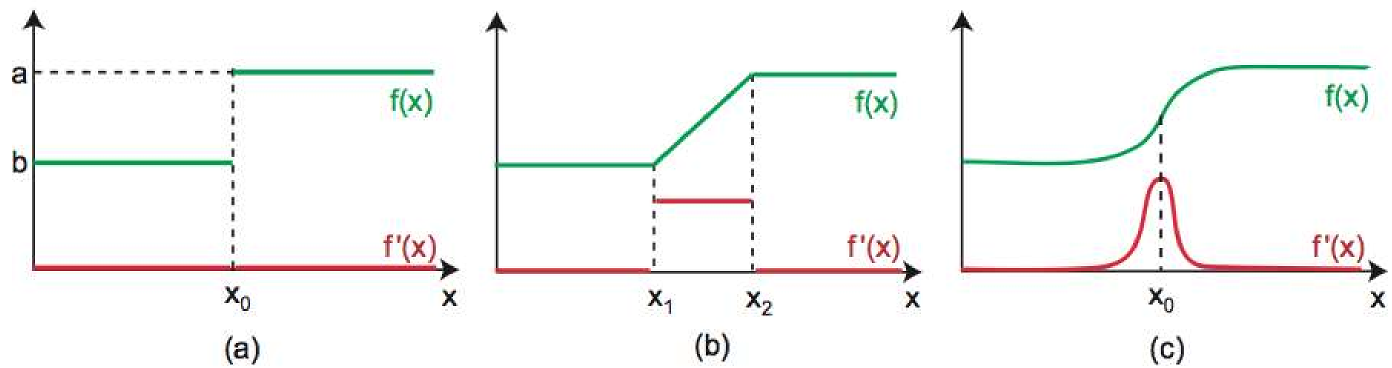
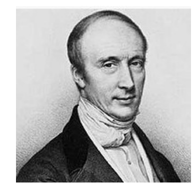


Figure 3.1. Function continuity. (a) Discontinuous function. (b) First-order $\mathcal{C}^0$ continuous function. (c) High-order $\mathcal{C}^k$ continuous function.

Cauchy definition of continuity



A function f is continuous iff

$$\forall \epsilon > 0, \exists \delta > 0 \text{ such that if } \|x - p\| < \delta, x \in \mathrm{C} \text{ then } \|f(x) - f(p)\| < \epsilon$$

C^0	graph of **derivative** of the function has "holes"
C^1	graph of **derivative** of function has "kinks"
C^k	first k derivatives of the function are continuous

Interpolation

Interpolation: Fundamental tool for signal reconstruction

1. **Reconstruction** formula

$$\tilde{f} = \sum_{i=1}^{N} f_i \phi_i \qquad \phi_i : \mathrm{D} \to \mathrm{C}$$ are basis (or interpolation) functions

2. **Interpolation**: reconstruction passes through (interpolates) the sampled values

$$\sum_{i=1}^{N} f_i \phi_i(p_j) = f_j, \forall j$$ because $\tilde{f}(p_i) = f(p_i) = f_i$

3. **Orthogonality** of basis functions

$$\phi_i(p_j) = \begin{cases} 1, & i = j, \\ 0, & i \neq j. \end{cases}$$

why? Just apply (2) to $f = \begin{cases} 1, p = p_j \\ 0, p \neq p_j \end{cases}$

4. **Normality** of basis functions

$$\sum_{i=1}^{N} \phi_i(x) = 1, \forall x \in \mathrm{D}$$

why? $\sum_{i=1}^{N} \phi_i(p_j) = 1, \forall p_j$ (sum (3) over $i = 1..N$)

and apply above to all $p_i \in D$

Piecewise Interpolation

Recall the interpolation formula

$$\tilde{f} = \sum_{i=1}^{N} f_i \phi_i$$

This becomes very inefficient if

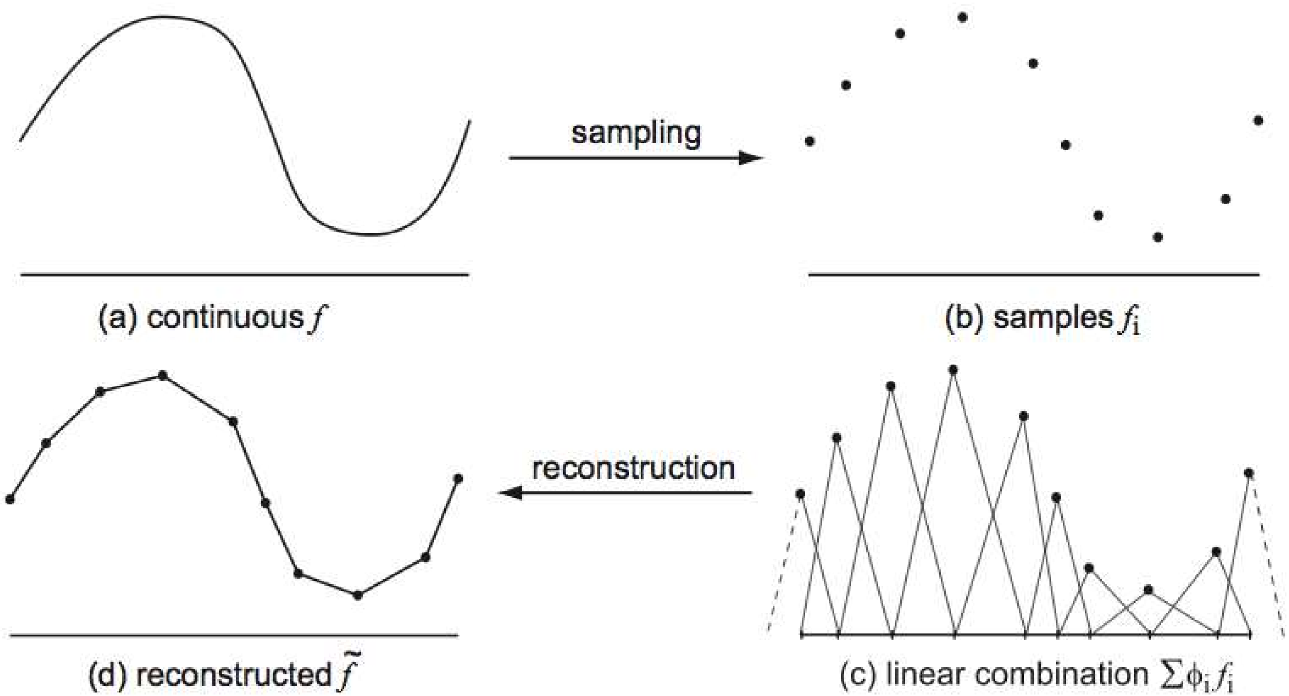
- N is very large and we have to evaluate ϕ_i at all these N points
- ϕ_i have complicated expressions

Practical basis functions

- are non-zero over small spatial 'pieces' of D only (limited support)
- have the same simple formula at all sample points p_i

⇒ Additional motivation for discretizing our spatial domain D into **cells**

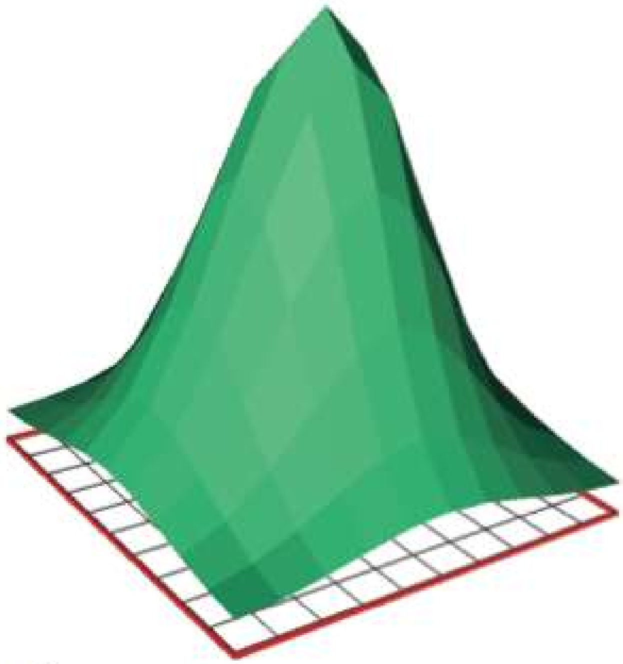
1D Example



Remarks

- interpolation & reconstruction goes cell-by-cell
- only need sample points at a cell vertices to interpolate over that cell

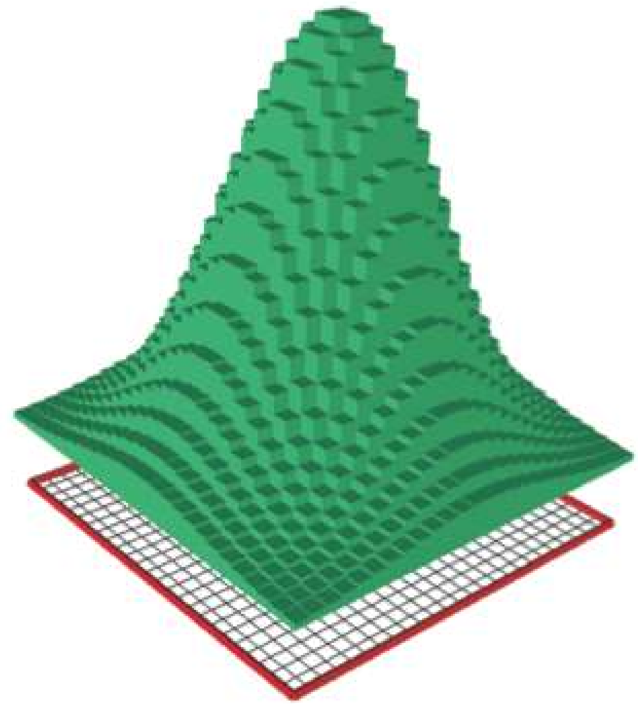
Bilinear interpolation



$$\Phi_1^1(r,s) = (1-r)(1-s),$$
$$\Phi_2^1(r,s) = r(1-s),$$
$$\Phi_3^1(r,s) = rs,$$
$$\Phi_4^1(r,s) = (1-r)s;$$

- 4 functions, one per vertex
- result: C^0 but never C^1 (why?)
- good for vertex-based samples

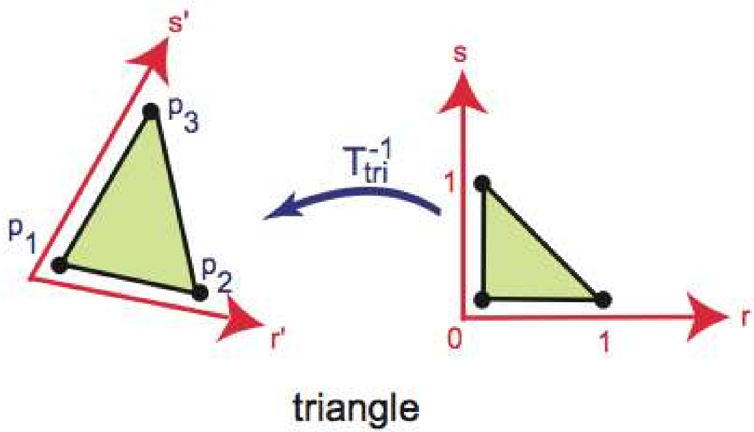
Constant interpolation



$$\phi_i^0(x) = \begin{cases} 1, & x \in c_i, \\ 0, & x \notin c_i. \end{cases}$$

- 1 functions per whole cell
- result: not even C^0
- good for cell-based samples

2D Cells: Triangles



$$\Phi_1^1(r,s) = 1 - r - s,$$
$$\Phi_2^1(r,s) = r,$$
$$\Phi_3^1(r,s) = s.$$

Remarks

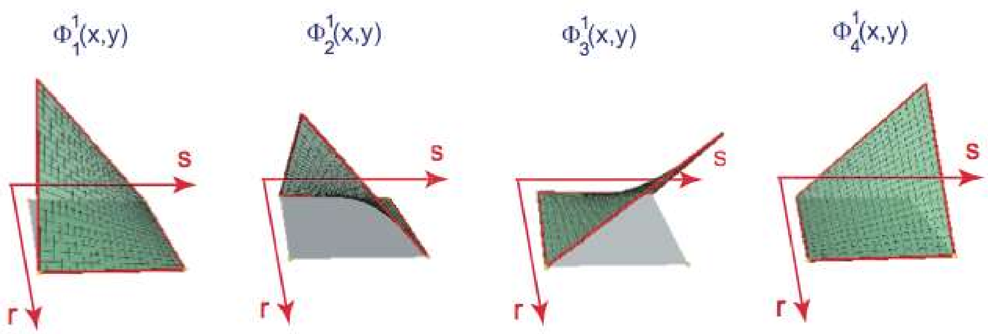
- in graphics/visualization, triangles used more often than quads
  - easier to cover complex shapes with triangles than quads
  - same computational complexity

From the Book: 2D Cells (Quads)

Same as in 1D case, but

- we have to decide on different cells; say we take quads
- quads → 4 vertices, 4 basis functions
- particular case: square cells = pixels

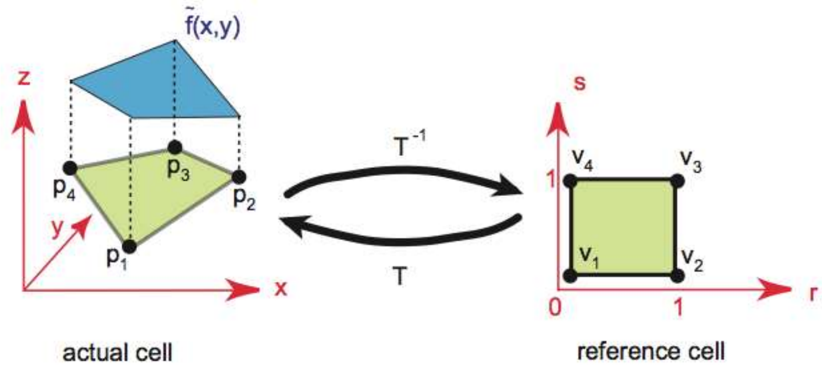
Bilinear basis functions



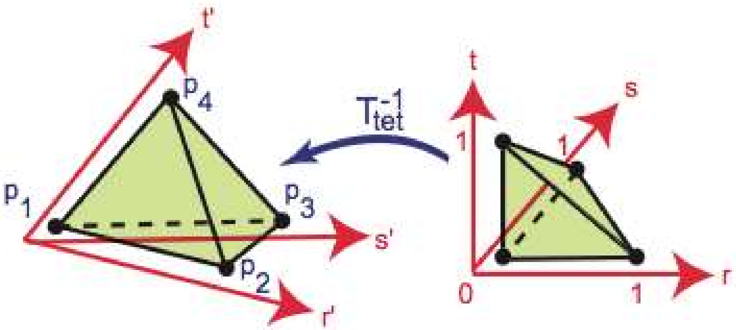
reference basis functions

$$\Phi_1^1(r,s) = (1-r)(1-s),$$
$$\Phi_2^1(r,s) = r(1-s),$$
$$\Phi_3^1(r,s) = rs,$$
$$\Phi_4^1(r,s) = (1-r)s;$$

Bilinear transforms



3D Cells: Tetrahedra



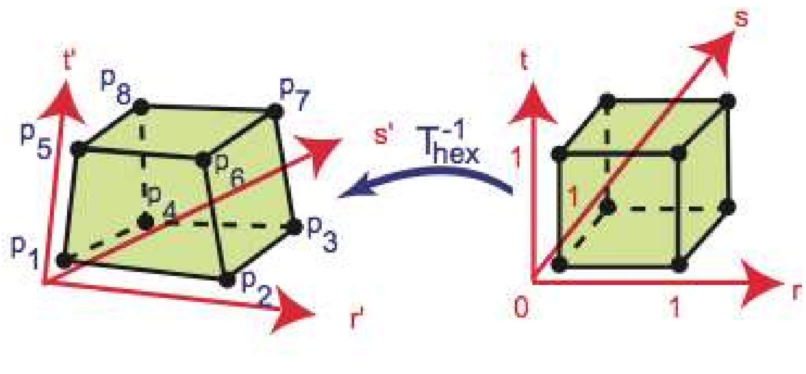
tetrahedron

$$\Phi_1^1(r,s,t) = 1 - r - s - t,$$
$$\Phi_2^1(r,s,t) = r,$$
$$\Phi_3^1(r,s,t) = s,$$
$$\Phi_4^1(r,s,t) = t.$$

Remarks

- counterparts of triangles in 3D
- interpolate volumetric functions $f: \mathbf{R}^3 \to \mathbf{R}$
- three parametric coordinates r, s, t
- **tri**linear interpolation

3D Cells: Hexahedra



hexahedron

$$\Phi_1^1(r,s,t) = (1-r)(1-s)(1-t),$$
$$\Phi_2^1(r,s,t) = r(1-s)(1-t),$$
$$\Phi_3^1(r,s,t) = rs(1-t),$$
$$\Phi_4^1(r,s,t) = (1-r)s(1-t),$$
$$\Phi_5^1(r,s,t) = (1-r)(1-s)t,$$
$$\Phi_6^1(r,s,t) = r(1-s)t,$$
$$\Phi_7^1(r,s,t) = rst,$$
$$\Phi_8^1(r,s,t) = (1-r)st.$$

Remarks

- counterparts of quads in 3D
- interpolate volumetric functions $f: \mathbf{R}^3 \to \mathbf{R}$
- **tri**linear interpolation
- particular case: cubic cells or voxels

Common Cell Types

0D
- point

1D
- line

2D
- triangle, quad, rectangle

3D
- tetrahedron, parallelepiped, box, pyramid, prism, …

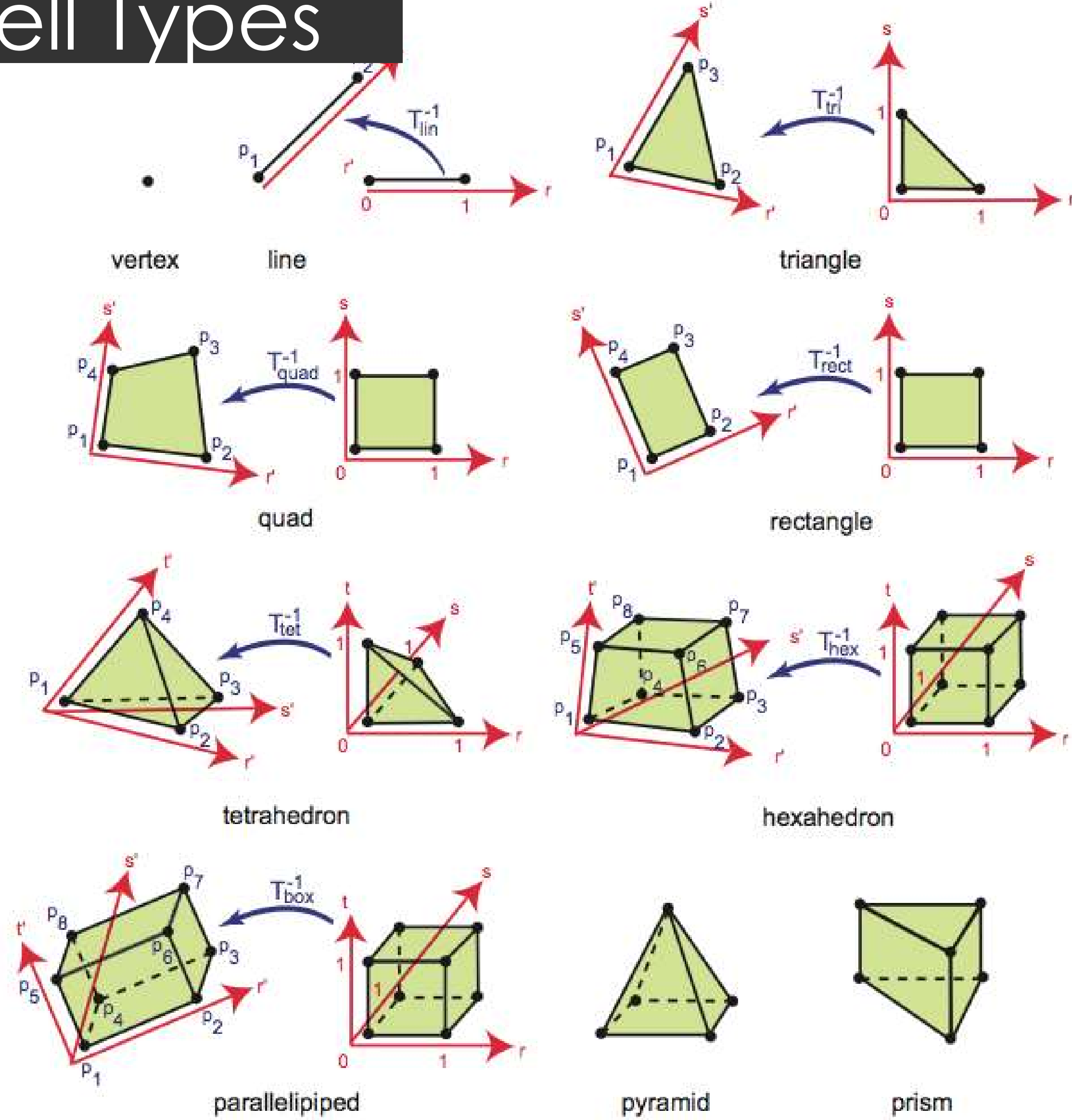


Figure 3.5. Cell types in world and reference coordinate systems.

Quadratic Cells

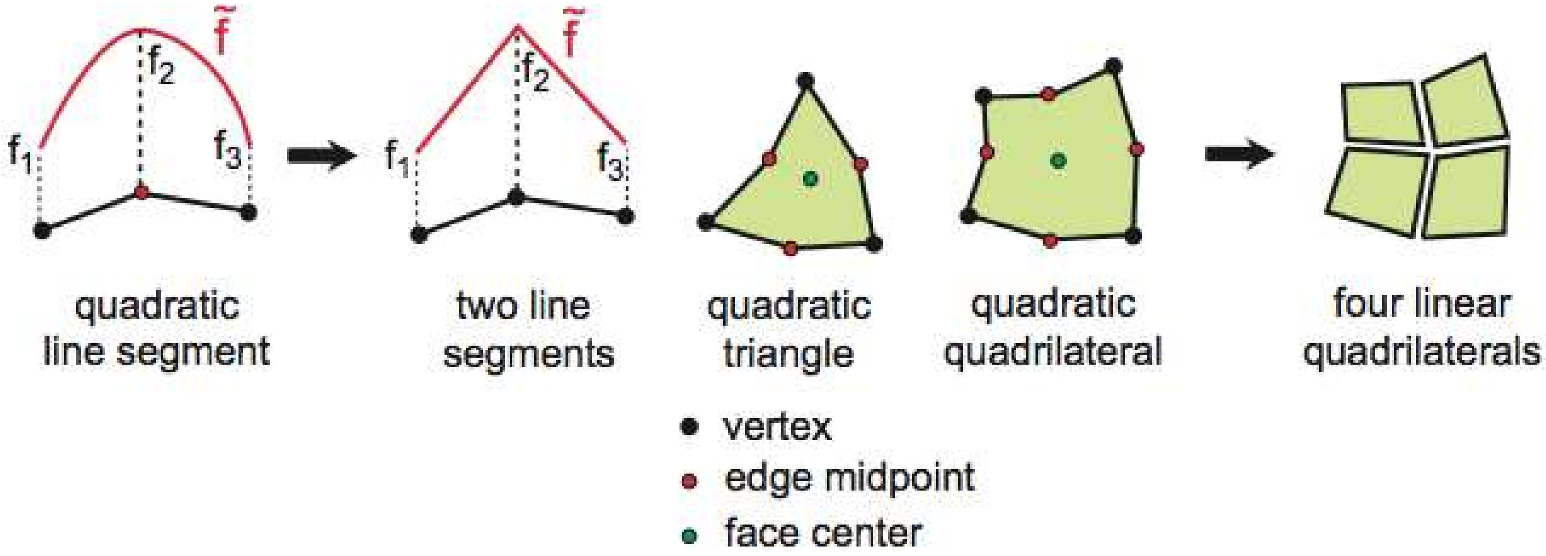


Figure 3.6. Converting quadratic cells to linear cells.

- allow defining quadratic basis functions
- higher precision for interpolation
- however, we need data samples at extra midpoints, not just vertices
- used in more complex numerical simulations (e.g. finite elements)
- split into linear cells for visualization purposes