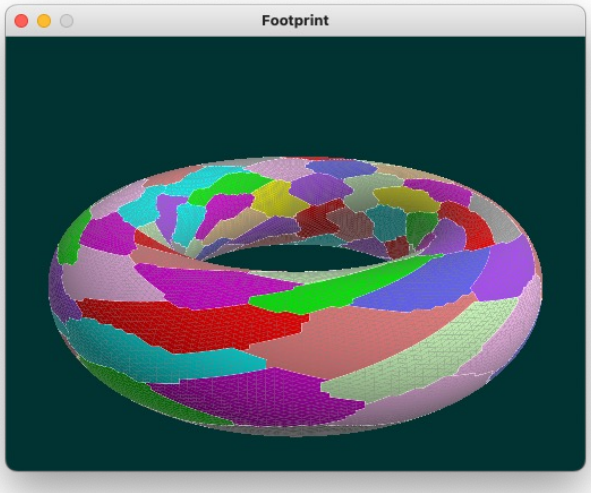
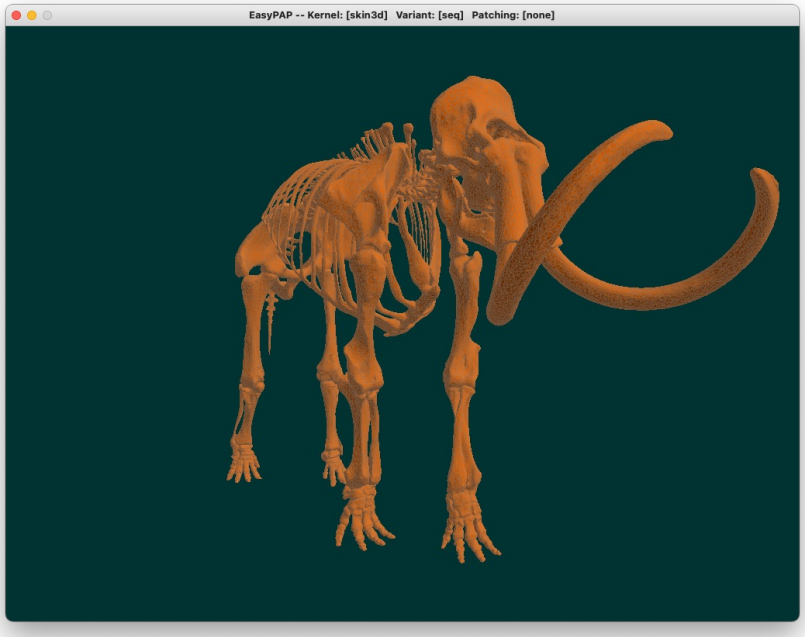


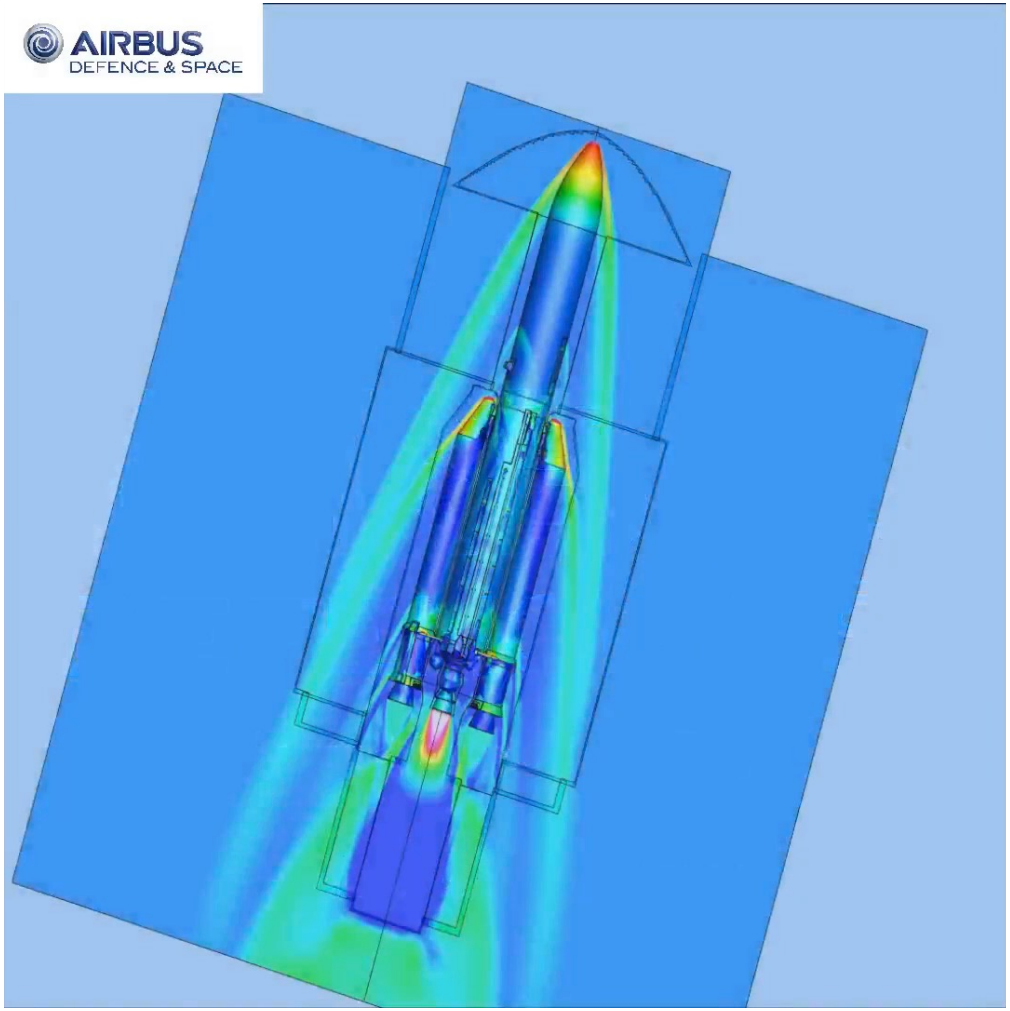
Data structures for 3d meshes

The EasyPAP Team

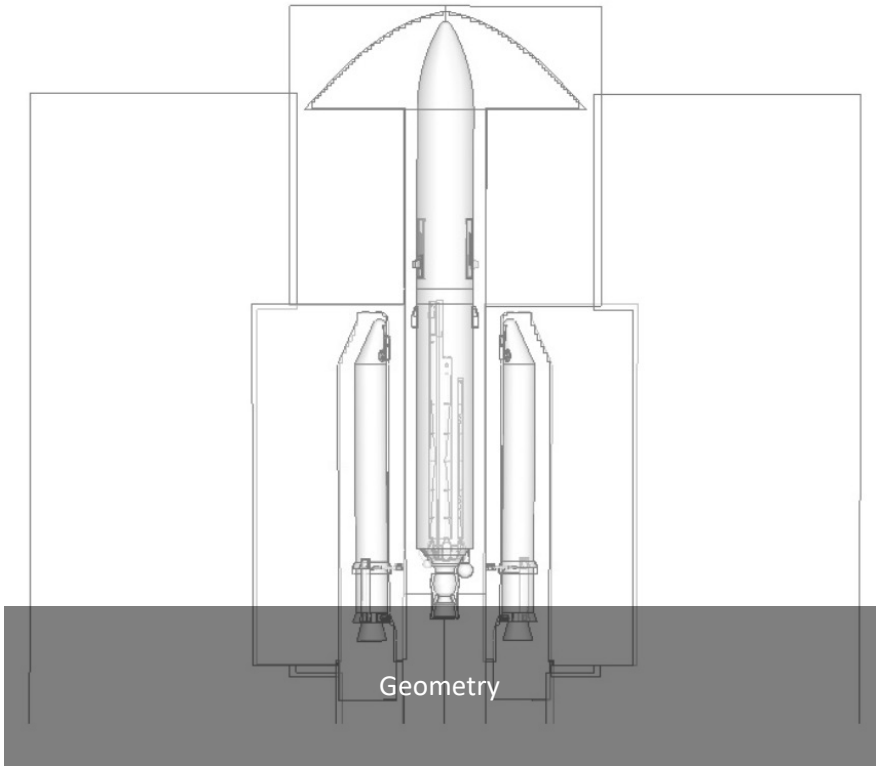


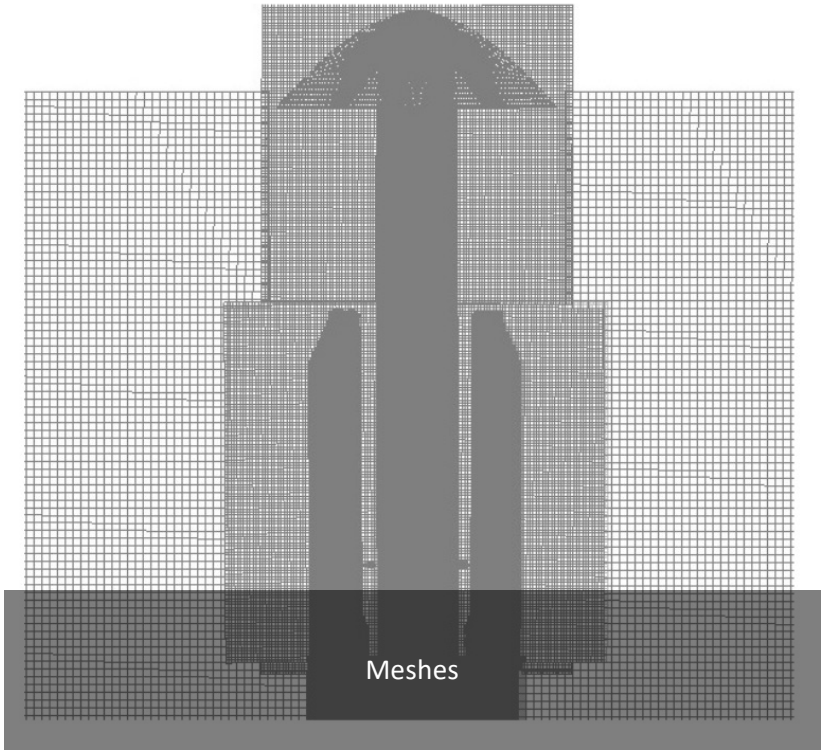
Numerical simulation

- Provide valuable insights and aiding in the understanding of complex fluid flow phenomena
  - difficult to analyze using analytical methods or physical experiments
    - blast wave propagation during rocket take-off
    - launch vehicle stage separations
    - noise generated by aircraft propellers



Numerical modeling of phenomena



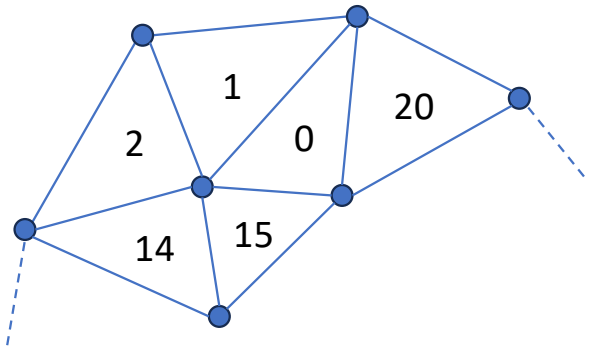


About Meshes

- Meshes com from .OBJ files
  - Straightforward text format
    - Vertices, faces, connectivity
    - [+ default partitioning]

```
v 1.3764 0.76354 0.0236
…
f 0 1 2
f 0 2 3
…
n 1 15 20
n 0 2
…
```

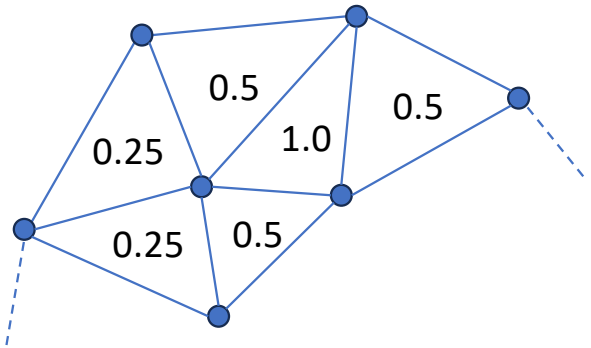
  - Most information is only used for 3D display



mesh

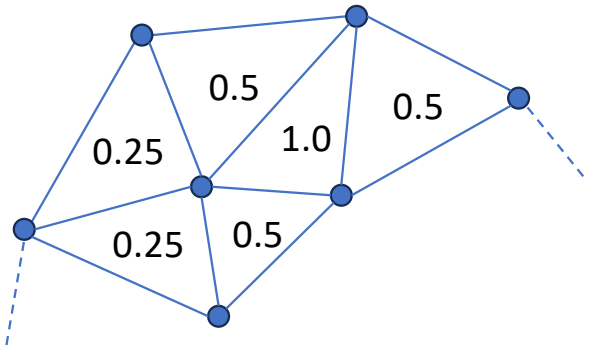
Doing computations on unstructured meshes

- A value representing a physical quantity (e.g. temperature) is attached to each cell
  - Normalized in [0.0..1.0]

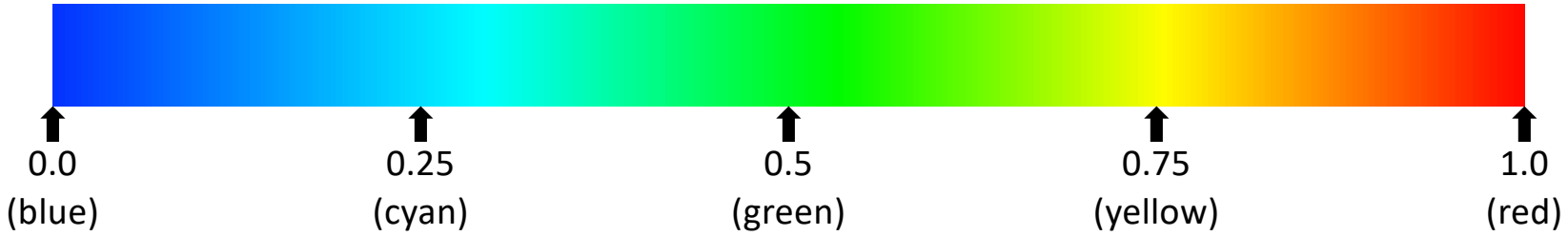


Doing computations on unstructured meshes

- A value representing a physical quantity (e.g. temperature) is attached to each cell
  - Normalized in [0.0..1.0]

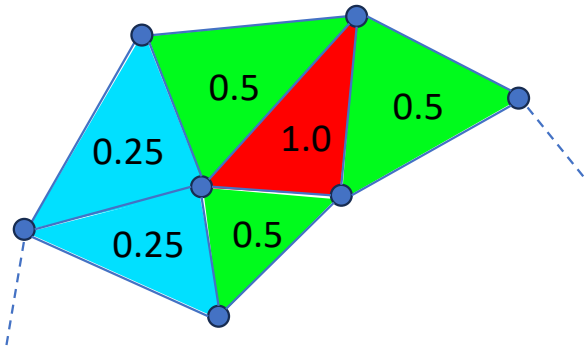


- Cells are colored by interpolation using a gradient palette

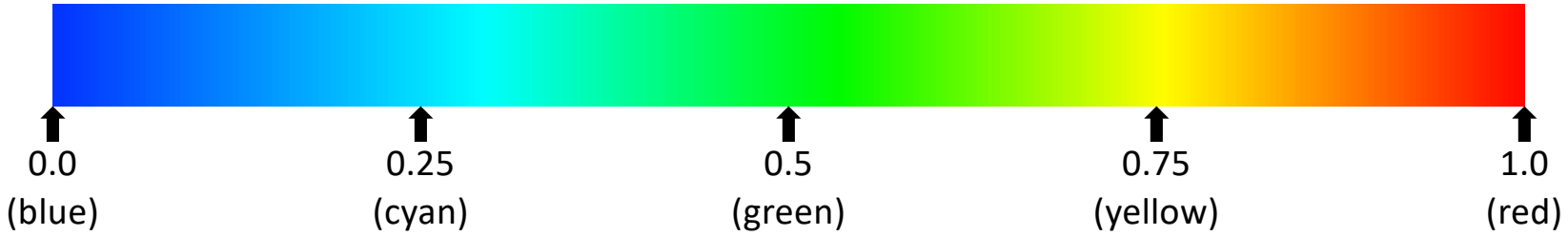


Doing computations on unstructured meshes

- A value representing a physical quantity (e.g. temperature) is attached to each cell
  - Normalized in [0.0..1.0]

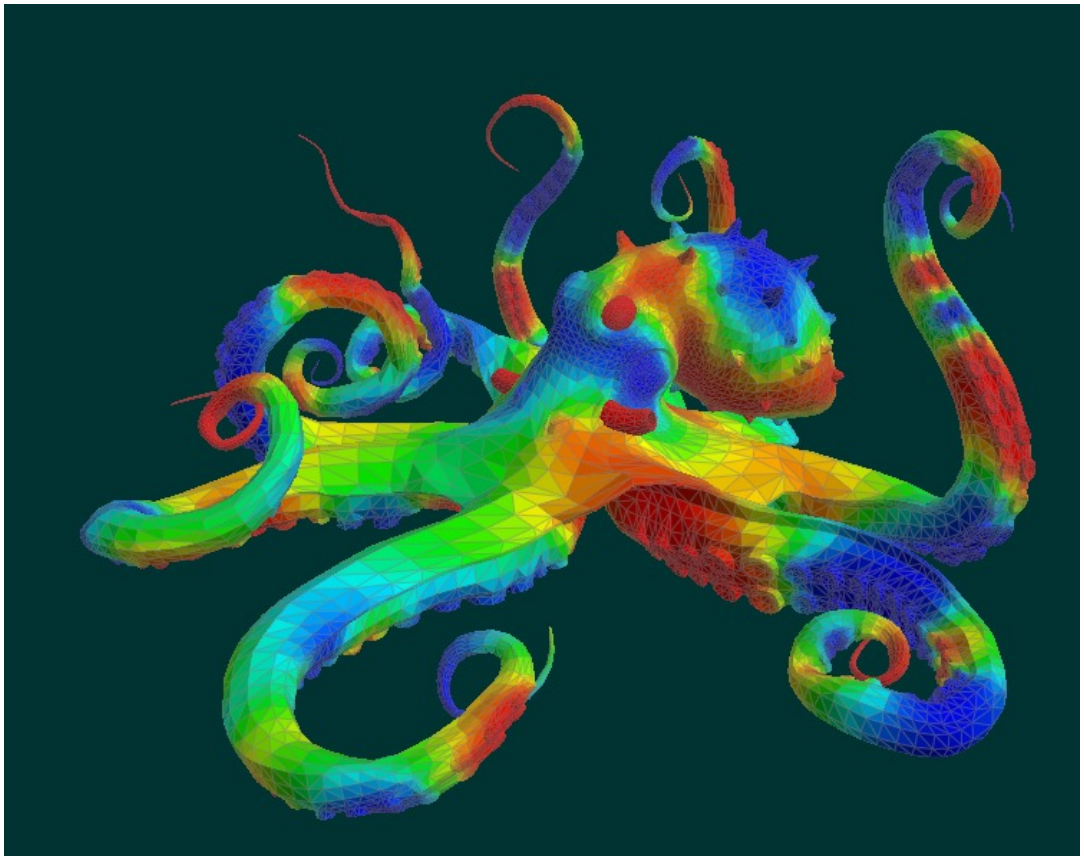


- Cells are colored by interpolation using a gradient palette



Doing computations on unstructured meshes

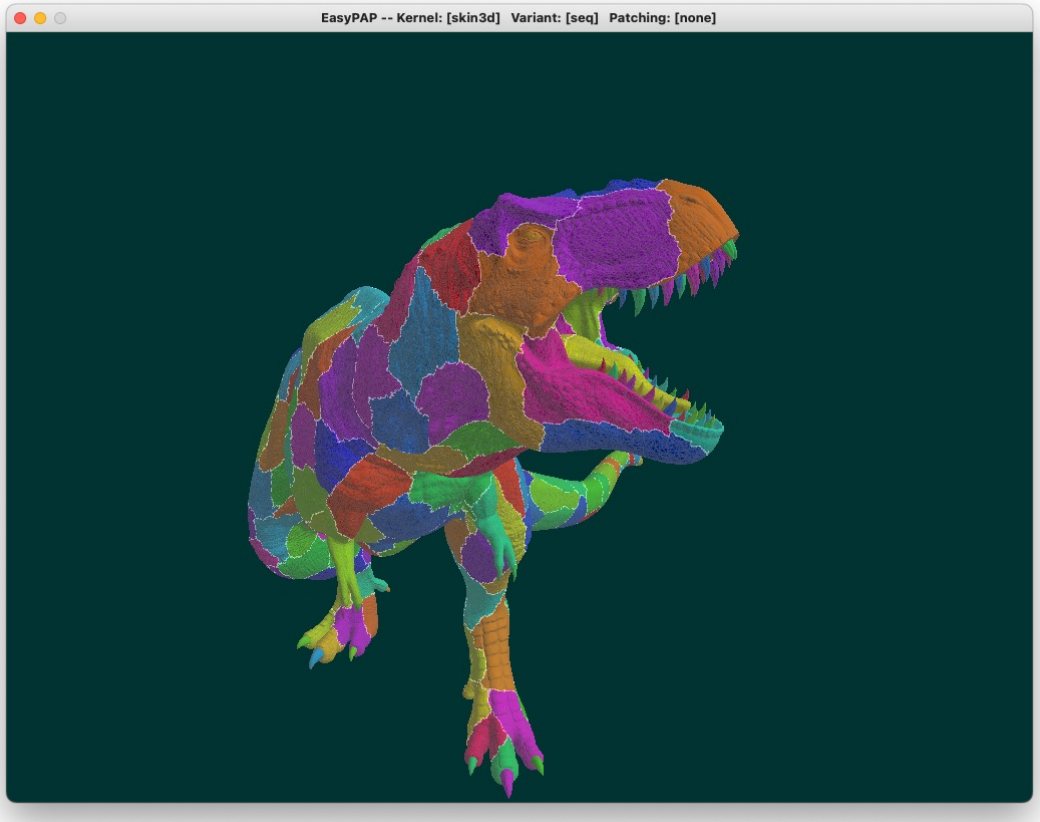
- A value representing a physical quantity (e.g. temperature) is attached to each cell
  - Normalized in [0.0..1.0]
- Cells are colored by interpolation using a gradient palette



Working with meshes in EasyPAP

- Pixels → Cells
  - Variable number of neighbors
- Colors → float values

- In EasyPAP, the programmer sees
  - NB_CELLS
  - cur_data (cell) and next_data (cell)
    - Arrays of floating-point values
    - Remember: values must be in range 0.0-1.0

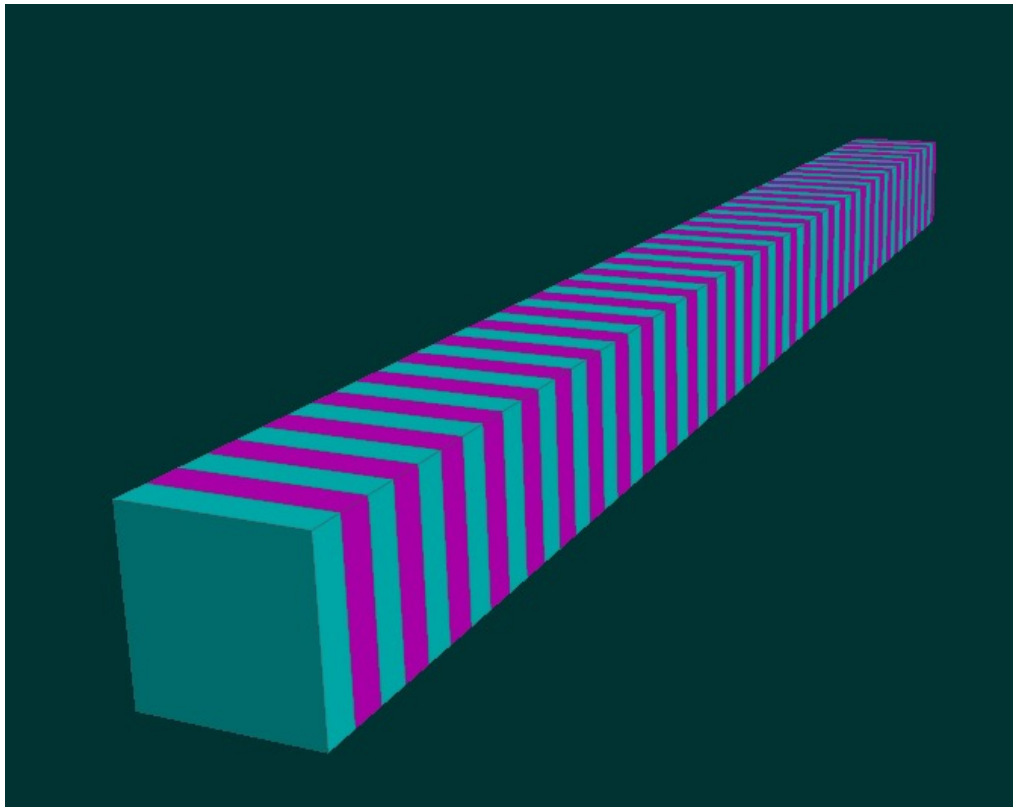


Setting a simple 2-point palette

```
void sample3d_config (char *param)
{
  // Choose color palette
  float colors[] = {0.0f, 0.8f, 0.8f, 1.f,  // cyan
                    0.8f, 0.0f, 0.8f, 1.f}; // pink
  mesh_data_set_palette (colors, 2);
}
```

Example alternating 0 and 1

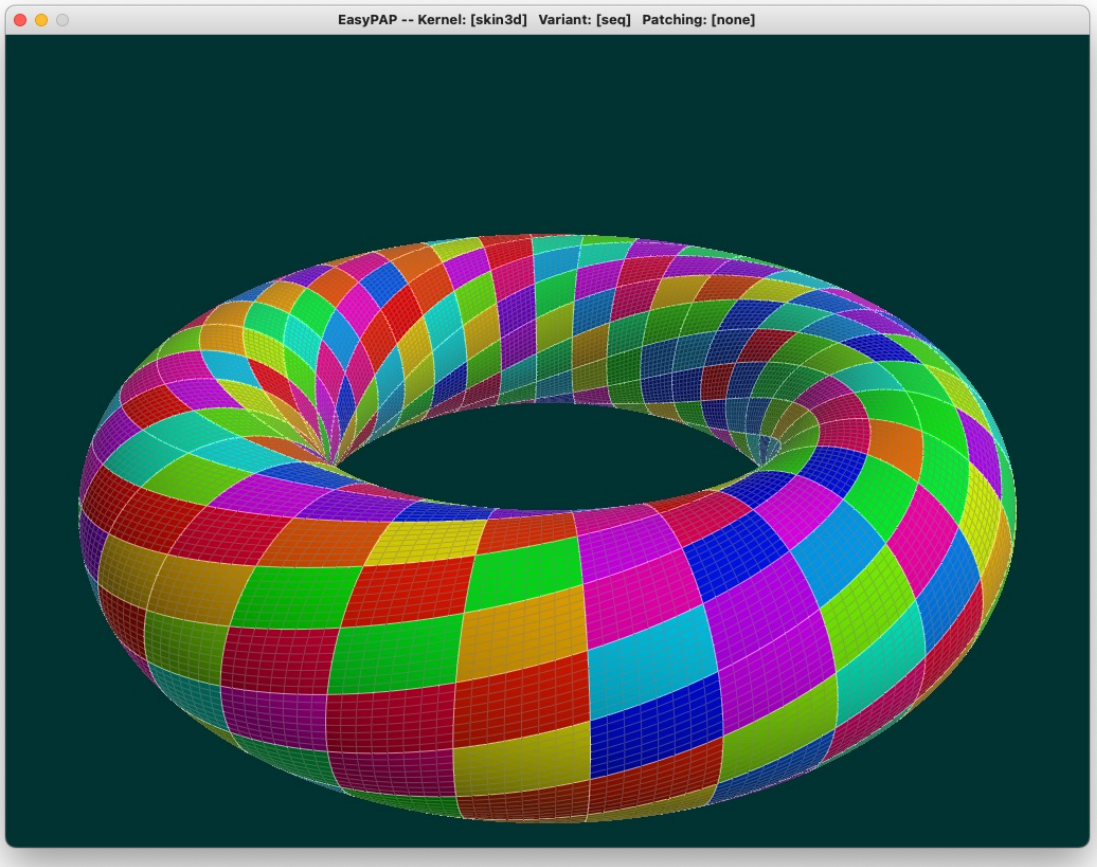
```
///////////////////////////// Sequential version (seq)
// Suggested cmdline:
// ./run -lm <mesh_file> -k sample3d -si
unsigned sample3d_compute_seq (unsigned nb_iter)
{
  for (unsigned it = 1; it <= nb_iter; it++)
    for (int c = 0; c < NB_CELLS; c++)
      cur_data (c) = (float)(c & 1);

  // Stop after first iteration
  return 1;
}
```



Where are my tiles?

- Pixels → Cells
  - Variable number of neighbors
- Colors → float values

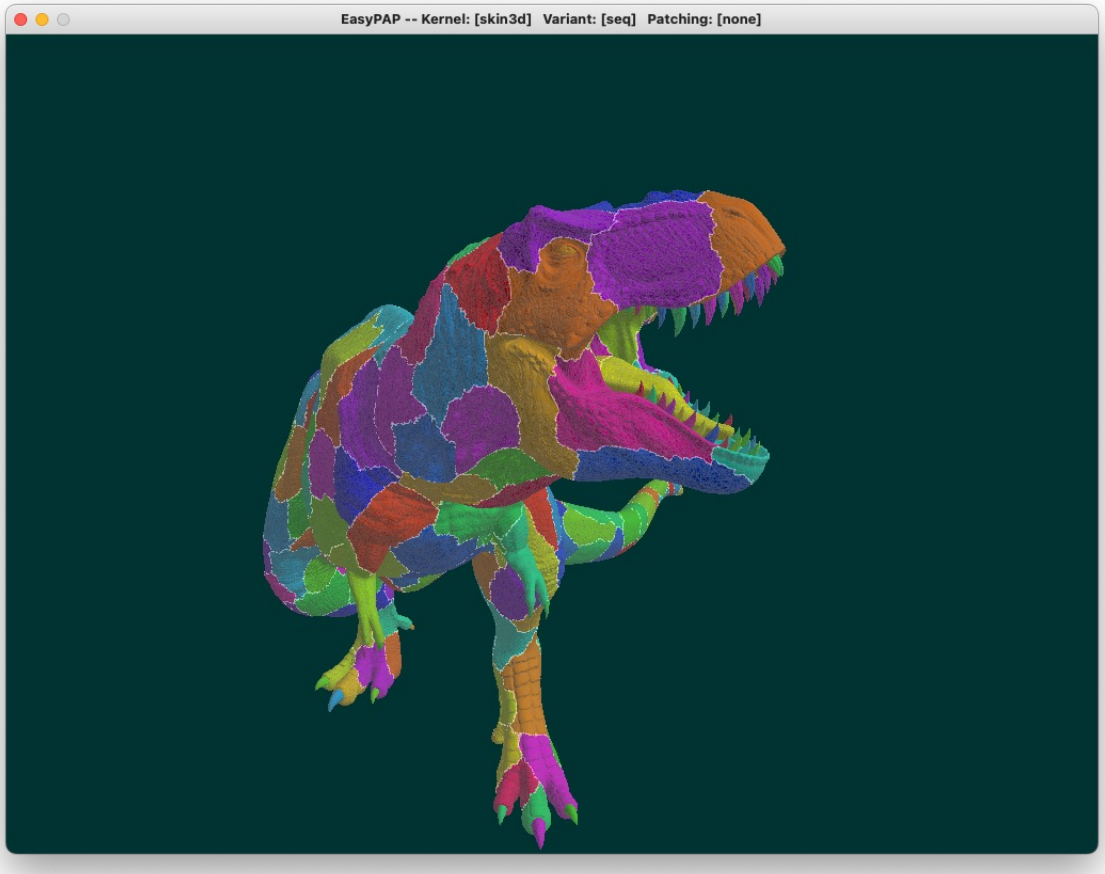
- Tiles → Patches
  - Obtained either by
    - Space filling curves
    - Scotch partitioning library



Where are my tiles?

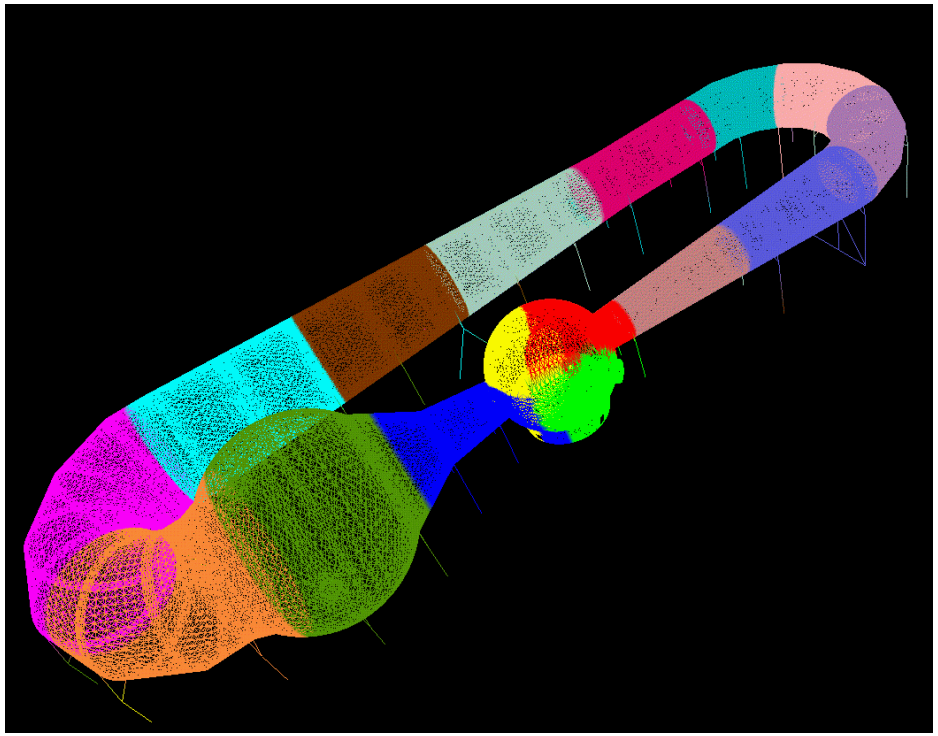
- Pixels → Cells
  - Variable number of neighbors
- Colors → float values

- Tiles → Patches
  - Obtained either by
    - Space filling curves
    - Scotch partitioning library



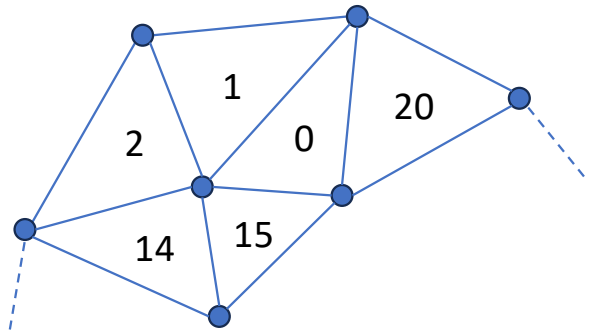
The Scotch Graph Partitioner

- Scotch is a graph partitioning library
  - F. Pellegrini, Univ. Bordeaux
- It optimizes workload distribution effectively
- Scotch is designed for parallel computing environments
  - It offers multiple efficient and flexible algorithms.
  - The library supports multiple graph formats.



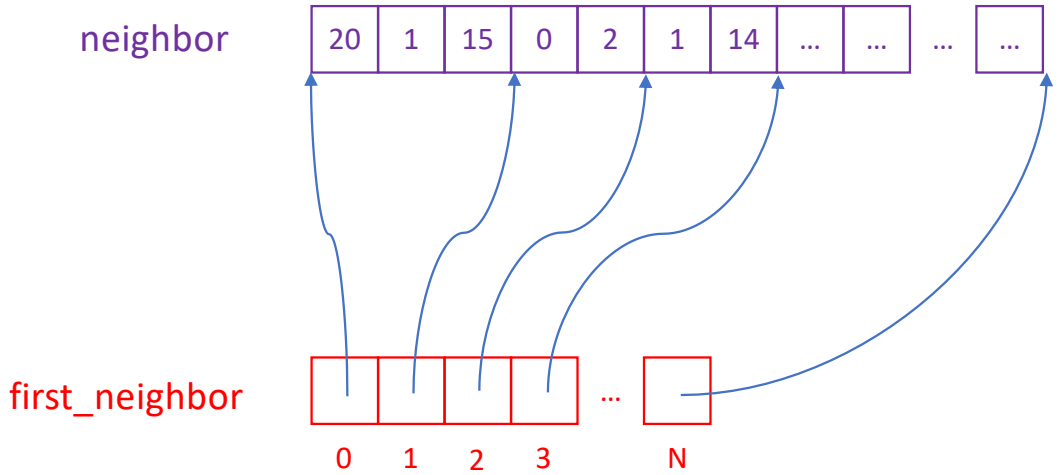
Accessing mesh connectivity

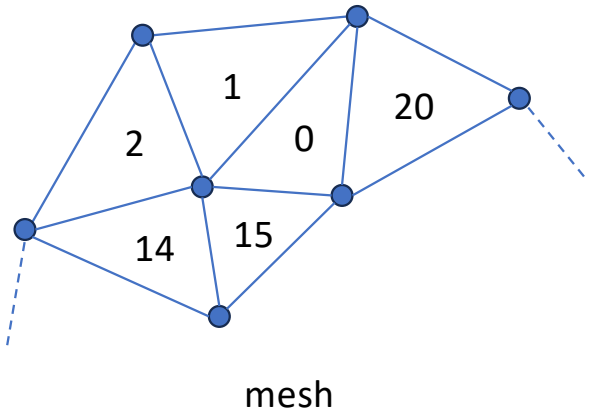
- In EasyPAP, the programmer sees
  - NB_CELLS
  - cur_data (cell) and next_data (cell)
- And
  - nb_neighbors (int cell)
  - nth_neighbor (int cell, int n)
  - max_neighbors ()
    - Max connectivity in the whole graph



mesh

Mesh connectivity:
a “compact edge array” is used by default

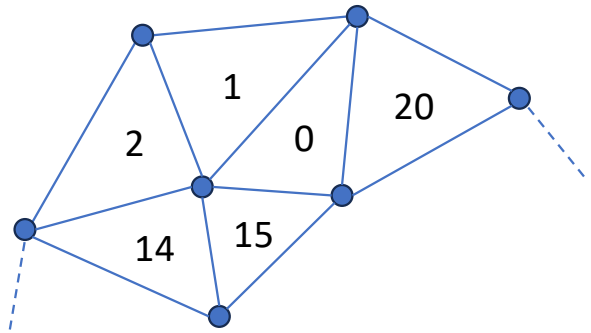




Neighbors of cell #i are stored in neighbors [first_neighbor[i] .. first_neightbor [i+1] – 1]

Mesh connectivity

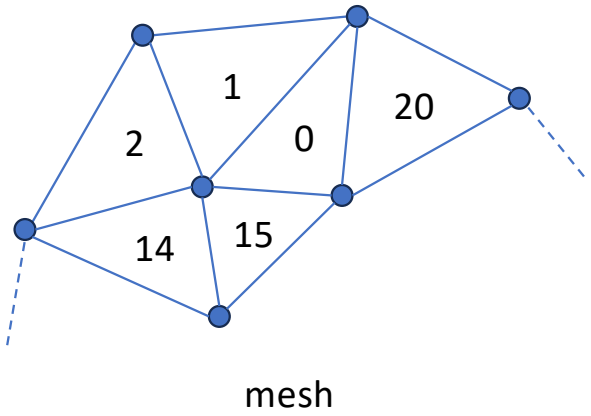
- Programmers can directly access the compact edge array
  - neighbor_start (int cell)
    - Index of first neighbor
  - neighbor_end (int cell)
    - Index of last neighbor + 1
  - neighbor (int index)



mesh

Accessing mesh connectivity in EasyPAP

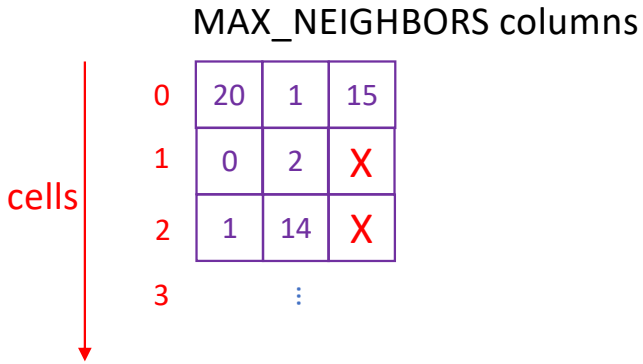
```
for (int nn = 0; nn < nb_neighbors (c); nn++) {
  int n = nth_neighbor (c, nn);
  float value = cur_data (n);
  …
}
// is equivalent to
for (int id = neighbor_start (c);
     id < neighbor_end (c);
     id++) {
  int n = neighbor (id);
  float value = cur_data (n);
  …
}
```

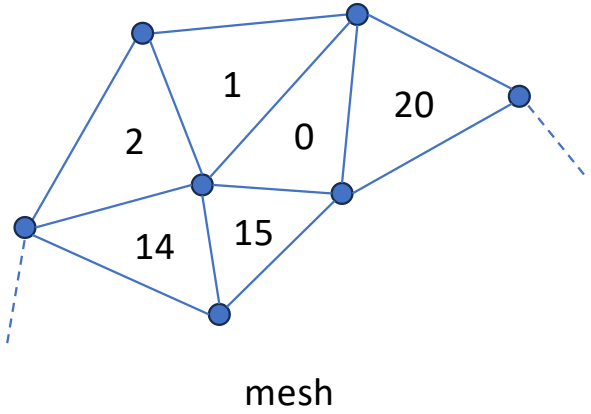


mesh

Towards a more regular data structure for GPUs and vector instructions

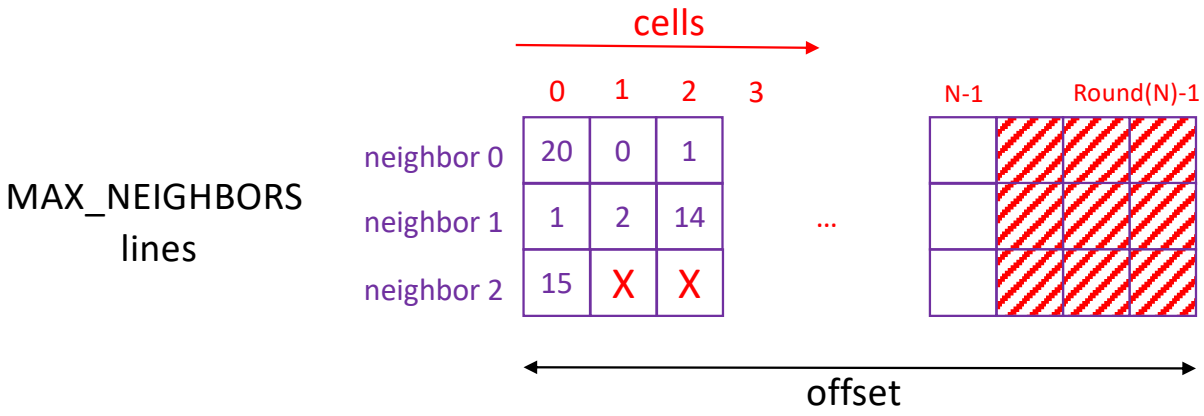
- Idea
  - MAX_NEIGHBORS is known
  - We can waste a little space and use a 2D array

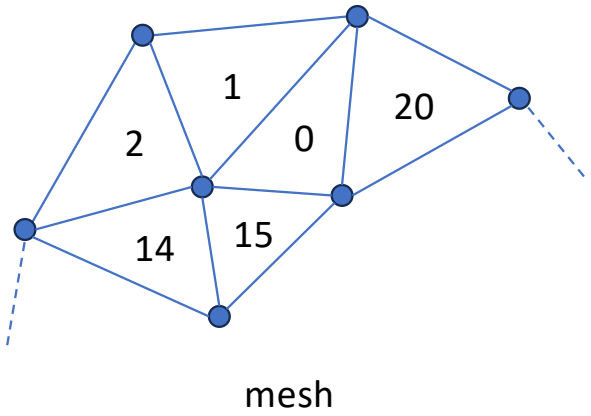




A SoA layout would better meet our needs

- 'neighbors_soa' is a 1D array padded for alignment purposes





This is what is used by default in AVX and OpenCL implementations