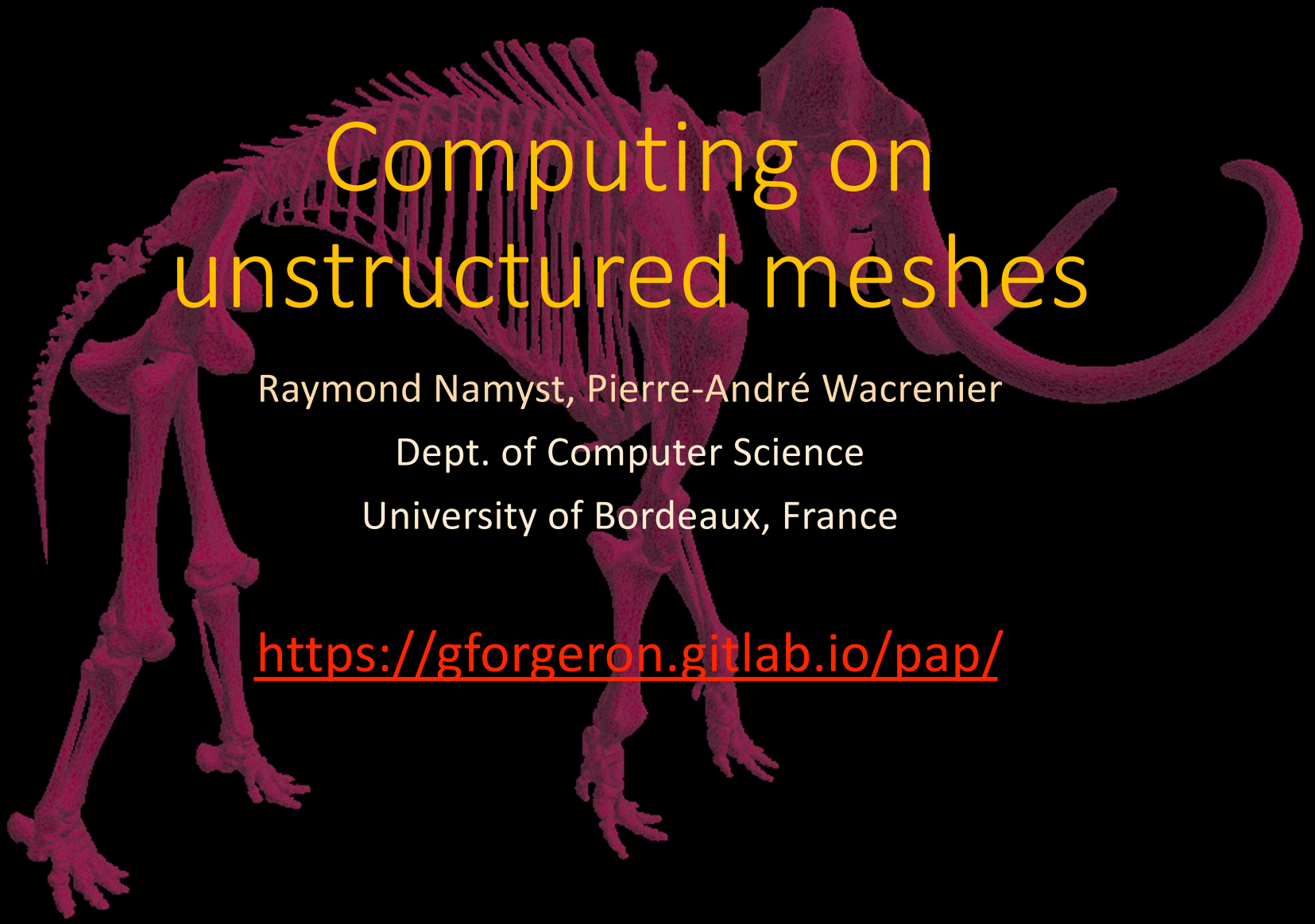


Computing on unstructured meshes

Raymond Namyst, Pierre-André Wacrenier

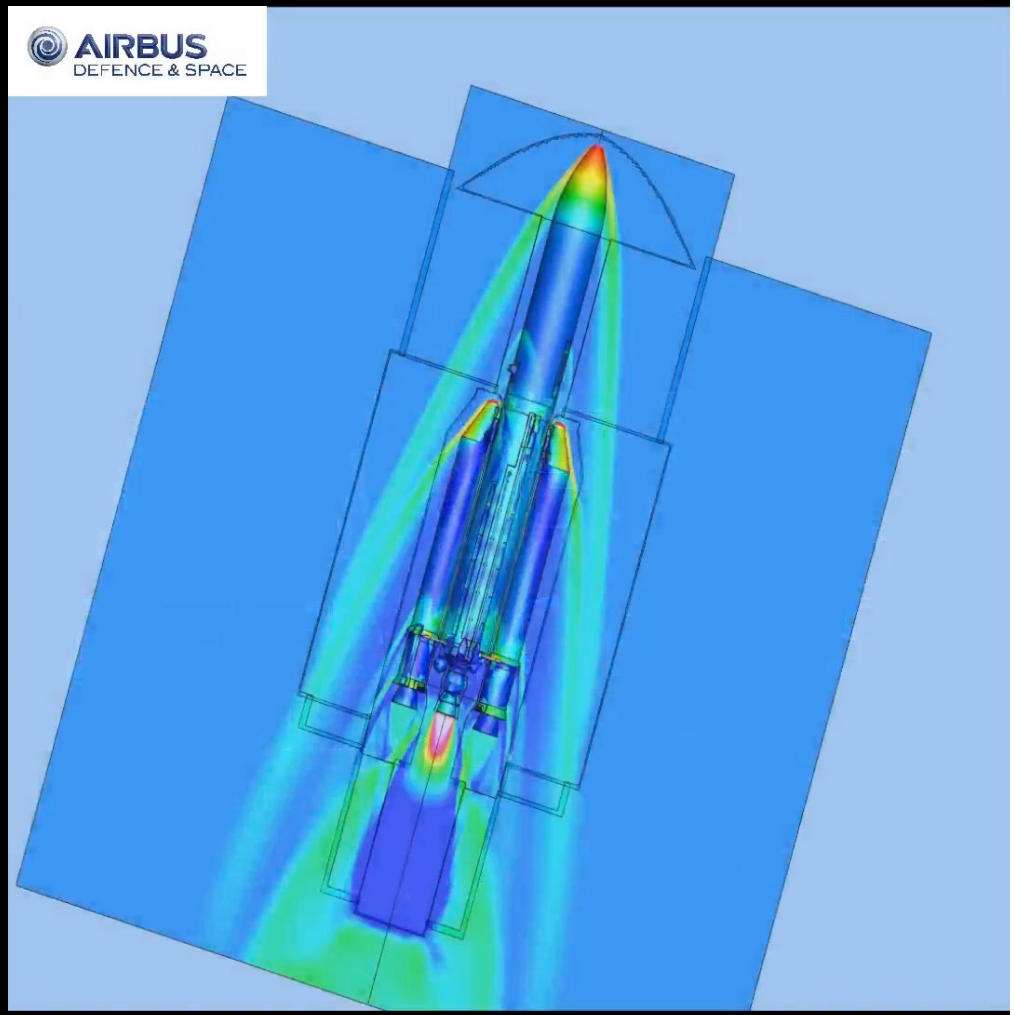
Dept. of Computer Science

University of Bordeaux, France

https://gforgeron.gitlab.io/pap/

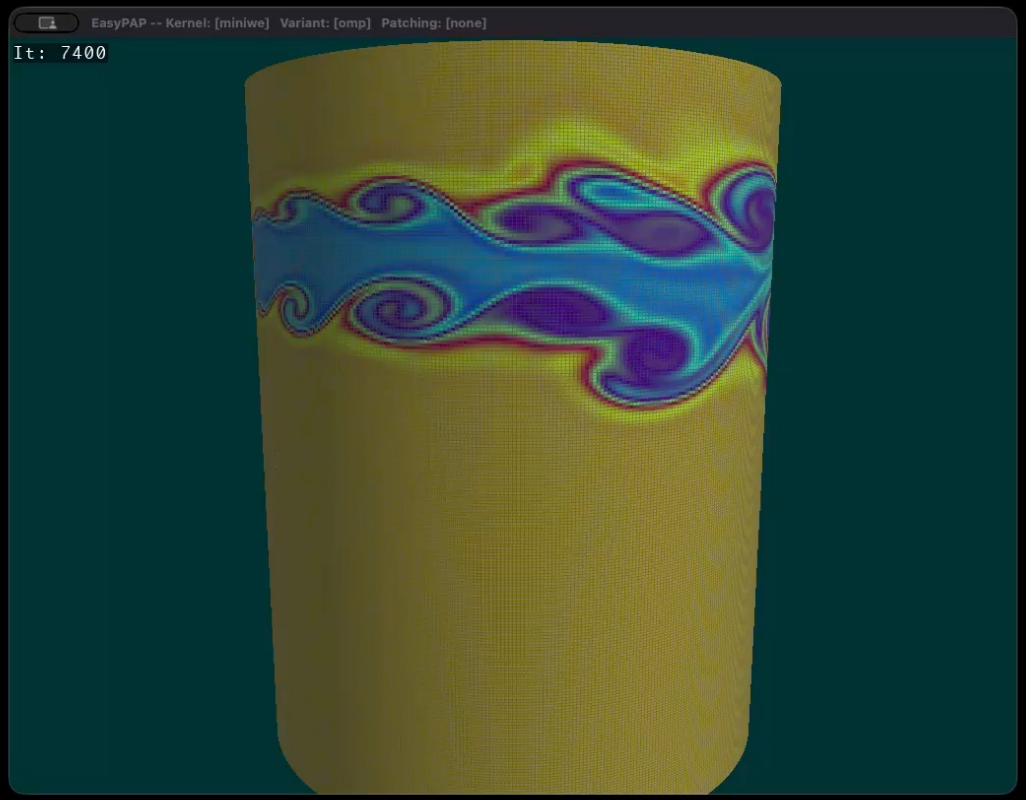
Numerical simulation

- Provide valuable insights and aiding in the understanding of complex 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
  - E.g. CFD simulations
    - Computational Fluid Dynamics

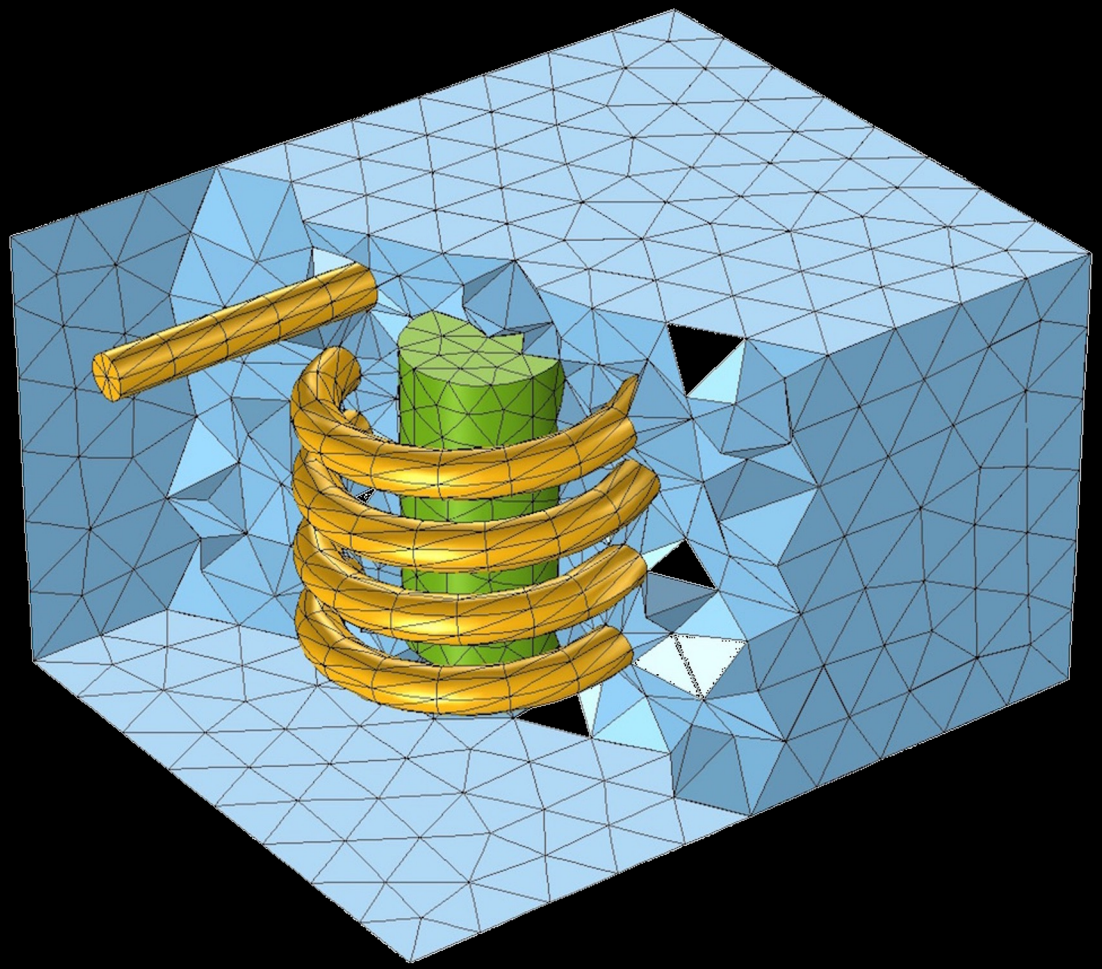


Numerical simulation

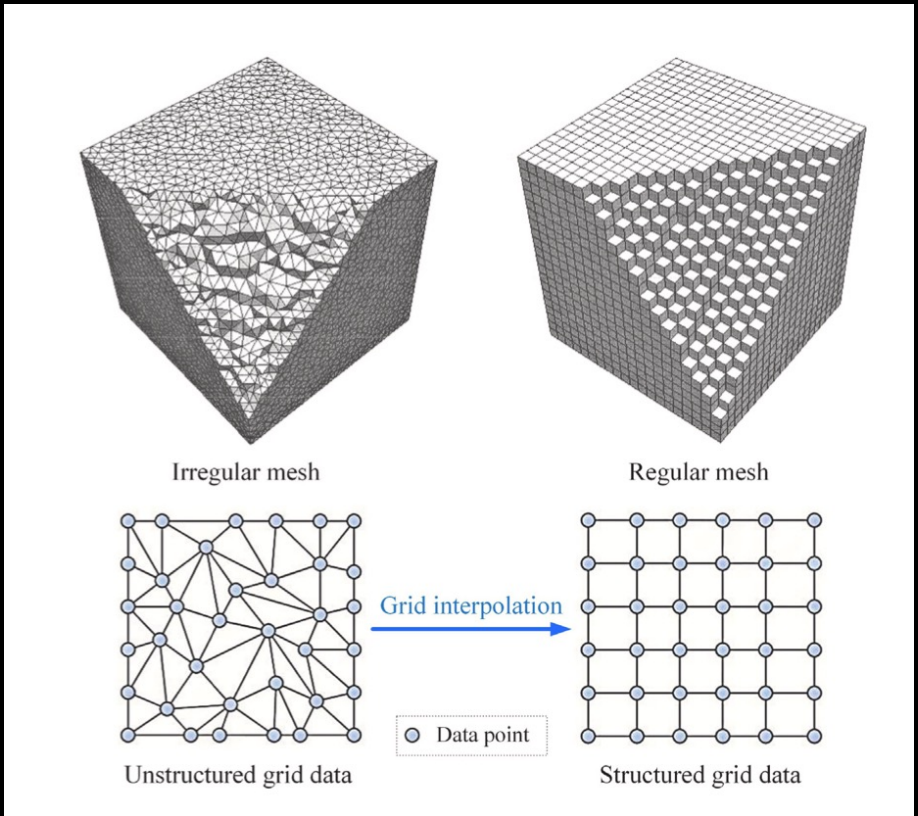
- Provide valuable insights and aiding in the understanding of complex phenomena
  - difficult to analyze using analytical methods or physical experiments
    - blast wave propagation during rocket take-off
    - launch vehicle stage separations
    - noise generated by aircraft propellers
  - E.g. CFD simulations
    - Computational Fluid Dynamics

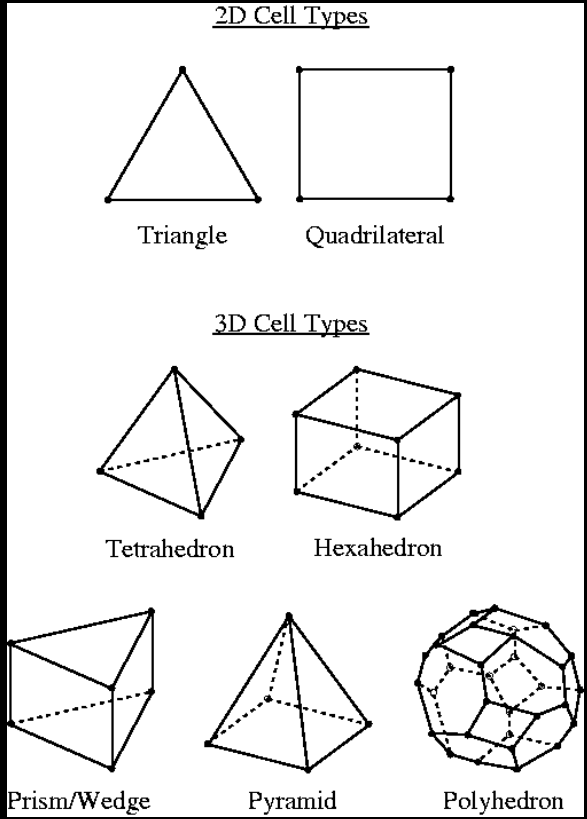


Numerical modeling of phenomena

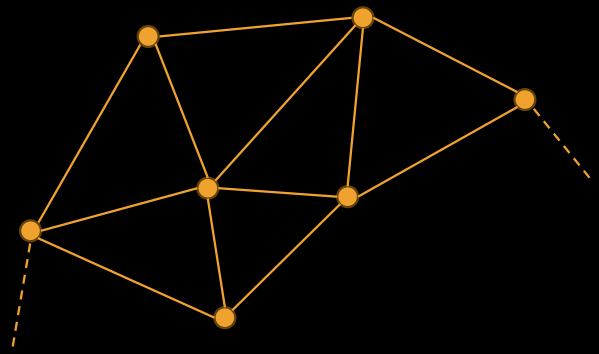


Structured vs unstructured





EasyPAP and Meshes

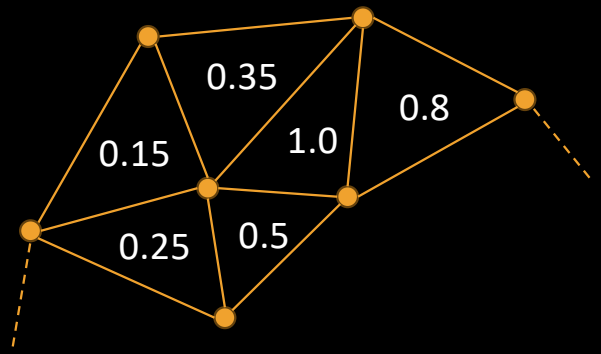


- Meshes come from .OBJ files
  - Straightforward text format
    - Vertices, faces, connectivity
    - [+ default partitioning]

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

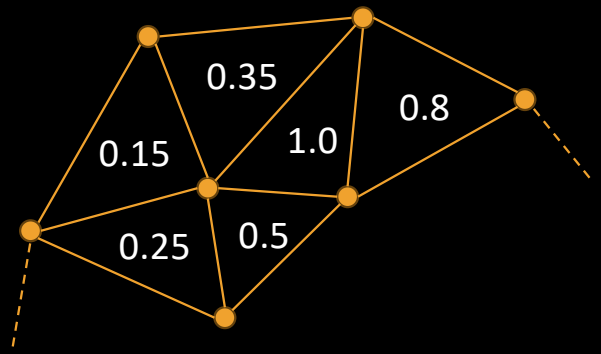
  - Most information is only used for 3D display

Doing computations on unstructured meshes



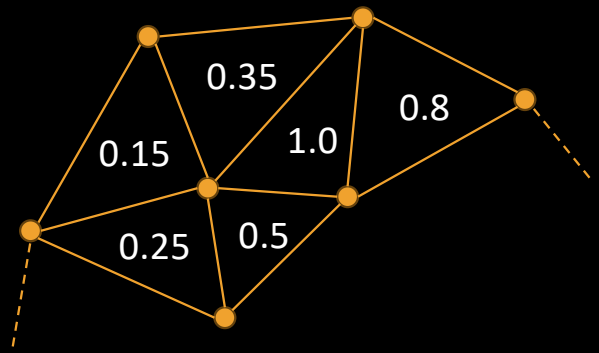
- A value representing a physical quantity (e.g. temperature) is attached to each cell
  - Normalized in [0.0..1.0]

Doing computations on unstructured meshes

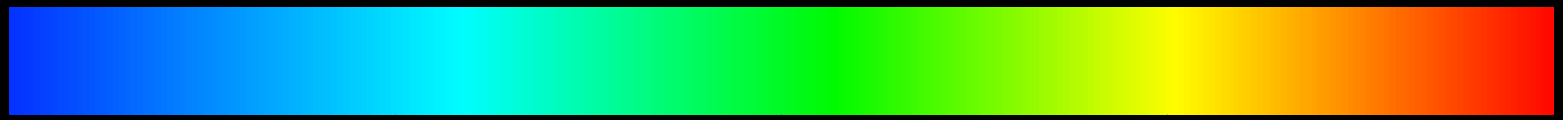


- A value representing a physical quantity (e.g. temperature) is attached to each cell
  - Normalized in [0.0..1.0]
- Cells are colored by interpolation using a gradient palette

Visualizing the values

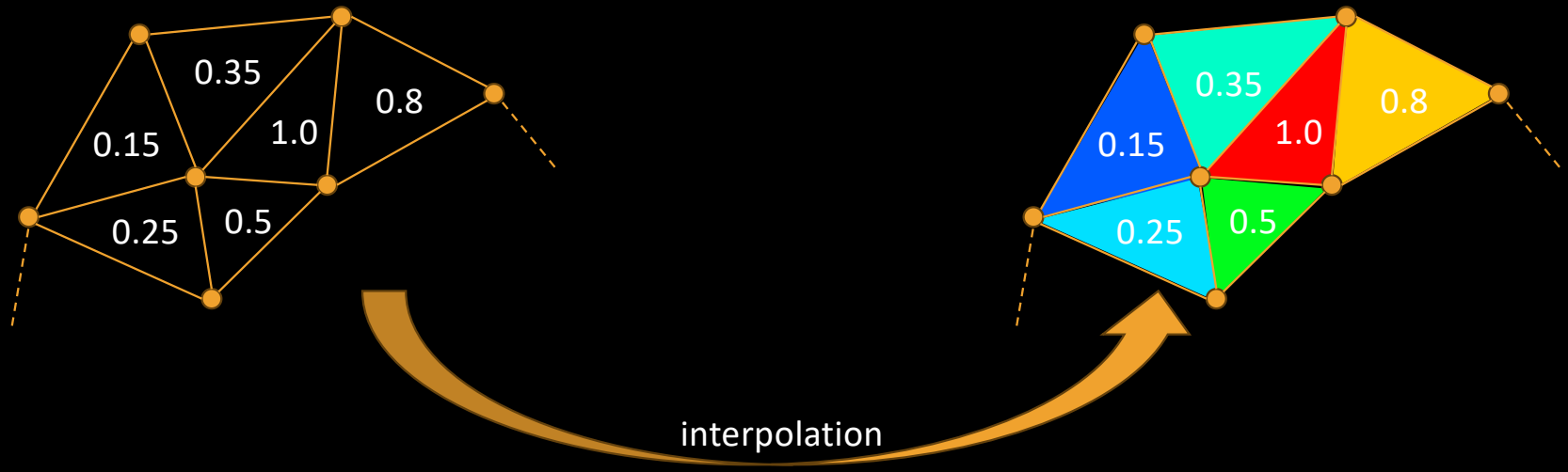


- A value representing a physical quantity (e.g. temperature) is attached to each cell
  - Normalized in [0.0..1.0]
- Cells are colored by interpolation using a gradient palette

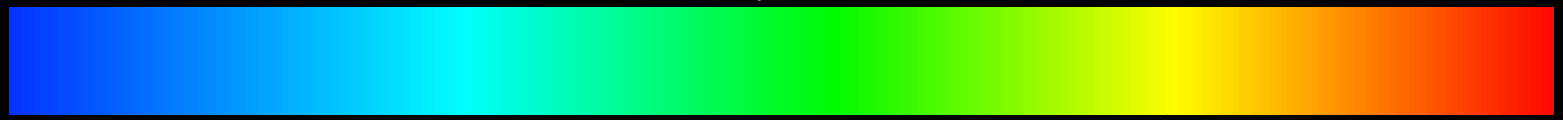


0.0 (blue) 0.25 (cyan) 0.5 (green) 0.75 (yellow) 1.0 (red)

Visualizing the values



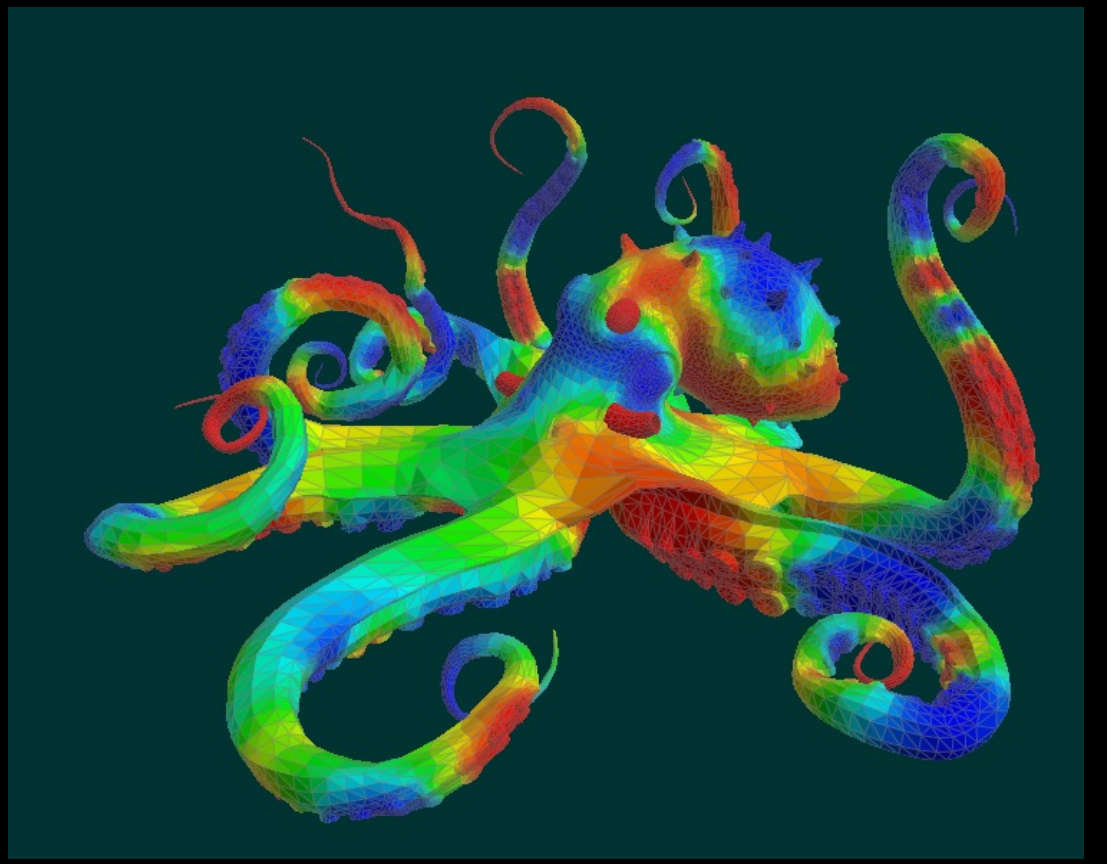
color palette



0.0	0.25	0.5	0.75	1.0
(blue)	(cyan)	(green)	(yellow)	(red)

Doing computations on unstructured meshes

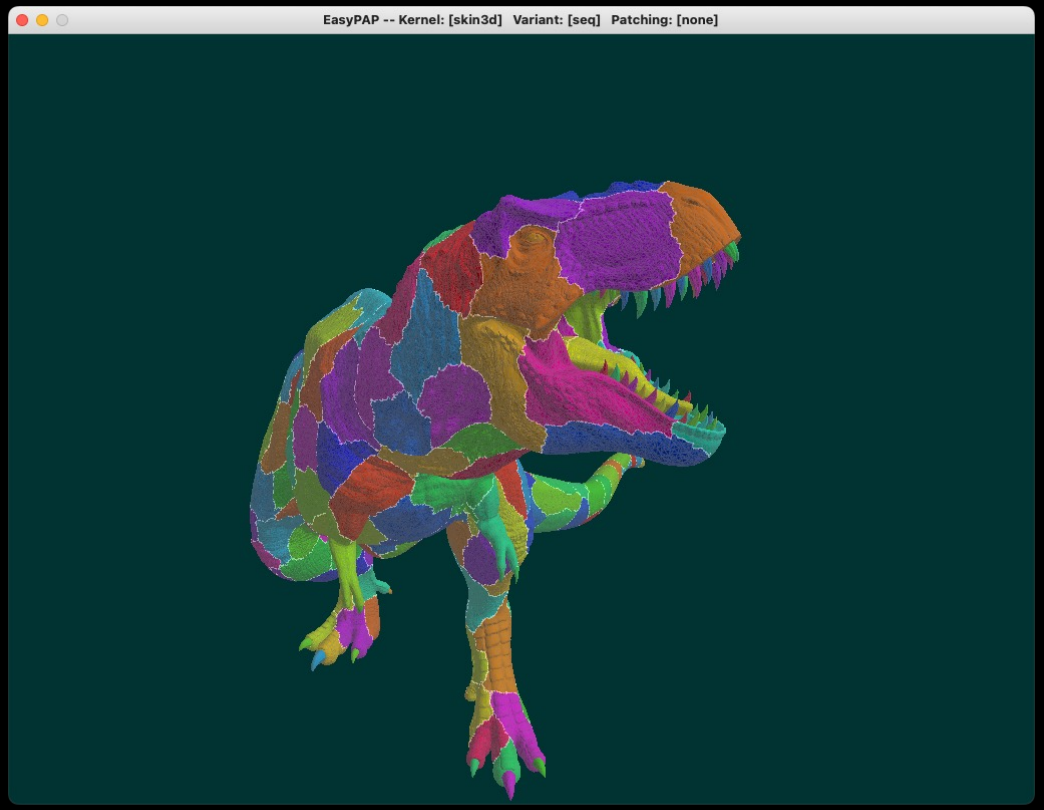
- A value representing a physical quantity (e.g. temperature) is attached to each cell
  - Normalized in [0.0..1.0]
- Cells are colored by interpolation using a gradient palette



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

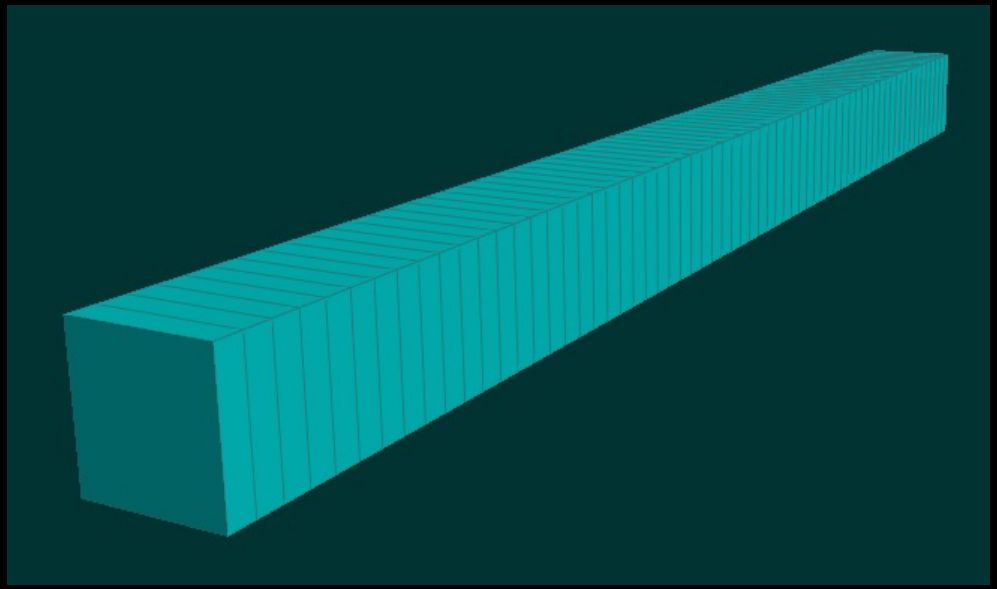


Setting a simple 2-point palette

```
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);
}
```

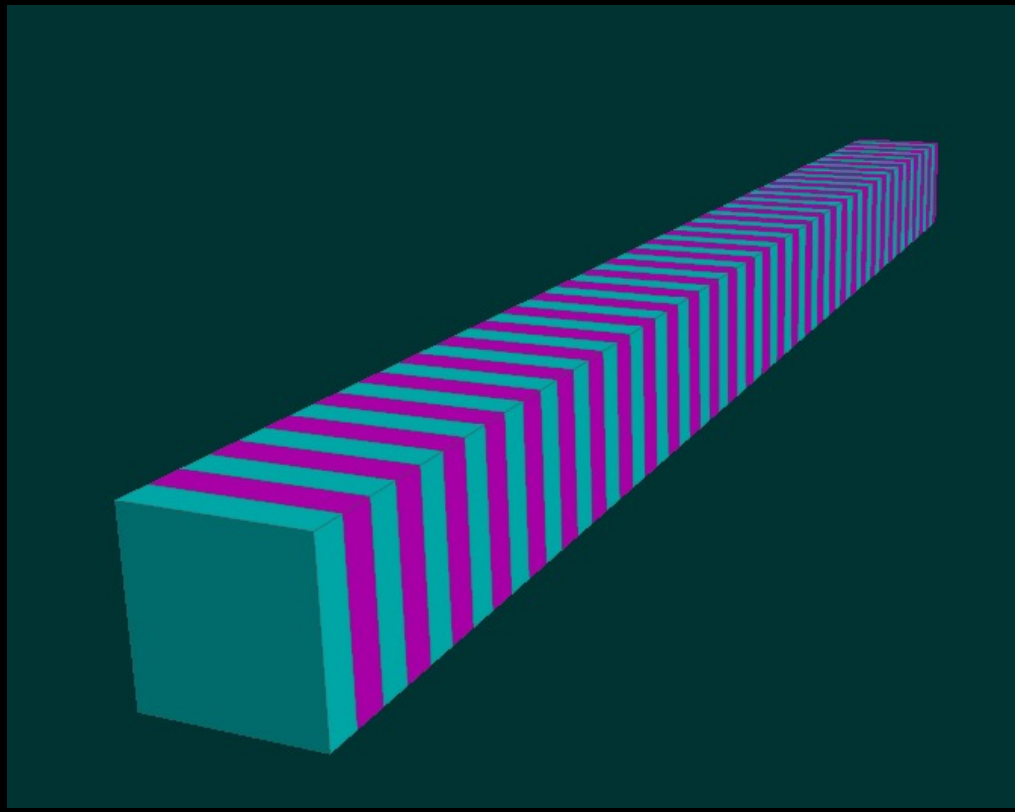
Example

```
///////////////////////////// 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) = 0.0;

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



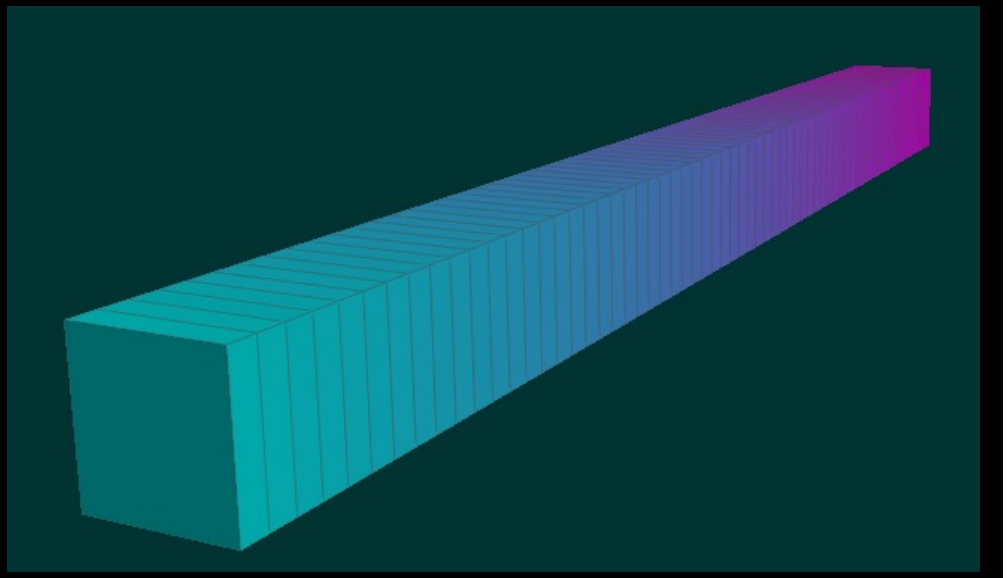
Example alternating 0 and 1

```
///////////////////////////// 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;
}
```

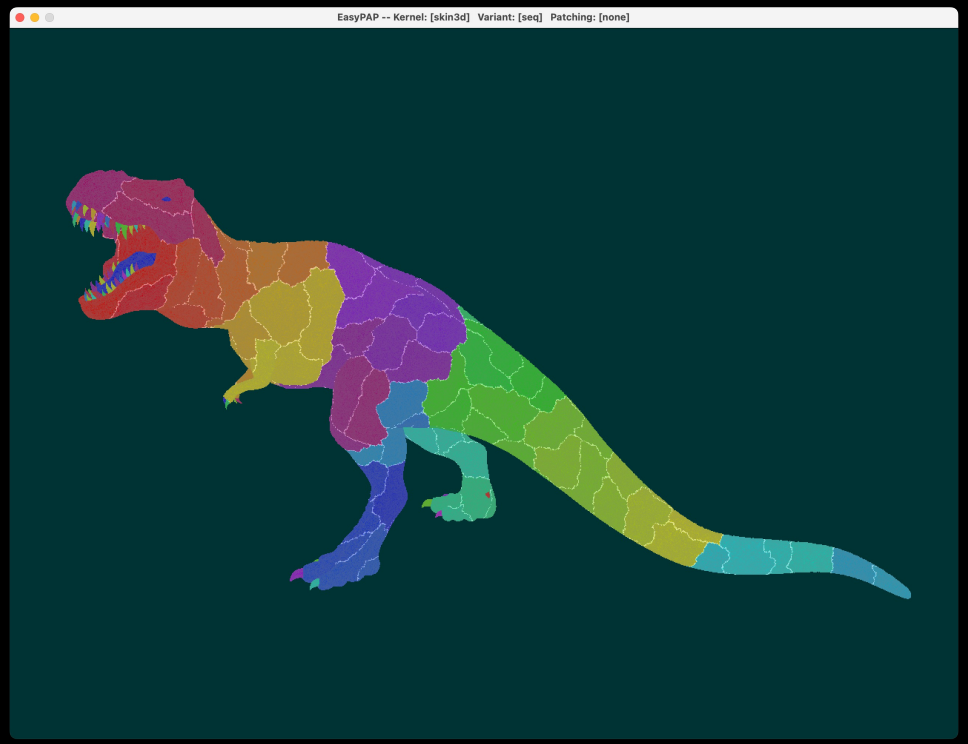


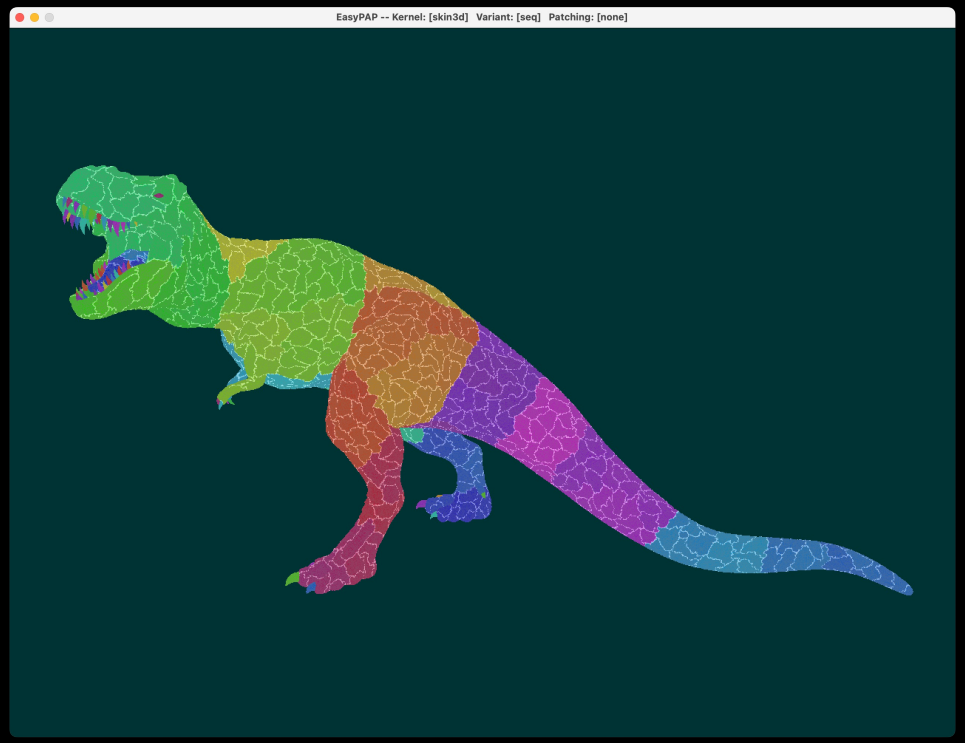
Example using the linear [0..1] spectrum

```
///////////////////////////// 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) = c / (float)(NB_CELLS-1) ;

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



Grouping cells in patches as a mean to control granularity of parallelism

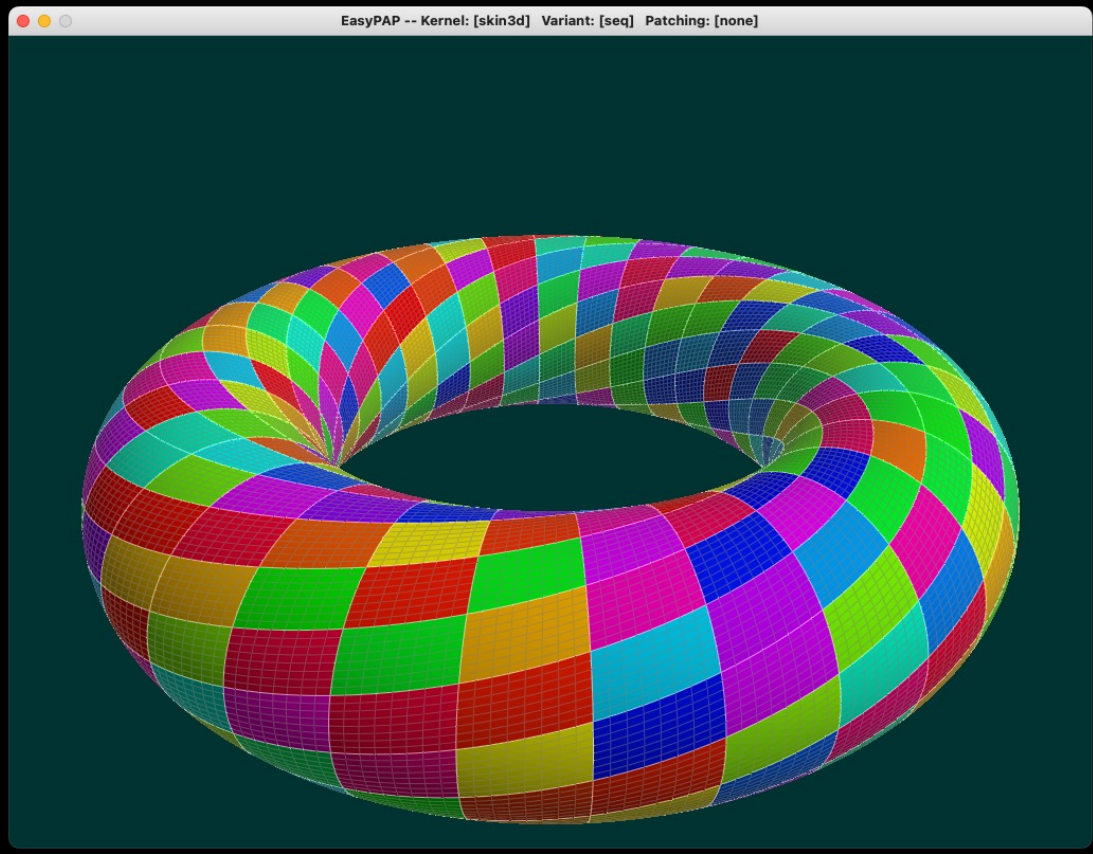




Where are my tiles?

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

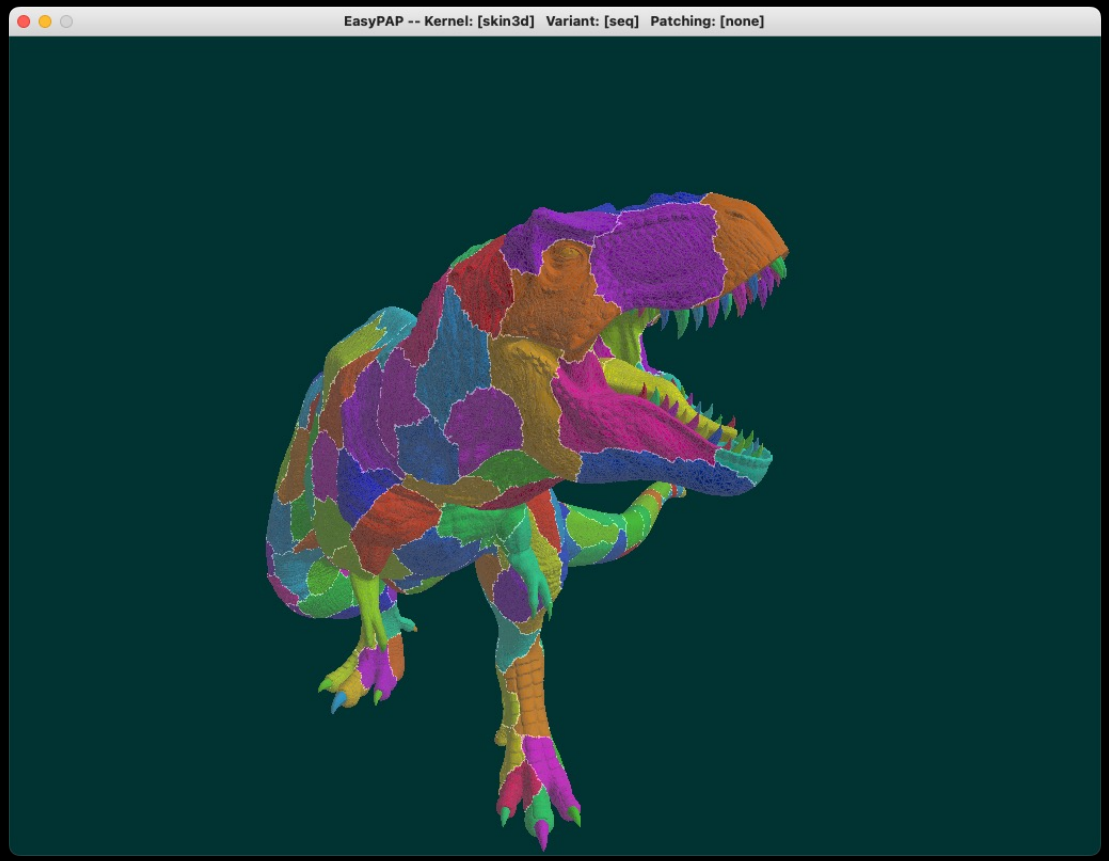
- Tiles → Patches
  - Obtained either by
    - Space filling curves
    - Scotch partitioning library



Where are my tiles?

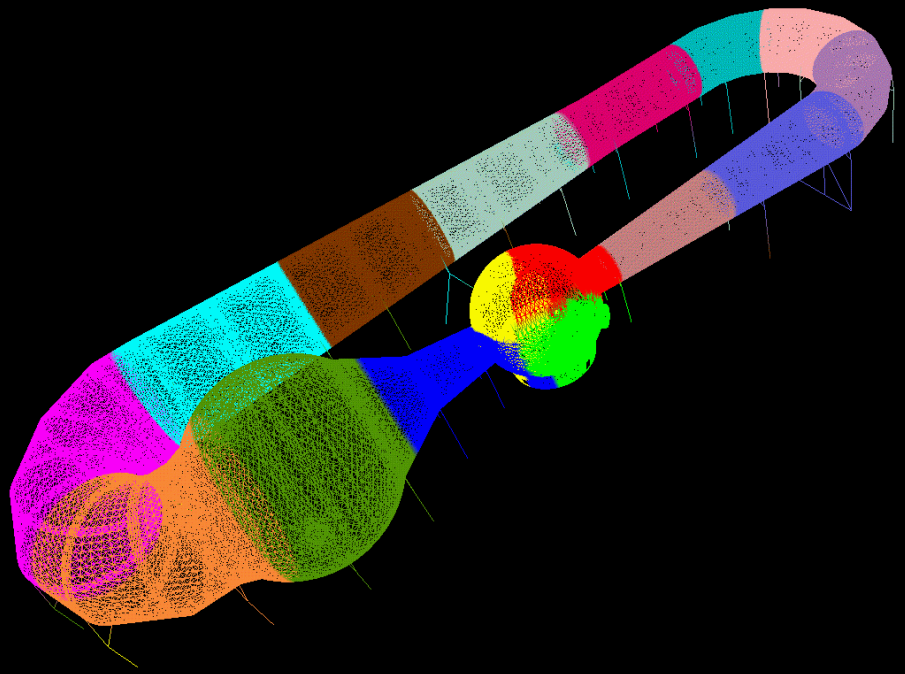
- Pixels → Cells
  - Variable number of neighbors
- Colors → float values

- 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
- Scotch helps workload distribution
- Scotch is designed for parallel computing environments
  - offers multiple efficient and flexible algorithms
  - supports multiple graph formats



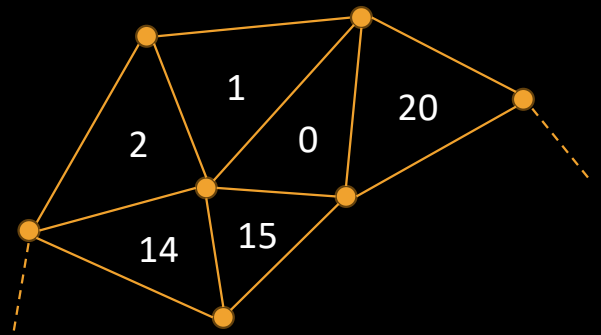
About partitions

- After the partitioning phase
  - Cells are sorted so that partitions are contiguous sets of cells numbers
- do_patch functions simply iterate on a subrange of cells

```
int sample3d_do_patch_default (int start_cell,
                               int end_cell)
{
  for (int c = start_cell; c < end_cell; c++)
    // Assign a value to each cell
    cur_data (c) = … ;

  return 0;
}

unsigned sample3d_compute_tiled (unsigned nb_iter)
{
  …
    for (int p = 0; p < NB_PATCHES; p++)
      do_patch (p);
  …
}
```

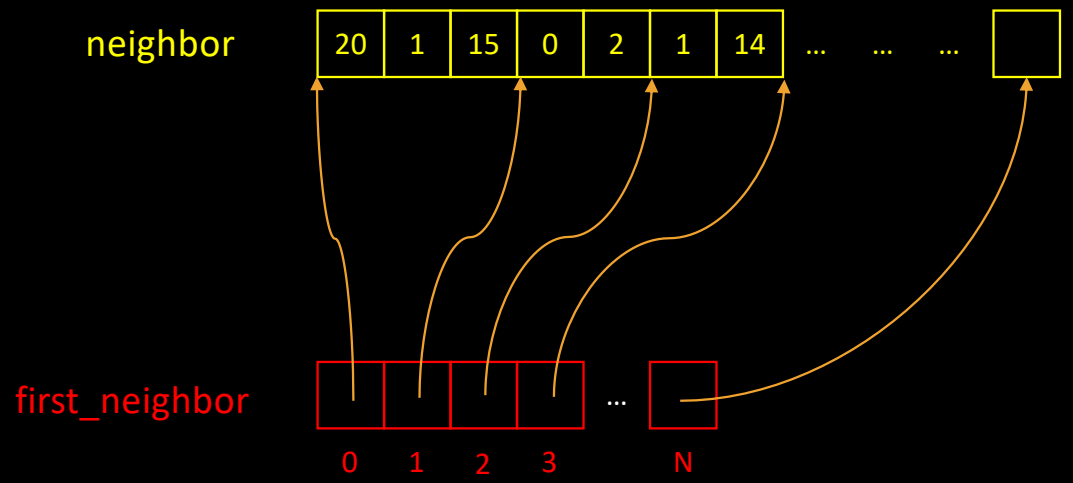
Accessing mesh connectivity

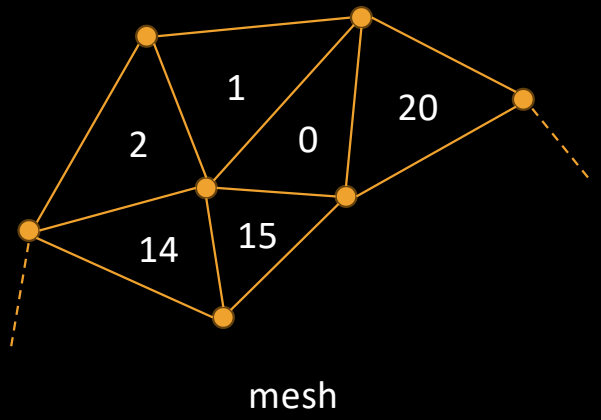
- In EasyPAP, the programmer sees
  - NB_CELLS
  - cur_data (cell) and next_data (cell)
- And
  - nb_neighbors (int cell)
  - nth_neighbor (int cell, int n)
  - max_neighbors ()
    - Max connectivity in the whole graph



mesh

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

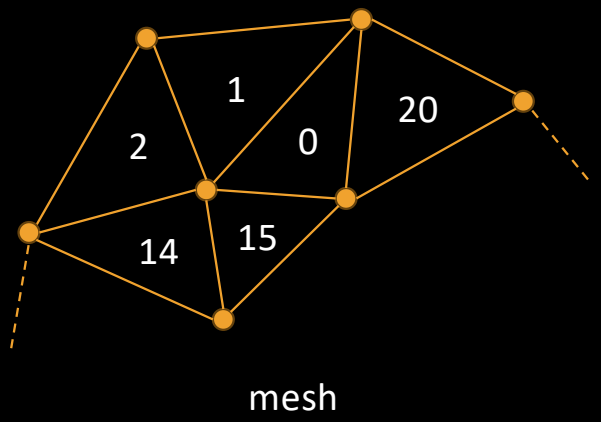




Neighbors of cell **i** are stored in: neighbors [first_neighbor[i] .. first_neightbor [i+1] – 1]

Mesh connectivity

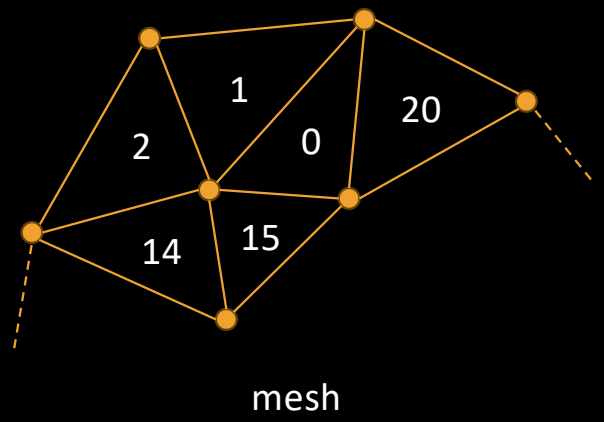
- Programmers can directly access the compact edge array
  - neighbor_start (int cell)
    - Index of first neighbor
  - neighbor_end (int cell)
    - Index of last neighbor + 1
  - neighbor (int index)



mesh

Accessing mesh connectivity in EasyPAP

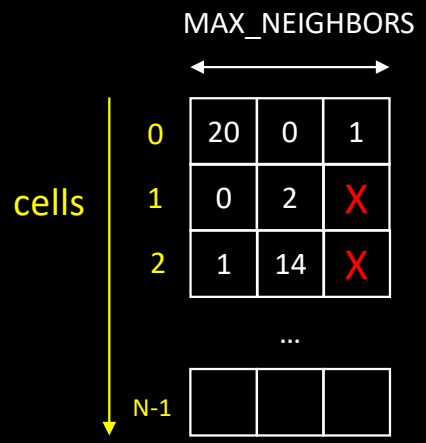
```
for (int nn = 0; nn < nb_neighbors (c); nn++) {
  int n = nth_neighbor (c, nn);
  float value = cur_data (n);
  …
}
// is equivalent to
for (int id = neighbor_start (c);
     id < neighbor_end (c); id++) {
  int n = neighbor (id);
  float value = cur_data (n);
  …
}
```

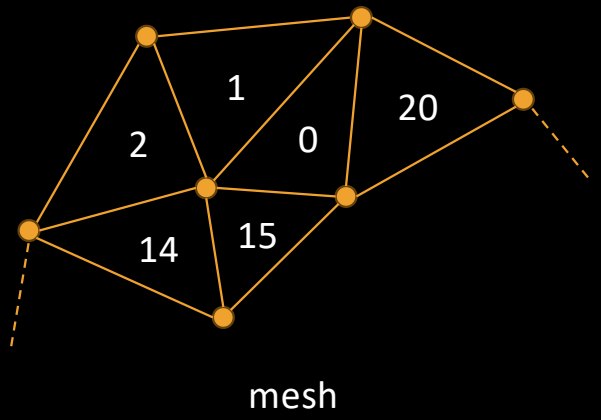


mesh

Towards a more regular data structure for GPUs and vector instructions

- Idea
  - We can waste a little space and use a 2D array!

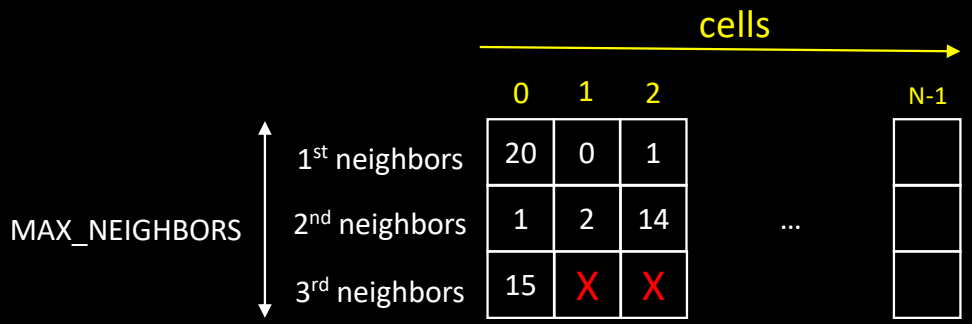


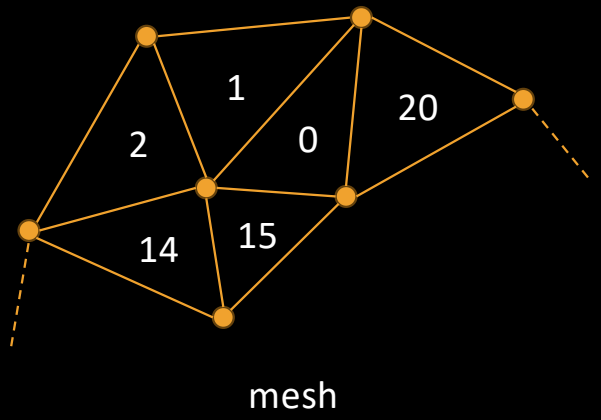


mesh

Towards a more regular data structure for GPUs and vector instructions

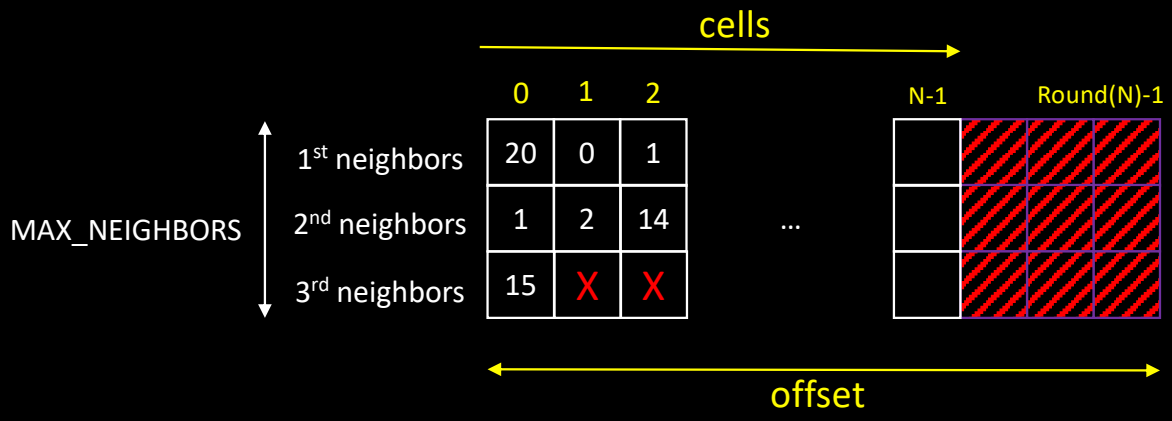
- Idea
  - Better use this data layout:

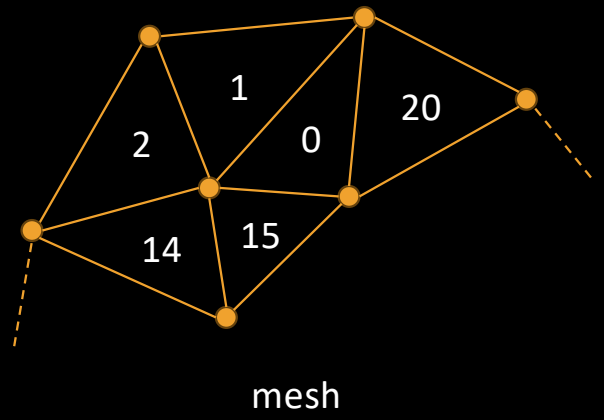




Towards a more regular data structure for GPUs and vector instructions

- Optimization
  - Use padding for alignment purposes





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

Illustration with the max3d OpenCL kernel

```
__kernel void max3d_ocl (__global float *in, __global float *out,
                         __global __read_only int *neighbor)
{
  const int index = get_global_id (0);
  if (index < NB_CELLS) {
    float m = in [index];

    for (int n = 0; n < MAX_NEIGHBORS; n++) {
      int addr = neighbor [n * get_global_size (0) + index]; // good, coalesced access
      if (addr != -1) {
        float v = in [addr]; // bad, arbitrary access
        m = max (m, v);
      }
    }
    out [index] = m;
  }
}
```

Illustration with the max3d CUDA kernel

```
static __device__ void max3d_cuda (float *in, float *out, int *neighbor_soa,
                                   unsigned nb_cells, unsigned max_neighbors)
{
   int index = blockIdx.x * blockDim.x + threadIdx.x;
   int offset = blockDim.x * gridDim.x;
   if (index < nb_cells) {
      float m = in[index];
      for (int n = 0; n < max_neighbors; n++) {
         int addr = neighbor_soa[n * offset + index]; // good, coalesced access
         if (addr != -1) {
            float v = in [addr]; // bad, arbitrary access
            m = max (m, v);
         }
      }
      out[index] = m;
   }
}
```

Additional resources
available on
http://gforgeron.gitlab.io/pap/