Scientific Computing WS 2018/2019

Lecture 8

Jürgen Fuhrmann

juergen.fuhrmann@wias-berlin.de

Matrix preconditioned Richardson iteration

M, A spd.

- Scaled Richardson iteration with preconditoner M

$$u_{k+1} = u_k - \alpha M^{-1}(Au_k - b)$$

- Spectral equivalence estimate

$$0 < \gamma_{min}(Mu, u) \leq (Au, u) \leq \gamma_{max}(Mu, u)$$

- $\Rightarrow \gamma_{min} \leq \lambda_i \leq \gamma_{max}$
- $\Rightarrow$ optimal parameter $\alpha = \frac{2}{\gamma_{max}+\gamma_{min}}$
- Convergence rate with optimal parameter: $\rho \leq \frac{\kappa(M^{-1}A)-1}{\kappa(M^{-1}A)+1}$
- This is one possible way for convergence analysis which at once gives convergence rates
- But ... how to obtain a good spectral estimate for a particular problem ?

Richardson for 1D heat conduction

- Regard the $n \times n$ 1D heat conduction matrix with $h = \frac{1}{n-1}$ and $\alpha = \frac{1}{h}$ (easier to analyze).

$$A = \begin{pmatrix} \frac{2}{h} & -\frac{1}{h} & & & & \\ -\frac{1}{h} & \frac{2}{h} & -\frac{1}{h} & & & \\ & -\frac{1}{h} & \frac{2}{h} & -\frac{1}{h} & & \\ & \ddots & \ddots & \ddots & \ddots & \\ & & -\frac{1}{h} & \frac{2}{h} & -\frac{1}{h} & \\ & & & -\frac{1}{h} & \frac{2}{h} & -\frac{1}{h} \\ & & & & -\frac{1}{h} & \frac{2}{h} \end{pmatrix}$$

- Eigenvalues (tri-diagonal Toeplitz matrix):

$$\lambda_i = \frac{2}{h}\left(1 + \cos\left(\frac{i\pi}{n+1}\right)\right) \quad (i = 1 \ldots n)$$

Source: A. Böttcher, S. Grudsky: Spectral Properties of Banded Toeplitz Matrices. SIAM,2005

- Express them in h: $n + 1 = \frac{1}{h} + 2 = \frac{1+2h}{h} \Rightarrow$

$$\lambda_i = \frac{2}{h}\left(1 + \cos\left(\frac{ih\pi}{1+2h}\right)\right) \quad (i = 1 \ldots n)$$

Richardson for 1D heat conduction: Jacobi

- The Jacobi preconditioner just multiplies by $\frac{h}{2}$, therefore for $M^{-1}A$:

$$\mu_{max} \approx 2 - \frac{\pi^2 h^2}{2(1+2h)^2}$$

$$\mu_{min} \approx \frac{\pi^2 h^2}{2(1+2h)^2}$$

- Optimal parameter: $\alpha = \frac{2}{\lambda_{max}+\lambda_{min}} \approx 1 \ (h \to 0)$
- Good news: this is independent of h resp. n
- No need for spectral estimate in order to work with optimal parameter
- Is this true beyond this special case ?

Richardson for 1D heat conduction: Convergence factor

- Condition number + spectral radius

$$\kappa(M^{-1}A) = \kappa(A) = \frac{4(1+2h)^2}{\pi^2 h^2} - 1$$

$$\rho(I - M^{-1}A) = \frac{\kappa - 1}{\kappa + 1} = 1 - \frac{\pi^2 h^2}{2(1+2h)^2}$$

- Bad news: $\rho \to 1 \quad (h \to 0)$
- Typical situation with second order PDEs:

$$\kappa(A) = O(h^{-2}) \quad (h \to 0)$$

$$\rho(I - D^{-1}A) = 1 - O(h^2) \quad (h \to 0)$$

- Mean square error of approximation $||u - u_h||_2 < h^\gamma$, in the simplest case $\gamma = 2$.

Iterative solver complexity I

- Solve linear system iteratively until $||e_k|| = ||(I - M^{-1}A)^k e_0|| \leq \epsilon$

$$\rho^k e_0 \leq \epsilon$$
$$k \ln \rho < \ln \epsilon - \ln e_0$$
$$k \geq k_\rho = \left\lceil \frac{\ln e_0 - \ln \epsilon}{\ln \rho} \right\rceil$$

- $\Rightarrow$ we need at least k_ρ iteration steps to reach accuracy ϵ
- Optimal iterative solver complexity - assume:
  - $\rho < \rho_0 < 1$ independent of h resp. N
  - A sparse ($A \cdot u$ has complexity $O(N)$)
  - Solution of $Mv = r$ has complexity $O(N)$.

  $\Rightarrow$ Number of iteration steps k_ρ independent of N
  $\Rightarrow$ Overall complexity $O(N)$

Iterative solver complexity II

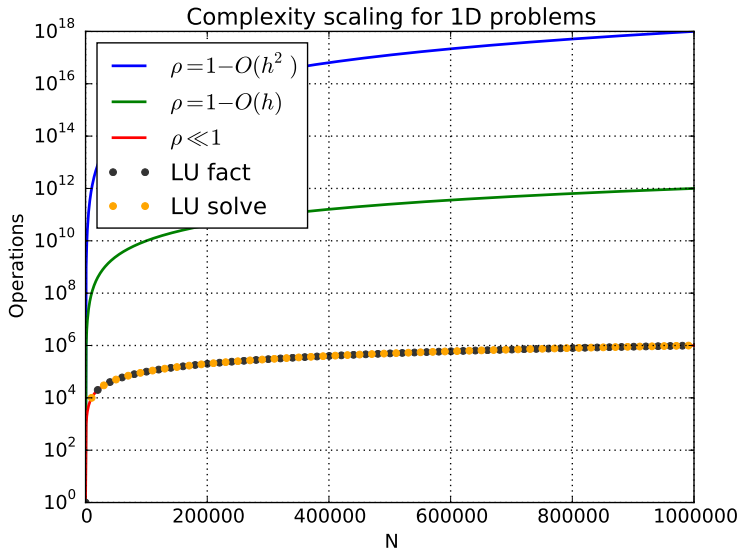
- Assume
  - $\rho = 1 - h^\delta \Rightarrow \ln \rho \approx -h^\delta \rightarrow k_\rho = O(h^{-\delta})$
  - d: space dimension $\Rightarrow h \approx N^{-\frac{1}{d}} \Rightarrow k_\rho = O(N^{\frac{\delta}{d}})$
  - $O(N)$ complexity of one iteration step (e.g. Jacobi, Gauss-Seidel)

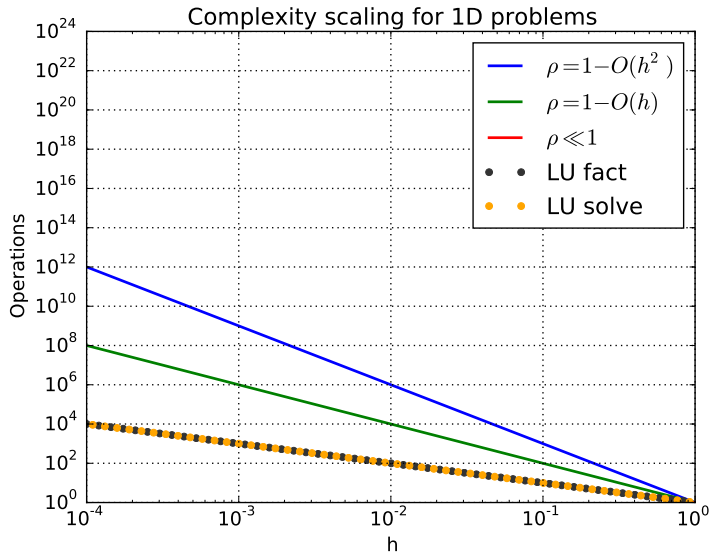
  $\Rightarrow$ Overall complexity $O(N^{1+\frac{\delta}{d}}) = O(N^{\frac{d+\delta}{d}})$
- Jacobi: $\delta = 2$
- Hypothetical "Improved iterative solver" with $\delta = 1$?
- Overview on complexity estimates

dim	$\rho = 1 - O(h^2)$	$\rho = 1 - O(h)$	LU fact.	LU solve
1	$O(N^3)$	$O(N^2)$	$O(N)$	$O(N)$
2	$O(N^2)$	$O(N^{\frac{3}{2}})$	$O(N^{\frac{3}{2}})$	$O(N \log N)$
3	$O(N^{\frac{5}{3}})$	$O(N^{\frac{4}{3}})$	$O(N^2)$	$O(N^{\frac{4}{3}})$

Solver complexity scaling for 1D problems

dim	$\rho = 1 - O(h^2)$	$\rho = 1 - O(h)$	LU fact.	LU solve
1	$O(N^3)$	$O(N^2)$	$O(N)$	$O(N)$

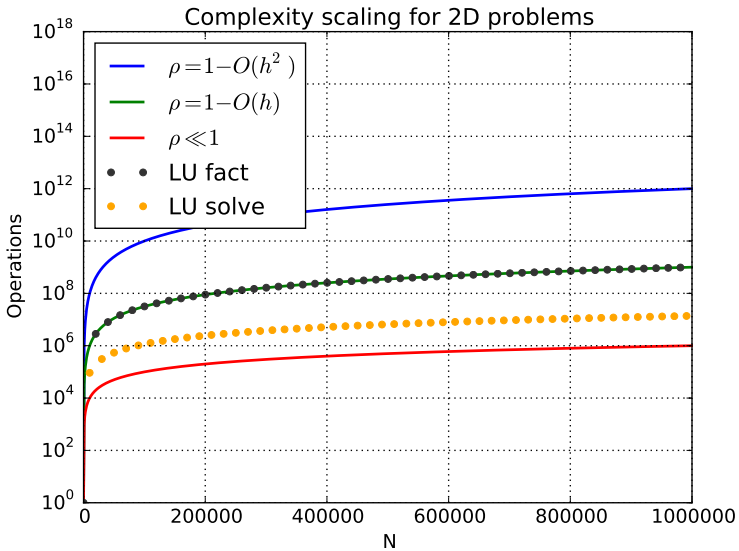


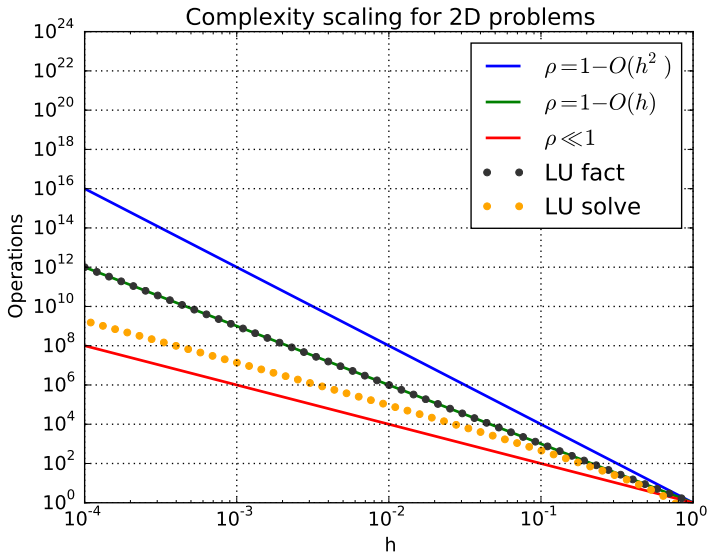


- Direct solvers significantly better than iterative ones

Solver complexity scaling for 2D problems

dim	$\rho = 1 - O(h^2)$	$\rho = 1 - O(h)$	LU fact.	LU solve
2	$O(N^2)$	$O(N^{\frac{3}{2}})$	$O(N^{\frac{3}{2}})$	$O(N \log N)$

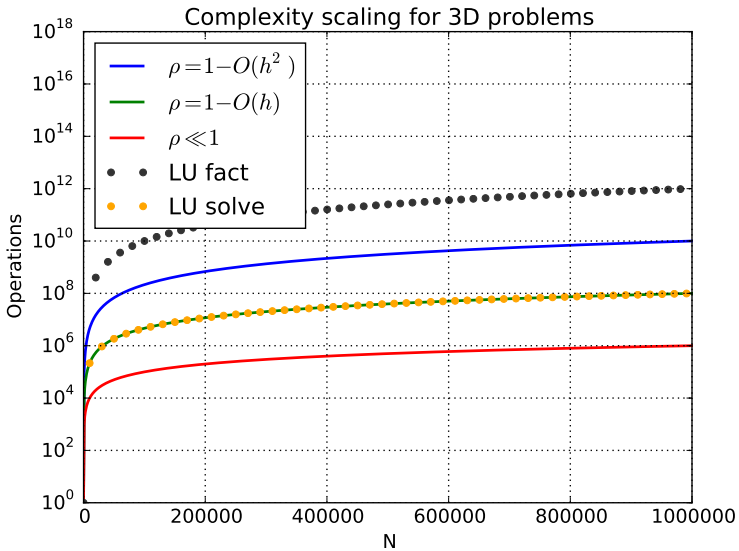


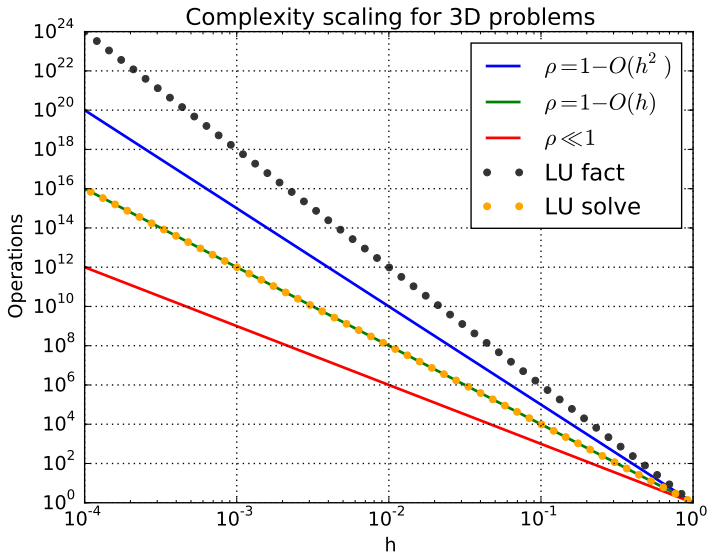


- Direct solvers better than simple iterative solvers (Jacobi etc.)
- On par with improved iterative solvers

Solver complexity scaling for 3D problems

dim	$\rho = 1 - O(h^2)$	$\rho = 1 - O(h)$	LU fact.	LU solve
3	$O(N^{\frac{5}{3}})$	$O(N^{\frac{4}{3}})$	$O(N^2)$	$O(N^{\frac{4}{3}})$





- LU factorization is extremly expensive
- LU solve on par with improved iterative solvers

What could be done ?

- Find optimal iterative solver with $O(N)$ complexity
- Find "improved preconditioner" with $\kappa(M^{-1}A) = O(h^{-1}) \Rightarrow \delta = 1$
- Find "improved iterative scheme": with $\rho = \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}$:

  For Jacobi, we had $\kappa = X^2 - 1$ where $X = \frac{2(1+2h)}{\pi h} = O(h^{-1})$.

$$
\begin{aligned}
\rho &= 1 + \frac{\sqrt{X^2-1}-1}{\sqrt{X^2-1}+1} - 1 \\
&= 1 + \frac{\sqrt{X^2-1}-1-\sqrt{X^2-1}-1}{\sqrt{X^2-1}+1} \\
&= 1 - \frac{1}{\sqrt{X^2-1}+1} \\
&= 1 - \frac{1}{X\left(\sqrt{1-\frac{1}{X^2}}+\frac{1}{X}\right)} \\
&= 1 - O(h)
\end{aligned}
$$

$\Rightarrow \delta = 1$

Solution of SPD system as a minimization procedure

Regard $Au = f$,where A is symmetric, positive definite. Then it defines a bilinear form $a : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$

$$a(u, v) = (Au, v) = v^T Au = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} v_i u_j$$

As A is SPD, for all $u \neq 0$ we have $(Au, u) > 0$.

For a given vector b, regard the function

$$f(u) = \frac{1}{2} a(u, u) - b^T u$$

What is the minimizer of f ?

$$f'(u) = Au - b = 0$$

- Solution of SPD system $\equiv$ minimization of f.

Method of steepest descent: iteration scheme

$$r_i = b - Au_i$$
$$\alpha_i = \frac{(r_i, r_i)}{(Ar_i, r_i)}$$
$$u_{i+1} = u_i + \alpha_i r_i$$

Let $\hat{u}$ the exact solution. Define $e_i = u_i - \hat{u}$, then $r_i = -Ae_i$

Let $||u||_A = (Au, u)^{\frac{1}{2}}$ be the *energy norm* wrt. A.

Theorem The convergence rate of the method is

$$||e_i||_A \leq \left(\frac{\kappa - 1}{\kappa + 1}\right)^i ||e_0||_A$$

where $\kappa = \frac{\lambda_{max}(A)}{\lambda_{min}(A)}$ is the spectral condition number.

Method of steepest descent: advantages

- Simple Richardson iteration $u_{k+1} = u_k - \alpha(Au_k - f)$ needs good eigenvalue estimate to be optimal with $\alpha = \frac{2}{\lambda_{max}+\lambda_{min}}$
- In this case, asymptotic convergence rate is $\rho = \frac{\kappa-1}{\kappa+1}$
- Steepest descent has the same rate without need for spectral estimate

Conjugate gradients IV - The algorithm

Given initial value u_0, spd matrix A, right hand side b.

$$
\begin{aligned}
d_0 &= r_0 = b - Au_0 \\
\alpha_i &= \frac{(r_i, r_i)}{(Ad_i, d_i)} \\
u_{i+1} &= u_i + \alpha_i d_i \\
r_{i+1} &= r_i - \alpha_i A d_i \\
\beta_{i+1} &= \frac{(r_{i+1}, r_{i+1})}{(r_i, r_i)} \\
d_{i+1} &= r_{i+1} + \beta_{i+1} d_i
\end{aligned}
$$

At the i-th step, the algorithm yields the element from $e_0 + \mathcal{K}_i$ with the minimum energy error.

Theorem The convergence rate of the method is

$$
||e_i||_A \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^i ||e_0||_A
$$

where $\kappa = \frac{\lambda_{max}(A)}{\lambda_{min}(A)}$ is the spectral condition number.

Preconditioned CG II

Assume $\tilde{r}_i = E^{-1} r_i$, $\tilde{d}_i = E^T d_i$, we get the equivalent algorithm

$$
\begin{aligned}
r_0 &= b - Au_0 \\
d_0 &= M^{-1} r_0 \\
\alpha_i &= \frac{(M^{-1} r_i, r_i)}{(Ad_i, d_i)} \\
u_{i+1} &= u_i + \alpha_i d_i \\
r_{i+1} &= r_i - \alpha_i A d_i \\
\beta_{i+1} &= \frac{(M^{-1} r_{i+1}, r_{i+1})}{(r_i, r_i)} \\
d_{i+1} &= M^{-1} r_{i+1} + \beta_{i+1} d_i
\end{aligned}
$$

It relies on the solution of the preconditioning system, the calculation of the matrix vector product and the calculation of the scalar product.

C++ implementation

```
template < class Matrix, class Vector, class Preconditioner, class Real >
int  CG(const Matrix &A, Vector &x, const Vector &b,
const Preconditioner &M, int &max_iter, Real &tol)
{ Real resid;
  Vector p, z, q;
  Vector alpha(1), beta(1), rho(1), rho_1(1);
  Real normb = norm(b);
  Vector r = b - A*x;
  if (normb == 0.0)   normb = 1;
  if ((resid = norm(r) / normb) <= tol) {
    tol = resid;
    max_iter = 0;
    return 0;
  }
  for (int i = 1; i <= max_iter; i++) {
    z = M.solve(r);
    rho(0) = dot(r, z);
    if (i == 1)
    p = z;
    else {
      beta(0) = rho(0) / rho_1(0);
      p = z + beta(0) * p;
    }
    q = A*p;
    alpha(0) = rho(0) / dot(p, q);
    x += alpha(0) * p;
    r -= alpha(0) * q;
    if ((resid = norm(r) / normb) <= tol) {
      tol = resid;
      max_iter = i;
      return 0;
    }
    rho_1(0) = rho(0);
  }
  tol = resid;   return 1;
}
```

C++ implementation II

- Available from http://www.netlib.org/templates/cpp//cg.h
- Slightly adapted for numcxx
- Available in numxx in the namespace netlib.

Next steps

- Put linear solution methods into our toolchest for solving PDE problems test them later in more interesting 2D situations
- Need more "tools":
  - visualization
  - triangulation of polygonal domains
  - finite element, finite volume discretization methods

Visualization in Scientific Computing

- Human perception much better adapted to visual representation than to numbers
- Visualization of computational results necessary for the development of understanding
- Basic needs: curve plots etc
  - python/matplotlib
- Advanced needs: Visualize discretization grids, geometry descriptions, solutions of PDEs
  - Visualization in Scientific Computing: paraview
  - Graphics hardware: GPU
  - How to program GPU: OpenGL
  - vtk

Python

- Scripting language with huge impact in Scientific Computing
- Open Source, exhaustive documentation online
  - https://docs.python.org/3/
  - https://www.python.org/about/gettingstarted/
- Possibility to call C/C++ code from python
  - Library API
  - swig - simple wrapper and interface generator (not only python)
  - pybind11 - C++11 specific
- Glue language for projects from different sources
- Huge number of libraries
- numpy/scipy
  - Array + linear algebra library implemented in C
- matplotlib: graphics library
  https://matplotlib.org/users/pyplot_tutorial.html

C++/matplotlib workflow

- Run C++ program
- Write data generated during computations to disk
- Use python/matplotlib for to visualize results
- Advantages:
  - Rich possibilities to create publication ready plots
  - Easy to handle installation (write your code, install python+matplotlib)
  - Python/numpy to postprocess calculated data
- Disadvantages
  - Long way to in-depth understanding of API
  - Slow for large datasets
  - Not immediately available for creating graphics directly from C++

Matplotlib: Alternative tools

- Similar workflow
  - gnuplot
  - Latex/tikz
- Call graphics from C++ ?
  - ???
  - Best shot: call C++ from python, return data directly to python
  - Send data to python through UNIX pipes
  - Link python interpreter into C++ code
- Faster graphics ?

Processing steps in visualization

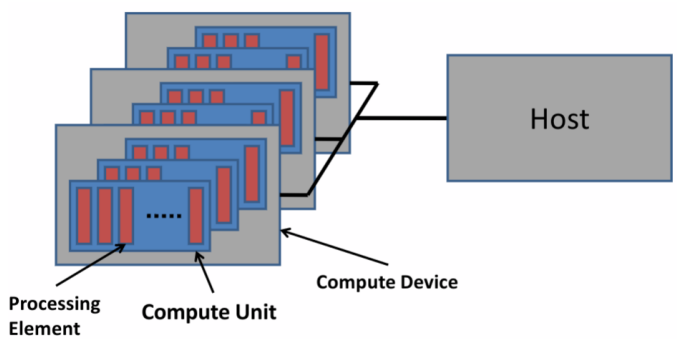
- Representation of data using elementary primitives: points, lines, triangles, ...
- Coordinate transformation form world coordinates to screen coordinates
- Transformation 3D $\rightarrow$ 2D - what is visible ?
- Rasterization: smooth data into pixels
- Coloring, lighting, transparency
- Similar tasks in CAD, gaming etc.
- Huge number of very similar operations

GPU aka “Graphics Card”

- SIMD parallelism “Single instruction, multiple data” inherent to processing steps in visualization
- Mostly `float` (32bit) accuracy is sufficient
- $\Rightarrow$ Create specialized coprocessors devoted to this task, free CPU from it
- Pionieering: Silicon Graphics (SGI)
- Today: nvidia, AMD
- Multiple parallel pipelines, fast memory for intermediate results

GPU Programming

- As GPU is a different processor, one needs to write extra programs to handle data on it – "shaders"
- Typical use:
  - Include shaders as strings in C++ (or load then from extra source file)
  - Compile shaders
  - Send compiled shaders to GPU
  - Send data to GPU – critical step for performance
  - Run shaders with data
- OpenGL, Vulkan



GPU Programming as it used to be

- Specify transformations
- Specify parameters for lighting etc.
- Specify points, lines etc. via API calls
- Graphics library sends data and manages processing on GPU
- No shaders - “fixed functions”
- Iris GL (SGI), OpenGL 1.x, now deprecated
- No simple, standardized API for 3D graphics with equivalent functionality
- Hunt for performance (gaming)

vtk

https://www.vtk.org/

- Visualization primitives in scientific computing
  - Datasets on rectangular and unstructured discretization grids
  - Scalar data
  - Vector data
- The *Visualization Toolkit* vtk provides an API with these primitives and uses up-to data graphics API (OpenGL) to render these data
- Well maintained, "working horse" in high performance computing
- Open Source
- Paraview, VisIt: GUI programs around vtk

Working with paraview

https://www.paraview.org/

- Write data into files using vtk specific data format
- Call paraview, load data

In-Situ visualization

- Using “paraview catalyst”
  - Send data via network from simulation server to desktop running paraview
- Call vtk API directly
  - vtkfig: small library for graphics primitives compatible with numcxx