Fast and smooth simulation of time dependent problems

Maciej Paszyński

Department of Computer Science
AGH University of Science and Technology, Kraków, Poland
home.agh.edu.pl/paszynsk
Outline

- Projection with isogeometric finite element method
- Generalization to time-dependent problems
- Explicit dynamics
- Example 1: Heat transfer
- Example 2: Non-linear flow in heterogeneous media
- Example 3: Tumor growth
- Implicit dynamics
- Example 4: Linear elasticity
- Example 5: Pollution problem
- Conclusions
Program Title: IGA-ADS
Code: git clone https://github.com/marcinlos/iga-ads
Licensing provisions: MIT license (MIT)
Programming language: C++
Nature of problem: Solving non-stationary problems in 1D, 2D and 3D
Solution method: Alternating direction solver with isogeometric finite element method

If you use this software in your work, please cite
Recursive definition of 1D B-splines


\[
N_{i,0}(\xi) = \begin{cases} 
1 & \text{if } \xi_i \leq \xi < \xi_{i+1}, \\
0 & \text{otherwise} 
\end{cases}
\]

\[
N_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi)
\]

Figure: Recursive formulae for B-spline basis functions and its explanation
Representation of B-splines by knot vectors


Figure: B-spline basis functions represented by knot vector 
\{0,0,0,1,2,3,4,4,5,5,5}\}

\[
\begin{align*}
&N_{1,2}, \quad N_{2,2}, \quad N_{3,2}, \quad N_{4,2}, \quad N_{5,2}, \quad N_{6,2}, \quad N_{7,2}, \quad N_{8,2} \\
&\begin{array}{c}
\text{0} \\
\text{1} \\
\text{2} \\
\text{3} \\
\text{4} \\
\text{5}
\end{array}
\end{align*}
\]
Tensor product definition of 2D B-spline basis functions

Figure: Tensor products B-splines basis functions

- 1D B-splines basis $B^x_1(x), \ldots, B^x_{N_x}(x), B^y_1(y), \ldots, B^y_{N_y}(y)$
- 2D B-splines basis $B_{i,j}(x, y) = B^x_i(x) \ast B^y_j(y)$
We want to approximate a \( \text{BITMAP}(x,y) \) with a linear combination of B-splines
\[
u(x, y) \approx \text{BITMAP}(x, y)
\]
where
\[
u(x, y) = \sum_{i,j} u_{i,j} B_i^x(x) B_j^y(y)
\]
How to construct a system of linear equations to get the coefficients \( u_{i,j} \)?
Projection with isogeometric finite element method

- We choose several “test functions” $v$, and use them to average the BITMAP at test functions supports
  \[ \int u(x, y)v(x, y)dx = \int \text{BITMAP}(x, y)v(x, y)dx \]
- Each selection of $v = B^x_i(x)B^y_j(y)$ leads to one equation
  \[ \int u(x, y)B^x_i(x) \ast B^y_j(y)dx = \int \text{BITMAP}(x, y)B^x_i(x)B^y_j(y)dx \]

Figure: One exemplary selection of $v = B^x_i(x)B^y_j(y)$
Projection with isogeometric finite element method

We end up with several equations
(one equation per one testing B-spline in 2D)

\[
\int u(x, y) B_1^x(x) B_1^y(y) dx = \int \text{BITMAP}(x, y) B_1^x(x) B_1^y(y) dx \\
\int u(x, y) B_1^x(x) B_2^y(y) dx = \int \text{BITMAP}(x, y) B_1^x(x) B_2^y(y) dx \\
\quad \vdots \\
\int u(x, y) B_k^x(x) B_l^y(y) dx = \int \text{BITMAP}(x, y) B_k^x(x) B_l^y(y) dx \\
\quad \vdots \\
\int u(x, y) B_{N_x}^x(x) B_{N_y-1}^y(y) dx = \\
\quad \int \text{BITMAP}(x, y) B_{N_x}^x(x) B_{N_y-1}^y(y) dx \\
\int u(x, y) B_{N_x}^x(x) B_{N_y}^y(y) dx = \\
\quad \int \text{BITMAP}(x, y) B_{N_x}^x(x) B_{N_y}^y(y) dx
\]
We approximate $u(x, y) \approx \sum_{i,j} u_{i,j} B_i^x(x) B_j^y(y)$

\[
\int u(x, y) B_1^x(x) B_1^y(y) \, dx = \int \text{BITMAP}(x, y) B_1^x(x) B_1^y(y) \, dx \\
\int u(x, y) B_1^x(x) B_2^y(y) \, dx = \int \text{BITMAP}(x, y) B_1^x(x) B_2^y(y) \, dx \\
\vdots \\
\int u(x, y) B_k^x(x) B_i^y(y) \, dx = \int \text{BITMAP}(x, y) B_k^x(x) B_i^y(y) \, dx \\
\vdots \\
\int u(x, y) B_{N_x}^x(x) B_{N_y-1}^y(y) \, dx = \\
\quad \int \text{BITMAP}(x, y) B_{N_x}^x(x) B_{N_y-1}^y(y) \, dx \\
\int u(x, y) B_{N_x}^x(x) B_{N_y}^y(y) \, dx = \\
\quad \int \text{BITMAP}(x, y) B_{N_x}^x(x) B_{N_y}^y(y) \, dx
\]
We approximate \( u(x, y) \approx \sum_{i,j} u_{i,j} B_i^x(x) B_j^y(y) \) to get

\[
\int \sum_{i,j} u_{i,j} B_i^x(x) B_j^y(y) B_1^x(x) * B_1^y(y) \, dx = \\
\int \text{BITMAP}(x, y) B_1^x(x) * B_1^y(y) \, dx
\]

\[
\int \sum_{i,j} u_{i,j} B_i^x(x) B_j^y(y) B_1^x(x) * B_2^y(y) \, dx = \\
\int \text{BITMAP}(x, y) B_1^x(x) * B_2^y(y) \, dx
\]

\[
\vdots
\]

\[
\int \sum_{i,j} u_{i,j} B_i^x(x) B_j^y(y) B_k^x(x) * B_1^y(y) \, dx = \\
\int \text{BITMAP}(x, y) B_k^x(x) * B_1^y(y) \, dx
\]

\[
\vdots
\]

\[
\int \sum_{i,j} u_{i,j} B_i^x(x) B_j^y(y) B_N^x(x) * B_{N_y-1}^y(y) \, dx = \\
\int \text{BITMAP}(x, y) B_N^x(x) * B_{N_y-1}^y(y) \, dx
\]

\[
\int \sum_{i,j} u_{i,j} B_i^x(x) B_j^y(y) B_{N_y}^x(x) * B_{N_y}^y(y) \, dx = \\
\int \text{BITMAP}(x, y) B_{N_y}^x(x) * B_{N_y}^y(y) \, dx
\]
Projection with isogeometric finite element method

We take the sum out
\[ \sum_{i,j} u_{i,j} \int B_i^x(x) B_j^y(y) B_k^x(x) \ast B_l^y(y) \, dx = \int \text{BITMAP}(x, y) B_k^x(x) \ast B_l^y(y) \, dx \]

and we end up with a system of linear equations

\[
\begin{bmatrix}
\int B_1^x B_1^y B_1^x B_1^y \, dx & \int B_1^x B_1^y B_2^x B_1^y \, dx & \cdots & \int B_1^x B_1^y B_{N_x}^x B_{N_y}^y \, dx \\
\int B_2^x B_1^y B_1^x B_1^y \, dx & \int B_2^x B_1^y B_2^x B_1^y \, dx & \cdots & \int B_2^x B_1^y B_{N_x}^x B_{N_y}^y \, dx \\
\vdots & \vdots & \ddots & \vdots \\
\int B_{N_x}^x B_{N_y}^y B_1^x B_1^y \, dx & \int B_{N_x}^x B_{N_y}^y B_2^x B_1^y \, dx & \cdots & \int B_{N_x}^x B_{N_y}^y B_{N_x}^x B_{N_y}^y \, dx
\end{bmatrix}
\begin{bmatrix}
u_{1,1} \\
u_{2,1} \\
\vdots \\
u_{N_x, N_y}
\end{bmatrix}
= \begin{bmatrix}
\int \text{BITMAP}(x, y) B_1^x(x) \ast B_1^y(y) \, dx \\
\int \text{BITMAP}(x, y) B_1^x(x) \ast B_2^y(y) \, dx \\
\vdots \\
\int \text{BITMAP}(x, y) B_{N_x}^x(x) \ast B_{N_y}^y(y) \, dx
\end{bmatrix}
Projection with isogeometric finite element method

\[
\begin{bmatrix}
\int B_{1,p}^x B_{1,p}^y B_{1,p}^x B_{1,p}^y \\
\int B_{2,p}^x B_{1,p}^y B_{1,p}^x B_{1,p}^y \\
\vdots \\
\int B_{N_x,p}^x B_{N_y,p}^y B_{1,p}^x B_{1,p}^y
\end{bmatrix}
\]

\[
\begin{bmatrix}
\int B_{1,p}^x B_{1,p}^y B_{1,p}^x B_{2,p}^y \\
\int B_{2,p}^x B_{2,p}^y B_{2,p}^x B_{1,p}^y \\
\vdots \\
\int B_{N_x,p}^x B_{N_y,p}^y B_{2,p}^x B_{1,p}^y
\end{bmatrix}
\]

\[
\cdots
\]

\[
\begin{bmatrix}
\int B_{1,p}^x B_{1,p}^y B_{N_x,p}^x B_{N_y,p}^y \\
\int B_{2,p}^x B_{1,p}^y B_{N_x,p}^x B_{N_y,p}^y \\
\vdots \\
\int B_{N_x,p}^x B_{N_y,p}^y B_{N_x,p}^x B_{N_y,p}^y
\end{bmatrix}
\]

**Figure:** The resulting system of linear equations can be solved with $O(N^{1.5})$ time in 2D, or $O(N^2)$ time in 3D
Fast isogeometric L2 projections


Projection problem matrix on 2D domain $\Omega = \Omega_x \times \Omega_y$:

$$M_{ijkl} = \int_{\Omega} B_{ij} B_{kl} \, d\Omega = \int_{\Omega} B^x_i(x) B^y_j(y) B^x_k(x) B^y_l(y) \, d\Omega = \int_{\Omega} (B_i B_k)(x) (B_j B_l)(y) \, d\Omega = \int_{\Omega_x} B_i B_k \, dx \int_{\Omega_y} B_j B_l \, dy = M_{ik}^{x} \cdot M_{jl}^{y}$$

$$M = M^{x} \otimes M^{y} \quad \text{(Kronecker product)}$$
Fast isogeometric L2 projections


**Two steps – solving systems with A and B in different directions**

\[
\begin{pmatrix}
A_{11} & A_{12} & \cdots & 0 \\
A_{21} & A_{22} & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & A_{nn}
\end{pmatrix}
\begin{pmatrix}
y_{11} \\
y_{12} \\
\vdots \\
y_{1n}
\end{pmatrix}
= 
\begin{pmatrix}
 b_{11} \\
 b_{12} \\
\vdots \\
 b_{1n}
\end{pmatrix}
\]

\[
\begin{pmatrix}
B_{11} & B_{12} & \cdots & 0 \\
B_{21} & B_{22} & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & B_{mm}
\end{pmatrix}
\begin{pmatrix}
x_{11} \\
x_{21} \\
\vdots \\
x_{m1}
\end{pmatrix}
= 
\begin{pmatrix}
y_{11} \\
y_{21} \\
\vdots \\
y_{m1}
\end{pmatrix}
\]

Two 1D problems with multiple RHS, linear cost $O(N)$

- $n \times n$ with $m$ right hand sides $\rightarrow O(n \times m) = O(N)$
- $m \times m$ with $n$ right hand sides $\rightarrow O(m \times n) = O(N)$
Fast isogeometric L2 projections


B-spline basis functions have **local support** (over \( p + 1 \) elements) \( \mathcal{M}^x, \mathcal{M}^y, \ldots \) – banded structure

\( \mathcal{M}_{ij}^x = 0 \iff |i - j| > p \)

Exemplary basis functions and matrix for cubics

\[
\begin{bmatrix}
(B_1, B_1)_{L^2} & (B_1, B_2)_{L^2} & (B_1, B_3)_{L^2} & (B_1, B_4)_{L^2} & 0 & 0 & \cdots & 0 \\
(B_2, B_1)_{L^2} & (B_2, B_2)_{L^2} & (B_2, B_3)_{L^2} & (B_2, B_4)_{L^2} & (B_2, B_5)_{L^2} & 0 & \cdots & 0 \\
(B_3, B_1)_{L^2} & (B_3, B_2)_{L^2} & (B_3, B_3)_{L^2} & (B_3, B_4)_{L^2} & (B_3, B_5)_{L^2} & (B_3, B_6)_{L^2} & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & (B_n, B_{n-3})_{L^2} & (B_n, B_{n-2})_{L^2} & (B_n, B_{n-1})_{L^2} & (B_n, B_n)_{L^2} & \vdots \\
\end{bmatrix}
\]

Multi-diagonal matrix can be factorized in linear \( \mathcal{O}(N) \) cost.
Generalization to time-dependent problems


**In general:** time dependent problem of the form

\[
\frac{\partial u(x, t)}{\partial t} - L(u(x, t)) = f(x, t)
\]

\(u(x, t)\) - the unknown field
(its meaning depends on the simulated physical phenomena)

\[
\frac{\partial u(x,t)}{\partial t}
\] - changes of the phenomena in time

\(L\) - the physics (well-posed linear spatial PDE)

\(f(x, t)\) - the forcing
with some initial state \(u_0(x, t)\) and boundary conditions
Generalization to time-dependent problems

We introduce the time steps

\[ u_0 \quad u_1 \quad \ldots \quad u_k \quad u_{k+1} \quad \ldots \quad u_N \]
\[ t_0 \quad t_1 \quad \ldots \quad t_k \quad t_{k+1} \quad \ldots \quad t_N \]

We approximate the time derivatives using two consecutive time steps

\[ \frac{\partial u}{\partial t} \approx \frac{u_{t+1} - u_t}{dt} \]

\[ \frac{u_{t+1} - u_t}{dt} - \mathcal{L}(u) = f \]

We also introduce the solution from the previous time step into the operator and the forcing (so-called explicit scheme)

\[ \frac{u_{t+1} - u_t}{dt} - \mathcal{L}(u_t) = f_t \]

\[ u_{t+1} = u_t + dt \ast \mathcal{L}(u_t) + f_t \]
Generalization to time-dependent problems

Sequence of projections: previous time step state + changes enforced by physics and forcing → next time step state

\[ u_{t+1} = u_t + dt \cdot \mathcal{L}(u_t) + f_t(x, t) \]
Applications to time-dependent problems (C++ code)

Examples
The “physics” operator $\mathcal{L} =$

- heat transfer problem $\mathcal{L}(u) = \Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$

- non-linear flow in heterogeneous media (oil extraction problem)
  $\mathcal{L}(u) = \nabla \cdot (\kappa (\mathbf{x}, u) \exp(\mu u) \nabla u) = \frac{\partial}{\partial x} \left( \kappa (\mathbf{x}, u) \exp(\mu u) \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left( \kappa (\mathbf{x}, u) \exp(\mu u) \frac{\partial u}{\partial y} \right)$

- propagation of the pollutant $\mathcal{L}(u) = \beta \cdot \nabla u - \nabla \cdot (K \nabla u) = \beta_x \frac{\partial u}{\partial x} + \beta_y \frac{\partial u}{\partial y} - \frac{\partial}{\partial x} \left( K_{11} \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left( K_{22} \frac{\partial u}{\partial y} \right)$

- propagation of elastic waves ($\mathcal{L}=$complicated)

- tumor growth ($\mathcal{L}=$very complicated)
Example 1: Heat transfer equation

The “physics” operator $\mathcal{L}(u_t) = \Delta u_t = \frac{\partial^2 u_t}{\partial x^2} + \frac{\partial^2 u_t}{\partial y^2}$

The forcing $f_t(x, t) = 0$

The initial state = ball of heat in the center of the domain

5000 time steps, time step size $dt = 10^{-7}$
Example 1: Heat transfer equation
Code for Example 1 (Heat transfer equation)

"problems/heat/heat_2d.cpp"

#include "problems/heat/heat_2d.hpp"
using namespace ads;
using namespace ads::problems;
pilot for the simulation
int main() {
  quadratic B-splines, 12 elements along axis
  dim_config dim{ 2, 12 };  
  5000 time steps, time step size $10^{-7}$
  timesteps_config steps{ 5000, 1e-7 };  
  we will need to compute first derivatives during the computations
  int ders = 1;
  some auxiliary objects for configuration and simulation
  config_2d c{dim, dim, steps, ders};
  heat_2d sim{c};
  run the simulation
  sim.run();
}
Implementation


executed once before the simulation starts
void before() override

executed after every simulation step
void after_step() override

executed once after the simulation ends
void after() override

t_0  t_1  ....  t_k  ....  t_N

step() compute_rhs() step() compute_rhs() step() compute_rhs() step() compute_rhs()

executed before every simulation step
void before_step() override

implementation of the simulation step
void step() override

implementation of generation of RHS
void compute_rhs() override
Code for Example 1 (Heat transfer equation)

"problems/heat/heat_2d.hpp"
#include "ads/simulation.hpp"
using namespace ads;
using namespace problems;
class heat_2d : public simulation_2d {
  ...
  implementation of the initial state
  double init_state(double x, double y, double z)
  executed once before the simulation starts
  void before() override
  executed before every simulation step
  void before_step() override
  implementation of the simulation step
  void step() override
  executed after every simulation step
  void after_step() override
  implementation of generation of RHS
  void compute_rhs() override
  executed once after the simulation ends
  void after() override
"problems/heat/heat_2d.hpp"

this function is called from before at the beginning of the simulation
the function returns the value of \( u_0 = u(x, y)|_{t=0} \)
computed at point \((x, y)\)

```cpp
double init_state(double x, double y) {
    double dx = x - 0.5;
    double dy = y - 0.5;
    double r2 = std::min(8*(dx*dx+dy*dy),1.0);
    return (r2 - 1) * (r2 - 1) * (r2 + 1) * (r2 + 1);
}
```
Example 1: Heat transfer equation

\[ \int_{\Omega} u_{t+1} v = \int_{\Omega} u_t v - dt \ast \int_{\Omega} \Delta u_t v \]

Integration by parts \(- \int_{\Omega} \Delta u_t v = \int_{\Omega} \nabla u_t \cdot \nabla v + \int_{\Gamma} \frac{\partial u}{\partial n} v ds\)

and incorporation of boundary condition \(\frac{\partial u}{\partial n} = 0\)

\[ \int_{\Omega} u_{t+1} v = \int_{\Omega} u_t v - dt \ast (\nabla u_t \cdot \nabla v) \]

value of test function \(a\) over element \(e\) at Gauss point \(q\)

\[
\text{value_type v = eval_basis(e, q, a);}
\]

value of \(u_t\) at Gauss point

\[
\text{value_type u = eval_fun(u_prev, e, q);}
\]

computations of double gradient

\[
\text{double gradient = u.dx*v.dx+u.dy*v.dy;}
\]

\(RHS = u_t - dt \nabla u_t \cdot \nabla v\)

\[
\text{double val = u.val*v.val - steps.dt * gradient;}
\]

scale by Jacobian and weight

\[
\text{rhs(a[0],a[1])+=val*w*J;}
\]
void compute_rhs() {
    auto& rhs = u; zero(rhs);
    for (auto e : elements()) {
        loop through elements
        double J = jacobian(e); compute Jacobian
        for (auto q:quad_points()){" loop through Gauss points
            double w = weight(q); Gauss weight
            for (auto a : dofs_on_element(e)){loop through dofs
                value of basis function $q$ over element $e$ at Gauss point $q$
                value_type v = eval_basis(e, q, a);
                value of $u_t$ at Gauss point
                this also computes derivatives and stored at *dx
                value_type u = eval_fun(u_prev, e, q);
                computations of double gradient
                double gradient = u.dx*v.dx+u.dy*v.dy;
                $RHS = u_t - dt\nabla u \cdot \nabla v$
                double val = u.val*v.val - steps.dt * gradient;
                scale by Jacobian and weight
                rhs(a[0],a[1])+=val*w*J;
            }
        }
    }
}
Example 2: Non-linear flow in heterogeneous media

Hydraulic fracturing - oil/gas extraction technique consisting in high-pressure fluid injection into the deposit

Example 2: Non-linear flow in heterogeneous media

\[ u_{t+1} = u_t + dt \mathcal{L}(u_t) + f_t(x, t) \]

\[ \mathcal{L}(u) = \nabla \cdot (\kappa(x, u) \exp(\mu u) \nabla u) = \frac{\partial}{\partial x} (\kappa(x, u) \exp(\mu u) \frac{\partial u}{\partial x}) + \frac{\partial}{\partial y} (\kappa(x, u) \exp(\mu u) \frac{\partial u}{\partial y}) \]

\( u \) represents pressure

The “physics” operator \( \mathcal{L}(u) = \nabla \cdot (\kappa(x, u) \exp(\mu u) \nabla u) \)

**Figure:** \( \kappa \) property of the terrain (permeability), \( \mu = 10 \)
Example 2: Non-linear flow in heterogenous media

Forcing $f$ represents **pumps** $P$ and **sinks** $S$, increasing and decreasing the pressure locally at $x_p \in P$, $x_s \in S$ locations:

$$f(x, t) = \sum_{p \in P} \phi (\|x_p - x\|) - \sum_{s \in S} u(x, t)\phi (\|x_s - x\|)$$

$$\phi(t) = \begin{cases} 
(t - 1)^2 & \text{for } t \leq r \\
(t + 1)^2 & \text{for } t > r \\
0 & \text{for } t \leq r
\end{cases}$$
Example 2: Non-linear flow in heterogeneous media
Code for Example 2 (Non-linear flow)

"problems/flow/flow.cpp"

#include "problems/flow/flow.cpp"
using namespace ads;
using namespace ads::problems;
pilot for the simulation
int main() {
    quadratic B-splines, 20 elements along axis
    dim_config dim{ 2, 20 };
10000 time steps, time step size 10^{-7}
    timesteps_config steps{ 10000, 1e-7 };
we will need to compute first derivatives during the computations
    int ders = 1;
some auxiliary objects for configuration and simulation
config_3d c{dim, dim, dim, steps, ders};
flow_3d sim{c};
run the simulation
sim.run();
}
"problems/flow/geometry.hpp"

this function is called from the before at the beginning of the simulation
the function returns the value of $u_0 = u(x, y, z)|_{t=0}$ computed at $(x, y, z)$

double init_state(double x, double y, double z) {
    double r = 0.1;
    double R = 0.5;
    return ads::bump(r, R, x, y, z);
}

// $r < R$ in $[0, 1]$
inline double bump(double r, double R, double x, double y, double z) {
    double dx = x - 0.5, dy = y - 0.5, dz = z - 0.5;
    double t = std::sqrt(dx * dx + dy * dy + dz * dz);
    return falloff(r / 2, R / 2, t);
}

this function is called from the compute_rhs
inline double falloff(double r, double R, double t) {
    if (t < r) return 1.0;
    if (t > R) return 0.0;
    double h = (t - r) / (R - r);
    return std::pow((h - 1) * (h + 1), 2);
"problems/flow/flow.hpp"

this functions is also called from the before at the beginning of the simulation

```cpp
void fill_permmeability_map() {
    for(auto e:elements()) {
        for(auto q:quad_points()){
            auto x = point(e, q);
            kq(e[0], e[1], e[2], q[0], q[1], q[2]) =
                env.permeability(x[0], x[1], x[2]);
        }
    }
}
```

```cpp
double permeability(index_type e, index_type q) const {
    return kq(e[0], e[1], e[2], q[0], q[1], q[2]);
}
```
double **forcing**(point_type x, double /*t*/) const {
    using std::sin;
    double pi4 = 4 * M_PI;
    four pumps one sink
    double dx1=x-0.25; dy1=y-0.25; dz1=z-0.25;
    double dx2=x-0.75; dy2=y-0.25; dz2=z-0.25;
    double dx3=x-0.25; dy3=y-0.75; dz3=z-0.25;
    double dy4=x-0.25; dy4=y-0.25; dz4=z-0.75;
    double dz5=x-0.5; dz5=y-0.5; dz5=z-0.5;
    return
        max(0,sin(pi4*dx1)*sin(pi4*dy1)*sin(pi4*dz1))
        +max(0,sin(pi4*dx2)*sin(pi4*dy2)*sin(pi4*dz2))
        +max(0,sin(pi4*dx3)*sin(pi4*dy3)*sin(pi4*dz3))
        +max(0,sin(pi4*dx4)*sin(pi4*dy4)*sin(pi4*dz4))
        -max(0,sin(pi4*dx5)*sin(pi4*dy5)*sin(pi4*dz5));
}
\[ \int_\Omega u_{t+1} v = \int_\Omega (u_t + f) v + dt \int_\Omega \nabla \cdot (\kappa(x, u) \exp(\mu u) \nabla u) v \]

Integration by parts \[ \int_\Omega \nabla \cdot (\kappa(x, u) \exp(\mu u) \nabla u) v = \int_\Omega (\kappa e^{10*u_t} \nabla u_t \cdot \nabla v) + \int_\Gamma \kappa e^{10*u_t} \frac{\partial u_t}{\partial n} v \]

and using Neumann b.c.

\[ \int_\Omega u_{t+1} v = \int_\Omega (u_t + f) v - dt \int_\Omega (K_q(x) e^{10*u_t} \nabla u_t \cdot \nabla v) \]

\( u_t \) value over element \( e \) at Gauss point \( q \)

value_type \( u = \text{eval\_fun}(u_{\text{prev}}, e, q); \)

the forcing is based on the location of pumps and sinks

\( h = \text{forcing}(x, t); \)

value of test function \( a \) over element \( e \) at Gauss point \( q \)

value_type \( v = \text{eval\_basis}(e, q, a); \)

\[ (-K_q(x) e^{10*u_t}, \nabla u_t)_{L^2} + (h, \nabla v)_{L^2} \]

val= -k*std::exp(10*u.val)*grad_dot(u, v)+h*v.val;

\[ (u_t + h, w)_{L^2} - dt \ast (\kappa e^{10*u_t} \nabla u_t, \nabla w)_{L^2} \ast \text{Jacobian} \ast \text{weight} \]

\( U(aa[0], aa[1], aa[2])+=(u.val*v.val+steps.dt*val)*w*J; \)
parallel processing of loop through elements
executor.for_each(elements(), [\&](index_type e) {
  double J = jacobian(e);
  for (auto q : quad_points()) {
    Gauss points
    double w = weight(q); auto x = point(e, q);
    value of permeability at Gauss point
    double k = permeability(e, q);
    \( u_t \) value over element \( e \) at Gauss point \( q \)
    value_type u = eval_fun(u_prev, e, q);
    the forcing is based on the location of pumps and sinks
    double h = forcing(x, t);
    for (auto a : dofs_on_element(e)) { test functions
      remapping local to global index for aggregation of RHS
      auto aa = dof_global_to_local(e, a);
      value of test function \( a \) over element \( e \) at Gauss point \( q \)
      value_type v = eval_basis(e, q, a);
      double val = - k*std::exp(10*u.val)*grad_dot(u, v)+h*v.val;
      U(aa[0],aa[1],aa[2])+=(u.val*v.val+steps.dt*val)*w*J;
    }
  }

  // the update of RHS must be synchronized when processed in parallel
  executor.synchronized( [ & ] { update_global_rhs(rhs, U, e); });
});
Example 3: Melanoma growth model
Example 3: Tumor Angiogenic Factor (TAF) c

- produced by oxygen-starved tumor cells
- signal to the vasculature – „more oxygen is needed here”
- influences vasculature evolution (discrete model)
Example 3: Tumor Angiogenic Factor (TAF) 

\[ \frac{\partial c}{\partial t} = \chi_c \Delta c - \gamma_c o c + c^+ \]

- \( \chi_c = 0.01 \) TAF diffusion rate, \( \gamma_c = 0.3 \) TAF decay rate
- \( c^+ \) - TAF “source” \( c^+ = b(1 - c) \) for \( o < o^{death} \)
- \( o^{death} = 0.01 \) TAF hypoxia rate
Example 3: Density of normal extra-cellular matrix (ECM)

- provides support for the cell structures

\[
\frac{\partial M}{\partial t} = -\beta_M Mb
\]

- \( \beta_M = 1.0 \) – ECM decay rate
Example 3: Density of degraded extra-cellular matrix \( A \)

\[
\frac{\partial A}{\partial t} = \gamma_A M b + \chi_{aA} \Delta A - \gamma_{oA} A
\]

- \( \gamma_A = 0.5 \) – production rate of artefacts
- \( \chi_{aA} = 0.01 \) – diffusion rate of degraded extra-cellular matrix
- \( \gamma_{oA} = 0.01 \) – decay rate of degraded extra-cellular matrix
Example 3: Tumor cell pressure

- $P$ – tumor pressure, present for tumor cell density exceeding $b^N$

\[
P = \begin{cases} 
0 & \text{for } b < b^N \\
\frac{b - b^N}{b^M - b^N} & \text{for } b^N \leq b \leq b^M
\end{cases}
\]
Example 3: Tumor cell flux

- $J$ – induced by pressure of tumor and extracellular matrix

\[ J = -D_b \cdot b (\nabla P + r_b \nabla A) \]

- $D_b$ – cell diffusion coefficient
- $r_b = 0.0001$ – tumor cell chemoattractant sensitivity
Example 3: Tumor cell density

- values between $b^m = 0$ (no cancer cells) and $b^M = 2$
- $b^N = 1$ – *normal* tumor cell density
  \[
  \frac{\partial b}{\partial t} = -\nabla \cdot J + b^- + b^+
  \]
- $b^+, b^-$ – tumor cell proliferation and apoptosis factors
Example 3: Tumor cell proliferation/death

\( b^+, b^- \) – governed by the oxygen concentration \( o \)

- \( o > o^{prol} = 0.1 \) – tumor cells multiply (\( b^+ > 0 \))
- \( o < o^{death} = 0.01 \) – tumor cells die (\( b^- > 0 \))

\[
\begin{align*}
    b^+ &= \frac{b}{T^{prol}} \left( 1 + \frac{\tau_b A}{\tau_b A + 1} P_b \right) \left( 1 - \frac{b}{b^M} \right) \quad \text{for } o > o^{prol} \\
    b^- &= -\frac{b}{T^{death}} \quad \text{for } o < o^{death}
\end{align*}
\]

- \( T^{prol} = 10 \) tumor cell proliferation time
- \( T^{death} = 100 \) tumor cell survival time
- \( \tau_b = 0.5 \) instantaneous reaction rate constant
- tumor cell grows if the density is not too high
Explicit time discretization:

\[
\begin{align*}
  b_{t+1} &= b_t + \Delta t \left( -\nabla \cdot J_t + b^- + b^+ \right) \\
  c_{t+1} &= c_t + \Delta t \left( \chi_c \Delta c_t - \gamma_c o_t c_t + c_t^+ \right) \\
  M_{t+1} &= M_t + \Delta t \left( -\beta M M_t b_t \right) \\
  A_{t+1} &= A_t + \Delta t \left( \gamma_A M_t b_t + \chi_{OA} \Delta A_t - \gamma_{OA} A_t \right)
\end{align*}
\]

\[
J = -D_b b \left( \nabla P + r_b \nabla A \right)
\]

\[
P = \begin{cases} 
  0 & \text{for } b < b^N \\
  \frac{b^N}{b^M - b^N} & \text{for } b^N \leq b \leq b^M
\end{cases}
\]

\[
b^+ = \frac{b}{T_{prol}} \left( 1 + \frac{\tau_b A}{\tau_b A + 1} P_b \right) \left( 1 - \frac{b}{b^M} \right) \quad \text{for } o > o^{prol}
\]

\[
b^- = -\frac{b}{T_{death}} \quad \text{for } o < o^{death}
\]
Numerical results

Initial state:
- tumor concentrated in the center of the domain
- constant ECM in each skin layer
- no TAF, no degraded ECM

Parameters:
- $80 \times 80$ elements
- quadratic B-splines ($p = 2$)
- $\Delta t = 10^{-3}$
- 30,000 time steps
- 8 hours of sequential simulation (around 1s / time step)
- around 40 minutes (12 times faster)
  with parallel GALOIS solver on 16 cores
Click in the middle
Click in the middle
2D simulation (3/3) Tumor cell density $b$

Click in the middle
Example 4: Propagation of elastic waves

Unit cube deformed by short impulse applied at the corner

\[
\begin{align*}
\rho \frac{\partial^2 u}{\partial t^2} &= \nabla \cdot \sigma + F \quad \text{on } \Omega \times [0, T] \\
u(x, 0) &= 0 \quad \text{for } x \in \Omega \\
\sigma \cdot \hat{n} &= 0 \quad \text{on } \partial \Omega
\end{align*}
\]

where \( \Omega = [0, 1]^3 \) is an unit cube, 
\( u \) is an unknown 3-dimensional displacement vector 
\( \rho \) is material density, 
\( f \) is the applied external force, 
\( \sigma_{ij} = c_{ijkl} \epsilon_{lk} \) is the stress tensor 
where \( \epsilon_{ij} = \frac{1}{2} (\partial_j u_i + \partial_i u_j) \) 
and \( c \) is rank-4 elasticity tensor (prescribed for a given material)
Example 4: Propagation of elastic waves

\[ \rho = 1, \]
\[ \mathbf{c} = 0 \text{ except for } c_{ijij} = c_{ijji} = 1 \text{ for } i, j = 1, 2, 3, \]

so that \( \mathbf{c} \) is positive definite and satisfies symmetry constraints stemming from its physical meaning.

The force applied is given by

\[ \mathbf{F}(\mathbf{x}, t) = -\phi(t/t_0) r(\mathbf{x}) \mathbf{p} \quad (2) \]
\[ \mathbf{p} = (1, 1, 1) \quad (3) \]
\[ t_0 = 0.02 \quad (4) \]

\[ \phi(t) = \begin{cases} 
  t^2(1 - t)^2 & \text{if } t \in (0, 1) \\
  0 & \text{otherwise} 
\end{cases} \quad (5) \]

\[ r(\mathbf{x}) = 10 \exp \left( -10 \| \mathbf{x} - \mathbf{p} \|^2 \right) \quad (6) \]

i.e. a short impulse directed towards the origin, applied at the opposite corner of the cube.
Example 4: Propagation of elastic waves

In order to utilize forward Euler integration scheme, the above system of 3 equations is converted to system of 6 equations by introducing additional variables corresponding to components of the displacement’s velocity:

\[ v_i = \frac{\partial u_i}{\partial t} \quad (7) \]

Corresponding weak formulation discretized with the Euler scheme is given by

\[
\begin{align*}
\langle u_i^{(t+1)}, w \rangle &= \langle u_i^{(t)} + \Delta t \ v_i^{(t)}, w \rangle \\
\langle v_i^{(t+1)}, w \rangle &= \langle v_i^{(t)} + \frac{\Delta t}{\rho} (\sigma_{ij,j} + F_i), w \rangle 
\end{align*}
\quad (8)
\]

for all \( w \in H^1(\Omega) \), where \( \langle \cdot, \cdot \rangle \) denotes standard scalar product in \( L^2(\Omega) \), and \( u_i, v_i \in H^1(\Omega) \).
Example 4: Propagation of elastic waves
Implicit dynamics

\[ \frac{du}{dt} - Lu = f \]

assuming constant coefficients and regular cube shape domain, where \( L = L_x + L_y \) is a separable differential operator, e.g. Laplacian, where \( L = L_x + L_y = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \). First, we apply the alternating direction method with respect to time. We introduce the intermediate time steps

\[
\frac{u_{t+0.5} - u_t}{dt} - L_x u_{t+0.5} - L_y u_t = f_t
\]

\[
\frac{u_{t+1} - u_{t+0.5}}{dt} - L_x u_{t+0.5} - L_y u_{t+1} = f_{t+0.5}
\]

We obtain

\[
u_{t+0.5} - dt \ast L_x u_{t+0.5} = u_t + dt \ast L_y u_t + dt \ast f_t
\]

\[
u_{t+1} - dt \ast L_y u_{t+0.1} = u_{t+0.5} + dt \ast L_x u_{t+0.5} + dt \ast f_{t+0.5}
\]
Implicit dynamics

We multiply by test functions $v$

$$\int (u_{t+0.5} - dt \ast \frac{\partial u_{t+0.5}}{\partial x})v = \int (u_t + dt \ast L_y u_t + dt \ast f_t)v$$

and we integrate by parts the second term

$$\int u_{t+0.5}v + dt \int u_{t+0.5} \frac{\partial u_{t+0.5}}{\partial x} \frac{\partial v}{\partial x} = \int (u_t + dt \ast L_y u_t + dt \ast f_t)v$$

We test and approximate with 2D B-splines

$$\sum_{i,j} u_{i,j}^{t+0.5} \int B_i^x(x)B_j^y(y)B^x_k(x) \ast B^y_l(y) + dt \ast \int \frac{\partial (B_i^x(x)B_j^y(y))}{\partial x} \frac{\partial (B^x_k(x)B^y_l(y))}{\partial x} = \int (u_t + dt \ast L_y u_t + dt \ast f_t)B^x_k(x)B^y_l(y)$$
Implicit dynamics

We have $\frac{\partial}{\partial x} B_j^y(y) = 0$ which results in

$$\frac{\partial}{\partial x}(B_i^x(x) \ast B_j^y(y)) = \frac{\partial}{\partial x}(B_i^x(x))B_j^y(y) + B_i^x(x)\frac{\partial}{\partial x}(B_i^y(y))$$

and

$$\sum_{i,j} u_{i,j}^{t+0.5} \int B_i^x(x) \ast B_j^y(y)B_k^x(x) \ast B_i^y(y)$$

$$+ dt \ast \int \frac{\partial((B_i^x(x)))}{\partial x} B_j^y(y) \frac{\partial(B_k^x(x))}{\partial x} B_i^y(y) =$$

$$\int (u_t + dt \ast L_y u_t + dt \ast f_t)B_k^x(x) \ast B_i^y(y)$$

so our left-hand-side matrix is the Kronecker product of

$$[\int (B_i^x(x)B_k^x(x) + dt \ast (\frac{\partial}{\partial x} B_i^x(x))(\frac{\partial}{\partial x} B_k^x(x)))dx] \ast$$

$$[\int B_j^y(y)B_i^y(y)dy]$$

and it can be factorized in a linear $O(N)$ cost.
Example 5: Pollution from a chimney with a wind

We seek the pollution concentration scalar field $c: \Omega \to \mathbb{R}$ such as:

$$
\begin{aligned}
\frac{\partial c}{\partial t} + u \cdot \nabla c - \nabla \cdot (K \nabla c) &= e & \text{on } \Omega \times [0, T] \\
\nabla c \cdot \hat{n} &= 0 & \text{on } \partial \Omega \times [0, T] \\
c(x, 0) &= c_0(x) & \text{on } \Omega
\end{aligned}
$$

(9)

where $\Omega = [0, 1]^3$,

$\hat{n}$ is a normal vector of the domain boundary,

$T$ is a length of the time interval for the simulation,

$u$ is the prescribed wind,

$e$ is the prescribed emission from the chimney,

$K$ is the diffusion,

and $c_0$ is an initial state.
Example 5: Pollution from a chimney with a wind

\[ \Omega = 5km \times 5km \times 5km \]
Mesh size = 100 \times 100 \times 100
Wind = F \ast (cosa(t), sina(t), v(t)) \text{ where}
\[ a(t) = \frac{\pi}{3}(\sin(s) + 0.5\sin(2.3s)) + \frac{3}{8\pi} \]
\[ v(t) = \frac{1}{3}\sin(s) \]
\[ s = t/150 \]
chimney \( e(p) = (r - 1)^2(r + 1)^2 \) where \( r = \min(1, (|p - p_0|/25)^2) \)
\[ p_0 = (3000, 2000, 2000) \]
Diffusion \( K = (50, 50, 0.5) \)
We run 300 time steps of the implicit method
Space instability - oscillations and reflections

Click in the middle
Conclusions and future work

Conclusions

- Explicit dynamics with isogeometric L2 projections
- Multiple applications (e.g. heat transfer, propagation of elastic waves, tumor growth)
- Possible extension to implicit dynamics (heat transfer, propagation of elastic waves, pollution simulations)
- Limitations: tensor product geometry, Kronecker product material coefficients

Future work

- Iterative solver for complicated geometry
- Iterative solver for complicated material coefficients
- Implicit dynamics for tumor growth simulations
- Incorporation of the spatial stabilization for the pollution simulations