N. Thuerey
You Xie

Technische Universität München
Fakultät für Informatik
Lehrstuhl für Games Engineering

Game Physics – Programming Exercise

Exercise 3 – *Diffusion Equation*

Task Overview

In this exercise, you will practice to solve a diffusion equation with explicit and implicit approaches. Please switch to the Ex3 branch of the template project (https://github.com/GamePhysicsTUM/gamephysicstemplate). You can find a basic solver, a preconditioned conjugate gradient (PCG) solver (pcgsolver.h & vectorbase.h) for exercise 3 and an example implementation (DiffusionSimulator.cpp/h) which shows the pipeline for the tasks and how to work with PCG solver. The PCG solver takes a matrix A, and vectors x and b to compute $x=A^{-1}b$ for the linear system Ax=b. Thus, you can use it below to compute the solution of the next time step. Your main task is to set up the correct matrix A and right hand side b. The initial value of x is less important, and typically zero.

The heat equation is a special and simplified version of the diffusion equation. Applying the heat equation to the temperature field (length: *m*, width: *n*) leads to the following equation:

$$T_t - \alpha \nabla^2 T = 0 \qquad (1)$$

with T being the temperature, $T \in \mathrm{R}^S$, s = m × n; T_t being the temperature's time derivative and α the diffusion coefficient.

Discretize the heat equation. In order to diffuse the temperature over time, and compute a solution at every time step, the heat equation above needs to be solved. Therefore, you have to discretize equation (1) with forward and central finite differences (forward for the time derivative T_t, central for the second order spatial derivative $\nabla^2 T$).

Hints: time is one-dimensional. You can also have a look at "Finite-Difference Approximations to the Heat Equation" by Recktenwald at page 7 (http://www.nada.kth.se/~jjalap/numme/FDheat.pdf). You can find some additional helpful information about explicit and implicit solves at the following page: http://hplgit.github.io/num-methods-for-PDEs/doc/pub/diffu/sphinx/._main_diffu001.html. It is recommended to start with very small resolutions, e.g. a 16 by 16 grid with: m × n = 16 × 16.

Demo requirements:

Your submission should contain the following demos and should support **interactive switching between these demos**:

- **Demo 1: Solve the 2d diffusion equation explicitly**

  - First take pen and paper, and manually discretize 2d heat equation applied to the temperature with forward and central finite differences.
  - In order to diffuse the temperature field at every time step, you need to solve the discretized heat equation. The easiest way to do so is by applying the explicit Euler scheme. For our heat equation, temperature values at time $t + \Delta t$ are computed solely from values at time t.
  - Solve the discretized heat equation with an explicit Euler. Implement the method diffuseTemperatureExplicit(…) in DiffusionSimulator.cpp. Call this function where all the magic happens and make sure that the temperature is always zero in boundary cells.
  - Interactively change values of m and n for the simulation.
  - For visualization, you could use spheres to represent every point position, and use color of the sphere to represent temperature of this point position (red for negative value and white for positive value). Then we can see how the equation is solved via observing color change.

- **Demo 2: Solve the 2d diffusion equation implicitly**

  - Now, implement an implicit backward Euler solver so that the simulation does not get unstable for any Δt and α. When using implicit Euler, the new approximations of T appear on both sides of the equation. (Again, check the Backward Euler scheme explained on the website above.) In order to solve this new, and more complicated, discretized heat equation, you need to set up the following equation system.

$$AT = b \tag{2}$$

    Here $A \in \mathrm{R}^{s \times s}$ is the coefficient matrix, and T is the new resulting temperature vector at time t. A collects all coefficients for T at $t + \Delta t$, while $b \in \mathrm{R}^{S}$ is a vector computed with the old temperature values at time t.
  - **Solve the discretized heat equation with an implicit Euler.** Setup the matrix A and vector b of equation system (2). Then, use the provided pcgsolver.h to solve the equation system by passing A, T and b to the solver. The example for DiffusionSimulator.cpp contains some information on how to do that. Set the temperature in boundary cells explicitly to zero in order to avoid positive values resulting from accumulating these boundary temperature values. Since zero equations cause problems in the solver, set the diagonal value of A for the boundary cells to 1.0 before

adding the other entries of A. Simulate your simulation again and use a parameter setting for Δt and α that was previously unstable for explicit integration. Show that it computes a solution without stability problems.

- Interactively change values of m and n for the simulation.
- Visualize the solution as for Demo 1.

- **Optional Demo 3: 3D implementations for demos 1 & 2**
  - Additionally solve 3d heat equation explicitly and implicitly.
  - Visualize the middle part of the volume using the same approach as demo 1 & 2 (temperature at boundary position is zero, so only the middle part of the volume should be visualized), or try to visualize the full 3D array.

Submission

The deadline for this exercise is on **Jan. 18, 23:59**.

If you are using Bitbucket/GitHub, please send your repository link to gamePhysTum@gmail.com, and make sure that our tutor has the permission accessing it. If you already shared the repository with your tutor, a notification email before the deadline is still necessary, pointing out which commit the tutor should use as your final version.

If you are not using Bitbucket/GitHub, please pack all your source files (.h and .cpp) and project files (.vcxproj) under the "Simulations" directory into a zip file, and name it "Group??_Ex3_VS201?.zip", and sent it to gamePhysTum@gmail.com. Make sure not to include the compiler temporary files (they will be under the Simulations/Win32/ directory). Your package should be smaller than 100kb.