Interactive Visual Data Analysis

Assignment 7 – *Arrow Glyphs and Particle Tracing*

In this assignment we will implement two of the most basic flow visualization techniques. Firstly, we will use simple arrow glyphs for direct visualization and secondly, we will implement a basic particle tracing system.

7.1. Arrow Glyphs

Add a mode to your tool that visualizes the vector field using simple arrow glyphs. Use a geometry shader to generate the glyph geometry on the fly.

- Generate point primitives in the vertex shader (one for every grid point of the vector field) using `D3D11_PRIMITIVE_TOPOLOGY_POINTLIST`.
  Hint: You cannot bind the 3D texture as input to the input assembler, but you can read from it within the vertex shader. Disable input layout and sample the texture based on `SV_VertexID`.
- Write a geometry shader that transforms the point stream into some arrow-like geometry of your choice. E.g. a 2D shape on a view aligned plane (see Slide on *View Aligned Geometry*), or a stretched 3D tetrahedra.
- To avoid visual clutter, allow for the user to select single, axis-aligned slices of the vector field at a time.

7.2. Particle Tracing

Implement a basic particle tracer using D3D11 compute shaders. In this first indirect flow visualization technique, massless particles (i.e. no inertia) are spawned at random positions across the whole flow domain and advected along the flow velocity.

- The attributes of all particles should be stored in a buffer. You can create a second buffer to hold random (CPU-generated) spawn positions.
- In each frame, launch a compute shader using `Dispatch()` which advects all particles using Euler integration and updates the particle buffer.
- Whenever a particle leaves the flow domain or reaches a user-defined maximum age, it should be removed or flagged as inactive.
- The exact seeding strategy is up to you, but it should ensure that the number of particles spawned per unit time stays roughly constant (so that after some time to stabilize, the total number of active particles also stays roughly the same). Note that creation/deletion of GPU resources is expensive, so you should not re-allocate the buffer regularly!
- Draw the particles as point primitives (i.e. one vertex per particle, drawn with `D3D11_PRIMITIVE_TOPOLOGY_POINTLIST`).
- **Ensure that your vector field scaling is correct!** Compare your result with the demo application in `data\vector\VortexStreetTest` which demonstrates the correct particle behavior for the VortexStreet data set.

7.3. Visualize!

The same procedure as every week! Commit some nice screenshots in PNG format to a folder called `screenshots/assignment7/` outside of your solution directory.

The working solution must be committed until **December 4, 09:00am**. If anything is not working as described here or if you want a specific SVN-Revision to be rated, explain yourself in the `readme.txt` file within your `solution` directory.