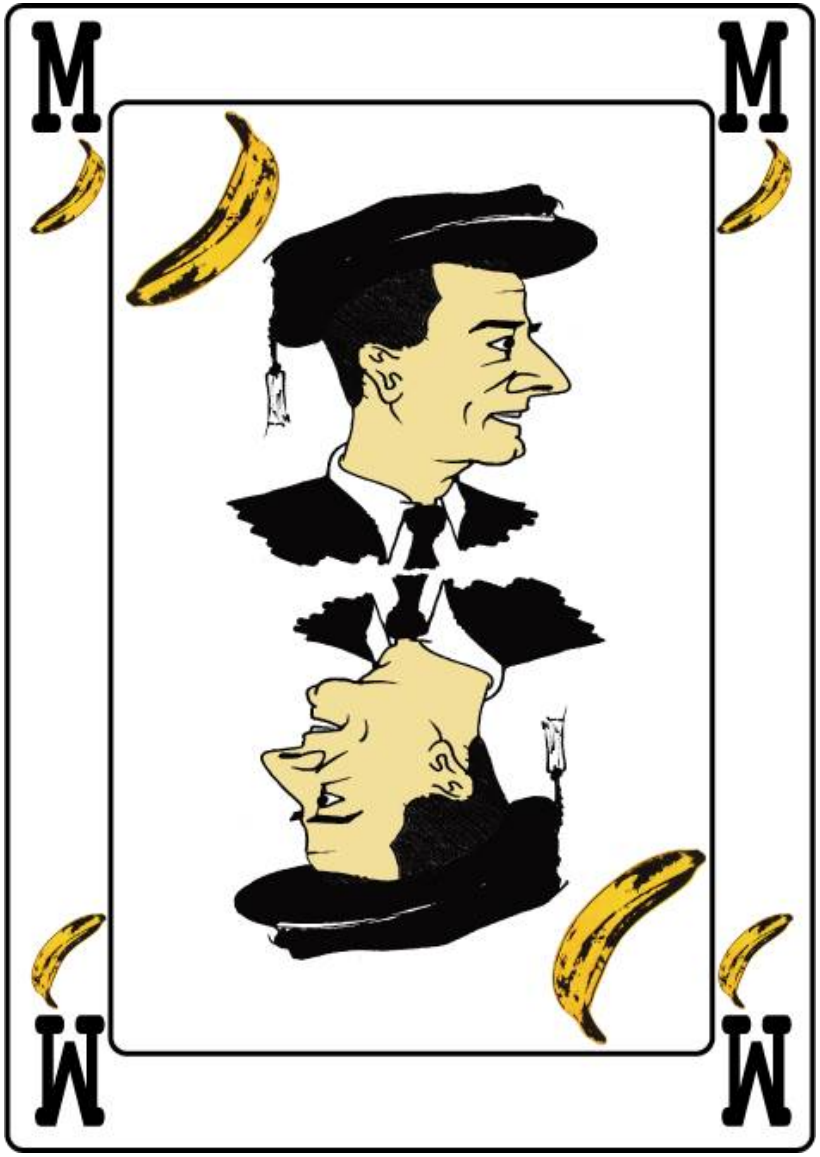
Modern approaches to Flow Visualization

Martin Ilčík

Institute of Computer Graphics and Algorithms

Vienna University of Technology

Outline



- General concept
- Recycling
- Geometry grids
- Glyphs
- Integration

IT'S NOT THE MINIMUM SET

Most of this tutorial goes
FAR FAR FAR
behind the VisLU requirements

Putting it all together…

- No matter that we haven't started yet
- What you'll get by your program
  - color coding
  - arrowplots
  - stramlines, pathlines
  - particles, LIC
- All are layers = independent images
  - combine together
  - save to a nice image

Selections, Focus and Context

- Layer selections
  - unique for each layer
  - histogram or scatterplot
  - smooth selections
- HSL transfer function
  - hue = channel value mapping
  - saturation = degree of interest
  - lightness = illumination

Transfer Function Recycling

- Additional to Bsp. 1
  - more scalar fields
  - arbitrary channel values
  - search for minimum & maximum
  - global vs. local time step
- Consequencies
  - more TFs
  - time dependent TFs
- Implementation
  - 2D Texture (Value x Time) for each channel

FlowVis Framework

- Basical data loading
  - curvilinear grids
  - channel handling
- Vector class with many operations
- Possible improvements
  - TimeChannel class
    - wrapping channels of all timesteps
    - global extrema
    - spatio-temporal interpolation
  - own lookup and interpolation method

Grids are getting worse

- Not Equidistant
  - membership to a cell not obvious
- Rectangular (not the Tube)
  - we can search for each axis separately
- Steady
  - don’t change in time
- Separate file for the grid
  - saves memory
  - you should do a lot of preprocessing

Grid lookups with textures

- Grid texture
  - for each texel stores it‘s cell adress
  - position -> cell (inverse of the grid file)
  - one 2D texture for all the data
- Data texture
  - for each cell stores it‘s value
  - one for each timestep
- Lookup for a position
  - color = TF(valueAt(cellAt(position)));
- Works fine, but not exactly...

Grid textures… WHAT'S WRONG?

- Texture interpolation linear
  - cellAt(position) interpolates lineary
- Crossing more cells
  - underlying grid nonregular
  - linear interpolation regular
- How to fix?
  - appropriate sampling = adapt resolution

Common vs. Float textures

- Integer textures
  - clamp to <0,1>
  - 16bits fixed point
  - supported anywhere
  - fast lookups
  - linear interpolation
- Float textures
  - arbitrary numbers
  - good accuracy
  - HW support problems
  - slow lookups
  - only GL_NEAREST for 32bit textures

Storing vectors in textures

- Best stored as
  - rgba = (packed_normalized, 1.0/length);
    - if length < 1.0 then rgba = (packed, 1.0);
    - in 2D use the BLUE for packing when length < 1.0
  - or normalized per component
    - using overal minumum and maximum of the channel
  - or floating point textures
- packing of a vector shorter or equal than 1.0
  - return (vector*0.5) + (0.5,0.5);
- unpacking
  - return (vector*2.0) – (1.0, 1.0);
- Don’t forget about our special coord’s system
  - y increases from the top to the bottom!

Many glyphs made fast (cookbook)

- Arrowplots ideal for GPU processing
  - many arrows
  - many lookups
  - attribute mapping
- Parallelism
  - not iterative
  - only local information needed
- Hardware support excellent
  - Point sprites
  - Shaders

Ingredients: Geometry primitives

- Point sprite extension
  - arbitrary sized rectangular points
  - autogenerated texturing coordinates
  - points always face to camera
- Shaders (use OpenGL 2.1)
  - behaviour with point sprites different
  - using texture lookups in vertex shader
  - rotation of texture coordinates in fragment
  - point size adaptation
- Don’t forget to update your drivers

Cooking: Create Point sprites

- Generate point sprites
  - place at desired locations
  - bind a glyph texture (e.g. arrow)
  - use GL_POINTS, store in a display list ☺
- Enable points rendering stuff
  - glEnable(GL_POINT_SPRITES);
  - glEnable(GL_VERTEX_PROGRAM_POINT_SIZE);
  - set glEnvTexf(GL_POINT_SPRITE_ARB, GL_COORD_REPLACE_ARB, GL_TRUE);
  - for each texture unit you want to use in fragment shader (with gl_PointCoord)
- Bind velocity texture to another texturing unit

Cooking: Vertex shader (SM3)

- Lookup velocity vector at glyph's position
  - negate the y component (remember our coordinates)
- Compute the rotation matrix
  - one acos with x component
  - if y < 0 subtract from 2*PI
- Point sprite troubles
  - texture coordinates not available in vertex shader
  - varying variables behave different, better don't use
  - unused texture unit's coords fine to pass the rotation matrix
- Read color coding

Cooking: Fragment shader

- Create a rotation matrix
  - just map the values from gl_TexCoord[n]
- Rotate around the glyph's center
  - if center at (0.5, 0.5) then
    - vec2 h = (0.5, 0.5);
    - newcoord = (rotMat * (gl_PointCoord – h)) + h;
- Additional attribute mapping
- Color mapping

Numerical Integration

- Particle tracing
  - movement in a vector field
  - previous positions create a trail
  - strongly sequential, like a ray traversal
- Integration
  - estimates the final position
  - ugly integrals and differential equations ☺
  - fast unstable explicit schemes
  - slow accurate implicit schemes

Methods using explicit schemes

- Euler
  - absolutely precise with lim dt → 0
  - we'll never make that
- Runge-Kutta
  - several euler steps combined together
  - best results for sampled grids
- Runge-Kutta-Fehlberg
  - adaptive stepsize
  - behind the scope of the VisLU

One particle GPU tracing

- Memory requirements
  - Velocity field texture
  - two 1x1 textures bound to two FBOs
    - ping-pong – one for reading one for writing
  - Fullscreen (1x1) quad for rendering
- GPU Numerical Integration
  - select a method
  - apply to the fragment's position
  - return the new position
  - one texture for reading, the other for writing
- For n^2 particles use nxn textures

One particle SW tracing

- Texture-lookups on our own
  - linear interpolation inside a cell
  - weighting values at vertices
- Cell search complicated
  - Quad-Trees
  - kD-Trees
  - local search
  - dimension reduction (only rectangular grids)

Euler step in detail + Enhancements

- input information
  - starting cell
  - position
  - evaluated movement vector
  - step size
- move along the vector
  - adapt step size to reasonable value
- find a cell for the new position
  - any idea where we landed?
  - local search fastest

Streamlines and Pathlines

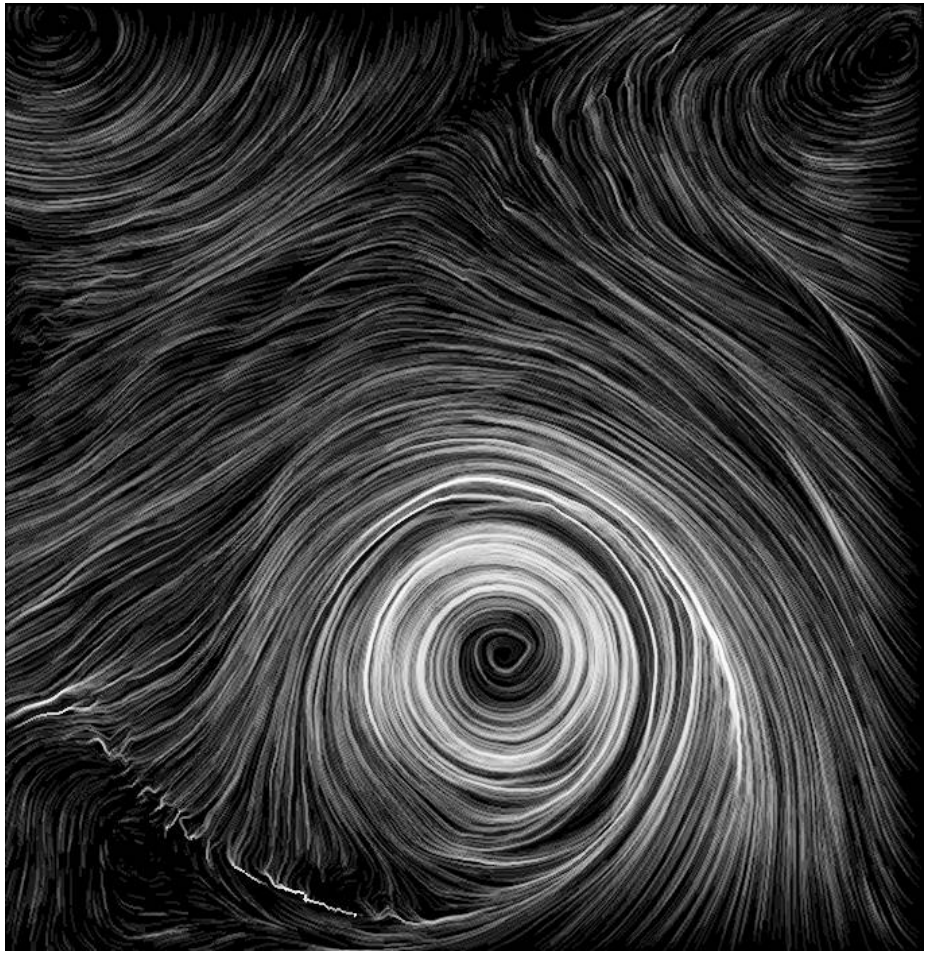
- Streamlines
  - only for one timestep
  - don’t cross eachother
- Pathlines
  - one line crosses many timesteps
  - projection 4D -> 3D
  - can cross
  - in steady fields = streamlines

2D Streamlines HW computation

- Take two big textures
  - fill the first column with initial positions
  - select a maximum length for a line
  - iterate maxlength times
    - advance one step further
    - write new position to the next column
    - swap textures, one for reading, one for writing
- Read the resulting texture
  - done only once
  - store as line strips in a display list or other
- For pathlines read two consequent timesteps

Living particle system

- Modifications to lines
  - shorten them
  - lower stepsize
  - store only a static display list
- Use vertex shader (SM3)
  - displacement
  - vertex texture lookup
- Additions to fragment shader
  - sequentially shift the values to the left



Mass alternative

- Many particles & short trails
- For n^2 particles
  - use nxn textures
  - for each step exactly one texture
- Faster computation
  - more parallel advection
  - no ping-pong needed
- Slower rendering
  - limited by samplers in vertex shader

Evenly spaced 2D lines in HW

- Bad parallelization
  - sequentially obtained information needed
  - seeding points discarded
- Very difficult … don't try… just think about it
  - a line divides the domain into two regions ☺
  - 2 seed points produced by each vertex
  - we have a set of synchronized processes
    - they all run each rendering
    - switch states (integrating, seeding, noop…)

LIC is much easier

- Texture based methods
  - no geometry generated
  - only simple integration
    - shown a few slides before
    - can be done in one pass
    - similar to the integration in VolVis assignment
- each fragment outputs only to itself
  - convoluted values directly written

Conclusions

- Use hardware rendering
- Use the best API for you
- Store the vector data in textures
- Shaders make things much simpler and faster

- Feel free to ask: ilcik at cg.tuwien.ac.at
- Thank you for your attention!