Vector Field Viewer

# 

### Authors:

|  |  |
| --- | --- |
| Dmitry Ogurtsov | ogurdima@gmail.com |
| Yacov Glushkin | yacov.gls@gmail.com |

### Instructor:

|  |  |
| --- | --- |
| Miri Ben-Chen | mirela@cs.technion.ac.il |

# Overview

VectorFieldsViewer is a program for visualizing of vector field on surface of mesh. The program is able to load a mesh in OBJ format and a field in field-per-face format. Time-variable vector fields are also supported. See User Guide for a detailed field format description.

To visualize the field, the program performs piecewise-linear particle movement simulation. Initially particles are spawned in the middle of each face. At each point of time particle location is being calculated using vector field direction/strength in the neighbourhood. This way when the simulation is finished we have a bunch of particle paths that we are able to display.

# Prerequisites

* Windows platform
* OpenGL v2.0
* .NET v4.0

# Features

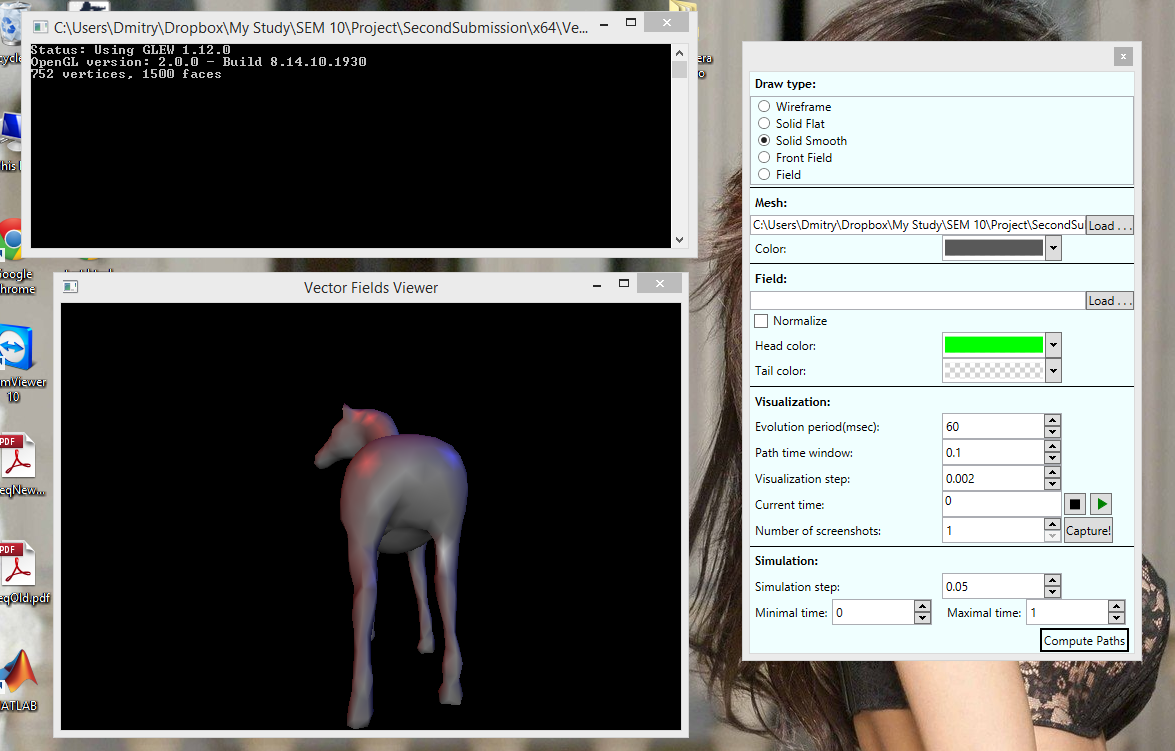
* 32 and 64 bits support, latter required for detailed visualization of large models
* Mesh loading in Object File Format
* Support for constant field or time-variable field expressed in terms of constant field snapshots
* Smooth field transition across different mesh faces achieved by translating internally field-per-face into field-per-vertex representation and interpolating vertex values
* OpenMP based parallel simulation
* Fixed Stack OpenGL real time visualization
* Powerful GUI control panel that allows controlling simulation and visualization parameters
* Support of visualization snapshots in PNG format

# User Guide

When you open the program

You see three windows:

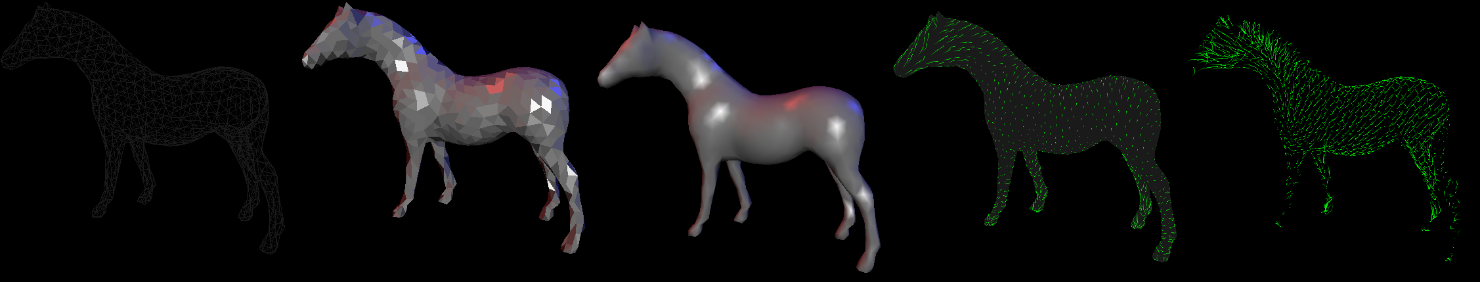
* Textual console – used for displaying information about current state of the program and some basic mesh / particle path statistics
* OpenGL window (with horse in it) – here you see the visual information
* GUI control panel – used for controlling the program



## Ways to control viewing from OpenGL window

* Use left mouse button to rotate the model
* Use middle mouse button or cursor to translate the model
* Use mouse wheel or ‘+’ and ‘-‘ keys (near backspace) for zoom
* Pressing ‘p’ will take snapshot of the window and save it in file capture.png in the application directory

## Different Draw Modes



## Control Panel Overview

|  |  |
| --- | --- |
| The control panel is divided into 5 sections. Let’s go through them one by one:   * Draw type   + Used for selecting how to draw a mesh. Wireframe, Solid Flat, Solid Smooth are standard ways to display a mesh. Select Field option to view all vector field lines (and no mesh) or Front Field option to show the mesh and vector field lines that are visible (are in front of the mesh)   + Note that Front Field and Field options will not work unless paths are already computed, so you need to press “Compute Paths” button before selecting one of these options * Mesh   + Used for specifying which mesh to display and what color should it be displayed with * Field   + Used for specifying what vector field will be loaded and used in next paths computation   + Normalize checkbox specifies whether the field should be automatically normalized (all vectors should be unit vectors)   + Head and Tail color dialogs allow you to set the way particle paths will look. See the next section for explanation of what is “Head” and “Tail” | C:\Users\Dmitry\AppData\Local\Microsoft\Windows\INetCache\Content.Word\GUI.PNG |
| * Visualization and Simulation   + Simulation specifies how the paths are computed   + Visualization specifies how the computed paths are drawn   + Both computation and visualization run between “Minimal time” and “Maximal time”   + “Simulation step” is the time difference that is added to the current time value in simulation (dt in formula t = t + dt). The smaller it is the finer the simulation is, but it also means that computed paths will be longer, thus the program will be more resource-hungry   + Once the paths are computed the visualization loop begins. It runs the following way:     - Every “Evolution period” miliseconds     - “Current time” = “Current time” + “Visualization step”   + During the visualization     - Path head is particle’s location in t = “Current time”     - Path tail is particle’s location in t = “Current time” – “Path time window”   + “Play” and “Stop” buttons are used to pause/continue the visualization loop   + You can take several snapshots that divide interval [Minimal time, Maximal time] uniformly by selecting the number of screenshots in the respective entry and pressing “Capture!” button. Directory selection dialog will open and once you click “Select Folder” the screenshots will be saved to the specified location * Last but not least, “Compute paths” button will start a fresh path computation using the parameters from the control panel   Note that visualization parameters and colors change is seen immediately in the OpenGL window, while changing simulation parameters, mesh or field will require “Compute paths” button to be pressed to take effect. |  |

## Troubleshooting

* If the program does not start or crashes immediately after the start go to C:\Users\<USER>\AppData\Local\Temp and delete files that look like this: .NETFramework,Version=v4.0.AssemblyAttributes. This is (not so) known .NET bug.
* If the program freezes/crashes and the last line in console "Configuring PathsManager..." you are probably out of memory, run 64 bit version

## Field file syntax

### Constant Field

* All lines are the same
* Each line represents a vector in 3D space
* Number of lines must be equal to number of faces of (already loaded) mesh
* File extension must be .vf
* Example:  
  0.008760250 0.004253466 0.005110653  
  0.021301756 -0.001675812 -0.004383483  
  ……

### Variable Field

* All lines are the same
* Each line represent a snapshot of vector field in a specific time
* Format: <time> <filePat h>
* filePath is path to file with Constant Field syntax
* Example:  
  0.001 frog\_s5\_0.001.vf  
  0.03 frog\_s5\_0.026.vf

## Nice mesh/field/parameter configurations to get you started

* Cat
  + mesh: cat.off
  + field: cat.vf normalized
  + simulation step: 0.01
  + everything else is default
* Teddy
  + mesh: teddy.off
  + field: teddy.vf normalized
  + simulation step: 0.01
  + everything else is default
* Frog
  + mesh: : frog\_s5.off
  + field: frog\_s5\_times.txt not normalized
  + simulation step: 0.01
  + everything else is default

# Technical Details and Challenges

* The program consists of a simulation/visualization core written in native C++ and a UI control panel written in C# WPF
  + Communication between the two is achieved through managed C++ mediator
  + The mediator is a managed dll consumed by the native C++ core
  + UI control panel is a .NET dll consumed by the managed dll
* Geometrical data is stored and manipulated using open-source OpenMesh framework (DCEL based data structure)
* Vector field data is loaded in field-per-face format and translated into field-per-vertex format. In general case each vertex is associated with a number of field snapshots, each containing time of the snapshot and the vertex field vector at that time. Field in vertex allowed to calculate field in particular point independent of other faces. That gives performance advantage over any n-ring neighborhood computation.
* Particle simulation is performed before the visualization. The simulation algorithm is as follows:
  + At each step let *t* represent current time, *loc* represents current particle location
  + Compute vertex field in *loc* at *t* by taking into account vector field snapshots in neighboring vertices
  + Move the particle to the next location according to the vector field (vector field is treated as speed of the particle at that time)
  + Check whether we crossed the boundary of a face. If so move the particle to the intersection point
  + Continue for *t=t+dt*
* Simulation involved several major difficulties that needed to be handled:
  + Intersection testing with edges accounting for numerical errors
  + Non-uniform distances between particle locations resulting in very long particle paths with extremely dense sub paths
  + Numerical instabilities caused by projecting vector fields onto neighboring faces. This created paths that “jump” between two specific faces without increasing *t*
* Visualization uses fixed-stack openGL (version 2). Multiple polylines representing particle paths are drawn each frame, requiring efficient data transition between CPU and GPU. Specific data structure the particle paths was required to achieve real-time performance even on modest models