OpenTissue

A Low-level OpenSource Physics API

Henrik Dohlmann and Kenny Erleben

Department of Computer Science

Copenhagen University

OpenTissue

- An Open-source Library for Physics-based Animation, released under the terms of the GNU Lesser General Public License.
- A low level application programming interface (API) and a playground for future technologies in middleware physics for games and surgical simulation.
- Developed mainly by the Computer Graphics Group at the Department of Computer Science, University of Copenhagen.
- There is a support/community mailinglist (subscribe by sending an email to opentissue-request@diku.dk with the subject: “subscribe”).

Background History of OpenTissue

- In November 2001, K. Erleben, H. Dohlmann, J. Sporring, and K. Henriksen started to collect a toolbox of code pieces
- The ambition was to ease project collaboration and teaching efforts.
- In the beginning, OpenTissue worked as a playground for experiments
- soon it became apparent that the code pieces in OpenTissue were conveniently reused again and again.
- This was the turning point for OpenTissue, and the first thoughts were seeded towards creating a toolbox for physics-based animation.
- Late August 2003 it became clear that students often re-invent the wheel during their project work, limiting the time set aside for getting a feeling and in-depth understanding of the simulation methods they work with. OpenTissue thus proved to be a valuable tool for student projects also, and OpenTissue was released under the terms of the GNU Lesser General Public License.
- Today OpenTissue works as a foundation for research and student projects in physics-based animation at the Department of Computer Science, University of Copenhagen (commonly known as DIKU).

Overview

- Rectilinear 3D Grid Data structures and Chan-Vese Segmentation Tools
- Twofold Mesh Data Structure and Tetra4 Mesh Data Structure
- OpenGL based signed distance map Computations and Voxelizer
- Quasi Static Stress-Strain Simulation (FEM)
- Relaxation based Particle System Simulation
- Dantzig LCP Solver, Path and Lemke wrappers (LCP solvers)
- CJK, SAT, VClip
- Mesh Plane Clipper and Patcher, QHull Convex Hull wrapper
- Script files for Maya and 3DMax
- Generic Bounding Volume Hierarchies
- Multi-body Dynamics and Volume visualization
- Constitutive Elastic Model for deformable objects
- And much more

Benefits and Drawbacks

Benefits

- It is FREE to use
- Latest technology, research and theory is constantly being added, no hidden secrets
- Common framework
- Low-level allows user to play around with the details in the simulation methods
- There is mailinglist support

Drawbacks

- Low-level requires user to be
  - Experience programmer
  - Have minimum knowledge of physics and numerical methods
- Many contributors means
  - It takes time for code to mature

Future Plans

- OpenTissue is currently a little messy, due to many contributors and ad-hoc extensions, we like to see it mature
- We would like to apply a Generic Programming style everywhere in OpenTissue (Mesh and BVH data structures are up for being refactored)
- We are trying to integrate OpenTissue in the Physics Based Animation Graduate Course at DIKU
- More simulation methods will be added in the future (SPH, Cloth etc.)
- Full support for KDeveloper on Linux
- Anonymeous Access to CVS repository

Particle Systems (1/8)

To setup a simple particle system, you will first need to instantiate a PS_ParticleSystem class

```
#include <OpenTissue/dynamics/particleSystem/ps_particlesystem.h>
PS_ParticleSystem psys;
```

The particle system needs a numerical integration method

```
#include <OpenTissue/dynamics/particleSystem/ps_eulerintegrator.h>
#include <OpenTissue/dynamics/particleSystem/ps_verletintegrator.h>
PS_EulerIntegrator euler;
PS_VerletIntegrator verlet;
```

Particle Systems (2/8)

In order to make particles move around you need to apply some forces to them

```
#include <OpenTissue/dynamics/particleSystem/ps_gridforcevectorfield.h>
#include <OpenTissue/dynamics/particleSystem/ps_gravityforce.h>
#include <OpenTissue/dynamics/particleSystem/ps_viscosityforce.h>
PS_GravityForce gravity;
PS_ViscosityForce viscosity;
PS_GridForceVectorField field;
```

Observe that a force instance represent a force type

Particle Systems (3/8)

The random force field need a vector map for its initialization

```
#include <OpenTissue/map/map.h>
OpenTissue::Map<OpenTissue::Vector3> vectorMap;
```

The map data structure is simply a 3D grid of nodes, each node contains an instance of the specified template data type. To make the simulation more interesting we will also add a static half-plane

```
#include <OpenTissue/dynamics/particleSystem/ps_planegeometry.h>
PS_PlaneGeometry plane;
```

Particle Systems (4/8)

The half-plane is initialized by setting its normal and distance to the origin,

```
plane.plane.d = -5;
plane.plane.n.set(0,1,0);
```

Gravity and viscosity are initialized by setting the coefficients of gravity and viscosity,

```
gravity.setGravity(0.98);
viscosity.setViscosity(.5);
```

Particle Systems (5/8)

The force field requires a bit more initialization, first we must allocate the map structure,

```
Vector3 minCoord(-25,-25,-10);
Vector3 maxCoord(25,25,25);
int I = 128;
int J = 128;
int K = 128;
vectorMap.create(minCoord,maxCoord,I,J,K);
```

The force field class have a few static methods to help initialize the “force grid”,

```
PS_GridForceVectorField::random(vectorMap,10);
field.init(vectorMap);
```

Particle Systems (6/8)

Lastly we need to connect everything to the particle system class

```
psys.setIntegrator(&verlet);
psys.add(&plane);
psys.add(&field);
psys.add(&gravity);
psys.add(&viscosity);
```

Particle Systems (7/8)

In your display method or call-back function you must draw the particle system and the geometries,

```
void Application::display(void)
{
  glColor3f(0.,0.3,0.);
  psys.draw();
  glColor3f(0.3,0.3,0.3);
  field.draw();
  glColor3f(0.,0.,0.3);
  plane.draw();
}
```

Particle Systems (8/8)

We need to add particles and to ask the particle system to simulate their motion,

```
void Application::run(void)
{
  psys.run(0.01666);
  Vector3 p;
  p.random(-25,25);
  if(p.z<0)
    p.z = 0;
  psys.birth(p);
}
```

Voxelization of Mesh Data (1/3)

The voxelization tool is defined in the header

```
#include <OpenTissue/map/util/voxelizer.h>
OpenTissue::Voxelizer<float> voxelizer;
```

The template argument indicates the data type stored in the voxels. The voxel map is simply a 3D grid, in OpenTissue this is called a map,

```
#include <OpenTissue/map/map.h>
Map<float> voxels;
```

The template argument indicates the data type stored in each node of the map.

Voxelization of Mesh Data (2/3)

We also need to setup a mesh

```
#include <OpenTissue/mesh/mesh.h>
#include <OpenTissue/mesh/io/defaultMeshIO.h>
Mesh mesh;
DefaultMeshIO io;
io.read("foo.msh",mesh);
```

Now the mesh object have been setup we need to allocate the voxel map,

```
Vector3 minCoord, maxCoord;
mesh.getMinMaxCoords( minCoord , maxCoord );
int resolution 20;
int I = ceil( maxCoord.x - minCoord.x / resolution );
int J = ceil( maxCoord.y - minCoord.y / resolution );
int K = ceil( maxCoord.z - minCoord.z / resolution );
voxels.create(minCoord,maxCoord,I,J,K);
```

Voxelization of Mesh Data (3/3)

Now we are ready for computing the voxelization

```
voxelizer.run(mesh,voxels);
```

Afterwards the voxel map contains the voxels,

```
    for ( int k = 0;k < K;++k )
      for ( int j = 0; j < J; ++j )
        for ( int i = 0;i < I;++i )
        {
          bool voxel = voxels.getValue(i,j,k);
          if ( voxel )
          {
            //... Do something
          }
        }
```

Distance Fields (1/3)

We will use two new classes, the brute signed distance field class and the binary map io class

```
#include <OpenTissue/map/util/bruteSignedDistanceField.h>
#include <OpenTissue/map/io/binaryMapIO.h>
BinaryMapIO<float> mapIO;
BruteSignedDistanceField<float> bruteSignedDistanceField;
```

The template argument indicates the data type of the signed distance map. We read in mesh geometry from an ascii file

```
Vector3 center;
DefaultMeshIO meshIO;
meshIO.read( "foo.msh", mesh );
Mesh mesh;
mesh.getCentroid(center);
center.negate();
mesh.translate(center);
```

Distance Fields (2/3)

Next we scale the mesh to be within unit cube size

```
Vector3 minCoord,maxCoord,extent;
mesh.getMinMaxCoords( minCoord, maxCoord );
extent.sub(maxCoord,minCoord);
scalar scale =1./ max(max(extent.x,extent.y),extent.z);
mesh.scale(Vector3(scale,scale,scale));
```

We allocate a map data structure to contain the signed distance map

```
mesh.getMinMaxCoords( minCoord, maxCoord );
scalar boxband = 0.2*minCoord.distance(maxCoord);
minCoord.sub( Vector3( boxband, boxband, boxband ) );
maxCoord.add( Vector3( boxband, boxband, boxband ) );
Map<float> dist;
int I = ...; int J = ...; int K = ....
dist.create( minCoord, maxCoord, I, J, K );
```

Distance Fields (3/3)

Now we are ready for computing the signed distance field and writing the computed distance map in binary format

```
bruteSignedDistanceField.run( mesh, dist );
mapIO.write( "goo.map", dist );
```

Level-set Segmentation (1/7)

We need two map data structures, one for the segmentation results and one for the medical image,

```
Map<OpenTissue::scalar> phi;
Map<unsigned short> U;
```

Observe that the first map is a scalar map, the last one is a 16 bit voxel map. The Chan-Vese level set segmentation is declared as follows

```
#include <OpenTissue/map/util/ChanVese.h>
ChanVese<OpenTissue::scalar,unsigned short> chanVese;
```

Notice that two template arguments are used

Level-set Segmentation (2/7)

We also need an iso surface extractor for visualization of the segmentation,

```
#include <OpenTissue/mc/isoSurfaceExtractor.h>
IsoSurfaceExtractor<OpenTissue::scalar> extractor;
```

Notice that the template argument. We need a tool for initialization of the scalar maps

```
#include <OpenTissue/map/util/fillMap.h>
FillMap<OpenTissue::scalar> filler;
```

Level-set Segmentation (3/7)

We want a visualization of the 3D medical image,

```
#include <OpenTissue/map/util/cutViewUtility.h>
CutViewUtility<unsigned short> cutview;
```

Observe the template argument. The cut-view utility class need three bitmaps

```
#include <OpenTissue/bitmap/bitmap.h>
Bitmap xcut,ycut,zcut;
int xcutvalue;
int ycutvalue;
int zcutvalue;
```

Level-set Segmentation (4/7)

Now we allocate and read in the medial 3D image

```
Vector3 minCoord(0,0,0);
Vector3 maxCoord(I,J,K);
U.create(minCoord,maxCoord,I,J,K);
RawMapIO<unsigned short> rawIO;
rawIO.read(rawfile,U);
```

Then we allocate and initialize maps for the segmentation

```
minCoord.set(-(U.I/2.0),-(U.J/2.0),-(U.K/2.0));
maxCoord.set(U.I/2.0,U.J/2.,U.K/2.);
phi.create(minCoord,maxCoord,U.I,U.J,U.K);
filler.blockify(phi,-100.,100.,5,11);
```

Level-set Segmentation (5/7)

Finally we allocate bitmaps for the cutview visualization

```
xcut.create(U.J,U.K);
ycut.create(U.I,U.K);
zcut.create(U.I,U.J);
```

Level-set Segmentation (6/7)

To interact with the application we have added a key-event handler

```
void Application::key(char action)
{
  switch (action)
  {
  case 's':
      chanVese.run(phi,U,0.01);
      break;
  case 'e':
      extractor.run(phi,0);
    break;
  }
}
```

Level-set Segmentation (7/7)

In the display method/call-back function we take care of the visualization

```
void Application::display(void){
  int maxTri = extractor.getNumTriangles();
  glBegin(GL_TRIANGLES);
  for(int idxTri=0; idxTri<maxTri; ++idxTri){
    int idx0 = extractor.getNode0(idxTri);...
    GLfloat x0 = extractor.getXcoord(idx0);...
    GLfloat nx0 = ...
    glNormal3f(nx0, ny0, nz0);glVertex3f(x0,y0,z0);
    glNormal3f(nx1, ny1, nz1);glVertex3f(x1,y1,z1);
    glNormal3f(nx2, ny2, nz2);glVertex3f(x2,y2,z2);
  }
  glEnd();
  cutview.getViewCutBitmaps(U,xcutvalue,ycutvalue,zcutvalue,xcut,ycut,zcu
  cutview.drawViewCut(U,xcutvalue,ycutvalue,zcutvalue,xcut,ycut,zcut);
};
```

Virtual Trackball (1/4)

The virtual trackball is defined in the header

```
#include <OpenTissue/trackball/Trackball.h>
bool bTrackballModeOn = false;
Trackball trackball;
```

The transformation given by the Trackball is easily extracted and applied to the current matrix stack in openGL

```
void glTrackballMatrix( void ){
  const Transform & Tmp = trackball.Transformation();
  int glindex = 0; GLdouble glmatrix[ 16 ];
  for ( unsigned int col = 0; col < 4; ++col )
    for ( unsigned int row = 0; row < 4; ++row )
      glmatrix[ glindex++ ] = Tmp[ row ][ col ];
  glMultMatrixd( glmatrix );
};
```

Virtual Trackball (2/4)

Tell trackball when dragging is started and stopped

```
void mouse( int Button, int State, int Xmouse, int Ymouse ){
  if ( bTrackballModeOn ){
    Real Xnorm = Xmouse;Real Ynorm = Ymouse;
    normalize( Xnorm, Ynorm );
    trackball.EndDrag( Xnorm, Ynorm );
    bTrackballModeOn = false;
  }
  if ( State == GLUT_DOWN ){
    Real Xnorm = Xmouse;Real Ynorm = Ymouse;
    normalize( Xnorm, Ynorm );
    trackball.BeginDrag( Xnorm, Ynorm );
    bTrackballModeOn = true;
  }
  glutPostRedisplay();
};
```

Notice the auxiliary variable to keep track of whether a dragging motion is in progress

Virtual Trackball (3/4)

Also tell the trackball when dragging is being done

```
void motion( int Xmouse, int Ymouse )
{
  if ( bTrackballModeOn )
  {
    Real Xnorm = Xmouse;
    Real Ynorm = Ymouse;
    normalize( Xnorm, Ynorm );
    trackball.Drag( Xnorm, Ynorm );
  }
  glutPostRedisplay();
};
```

Virtual Trackball (4/4)

All that is left is to apply the trackball transformation

```
void display()
{
  glClear( ... );
  glMatrixMode( GL_MODELVIEW );
  glLoadIdentity();
  gluLookAt( eyex, eyey, eyez, ... );
  glTrackballMatrix();
  // .... Do your own rendering here ...
  glFinish();
  glutSwapBuffers();
};
```

Thanks to our Contributors

A small incomplete list of some of the people who made contributions to the OpenTissue project includes

- Micky K Christensen (DIKU): C++ Operators on Math Classes.
- Micky K Christensen (DIKU) and Anders Fleron (DIKU): PUG XML.
- Knud Henriksen (DIKU): Trackball.
- Henrik Dohlmann (DIKU): RenderTexture, Isosurface Extractor, DeVIL support, KDE support.
- Kenny Erleben (DIKU): Bounding Volume Hierarchies, VClip, GJK, Multibody Dynamics, Particle System, B-Splines.
- Kenny Erleben (DIKU) and Henrik Dohlman (DIKU): Map, Mesh, and TetraMesh data structures, div. tools and utilities, Finite Element Method (Quasi static stress strain), Volume Visualization using View Aligned Slabbing with 12 bit preintegration.
- Niels Boldt (DIKU): Gauss Seidel, Particle System Solid Mesh.
- Bjarke Jakobsen (IMM, DTU): Brick Volume Render.