Корж Оксана Васильевна

Московский Государственный Университет им. М.В.Ломоносова
Факультет Вычислительной Математики и Кибернетики

Визуализация научных данных на суперкомпьютерах

Часть 4

Москва – 2011

План презентации

- **Библиотека визуализации VTK**
- **http://www.vtk.org/VTK/help/examplecode.html**

Визуализация на Blue Gene/P

Возможности визуализации на комплексе Blue Gene /P в МГУ

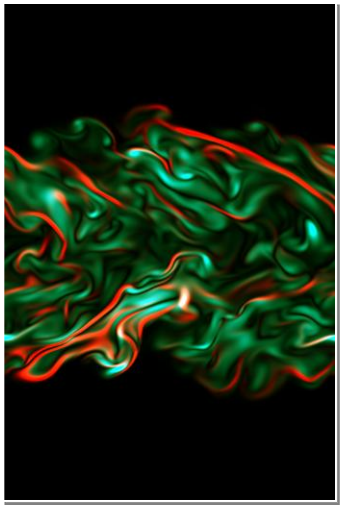
Визуализация на Blue Gene/P

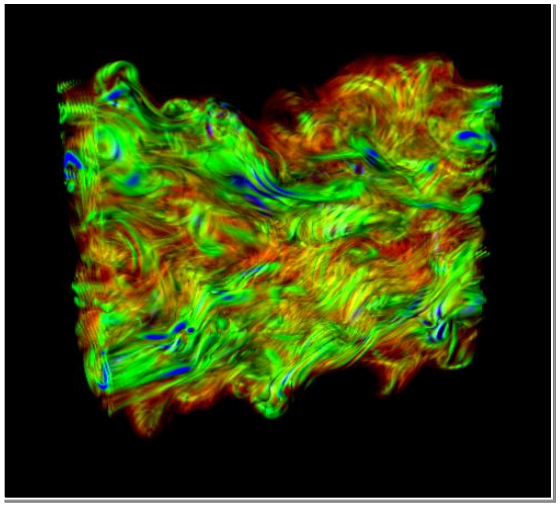
vtk

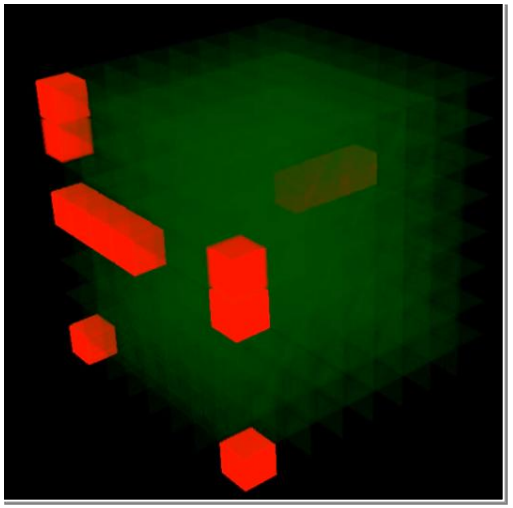
MSU_VIZ

ParaView
Parallel Visualization Application

VisIT



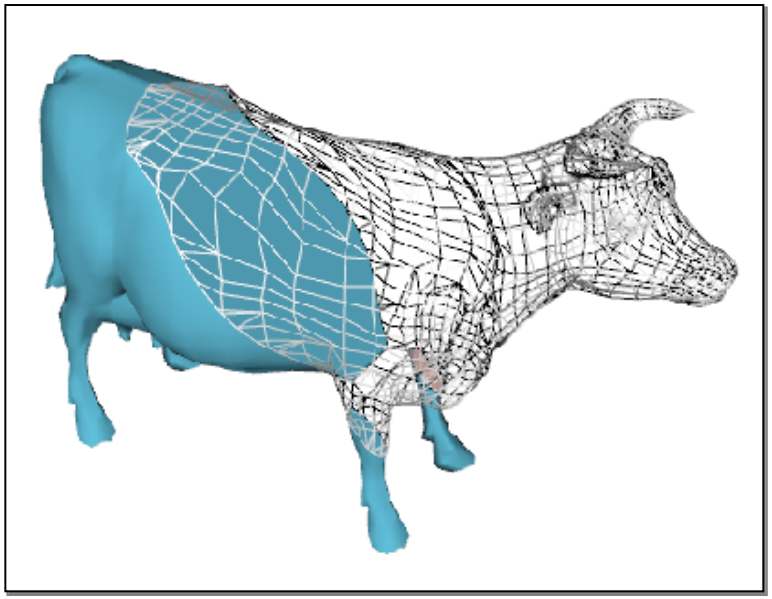


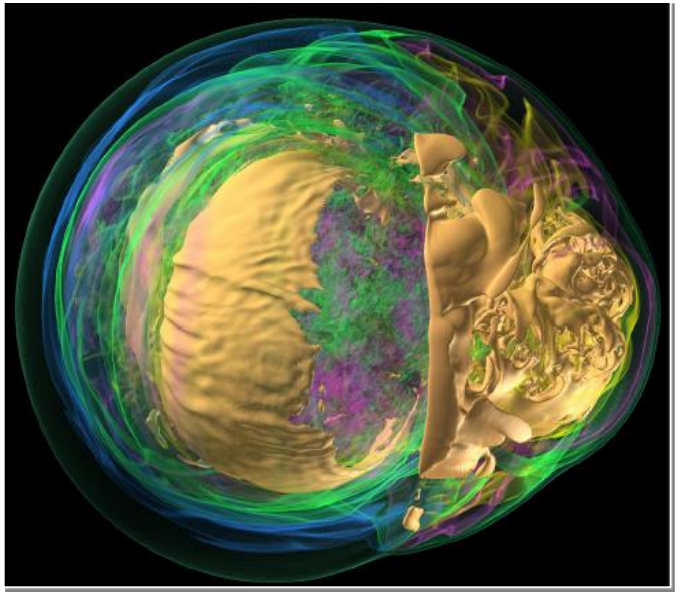


Библиотека визуализации VTK

VTK: обзор основных возможностей библиотеки

- **Visualization ToolKit (VTK) – это свободно распространяемая библиотека C++ классов для построения трехмерной графики и визуализации. Эта библиотека представляет собой объектно-ориентированный инструментарий для визуализации данных.**
- **В системе реализовано два подхода к визуализации данных. Первый подход – это графическая модель данных, которая является абстрактной моделью трехмерной графики. Второй подход – это визуализационная модель, которая представляет собой управляемый потоком данных процесс визуализации.**





Библиотека визуализации VTK

VTK: обзор основных возможностей библиотеки

- **Основные объекты графической модели:**
- **- Render Master – координирует устройство независимые методы и создает окно трассировки;**
- **- Render Window (окно трассировки) – управляет окном на экране дисплея, один или более трассировщиков рисуют в этом окне при создании кадра;**
- **- Renderer (трассировщик) – управляет процессом трассировки;**
- **- Light (источник света) – освещает акторов а сцене;**

Библиотека визуализации VTK

VTK: обзор основных возможностей библиотеки

- Camera (камера) – определяет точку обзора, точку фокуса и другие характеристики камеры;

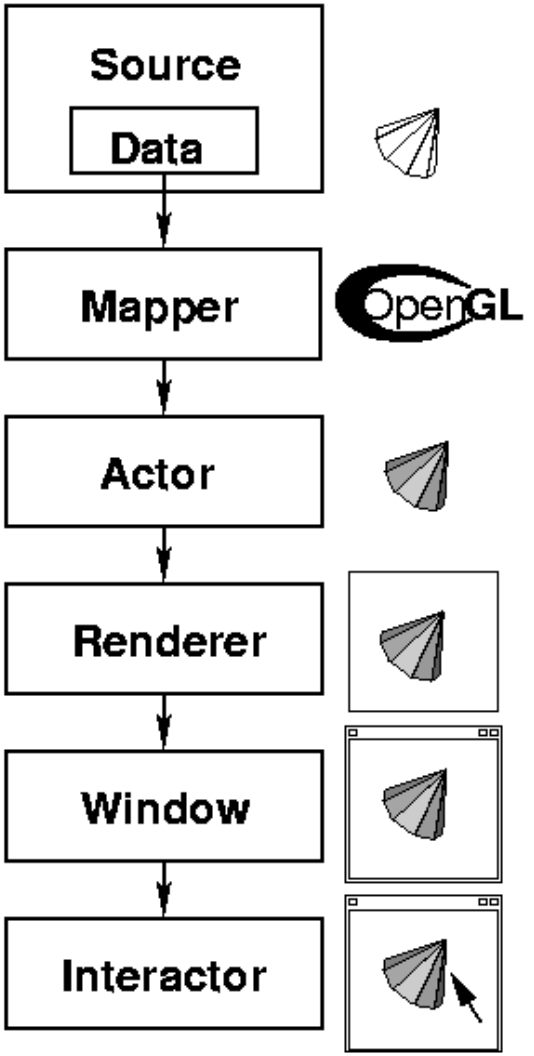
- Actor (актор) – объект, который прорисовывается трассировщиком в сцене;

- Property (свойство) – содержит характеристики актора, которые используются при трассировке, – цвет, отражающая способность, текстура, тип прорисовки, стиль затенения;

- Mapper (карта) – содержит геометрические характеристики акторов, карту положения объектов в сцене.

Библиотека визуализации VTK

VTK: схема обработки графики на питоне



Either reads the data from a file or creates the data from scratch.

Moves the data from VTK into OpenGL.

For setting colors, surface properties, and the position of the object.

The rectangle of the computer screen that VTK draws into.

The window, including title bar and decorations.

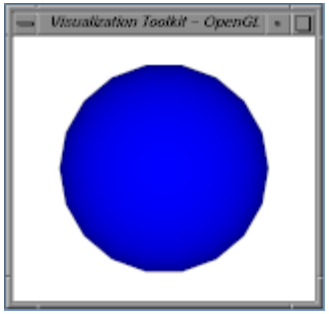
Allows the mouse to be used to interact wth the data.

```
from vtkpython import *

cone = vtkConeSource()
cone.SetResolution(10)

coneMapper = vtkPolyDataMapper()
coneMapper.SetInput(cone.GetOutput())

coneActor = vtkActor()
coneActor.SetMapper(coneMapper)

ren = vtkRenderer()
ren.AddActor(coneActor)

renWin = vtkRenderWindow()
renWin.SetWindowName("Cone")
renWin.SetSize(300,300)
renWin.AddRenderer(ren)

iren = vtkRenderWindowInteractor()
iren.SetRenderWindow(renWin)
iren.Initialize()
iren.Start()
```

Библиотека визуализации VTK

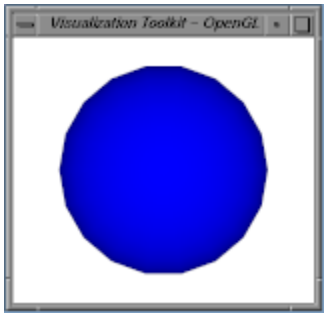
VTK: обзор основных возможностей библиотеки

```
#include "vtkSphereSource.h"
#include "vtkPolyDataMapper.h"
#include "vtkProperty.h"
#include "vtkActor.h"
#include "vtkRenderWindow.h"
#include "vtkRenderer.h"
#include "vtkRenderWindowInteractor.h"
```



Библиотека визуализации VTK

VTK: обзор основных возможностей библиотеки

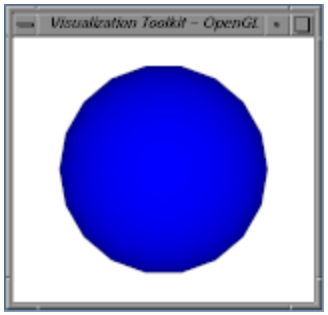



```
int main ()
{
  // create sphere geometry
  vtkSphereSource *sphere = vtkSphereSource::New();
  sphere->SetRadius(1.0);
  sphere->SetThetaResolution(18);
  sphere->SetPhiResolution(18);
  // map to graphics library
  vtkPolyDataMapper *map = vtkPolyDataMapper::New();
  map->SetInput(sphere->GetOutput());
  // actor coordinates geometry, properties, transformation
  vtkActor *aSphere = vtkActor::New();
  aSphere->SetMapper(map);
  aSphere->GetProperty()->SetColor(0,0,1); // sphere color blue
```

Библиотека визуализации VTK

VTK: обзор основных возможностей библиотеки

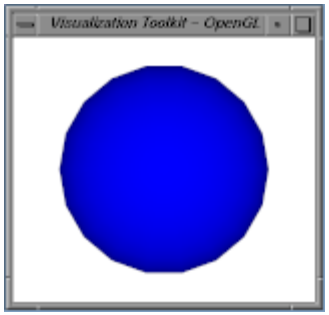
```
  vtkRenderer *ren1 = vtkRenderer::New();
  vtkRenderWindow *renWin = vtkRenderWindow::New();
  renWin->AddRenderer(ren1);
  // an interactor
  vtkRenderWindowInteractor *iren =
vtkRenderWindowInteractor::New();
  iren->SetRenderWindow(renWin);
  // add the actor to the scene
  ren1->AddActor(aSphere);
  ren1->SetBackground(1,1,1); // Background color white
  // render an image (lights and cameras are created
automatically)
  renWin->Render();
  // begin mouse interaction
   iren->Start();
```



Библиотека визуализации VTK

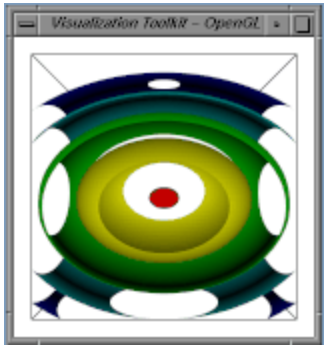
VTK: обзор основных возможностей библиотеки

```
  // release memory and return
  sphere->Delete();
  map->Delete();
  aSphere->Delete();
  ren1->Delete();
  renWin->Delete();
  iren->Delete();
  return EXIT_SUCCESS;
}
```



Библиотека визуализации VTK

VTK: обзор основных возможностей библиотеки

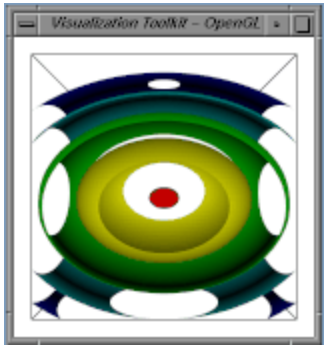



```
// F(x,y,z) = 0.5*x^2 + 1.0*y^2 + 0.2*z^2 + 0.1*y*z + 0.2*y

#include "vtkQuadric.h"
#include "vtkSampleFunction.h"
#include "vtkContourFilter.h"
#include "vtkOutlineFilter.h"
#include "vtkPolyDataMapper.h"
#include "vtkActor.h"
#include "vtkProperty.h"
#include "vtkRenderWindow.h"
#include "vtkRenderer.h"
#include "vtkRenderWindowInteractor.h"
#include "vtkImageData.h"
```

Библиотека визуализации VTK

VTK: обзор основных возможностей библиотеки

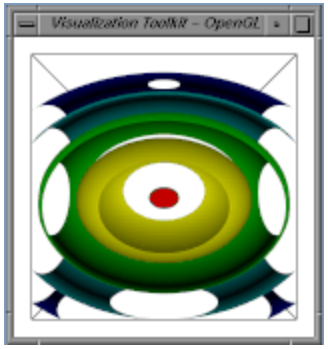
```
int main ()
{
  // -- create the quadric function object --
  // create the quadric function definition
  vtkQuadric *quadric = vtkQuadric::New();
  quadric->SetCoefficients(.5,1,.2,0,.1,0,0,.2,0,0);
  // sample the quadric function
  vtkSampleFunction *sample = vtkSampleFunction::New();
  sample->SetSampleDimensions(50,50,50);
  sample->SetImplicitFunction(quadric);
  // Create five surfaces F(x,y,z) = constant between range specified
  vtkContourFilter *contours = vtkContourFilter::New();
  contours->SetInput(sample->GetOutput());
  contours->GenerateValues(5, 0.0, 1.2);
  // map the contours to graphical primitives
  vtkPolyDataMapper *contMapper = vtkPolyDataMapper::New();
  contMapper->SetInput(contours->GetOutput());
  contMapper->SetScalarRange(0.0, 1.2);
  // create an actor for the contours
  vtkActor *contActor = vtkActor::New();
  contActor->SetMapper(contMapper);
```



Библиотека визуализации VTK

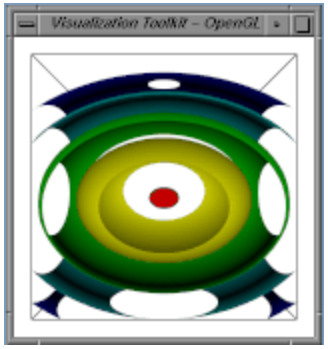
VTK: обзор основных возможностей библиотеки

```
// -- create a box around the function to indicate the sampling volume --
// create outline
vtkOutlineFilter *outline = vtkOutlineFilter::New();
outline->SetInput(sample->GetOutput());
// map it to graphics primitives
vtkPolyDataMapper *outlineMapper = vtkPolyDataMapper::New();
outlineMapper->SetInput(outline->GetOutput());
// create an actor for it
vtkActor *outlineActor = vtkActor::New();
outlineActor->SetMapper(outlineMapper);
outlineActor->GetProperty()->SetColor(0,0,0);
```



Библиотека визуализации VTK

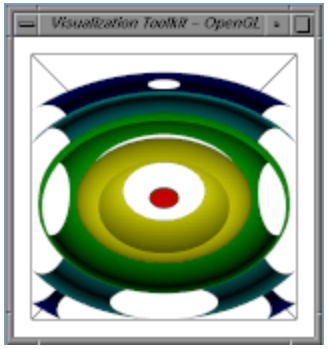
VTK: обзор основных возможностей библиотеки




```
  // -- render both of the objects --
  // a renderer and render window
  vtkRenderer *ren1 = vtkRenderer::New();
  vtkRenderWindow *renWin = vtkRenderWindow::New();
  renWin->AddRenderer(ren1);
  // an interactor
  vtkRenderWindowInteractor *iren = vtkRenderWindowInteractor::New();
  iren->SetRenderWindow(renWin);
// add the actors to the scene
  ren1->AddActor(contActor);
  ren1->AddActor(outlineActor);
  ren1->SetBackground(1,1,1); // Background color white
  // render an image (lights and cameras are created automatically)
  renWin->Render();
  // begin mouse interaction
  iren->Start();
```

Библиотека визуализации VTK

VTK: обзор основных возможностей библиотеки




```
  // release memory and return
  quadric->Delete();
  sample->Delete();
  contours->Delete();
  contMapper->Delete();
  contActor->Delete();
  outline->Delete();
  outlineMapper->Delete();
  outlineActor->Delete();
  ren1->Delete();
  renWin->Delete();
  iren->Delete();
  return EXIT_SUCCESS;
}
```

Библиотека визуализации VTK

VTK: работа с изображениями

```
#include <vtkSmartPointer.h>
#include <vtkImageViewer2.h>
#include <vtkJPEGReader.h>
#include <vtkRenderWindow.h>
#include <vtkRenderWindowInteractor.h>
#include <vtkRenderer.h>
int main(int argc, char* argv[])
{
	//Verify input arguments
	if ( argc != 2 ) {
		std::cout << "Usage: " << argv[0] << " Filename(.jpeg)" << std::endl;
		return EXIT_FAILURE;
	}
```

Библиотека визуализации VTK

VTK: работа с изображениями

```
        //Read the image
        vtkSmartPointer<vtkJPEGReader> jpegReader =
vtkSmartPointer<vtkJPEGReader>::New();
        jpegReader->SetFileName ( argv[1] );

        // Visualize
        vtkSmartPointer<vtkImageViewer2> imageViewer =
vtkSmartPointer<vtkImageViewer2>::New();
        imageViewer->SetInput( jpegReader->GetOutput() );
        vtkSmartPointer<vtkRenderWindowInteractor> renderWindowInteractor =

                vtkSmartPointer<vtkRenderWindowInteractor>::New();
        imageViewer->SetupInteractor(renderWindowInteractor);
        imageViewer->Render();
        imageViewer->GetRenderer()->ResetCamera();
        imageViewer->Render();
        renderWindowInteractor->Start();
        return EXIT_SUCCESS;
}
```

Библиотека визуализации VTK

VTK: обзор основных возможностей библиотеки

```
#include <vtkImageData.h>
#include <vtkJPEGWriter.h>
#include <vtkSmartPointer.h>
#include <vtkImageCanvasSource2D.h>
#include <vtkImageCast.h>

int main ( int argc, char *argv[] )
{
  std::string outputFilename;
  if ( argc > 1 )
    {
    outputFilename = argv[1];
    }
  else
    {
    outputFilename = "output.jpg";
    }
```

Библиотека визуализации VTK

VTK: обзор основных возможностей библиотеки

```
int extent[6] = {0, 99, 0, 99, 0, 0};
  vtkSmartPointer<vtkImageCanvasSource2D> imageSource =
      vtkSmartPointer<vtkImageCanvasSource2D>::New();
  imageSource->SetExtent(extent);
  imageSource->SetScalarTypeToUnsignedChar();
  imageSource->SetNumberOfScalarComponents(3);
  imageSource->SetDrawColor(127, 45, 255);
  imageSource->FillBox(0, 99, 0, 99);
  imageSource->SetDrawColor(255,255,255);
  imageSource->FillBox(40, 70, 20, 50);
  imageSource->Update();
```

Библиотека визуализации VTK

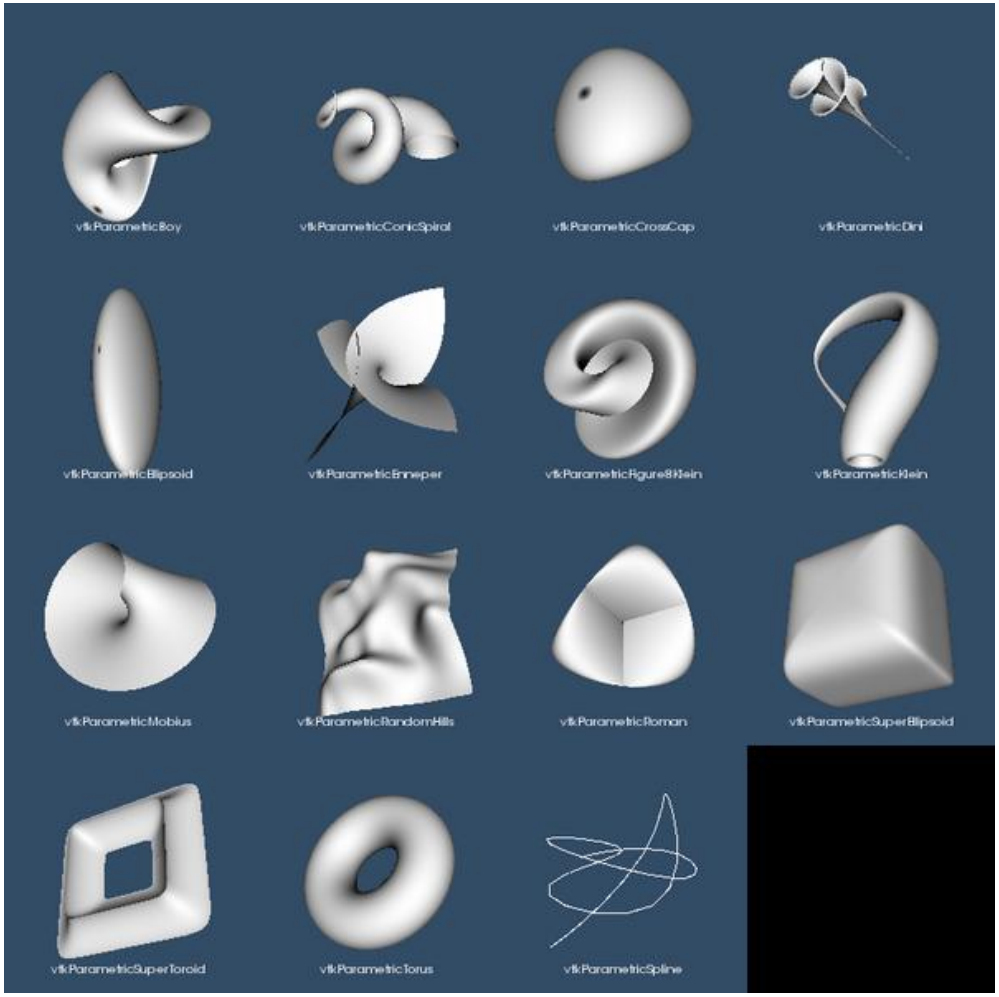
VTK: обзор основных возможностей библиотеки

```
  vtkSmartPointer<vtkImageCast> castFilter =
      vtkSmartPointer<vtkImageCast>::New();
  castFilter->SetOutputScalarTypeToUnsignedChar ();
  castFilter->SetInputConnection(imageSource->GetOutputPort());
  castFilter->Update();

  vtkSmartPointer<vtkJPEGWriter> writer =
    vtkSmartPointer<vtkJPEGWriter>::New();
  writer->SetFileName(outputFilename.c_str());
  writer->SetInputConnection(castFilter->GetOutputPort());
  writer->Write();

  return EXIT_SUCCESS;
}
```

Библиотека визуализации VTK

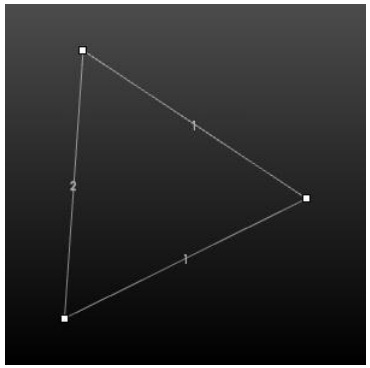
VTK: параметрические кривые

```
#include <vtkParametricTorus.h>
#include <vtkParametricBoy.h>
#include <vtkParametricConicSpiral.h>
#include <vtkParametricCrossCap.h>
#include <vtkParametricDini.h>
#include <vtkParametricEllipsoid.h>
#include <vtkParametricEnneper.h>
#include <vtkParametricFigure8Klein.h>
#include <vtkParametricKlein.h>
#include <vtkParametricMobius.h>
#include <vtkParametricRandomHills.h>
#include <vtkParametricRoman.h>
#include <vtkParametricSpline.h>
#include <vtkParametricSuperEllipsoid.h>
#include <vtkParametricSuperToroid.h>
#include <vtkParametricTorus.h>
```



Библиотека визуализации VTK

VTK: обзор основных возможностей библиотеки

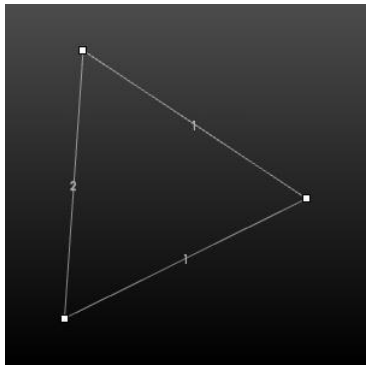



```
#include <vtkSmartPointer.h>
 #include <vtkDataSetAttributes.h>
 #include <vtkDoubleArray.h>
#include <vtkMutableUndirectedGraph.h>
#include <vtkCircularLayoutStrategy.h>
#include <vtkDoubleArray.h>
#include <vtkGraphLayoutView.h>
 #include <vtkIntArray.h>
 #include <vtkMutableUndirectedGraph.h>
 #include <vtkRenderWindowInteractor.h>
int main(int, char *[])
 {
      vtkSmartPointer<vtkMutableUndirectedGraph> g =
      vtkSmartPointer<vtkMutableUndirectedGraph>::New();
      // Create 3 vertices
      vtkIdType v1 = g->AddVertex();
      vtkIdType v2 = g->AddVertex();
      vtkIdType v3 = g->AddVertex();
      // Create a fully connected graph
      g->AddEdge(v1, v2); g->AddEdge(v2, v3); g->AddEdge(v1, v3);
```

Библиотека визуализации VTK

VTK: обзор основных возможностей библиотеки

```
// Create the edge weight array
vtkSmartPointer<vtkDoubleArray> weights = vtkSmartPointer<vtkDoubleArray>::New();
weights->SetNumberOfComponents(1);
weights->SetName("Weights");
// Set the edge weights
weights->InsertNextValue(1.0);
weights->InsertNextValue(1.0);
weights->InsertNextValue(2.0);
// Add the edge weight array to the graph
g->GetEdgeData()->AddArray(weights);
```




```
 std::cout << "Number of Weights: " << vtkDoubleArray::SafeDownCast( g->GetEdgeData()-
>GetArray("Weights"))->GetNumberOfTuples() << std::endl;
for(vtkIdType i = 0; i < weights->GetNumberOfTuples(); i++) { double w = weights-
>GetValue(i); std::cout << "Weight " << i << " : " << w << std::endl; }

vtkSmartPointer<vtkGraphLayoutView> graphLayoutView =
vtkSmartPointer<vtkGraphLayoutView>::New();
graphLayoutView->AddRepresentationFromInput(g);
graphLayoutView->SetEdgeLabelVisibility(true);
graphLayoutView->SetEdgeLabelArrayName("Weights");
graphLayoutView->ResetCamera();
graphLayoutView->Render();
graphLayoutView->GetInteractor()->Start();
return EXIT_SUCCESS;
}
```

Библиотека визуализации VTK

VTK: обзор основных возможностей библиотеки

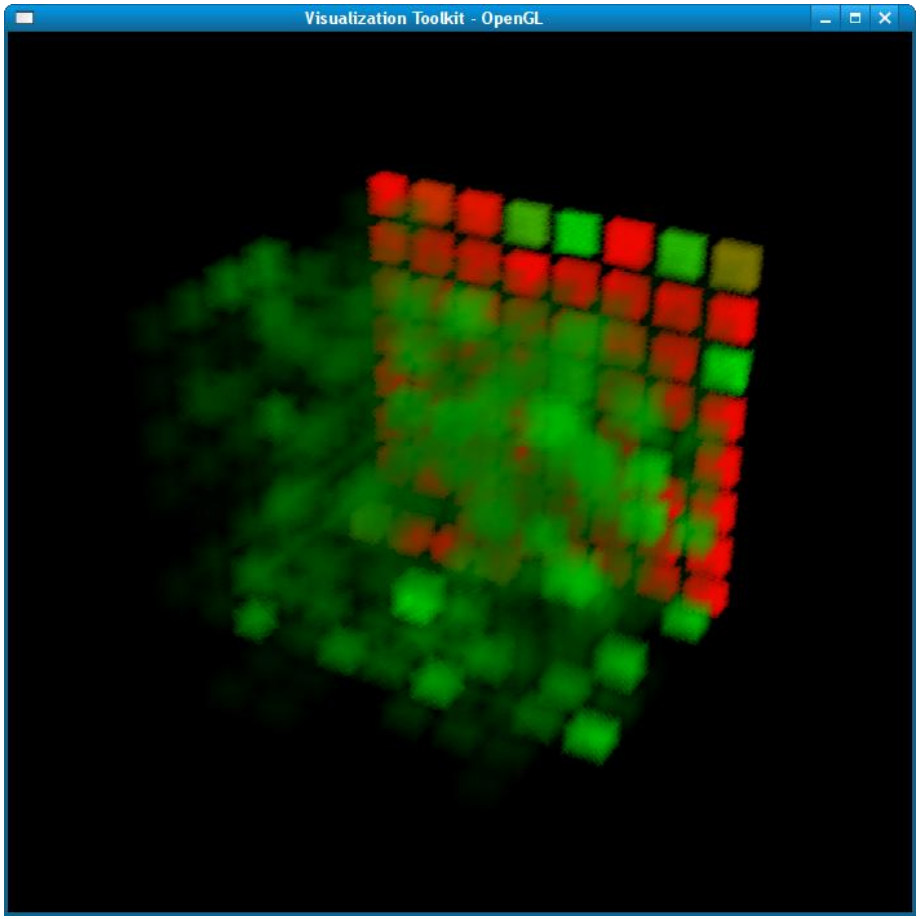
- **Предусмотрено пять типов данных: vtkPolyData (полигональные модели), vtkStructuredPoints (структурированный набор точек), vtkStructuredGrid (структурированная решетка), vtkUnstructuredGrid (неструктурированная решетка), vtkPointSet (неструктурированные точки). Основой представления данных является концепция ячеек. Каждый набор данных состоит из одной или более ячейки, каждая ячейка – это графические примитив при визуализации**

- **Визуализационная модель основана на data-flow парадигме. В такой парадигме модули соединены между собой в одну сеть и осуществляют алгоритмические операции над данными, которые представляют собой потоки в сетях. Преимуществом такого подхода является его гибкость и быстрая адаптация к данным различного типа или новой реализации алгоритма.**

- **Визуализационная модель состоит из двух основных типов объектов: обрабатывающих объектов и объектов данных. Обрабатывающие объекты соответствуют алгоритмическим модулям, объекты данных – потоки в сетях. Обрабатывающие объекты в свою очередь подразделяются на источники, фильтры и отображения. Источника инициализируют потоки данных. Фильтры осуществляют заданное преобразование над входными данными. Отображения являются окончанием путей в сети.**

Библиотека визуализации VTK

VTK: обзор основных возможностей библиотеки



Библиотека визуализации VTK

VTK: обзор основных возможностей библиотеки

```
vtkRenderer *ren1 = vtkRenderer::New();
vtkRenderWindow *renWin = vtkRenderWindow::New();
vtkRendererSource * renSrc = vtkRendererSource::New();
vtkImageData * Image = vtkImageData::New();
vtkLight * light1 = vtkLight::New();
renWin->AddRenderer(ren1);
//renWin->OffScreenRenderingOn();
```

Библиотека визуализации VTK

VTK: обзор основных возможностей библиотеки

```
renSrc->SetInput(ren1);
Image->SetDimensions(128,128,128);
Image->SetScalarTypeToUnsignedShort();
Image->SetNumberOfScalarComponents(1);
Image->AllocateScalars();
Image->SetSpacing(1, 1, 1);
Image->SetOrigin(0.0, 0.0, 0.0);
light1->SetColor(1,1,1);
```

Библиотека визуализации VTK

VTK: обзор основных возможностей библиотеки

```
unsigned short * ptr;
ptr = (unsigned short *)Image->GetScalarPointer();

for (int i=0; i<H*W*D; i++)
{
*ptr++= 128;
}
```

Библиотека визуализации VTK

VTK: обзор основных возможностей библиотеки

```
vtkPiecewiseFunction *oTFun = vtkPiecewiseFunction::New();
oTFun->AddSegment(0, 0.0,255,0.1);

vtkColorTransferFunction * cTFun = vtkColorTransferFunction::New();
cTFun->SetColorSpaceToRGB();
cTFun->AddRGBPoint(0, 0, 1, 0);
cTFun->AddRGBPoint(100, 1, 1, 0);
cTFun->AddRGBPoint(200, 1, 0, 0);
```

Библиотека визуализации VTK

VTK: обзор основных возможностей библиотеки

```
vtkVolumeProperty *volumeProperty = vtkVolumeProperty::New();
volumeProperty->SetColor(cTFun);
volumeProperty->SetScalarOpacity(oTFun);
volumeProperty->SetInterpolationTypeToNearest();
```

Библиотека визуализации VTK

VTK: обзор основных возможностей библиотеки

```
vtkVolumeRayCastCompositeFunction *compositeFunction =
      vtkVolumeRayCastCompositeFunction::New();
vtkVolumeRayCastMapper *volumeMapper =
vtkVolumeRayCastMapper::New();
volumeMapper->SetInput(Image);
volumeMapper->SetVolumeRayCastFunction(compositeFunction);
vtkVolume *volume = vtkVolume::New();
volume->SetMapper(volumeMapper);
volume->SetProperty(volumeProperty);
```

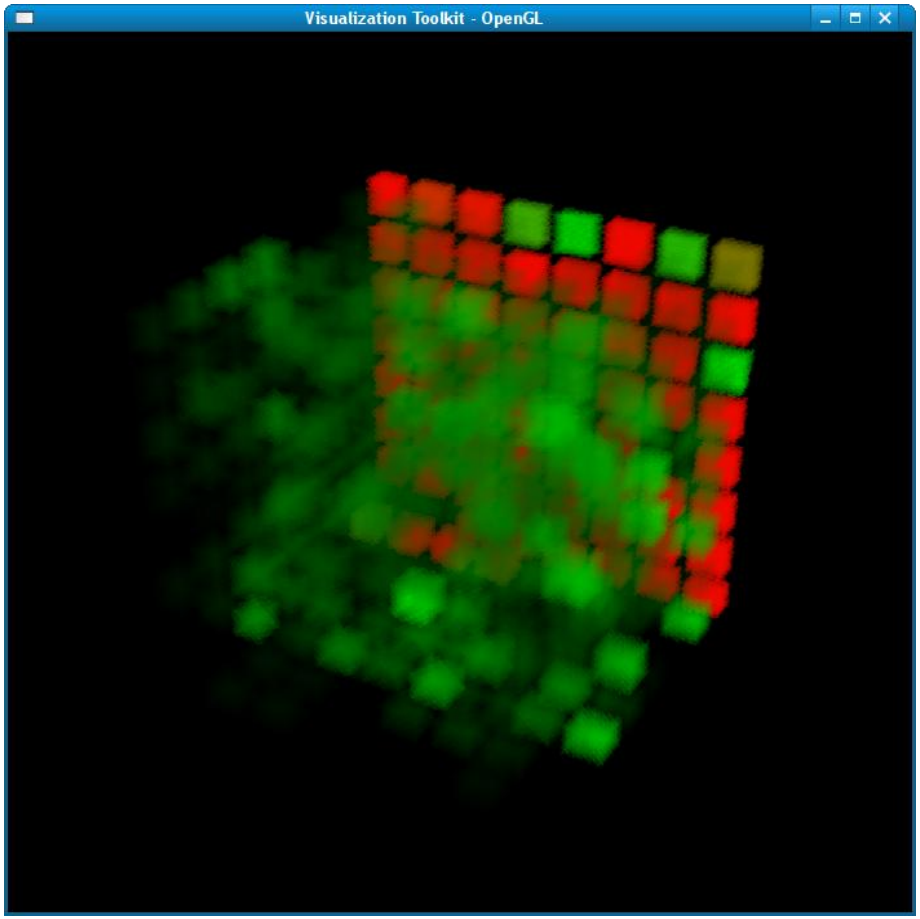
Библиотека визуализации VTK

VTK: обзор основных возможностей библиотеки

```
light1->SetPosition(200.0,300.0,400.0);
light1->SetFocalPoint(64.0,64.0,64.0);
ren1->AddVolume(volume);
ren1->AddLight(light1);
ren1->GetActiveCamera()->SetFocalPoint(64.0,64.0,64.0);
ren1->GetActiveCamera()->SetPosition(-301.0,256.0,301.0);

/* data to volume input */
renWin->SetSize(800,800);
renWin->Render();
```

Библиотека визуализации VTK

VTK: обзор основных возможностей библиотеки



Библиотека визуализации VTK

VTK: запуск на Blue Gene P

1. Каждый раз после входа в систему один раз выполнить

```
export PATH=$PATH:/home/oxanad/cmake/bin
export VTK_DIR=/home/oxanad/VTK-ffen
```

Создаем проект:

2. Создать директорию для проекта vtk_proj (mkdir vtk_proj). Зайти в нее (cd vtk_proj)

положить в нее файл school_viz.cpp с кодом программы

Библиотека визуализации VTK

VTK: запуск на Blue Gene P

3. Создать файл CMakeLists.txt со следующим содержимым

```
PROJECT (school_viz)
FIND_PACKAGE(VTK REQUIRED)
INCLUDE(${VTK_USE_FILE})
ADD_EXECUTABLE(school_viz school_viz.cpp)
TARGET_LINK_LIBRARIES(school_viz vtkRendering
    vtkVolumeRendering)
```

Библиотека визуализации VTK

VTK: запуск на Blue Gene P

компилировать следующими двумя командами

cmake . (после cmake точка через пробел)

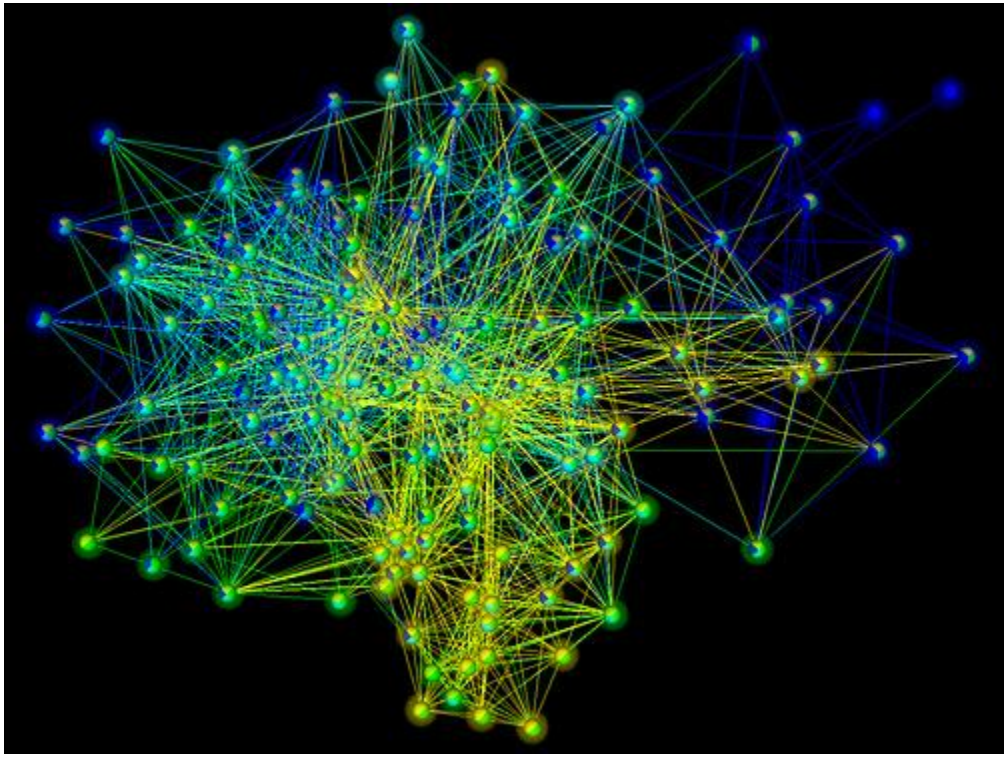
make

Запускать полученный бинарик school_viz следующим образом:

LD_LIBRARY_PATH=/home/oxanad/inst/ffmpeg/lib ./school_viz

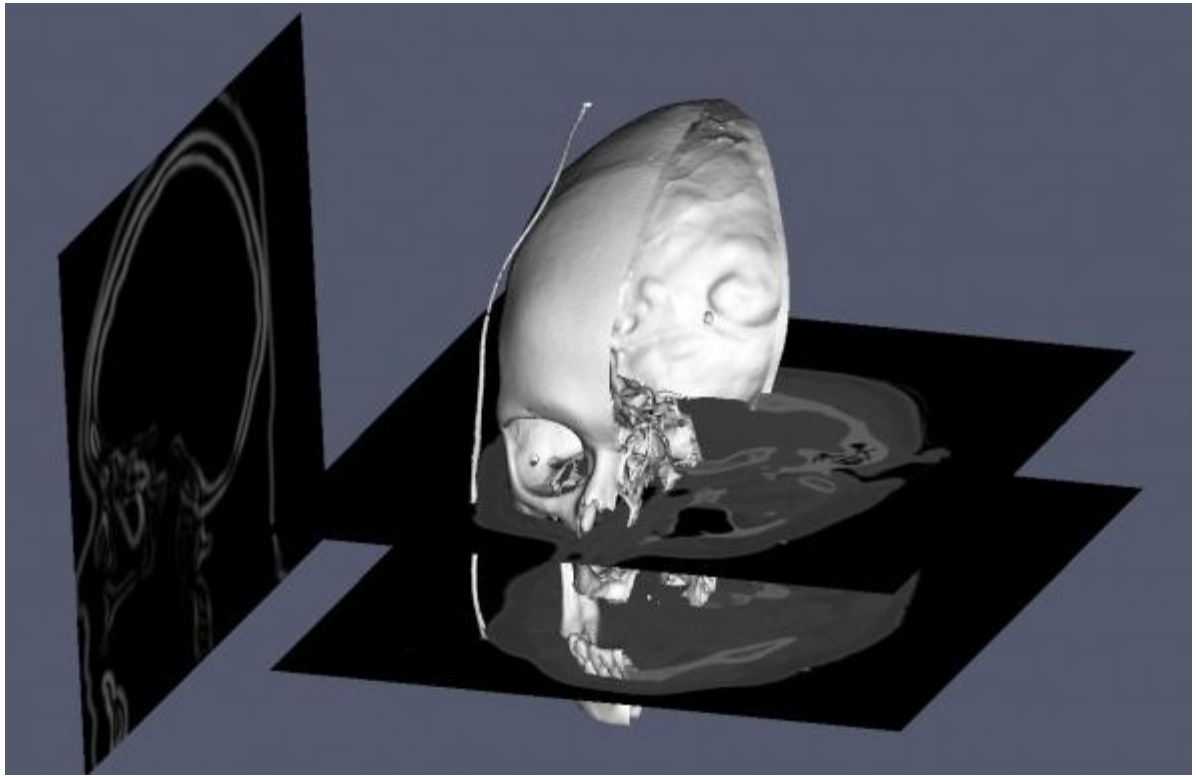
Библиотека визуализации VTK

VTK: Примеры Parallel Process Graph



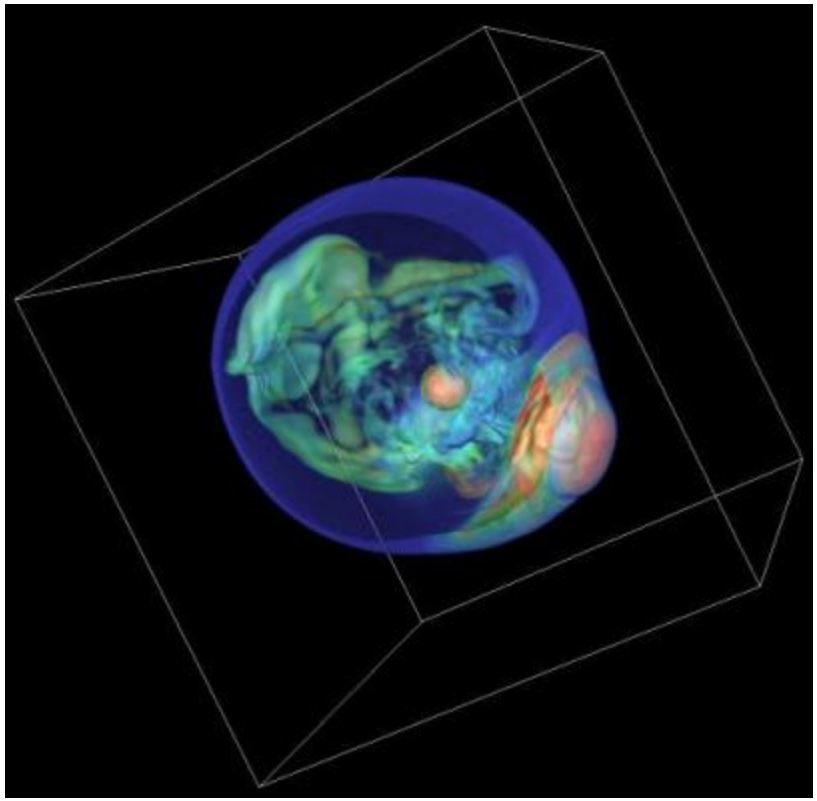
Библиотека визуализации VTK

VTK: Примеры Volume rendering and image display from the visible woman dataset



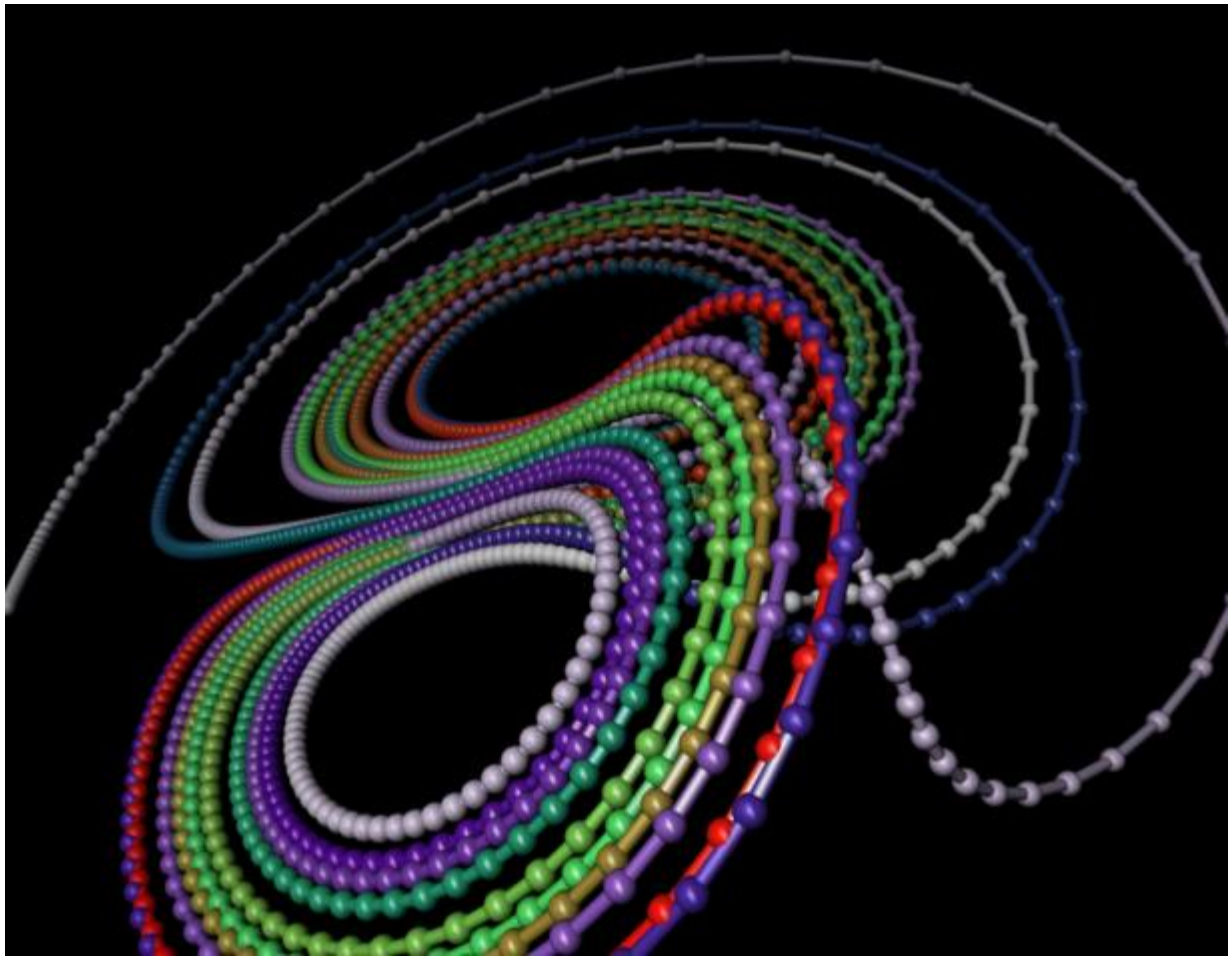
Библиотека визуализации VTK

VTK: Примеры supernova



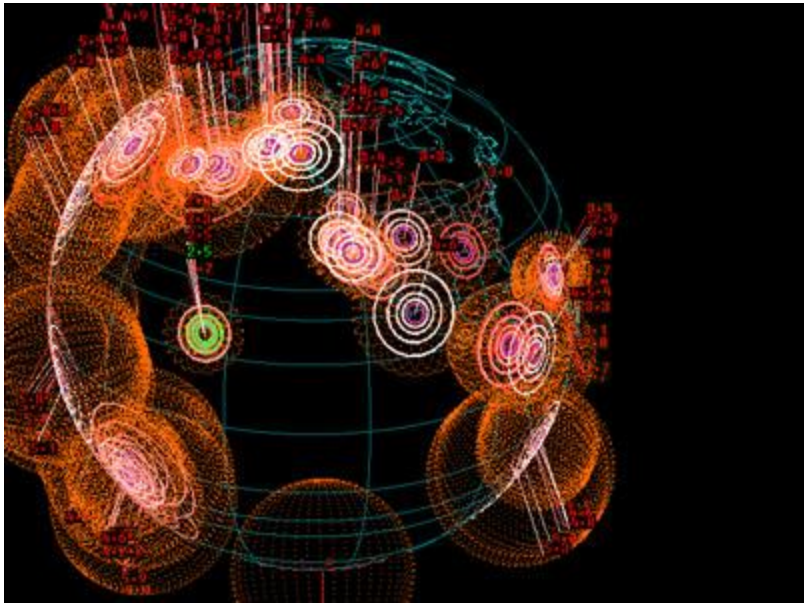
Библиотека визуализации VTK

VTK: Примеры Lorentz chaotic weather equations



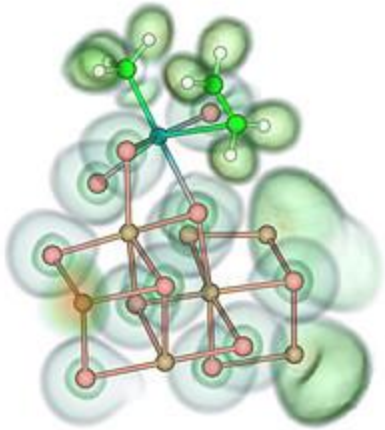
Библиотека визуализации VTK

VTK: Примеры EarthQuake 3D



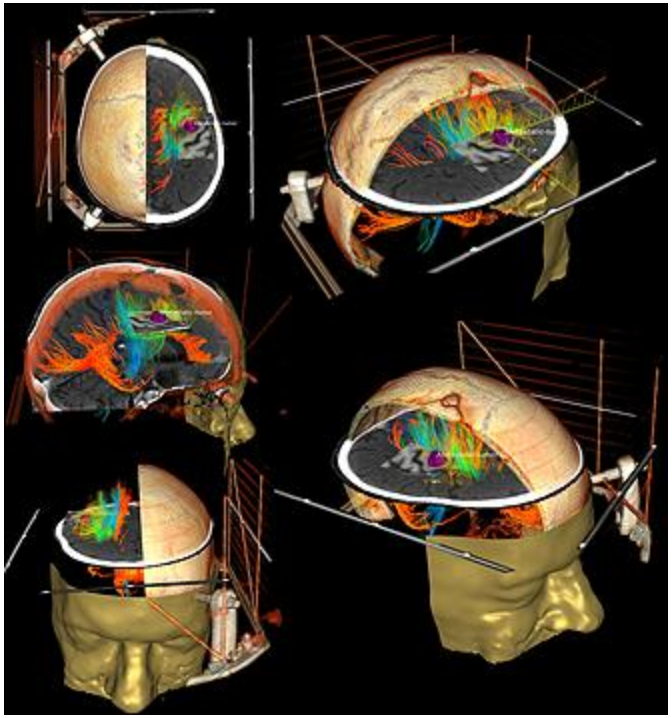
Библиотека визуализации VTK

VTK: Примеры Molecula



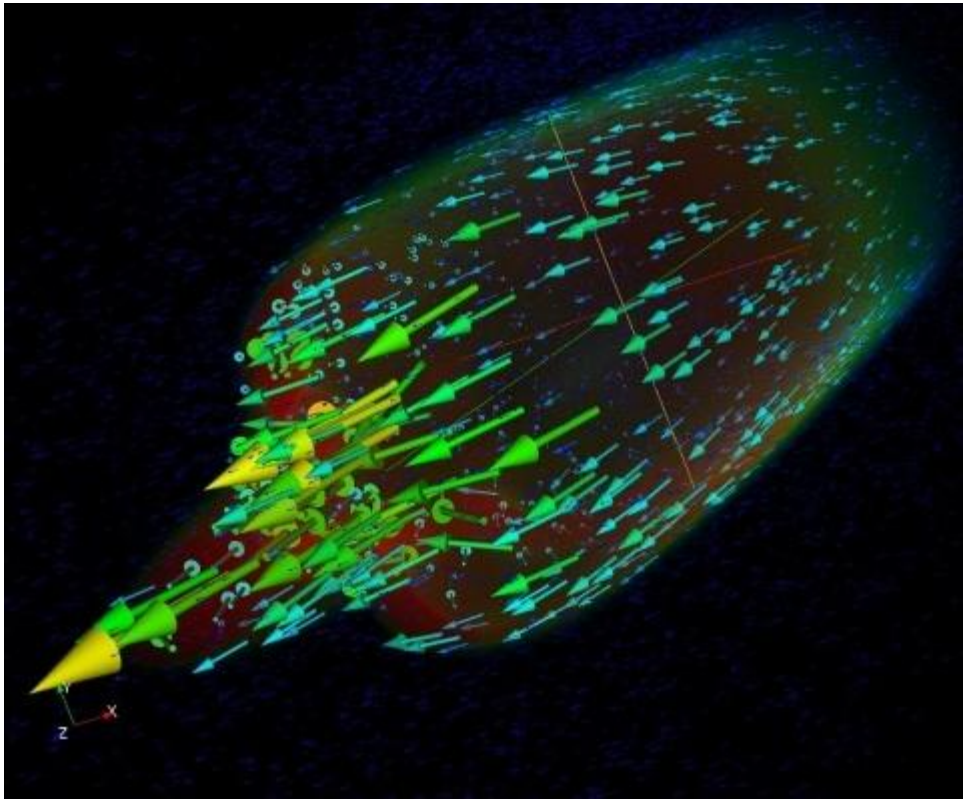
Библиотека визуализации VTK

VTK: Примеры Brain Connectivity and Tumor Visualization for Radiosurgery



Библиотека визуализации VTK

VTK: Примеры The magnetic jet



Библиотека визуализации VTK

VTK: запуск на Blue Gene P

http://www.vtk.org/VTK/help/examplecode.html

http://www.vtk.org/VTK/help/documentation.html

http://www.vtk.org/doc/nightly/html/classes.html

/home/oxanad/VTK-ffen/Examples/

Спасибо за внимание

oxanad@mail.ru