COMPSCI 716 S2 T - Assignment 3
Solution Hints

Computer Science

This assignment is worth 10% of your final mark for COMPSCI 716 S2 T.

1. Introduction (12 Marks)

(i) Read the paper "Top Scientific Visualization research Problems" (see 716 web page, Chapter 1) and write the answers to the following questions into the Word document `Ass3Answers.doc`.

(a) [2 Marks] An important research problem is to quantify the effectiveness of a visualization algorithm. In question 3 of this assignment you have to visualize a sorting algorithm. Suggest a way to evaluate the effectiveness of your visualization of the sorting algorithm, e.g. how can you measure whether your visualization is better than that of a classmate? Please explain your answer.

Solution Hint: You first have to define what "better" means. In this application the visualization is used to improve the understanding of sorting algorithms. Hence one way to measure the effectiveness of the visualization is to use a group of people and ask them after watching the visualization of the sorting algorithm questions such as "what element will be swapped in the next step" or "what does the array look like at the end of the next iteration of this sorting algorithm". Alternatively we could ask each person to describe the algorithm in pseudo code. Of course previous experiences and intelligence levels of the users must be taken into account in order to achieve a fair comparison.

(b) [2 marks] Find at least three references for techniques to visualize uncertainty on the web. Write the URLs with a short explanation of the method (1-2 sentences per URL) into the answer sheet.

Solution Hint: Use Google ;-)

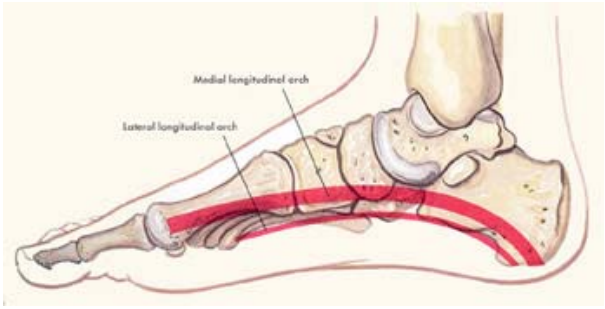
(c) [2 marks] Give three examples of applications where it is useful to display visualizations on a PDA or mobile phone.

Solution Hint: Any application where the user is mobile and is not able to carry around a large device, e.g. navigation systems (travelling, tramping, caving, military…), patient information for a doctor, information systems for managers (e.g. company information, share market data, …)

(ii) Read page 69-83 of the handout "Chapter 4 - Scientific Visualization" (see 716 web page) and write the answers to the following questions into the Word document `Ass3Answers.doc`.

(a) [2 marks] Read section 4.1 of the handout and have a look at the two images below (for colour illustrations please view the electronic copy of this handout on the 716 web page). The image on the left shows a CT (Computed Tomography) slice image of a foot and the image on the right shows an illustration of an anatomy textbook. Explain why CT images alone represent a bad visualization and what information must be added to the visualization in order to make it more informative.

Solution Hint: A CT image displays only the data for a single slice through the foot, whereas the illustration shows its entire 3D structure. We can obtain an image of the full 3D structure of the foot by visualizing the complete CT data set using direct volume rendering. Different opacity and colour transfer functions must be used for different tissue types and gradient based shading must be applied in order to emphasize tissue boundaries. The visualization can be improved by annotating it with text and markers. Markers could be integrated directly into the 3D visualization by adding them to the CD data set using clearly differentiable density values and by adjusting the transfer functions accordingly.



(b) [2 marks] Suggest techniques to visualize a 2D deformation field, which gives for each point of an object the corresponding location of the deformed object.

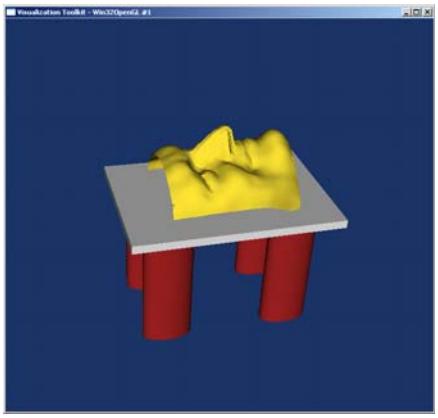
Solution Hint: Visualize the object contour together with an overlayed 2D grid fixed to it and add to this the visualization of the transformed object and grid.

(c) [2 marks] Chernoff faces (see page 94 of the handout "Chapter 4 - Scientific Visualization") utilize the brain's ability to quickly recognize small changes in facial expressions. Think about how you recognize facial expressions and then identify 4-5 basic facial features which can be use to classify facial expressions.

Solution hints: Shape of the mouth/lips (e.g. smile), Shape of the eyes (e.g. wide open when expressing surprise), shape of the eyebrows (e.g. lifting when surprised), shape of the nose (e.g. wrinkle one's nose to indicate disgust), smile lines / frown lines, posture (e.g. lowered head when depressed), movements (nod/shake head), …

2. Introduction to VTK *(10 Marks)*

[10 Mark] Download the .NET solution Ass3Q2.zip and complete the file Ass3Q2.cxx so that it renders a scene similar to the one below. The scene consists of a yellow face which lies (approximately centred) on a table. The table has a white plate and four reddish legs with an ellipsoidal cross section. The render window has a size of 300x300 pixels and the camera has an associated trackball (see the `VTK_Cone_Tutorial.zip` for an example).



```
/* ==================================
   = Sample Solution to Assignment 3 =
   = (c) Burkhard Wuensche           =
   = September 2006                  =
   ================================== */

// First include the required header files for the VTK classes we are using.
#include "vtkCylinderSource.h"
#include "vtkCubeSource.h"
#include "vtkPolyDataMapper.h"
#include "vtkRenderWindow.h"
#include "vtkRenderWindowInteractor.h"
#include "vtkCamera.h"
#include "vtkActor.h"
#include "vtkRenderer.h"
```

```
#include "vtkInteractorStyleTrackballCamera.h"
#include "vtkProperty.h"
#include "vtkTransform.h"
#include "vtkTransformPolyDataFilter.h"
#include "vtkPolyDataReader.h"
#include "vtkPolyDataNormals.h"
#include "vtkDecimatePro.h"
#include "vtkSmoothPolyDataFilter.h"

int main( int argc, char *argv[] )
{
  // Next we create an instance of vtkConeSource and set some of its
  // properties. The instance of vtkConeSource "cone" is part of a
  // visualization pipeline (it is a source process object); it produces data
  // (output type is vtkPolyData) which other filters may process.
  vtkCylinderSource *cylinder = vtkCylinderSource::New();
  cylinder->SetHeight( 3.0 );
  cylinder->SetRadius( 1.0 );
  cylinder->SetResolution( 20 );

  vtkCubeSource *cube = vtkCubeSource::New();
  cube->SetXLength(3.5);
  cube->SetYLength(0.2);
  cube->SetZLength(5);
  cube->SetCenter(1, 1.6 ,1.5);

  // In this example we terminate the pipeline with a mapper process object.
  // (Intermediate filters such as vtkShrinkPolyData could be inserted in
  // between the source and the mapper.)  We create an instance of
  // vtkPolyDataMapper to map the polygonal data into graphics primitives. We
  // connect the output of the cone souece to the input of this mapper.
  vtkTransform *cylinderTransform = vtkTransform::New();
  cylinderTransform->Identity();
  cylinderTransform->Scale(0.3, 1, 0.6);
  vtkTransformPolyDataFilter *cylinderPolyDataTransform =
                                    vtkTransformPolyDataFilter::New();
  cylinderPolyDataTransform->SetInput( cylinder->GetOutput() );
  cylinderPolyDataTransform->SetTransform(cylinderTransform);
  cylinderPolyDataTransform->Update();
  vtkPolyDataMapper *cylinderMapper = vtkPolyDataMapper::New();
  cylinderMapper->SetInput( cylinderPolyDataTransform->GetOutput() );

  vtkPolyDataMapper *cubeMapper = vtkPolyDataMapper::New();
  cubeMapper->SetInput( cube->GetOutput() );

  // Create an actor to represent the cone. The actor orchestrates rendering
  // of the mapper's graphics primitives. An actor also refers to properties
  // via a vtkProperty instance, and includes an internal transformation
  // matrix. We set this actor's mapper to be coneMapper which we created
  // above.
  vtkProperty *propertyTable = vtkProperty::New();
  propertyTable->SetColor(1.0, 0.2, 0.2);
  propertyTable->SetDiffuse(0.7);
  propertyTable->SetSpecular(0.4);
  propertyTable->SetSpecularPower(20);

  vtkActor *cylinderActor1 = vtkActor::New();
  cylinderActor1->SetMapper(cylinderMapper);
  cylinderActor1->SetPosition(0, 0, 0);
  cylinderActor1->SetProperty(propertyTable);
  vtkActor *cylinderActor2 = vtkActor::New();
  cylinderActor2->SetMapper(cylinderMapper);
  cylinderActor2->SetPosition(2, 0, 0);
  cylinderActor2->SetProperty(propertyTable);
  vtkActor *cylinderActor3 = vtkActor::New();
  cylinderActor3->SetMapper(cylinderMapper);
  cylinderActor3->SetPosition(0, 0, 3);
  cylinderActor3->SetProperty(propertyTable);
  vtkActor *cylinderActor4 = vtkActor::New();
  cylinderActor4->SetMapper(cylinderMapper);
  cylinderActor4->SetPosition(2, 0, 3);
  cylinderActor4->SetProperty(propertyTable);

  vtkActor *cubeActor = vtkActor::New();
  cubeActor->SetMapper(cubeMapper);

  // Read a data file. This originally was a Cyberware laser digitizer scan
```

```
// of Fran J.'s face. Surface normals are generated based on local geometry
// (i.e., the polygon normals surrounding eash point are averaged). We flip
// the normals because we want them to point out from Fran's face.
vtkPolyDataReader *face= vtkPolyDataReader::New();
face->SetFileName("fran_cut.vtk");
vtkSmoothPolyDataFilter *smoother = vtkSmoothPolyDataFilter::New();
smoother->SetInputConnection(face->GetOutputPort());
smoother->SetNumberOfIterations(50);
vtkPolyDataNormals *normals= vtkPolyDataNormals::New();
normals->SetInputConnection(smoother->GetOutputPort());
normals->FlipNormalsOn();
vtkPolyDataMapper *faceMapper= vtkPolyDataMapper::New();
faceMapper->SetInputConnection(normals->GetOutputPort());
vtkActor *faceActor= vtkActor::New();
faceActor->SetMapper(faceMapper);
faceActor->GetProperty()->SetColor(1.0, 0.9, 0.2);
faceActor->SetScale(20.0, 20.0, 20.0);
faceActor->RotateX(90);
faceActor->RotateY(60);
faceActor->SetPosition(1.4, 0.6, 4);

// Create the Renderer and assign actors to it. A renderer is like a
// viewport. It is part or all of a window on the screen and it is
// responsible for drawing the actors it has.  We also set the background
// color here.
vtkRenderer *ren1= vtkRenderer::New();
ren1->AddActor( cylinderActor1 );
ren1->AddActor( cylinderActor2 );
ren1->AddActor( cylinderActor3 );
ren1->AddActor( cylinderActor4 );
ren1->AddActor( cubeActor );
ren1->AddActor( faceActor );
ren1->SetBackground( 0.1, 0.2, 0.4 );

//
// Finally we create the render window which will show up on the screen.
// We put our renderer into the render window using AddRenderer. We also
// set the size to be 300 pixels by 300.
//
vtkRenderWindow *renWin = vtkRenderWindow::New();
renWin->AddRenderer( ren1 );
renWin->SetSize( 300, 300 );

//
// The vtkRenderWindowInteractor class watches for events (e.g., keypress,
// mouse) in the vtkRenderWindow. These events are translated into
// event invocations that VTK understands (see VTK/Common/vtkCommand.h
// for all events that VTK processes). Then observers of these VTK
// events can process them as appropriate.
vtkRenderWindowInteractor *iren = vtkRenderWindowInteractor::New();
iren->SetRenderWindow(renWin);

//
// By default the vtkRenderWindowInteractor instantiates an instance
// of vtkInteractorStyle. vtkInteractorStyle translates a set of events
// it observes into operations on the camera, actors, and/or properties
// in the vtkRenderWindow associated with the vtkRenderWinodwInteractor.
// Here we specify a particular interactor style.
vtkInteractorStyleTrackballCamera *style = vtkInteractorStyleTrackballCamera::New();
iren->SetInteractorStyle(style);

// Unlike the previous scripts where we performed some operations and then
// exited, here we leave an event loop running. The user can use the mouse
// and keyboard to perform the operations on the scene according to the
// current interaction style. When the user presses the "e" key, by default
// an ExitEvent is invoked by the vtkRenderWindowInteractor which is caught
// and drops out of the event loop (triggered by the Start() method that
// follows.
iren->Initialize();
iren->Start();

// Final note: recall that an observers can watch for particular events and
// take appropriate action. Pressing "u" in the render window causes the
// vtkRenderWindowInteractor to invoke a UserEvent. This can be caught to
// popup a GUI, etc. So the Tcl Cone5.tcl example for an idea of how this
// works.
```

```
  //
  // Free up any objects we created. All instances in VTK are deleted by
  // using the Delete() method.
  cylinder->Delete();
  cylinderMapper->Delete();
  propertyTable->Delete();
  cylinderActor1->Delete();
  cylinderActor2->Delete();
  ren1->Delete();
  renWin->Delete();
  iren->Delete();
  style->Delete();

  return 0;
}
```

3. Visualisation of a Sorting Algorithm (23 Marks)

This question is a **peer programming task**. You are allowed to choose **one partner** and you can both submit the same solution. If you decide to do this task together with another student clearly state in the document `Ass3Answers.doc` who your partner is (Name, UPI). You will both get the same mark for this part of the assignment. If you prefer to work alone you may do so.

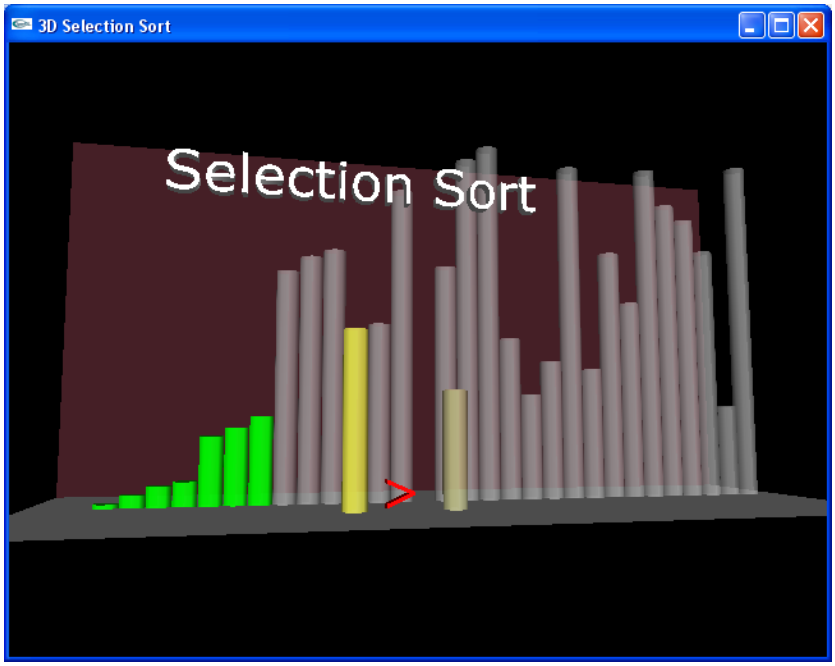
(b) [5 marks] Think about what "features" of the sorting algorithm you need to visualize in order to demonstrate to the viewer the basic idea behind the algorithm. For example, QuickSort uses a "divide-and-conquer" principal, so the visualization should clearly indicate how the array is divided into smaller sub-arrays and how these are merged subsequently. How can you use knowledge from "Visual Perception" research in order to make your visualization more efficient (see section 4.4 of the handout "Chapter 4 - Scientific Visualization")? Write the answer to this question into the Word document `Ass3Answers.doc`.

Solution hints:

- Identify the basic steps of the algorithm (divide, merge, swap, …)
- Identify invariants (e.g. when performing a bubble sort, after iteration *n* the last *n* elements are in the correct order)
- Think about how you prove the correctness of such an algorithm – the properties you use when proving its correctness can often be used to visualize it effectively!
- Use animations to clearly show changes in the order of the array. Draw attention to important regions (e.g. before two elements are swapped mark them by arrows or using blinking or hot colours)
- Use perceptual hints to show the evolution of the algorithm, e.g. use a 3D representation and put a copy of the array representing the current iteration of the algorithm in front). If you use a 2D representation you can mark the array representing the current iteration using highlighting, bold lines, enlargement etc.
- The evolution of the algorithm could also be shown using a graph (especially useful for Quicksort).

(c) [10 marks] Implement an effective visualization of the sorting algorithm chosen by you. Please submit both a .NET solution (without binaries) and a working executable. Think carefully how you can make the best use of the screen space, color, shape, animations, markers etc. in order to clearly visualize the working of the sorting algorithm. If you complete this part of the question you do not need to submit a separate program for part (a).

Solution hints: The picture below shows an example of a good solution. The value of a number is represented by the height of a pillar. The sorted list is clearly indicated in green. The two elements currently being compared are emphasized by offsetting them and the result of the comparison is shown by using different colours and an appropriate symbol. The animation of the pillars further improves the understanding of the algorithm.



(d) [3 marks] Think about how your program could be modified in order to visualize the sorting of an array with 10,000 elements. Write the answer to this question into the Word document `Ass3Answers.doc` (no implementation is necessary).

Solution hints:

- Use techniques from information visualization such as “Focus & Context”. For example, when visualizing bubble sort you could show the entire array as an “abstract object” with the sorted and unsorted region indicated by different colours. You could then show the region where action happens in detail using a zoom window.