Lecture I
INTRODUCTION TO VISUALIZATION

Visualization is an area that is intimately related to Computer Graphics and Geometric Modeling. In this course, we will introduce you to several issues of current Visualization research. One of our special interest is (1) very large data sets and (2) very narrow and unreliable bandwidths (called **thinwires**). Wireless connections and internet access typify thinwires.

§1. Course Information

The course home page is at URL `http://cs.nyu.edu/ yap/classes/visual`.

OPERATING SYSTEM: The official programming environment is some unix like environment like Solaris or Linux. If you much prefer to work on Windows, we strongly recommend the free *Cygwin* system that sits on top of Windows. Cygwin is a unix look-alike. it is easy to install (and to update). You have convenient access between Windows and Cywin (no need to dual boot).

PROGRAMMING LANGUAGE: Programming will be done in Java, especially *Java2D* API.

TEAMS: This is a hands-on, projects-oriented course. We allow you to do your homework AND projects in teams. You must register teams with me for approval. I recommend teams of 1-3. The composition of your teams may change throughout the semester.

PREREQUISITES: Fundamental Algorithms, Java programming. Knowledge of computer graphics, geometric modeling and networking are not considered prerequisites.

GRADING: 25% homework, 15% class participation and short quizzes, 60% final project.

MAILING LIST: Please sign up for the class mailing list immediately. See instructions in the class webpage.

TEXT BOOK: We strongly recommend the book *JAVA 2D Graphics*, by Jonathan Knudsen from O'Reilly, 1999. For general web-based issues, we suggest *Core WEB Programming* (Second Edition), by M. Hall and L. Brown, from Sun Microsystem Press (Prentice Hall), 2001. Another reference is *Graphic Java* (Third Edition), by D. M. Geary, Sun Microsystem Press (Prentice Hall), 1999.

§2. Introduction

Visualization is a basic tool used in many disciplines to help researchers understand their data. The fact that our visual system is one of the most highly developed faculties may partly explain why converting data to visual form is frequently the best route to understanding. We need to visualize a variety of datasets: images, maps, fluid flows (e.g., weather patterns), volumetric data (e.g., brain tissues), geometric models (e.g., aeroplane or engine), large graphs (e.g., internet sites), or just abstract tabular data. A major issue arise from the fact that these datasets can be very large. Further, if we want to do visualization on the internet, this is compounded with severe bandwidth limitations (we call this "thinwire visualization").

Is it just Computer Graphics? Visualization is closely intertwined with two other subjects: Computer Graphics and Geometric Modeling.

(1) Geometric Modeling is about how to represent geometric models in computers. The computational and mathematical properties of the representation is of inherent interest. Such goals are intimately connected to CAD/CAM applications. For instance, Geometric Modeling is interested in studying different classes of surface patches for their flexibility and power in expressing certain properties that modelers need.

(2) Computer Graphics is traditionally concerned with rendering geometric models on the screen or other output device. Photorealism and speed are two major rendering goals. For instance, this calls for the study of light models. But the field has also grown so much that in recent years, non-geometric models are also addressed. Sometimes it is hard to tell Comuter Graphics and Visualization apart.

(3) Visualization is about understanding models and data through their graphical representation. To illustrate the difference between computer graphics and visualization, suppose a user has a complex geometric model that needs to be "understood". This is a somewhat vague goal that depends on what the user is trying to accomplish: an engineer's goal may be to understand how to manufacture this object; a designer's goal may be to understand the shapes and their functioning in the overall design; a user may wish to check out its manueverability. Photorealistic rendering of the model is only one possible approach to understanding it. But for a model with complex internal details, photorealism may not be appropriate. Instead, we want the ability to see through the object (with X-ray eyes).

Geometric modeling focuses on the modeling elements (polyhedral or surfaces) and the synthesis of models (methods for putting together the modeling elements). But usually (1) is not very useful if there is no way to display or visualize the model. But sometimes we can query a model and get an answer without any display. E.g., we might have a map data model and we can perform queries and get a non-visual answer.

Usually (2) assumes the model is given and focus on the issue of transforming this into images on the screen. This is called rendering, and is the central issue of computer graphics. Graphics has also gotten into an area that is somewhat between the rendering issue and the "modeling" part. That is, how do you model natural images of clouds, waves, mountain terrains, forests and fire? A central goal of rendering is realism.

Finally, (3) is about understanding, and so we view the realism goal of computer graphics as only one way towards understanding an object. But if non-realistic images are produced to help understanding, this is not a bad thing. For instance, we are quite happy to display a house in which part of the walls are removed, or walls are made transparent, so that we can look inside. This is surely unreal. The original data need not even be geometric. For instance, there is great interest in visualizing graphs (the field called Graph Drawing arose around this interest in the 1990s).

What about the web? The world wide web (www) is arguably the biggest social phenomenon of the last decade. But what has it to do with visualization? Clearly much of the communication on the web is visual. For us, it highlights the main issues of thinwire communication, recalling its popular sobriquets such as "world wide wait" or "waiting without whining". Wireless communication is another example of thinwire. Visualization in such a context is of interest. Conventional visualization techniques fail in such settings, and we want to investigate new techniques.

Let us take one element in thinwire visualization that distinguishes it from conventional visualization: the computational model. Conventional visualization has the standard CPU, main memory and secondary memory model of computation. When large models are to be visualized, it is assumed to be stored in secondary memory. The key issue how to organize the data so that the necessary parts of this model can be brought into main memory in a timely fashion. Two important classes of techniques here are database management and I/O efficient algorithms. In contrast, the computational model for thinwire visualization is the client-server model. The server is assumed to hold the model and the client is accessing this model over a network. The bandwidth is several orders of magnitude smaller than the above “I/O bottleneck”. This bandwidth is also unreliable. So all the standard techniques of walk-through or fly-over applications breaks down when applied to thinwire visualization.

Know Thy Dataset. Researchers have developed many techniques in recent years to address these visualization issues. These techniques are highly dependent on the nature of the datasets. Graphic datasets are well known to be very diverse – in the trade, there are many books on graphic data formats. This proliferation of formats is indicative of the rich nature of graphic data. In this course, we will take a close look at some important types of formats or datasets. In particular, you will learn about the TIGER Dataset. Note that the TIGER data is 2-dimensional, but we will also consider higher dimensional data visualization.

In Summary, the technical basis for visualization includes computer graphics, computational geometry amd geometric modeling. Thinwire visualization requires some networking background (such as TCP/IP protocol). We will provide such background as needed. This course address the following topics:

- Geometric Modeling and Computational Geometry:
- Thinwire visualization: techniques to achieve “responsive” visualization, including proxy approaches.
- Foveation: the human visual system is foveated – how can we exploit this? This is connected to wavelet analysis.
- Large data models: Techniques here include hierarchical (level of detail) modeling, I/O efficient algorithms, and database management issues.
- Volume visualization: In many scientific fields, the data set is volumetric. E.g., MRI data, atmostpheric or ocean data. The method to visualize these are rather different than the geometric model.
- Important data sets: We introduce some important types of data: geometric models such as found in CAD/CAM models, vector and elevation datasets which arise in geographic information system (GIS), and volumetric data.

§3. Computing Environment

Just as having a comfortable home is the first priority to get settled into a new place, we want to have a comfortable computing environment. So let us get started. What I will recommend here is available for anyone to install if they have web access. However, I will try to help you if you face a situation that is different from my assumptions – so do not be shy to ask.

Operating System. My advice here will be in the setting of a unix-like environment. Examples are Linux or Cygwin. For many students with a Windows environment, Cygwin is highly recommended. You can freely download and install Cygwin directly from the web with very little effort. Go to website URL `http://sources.redhat.com/cygwin/` hosted by RedHat. The big advantage is that you can see and access all your regular Windows files from Cygwin, and vice-versa. In short, you do not need to dual boot.

Editors. You will need an editor to enter and modify programs. Your favorite editor will do, but if you have always used a WYSIWYG-type editor, I suggest that you invest a little time to learn a powerful keyboard entry editors. Why? It is a major productivity boost, assuming you have already learned to touch-type. I highly recommend a free editor called "GVIM" which is a GUI-enhanced version of the original "VI" editor (found on most unix systems). Please go to the links page under our home to see some discussion of editor choices.

Java. I will assume that you have some version of Java available in your environment. The latest version of Java is 1.4, but for our purposes you just need Java 1.3 or later. Again, I have tried to collect some useful information in our class links page. To get you started immediately, we would like you to compile and run some programs that is collected in the `progs` . To get all the files in the directory called "awt", just download the tar file called "introGUI.tar". Untar this file into a directory in your own computer. Follow the directions next to run the programs here.

Make Program. Assume you have downloaded and untarred the "introGUI.tar" file in some directory. If you are now in that directory, you should see you see a file called "Makefile". In unix-like environments, there is a program called "make" that reads and executes this file. In our case, you should be able to type:

```
> make p=EmptyFrame
```

to compile and run the program called `EmptyFrame.java`. This is possibly the simplest program imaginable using the GUI system of Java. There is a graded sequence of such examples (the next one is called `EmptyFrame1.java`) that you can next compile and run: just use the name `EmptyFrame1` instead of `EmptyFrame` for the make program.

Why use `make`? The nice thing is that it can automate many chores you need to do in developing programs. Also, the Makefile is a concise document of what is needed to compile and run your programs. So another person can use your programs just by reading your Makefile. We will see more of this in this course.

If you could not run make, you can still compile and run the programs by hand in the usual way:

```
> javac EmptyFrame.java
> java EmptyFrame
```

END OF LECTURE