ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 2004
Seria: GEOMETRIA I GRAFIKA INŻYNIERSKA z. 6 Nr kol. 1615

Róbert TORNAI
Institute of Informatics
University of Debrecen, Debrecen

GRAPHICS CLASSES FOR MOBILE EQUIPMENTS

Summary. The aim of my work is to inspire students to use the basic graphics algorithms in practice. The limited capabilities of mobile machines give the opportunity to teach useful programming tricks and tips. Programming these equipments enhances the ability of abstraction and problem solving too.

SYSTEMY GRAFICZNE DLA PRZENOŚNEGO SPRZĘTU KOMPUTEROWEGO

Streszczenie. Celem pracy jest zachęcenie studentów do stosowania w praktyce podstawowych algorytmów graficznych. Ograniczone zdolności przenośnego sprzętu komputerowego skłaniają do uczenia się programistycznych „sztuczek" użytkowych i poznawania dobrych rad. Wyposażenie programistyczne rozszerza zdolności do myślenia i znajdowania rozwiązań.

1. Introduction

The evolution of computer graphics started slowly. At the beginning computer graphics meant mainly monochrome displays with slow refresh rates. Firstly game consoles and computer games drove the evolution, later business applications as CAD, medical or entertainment programs gained a great influence. With time the compatibility issues became a serious limiting factor in the evolution of graphics equipments. The mobile solutions have the chance to use the experiences gained so far and to avoid the pitfalls.

Standards are important because they ensure compatibility and continuity of developments. Moreover, they are fundamental for portability. Mobile equipment manufacturers have a great influence on mobile standards. Accessible standards that are easy to learn and easy to use will stimulate mobile applications [1].

Today mobile equipments' possibilities are comparable to the computers of 10 or 15 years ago. The different file transfer techniques (from the simplest as using service cable to the wireless ones as GPRS, WAP or IrDA download) inspire people to have commercial or own useful applications in their equipments, not only the built in ones. My conviction is that as evolution continues, more and more graphical field gains ground on mobile equipments. This article will focus on mobile phones in the followings.

2. Possibilities and limitations

Mophun C++ is unhappily supported only by Sony-Ericsson. So, we have to choose Java if we want our program to be used widespread. Almost all of the new mobile phones are Java capable. This means that the target group, who may need mobile applications, already has the necessary hardware.

Independently from the chosen programming language there are hardware limitations. The CPU of this kind of machines has only fixed point unit. There is no support for hardware floating point machinery. Using fix point numbers seems to be the only alternative for this problem. Another limitation comes from the relatively small resolution, from the physical size and from the small number of displayable distinct colors (called *color depth*). What's more, the response time could be better as well. The answer for the problem above is time. According to Moore's law [2] (which says the size and cost of electronics shrinks by a factor of two every eighteen months) will provide for more processing power, memory, and storage space. These facts emphasize the role of developing scalable programs.

Fortunately Java itself has no restrictions about resolution, physical size or the color depth. However, Java is lack of some tools. MIDP 1.0, supported by WTK 1.04, has classes to manage access to the screen and to query the state of the keyboard through its several methods. Putting aside the basic functions, there is no support for visualizing any 3D object, especially surfaces. Clipping for different kind of regions is not ensured, only for rectangles. MIDP 2.0 developments get compiled with WTK 2.0 beta. It is extended with package *Game* having new classes, and the existing classes have some useful new functions as *fillTriangle* which makes it possible to visualize surfaces approximated by adjacent triangles. Because of its bigger size, MIDP 2.0 is not commonly contained yet in mobile phones nowadays.

Class *Math* does not have functions *sin* and *cos*, needed by rotation transformations. Therewith a matrix implementation is needed for multiplying transformation matrices and point coordinates together. Determining distances requires the computation of square roots. Moreover, there is no built in support for sorting in Mobile Java. Quick sorting is essential. (e.g. Painter's algorithm heavily depends on it.) There is no kind of class that stores geometric information.

3. Implemented basic graphics algorithms

There is a built in method for clipping to rectangle areas, so I focused on convex and concave area clipping, and in or out examinations for convex and concave areas. Class *CB_Rep* (developed classes' names will start with capital *'C'*) implements boundary representation for solid object modeling. This class stores geometric and topology information of spatial objects. Class *CGraph* encapsulates in or out examinations of points and line drawing for convex and concave windows (that can contain holes or even islands in the holes recursively in arbitrary deep) [4] Furthermore, this class implements Painter's algorithm needed for surface representation. This method sorts the parameter array containing triangles according to their weight points' *z* coordinates. The function *WindowToViewport* helps repositioning and resizing the projected picture of the space.

All the following classes rely on a powerful fix point class that takes over the role of the real number types. Class *CPoint2D* and *CPoint3D* represent two or three-dimensional points having homogeneous coordinates. These classes have a normalizing function and what is more important, a function to determine the Descartes coordinates of a point. Moreover, they implement scalar and vectorial product of vectors. These methods are essential to compute intersection points of lines, or to determine on which side of a line a point is. *CMatrix* is a container class for transformation (translate, rotate, mirror, scale) and projection (central, parallel, axonometric) matrices having fixed point homogeneous coordinates. [3] It implements the desired matrix functions. The transformation and projection matrices can be multiplied together into one matrix. Using the result matrix instead of applying the transformation matrices one by one yields a huge increase in execution speed.

To achieve the defined goals, we need to take care of some mathematical functions as *sin*, *cos* or *sqrt* too. I decided to store the fixed point values in arrays for functions *sin* and *cos*. This old trick can be combined with another one: instead of the traditional 360 degrees the circle is divided into 256 pieces. If there is a need for more precision, it is advisable to divide the circle into pieces of a power of 2, because bit masking is an efficient way to determine the correct angle in the case of overflow or underflow. For computing the square root of a number, I have implemented the formula of Newton that converges to the square root of the parameter value in relatively small number of iterations.

4. Performance issues

Programming languages having a garbage collector do not fit for real-time program developing. So, the number of dynamically allocated objects has to be kept as low as possible by avoiding returning objects of user classes. Instead of doing so, the methods will have an extra parameter taken as a reference to a previously allocated object. The amount of time the garbage collector uses can be reduced this way.

This programming style means as if we were programming in pure C. An essential object-oriented feature is lost this way, but the performance gain worth it. (At least till the mobile machines will have enough horsepower for the overhead.) I think that only those applications can rely on serious computational power yet for a time that follows this pure C style.

Since Java's interpreter and the fact that there is no possibility to write inline functions or macros, performance can increased by reducing the number of function calls which cause a significant overhead. Let's try to avoid the unnecessary calls by storing semi results and unchanging values in variables.

5. Teaching programming via computer graphics

I am always trying to use my lectures to broaden the field of view of my students by showing them programming techniques and tricks, not necessarily related with graphical problems' solutions.

I can see a sharp border between old school and new school programmers. Old school programmers took the full path from the beginning. This means they are used to hardware with limited capabilities. Restrictions always enhanced inventiveness, so they have the ability to develop tricky algorithms or improve the code manually. They are usually familiar with many programming languages such as imperative ones like Basic, Assembly, Pascal or C (perhaps even Fortran, Ada or PL1), declarative ones such as Prolog or Lisp, or with the object oriented Java or C++. (There are of course database handler languages as Clipper. However, since these languages have a special application field, not necessarily are used by all of the programmers.) These circumstances developed their program planning and designing ability. Abstraction and abstract thinking are essential. Programmers of old school are not bound by one or two programming language. They are solving problems independently from the implementation issues and they are able to choose the best tool for coding. Conversely, programmers of new school (who started to deal with informatics in the 90s) usually have poor abstraction abilities. When new technologies become available, there is no appropriate base for comparison.

I try to pass over the benefits of the old school to my students. Real problems help to bring closer to them important techniques they might have learnt. The significance of fix point over floating point computation is visible only when performance becomes a serious limiting factor. Who is familiar with assembly programming knows about the amount of CPU cycles that instructions take up, or with macros that can be replaced by defines and inline functions. Incremental algorithms are more important than on desktop machines. 80% of time is spent in the core part of the iterations. The most important thing is to reduce the load at these points of a program. The implemented graphical problems are ideal for acquiring program planning and design. Students can learn how to split up problems into smaller ones that are easier to handle. Besides students can see multiple ways to solve particular problems. Many times a totally different approach ensures only significant improve in execution speed. Developing in practice enforce students to acquire the use of handy tools such as Debugger or Profiler.

6. Results

With the help of the developed classes I succeeded in writing applications using 3D computer graphics. I have tested the MIDP 1.0 applications on my Siemens S55. It has turned out that even on this top model the real-time animation is applicable only to simpler wire frame models. However, in my opinion the greatest result is that 3D surfaces can be visualized even if real-time motion is not possible. I think that mostly business and game programs will enjoy these new graphical benefits. Paired with Bluetooth or other wireless solution, medical monitoring equipments can profit from it as well.

Perhaps the class *CFixPoint* can be improved to handle varying fix point according to the actual value. Doing so may increase precision.

Nevertheless, the developed classes can serve as a good basic for further developments, extensions. My plan is to integrate the classes into a package. Computer graphics perhaps becomes more interesting for students by their mobile equipments. By teaching the applied techniques through these graphics classes too, supposedly will inspire the students to write own programs which using and extending them. The applied techniques and tricks or even the approach to the problems can be used widely in other fields of programming.

The object-oriented design makes it possible to transfer the classes to C++. I am planning to create a Mophun C++ port of the graphics classes. Hopefully this port will result in a quicker system for Sony-Ericsson machines. I would like to test my programs not only on mobile phones. If there is a chance, I will perform a comparative performance test. Teaching of computer graphics can benefit from the developed classes mostly, where studying concrete implementations can be more important and useful than using them.

BIBLIOGRAPHY

1. http://www.nokia.com/nokia/0,,27155,00.html May 5, 2003.
2. Moore G.: Cramming more components onto integrated circuits. Electronics, 38, 8, 1965.
3. Juhász I.: Számítógépi geometria és grafika. Miskolci egyetemi kiadó, Miskolc 1995.
4. Foley J., van Dam A., Feiner S., Hugues J., and Phillips R.: Introduction to Computer Graphics. Addison Wesley, 1993.

Recenzent: Dr Edwin KOŹNIEWSKI

Abstract

While the mobile equipment penetration is larger and larger these days, the quality of the graphics of mobile applications is not satisfying. One reason can be that development tools do not support graphics, especially 3d graphics at an acceptable level. Since almost all of the development tools are object-oriented for these machines the aim of my work is to provide a class library specialized for mobile equipments that speeds up development time and ensures a higher quality.

Although commercial products as OpenGL ES (Embedded System) or 3DObjects for Java are on the way to conquer this area, I think that a maybe less complex class library ready right now has its own usage field. Moreover, the main benefit of an own framework is that it can easily be used for educational purposes. The developed class library will help me to extend the scope of my course covering basic computer graphics problems. Since all source files are given, when new graphics problems arise, the students can extend my graphics classes themselves, which is an efficient way to improve their knowledge in my opinion.