Design of Programming Exercises for a MOOC Course of Computer Graphics

Peng-bo BO and Xiao-miao ZHANG*
Harbin Institute of Technology, Weihai, China
*Corresponding author

Keywords: Massive open online course, Programming exercises, Computer graphics.

Abstract. In order to improve teaching effectiveness of a MOOC course, namely Computer Graphics (CG), a careful design of several programming exercises is requisite. By analyzing fundamental knowledge points in CG, five programming exercises are designed and explained in this paper. A program is provided for each exercise which is a framework with empty functions to be completed by a student. This set of exercises, as a significant supplement to video lectures and problem sets, is of great help to students to understand fundamental concepts in CG and master basic skills for graphics programming.

Introduction

Massive open online course (MOOC) is a platform which consists of both traditional teaching materials and user forums where teachers and students can communicate interactively [1]. MOOC makes distance learning a more efficient and effective learning way. Moreover, it has become one popular teaching and learning mode in colleges and universities [2].

In addition to video lectures and problem sets, programming practices are significant components of many courses in computer science, such as data structure, computer graphics and operating system. Therefore, a careful design of appropriate programming experiments is of great importance to students’ deep understanding of knowledge. Appropriate programming exercises should cover most fundamental knowledge in a course and accomplishing these exercises can help a student to grasp both theoretical knowledge and programming skills. In this paper, a design of five experiments of programming practice for a course, namely Computer Graphics (CG), is proposed and explained in details [3].

Design of Experiments

By investigating the core contents in CG, five experiments are designed to help a student to understand the fundamental concepts in CG. This set of experiments include line generation, geometric transformation, parametric curve, polygonal mesh and lighting model. The proposed five programming practices are not independent and dependencies of them are illustrated in Figure. 1.

![Figure 1. Knowledge structure of proposed experiments.](image-url)
**Line Generation**

Generation algorithms for simple 2D geometric shapes are fundamental to a Graphical system. Since these basic algorithms are commonly called thousands of times in rendering a single image, the efficiency of them is very important. The DDA algorithm and Bresenham algorithm are two classical algorithms for straight line generation which are introduced in all computer graphics textbooks [3].

A program framework is provided for this experiment which uses Windows GDI programming interfaces. GDI is a collection of functions designed for displaying 2D shapes in Windows. In order to simplify the usage of GDI functions, a class named CDC and several classes derived from it have been developed by Microsoft [4]. Our program is developed with MFC Document/View framework.

The client region of the program window is split into an array of squares where each square is a representation of a pixel in a displaying device. A function called `FillOnePixel(int x, int y)` has been implemented to fill in one square specified by its coordinate \((x, y)\).

The students are required to implement two straight line generation algorithms, namely the DDA algorithm and the Bresenham algorithm, based on the provided program. A student is supposed to answer 2 questions based on their implementation of these 2 algorithms: 1) What is the difference in running time of these 2 algorithms when rendering the same line segment; 2) Are the resulting pixels generated by these algorithms exactly the same? Figure 2 shows the interface of this program. Figure 2(a) shows result of the DDA algorithm; Figure 2(b) shows result of the Bresenham algorithm.

![Figure 2. Straight line generation.](image)

This experiment helps the students to understand the concept of basic shape generation in a laser-scanning displaying device.

**Geometric Transformation**

Geometric transformations play a key role in modeling and viewing phases in a graphical rendering pipeline. Three fundamental geometric transformations in 3D include translation, rotation about an axis passing through the origin of a coordinate system and scaling about the origin of a coordinate system. A complicated transformation can be realized by composing a sequence of basic transformations using multiplications of transformation matrices. A technique in geometric modeling closely related to geometric transformation is the so called layered modeling. In many cases, a complicated model is designed as a collection of simple models by applying geometric transformations. Therefore, deep understanding of layered geometric modeling is of great importance to learn CG.
A program based on OpenGL is provided for this experiment [5]. In this program, all necessary settings for OpenGL are already done. A function which is capable of displaying a unit cube is provided in this program as well. Figure 3(a) shows the interface of the original program. The task for students is to complete the program to model and render a robot by applying appropriate geometric transformations to a unit cube. What are to be implemented by a student are three functions responsible for basic geometric transformations, namely Translate, Rotate and Scale. Figure 3 shows the interface of the original program and an expected result. Observing the results by various transformations helps the students to master modeling techniques using layered transformations which is fundamental to modeling complicated geometric scene.

**Polygonal Mesh**

A number of techniques for 3D representations are introduced in CG, among which boundary representation (B-rep) is one of the most important ones. Nowadays, complicated 3D models in commercial applications are represented by polygonal meshes. In order to help the students to fully understand the techniques of surface modeling and get prepared for graphics programming in their future career, a data structure of mesh is introduced in this experiment. The state-of-the-art data structure for a polygonal mesh representation is the half-edge data structure [6].

To simplify the concept about half-edge data structure, three structures are defined using C language, namely HE_edge, HE_vert and HE_Vertex. The definitions of these classes are given in Figure 4.

```c
struct HE_edge {   struct HE_vert { 
    HE_vert* vert;   float x; 
    HE_edge* pair;   float y; 
    HE_face* face;   float z; 
    HE_edge* next;   HE_edge* edge; 
}; 
};
```

Figure 4. Half-edge data structure of polygonal mesh.

Based on the definitions of HE_edge, HE_vert and HE_face, a C++ class called Mesh3d has been implemented and provided for students. Mesh3d contains most important functionalities for operations on a mesh, as well as functions for importing and exporting a mesh data file. The data file has the format frequently used in state-of-the-art commercial software, namely the obj file. Using member functions of Mesh3d, one can easily construct a mesh object and perform various querying operations on a mesh. In addition, a function capable of displaying one triangle is provided in Mesh3d.
Based on the class \textit{Mesh3d}, the students are required to complete a program to load a 3D model from a data file and render the model in a displaying device. In addition to rendering all faces in a model, all edges in a model are to be rendered at the same time. Another task for the students is to develop a functionality for changing viewing directions. This needs basic geometric transformations, especially a rotation operation, which has been introduced in the second experiment. Figure 5 shows the program interface with a model of a human head displayed in two different viewing directions. By accomplishing this experiment, a student is supposed to understand the half-edge data structure very well.

\textbf{Parametric Curve}

Design of curve and surface is another important topic in CG. Parametric curves and surfaces introduced in CG course include Hermite curve, Bezier curve, B-spline curves and their surface counterparts [3]. For this part, an experiment is designed and students are required to implement the Hermite curve in 2D. A program is provided which is capable of rendering simple 2D shapes. This program is basically a variation of the program used in the first experiment.

Based on the provided program, a student can specify a data point on plane by clicking the left button of a mouse. The position of a click point gives a data point \( P \) immediately. Tangent vectors are specified by clicking an additional point \( Q \) for each data point \( P \). That is, a user may clicking a sequence of data points: \( P_1, Q_1, P_2, Q_2, \ldots \), where \( P_i \) act as data points and \( Q_i-P_i \) act as tangent vectors.

Students are required to implement the Hermite curve in 2D. Suppose we are given two points \( P_1:(x_1, y_1) \) and \( P_2:(x_2, y_2) \) and corresponding vectors \( T_1:(s_1, t_1) \) and \( T_2:(s_2, t_2) \), a Hermite curve \( C(u) \) between \( P_1 \) and \( P_2 \) is computed using the standard formula of Hermite curve. By definition, \( C(u) \) takes \( T_1 \) and \( T_2 \) as tangent vectors at \( P_1 \) and \( P_2 \) respectively, i.e., \( C'(0)=T_1, C'(1)=T_2 \).

Once a function for a piece of Hermite curve is implemented, one can use it to construct a composite Hermite curve. Each piece of Hermite curve acts as an independent curve. The fact that two neighboring curves share a common tangent vector at their joining point leads to a \( C^1 \) continuous curve. This experiment helps a student to understand the concept of a parametric curve, which is basic to understand parametric surfaces. Figure 6 shows results of this experiment. Figure 6(a) is one...
piece of Hermite curve. Figure 6(b) shows a composite Hermite curve which consists of 4 pieces of Hermite curves segments.

![Image](image1.png)

(a) (b)

Figure 7. Lighting result by Phong model.

**Lighting Model**

The lighting effect is the most important factor of realistic rendering [3]. In this experiment, a student is required to implement the Phong lighting model and use it to render a Bézier surface. Since a surface is usually rendered as a mesh, several functions are provided in the program which are responsible for computing a point as well as associated surface normal on a Bézier surface. A mesh is generated for a Bézier surface by taking sample points on it. For simplicity, we arrange the sample point into a grid and use two 2-dimensional arrays to store the surface points and associated surface normal. For the purpose of mesh representation, the class Mesh3d based on half-edge data structure is used here, which is introduced in the third experiment.

A student is required to complete a function for computing a color on a mesh vertex using the Phong lighting model and then render the mesh surface using computed color. Using the Phong lighting model, one can compute a radiance at a point on a surface. When taking \( R, G, B \) components of the light, one gets a colored shading result with realistic lighting effect. Figure 7 shows a rendered surface with the color computed using the Phong model in two different views.

**Performance Evaluation**

To evaluate the performance of a student, in addition to checking students’ coding work, a technical report is required for each experiment. After finishing an experiment, source code, an executable file and a technical report are required to be uploaded into a MOOC system. The evaluation is conducted by the students as well as by the teacher. The work of each student is evaluated by three students. The teacher will examine the code and technical report of all students and a final score is automatically generated by considering the evaluation given by both the students and the teacher. From 2012 to 2017, over 500 students in Harbin Institute of Technology at Weihai have conducted the programming practices introduced in this paper. It has been demonstrated that these experiments have remarkable impact on study performance of the students.

**Summary**

CG course is one of the most important professional courses in the major of computer science. One of the most difficult and time-consuming task in teaching CG is the design of experiments or programming practices. In this paper, authors propose details of the design of five fundamental experiments.

Although these experiments cover the most basic knowledge in CG, more programming exercises are needed to cover other important contents. Two important topics are projective transformation and global lighting model. In order to help a student to understand the concept of projective transformation, a viewing volume should be modeled and both 3D objects in a scene and their projections on the viewing plane should be rendered. One of the most important algorithm of global
lighting is the light tracing algorithm. In order to implement the light tracing algorithm, recursive functions should be used.

Acknowledgement

This research was financially supported by the Humanities & Social Sciences Research Foundation of Shandong Province (Grant No. J17RB231) and Innovation Research Course Foundation of Harbin Institute of Technology, Weihai.

References


