The Hybrid Approach to Teaching Graphics by Incorporating Cg Shading Language

Ying Tang Jing Fan Xujia Qin

Abstract—**It is regarded as a long-standing difficulty in the computer graphics course that how to balance the education of the fundamental theoretical concept with the goal of motivating students by enabling them to produce visually interesting projects. In this paper we propose the hybrid approach to teaching graphics by incorporating [1]Cg shading language. By writing programs with Cg, students are exposed to the underline functionality obscured by the fixed-function graphics API. Our teaching experience indicates that students have a comprehensive understanding of the algorithms and mathematical concepts of the low-level graphics pipeline. The students' feedbacks show that they are satisfactory with the teaching effects and fully motivated and inspired by the assigned projects.**

Index Terms—**CG Shading language, Computer graphics, pedagogy.**

I. Introduction

With the fast development of GPU (Graphics Processing Unit) as the core components in computer display systems, many computer graphics related fields become flourishing these years. In order to meet the increasing talents shortage posed by fast developing media and game industry, the major of **DM** (Digital Media) has been newly-opened and established in many universities in China to train the qualified students in latest years. The computer graphics course has been recognized as the important compulsory subject for the major of DM. By being taught computer graphics, the students are expected to be equipped with fundamental mathematical concepts of computer graphics as well as techniques to produce visually-interesting images. This course aims to not only lay the solid theoretical foundation for the succeeded professional courses, but also inspire the career or research interests of students to engage in computer graphics related areas.

The traditional computer graphics course mainly focuses on the detailed raster-level algorithms with the emphasis on understanding bottom mathematical concepts and theories [1]. This kind of teaching style is categorized as *bottom-up* [9], in which students are required to implement basic stages of rendering pipeline, such as clipping, line/triangle conversion, etc. By teaching computer graphics in bottom-up style, the students are more familiar with low-level algorithms and theories that underline modern rendering pipeline. However, according to the author's teaching experience, for students with not so good mathematical background, it is difficult for them to totally understand these algorithms. Moreover, most students are discouraged by the details and many special cases needed to be handled in these algorithms. They have difficulty to produce visually interesting images by implementing all detailed functions by themselves. Thus their interests in computer graphics are damaged. This is particular harmful, since many students are drawn to computer graphics by the desire to produce visually appealing images. In contrast with the bottom-up teaching style, the *top-down* teaching style becomes popular for computer graphics courses in many universities [2][4]. In this manner, students are taught to use graphics API (*e.g.* OpenGL) to draw images on the screen [3]. The low-level algorithms are enclosed in these APIs, which hides the details to the users. It is much easier for students to render attractive images in this teaching manner compared with the above bottom-up style. However, the students would get too involved in the technical details of the chosen API and do not have enough theoretical understanding of the low-level algorithms behind the API [9].

In this paper, we describe the hybrid approach to teaching graphics that has been adopted in our class. This approach combines the merits of bottom-up and top-down teaching styles by incorporating Cg shading language developed by Nvidia Corporate [5]. The goal

Manuscript received July 21, 2008. This work was supported in part by the Project of Zhejiang Provincial Education Department (20070185), University Excellent Courses Program (YX0706, JP0808), Graduate Teaching Project "The Research to Strengthen The Evaluation of Degree Program" of Zhejiang University of Technology.

Ying Tang is with the Software College, Zhejiang University of Technology, Hangzhou, China (phone: 86-571-85290034; e-mail: ytang@ zjut.edu.cn).

Jing Fan is with the Software College, Zhejiang University of Technology, Hangzhou, China (e-mail:fanjing@zjut.edu.cn).

Xujia Qin is with the Software College, Zhejiang University of Technology, Hangzhou, China (e-mail:qxj@zjut.edu.cn).

of our teaching approach is to lay the solid theoretical foundation as well as exciting and cultivating students' interests for graphics. The students are expected to produce visually interesting images without neglecting underline theoretical details. To achieve this goal, we remove the raster-level algorithms in our class and adopt Cg shading language to write programs. The Cg shading language provides us the flexible ability of vertex and pixel programming, which are unavailable for fixed rendering pipeline. Such ability exposes the relevant low-level details of the graphics pipeline to the programmers. By teaching students write programs with Cg rather than calling ready functions provided in graphics API (such as OpenGL), the students will have better understanding of the underline graphics theories compared with top-down teaching manner. The students are more likely to produce appealing images since they do not need to implement all low-level algorithms from scratch. The feedbacks from the class also verify the effectiveness of our approach.

II. Cg Shading Language

For nearly the past ten years, graphics hardware has been developing at a speed almost three times faster than Moore's Law predicts [7]. As a consequence of this fact, the contents of the graphics courses should be frequently updated to catch up the advances of the technologies. The programmable rendering pipeline, compared to the traditional fixed rendering pipeline, boosts the technological advancements in graphics as the main driving force. So it is necessary to include the introduction to the programmable rendering pipeline in the computer graphics course. However, despite the fact that the shaders for programmable rendering pipeline are ubiquitous in modern graphical applications, it is common that these contents are not contained in the introductory computer graphics course.

The Cg (C for Graphics) shading language is a high-level shading language developed by Nvidia in close collaboration with Microsoft for programming vertex and pixel shaders. The shader programs are graphics processing functions operating the programmable parts of the rendering pipeline. The vertex and pixel shaders were programmed at a very low level with assembly language of the graphics processing unit before Cg was created. The portable, high-level Cg language makes shader development much easier.

Cg is very similar to Microsoft's HLSL. CG is based on the C programming language and they share the same syntax. Some features of C were modified and new data types were added to make Cg more suitable for programming graphics processing units. It is very easy for students to grasp the Cg language who are already familiar with C or C++ syntax and flow control. The Cg language is also very similar to Microsoft's High Level Shading Language (HLSL). So students with experience to one shading language would rather easily get familiar with the other. There are also many Cg shading language reference materials on the web, including reference texts, developing toolkits and forums. The classical book of "Cg Tutorial" is suitable for the class use since it contains a complete specification as well as many useful example codes.

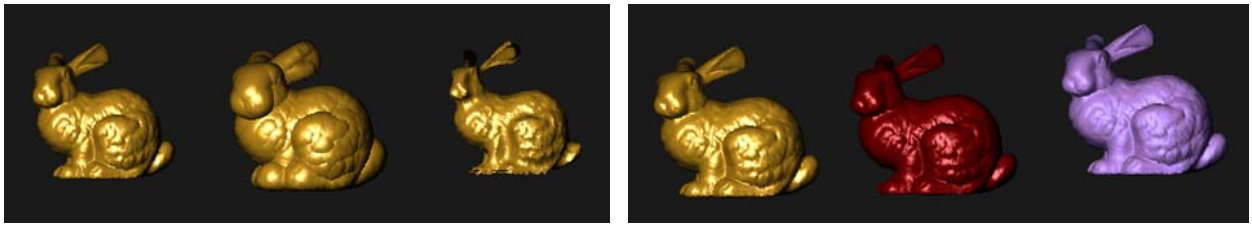
III. Courses Incorporating Cg

Traditional top-down approaches to teaching graphics focus on the use of fixed-function OpenGL API. In this course we transfer to the programmable graphics pipeline by incorporating Cg in the OpenGL API. In the following of this section, we will first introduce the curriculum of the course, then explain the benefits of using Cg.

A. Curriculum

The text book of [3] provides us a good reference on how to organize and present the materials. This text book includes rich contents which cover nearly all topics in our courses. It introduces the fundamental theories, concepts and algorithms of graphics in a top-down manner based on OpenGL, the cross-platform open graphics library. As for the parts in the text book where the theoretical concepts are not introduced thoroughly, we ask the students to refer to the book of [9] which provides excellent description for low-level algorithms. The students are also asked to read the book of [10] to get familiar with OpenGL programming interface and program writing.

In our course, we first give the overview of the rendering pipeline. Then each stage of the pipeline is introduced in detail by providing competent treatment of low-level algorithms. As for the application stage of the pipeline, the course about geometry and modeling is given, where different 3D geometric representation methods are introduced. For the geometry stage we give three lectures, which are about transformations, lighting and shading. For the rasterization stage, we arrange two



(a) (b)

Figure 1: Some pictures generated by students. (a) 3D models enlarging and shrinking in normal direction implemented by vertex shader in Cg. (b) Phong Shading results implemented by pixel shader in Cg.

lectures which are about rasterization and texturing, and visibility and culling. The Cg shading language is introduced by two lectures. The first is arranged at the beginning of the semester, where simple syntax and how to setup running environment are introduced. So the students can finish the assignments by writing programs with Cg. The second lecture is arranged at the end of the semester, where we discuss advanced rendering techniques and algorithms that are implemented with Cg.

For the assignments of this course, the students are asked to implement several stages of the pipeline by using OpenGL API and Cg, like geometry transformations and lighting, as well as a big project where students are trained to implement complex rendering results. In a semester-long course, we assign four substantial programming problems. The first is a short assignment where students are asked to write a simple Cg program. Students would get familiar with the syntax and running environment of Cg by this project. The second is about manipulating the geometry transformation matrixes with Cg. The students are expected to understand the mathematical theories underlying these transformations and learn to set parameters for the transformation matrixes. The third is about the lighting computation. Students are expected to use Cg to simulate some classical lighting effects (*e.g.* Phong Local Lighting Model) as well as some special lighting effects (*e.g.* Gooch Shading). The final project is open-ended. The students are asked to implement the advanced effects achievable only with Cg. They can choose any special effects that they want to simulate, such as fire, smoke or bump-mapping. To give student a starting point for their assignment, we provide the example source codes in both fixed-function and Cg form. All of the four assignments can be incrementally added to a single application program to get the final big one. The students have a sense of achievement with the final full-functioned program. Figure 1 shows some assignment results implemented with Cg.

It is important to understand that the Cg shading language is only used as a tool to extend and enhance students' understanding. It is of little value to teach the Cg for its own sake here.

A. The Benefits of Using Cg

The programmable OpenGL API discloses the parts operating on vertices and pixels to the developers which were used to be enclosed in the fixed-function API. Thus the responsibility of applying transformations is shifted from the API to the developers. By using Cg to write vertex and pixel shaders, the students are forced to understand the mathematical transformations and computations that performed by the pipeline. Here we show a few examples.

In Figure 2 we present a simple Cg vertex program for computing the light for each vertex by Phone lighting model. In the fixed-function graphics pipeline, this computation procedure is transparent to the developers. What they can do is limited to setting some parameters. By writing this vertex program, the programmers have to be familiar with the lighting model. They should understand that an object's surface color is the sum of emissive, ambient, diffuse, and specular lighting contributions. They also need to know what each term is about and how to compute these terms. Since Cg syntax is extremely straightforward, students are easy to connect the mathematical equations to the Cg functions. The behavior of each stages of the pipeline is no longer obscured by fixed-function OpenGL API.

ISBN: 978-988-98671-0-2

```
void basicLight(float4 position  : POSITION,
                float3 normal1   : NORMAL,
            out  float4 oPosition:POSITION,
            out  float4 color      :COLOR,
        uniform  float4x4  modelViewProj,
        uniform  float3  globalAmbient,
        uniform  float3  lightColor,
        uniform  float3  lightPosition,
        uniform  float3  eyePosition,
        uniform  float3  Ke,
        uniform  float3  Ka,
        uniform  float3  Kd,
        uniform  float3  Ks,
        uniform  float    shininess)
{
  oPosition = mul(modelViewProj, position);

  float3  P = position.xyz;
  float3  N = normal;

  float3  emissive = Ke;

  float3  ambient = Ka * globalAmbient;

  float3  L = normalize(lightPosition – P);
  float  diffuseLight = max(dot(N,L), 0);
  float3 diffuse = Kd * lightColor * diffuseLight;

  float3  V = normalize(eyePosition – P);
  float3  H = normalize( L + V );
  float  specularLight = pow(max(dot(N,H),0),
                             shininess);
  float3 specular = Ks * lightColor *
                             specularLight;

  color.xyz = emissive + ambient + diffuse
              + specular;
  color.w = 1;
}
```

Figure 2: The Cg vertex program for Phong lighting model.

```
glEnable(GL_TEXTURE_CUBE_MAP);
```

Figure 3: The fixed-function OpenGL API to set cube mapping.

```
void  Reflection_V( float4 position : POSITION,
                    float2texCoord : TEXCOORD0,
                    float3  normal     : NORMAL,
                out   float4  oPosition : POSITION,
                out float2  oTexCoord : TEXCOORD0,
                out float3  R              : TEXCOORD1,
                uniform float3  eyePositionW,
                uniform float4x4  modelViewProj,
                uniform float4x4  modelToWorld)
{
    oPosition = mul(modelViewProj, position);
    oTexCoord = texCoord;

  float3 positionW = mul(modelToWorld,
position).xyz;
    float3  N = mul((float3x3)modelToWorld,
normal);
   N = Normalize(N);

   float3  I = positionW- eyePositionW;
   R = reflect(I, N);
}
```

Figure 4(a): The vertex program of Reflective Environment Mapping.

```
void  Reflection_F(
                float2texCoord: TEXCOORD0,
                float3  R:     TEXCOORD1,
            out  float4  color : COLOR,
            uniform  float  reflectivity,
            uniform  sampler2D  decalMap,
        uniform samplerCUBE  environmentMap)
{
    float4 reflectedColor =  texCUBE(
                                environmentMap, R);
    color = reflectedColor;
}
```

Figure 4(b): The fragment program of Reflective Environment Mapping.

For complicated cubic environment mapping, the code with Cg is more elucidating. In Figure 3 we show the fixed-function OpenGL API necessary to perform the cube mapping. Here we just turn on this function by invoking glEnable() function with the parameter of GL_TEXTURE_CUBE_MAP. The programmer needs not to know how this algorithm works to get the texture color. The whole procedure is transparent to the programmer. However, if the programmer writes this algorithm in Cg as the sample code displayed in Figure 4, he/she has to be clear about every step of this

algorithm. Figure 4(a) and Figure 4(b) show the vertex program and the fragment program respectively. The vertex program computes R the reflected eye vector, which is highlighted in the Figure 4(a). The fragment program performs the corresponding texture lookup according to R and set the output color for the fragment. This example illustrates one of the fundamental advantages of using Cg: shaders expose functionality the traditional API obscures.

IV. Class Feedbacks

We have collected the students' responses to this course after the semester finishes and we find that these course evaluations have some instructive value.

The students were asked to evaluate the qualities of teaching materials, teaching methods and teaching effects. We have conducted the surveys to two classes of 60 students. Table 1 shows the survey results.

Table 1: Students' rating of our computer graphics course

	Poor	Fair	Satisfactory	Very Good	Excellent
Teaching Materials	0%	0%	15%	23%	62%
Teaching Methods	0%	0%	16%	23%	61%
Teaching Effects	0%	0%	15%	22%	63%

From Table 1, we can see that nearly 85% of the class rated the qualities of the teaching materials, methods and effects as "Very Good" or "Excellent".

We also consulted students' concrete opinions towards this class. We find there is still room to improve our teaching. For example, some students find the Cg programming environment difficult to set up and feel a little frustrated with it. So in the future class we will try to build this programming environment for the students and explain this procedure clear for them in the class.

V. 5. Conclusions and Future Work

In this paper we have described a hybrid approach to teaching computer graphics by incorporating Cg shading language. This approach successfully combines the benefits of the top-down and bottom-up teaching methods. By using Cg to write shaders, the students are exposed to the underline functionality of the graphics pipeline which is usually obscured by fixed-function OpenGL API. So the students have better understanding of the low-level algorithms. Meanwhile it is also much easier for them to produce visually interesting pictures with the help of Cg, which is very exciting and motivating. As computer graphics hardware continues the exponential advancements, we believe that programming shaders will play an increasingly important role in real-time applications [6] [8]. It may come to this point that the students are expected to use shading language as a common tool to develop their applications. So students with no prior experience with shading language will be at the disadvantage.

In the future, we plan to introduce more special effects that can only be achieved with Cg for this course. By introducing these topics, students will get more inspired to see the wonderful simulation results. They will also become more familiar with Cg and be good at applying it to many graphics problems.

References

[1] Tang Rongxi, Wang Jiaye, Peng Qunsheng. Computer Graphics Tutorial. Science Publisher, China, Revised Edition, 2001.

[2] K. Sung and P. Shirley. A top-down approach to teaching introductory computer graphics. In SIGGRAPH '03: Educators program from the 30th annual conference on Computer graphics and interactive techniques, pages 1–4, New York, NY, USA, 2003. ACM Press.

[3] E. Angel. Interactive Computer Graphics: A Top-Down Approach Using OpenGL. Addison-Wesley, fourth edition, 2005.

[4] S. Cunningham. Powers of 10: the case for changing the first course in computer graphics. In SIGCSE '00: Proceedings of the 31st SIGCSE technical symposium on Computer science education, pages 46–49, New York, NY, USA, 2000. ACM Press.

[5] R. Fernando and M. Kilgard. The Cg Tutorial: The Definitive Guide to Programmalbe Real-Time Grapics. Addison-Wesley, 2003.

[6] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan. Brook for GPUs: stream computing on graphics hardware. SIGGRAPH '04: Proceedings of the 32nd annual conference on Computer graphics and interactive techniques, 23(3):777–786, 2004.

[7] M. Kilgard. NVIDIA graphics, Cg, and transparency. SIGGRAPH 2006 Course Notes, 2006.

[8] M. McCool, S. D. Toit, T. Popa, B. Chan, and K. Moule. Shader algebra. SIGGRAPH '04: Proceedings of the 32nd annual conference on Computer graphics and interactive techniques, 23(3):787–795, 2004.

[9] P. Shirley. Fundamentals of Computer Graphics. A. K. Peters, Ltd., second edition, 2005.

[10] D. Shreiner, M. Woo, J. Neider, and T. Davis. OpenGL Programming Guide: The Office Guide to Learning OpenGL Version 2. Addison-Wesley Professional, fifth edition, 2005.