Shading in a distributed environment

Goga Nicolae[1], Florica Moldoveanu[2], Alexandru Telea[1]
[1]*Computer Science Department,*
Eindhoven University of Technology,
the Netherlands,
[2]*Computer Science Department,*
Politehnica University,
Romania,
goga@win.tue.nl, fm@cs.pub.ro, alext@win.tue.nl

Abstract

This paper presents a tool for light-shading in a distributed environment. A parallelization method based on a concurrent model is described. The purpose of this work is to lower the image generation time in the complex 3D scenes synthesis process. The experimental results concerning the speedup of light shading algorithm are also presented.

1 Introduction

Computer-generated images can achieve a high degree of realism with light shading mapping [3]. This technique is used to make the images of three-dimensional objects more interesting and apparently more complex.

There are different methods used to provide a light shading. Among these, three major methods are very popular and they are: (1) The most well known is flat, or constant, shading. This approach applies an illumination model once to determine a single intensity value that is then used to shade an entire polygon, and holding the value across the polygon to reconstruct the polygon's shade (2) Gouraud shading, also called intensity, or color, interpolation shading. Gouraud shading extends the concept of interpolated shading applied to individual polygons by interpolating polygon vertex illumination values that take into account the surface being approximated. Each polygon is shaded by linear interpolation of vertex intensities along each edge and then along each scan line. (3) Phong shading also known as normal-vector interpolation shading interpolates the surface normal vector rather than the intensity. Interpolation occurs across a polygon span on a scan line, between starting and ending normals for the span.

Of these methods, the first two are used in games consoles. In this paper we address the case of constant shading. This approach is valid if several assumptions are true: a) the light source is at infinity; b) the viewer is at infinity; c) the polygon represents the actual surface being modeled, and is not an approximation to a curved surface. Because of these assumptions, flat shading is sometimes no that realistic, but because it is fast is used in practice in different domains.

One of the major problems for the light shading is that because of the complexity of the computations the generation of the image is slow. One way to solve this problem is to divide the computations among more processors/machines and to execute them in parallel [4]. The current paper presents the tool ConstShadIm which applies an constant shading model to the objects of the scene. The serial and the parallel modules are presented together with the comparison of their performances. ConstShadIm is designed such that it can be simply extended through the introduction of new shading models.

The aim of our tool is to better understand the effect of distributed shading computations with respect to acceleration. This understanding is very important when designing multi-layer parallel shading architectures, such as the current pixel shaders existing in the nowadays graphics cards [5, 1]. Specifically, one can design new modules for ConstShadIm to test the effect of distributing shading computations with respect to the type of computation (shading).and type of distribution (inter-module communication).

This paper is organised as fellows. Section 1 presents a general theory of light shading. Section 3 describes the tool ConstShadIm. Section 4 describes the experimental results of the comparison. Section 5 gives the conclusions.

2 Flat Shading

For the shading model [2] the following mathematical definitions are used:

- L: light source vector (direction) striking the surface
- R: light reflected vector
- V: viewpoint vector (eyepoint) looking at the surface
- N: normal vector (perpendicular) to the surface
- I: light intensity

The light reflected at a surface point is represented in Figure 1.

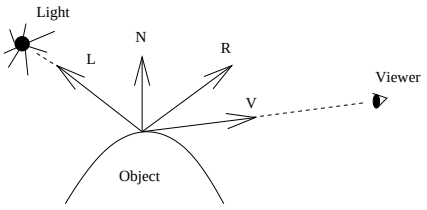

Figure 1. Light reflection.

When the shading is applied to a surface point, then it is computed taking into account the other elements in the scene. Light ray may hit several surfaces before it reaches the viewer. For a better approximation, higher cost is needed.

Local illumination models are used in computer graphics to achieve interactivity. Some tricks are used in conjunction with some models to approximate a global solution. Global illumination models are used to improve image quality.

When ambient light is used, it approximates indirect light reaching the surface. Some amount of diffuse light is added to the environment. This is a trick that tries to achieve a global effect using a local model.

$$A = I_a \times k_a$$

where I_a is the ambient intensity and $0 \leq k_a \leq 1$ describes how much of the ambient light is reflected by the surface material (out tool uses the ambient light trick)

Assuming that N and L have been normalized, the perceived intensity at the surface point due to illumination is

$$A = I_a \times k_a + I_l \times k_d \times (N \times L)$$

where I_l is the intensity of the point light source and $0 \leq k_d \leq 1$ is the diffuse reflection coefficient of the surface material and tells how much of the incident light is diffusely reflected by the material.

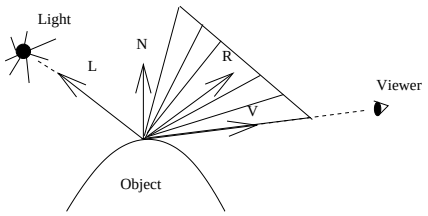

Figure 2. Specular reflection.

On ideal surfaces, light is reflected only in the direction R. If the surface is not a perfect mirror, light is reflected inside a cone of directions around ideal reflected direction. The rapid fall of the *specularly* reflected intensity is approximated as $cos^n(\alpha)$, where α is the angle between R and V. For normalized R and V vectors, the resulting illumination models becomes:

$$A = I_a \times k_a + I_l \times (k_d \times (N \times L) + k_d \times (V \times R))^n$$

where $0 \leq k_s \leq 1$ is the specular reflection coefficient.

When the effect of multiple light sources are added the formula becomes:

$$A = I_a \times k_a + \sum_{1}^{lights} I_l \times (k_d \times (N \times L) + k_d \times (V \times R))^n$$

A shading model determines when the illumination model is evaluated, e.g. once per polygon, once per vertex or once per pixel. Flat shading is an illumination model evaluated once per polygon, i.e. all pixels of the polygon are shaded with the same color. It introduces intensity discontinuities and match band effect, i.e. discontinuities in intensity or in its first derivatives. Valid only if:

1. the light source is at infinity, so $N \times L$ is constant across the polygon face
2. the viewer is at infinity, so $N \times V$ is constant across the polygon face;
3. the polygon represents the actual surface being modeled, and is not an approximation to a curved surface.

The implications of using fast-shading in graphics is that for high realism and detail levels, a large number of flat-shaded polygons is needed for a good-looking image. This technique is used for example by the CAD/CAM designers.

3 Light shading in a distributed environment

The tool ConstShadIm implements the constant shading method. It allows the use of more light sources and the efect of the ambient light. Each object of the scene is described by means of polygons. Each polygon is defined by its vertices. For constructing the image, a Z-buffer algorithm is used. The tool allows operations as rotations, translations and scales for the objects in the scene.

The shaded image is constructed in an internal buffer and after that displayed on the screen. The serial module can be summarised as follows:

```
for all polygons of the scene execute
 {
  compute the color of the pixels of the
  polygon according to flat-shading


  draw the filled polygon to the Z-buffer
  }

draw the Z-buffer to screen
```

The programming methods used is the OOP technique. The classes are represented in Figure 3.

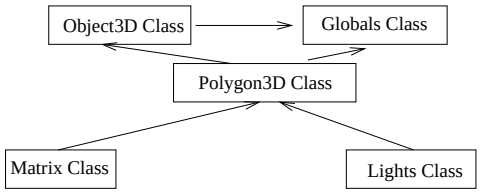

Figure 3. Object classes

The class Lights contains data and methods concerning the computations of the flat shading. The class Polygon3D has data and methods related to a polygon of the scene. The class Matrix implements operations such as rotations or translations. The class Objects3D stores data and methods regarding objects of the scene. The class Globals contains globals data and methods such as the Z-buffer and the drawing of the Z-buffer. Using the OOP technique the tool can be easily extended with new features such as new types of shading implemented as new methods of the class Lights.

The parallel module of the tool uses a concurrent processes model and is based on the data partitioning method. The number of processes used depends on the size of the graphical database and the number of connected machines in the network. The processes are started at the same moment of time for performing the computations in parallel. The user decides in advance the distributions of the processes on the machines. In this way the task distribution is static and no execution time is consumed for performing this operation.

Each process has access at a graphical database which stores the polygons of the scene. This database can be located on a shared network directory or it can be copied locally on the connected machines. Each process has assigned a portion from the total number of polygons to compute. In total a number of n Z-buffers are produced, where n represents the total numbers of processes. These Z-buffers should be combined for the image to be displayed on a target screen. The transmitions of the Z-buffers to the process which displays the image on the screen cause an overhead for the parallel module.

Performance tests were achieved using two implementations of the flat-shading algorithm: the serial module and the parallel module.

4 Experimental results

The measurements were performed on a network formed by identical machines. Their characteristics are presented in table 1.

Table 1. The Network Architecture

Machine	Characteristics
Processor	Intel Pentium III
Frequency	1 GHz
Memory	256 MB

- Screen: Phillips JMW Computers
- SO: Windows XP Professional
- Communication: at socket level, TCP/IP protocol
- Length of message transmited: 200 kb aproximatelly, for the parallel module

The technical conditions imposed to have the parallel module running on two machines (the parallel module consisted on two processes). For the serial and the parallel modules we performed 5 measurements of the execution time and the obtained values are presented in table 2. The scene contained nine identical objects. Each object has a number of 1976 vertices and 3751 polygons Three point light sources were considered.

Table 2. Execution Time

Nr	Serial	Paralel
1	15.21 s	10.32s
2	13.45 s	9.41s
3	14.73s	11.02s
4	14.25s	9.75 s
5	13.89s	10.67s

Making the average of these measurements we obtained the following values of the execution time.

- serial module: $T_s = 14.30s$
- parallel module:
  - $T_p = 10.23s$
  - SpeedUp $= \frac{14.30}{10.23} = 1.39$

Measurements of the execution time were performed varying the number of the objects of the scene. The objects were identical and contained the number of polygons and vertices as indicated above, i.e. 1976 vertices and 3751 polygons per object. Three lights were considered. For each scene a number of five measurements were performed. The experimental results obtained are presented in Table 3.

Table 3. Execution time for different numbers of objects (average values)

Nr. Objects	Serial	Parallel
1	2.45s	4.34s
3	6.41s	6.21s
6	9.85s	8.35s
9	14.30s	10.23s

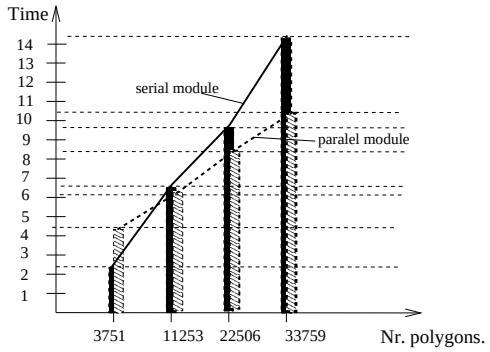

Figure 4. Execution time vs. nr. polygons

What is interesting to observe is that the experimental results show that a decrease of the computation time for the parallel module vs. the serial module happens only when starting with a number of polygons above 11000 (the case of three identical objects). For a number of polygons less than 11000 the overhead caused by the transmission of the Z-buffer is greater than the efficiency caused by the splitting of the computations. The point of equilibrium is obtained around the value of 11000. This is also represented in the Figure 4 by the two curves for the serial and the parallel module.

5 Conclusions

The purpose of this work is to present a tool for light shading and to study how the performances enhance for shading algorithm in a distributed environment. To have a comparison between the serial and the parallel module we adopted a simple scheme of parallelisation based on concurrent processes model. The tool was developed so that it can be simply extended for new graphic functionalities such as the new shading methods. One of the future aims of the tool is to simulate parallelization scenarios similar to the ones occuring in the nowadays shading architectures [5, 1].

In the parallel module an extra time is necessary for the communication between the processes. This extra requirement is not neglijable but benefit may be gained through parallel processing. Tests were also performed in order to investigate the influence of varying the number of polygons in the scene concerning the execution time.

The tests showed that for a sufficiently greater number of polygons in the scene (in our case above 11000) a benefit is gained through parallel processing. For a small number of polygons in the scene no benefit is gained by parallel processing and consequently the serial method works faster than the parallel one. This is explained by the overhead introduced by the communication between the parallel processes which is the usual draw-back of the paralelisation

References

[1] General-purpose computation using graphics hardware. URL: http://www.gpgpu.org.

[2] T. Akenine-Moller and E. Haines. Real -time rendering. 2002.

[3] J. Foley, A. Adam, S. Feiner, and J. Hughes. Computer graphis -principles and practice. 1997.

[4] N. Goga, R. Zoea, and A. Telea. Texture mapping in a distributed environment. *Proceedings of IV'03, IEEE Computer Society Press*, 2003.

[5] K. Proudfoot, W. R. Mark, S. Tzvetkov, and P. Hanrahan. A real-time procedural shading system for programmable graphics. *SIGGRAPH*, pages 159–170, 2001.