Texture mapping in a distributed environment

Goga Nicolae[1], Zoea Racovita[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, zoe.racovita@siemens.com, alext@win.tue.nl

Abstract

This paper presents a tool for texture mapping in a distributed environment. A parallelization method based on the master-slave 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 texture mapping algorithm are also presented.

1 Introduction

Computer-generated images can achieve a high degree of realism with texture mapping. This technique is used to make the images of three-dimensional objects more interesting and apparently more complex. With texture mapping, the effect of ”shiny plastic” of the object surface is diminished and the material properties may be exhibit.

There are many methods used to provide a textural impression. Among these, three major methods are very popular and they are: (1) General texture mapping that consists of ”painting” a picture onto a smooth surface; the texture becomes part of the object database. (2) Bump mapping techniques, also known as normal perturbation, adds the appearance of roughness to the surface. The method apparently alters the geometry of the surface by perturbing the normal vector used in the light calculation. (3) View-dependent mapping techniques show whereby an associated environment is reflected in the object. The pattern seen by the viewer depends on the viewing direction. These techniques can be regarded as approximations to ray-tracing method.

In this paper we address only the case of general texture mapping. The texture, a pre-defined image in a specific domain, is mapped onto a 3D object. Each point on the object has to be associated with an element in texture space. In general this involves finding two (u, v) values in texture domain from a three-dimensional coordinate.

One of the major problems for the texture mapping 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 paralel. The curent paper presents the tool MapIm which maps diferent images on diferent 3D-surfaces. The serial module and the paralel modules are presented together with the comparison of their performances. MapIm is designed such that it can be simply extended through the introduction of new object types for 3D surfaces.

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

2 Texture Mapping

This technique corresponds to the transformation from one coordinate system, texture space, to another space, 3D-object space. If both spaces are represented parametrically, the texture mapping is straightforward, since a parametric patch, by definition, already possesses u,v-values over its surface. In the polygonal representation, which is the standard object representation in rendering techniques, each vertex of the polygon is associated with a texture map coordinate (u,v). All the possibilities of texture mapping are summarised in figure 1.

Proceedings of the Seventh International Conference on Information Visualization (IV'03)
1093-9547/03 $17.00 © 2003 IEEE

Most of the work is concerned with mapping directly from 2D-texture space to scene space and an inverse mapping is commonly employed. The screen space is uniformly sampled and the inverse image of a pixel in texture space is formed. The corresponding quadrilateral in the texture space is sampled and filtered and a texture value returns to the pixel. When the projection of the surface in screen space is small, a large number of texels will map into a single pixel. Accurate mapping produces, in general, a curvilinear quadrilateral. In this case, it is important to address the aliasing problem correctly. Different anti-aliasing techniques have been proposed by [2].

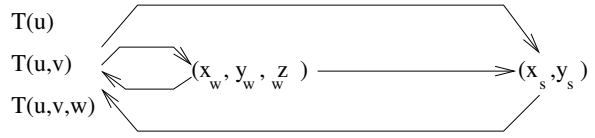

Figure 1. Texture mapping transformations

[1] proposed a practical mapping technique that does not impose any constraints on the shape of the objects. The method consists of finding two mappings:

- Mapping from 2D-texture space to a simple intermediate 3D surface, as sphere or cylinder.

$$T(u,v) \xrightarrow{\text{S–mapping}} T'(x_i, y_i, z_i)$$

- Mapping from the intermediate 3D surface to the object surface.

$$T'(x_i, y_i, z_i) \xrightarrow{\text{O–mapping}} T(x, y, z)$$

These combined operations distort the texture pattern onto the object in a natural way. Various methods were proposed for the two mapping functions. The O-mapping bases on the intersection of a vector with one of the two surfaces. Thus, the vector may be a reflected view ray, surface normal at (x, y, z) or surface normal at (x_i, y_i, z_i).

3 The Texture Mapping Method in a Distributed Environment

MapIm implements an algorithm which uses the inverse texture mapping method. The 2D texture is mapped onto simple 3D objects as sphere, cylinder and cone trunk. For these objects we have chosen the polygonal representation. The polygons, defined only by their vertices, are parameterised so that a mapping can be derived for all interior points. The average texel value in a quadrilateral corresponding to a pixel is computed in order to obtain the pixel colour.

The mapping functions are:

- Sphere:

$$(x = r\cos(\theta)\sin(\phi), y = r\sin(\phi)\sin(\phi), z = r\cos(\theta))$$

with $0 \leq \theta \leq \pi$ and $0 \leq \phi \leq 2\pi$

$$(u, v) = (\frac{1 + \frac{\theta}{\pi}}{2}, \frac{(\frac{z}{r} + 1)}{2})$$

with $\theta = \text{arctg}(\frac{y}{x})$ and $\phi = \arccos(z)$

- Cylinder:

$$(x = r\cos(\theta), y = r\sin(\theta), z = hw)$$

with $0 \leq \theta \leq 2\pi$ and $0 \leq w \leq 1$

$$(u, v) = (\frac{\theta}{2\pi}, \frac{z}{h})$$

with r, h- the cylinder's radius and height

- Cone trunk:

$$(x = Hw, y = Rw\cos(\theta), z = Rw\sin(\theta))$$

with $0 \leq \theta \leq 2\pi$ and $(H - h)/H \leq w \leq 1$

$$(u, v) = (\frac{\text{arctg}(\frac{z}{y})}{2\pi}, \frac{x + h - H}{h})$$

with $H = \frac{hR}{R-r}$; r, R, h - the radii and height

where $0 \leq u, v \leq 1$

The parallel modules of the algorithm uses the master-slave model and is based on the data partitioning method. The master-process assigns tasks to the slave-processes. Each task processes a number of n image polygons. The task size, n, assigned by the master-process to a slave-process depends on the size of the graphical database and the connected machines in network. The task distribution scheme is static and thus, no additional work is required to perform this operation. The graphical database generated by each process consists of the polygonal representation of elementary objects: sphere, cylinder and cone trunk. The texture image, represented as a bitmap, is copied locally on all connected machines. The master-process tells to a slave-process what is the portion of each object that it will process. All the processes contribute to the generated image of a 3D object, but only the master-process write in the memory buffer in order to display the image.

Performance tests were achieved using three implementation of the texture mapping algorithm: the serial module and two parallel modules. The serial module is summarised as follows:

Proceedings of the Seventh International Conference on Information Visualization (IV'03)
1093-9547/03 $17.00 © 2003 IEEE

```
For all polygons of the scene execute
   {
    Determine the polygon projection in the
    screen space

    For all pixel of the projected polygon
    execute
    {
        Apply inverse transformation from
        screen space to object 3D space

        Apply inverse transformation from
        object 3D space to texture space

        Assign pixel colour the average of
        the appropriate texels in the
        texture space
    } }
```

In the parallel implementation each process, including the master, executes the same computations as in serial module, but on different data set. The two parallel modules that were implemented are quite similar. They differ in the method used by the slave-processes for sending messages to the master. In the first case the slave-process sends a single message to the master, when it finishes its job. The message contains the pixel colours, for all assigned polygons. In the second parallel module, the slave sends a message after each processed polygon. The messages are short and the communication overhead is small in comparison with the computing time. In both cases the message length depends on the number of pixels which are covered by polygons in screen space.

The programming method we used is the OOP technique. The class structure is represented in figure 2:

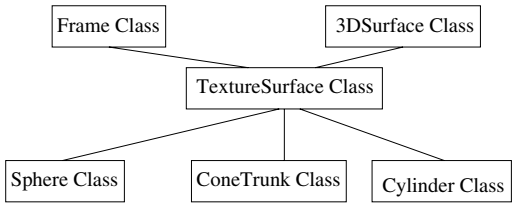

Figure 2. The class structure of the application

The Frame class contains the methods concerning the direct and inverse transformations from the projection plane into screen space. The 3DSurface class is an abstract class with the methods for representing the 3D surfaces in Cartesian and spherical coordinates. The TexturedSurface class is also an abstract class, derived from the Frame class and the 3DSurface class. It adds the displaying methods using the texture space. The derived classes from TexturedSurface class, Sphere class, Cylinder class and ConeTrunk class defines the three types of object that were mapped. Using this class structure we can easily extend the tool by introducing new object types.

4 Experimental Results

The measurements were performed on two machines, which are connected in network. Their characteristics are presented in the table 1.

Table 1. The Network Architecture

Host Name	Workstation	Process
DISCO	SUN AXIL 320	Master
HP1	HP Model 715/33	Slave

- Screen: HP A2088A
- Communication: at the socket level, TCP/IP protocol.
- Message length: 10KB approximately for the first parallel module and 500 bytes for the second parallel module.

Technical conditions impose to have only two processes that work on two machines. The message type makes the difference between the parallel modules. In the first parallel module, the pixel colours of all assigned polygons are assembled in one message, which is transmitted at a time. The second parallel module transmits one message for every processed polygon. One byte is used to represent the colour of a pixel. Theoretically better performances are obtained when sending one big message at a time than many small messages at different moments of time. Although the first method seams to be faster than the second one, the experimental results proved the contrary. These results may be explained through the different speeds of the two connected machines that have the same task size. It is rather difficult to compare an execution time obtained in a heterogeneous environment. O correct estimation involves varying task size as machine speed.

For all of the three modules we performed five measurements of the execution time and the obtained values are presented in table 2. The image contains three objects: a sphere, a cylinder and a cone trunk (figure3).

Table 2. Execution Time

Serial	Parallel I	Parallel II
5.704 s	4.638 s	4.471 s
5.800 s	4.942 s	4.668 s
5.549 s	5.561 s	4.603 s
5.597 s	5.776 s	4.317 s
5.902 s	4.560 s	4.466 s

Proceedings of the Seventh International Conference on Information Visualization (IV'03)
1093-9547/03 $17.00 © 2003 IEEE

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

- serial module: $\mathrm{Ts} = 5.71s$
- 1st parallel module: $\mathrm{Tp}_1 = 5.09s$, $\mathrm{Speedup}_1 = \frac{\mathrm{Ts}}{\mathrm{Tp}_1} = \frac{5.71}{5.09} = 1.12$
- 2nd parallel module: $\mathrm{Tp}_2 = 4.55s$, $\mathrm{Speedup}_2 = \frac{\mathrm{Ts}}{\mathrm{Tp}_2} = \frac{4.55}{5.09} = 1.25$

Measurements of the execution time were performed varying the image resolution. The objects were defined so that they cover approximately an image region of the same size. This involves the same amount of computation in order to determine the pixel colours. The experimental results we obtained are presented in table 3.

Table 3. Execution Time For Different Image Resolutions

R	Cy	Sp	Co	T
50 x 50	0.1970	0.2000	0.1781	0.5752
100 x 100	0.3340	0.3127	0.3573	1.0041
150 x 150	0.5736	0.6042	0.4142	1.5921
200 x 200	0.8922	1.3809	0.6521	2.9252
250 x 250	1.1832	1.5845	1.1031	3.8709
300 x 300	2.4083	2.6335	1.4391	6.4810

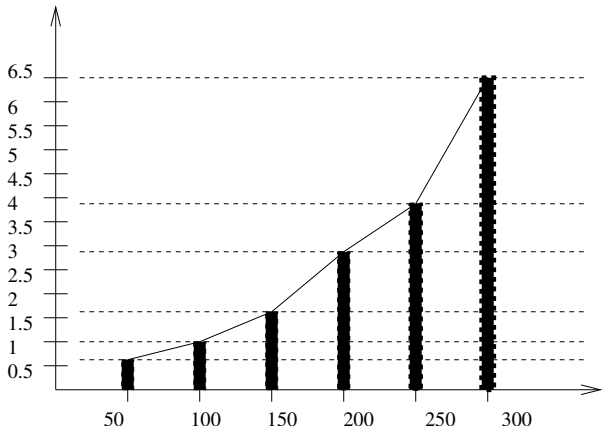

Figure 3. Execution time vs. resolution

In the table above by *R* we mean Resolution, by *Cy* the Cylinder mapping execution time, by *Sp* the Sphere mapping execution time, by *Co* the Cone trunk mapping execution time, and by *T* the total execution time. All times are given in seconds.

We obtained an asymptotic curve of the execution time when the image resolution increases (figure 3). The measurements were performed separately for any object in order to highlight the influence of number of pixels covered by the object's projection.

5 Conclusions

The purpose of this work is to present a tool for texture mapping and to study how the performances enhance for a texture-mapping algorithm in a distributed environment. To have a comparison between the serial module and a parallel one, we have adopted a simple scheme of parallelization based on the master-slave model. The tool was developed so that it can be simply extended through introduction of the new object types by adding new classes derived from the generic TexturedSurface class.

In the parallel module an extra time is necessary to communicate between the master and the slave process. These extra requirements are not negligible, but we gain benefit through parallel processing. Tests were also performed in order to investigate the influence of the image resolution concerning the execution time.

Theoretically better performances are obtained when sending one big message at a time than many small messages at different moments of time. Although the first method seams to be faster than the second one, the experimental results proved the contrary. These results may be explained through the different speeds of the two connected machines that have the same task size. It is rather difficult to compare an execution time obtained in a heterogeneous environment. O correct estimation involves varying task size as machine speed.

References

[1] E. Bier and K. Sloan. Two-part texture mapping. *IEEE Computer Graphics and Applications*, pages 40–53, 1986.

[2] P. Heckbert. Survey of texture mapping. *IEEE Computer Graphics and Applications*, pages 56–67, 1986.

Proceedings of the Seventh International Conference on Information Visualization (IV'03)
1093-9547/03 $17.00 © 2003 IEEE