Fourth Eurographics Workshop on Parallel Graphics and Visualization (2002)
D. Bartz, X. Pueyo, E. Reinhard (Editors)

Design and Implementation of A Large-scale Hybrid Distributed Graphics System

Jian Yang†, Jiaoying Shi, Zhefan Jin, Hui Zhang

Department of Computer Science, Zhejiang University, Hangzhou, Zhejiang, P.R.China

Abstract

Although modern graphics hardware has strong capability to render millions of triangles within a second, huge scenes are still unable to be rendered in real-time. Lots of parallel and distributed graphics systems are explored to solve this problem. However none of them is built for large-scale graphics applications.

We designed AnyGL, a large-scale hybrid distributed graphics system, which consists of four types of logical nodes, Geometry Distributing Node, Geometry Rendering Node, Image Composition Node and Display Node. The first two types of logical nodes are combined to be a sort-first graphics architecture while the others compose images. A new state tracking method based on logical timestamp is also pro-posed for state tracking of large-scale distributed graphics systems. Besides, three classes of compression are employed to reduce the requirement of network bandwidth, including command code compression, geometry compression and image compression. A new extension, global share of textures and display lists, is also implemented in AnyGL to avoid memory explosion in large-scale cluster rendering systems.

Categories and Subject Descriptors (according to ACM CCS): I.3.2 [Computer Graphics]: Graphics Systems-Distributed/network graphics; I.3.4 [Computer Graphics]: Graphics Utilities-Software support, Virtual device interfaces; C.2.4 [Computer-Communication Networks]: Distributed Systems-Client/Server, Distributed Applications;E.4 [Coding and Information Theory]: Data compaction and compression

Keywords: Large-scale Cluster Rendering, Parallel Rendering, Tiled Displays, Image Composition, Remote Graphics, Virtual Graphics, Logical Timestamp, Geometry Compression, Image Compression, Global Share, Memory Explosion

1. Introduction

The interactive computer graphics architecture has developed through four generations in the past two decades[1]. Not only the performance improvement of computer graphics hardware exceeds the Moore's Law, but also modern computer graphics hardware possesses more transistors than modern CPU does. However, many applications, such as scientific visualization of large data sets, high resolution display and photo-realistic rendering, are still unable to run in real-time on high end of modern graphics hard-ware. Thus, the main goal of graphics architecture research is to improve the performance of the overall system architecture.

Interactive graphics hardware on desktop costs from tens to thousands of dollars. Cluster rendering with commodity components becomes new trend to substitute supercomputer which costs millions even hundred millions of dollars. WireGL[2, 22] is a good example for high performance distributed graphics system. Different from WireGL, AnyGL parallelizes more graphics pipeline stages and adopts a new state tracking mechanism, logical timestamp, for parallel rendering on hundreds of nodes to provide high scalability. We designed AnyGL, a large-scale hybrid distributed graphics system. It consists of four kinds of logical nodes, Geometry Distributing Node(G-node), Geometry Rendering Node(R-node), Image Composite Node(C-node) and Image Display Node (D-node) as shown in Figure 1. The G-nodes take in immediate-mode OpenGL commands, pack and distribute them to R-nodes according to the bounding box com-

† Jian Yang and Hui Zhang left Zhejiang Univerisity since June, 2002.

© The Eurographics Association 2002.

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org

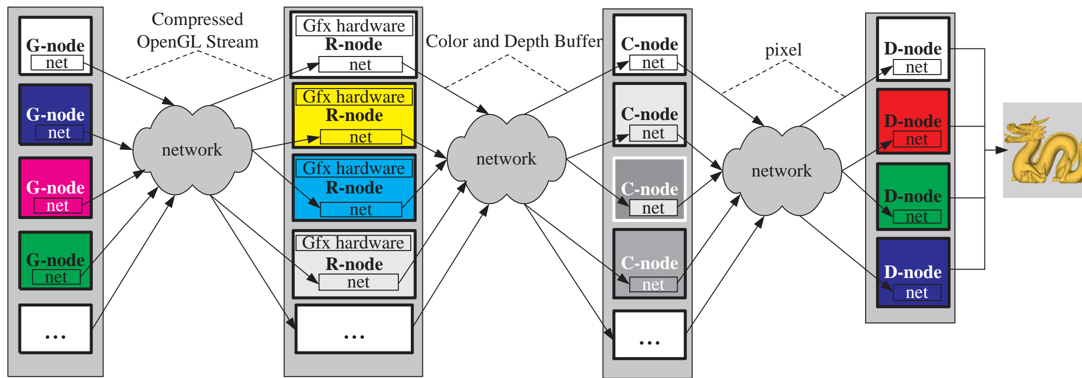

Figure 1: *The main architecture of AnyGL consists of four kinds of logical nodes, Geometry Distributing Node , Geometry Rendering Node, Image Composite Node and Display Node. OpenGL command streams and framebuffers are compressed before transmission.*

puted by the current model view matrix. The R-nodes receive the OpenGL command packets from G-nodes, decode the packets and call their corresponding OpenGL hardware commands. C-nodes compose the images with the depth values transmitted from R-nodes. D-nodes reassemble and display the final images.

G-nodes do same task as clients of WireGL and R-nodes are similar to pipe servers of WireGL. However they are different since a new state tracking method named logical timestamp is implemented in AnyGL. It records the logical timestamps when the state variables of graphics context are modified. Each G-node maintains a few of virtual graphics contexts. Context difference is executed before transmitting command packets. Each R-node also maintains a few of virtual graphics contexts. R-nodes do software context switches when they received command packets.

AnyGL fully exploits compression including command code compression, geometry compression and installable image compression.

2. Related Works

Molnar et al.[3] classified the parallel graphics architecture into three kinds, sort-first, sort-middle and sort-last, by sorting stages.

2.1. Hardware Architecture

Lots of parallel graphics hardware architectures are built on complex standalone accelerators to exploit internal parallelism. They always cost thousands even millions of dollars.

SGI's RealityEngine[4] is a sort-middle tiled architecture which uses a shared high-speed bus to broadcast state commands and primitives. The granularity of task partition is very fine since RealityEngine broadcasts one triangle each time and dispatches every 2 scan lines to one rasterization processor. Pixel-plane 5[5] is also a sort-middle tiled hardware architecture, which distributes primitives from a retained-mode scene description and composes framebuffers by high-speed ring network. The rasterization and fragment stages are executed as SIMD. Eldridge et al.[6] described Pomegranate, a scalable graphics system based on point-to-point communication. Pomegranate is a sort-anywhere architecture, which simulates parallel rendering on five stages, i.e., geometry processing, rasterization, texture mapping, fragment and display.

PixelFlow[7] is a sort-last architecture. Independent graphics pipeline renders a fraction of the scene into independent framebuffer. PixelFlow composes these framebuffers into a single image for final display. The Evans Sutherland Freedom 3000[8] and the Kubota Denali[9] are also examples of fragment sorting architectures. The two architectures process one triangle only once in stages of geometry and resterization. Sort-last architectures will cause significant load imbalance when geometry objects are large.

To reassemble images on clusters, Compaq Research developed a system called Sepia to perform image composition using ServerNet-II networking technology[10]. Sepia reads color and depth buffers over system bus and composes pixels by fast framebuffer access PCI cards.

2.2. Software System

In the research road map of parallel graphics architecture, lots of software parallel graphics systems are designed and implemented for different applications.

An OpenGL stream codec toolkit, GLS[11], tracks, packs, con-catenates and decodes OpenGL commands into streams, which provides the basic idea of OpenGL command packing for remote rendering. GLR[12] furthers this idea to do OpenGL

© The Eurographics Association 2002.

remote rendering as C/S model so that low-end graphics workstations exploit the rendering capacity of supercomputer's graphics system by sending OpenGL commands to supercomputer and reading back color buffers from supercomputer for display.

GLX[13] and X Window provide necessary protocols for OpenGL rendering on X Windows. Small portion of state commands are tracked in GLX such as pixel formats and framebuffers. Parallel Mesa is another good example for sort-last graphics architecture[17]. The multi-projector system of Princeton is a sort-first graphics system[23, 31]. Only one application process emits triangles to remote rendering nodes. It focuses on OpenGL commands distributing and tiled rendering. Igehy et al.[21] studied parallel rendering of order immediate-mode API in Argus and described a parallel graphics programming interface which breaks up the bottleneck of serialization host interface.

WireGL[2, 22] has solved several crucial problems of cluster rendering on commodity components including bucket rendering, distributed rendering and real-time image reassembling. "Dirty bits" is introduced to track OpenGL states on both client and server sides. "Lazy update" synchronizes graphics contexts for immediate-mode OpenGL API. Virtual context difference and software context switch will be completed in a few milliseconds in general cases. Lightning-2[14] reassembles multi-DVI[20] video sources into the final images which provides high-resolution display for WireGL. The parallel programming interface is also implemented in WireGL. But the scalability of WireGL is limited to 32 nodes.

Peter Kipfer[15] designed distributed lighting networks. CORBA is employed to render object-oriented scenes with complex physical lighting. MUDVE[16] is another distributed rendering system based on CORBA. It subdivides VRML scene in object space and transmits VRML nodes to remote rendering nodes. Color and depth buffers are composed by a master node.

2.3. Scalability

Almost all distributed and parallel graphics systems are challenged by the serialization host interface except for the parallel graphics programming interface described by Igehy et al.[21]. The interface allows applications to input triangles by multi-processes and to render primitives in order.

Bus bandwidth challenges RealityEngine on high scalability since all information must be broadcast. Fragment processing approximates to linear speed-up for PixelFlow[7].

Although in Argus[24] texture share in texture mapping stage is studied, memory explosion still exists for textures and display lists in distributed graphics applications. AnyGL develops global share extensions for textures and display lists to avoid memory explosion in distributed and parallel graphics systems.

To obtain high scalability, state tracking based on logical timestamp is designed in AnyGL and three kinds of compression are implemented to reduce the bandwidth requirements for geometry and image transmission through network.

3. Architecture of AnyGL

3.1. Geometry Distributing Node

In AnyGL, OpenGL commands are divided into four categories, primitive commands, state modification commands, remote remapping commands and special commands which differs with WireGL[2].

Primitive commands are the most frequently called commands such as glVertex, glNormal, and glBitmap etc. They do not modify the state variables of graphics context and are simply packed into OpenGL command buffers. A physical packet is divided into two logical buffers[19], one is for packing OpenGL command codes and the other is for packing the parameters of OpenGL commands. Some different protocol packets are designed in AnyGL for geometry compression and image compression. When geometry compression is configured in AnyGL, geometry packets will be compressed before sending them to remote nodes. (Source code of packing and unpacking functions are derived from WirGL[2])

State modification commands are not packed into command buffer directly. State tracking based on logical timestamp is designed to track OpenGL state changes. Before sending out a geometry buffer, context difference computes the state changes of virtual graphics context and transfers these changes into OpenGL commands which are packed into OpenGL stream packet. See Section 4 for details of tracking states based on logical timestamp.

Remote remapping commands are the commands related to display list, texture and vertex array. Each texture has a unique ID generated by OpenGL. When each G-node generates a texture with nID, each R-node will generate several different textures with same texture ID. This will confuse R-nodes to set correct current texture by nID. A scheme is applied in AnyGL to map textures to different local texture object IDs using their G-node ID and texture IDs. This means, textures, display lists and vertex arrays have different definitions on G-nodes and R-nodes.

Special commands consist of WGL(GLX) functions, glClear, glFlush, glFinish and SwapBuffers. Errors will occur when glClear and glFinish are called by several processes within a frame. Generally, these commands are broadcast to all R-nodes.

There are two kinds of OpenGL extensions for parallel graphics programming in AnyGL. One is parallel graphics programming interface first implemented in Argus by Igehy, the other is the global share interface for display lists and

© The Eurographics Association 2002.

texture objects in cluster rendering. See Section 5 for details of global share interface extension.

G-node determines the destination R-node of command packet according to the bounding boxes computed from a set of glVertex commands and bounding boxes are normalized before transmission. G-nodes store the screen subdivisions of G-nodes. When a command packet spans several G-nodes, not only the packet must be transmitted to all spanned R-nodes, but context difference must be executed each time before transmission.

3.2. Geometry Rendering Node

When a R-node receives an OpenGL command package from G-node, it first determines whether context switch is needed. If it receives two continuous packages from two different G-nodes, software context switch must be exploited to set the correct OpenGL hardware context for following rendering. Four kinds of OpenGL commands are executed in their different ways.

Primitive commands call the corresponding OpenGL hardware API directly since they do not modify the graphics context.

State commands are tracked on R-nodes. Similar to state tracking on G-nodes, R-nodes adopt logical timestamp to record the calling of state modification commands.

Special commands such as glClear, SwapBuffers execute only once for each frame. Swapbuffers executes until all previously received commands have been executed. Now only fixed pixel format is supported. To synchronize SwapBuffers for all G-nodes, C-nodes and D-nodes, a global barrier operation is executed before calling hardware Swapbuffers.

Object remapping. When a new texture is defined in the R-node, the texture is created and mapped to its original object by its G-node ID and texture ID. Then the R-node sets it active. When there is another texture with same ID from a different G-node, they will be distinguished by different local texture ID. A same scheme is used for display list.

The color buffer and depth buffer are read back by calling glReadPixels and sent to C-nodes or D-nodes.

3.3. Image Composite Node

It is not necessary for C-nodes to be equipped with graphics accelerated cards since they only receive image and depth data from R-nodes. To reduce the requirements of network bandwidth, each C-node corresponds to a small area of the screen.

The C-node is the core of sort-last architecture. It maintains a color buffer and a depth buffer. When it receives a glClear command, it clears the color buffer and fills depth buffer with the maximal depth value. Within each frame, it will receive image and depth data from its connected R-nodes. The first received image and depth data are directly copied to its local color buffer and depth buffer. Later the color buffer is composed according to the depth function of current context. When all buffers are composed and SwapBuffers commands are received, the C-node copies the image to window and sends it to the D-nodes.

3.4. Display Node

The D-node reassembles color data from R-nodes and C-nodes and displays it on the final display devices such as screens, projectors or other output devices.

Display node assembles images with software and serves as a visualization server in AnyGL. The requirement of network bandwidth is smaller than that of C-nodes since they only receive color data from C-nodes and R-nodes.

4. State Tracking Based on Logical Timestamp

OpenGL graphics pipeline has a state machine which is maintained as graphics context. The change of graphics context must be tracked for parallel OpenGL so that OpenGL commands will run under the correct context. OpenGL state modification commands change the value of OpenGL context variables. WireGL[19] uses "dirty bits" to indicate the state modification and "lazy update" to track context modification. However it is not suitable for large-scale parallel rendering applications.

To gain high scalability, state tracking based on logical timestamp is used in G-nodes and R-nodes. Logic timestamp is the logical value of state modification command calling. Each state variable corresponds to a logical timestamp. Figure 2 shows the basic process of the state tracking of logical timestamp in AnyGL.

State tracking based on logical timestamp consists of five separate processing functions, state commands tracking, context difference, context synchronization, software context switch and state tracking of R-nodes. as shown in Figure 2. C-nodes keep the states of pixel format, depth function, blend function and scissor function so that we focus on state tracking on G-nodes and R-nodes.

Both virtual graphics context and logical timestamp are organized into a hierarchy tree for fast comparison of logical timestamp and quick context difference and context switch. The virtual context divides states into 18 categories according to OpenGL specification[25].

4.1. State Tracking On Geometry Distributing Node

Each G-node maintains N+1 virtual graphics contexts ordered from 0 to N, where N is the number of R-nodes the G-node connects. These virtual graphics contexts are created by same default values. An application virtual context tracks

© The Eurographics Association 2002.

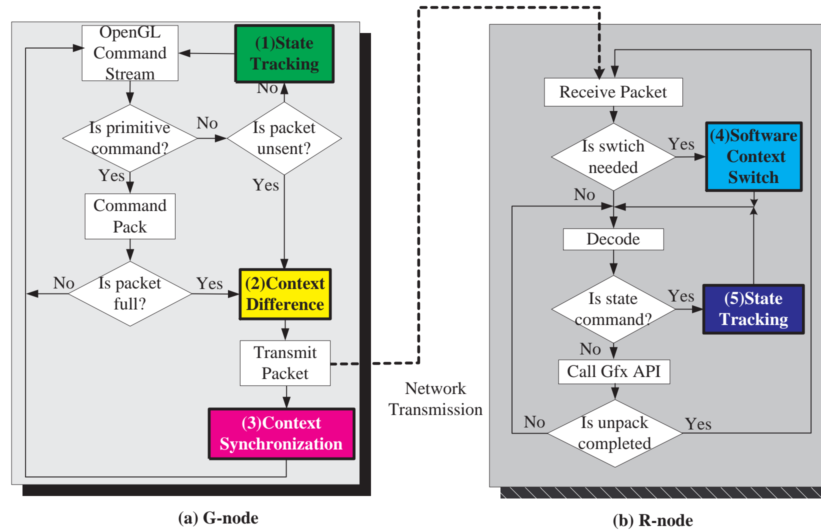

Figure 2: *State Tracking Based on Logical Timestamp, where (a) is the G-node and (b) is the R-node. Five state tracking functions are marked by color filling, where (1) is state tracking of G-nodes, (2) is context difference, (3) is context synchronization, (4) is software context switch and (5) is state tracking of R-nodes.*

```
glLightf_state(light, pname, param){
        check validate of parameters;
        check change of  the light state  ;
        if (changed) {
          lights[light].pnamex = param;
          lights[light].pnamex.timestamp++;
          lights[light].timestamp++;
        }
}
```

Figure 3: *Pseudocode for state tracking of glLightf on G-node. Notice that timestamp increase by 1 when state of light is changed.*

the callings of OpenGL modification commands in this process and then records logical timestamp. Other virtual contexts represent the current virtual contexts of this node on remote R-nodes. A virtual context is set active when it represents the destination of command packet transmitting.

When a state modification command is called by an application process and the value of the state variable is modified, the logical timestamp will increase by 1 and the state variable of application virtual context is set to the current value. But no commands are packed to command buffer. Each state modification command has a corresponding state tracking function. Figure 3 shows the tracking of lightf on G-node.

Before geometry packets are transmitted to remote R-nodes, context difference is executed between application virtual context and the active virtual context. Context difference converts different states into packing functions of OpenGL state commands when the logical timestamp of application virtual context and that of active virtual context are unequal. These OpenGL calls are packed into OpenGL command buffers and sent to the destination over network so that only minimum parts of the OpenGL context are packed. Figure 4 is a portion of context difference of light and material.

When the timestamp of application virtual context context overflows, timestamps of all contexts are hashed to 0 to 1024. Timestamps of application virtual context is set to 1024. If the timestamp is hashed to existing hash item and the values are unequal, a new item will be inserted into hash table.

Context synchronization is performed after context difference. Unlike context difference, application virtual context is simply copied to active virtual context if the logical timestamps of them are unequal. After context synchronization, the logical timestamp of active virtual context equals to that of application virtual context.

4.2. State Tracking On Geometry Rendering Node

R-node maintains M+1 virtual contexts, where M is the number of its connected G-nodes. An application virtual context represents the current hardware context. Other virtual contexts represent the current graphics contexts of the connected G-nodes. A virtual con-text is set active when R-node receives a geometry buffer from the corresponding G-node whose context the virtual context represents. Software

© The Eurographics Association 2002.

```
glLight_Diff(src, dest){
   if timestamp of src, dest are equal return;
   if (lighting.timestamp of src,dest not equal &
      dest.lighting.enable)
         pack glEnable(GL_LGHTING);
      else
         pack glDisable(GL_LGHTING);
   if( dst.lighting){
   for(I = 0; I I< num_lights, I++)
      if(light[I].timestamp of src,dest not equal){
         pack changed parameters    of dest    into geom  etry
               packet;
      }
   }
   call difference of material;
}
```

Figure 4: *A portion of code for difference of light and material. The different states are packed into geometry buffers.*

context switch occurs when a R-node receives a command packet from a G-node different with the last G-node. The process of software context switch is similar to that of context difference while software context switch calls OpenGL hardware API. If the two logical timestamps of two contexts' corresponding variables are equal, no state modification commands will be called. Otherwise, a set of state modification commands are called to change the hardware context and the logical timestamp of application virtual application increases by 1 and the logical timestamp of active virtual context is set equal to that of application virtual context. When software context switch is completed, the active virtual context is synchronized with application virtual context. Figure 5 shows a port of code for software context switch of transformation. When a state modification command is

```
glTransform_switch    (src, dest){
   if(transform   .timestamp of  src ,dest        ar e  equal)
      return   ;
if(modelview  .timestamp    of src, dest not equal     ){
   //call hardware api
   glLoadMartix(   dest _model_view);
   src. modelview   = dest .modelview;
   src.  modelview  .timestamp =
     dest.     modelview  .timestamp;
   }
   call  projection switch     of src , dest;
}
```

Figure 5: *A portion of code for software context switch of transformation. Software context switch occurs when timestamp of src and dest are not equal.*

called, the timestamp of application virtual context will add 1 and R-node calls the hardware state modification calls. The whole process is very similar to the state tracking of G-nodes except that OpenGL hardware commands are called while packing of state commands is called by state tracking of G-node.

Although the state tracking based on logical timestamp also uses lazy update of WireGL, this algorithm does not limit the node number of G-nodes and R-nodes. Furthermore, it is easier of implementing and programming.

5. Compression of Command Code, Geometry and Image

5.1. Overview of Compression

Many methods are available for data compression. RLE is very efficient for grayscale image compression. LZW compresses text effectively. Quantization is good at numerical compression. Image compression has been studied for a long time. JPEG is a popular standard of loss-quality image compression. There are many re-searches on video compression, where MPEG-4 compresses video in high ratio and good quality quickly.

Geometry compression has got its focus since Deering[27] escribed two problems of hardware rendering as geometry compression and topology compression. He gave some basic methods for the compression of normal, color and position. Later researches on geometry compression are focused on topology compression. Buck et al.[19] tried normal and position compressions where only simple linear prediction and quantization are adopted.

5.2. Command Code Compression

Not only the frequently called commands are primitive commands such as glBegin, glEnd, glVertex, glColor, glNormal and glTexCoord, but these commands appear in fixed orders and modes. LZW is used for command code compression since it dynamically generates dictionary. Command code compression ratio exceeds 1:4 in lots of examples.

5.3. Geometry Compression

Network bandwidth of 100∼200 MB/s such as Myrinet[26] is low compared with that of system bus of 1∼2 GB/s. The low bandwidth challenges the scalability and limits the performance of latest modern graphics hardware. We choose normal, color and position for compression in AnyGL.

Generally geometry compression will decrease the precision of geometry primitive attributes. To satisfy the precision requirements of various applications, programmers can adjust geometry compression ratio in AnyGL.

Normal compression described by Deering is implemented in AnyGL. A lookup table is designed for fast normal compression and decompression.

© The Eurographics Association 2002.

Position compression. An adaptive position compression scheme is used in AnyGL to determine the compressed bits. AnyGL compresses vertex position before transmission of command packets. The compression precision will be adjusted for user applications. DPCM model is employed to compress x-y-z positions which are the parameters of glVertex3f. Four types of predictors are designed for 10 kinds of different OpenGL primitives.

(1)Linear prediction is used to predict the next vertex when the primitives are GL_POINT, GL_LINES, GL_LINESTRIP, GL_TRIANGLES and GL_QUADS. The next vertex can be predicted by previous k vertices according to equation (1).

$$P_n(\lambda, V_{n-1}, ..., Vn-k) = \sum_{i-1}^{k} \lambda_i V_{n-i} \qquad (1)$$

where P_n is the predicted position of the nth vertex,$\lambda = (\lambda_1, \lambda_2, ..., \lambda_k)$ are the coefficients and $V_1, V_2, ..., V_{n-1}$ represent positions of previous n-1 vertices. (2) Circle prediction. Primitives GL_POLYGON and GL_LINE_LOOP have same characteristics so that they tend to 'close' to form an n-edge polygon. Figure 6 shows the algorithm of this predictor. Given $V_1, V_2, V_{n-1}, P_{n-1}$ will continue the trend like previous vertices, which means stepping forward an average length with same deflection angle. Equation(2) is used for calculating P_n.

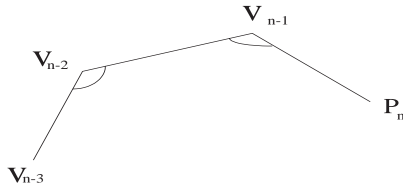

Figure 6: *Circle law prediction for primitives of GL_POLYGON and GL_LINE_LOOP. V_{n-3}, V_{n-2} and V_{n-1} are previous vertices, and $\alpha isthedeflectionangle.P_n$ is the predicted position.*

$$|V_{n-1}P_n| = \frac{|V_{n-2}V_{n-1}| + |V_{n-3}V_{n-2}|}{2} \qquad (2)$$

(3) Parallelogram prediction[30]. Triangle stripes are mostly used and the parallelogram prediction is good at their prediction as shown in Figure 7 While the parallelogram prediction suggests that a triangle strip tends to step forward evenly as the vertices issue, quad stripes have the same characteristics. So a quad is predicted by using parallelogram prediction twice as shown in (d) of Figure 7. The predicted position is computed by equation(3).

$$P_n = V_{n-2} + V_{n-1} - Vn - 3 \qquad (3)$$

(4) Triangle fan prediction. Triangle fans also tend to close as a n-edge polygon. As shown in Figure 8, P_n tends to go forward a certain angle at the direction which the fan opens. V_1 is the center vertex position of triangle fan and V_{n-2} and V_{n-1} are two previous issued vertices. is the deflection angle of this prediction. P_n is the predicted position. P_n is computed by equation(4).

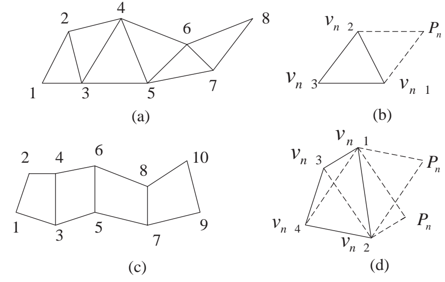

Figure 7: *Parallelogram Prediction, where (a) are triangle strips, (b) is parallelogram prediction, (c) are a quad strips and (d) represents by using parallelogram prediction twice. Number 1∼9 represent the issue order of these vertices. $V_{n-4}, V_{n-3}, V_{n-2} and V_{n-1}$ are previous issued vertices, P_n and P_{n+1} are the predicted positions.*

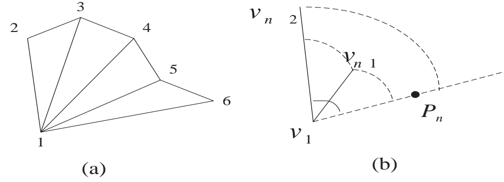

Figure 8: *Triangle Fan Prediction, where (a) shows a triangle fan, (b) is the predicted P_n.*

$$|V_1 P_n| = \frac{|V_1 V_{n-1}| + |V_1 V_{n-2}|}{2} \qquad (4)$$

Overflow is an unavoidable problem for numerical quantization. The maximum value of the quantization marks quantization overflow occurs. For instance, 10-bits compression value ranges from 0∼1022 while 1023 represents failure of quantization and new quantization process will be initialized for following compression.

5.4. Image Compression

There are lots of image compression algorithms which are developed for many years. RLE and JPEG-LS are good lossless compression methods for grayscale images. HUFFYUV[28] shows high compression ratio from 1:2 to 1:6 for lossless color image compression in real-time.

VCM[29] is employed in AnyGL for image compression on R-nodes, C-nodes and D-nodes. Programmers can choose different image compression algorithms or install new compression algorithms. HUFFYUV is the default compression

© The Eurographics Association 2002.

algorithm. The name of compression algorithm is embedded in image transmission protocol packets so that each node is able to choose corresponding compression algorithm independently. The decompression node retrieves the name of compression algorithm from the packet and initializes compression processing.

6. Global Share Extension for Parallel Graphics

Global share extension is different from the parallel graphics programming interface[21] which focuses on solving the bottleneck of input rate and order immediate-mode API for parallel rendering. Global share extension emphasizes on the problem of memory explosion in distributed and parallel rendering.

6.1. Memory Explosion

There is an obvious memory explosion for parallel graphics systems without global share of textures. For example, a parallel application consists of N G-nodes and N R-nodes using M different textures. Each G-node stores a copy of M different textures. However, each R-node must store N copies of textures generated by each G-nodes and memory allocation for each R-node is magnified to N times. In simple words, each R-node will allocate *M* MB memory to store textures when each texture occupies 1 MB memory. Display list slows down the efficiency of parallel graphics applications for same reason.

Igehy et al.[24] have designed texture share scheme at texture mapping stage for parallel graphics, but no interface of global share is provided for distributed/parallel graphics. The problem of memory explosion still exists when an application inputs scene from multi-processes. Global share of textures will speed up the texture access and decrease the overhead of context switch of textures.

6.2. Global Share Extension in AnyGL

Two extensions are implemented in AnyGL to solve the previous problem, global share of textures and display lists. When several processes use same texture simultaneously, the texture will be defined as global share. Figure 10 shows the basic scheme for global share. R-node maps the global share texture to same texture as shown in Figure 9. The global share extension of textures is marked by the parameter GL_GLOBAL_TEXTURE_nD, where n is 1, 2 and 3. AnyGL adopts similar state tracking method for global share textures as local textures. R-nodes determine global share textures are equal only if their global share ID and filter parameters are equal.

Two kinds of global share managers are implemented where one resides on G-node and the other on R-node. Every G-node can define global share textures separately. The

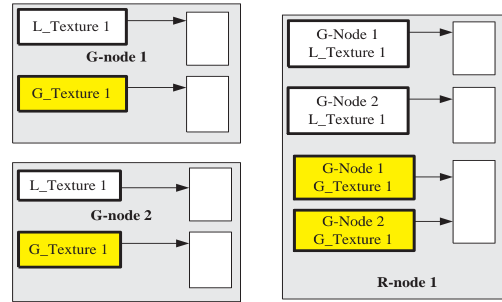

Figure 9: *Global Share of Textures. L_Texture is a local texture and G_Texture represents the global share texture. Global shared texture G_Texture 1 from G-node 1 and G-node 2 is linked to same memory location in R-node while local textures are linked to different memory locations.*

transmission of global share textures is similar to that of local textures except a global share mark is added to the transmission protocol. When R-node receives a global share texture definition and the global share texture with same ID exists, R-node maps this texture to the memory location of the existing global share texture, otherwise global share manager creates a new texture. Memory explosion is avoided by global share extension scheme.

Global share of display lists are treated in the similar method. OpenGL 1.3 does not provide object definitions for vertex array and display list while OpenGL 2.0[32] introduces more object types which application can access.

7. Test Results Analysis

7.1. State Tracking of AnyGL vs. WireGL

Two major parameters are defined for performance measurement of state tracking. They are overhead of context difference and overhead of software context switch. Context difference and soft-ware context switch are the most time-consuming processes for state tracking. Context synchronization is always completed within a few hundreds of nanoseconds. Seven applications are chosen to test the overhead of state tracking both in AnyGL and WireGL. Three types of PCs are selected to test the performance of state tracking. Type A is equipped with a Celeron II 800Mhz CPU, 100Mhz bus and 256 MB SDRAM memory but no 3D graphics accelerated card. Type B consists of a Pentium III 600Mhz CPU, 256 MB SDRAM, 100Mhz bus and an ASUS V6800 VGA card. Type C uses a Pentium IV 1.5Ghz, Intel 850 motherboard, 512 MB RAMBUS and a Geoforce 3 display card. Table 1 shows the results of context difference on above hardware platforms. Time is measured in microseconds. Table 2 shows the results of overhead test of software context switch.

The results show that AnyGL will run as fast as WireGL

© The Eurographics Association 2002.

Application	PIII 600Mhz		PIV 1.5Ghz	
	AnyGL	WireGL	AnyGL	WireGL
atlantis	5.199	4.657	4.408	4.408
gears	4.502	4.237	3.392	3.620
ideas	5.607	5.036	4.390	4.333
moth	4.193	3.664	3.318	3.077
rc	7.126	6.127	4.995	4.466
worms	4.547	4.547	3.667	3.308

Table 1: *Context Difference of AnyGL vs. WireGL (unit: microsecond).*

Application		PIII 600Mhz		PIV 1.5Ghz	
		AnyGL	WireGL	AnyGL	WireGL
parview	max	180.470	306.464	115.254	185.324
	min	1.956	1.676	1.118	1.118
	average	3.867	3.764	2.354	2.365

Table 2: *Context Switch of AnyGL vs. WireGL(unit: microsecond). This application involves lots of transformations, materials, lightings and polygon modes. All meshes are represented by triangle stripes and each triangle stripe is set with individual material. We execute this application with 4 G-nodes and 4 R-nodes. Each G-node sets up its graphics context individually.*

on both context difference and context switch. AnyGL is able to switch context by software about half million times within a second. Also approximate linear speed-up of state tracking is obtained when advanced hardware are adopted for AnyGL as shown in Table 1 and Table 2.

7.2. Benefits of Global Share Extension

Visualization and simulation applications always use lots of texture instead of complex meshes, for example, flight simulators and ship simulators occupy hundred millions bytes of texture memory for one scene. Global share extension will decrease the memory allocation of duplicated textures when cluster rendering is adopted for these applications. It is true that global share extension is efficient for parallel and distributed graphics applications. When no global share extension is adopted, memory are exhausted for cluster terrain rendering and city walkthroughing.

7.3. Geometry Compression

Tree meshes, teapot, bunny, and dragon, are tested for geometry compression of AnyGL. Teapot and bunny are arranged as triangles while dragon is organized into triangle strips. To obtain satisfying visual effects, normal is compressed into 17.66-bits and position is compressed into 36-bits. Table 3 is the compression result per frame. Figure 10 shows the images rendered by original data and compressed data. Real-time geometry compression will cause some splits which are are shown as black points in Figure 10 of (b) and (d) . The reason is that a same vertex will be compressed into different values when this vertex exists in different triangles. This fact decreases the usability of geometry compression for distributed rendering applications unless an artful algorithm is used to enforce the compressed vertex to be adjusted at predefined 3D grid cell when this vertex is shared by several triangles.

	original data size	compressed data size	compression rate
teapot	45500	13247	0.29
bunny	2481996	751745	0.30
horse	3330236	1035533	0.31
dragon	33831108	13949870	0.36

Table 3: *Result of Geometry Compression. Original data size is the data size per frame without geometry compression. Compressed data size is the data size per frame when op-code is compressed by LZW, normal is compressed into 17.66-bits and position is compressed into 36-bits.*

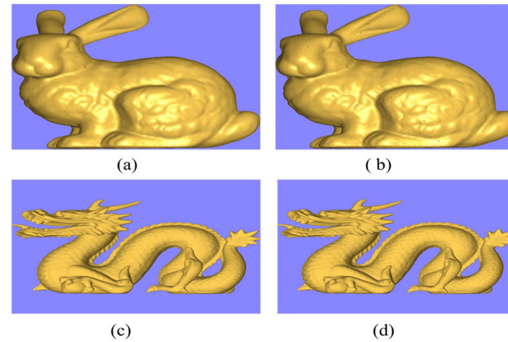

Figure 10: *Geometry Compression., where (a) and (c) are pictures rendered by original meshes, (b) and(d) are rendered by geometry compression. Original picture size is 800 X 600.*

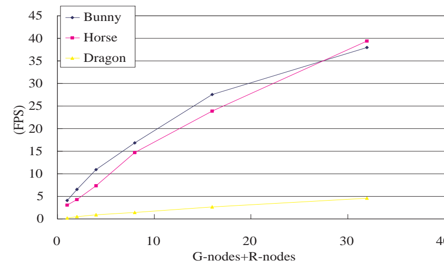

Figure 11: *Simulation result of AnyGL on 100Mbps switched network. G-nodes and R-nodes vary by 2, 4, 8, 16, 32.*

© The Eurographics Association 2002.

7.4. Scalability

Scalability of parallel system is divided into three parts, component scalability, system size scalability and problem size scalability. AnyGL shows good scalability in these three aspects by simulations.

AnyGL shows good capability in component scalability. AnyGL packs 6,602,667 OpenGL primitive commands per second on Celeron II 800Mhz, 10,777,125 commands on Pentium III 600Mhz and 17,084,192 commands on Pentium IV 1.5hz. The performance of state tracking is increased with the improvement of the performance of PC platforms shown in Table 1 and Table 2. Rendering capability speeds up linearly if latest powerful graphics accelerated adapter is equipped for AnyGL.

Figure 11 is the simulation result of AnyGL on 100Mbps switched network. Since we have not built a real system to verify the scalability of AnyGL, a simulation program is built to test its performance. A benchmark will be programmed to verify the scalability of AnyGL on real system in the near future.

Three kinds of scene size problems are solved in AnyGL. They are known as number of triangles, texture memory size and computation-limited applications. Huge scenes consisting of hundred millions of triangles will be packed by multi-processes at the same time. Some applications require remote rendering at large-scale. Global share extension is good at minimizing texture memory cost when the application renders on distributed and parallel graphics architectures.

8. Conclusions & Further Research

AnyGL is designed for large-scale distributed rendering and parallel rendering based on PC cluster. Several new methods benefits cluster rendering application on performance and scalability. First, state tracking based on logical timestamp breaks up the scalability of G-nodes and R-nodes. Second, Sort-first and sort-last parallel rendering architectures are integrated. Three kinds of compressions also will increase efficiency when limited network bandwidth is used. Global share extension avoids the problem of memory explosion when lots of global textures are used in parallel graphics applications.

Although AnyGL provides a solution for large-scale distributed rendering, several problems will still exist. Dynamic configuration will be implemented in future research so that application will maintain load balance at run time. The following questions will be considered as future research on cluster rendering and parallel graphics architecture.

The greatest latency occurs during reading back and transmitting of color and depth data. It always takes 10 microseconds or more to read back a 1024 X 768 image for AGP 2X. There are some designs for fast color data transmission such as Metabuffer[18], Lightning-2[14] and sepia[10]. Both AGP and PCI bridges are required for color data composition and depth data transmission.

Although Chromium[33] has introduced streams into cluster rendering, it is not designed for programmable GPUs. OpenGL 2.0[32] simplifies the extensions of OpenGL 1.3 and provides a shading language of high level API. New state tracking method is necessary for parallel rendering of OpenGL 2.0 and global share extension should be expanded for the eight types of new objects defined in OpenGL 2.0.

Direct3D is different from OpenGL since it is built on COM. Thus, it will be interesting to build a cluster rendering system on Direct3D.

This work is supported by the key project No. 60033010 and the program for Innovative Research Group No.60021201 of NSF of China.

References

1. Akeley, K. RealityEngine Graphics. *Computer Graphics (Proceedings of SIGGRAPH 1993)*, 27, pages 109-116
2. Greg Humphreys, Matthew Eldridge, Ian Buck Gordon, Stoll Matthew Everett, Pat Hanrahan. WireGL: A Scalable Graph-ics System for Clusters. *Proceedings of SIGGRAPH 2001*, August 2001.
3. S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs. A Sorting Classification of Parallel Rendering. *IEEE Computer Graphics and Algorithms*, pages 23-32, July 1994.
4. J. Montrym, D. Baum, D. Dignam, and C. Migdal. InfiniteReality: A Real-Time Graphics System. *Proceedings of SIGGRAPH 97*, pages 293-302, August 1997.
5. H. Fuchs, J. Poulton, J. Eyles, T. Greer, H. Goldfeather, D.Ellsworth, S. Molnar, G. Turk, B. Tebbs, and L. Israel. Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories. *Proceeding of SIGGRAPH 89*, pages 79-88, July 1989.
6. M. Eldridge, H. Igehy, and P. Hanrahan. Pomegranate: A Fully Scalable Graphics Architecture. *Proceedings of SIGGRAPH 2000*, pages 443-454, July 2000.
7. S. Molnar, J. Eyles, and J. Poulton. PixelFlow: High-Speed Rendering Using Image Composition. *Proceedings of SIG-GRAPH 92*, pages 231-240, August 1992.
8. Freedom 3000 Technical Overview. Technical report, Evans Sutherland Computer Corporation,October 1992.
9. *Denali Technical Overview*. Technical report, Kubota Pacific Computer Inc., March 1993.

© The Eurographics Association 2002.

10. A. Heirich and L. Moll. Scalable Distributed Visualization Using Off-the-Shelf Components. *IEEE Parallel Visualiza-tion and Graphics Symposium*, pages 55-59, October 1999.

11. OpenGL stream codec specification. http://www.opengl.org/Documentation/Specs.html.

12. M. Kilgard. GLR, an OpenGL Render Server Facility. *Pro-ceedings of X Technical Conference*, February 1996.

13. M. Kilgard. OpenGL Programming for the X Window Sys-tem. Addison-Wesley,1996.

14. G. Stoll, M. Eldridge, D. Patterson, A. Webb, S. Berman, R.Levy, C. Caywood, M. Taveira, S. Hunt, and P. Hanrahan. Lightning-2: A High-Performance Display Subsystem for PC Clusters. *Proceedings of SIGGRAPH 2001*, August 2001.

15. Peter Kipfer. Distributed Lighting Networks. Technical Re-port, University of Erlangen-NÍźrnberg, 1999.

16. Mengzhou Yang, Xiaohong Jiang, Zhigeng Pan, Jiaoying Shi. MUDVE: Implement of A Multi-User Distributed Virtual Environment. *Journal of Computer-Aided Design Com-puter Graphics*, **13**(2):173-178,2001.

17. Tulika Mitra, Tzi-cker Chiueh. Implementation and Evaluation of Parallel Mesa Library. *IEEE International Confer-ence on Parallel and Distributed Systems*, December 1998.

18. W. Blanke, C. Bajaj, D. Fussel, and X. Zhang. The Me-tabuffer:A Scalable Multiresolution Multidisplay 3-D Graphics System Using Commodity Rendering Engines. TR2000-16, University of Texas at Austin, February 2000.

19. I. Buck, G. Humphreys, and P. Hanrahan. Tracking Graphics State for Networked Rendering. *Proceedings of SIG-GRAPH/Eurographics Workshop on Graphics Hardware*, August 2000.

20. Digital Visual Interface Specification. http://www.ddwg.org.

21. H. Igehy, G. Stoll, and P. Hanrahan. The Design of a Parallel Graphics Interface. *Proceedings of SIGGRAPH 98*, pages 141-150, July 1998.

22. G. Humphreys, I. Buck, M. Eldridge, and P. Hanrahan. Dis-tributed Rendering for Scalable Displays. *IEEE Supercomputing 2000*, October 2000.

23. R. Samanta, J. Zheng, T. Funkhouser, K. Li, and J. P. Singh. Load Balancing for Multi-Projector Rendering Systems. *Proceedings of SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 107-116, August 1999.

24. H. Igehy, M. Eldridge, and P. Hanrahan. *Parallel Texture Caching. Proceedings of SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 95-106, August 1999.

25. OpenGL Specifications. http://www.opengl.org/Documentation/Specs.html.

26. N. Boden,D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J.Seizovic, and W. Su. Myrinet: A Gigabit-per-second Local Area Network. *IEEE Micro*,pages 29-36, February 1995

27. M. Deering, Geometry Compression, *Computer Graphics, Proceedings Siggraph'95*, 13-20, August 1995.

28. Ben Rudiak-Gould. Huffyuv v2.1.1 Manual. http://www.math.berkeley.edu /∼ benrg/huffyuv.html.

29. Video Compression Manager. Manual of Windows SDK. Microsoft Corporation, 2001.

30. C. Touma and C. Gotsman. Triangle Mesh Compression, *Proceedings Graphics Interface 98*, pp. 26-34, 1998.

31. R. Samanta, T. Funkhouser, K. Li, and J. P. Singh. Sort-First Parallel Rendering with a Cluster of PCs. *SIGGRAPH 2000 Technical Sketch*, August 2000.

32. OpenGL 2.0 Specification Proposals. http://www.3dlabs.com/support/developer/ogl2/index.htm.

33. Greg Humphreys, Mike Houston, Yi-Ren Ng Randall Frank, Sean Ahern Peter Kirchner, Jim Klosowski. Chromium: A Stream Processing Framework for Interactive Rendering on Clusters. To Appear in *SIGGRAPH 2002*.

© The Eurographics Association 2002.