XÂY DỰNG MÔ HÌNH XỬ LÝ SONG SONG, ĐA MÀN HÌNH TRONG KỸ THUẬT ĐỒ HỌA 3D

Nguyễn Văn Trường, Nguyễn Công Định
Trung tâm Công nghệ Mô phỏng, Học viện KTQS

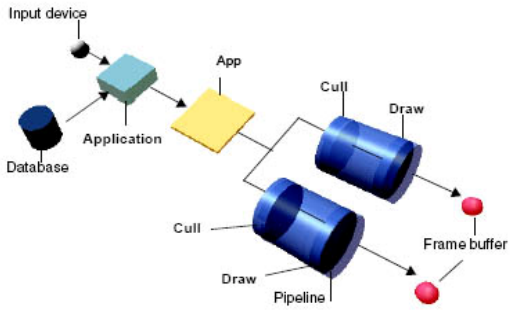
Tóm tắt.

Trên cơ sở thư viện đồ họa chuyên dụng ta có thể hiển thị nhanh và cải thiện chất lượng các cảnh đồ họa 3D nhờ các thuật toán chọn lọc, sắp xếp phân loại cùng một số thuật toán khác. Bài báo đề cập đến một mô hình xử lý theo chế độ đa nhiệm song song, đa màn hình hiển thị. Mô hình ứng dụng thư viện mã nguồn mở OpenSG cho phép tối ưu dữ liệu và tăng tốc độ hiển thị đồ hoạ 3D khi dữ liệu lớn hỗ trợ phát triển nhanh các ứng dụng đồ hoạ chất lượng cao.

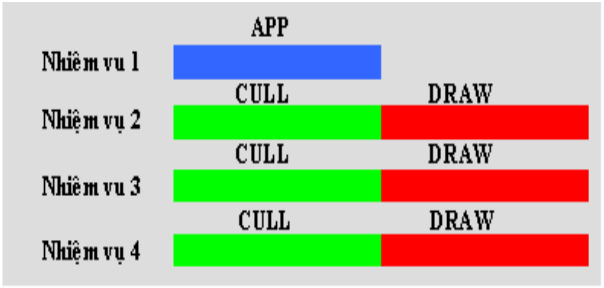
Từ khóa: song song, đa luồng, đa màn hình, đồ họa 3D, mô phỏng, thực tại ảo.

1. Đa nhiệm, hiển thị đa luồng và nhiều màn hình hiển thị theo phương pháp truyền thống:

Phương pháp truyền thống khi hiển thị cảnh đồ họa 3D thời gian thực sử dụng ba pha riêng biệt APP, CULL, DRAW: APP làm nhiệm vụ cập nhật dữ liệu động, bao gồm vị trí camera, vị trí của các đối tượng chuyển động, CULL phụ thuộc vào APP làm nhiệm vụ lọc cảnh và sắp xếp đối tượng theo độ ưu tiên trong khung nhìn để tăng tốc độ hiển thị cảnh đồng thời tùy thuộc vào việc cập nhật vị trí của camera, tạo dữ liệu hiển thị theo kiểu danh sách để pha DRAW vẽ cảnh lên màn hình. Quá trình vẽ là quá trình duyệt qua danh sách và thông báo cho OpenGL xử lý ta có thể xem hình 1 dưới đây.



Hình 1: Ba pha trong hiển thị đồ họa thời gian thực



Hình 2 - Chia những pha thành những nhiệm vụ song song cho hệ thống có nhiều màn hình hiển thị đồ họa

Trong một hệ thống có nhiều màn hình hiển thị đồ họa, CULL và DRAW trở nên rất cần thiết vì các pha này sẽ sản sinh ra danh sách hiển thị và thực hiện vẽ trên các màn hình ở các khung nhìn khác nhau. Tuy nhiên trong hệ thống đó vẫn chỉ cần một pha APP chung để cập nhật dữ liệu. Dưới đây là trình tự yêu cầu cần thực hiện theo quan điểm đa nhiệm cho nhiều màn hình hiển thị. Với mô hình máy đơn cần xử lý các pha theo từng thời kỳ (APP, CULL0, DRAW0, CULL1, DRAW1, CULL2, DRAW2…). Theo trình tự này cần nhiều thời gian để thực hiện một khung hình qua trình tự thực hiện các pha.

Để phân nhiệm ta định ra hai nhiệm vụ sau được miêu tả như ở hình 2:

- Nhiệm vụ thực hiện pha APP.
- nhiệm vụ CULL / DRAW cho mỗi màn hình.

Với hệ thống đa xử lý, tất cả các nhiệm vụ có thể được thực hiện song song trên từng bộ xử lý riêng biệt. Hơn nữa, CULL / DRAW có thể được chia ra từng phần để có thể chạy song song trong hệ thống.

Có hai mô hình thực hiện trên hệ thống xử lý song song:

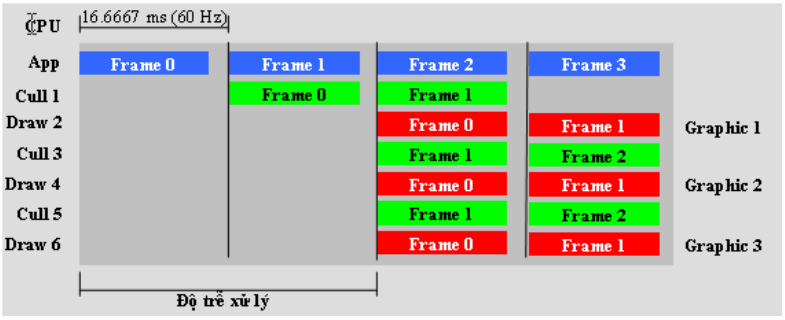
- Chia cắt một nhiệm vụ lớn thành nhiều nhiệm vụ nhỏ độc lập để thực hiện song song từng nhiệm vụ trên mỗi máy để giảm bớt thời gian xử lý.
- Thực hiện nhiệm vụ lớn song song trên các máy sao cho đồng bộ về thời gian trên hệ thống.

Với hai mô hình này ta tách pha CULL/DRAW từ pha APP, rồi sau đó tiếp tục chia pha CULL và DRAW thành nhiều nhiệm vụ chạy song song riêng biệt theo mô hình thứ nhất. Ghép CULL/DRAW thành một nhiệm vụ kép cho mỗi hệ thống hiển thị theo mô hình thứ hai.

Nhưng xuất hiện vài vấn đề ở từng giai đoạn khi chạy song song. Trước hết, những pha phải xử lý dữ liệu ra từng kỳ. Nghĩa là pha APP phải kết thúc làm việc trên dữ liệu trước pha CULL. Tương tự như vậy pha DRAW không thể bắt đầu xử lý dữ liệu khi mà pha CULL chưa làm việc xong. Tuy nhiên, APP không cần phải đợi cho đến khi cả hai pha CULL và DRAW làm việc xong mà vẫn có thể xử lý dữ liệu ở khung hình tiếp theo và hệ thống có thể vận hành theo như hình 3 mô tả dưới đây. Bên cạnh đó dữ liệu dùng chung giữa hai giai đoạn phải được bảo vệ hoặc dùng bộ đệm. Bên cạnh đó dữ liệu đang ghi bởi pha ở giai đoạn này không thể đọc ở cùng pha chạy song song trên hệ thống. Điều này đòi hỏi cần có hệ thống quản lý dữ liệu lớn để xử lý dữ liệu 3D. Đây chính là mô hình của hãng SGI đưa ra và có hiệu quả trong các sản phẩm đồ họa của hãng như ở hình 3. Nhưng vài năm gần đây đã trở lên lạc hậu do một vài lý do sau:

- Vấn đề thời gian thực, khi các sản phẩm mô phỏng yêu cầu độ hiển thị khung hình là 60 Hz nghĩa là mỗi khung hình cần 16.667 mili giây để các pha thực hiện xong nhiệm vụ của nó. Vào những năm 90, SGI đã phát triển kỹ thuật đồ họa thời gian thực với những bộ xử lý có thể đạt các yêu cầu trên nhưng hệ thống máy tính có tốc độ thấp hơn so với hiện tại. Trong khi tốc độ đồ họa phụ thuộc rất nhiều vào hai pha APP và CULL.
- Hơn thế nữa, hệ thống băng thông rộng phát triển giảm bớt thời gian liên thông giữa các pha ở máy chủ và các máy trạm, cùng với kỹ thuật xử lý đa luồng sẽ có những kết quả khả quan mới.

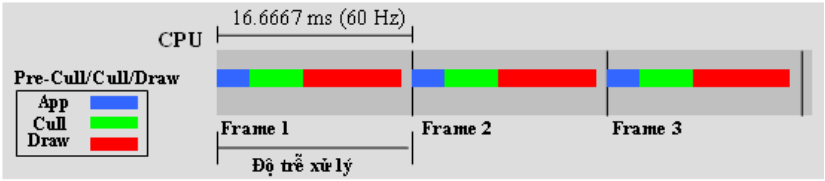
- Một vấn đề cuối cùng rất cần thiết đó là những yêu cầu đối với thiết bị mô phỏng là phải đáp ứng được các tương tác của người dùng. Vì vậy yêu cầu hình ảnh trực quan và sinh động với tốc độ hiển thị thời gian thực.



Hình 3 - Mô hình xử lý song song "Truyền thống"

2. Cách tiếp cận mới

Các ứng dụng đồ họa 3D chất lượng cao chạy trên phần cứng hiện thời mong muốn có số khung hình là 60 cần phải có thời gian xử lý của pha Pre - CULL (Có thể hiểu như pha APP – theo mô hình cũ) và CULL nằm trong khoảng 1 mili-giây đến 3,5 mili-giây để thực hiện một khung hình. Xét hệ thống xử lý đơn, một màn hình hiển thị. Với yêu cầu giảm bớt thời gian trên pha Pre – CULL và CULL, sơ đồ các pha có dạng như hình 4.



Hình 4 - Bộ xử lý đơn, một màn hình hiển thị theo mô hình pha Pre – CULL và CULL

Qua sơ đồ trên ta thấy tất cả các pha đều nằm trong một khung hình, và như vậy độ trễ có thể giảm được trong mỗi khung hình. Nhưng pha DRAW chiếm quá nửa thời gian thực hiện một khung hình. Các ứng dụng đồ họa có các lợi thế từ các đồ thị khung cảnh, mà tại đây pha CULL sẽ loại bỏ các đối tượng không thuộc khung hình cần hiển thị để đưa vào các nhánh của đồ thị nhằm tối ưu hóa dữ liệu.

Xét mô hình đa xử lý với nhiều màn hình hiển thị. Cần phải tận dụng hết lợi ích của hệ thống đa xử lý bằng việc sử dụng luồng chính chạy pha Pre - CULL, và các pha CULL/DRAW ở các hệ thống con. Để làm được điều này chúng ta giả thiết hai khía cạnh về quản lý dữ liệu :

1) Dữ liệu được ghi bởi pha Pre –CULL có thuộc tính toàn cục.

2) Dữ liệu được sinh ra ở các pha CULL mang tính cục bộ nhưng có thể tiếp cận được bởi mỗi cặp nhiệm vụ CULL/DRAW.

3. Ứng dụng thiết kế Mô hình xử lý song song

Mô hình tổng quát được thiết kế theo sơ đồ chung như trên hình 5:

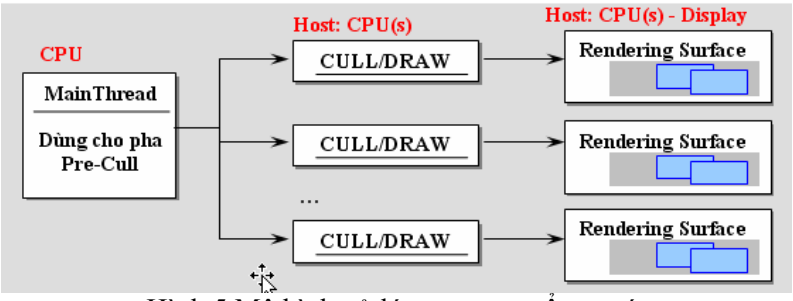
Luồng Chính. Luồng thực hiện Pre - CULL. Khai báo trong hệ là một CPU đảm nhiệm nhiệm vụ đó.

CULL / DRAW. Có thể chạy như một luồng đơn, hoặc những luồng riêng biệt phụ thuộc vào mô hình xử lý được chọn từ mục trước. Điều này có thể được xác định bởi hostname trong hệ thống, và một tham số chỉ định CPU nào trên hệ thống sẽ làm việc với nó để hiển thị cảnh.

Rendering Surface(Trả lại Bề mặt)

Đây là kết quả hiển thị đồ họa trên màn hình máy tính và được xác định bằng các tham số sau:

Host	Tên máy trạm nơi sẽ thực hiện kết quả hiển thị đồ họa
Display	Màn hình hiển thị
Viewport	Viewport là khung hình hiển thị bên trong bàn hình hạn chế hình ảnh hiển thị

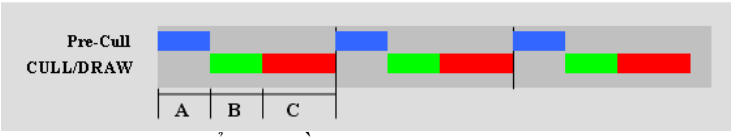


Hình 5 Mô hình xử lý song song tổng quát

Có hai mô hình trong mục trước đã đề cập để thực hiện mô hình đa nhiệm (multi-task), đa màn hình hiển thị đồ họa (multi – display). Sự khác nhau là ở chỗ có quyết định ghép hai pha CULL/DRAW thành một hay không. Ở đây chúng tôi đưa ra hai phương pháp mỗi phương pháp có những đặc tính riêng.

Mô hình A

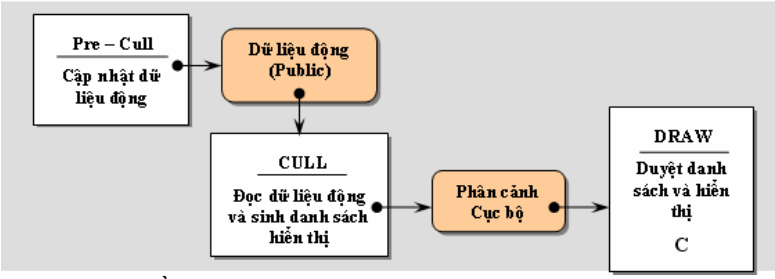
Mô hình này ghép hai pha CULL/DRAW thành một nhiệm vụ kép được miêu tả theo sơ đồ như hình 6 dưới đây.



Hình 6: Ba pha hiển thị đồ họa thời gian thực theo mô hình A

Mô hình này giả thiết một khung hình được thực hiện theo thứ tự, và dùng một luồng cho nhiệm vụ kép CULL/DRAW. Thời gian mỗi tiến trình B và C được thực hiện theo sơ đồ hình 7. Pha

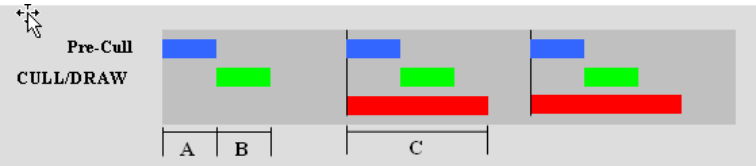
Pre – Cull cập nhật dữ liệu động trong đồ thị khung cảnh. Dữ liệu động này bao gồm vị trí camera, định vị những đối tượng chuyển động bên trong cảnh, số khung hình thời gian trôi qua, và đồng bộ những phương tiện quản lý dữ liệu khác. Dữ liệu này được giả thiết là dữ liệu toàn cục, được cấp phát và có thể lấy được bởi ứng dụng. Như vậy, pha CULL phải đợi cho đến khi Pre – CULL kết thúc tiến trình. Khi pha Pre – Cull thực hiện xong sẽ báo hiệu cho pha CULL thực hiện. CULL sẽ đọc dữ liệu động đã được cập nhật, và sinh dữ liệu để hiển thị, dữ liệu này mang tính cục bộ không cho ứng dụng tiếp cận nhưng có thể lấy được từ pha DRAW. Dữ liệu này được xử lý và phân ra từng kỳ. Pha DRAW sẽ duyệt qua danh sách và hiển thị cảnh đồ họa.



Hình 7. Sơ đồ miêu tả quan hệ dữ liệu giữa các pha ở mô hình A

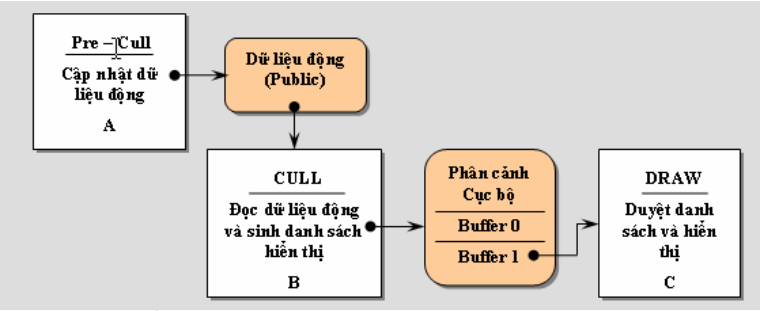
Mô hình B

Mô hình này tách riêng những luồng riêng biệt cho hai pha CULL / DRAW như hình 8.



Hình 8: Ba pha hiển thị đồ họa thời gian thực theo mô hình B

Dữ liệu cho mô hình này được mô tả theo sơ đồ hình 9 sau đây.



Hình 9. Sơ đồ miêu tả quan hệ dữ liệu giữa các pha ở mô hình B

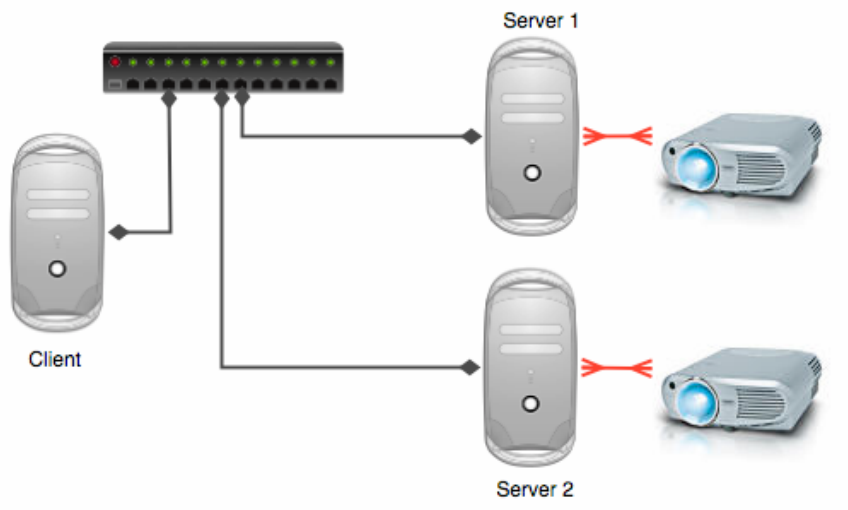
Sơ đồ này khác với sơ đồ của mô hình A là dùng một luồng cho nhiệm vụ kép CULL/DRAW . Ở đây pha DRAW sẽ không duyệt dữ liệu được cung cấp bởi pha CULL một cách trực tiếp mà sẽ qua hai bộ đệm. Dữ liệu phát sinh từ pha CULL sẽ được ghi vào bộ đệm Buffer 0 trong khi pha vẽ sẽ đọc dữ liệu từ bộ đệm Buffer 1. Khi đồng bộ giữa hai pha CULL và DRAW những

con trỏ chỉ tới những bộ đệm sẽ được trao đổi. Cách tiếp cận này yêu cầu khi phân cảnh ở pha CULL cần có hai bộ đệm cục bộ, và phải bổ sung việc đồng bộ giữa hai pha CULL và DRAW.

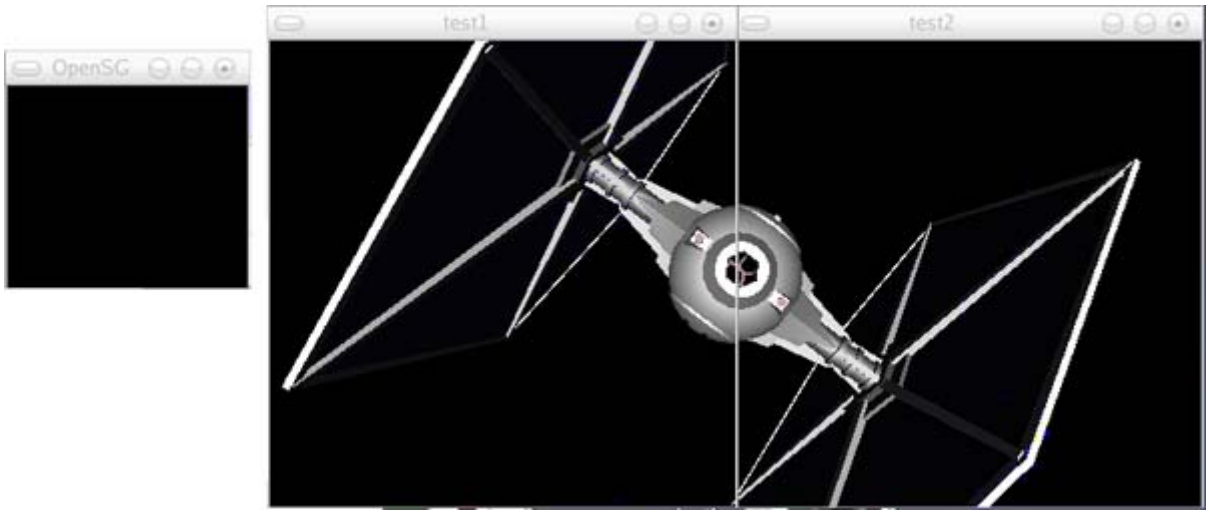
4. Ứng dụng thư viện OpenSG trong xây dựng sản phẩm mô phỏng

Hiện nay có nhiều thư viện đồ họa 3D đã tích hợp các thuật toán xử lý song song cảnh đồ họa 3D thời gian thực cho phép xây dựng các sản phẩm mô phỏng với dữ liệu lớn. Nổi bật hơn cả đó là thư viện OpenSG www.opensg.org. OpenSG là một hệ thống thư viện mã nguồn mở (LGPL) cho phép sử dụng tự do. Thư viện này chạy được trên nền các hệ điều hành IRIX, Windows và Linux.

Yêu cầu sức mạnh tính toán trong một ứng dụng gần như là vô hạn - hoặc yêu cầu lập trình tính toán chuyên ngành hoặc yêu cầu chất lượng cao hiển thị dữ liệu. Để làm được điều đó cần phải có sức mạnh của một siêu máy tính hoặc một nhóm máy cùng xử lý theo cơ chế song song. Nhưng siêu máy tính đắt tiền hơn nhiều lần. Việc dùng nhiều máy tính theo cơ chế song song ngày càng được quan tâm trong xây dựng các sản phẩm mô phỏng.
Dưới đây là mô hình ứng dụng 3 máy theo cơ chế song song.



Hình 10 – Mô hình 3 máy tính theo có chế xử lý song song hiển thị 2 màn hình
Trong sơ đồ này cần ba máy: Máy trạm chạy ứng dụng thực hiện hai pha APP và CULL (cập nhật dữ liệu, điều khiển tương tác người máy, thực hiện tính toán mô phỏng chuyên ngành.). Hai máy chủ chỉ thực hiện pha DRAW nghĩa là chỉ hiển thị cảnh thông qua hai camera khác nhau mô phỏng hai mắt của con người . Với thư viện OpenSG ta có thể thực hiện xử lý song song trên 48 máy.



Hình 11. Một cảnh trả lại trong hai cửa sổ độc lập

Hình ảnh này được hiển thị trên 3 cửa sổ: một cửa sổ nhỏ chỉ cho phép tương tác mô phỏng (sự chuyển động chuột hay bàn phím). Hai cửa sổ còn lại hiển thị một cảnh như thể là được hiển thị trong một cửa sổ. Để làm được điều này cần hai chương trình khác nhau (Client & Server). Dưới đây là toàn bộ mã chương trình được rút gọn.

Chương trình chạy trên máy trạm

```
// all needed include files
#include <OpenSG/OSGGLUT.h>
#include <OpenSG/OSGConfig.h>
#include <OpenSG/OSGSimpleGeometry.h>
#include <OpenSG/OSGGLUTWindow.h>
#include <OpenSG/OSGSimpleSceneManager.h>

#include <OpenSG/OSGMultiDisplayWindow.h>
#include <OpenSG/OSGSceneFileHandler.h>

OSG_USING_NAMESPACE
using namespace std;

SimpleSceneManager *mgr;
NodePtr scene;

int setupGLUT( int *argc, char *argv[] );

int main(int argc, char **argv)
{
#if OSG_MINOR_VERSION > 2
   ChangeList::setReadWriteDefault();
#endif
   osgInit(argc,argv);

   int winid = setupGLUT(&argc, argv);
```

```
    //this time we'll have not a GLUTWindow here, but this one
    MultiDisplayWindowPtr multiWindow = MultiDisplayWindow::create();

    //set some necessary values
    beginEditCP(multiWindow);
       // we connect via multicast
       multiWindow->setConnectionType("Multicast");
       // we want two rendering servers...
       multiWindow->getServers().push_back("Server1");
       multiWindow->getServers().push_back("Server2");
    endEditCP(multiWindow);

    //any scene here
    scene = makeTorus(.5, 2, 16, 16);

    // and the ssm as always
    mgr = new SimpleSceneManager;

    mgr->setWindow(multiWindow);
    mgr->setRoot  (scene);
    mgr->showAll();

    multiWindow->init();

    glutMainLoop();

    return 0;
}

void display(void)
{
    //redrawing as usual
    mgr->redraw();

    //the changelist should be cleared - else things
    //could be copied multiple times
    OSG::Thread::getCurrentChangeList()->clearAll();

    // to ensure a black navigation window
    glClear(GL_COLOR_BUFFER_BIT);
    glutSwapBuffers();
}

void reshape(int w, int h)
{
    glutPostRedisplay();
}
```

```
void mouse(int button, int state, int x, int y)
{
   if (state)
      mgr->mouseButtonRelease(button, x, y);
   else
      mgr->mouseButtonPress(button, x, y);
   glutPostRedisplay();
}

void motion(int x, int y)
{
   mgr->mouseMove(x, y);
   glutPostRedisplay();
}

int setupGLUT(int *argc, char *argv[])
{
   glutInit(argc, argv);
   glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_DOUBLE);

   int winid = glutCreateWindow("OpenSG");

   glutReshapeFunc(reshape);
   glutDisplayFunc(display);
   glutMouseFunc(mouse);
   glutMotionFunc(motion);

   return winid;
}
```

Chương trình chạy trên máy chủ

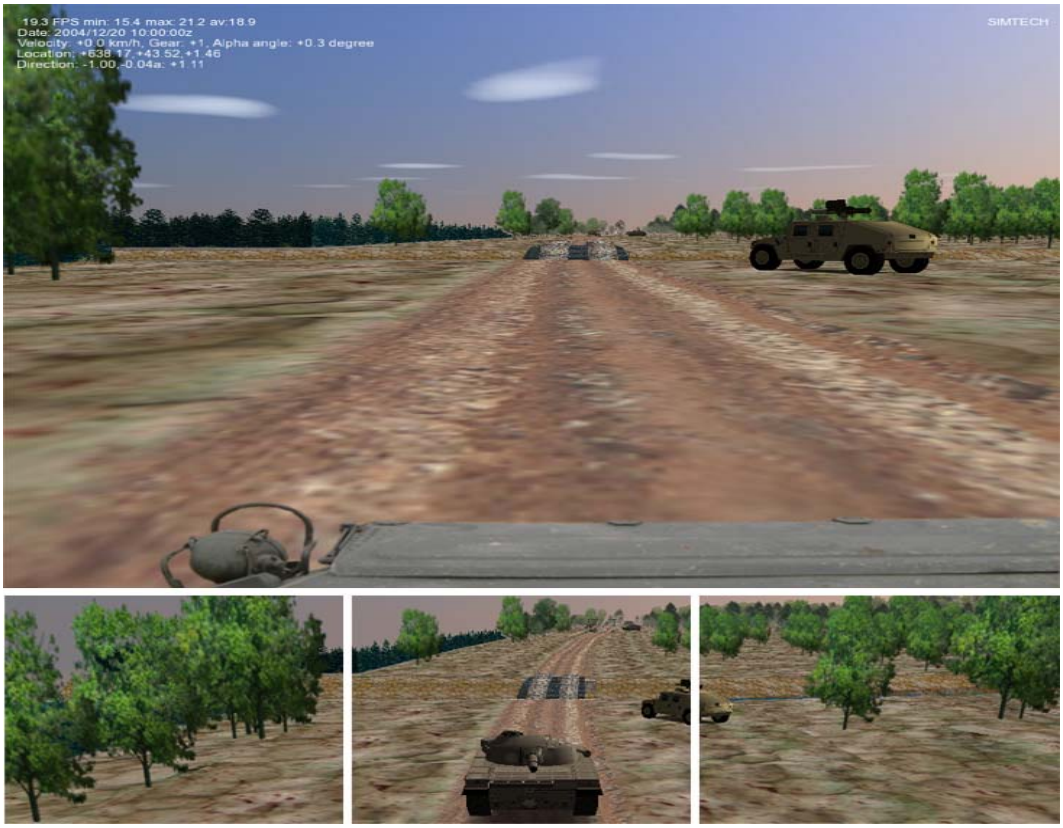
```
#include <OpenSG/OSGGLUT.h>
#include <OpenSG/OSGConfig.h>
#include <OpenSG/OSGClusterServer.h>
#include <OpenSG/OSGGLUTWindow.h>
#include <OpenSG/OSGRenderAction.h>

OSG_USING_NAMESPACE
using namespace std;

GLUTWindowPtr   window;
RenderAction    *ract;
ClusterServer  *server;

void display();
void reshape( int width, int height );

int main(int argc,char **argv)
```

```
{
   int            winid;

   // initialize Glut
   glutInit(&argc, argv);
   glutInitDisplayMode( GLUT_RGB |GLUT_DEPTH | GLUT_DOUBLE);

   if (!argv[1]){
         cout << "No name was given!" << endl;
         return -1;
   }

   // init OpenSG
#if OSG_MINOR_VERSION > 2
   ChangeList::setReadWriteDefault();
#endif
   osgInit(argc, argv);

   winid = glutCreateWindow(argv[1]);
   glutDisplayFunc(display);
   glutIdleFunc(display);
   glutReshapeFunc(reshape);
   glutSetCursor(GLUT_CURSOR_NONE);

   ract=RenderAction::create();

   window     = GLUTWindow::create();
   window->setId(winid);
   window->init();

   //create a new server that will be connected via multicast
   //argv[1] is the name of the server (at least it should be...)
   server     = new ClusterServer(window,argv[1],"Multicast","");
   server->start();

   glutMainLoop();

   return 0;
}

void display()
{
      //simply execute the render action
      server->render(ract);
      //and delete the change list
      OSG::Thread::getCurrentChangeList()->clearAll();
}
```

```
void reshape( int width, int height )
{
      window->resize( width, height );
}
```

5. Một số kết quả

Sử dụng mô hình trên, chúng tôi đã xây dựng thành công mô hình song song, đa màn hình hiển thị trên nhiều máy. Số khung hình đạt 60 fps bảo đảm việc tương tác qua lại của người dùng, đáp ứng được yêu cầu đạt ra của các thiết bị tập lái (Hiện được ứng dụng trong sản phẩm hệ thống tập lái xe BMP-1 do TTCN Mô phỏng thực hiện. Cảnh mô phỏng đồ họa 3D được hiển thị như ở hình 12)



Hình 12. Cảnh 3 chiều trong cabin tập lái xe BMP-1

6. Kết luận

Với cách tiếp cận trong bài báo chúng ta có thể sử dụng lợi thế của phần cứng hiện nay để xây dựng mô hình đa nhiệm, song song, đa luồng và đa màn hình hiển thị trong đồ họa. Mô hình nêu trên được ứng dụng tốt trong các sản phẩm mô phỏng mà cần có dữ liệu lớn và đòi hỏi tương tác thời gian thực.

• Tuy vậy, việc hoàn thiện mô hình nêu trên cần được đầu tư thêm nhiều thời gian và công sức, tập hợp được lực lượng đông đảo các cán bộ chyên môn cùng tham gia nhất là đội ngũ chuyên gia trong các chuyên ngành.

7. Tài liệu tham khảo

[1]. *Open Scenegraph*, (Robert Osfield) 2004. www.openscenegraph.org
[2]. *Open SG*, (Dirk Reiners) 2000. www.opensg.org
[3]. Silicon Graphics Inc. SGI OpenGL Optimizer Whitepaper. http://www.sgi.com/software/optimizer/whitepaper.pdf, 1998.
[4]. *Net Juggler Guide*, (Allard, J. et al) http://netjuggler.sourceforge.net 2001
[5]. *MPI: The Complete Reference*, (Snir, M. et al) MIT Press, 1996
[6]. osgVRPN: http://mew.cx/osg/
[7]. VRPN: http://www.cs.unc.edu/Research/vrpn/
[8]. OpenThread: http://openthread.sourceforge.net 2004.
[9]. Open Producer : http://Producer.sourceforge.net 2004.