Chess Gaming and Graphics using Open-Source Tools

Tong Lai Yu
California State University
Department of Computer Science and Engineering
San Bernardino,
CA 92407

Abstract

This paper explores the use of open-source resources to create a high-quality chess game with 2-D and 3-D graphical features. In our project, the popular open-source computer chess engine, Beowulf, is used as our chess engine. Beowulf is simple and straightforward to use but it is single-threaded; we make use of the platform-independent SDL (Simple DirectMedia Layer) libraries to make the engine multi-threaded so that it becomes more robust, and interactive and can handle multimedia features. Some recent developments in graphics including OpenGL Shading Language (GLSL), shadow casting, and texture mapping are used to enhance the graphical features of the game. In some cases, Blender, a 3D content creation suite is used to create some graphical objects in 3D Max format that saves an object as a composite of many meshes along with relevant information like lighting, camera position and UV mapping. A 3D Max object can then be easily parsed and processed by a C/C++ program by making use of the 3dslib library. Graphical objects are also created by the technique of Surface of Revolution, which makes the program more flexible and faster to run; display list is employed to improve the execution speed of the program.

1. Introduction

In recent years, video games have gained popularity in a wide variety of applications, including entertainment, education, technology skill training, and prevention or cure of chronic diseases like Alzheimer's disease or memory impairment. Video chess game is a typical video game example that can be used in education and mental training. Chess itself is a good exercise for mind that helps develop mental abilities like critical thinking, abstract reasoning, problem solving, pattern recognition, strategic planning, and creative thinking. Indeed, chess is part of the curriculum in Russia and some Eastern European countries. Over the past few decades, there were quite a lot of research of chess gaming[1,2]. However, the research mainly focused on the development of effective algorithms that computer programs can effectively execute in real time to play against humans. Some research also explored the use of distributed computing to build an effective chess engine. In this paper, our concern is on how one can put together some existing platform-independent open-source resources to create a high-quality interactive video chess game. The open-source programs or libraries that we have used in this project include the chess game engine Beowulf[3,4], 3-D content creation suite Blender[5], 3DS library 3dslib[6], Simple DirectMedia Layer (SDL) library[7], Mesa OpenGL and OpneGL Shading Language (GLSL) libraries[8]. The resulted chess game is platform-independent and can be compiled and run in popular computing systems such as MS Windows, MAC OS X, Linux, and FreeBSD etc; it can be easily adopted to run in many embedded hand-held devices like iphones and other mobile phones.

2. Multithreaded and Multimedia Chess Game Engine

In our project, the chess game engine is based on Beowulf which was primarily developed by Colin Frayn at the University of Birmingham, UK and Dann Corbit of USA[3]. To find a good move, Beowulf searches a game tree using the Negamax Search algorithm, which is a slightly variant formulation of minimax search[9,10] that relies on the zero-sum property of a two-player game. The game tree can be enormously simplified by Alpha-Beta pruning[11]. Its strength is estimated to be around 2300 ELO (International Master Standard) on a fast computer.

Beowulf is written in standard C. However, it is text-based and single-threaded. We employed functions of the Simple DirectMedia Layer (SDL) library[7] to add 2D graphics, multi-threaded features and sound effects to the game engine. The SDL library is also written in C

and can be easily integrated with OpenGL; it is a cross-platform multimedia library designed to provide low level access to audio, keyboard, mouse, joystick, 3D hardware via OpenGL, and 2D video framebuffer; it has been used by MPEG playback software, emulators, and many popular games. By making it multi-threaded, the chess engine becomes more robust and a lot more interactive, allowing music to be played while a game is in progress. Sound or music features can be incorporated in the game using the SDL libraries or the Open Audio Library (OpenAL)[12,13]. The C/C++ standard template library (STL) containers such as vectors and queues can be conveniently used to record the states of the game so that moves can be undone. Images may be created using the open-source GNU Image Manipulation Program (GIMP)[14].

Figure 1 shows a sample screen display of the 2D chess game built using STL, Beowulf and SDL graphical interfaces. (At this point, OpenGL is not needed as the game is only two-dimensional and SDL provides very efficient graphics rendering mechanisms.) The program can be compiled and run in any OS that SDL supports, including Linux, Windows, Windows CE, BeOS, MacOS, Mac OS X, FreeBSD, NetBSD, OpenBSD, BSD/OS, Solaris, IRIX, and QNX. Besides testing the chess program in RedHat Linux, we have also tested it in an Embedded Linux system running with ARM-9 processor at about 120 MHz[15]. Because of the direct-access graphical support of SDL, one does NOT need X-Window in the embedded system to run the chess program. The embedded version works similar to the regular version, except that all the graphical images are scaled down by half because of the small screen display of embedded systems.

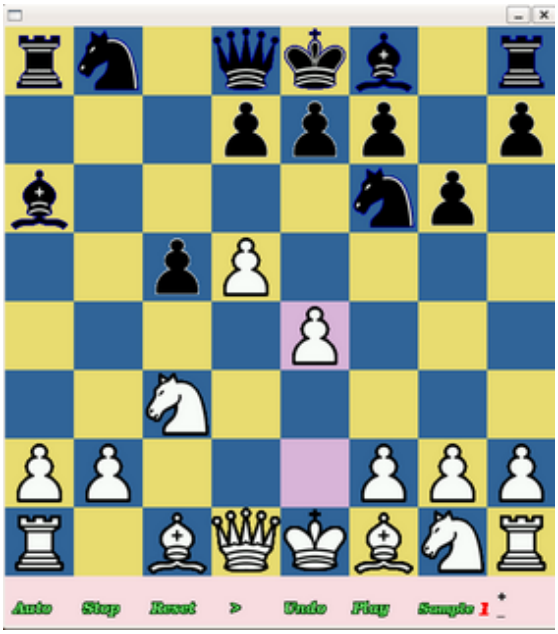

Figure 1. Sample 2D Chess Game.

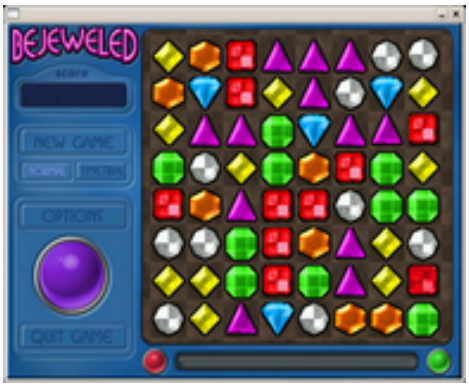

Figure 2. Embedded Game of Bejeweled.

The open-source graphics tools discussed can be also applied to the development of many simple 2D games. As an example, Figure 2 shows another popular multimedia game, Bejeweled, that we have developed for an ARM-based embedded system[16,17] in a very short time using these open-source tools.

3. Three Dimensional Graphics

Very often, a high-quality chess program may incorporate more fancy 3D graphics in the game. For instance, a player may want to rotate the game board, view it at different angles, zoom it, translate it, or deform certain chess pieces during the game to create special entertaining effects. To achieve these, one must go beyond the SDL graphics and use more sophisticated graphical tools and programming techniques. In the project, we used Blender as our graphics modeler to create 3D graphics models. Blender[5,18] is a free open source 3D content creation suite, available for all major operating systems under the GNU General Public License. Using Blender, one can easily create 3D graphics models consisting of objects like a chess board or chess pieces with textures and lighting effects; a graphics model can be saved as a mesh of polygons along with relevant information like UV mapping, material properties, light sources and camera positions which can then be parsed and processed by a program written in a high-level language like C/C++ or Java. We have used the 3DS format[6], the native format of Autodesk's 3D Studio MAX to process 3D models. The 3DS format has become an industry standard for transferring models between 3D programs, or for storing models for 3D resource catalogs. It can be easily converted to **collada**[6], a new interchange file format for interactive 3D applications based on open standard XML schema. Instead of writing programs to parse 3DS files, we used another popular open-source library, 3dslib[6] to manage 3DS files. The following piece of C/C++ code shows how to ren-

der a 3DS model using the 3dslib library and OpenGL.

```
-------------------------------------------------------
struct Vertex {
  float x, y, z;
};

void display_meshes (  Lib3dsFile *f )
{
  struct Vertex *vs, v;
  Lib3dsMesh *mesh;

  Lib3dsMaterial *mat=0;    //mesh materials
  init_texture();           //initialize texture features

  glEnable( GL_CULL_FACE );
  glCullFace ( GL_BACK );
  Lib3dsNode *node;     //3DS node

  for ( int k = 0; k < f->nmeshes; ++k ) {
    mesh = f->meshes[k];
    vs = getVertices(f->meshes[k]);//get a vertice of mesh

    //calculate normal
    float  (*normals)[3] = (float (*)[3])
      malloc (3*3*sizeof(float) * mesh->nfaces);
    lib3ds_mesh_calculate_face_normals ( mesh, normals );
    for ( int i = 0; i < mesh->nfaces; ++i ) {
      if (mat) {            //set material properties
        float s;
        glMaterialfv(GL_FRONT, GL_AMBIENT, mat->ambient );
        glMaterialfv(GL_FRONT, GL_DIFFUSE, mat->diffuse);
        glMaterialfv(GL_FRONT, GL_SPECULAR, mat->specular);
        s = pow(2, 10.0*mat->shininess);
        if (s>128.0)
         s=128.0;
        glMaterialf(GL_FRONT, GL_SHININESS, s);
     }  //if ( mat )
     //display the mesh
     glBegin ( GL_TRIANGLES );
     for ( int j = 0; j < 3; ++ j ) {
       glNormal3fv(normals[3*i+j]);
       int ii = mesh->faces[i].index[j];
       v = vs[ii];
       if ( mesh->texcos )    //texture coordinates
         glTexCoord2f(mesh->texcos[ii][0],
                      mesh->texcos[ii][1]);
       glVertex3fv( ( float*)&v );
     }
     glEnd();
   } //for i
   free ( normals );
  }  //for k
  free ( vs );
}

void display()
{
  Lib3dsCamera *camera;
  Lib3dsFile *f = get3dsFile ( (char *) filename );

  if ( f->ncameras > 0 ) {
    camera = f->cameras[0];
    glMatrixMode ( GL_PROJECTION );
    glLoadIdentity();
    gluPerspective ( camera->fov, 1.0,
        camera->near_range, camera->far_range );
    glMatrixMode ( GL_MODELVIEW );
    glLoadIdentity();
    gluLookAt(camera->position[0], camera->position[1],
      camera->position[2], camera->target[0],
      camera->target[1], camera->target[2], 0, 1, 0 );
  }
  display_meshes (  f );
  glFlush();
  glutSwapBuffers();
}
-------------------------------------------------------
```

One can see from the code that the rendering process is quite straightforward. It uses the two OpenGL utility commands, **gluPerspective()** and **gluLookAt()** and the camera information saved in the 3DS file to specify the view of the graphics model. It also makes use of the information of textures, material properties, normals and triangular meshes saved in the file to render the 3D model. The only crucial feature that is not shown is the use of lighting information contained in the 3DS file. Once a 3DS file is parsed, we can apply any affine transformation to the image objects; we can rotate, translate or zoom the chess board or any chess piece. The texture of an object can be easily changed with customized texture images. Lighting and cameras can be adjusted according to needs. Moreover, shadows and special shading effects can be added to the model using OpenGL Shading Language (GLSL)[19], which basically has the same syntax as C/C++ and has become a very popular tool in video game development.

This is a very good method of creating 3D interactive graphics for a chess program if the program runs in a powerful computer. However, a realistic 3DS model created by Blender usually consists of a numerous number of polygon meshes and it is computing-intensive to process a large number of meshes. Such a model may not be appropriate to be run in slower devices like embedded systems. For slower systems, a better method to generate 3D chess pieces is to create them using the technique of "Surface of Revolution"[20], in which a 3D object is produced by first drawing a profile and then rotating the profile about an axis in discrete steps. In our project, we used B-spline curves to generate profiles that can efficiently generate 3D chess piece images and by revolving the profiles around the z-coordinate axis.

A B-spline curve with n control points and n blending functions $N_{k,m}(t)$ of order m is described by the following equation:

$$P(u) = \sum_{k=0}^{n-1} P_k N_{k,m}(u)$$

where $P_0, ..., P_{n-1}$ are the control points and the blending functions can be calculated recursively:

$$N_{k,m}(u) = \left(\frac{u - u_k}{u_{k+m-1} - u_k}\right) N_{k,m-1}(u) + \left(\frac{u_{k+m} - u}{u_{k+m} - u_{k+1}}\right) N_{k+1,m-1}(u)$$

$$N_{k,1}(u) = \begin{cases} 1 & \text{if } u_k < u \leq u_{k+1} \\ 0 & \text{otherwise} \end{cases}$$

for k = 0, 1, ..., n - 1. In the above formulas, $(u_0, u_1, ..., u_{n+m-1})$ is a knot vector, which can be calculated according to the following rules[20]:

1. There are totally n + m knots, denoted as

$u_0, ..., u_{n+m-1}$

2. First m knots, $u_0, ..., u_{m-1}$ all have value 0
3. Knots $u_m, ..., u_{n-1}$ increases in increments of value 1, from 1 to n - m.
4. The final m knots, $u_n, ..., u_{n+m-1}$ are all equal to n - m + 1.

Figure 3 shows a pawn that was generated using this method; the pink squared dots are the control points used to produce the profile and the pawn object is obtained by revolving the profile around the z-axis. By adjusting the control points, one can easily adjust the shape of the object. Also, display-lists[20] can be used to speed up the rendering of these kind of objects. Generating image objects using surface revolution also makes the program a lot more flexible when it is run in a wide range of platforms. More slices and steps can be used when it is run in a fast system to give higher quality images. In embedded systems where the processor speed is much slower, the number of slices and steps used can be significantly reduced.

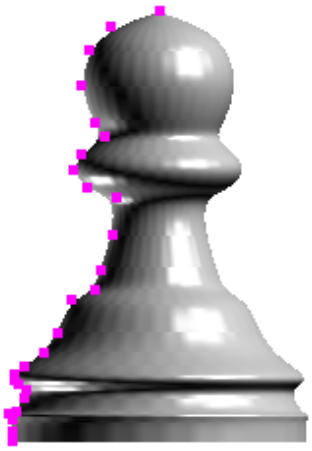

Figure 3. Pawn generated by Surface Revolution.

4. Conclusions

In conclusion, one can utilize open-source resources to develop a high-quality multimedia 3D graphics game like chess which can run in different platforms including embedded systems. The use of open-source tools significantly shorten the development time and reduce the development cost to a minimum. The learning of the tools is a valuable education experience for a developer; the skill can be reused again to develop different types of video games or image processing applications[21]. Therefore, these tools are also very valuable resources for education institutions like some campuses of California State University that have curricula teaching video games, graphics and image processing. Relying on open source tools enables students to affordably undertake development of video game software and other related multimedia projects for profit, which can fuel their interest and hope even if the profit is insignificantly small.

Acknowledgments

This research is supported by the CSUSB 2008 Summer POD funding. The author thanks Dr. David Turner for introducing the 3DS and collada 3D file formats.

References

[1] Colin Frayn and Carlos Justiniano, " The ChessBrain Project Massively Distributed Chess Tree Search" in *Advanced Intelligent Paradigms in Computer Games*, Springer Berlin / Heidelberg, p.92 - 115, 2007.

[2] Colin M. Frayn, Carlos Justiniano, Kevin Lew, "ChessBrain II A Hierarchical Infrastructure for Distributed Inhomogeneous Speed-Critical Computation", *IEEE Symposium on Computational Intelligence and Games* , Reno NV, May 2006.

[3] http://www.frayn.net/beowulf

[4] http://www.frayn.net/beowulf/theory.html

[5] http://www.blender.org

[6] http://www.lib3ds.org, http://www.collada.org

[7] http://www.libsdl.org

[8] http://www.mesa3d.org

[9] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley, 1984.

[10] N. J. Nilsson, *Artificial Intelligence: A New Synthesis*, Morgan Kaufmann Publishers, Inc. 1998.

[11] Knuth, D. E., and Moore, R. W., "An Analysis of Alpha-Beta Pruning". *Artificial Intelligence* Vol. 6, No. 4: 293 - 326, 1975.

[12] http://www.openal.org

[13] John R. Hall, *Programming Linux Games*, Linux Journal Press, 2001.

[14] http://www.gimp.org

[15] A.N. Sloss, D. Symes, and C. Wright, *ARM System Developer's Guide*, Elsevier Inc., 2004.

[16] T.L. Yu, "Implementing a 16-bit Embedded Ogg Vorbis Decoder", *Proceedings of ICCSA 2006*, San Diego, June, 2006.

[17] T.L. Yu, "Implementation of Video Player for Embedded Systems", *Proceedings of IASTED on Signal and Image Processing*, p. 25-30, Honolulu, Hawaii, August 20-22, 2007.

[18] http://the-labs.com/Blender/3DS-details.html

[19] R.J. Rost, *OpenGL Shading Language*, Second Edition, Addison Wesley, 2006.

[20] F.S. Hill, and S.M. Kelly, *Computer Graphics Using OpenGL*, 3rd Edition, Prentice Hall, 2007.

[21] T.L. Yu, "A Framework for Very High Performance Compression of Table Tennis Video Clips", *Proceedings of IASTED on Signal and Image Processing*, P.167-172, Kailua-Kona, Hawaii, August 18-20, 2008.