Implementation of 3D Snooker Simulator

Ying Feng

A Major Report

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Computer Science at

Concordia University

Montreal, Quebec, Canada

April 2004

Your file Votre référence
ISBN: 0-612-91029-6
Our file Notre référence
ISBN: 0-612-91029-6

Abstract

Implementation of 3D Snooker Simulator

Ying Feng Master of Computer Science
Concordia University, 2004

The implementation of 3D Snooker Simulator project is one part of the 3D Snooker Simulator project. It mainly focuses on porting the graphic and user interface of the existing 2D Snooker Simulator from 2D to 3D by using OpenGL and ANSI C++.

In the report, a brief introduction is given to the current situation of game programming, game programming using OpenGL and the features of the 2D Snooker Simulator. In the last decade, 3D graphic and user interface has become the main stream of computer-based game development and OpenGL is the main graphics-programming library, software interface to graphic hardware, and a "standard" for 3D graphics.

The target of the project is to implement the Snooker Simulator as a platform-independent 3D game with user-friendly, game-like User interface while preserving all the functionalities of the 2D version. The author describes how to reach these goals from designing, 3D object modeling and implementation. Meanwhile, lots of implementation problems and solutions are discussed in the report.

Finally, the author points out that the current 3D Snooker Simulator has achieved most of the design goals, but still has space for improvement.

Table of Contents

1. INTRODUCTION 1

2. BACKGROUND 4

2.1 GAME PROGRAMMING 4
2.2 OPENGL GAME PROGRAMMING 5
2.3 2D SNOOKER SIMULATOR 7

3. DESIGN 9

3.1 PROJECT GOAL 9
3.2 3D MODEL 9
3.2.1 Scene 9
3.2.2 The Snooker Table 13
3.2.3 Views 14
3.2.4 Lighting 15
3.3 ANIMATION 16
3.3.1 Ball's Movement 16
3.3.2 Camera's Movement 16
3.4 CLASSES AND FILES 16

4. IMPLEMENTATION 19

4.1 THE SNOOKER TABLE 19
4.1.1 Polygon Tessellation 19
4.1.2 Nurbs Surface 21
4.2 THE BALL'S MOVEMENT 23
4.3 THE CAMERA'S MOVEMENT 24
4.3.1 In Main Window 24
4.3.2 In Fine Window 36
4.4 IMPLEMENTATION PROBLEMS AND SOLUTIONS 39
4.4.1 Draw a circle in Fine Adjust Window 39
4.4.2 Issues with Fine Adjust Window 40
4.4.3 Continuous moving by setting Timer 41
4.4.4 Menu Selection 42
4.4.5 Blending 44
4.4.6 Change the cursor when mouse entering into certain area 45
4.4.7 Simulating Ball-like cursor on Snooker Table 46
4.4.8 Position balls on Snooker Table 46
4.4.9 Multiple windows management 47
4.4.10 Tool Tips 48
4.4.11 Glui Window 48
4.4.12 Display List 50
4.4.13 Calculate Time-increment for best performance 50
4.4.14 Fix out-of-virtual-memory defect 51

5. CONCLUSION 53

6. REFERENCES..55

APPENDIX A. MAIN SOURCE CODE FOR 3D IMPLEMENTATION............56

1. DISPLAY PARAMETER SET WINDOW USING GLUI.. 56
2. DRAW BALL-SHAPE CURSOR 60
3. DISPLAY FINE ADJUST WINDOW.. 61
4. DISPLAY MAIN WINDOW... 65

APPENDIX B. USER MANUAL...68

INTRODUCTION 68
Introduction to the 3D Snooker Simulator... 68
Introduction to Snooker Game.. 69
INSTALLATION 73
System Requirement... 73
Install/Uninstall:.. 73
PLAY GAME.. 74
Start the Snooker Simulator... 74
Introduction to Scenes: 74
Playing a Shot 79
QUESTION AND ANSWERS 81
1. What is DEMO 81
2. What is Time Testing.. 81
3. How to Set Players' mode.. 81
4. How to set the Environment Parameters.. 82
5. What's the difference between Detail report and Simple report?.......................... 83
6. What is the Tracking Mode... 84
7. What is the Advisor Mode 84
8. Can I leave the game temporarily without closing it? 84
9. How to close the game... 85
10. How to adjust in Adjustment Window... 85
11. What is Adjustment Limit.. 86
12. What is Automatic Players... 86
13. How to edit the Data Files 87
14. How about Environment Parameters 89

List of Figures

Figure 3.1 Game Scene........10
Figure 3.2 Scoreboard options........11
Figure 3.3 The Snooker Table........13
Figure 3.4 Outline of the snooker table........13
Figure 3.5 Snooker table of Top View........14
Figure 3.6 The snooker table of Side View........15
Figure 3.7 The fine window........15
Figure 4.1 Camera and balls positions........36
Figure 4.2 Calculate moving distance........37
Figure 4.3 Calculate Circle rotation........39
Figure 4.4 Main Menu Page........43
Figure 4.5 Play-time Taskbar........44
Figure 4.6 Taskbar with blending........45
Figure 4.7 Multi-window structure........48
Figure 4.8 Player Mode Setting........49
Figure 4.9 Parameter Setting........50
Figure A.1 Main menu page........74
Figure A.2 Game page........76
Figure A.3 Taskbar........76
Figure A.4 Snooker table at Top View........77
Figure A.5 Snooker table at Side View........78
Figure A.6 Player mode set window........81
Figure A.8 Environment parameters set window........82
Figure A.9 Detail Report........83
Figure A.10 Simple Report........83
Figure A.11 Tracking mode........84
Figure A.12 Fine-Adjust window........85

1. Introduction

The 3D Snooker Simulator (abbreviated as the 3D version) is the successor version of the 2D Snooker Simulator (abbreviated as the 2D version) that was developed by Professor Peter Grogono. The 3D version reserves all the functionalities of the 2D version, including four options of playing mode – manual, track, advise and auto, time testing, demo, parameter setting for game environment and players.

The 2D version has a 2D user interface which is based on Fastgraph library©™ and win32 GDI. It's platform-dependent software. One main task of this project is to implement a user-friendly, game-like 3D user interface. To make it possible and easy to be ported to other platforms in the future, the whole project is implemented only with OpenGL and ANSI C++.

The 2D version has already embedded a good mechanism to calculate the ball's movements according to the environment properties, such as ball-ball collision, ball-table collision and hitting position, speed and direction. It also supplies an optimized algorithm to select the best action for an auto player. In the 3D version, all of these logic components are reserved.

In the 2D version, the UI components are not separated very clearly from logic components. How to integrate the new 3D user interface with the existing tested complicated logic part is the most difficult part of the project. By using software reverse-engineering approach, we isolated the classes and functions only or mainly for user interface from those classes/functions for game logic and defined the interfaces between the UI components and logic components. With this approach, we can focus our attention only on UI components without affecting the logic parts. This method saved us lots of time on UI development and system testing.

To support UI implementation with OpenGL, a set of classes based on an extendible hierarchical structure is designed and implemented. With the help of these classes, it's very easy to create autohidable/ non-autohidable windows with customized position, size, background, contained controls, and controls of different size, position, shape, text, image, tool tip, border-style, font, etc. (Some functionalities are not implemented completely yet).

The followings are the new features implemented in the 3D version:

- 3D snooker table with wooden cover
- Light reflection on balls
- Multiple views – Top view, Side view and Fine view
- Continuous moving – zooming, turning, moving
- Foldable Scoreboard
- Auto-hidable subwindow
- Tool tip for immediate help
- Exact cue speed shown as tooltip
- Full screen game
- Change cursor to show the safe hitting area of the cue ball in the Adjust Window

- Separate main menu page to make UI clearer
- Animated 3D cue
- External resources, supplies a possibility to use change-skin technology to make the UI more funny (not implemented yet)

The following old features are reserved and/or re-implemented in the 3D version.

- Four play modes – manual, trace, advise and auto
- AI for auto/advise play mode
- Player parameter setting
- Environment parameter setting
- Time testing for best setting
- Demo
- Sound effect control
- Cue speed control
- Cue ball hitting position control
- 3D view for adjusting colliding pot between cue ball and target ball more precisely
- Ball-like cursor in Snooker table for positioning a ball precisely
- Keyboard support

2. Background

2.1 Game Programming

In the last decade, computer games have grown into a multi-billion-dollar market. As a computer-based game, it mainly consists of following elements. [OpenGL Game]

- Graphics
- Input
- Music and sound
- Game logic and artificial intelligence
- Networking
- User interface and menu system
- A good game interface makes playing the game as easy as possible

As to an individual game, some aspects may be not present, such as networking for a non-network game, but these elements have already covered all the essential aspects of a game.

Firstly, a game must be amusing and interesting. A beautiful user interface and exciting 2D/3D graphics are vital to a game, they must be able to seize the players' attentions at the first watch.

Then, the game must be powerful but easy to use, nobody is willing to spend several hours just for learning how to play the game. So, a clear, friendly user interface with convenient input approach is very important to give players a relaxed playing environment.

Anyway, to achieve a fascinating game, high artificial intelligent is the key factor. Beautiful UI, exciting music and graphics can only pull the players to the play station, the game itself is the key to keep the players staying there and playing. From all the aspects we discussed above, we know PC Game is a complicated engineering system. It involves many fields of computer science, hardware, software, multi-media, computer graphic, etc.

2.2 OpenGL Game Programming

As the developing of the PC Game industry, the players are not satisfied with 2D images again. They want the game images to be funnier, as close to real life as possible, even more interesting and exciting.

3D user interface is becoming the main stream of PC game development. Many powerful 3D development environments are available for developers to create 3D graphics without any low-level programming knowledge, such as SoftImage, 3D Studio, Maya, Photo Styler, INFERNO. All these applications are based on low-level 3D API, such as OpenGL, DirectX, Heidi.

OpenGL is not only a graphics-programming library, but also a software interface to graphic hardware, and a "standard" for 3D graphics. More and more hardware manufacturers and operating system developers promise to support OpenGL, and include OpenGL library as one part of their products, such Microsoft's window system. OpenGL has become the main 3D graphic standard, main 3D graphic developing tool in last ten years.

OpenGL is also an excellent tool and powerful library to develop PC games for the following advantages:

- Device independence. OpenGL is an interface between hardware and software. With OpenGL's support, developers needn't to care about the technical details of display devices, but put all their attentions on the 3D graphic itself.
- Simple, easy to learn. OpenGL is just a set of statements, all the complicated algorithms are encapsulated very well. This feature can shorten the learning curve and accelerate the development speed.
- Powerful, flexible feature set. OpenGL is powerful and flexible. Developers can implement any kind of feature they want by using OpenGL.
- Cross-platform portability. OpenGL is platform-independent, it can be ported to any system that supports OpenGL. If the game is developed with pure OpenGL, it can be compiled and run on many systems, so as to save development cost (money and time) and extend the potential market of the game.
- Established: More and more third-party libraries are available in the market. All these tools are tested very well. This makes OpenGL more powerful, stable and easier.
- Performance. OpenGL are implemented at assembly level and all the algorithms and codes are optimized. With hardware's support, OpenGL

can gain very good performance enough to develop any complicated 3D graphics.

- Interactive: OpenGL is good at developing interactive applications. Good interactivity is the vital feature of Games.

2.3 2D Snooker Simulator

The 2D Snooker Simulator was developed based on a snooker simulation that runs on PC under DOS, which was also developed by Professor Grogono. 2D Snooker Simulator did a good job on simulating the motions of the balls much accurately.

As an interesting feature of 2D Snooker Simulator distinctive from other snooker games, environment properties are considered for calculating the movements of balls. The player can control these properties by modifying the parameter settings. So, it's not only a game, but also a real simulator of snooker game.

In the simulator, the effects of ball/ball and ball/cushion collisions are simulated by solving differential equations numerically, rather then by relying on analytical solutions of simplified equations. The behavior of balls in the simulation is very similar to the behavior of balls on a real table. There are some limitations, however the cue is always horizontal, so massé shots are not possible, and balls remain on the table. [The Snooker Simulation]

Meanwhile, 2D Snooker Simulator is very intelligent. It supports four playing modes – manual, track, advise and auto. The player can play snooker by himself with/without tracking, or watch the computer simulating the game. He can also play against the simulated player. It is worth mentioning that the game supports personalized auto-play/advise mode. Following an auto-play algorithm, the computer can calculate and select the best action according to current situation and the selected auto-player's ability is not always the ideal action. This mechanism increases the amusement of the game.

The graphics and user interface of 2D Snooker Simulator was developed using Fastgraph™ and window GDI. The applications tried to simulate 3D effect in 2D context, such as light reflection on the balls, pseudo 3D view for fine adjustment .The user interface is clear and concise with enough help information. But it's based on standard windows with standard drop-down menus. As a game, it's not funny and exciting enough.

3. Design

3.1 Project Goal

The 3D Snooker Simulator is based on the 2D version. We will keep all the useful functions in the new version. Meanwhile, we're trying to improve the game in following aspects:

- 3D graphic and User Interface with multiple views, allows players to switch between different views to get more details as real game
- Funny and interesting UI, like a real game
- More friendly UI, allows the player to get help more convenient and quickly
- Platform independent

3.2 3D Model

3.2.1 Scene

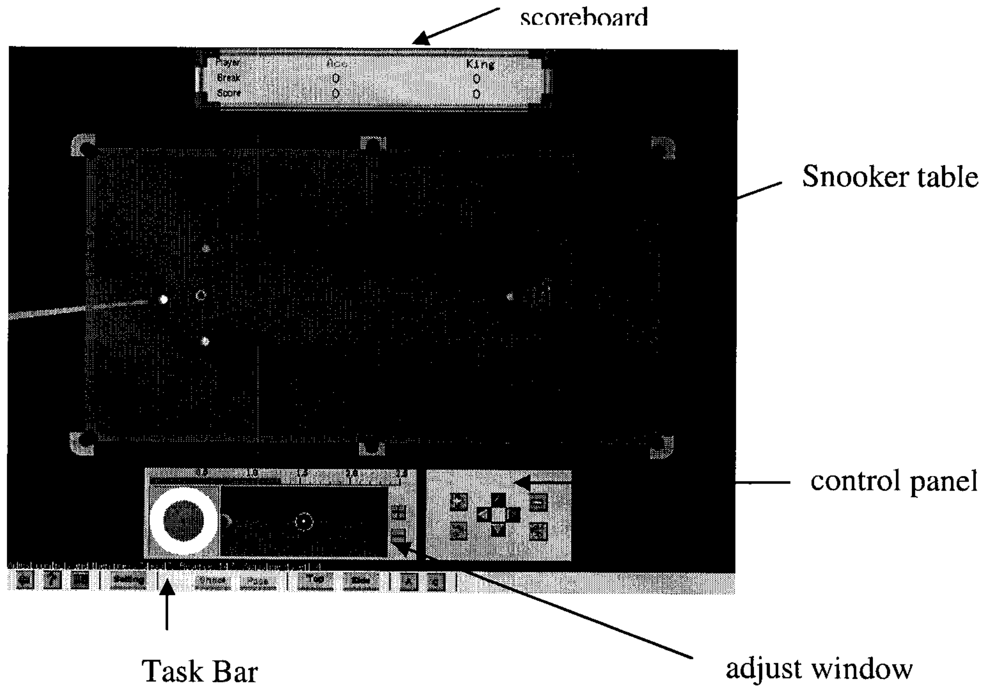
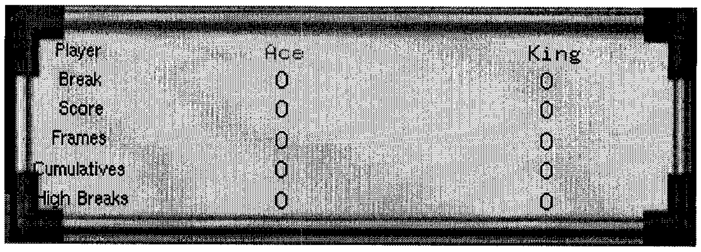


Figure 3.1 Game Scene

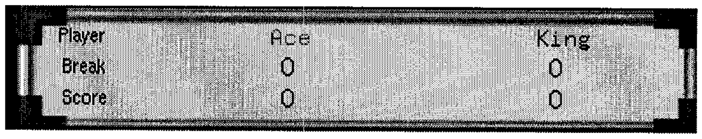
As shown in the above figure, the scene of the game consists of five parts:

1. *Scoreboard:* Scoreboard is used to display the players' information (player's name, who is the current striker, break, score, frames, cumulative and high breaks). To save display space, two types of Scoreboard are supplies, simple report and detail report. To toggle between these two options, the user can click "▤" on the Task Bar or double click on the Scoreboard (See figure 2)



Player	Ace	King
Break	0	0
Score	0	0
Frames	0	0
Cumulatives	0	0
High Breaks	0	0

Detail Report



Player	Ace	King
Break	0	0
Score	0	0

Simple Report

Figure 3.2 Scoreboard options

2. *Snooker table:* The Snooker table is the area where the players actually play the game. Two views are supplied to the players for observing the situation more convenient and clearer – Top View and Side view.

3. *Control Panel:* The Control Panel allows the players to change the viewpoint to the Snooker table. It contains 8 controls. Players can use these controls to adjust the camera's position and focal distance to get desired views. Some controls may not be accessible in a specific view to avoid ambiguities.

4. *Adjust Window:* The Adjust Window consists of three controls – speed set control, cue ball adjust control and fine adjust window. Speed set control allows players to set the cue speed by clicking on the calibrated scale, the accurate speed is shown when the mouse locates above the scale; Cue ball adjust control allows players to set the position where the cue will hit the cue

ball. Fine adjust window is a real 3D view, it allows players to adjust the hitting position between cue ball and target ball more precisely by clicking on the window. The distance between clicking position and the center of mapping circle of cue ball defines the adjustment degree. All the adjustments will take effect immediately.

5. *Task Bar:* The Task Bar provides players a clear and convenient environment to access all the functions available when a frame is in playing. (Time testing and demo may only be accessible when there is no frame in playing, so we exclude these two functions from Task Bar, and include them in the Main Menu Page. The playing frame will be terminated before switching to the Main Menu Page,)

3.2.2 The Snooker Table

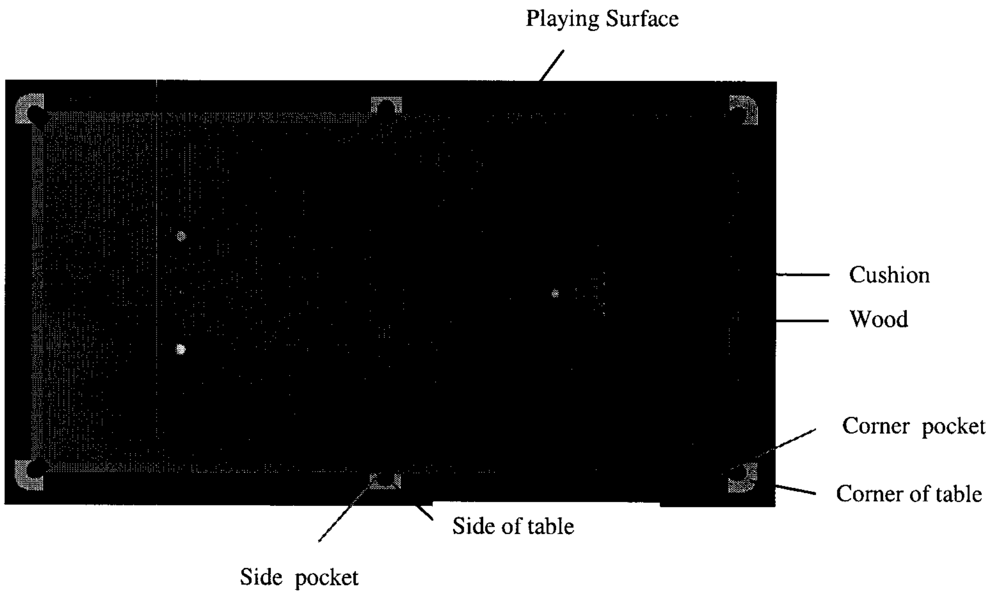


Figure 3.3 The Snooker Table

The playing surface is 350 cm by 175 cm. This is the distance between the cushions (rails) at each end (side) of the table.

Beyond the cushions, there is:

4 cm of green cloth.

8 cm of brown wood.

1 cm of metal trim.

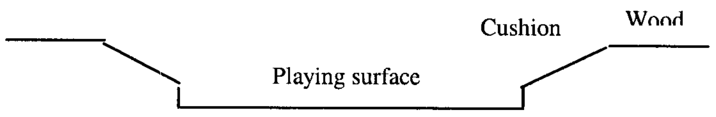


Figure 3.4 Outline of the snooker table

The top of the cushion is 3.75 cm higher than the table surface. The cushion is about 1 cm thick, so the gap between the table surface and the bottom of the cushion is 2.75cm.

3.2.3 Views

The 3D Snooker Simulator provides three different views that allow players to observe the balls from different viewpoint – Top View, Side View and Fine View. Snooker table has two views -- Top View and Side View. On Top View, the camera originally locates at (0.0, 0.0, 225.0), this view gives players a full view of the table. On Side View, the camera originally locates at (-100.0, 0.0, 30.0), this view allows players to observe every ball clearer from any horizontal viewpoint.

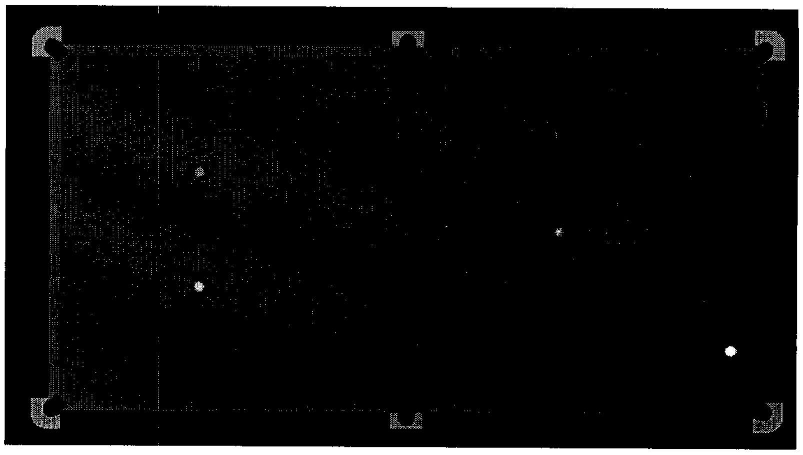


Figure 3.5 Snooker table of Top View

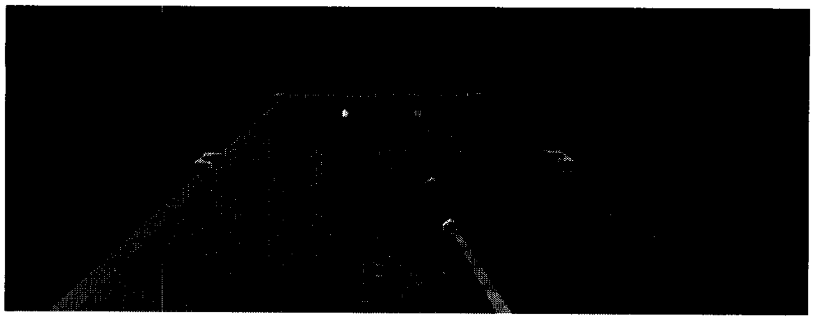


Figure 3.6 The snooker table of Side View

The Fine View is only available in the Fine Adjust Window. The camera locates right beside the cue ball, and keeps aiming at the target ball. This view allows players to get the position relationship between cue ball and target ball much clearer.

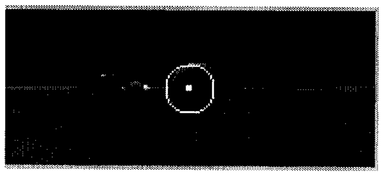


Figure 3.7 The fine window

3.2.4 Lighting

In main window, there is a white light positioning at (0.0, 0.0, 50.0), so the snooker table is bright and clear enough but not too blazing.

3.3 Animation

3.3.1 Ball's Movement

The movement of a ball is based on three factors – speed, direction and spin angle. The moving speed of a ball is calculated from the cue speed set by players. The moving direction is decided by the position of the cue ball and the selected target ball. The spin angle will also influence the ball's moving direction.

3.3.2 Camera's Movement

There are three cameras in the scene. We provide many different approaches to move each camera, so as to observe the snooker table and the balls' movements as players want.

On the "Top View", the camera can move upward, downward, rightward, leftward, and can zoom in and zoom out.

On the "Side View", the camera can move upward, downward, rightward, leftward, forward, backward, turn left and turn right.

On the fine window, the camera can only zoom in and zoom out.

3.4 Classes and Files

After read and comprehend the source code of the 2D Snooker Simulator, we isolated the UI classes/functions from those game logic classes/functions. Our porting job mainly focuses on these UI classes/functions. Meanwhile, to support implementing 3D UI, we define and implement a set of classes for creating and managing windows and controls easily.

The modified existing classes mainly include:

- Ball – display balls and their movements
- Control – Manage cue speed setting, cue ball hitting and fine adjustment
- Cue – display cue and its animation
- Path – display path of ball movement
- Table – display snooker table and different views
- Snooker.cpp which contains entry point of the application and a lot of global functions

The new created classes to support 3D UI implementation include:

- VirtualWindow – base class for all windows
- MainWindow – main window inherits from VirtualWindow
- GluiWindow – Glui window inherits from VirtualWindow
- VirtualControl – base class for all controls
- ControlCollection – manage a set of control
- ButtonCtl – button control inherits from VirtualControl
- RoundButtonCtl -- round button control inherits from ButtonCtl
- LabelCtl – label control inherits from VirtualControl
- AutoHidablePanel – Auto hidable window inherits from VirtualWindow and implements IAutoHidable interface
- AdjustWin – Container window inherits from VirtualWindow
- FineAdjustWin – adjust hitting position more finely, inherits from VirtualWindow
- PanelCueSpeed – To set cue speed, inherits from VirtualWindow

- PanelCueBall – To set hitting position on the cue ball for cue, inherits from VirtualWindow
- MainMenuWin – Supply main menus, inherits from VirtualWindow
- NominateWin – to select nominated color balls, inherits from VirtualWindow
- PanelControl – window to control camera, inherits from VirtualWindow
- PanelHelper – window to display help topics, inherits from VirtualWindow
- GluiHelperWin – help window for Glui windows, inherits from GluiWindow
- ParameterSetWin – Glui window for environment parameter setting, inherits from GluiWindow
- PlayerModeSetWin – Glui window for player mode setting, inherits from GluiWindow
- Scoreboard – foldable window to show scores, inherits from VirtualWindow
- SettingWin – window for parameter setting and sound control, inherits from VirtualWindow
- ToolTipWin – tooltip window for immediate help on controls, inherits from VirtualWindow
- TaskBar – window containing all button accessible when game is in playing, inherits from VirtualWindow

Note: For more details on the hierarchical relationship between new created windows, please see another report of this project on "3D User Interface Architecture"

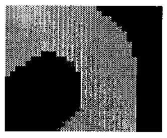
4. Implementation

4.1 The Snooker Table

The snooker table consists of many surfaces, most of them are rectangles that can be simply defined by its four vertexes, while others are more complicated, such as table corners. Here we mention two kinds of methods to draw these complicated surfaces: tessellation and Nurbs.

4.1.1 Polygon Tessellation

OpenGL can directly display only simple convex polygons. " A polygon is simple if the edges intersect only at vertices, there are no duplicate vertices, and exactly two edges meet at any vertex." [The Red Book]. But in snooker table, there are many concave polygons, like "the side corner of the table", it should be draw as this:


Side corner of Table

So we need to use tessellation, which can subdivide a concave polygon into simple convex polygons before they can be displayed, and OpenGL provides a collection of routines that perform tessellation. The code and explanation is as follows:

```
m_tess = gluNewTess();                    // Create a new tessellation object
void Table::side_table(void)
{
  MyPoint pt[] =                        // Define key points
  {
          {0.0,13.0, 0.0},
            {9.0, 13.0, 0.0},
            {9.0, 4.0, 0.0},
            {4.5, 4.0, 0.0},
            {4.0, 8.0, 0.0},
            {2.0, 10.0, 0.0},
            {0.0, 10.5, 0.0},
  };
  //------------------------------------------------------------------------------------------------------
  // Use gluTessCallback() several times to register callback funtions to perform operation
  //  during the tessellation.When the tessellation algorithm detects an intersection and must
  //  call the callback functions.
  //------------------------------------------------------------------------------------------------------
  gluTessCallback(m_tess, GLU_TESS_BEGIN, (void (__stdcall *)(void))&glBegin);
  gluTessCallback(m_tess, GLU_TESS_VERTEX,(void (__stdcall
  *)(void))&glVertex3dv);
gluTessCallback(m_tess, GLU_TESS_END, &glEnd);
        gluTessCallback(m_tess, GLU_TESS_ERROR, (void (__stdcall *)(void))
&errorCallback);
 //------------------------------------------------------------------------------------------------------
// gluTessBeginPolygon() and gluTessEndPolygon() begins and ends the specifiction of a polygon
to
//  be tessellated and associated a tessellation object.
// GluTessBeginContour() and GluTessEndContour() begin and end the specification of a closed
// contour which is a portion of a polygon. It consists of some calls to gluTessVertex().
//----------------------------------------------------------------------------------------------------------
gluTessBeginPolygon(m_tess,0);                    // top surface of corner of table
gluTessBeginContour(m_tess);
for(int i = 0; i< 7; i++)
```

```
{
        gluTessVertex(m_tess, pt[i], pt[i]);


        gluTessEndContour(m_tess);
        gluTessEndPolygon(m_tess);
}
```

4.1.2 Nurbs Surface

Nurbs surface is very important for modeling complicated surface, it's powerful and flexible for drawing any type of surface. Though it consumes more space and time for rendering, we still use it to draw the vertical walls of table corners.

To draw a Nurbs surface, the main procedures are:

1. Create a Nurbs Render

```
void Table::init_nurbs()
{
        m_theNurb = gluNewNurbsRenderer();
        gluNurbsProperty(m_theNurb, GLU_SAMPLING_TOLERANCE, 25.0);
        gluNurbsProperty(m_theNurb, GLU_DISPLAY_MODE, GLU_FILL);
        gluNurbsCallback(m_theNurb, GLU_ERROR,
                (void (__stdcall *)(void)) errorCallback);
}
```

2. Declare global array for control points of the Nurbs surface

```
GLfloat cpwpoints[6][4][3];        //contral points of corner pocket wall
```

3. Initialize the control point array for the Nurbs surface

```
void Table::init_corner_pocket_wall(void)

{

        GLfloat z_coord = 0.0;

        for(int v = 0; v < 4; v++)

        {

                cpwpoints[0][v][0] =-2.3;

                cpwpoints[0][v][1] =4.85;

                cpwpoints[1][v][0] = -4.0;

                cpwpoints[1][v][1] = 2.3;

                cpwpoints[2][v][0] = -6.0;

                cpwpoints[2][v][1] = 1.2;

                cpwpoints[3][v][0] = -9.0;

                cpwpoints[3][v][1] = 0.2;

                cpwpoints[4][v][0] = -12.0;

                cpwpoints[4][v][1] = 0.0;

                cpwpoints[5][v][0] = -14.0;

                cpwpoints[5][v][1] = 0.0;


                if( v == 0 || v == 1)

                        z_coord = 1.875 - 0.9375 * v;

                else

                        z_coord = 1.875 - 0.9375 * (v+1);

                cpwpoints[0][v][2] = z_coord;

                cpwpoints[1][v][2] = z_coord;

                cpwpoints[2][v][2] = z_coord;

                cpwpoints[3][v][2] = z_coord;

                cpwpoints[4][v][2] = z_coord;

                cpwpoints[5][v][2] = z_coord;
        }
    }
```

4. Draw the Nurbs surface

```
    void Table::corner_pocket_wall(void)
    {
        GLfloat tknots[12] = {0.0,0.0,0.0,0.0,0.0,0.0, 1.0,1.0, 1.0,1.0, 1.0,1.0};
        GLfloat sknots[8] = {0.0,0.0,0.0,0.0,1.0,1.0, 1.0,1.0};
        gluBeginSurface(m_theNurb);
        gluNurbsSurface(m_theNurb,
                        12, tknots, 8, sknots,
                        12, 3, &cpwpoints[0][0][0],
                        6, 4, GL_MAP2_VERTEX_3);
        gluEndSurface(m_theNurb);
    }
```

4.2 The Ball's Movement

The ball's movement is implemented by redrawing the balls at new position periodically according to the Time-increment and Display factor settings. The game logic classes, i.e. Path and Planner, are responsible for calculating the new position of each ball on the snooker table according to the cue speed, hitting position and spin angle, current position as well as environment properties. When new positions are set for all balls, a redisplay event is raised. The OpenGL will call the display function to redraw the balls at the earliest available time. The smooth moving of balls mainly depends on the environment settings – Time increment and Display factor, and machine's speed to a great extend.

```
void Ball::DrawBall()
{
        if(visible)
        {
                glPushMatrix();
                glTranslatef(xs,ys,zpos);
                glRotatef(zspin,0.0,0.0,1.0);
                glRotatef(yspin,0.0,1.0,0.0);
                glRotatef(xspin,1.0,0.0,0.0);
                glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE,
RGBColors[colour]);
                glMaterialfv(GL_FRONT, GL_SHININESS, polished);
                glMaterialfv(GL_FRONT, GL_SPECULAR, white);
                gluSphere(qobj, 2.7, 32, 16);
                glPopMatrix();
        }
}
```

4.3 The Camera's Movement

As a 3D object, the most important feature of the object is that it can be observed from any viewpoint. We provide many kinds of camera movements in main window and fine window to enhance this feature.

4.3.1 In Main Window

- **From the Top View**

The camera originally locates at (0.0, 0.0, 225.0), and the play can control the camera to go up, down, left, right, zoom in and zoom out.

➢ **Up**

Up When the Up button is clicked each time, the camera will move up a little bit. To achieve such effect, we modify the camera's Y-coordinate and aiming point's Y-coordinate. Each time, the camera's Y-coordinate is added by 2.0, and camera's aiming point's Y-coordinate is added by 2.0.

At the same time, we limit the up-bound of Y-coordinate, so that the snooker table can always be seen on the scene.

```
if(top_positionY < 87.5)
 {
        top_positionY +=2.0;
        top_aimY +=2.0;
        mainWindow->RefreshWin();
 }
```

➢ **Down**

Down When the Down button is clicked each time, the camera should move down a little bit. We modify the camera's Y-coordinate and aiming point's Y-coordinate. Each time the camera's Y-coordinate is subtracted by 2.0, and camera's aiming point's Y-coordinate is subtracted by 2.0.

At the same time, we limit the down-bound of Y-coordinate, so that the snooker table can always be seen on the scene.

```
if(top_positionY >=-87.5)
{
        top_positionY -=2.0;
        top_aimY -=2.0;
        mainWindow->RefreshWin();
}
```

➤ **Left**

◀ Left

When the Left button is clicked each time, the camera should move to the left a little bit, here we change the camera's X-coordinate and aiming point's X-coordinate. Each time, the camera's X-coordinate is subtracted by 2.0, and camera's aiming point's X-coordinate is subtracted by 2.0.

We also define a limitation to the left-bound of X-coordinate, so that the snooker table can always be seen on the scene.

```
if (top_positionX >=-175)
{
        top_positionX -=2.0;
        top_aimX -=2.0;
        mainWindow->RefreshWin();
}
```

- **Right**

Right

When the Right button is clicked each time, the camera should move to the right a little bit, here we change the camera's X-coordinate and aiming point's X-coordinate. Each time the camera's X-coordinate is added by 2.0, and camera's aiming point's X-coordinate is added by 2.0.

We also limit the right-bound of X-coordinate to ensure the snooker table to be visible on the scene at any time.

```
if(top_positionX <=175)
{
        top_positionX +=2.0;
        top_aimX +=2.0;
        mainWindow->RefreshWin();
}
```

- **Zoom In**

Zoom In

When the Zoom In button is clicked, the snooker table and balls should seem to be a bit larger. Here we change the alpha value of the viewing volume for the perspective projection, that is, we deduct the alpha value by 0.25 each time, and we limit the bound of the alpha value.

```
if(alpha>0.25)
{
        alpha -=0.25;
        mainWindow->RefreshWin();
}
```

➢ **Zoom Out**

Zoom Out

When the Zoom Out button is clicked, the snooker table and balls should seem to be a bit smaller. Here we change the alpha value of the viewing volume for the perspective projection, that is, we increase the alpha value by 0.25 each time. We also set a limitation to the bound of the alpha value.

```
if(alpha <180)
{
        alpha +=0.25;
        mainWindow->RefreshWin();
}
```

- **From the Side View**

The camera originally locates at (-100.0, 0.0, 30.0), and it can move upwards, downwards, leftwards, rightwards, forwards, and backwards and turn to left and right.

➢ **Up**

Up

When the Up button is clicked each time, the camera should move up a little bit. Here we change the camera's Z-coordinate and aiming point's Z-coordinate and X-coordinate. Each time the camera's Z-coordinate is added by 1.0, and camera's aiming point Z-coordinate is added by 0.2, X-coordinate is added by 0.1.

At the same time, we limit the up-bound of Z-coordinate.

```
if(z_distance<=70.0)
{
     z_distance +=1.0;
    x_aim +=0.1;
    z_aim +=0.2;
    mainWindow->RefreshWin();
}
```

➢ **Down**

Down When the Down button is clicked each time, the camera should move down a little bit. Here we change the camera's Z-coordinate and aiming point's Z-coordinate and X-coordinate. Each time the camera's Z-coordinate is subtracted by 1.0, and camera's aiming point Z-coordinate is subtracted by 0.2, X-coordinate is subtracted by 0.1.

At the same time, we limit the down-bound of Z-coordinate.

```
if(z_distance >=27)
{
     z_distance-=1.0;
     z_aim -=0.2;
     x_aim -=0.1;
     mainWindow->RefreshWin();
}
```

➢ **Left**

Left When the Left button is clicked each time, the camera should move to the left a little bit. But in the Side-view, it's more difficult to move it than in the Top view since the camera may move to any angle in this view. The aim point should also move correspondingly to keep the line between camera and aim position the same angle to the axis of the table. The calculation formula is implemented in GetMoves function. Here, we call GetMoves function to calculate the movX and movY. The camera's X-coordinate and aiming point's X-coordinate is increased by movX everytime, and the camera's X-coordinate and aiming point's X-coordinate is incremented by movY.

To prevent the table moving out of the screen, the right-bound of X-coordinate is set.

```
double movX = 0;
double movY = 0;

if(y_distance<=175.0 )
{
        GetMoves(5,x_distance,y_distance,x_aim,y_aim,movX, movY,1);
        x_distance += movX;
        y_distance += movY;
        x_aim += movX;
        y_aim += movY;
        mainWindow->RefreshWin();
}

void GetMoves(double moveDis,double camX, double camY, double aimX, double aimY,
double &movX, double &movY, int type)
{
        double dis = sqrt(sqr(aimX-camX)+sqr(aimY-camY));
        if(dis == 0)
        {
                movX = 0;

                movY = 0;

                return;

        }

        movY = moveDis * (aimX-camX) / dis;

        movX = moveDis * (aimY-camY) /dis;


        if(type)

        {

                movX *= -1;

        }

        else

        {

                movY *= -1;

        }


        return;

}
```

➢ **Right**

Right

When the Right button is clicked each time, the camera should move to the right a little bit. As discussed above, in the Side-view, camera may move to any angle in the Side View, the aim point should also move correspondingly to keep the line between camera and aim position the same angle to the axis of the table. We call GetMoves function to calculate the movX and movY, and add the camera's X-coordinate and aiming point's X-coordinate by movX every time, and the camera's X-coordinate and aiming point's X-coordinate by movY.

We also limit the left-bound of X-coordinate to prevent the table moving out of the screen.

```
double movX = 0;
double movY = 0;

if(y_distance >=-175.0)
{
        GetMoves(5,x_distance,y_distance,x_aim,y_aim,movX, movY,0);
        x_distance += movX;
        y_distance += movY;
        x_aim += movX;
        y_aim += movY;
        mainWindow->RefreshWin();
}
```

- **Forward**

Forward

When the Forward button is clicked each time, the snooker table should seem to be closer to the camera a little bit. Here we change the camera's X-coordinate and aiming point's X-coordinate and Z-coordinate. Each time the camera's Z-coordinate is added by 2.0, and camera's aiming point Z-coordinate is added by 0.05, X-coordinate is added by 0.2.

At the same time, we limit the bound of X-coordinate.

```
if(x_distance<=-50.0 && VIEW_MODE==SIDE_VIEW)
{
        x_distance +=2.0;
        x_aim +=0.2;
        z_aim +=0.05;
         mainWindow->RefreshWin();
}
```

- **Backward**

Backward

When the Backward button is clicked each time, the snooker table should seem to be farther to the camera a little bit. Here we change the camera's X-coordinate and aiming point's X-coordinate and Z-coordinate. Each time the camera's Z-coordinate is subtracted by 2.0, and camera's aiming point Z-coordinate is subtracted by 0.05, X-coordinate is subtracted by 0.2.

At the same time, we limit the bound of X-coordinate.

```
if(x_distance>=-210.0 && VIEW_MODE==SIDE_VIEW)
```

```
{
    x_distance -=2.0;
     x_aim-=0.2;
    z_aim -=0.05;
    mainWindow->RefreshWin();
}
```

➢ **Turn Right**

Turn Right

When the Turn Right button is clicked each time, the snooker table should turn to right a little bit. Here we change rotating angle's value of the snooker table. Each time the angle is incremented by 2.0 degrees.

```
if(VIEW_MODE==SIDE_VIEW)
{
   angle += 2.0;
  mainWindow->RefreshWin();
}
```

➢ **Turn Left**

Turn Left

When the Turn Left button is clicked each time, the snooker table should seem to be turned left a little bit. Here we change rotating angle's value of the snooker table. Each time the angle is decremented by 2.0 degrees.

```
if(VIEW_MODE==SIDE_VIEW)
{
        angle -=2.0;
        mainWindow->RefreshWin();
}
```

4.3.2 In Fine Window

In fine window, the camera is always located on the line from cue ball to target ball, and aiming at target ball. As the figure shows below:

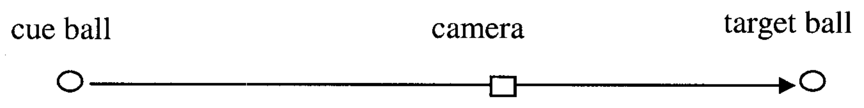


Figure 4.1 Camera and balls positions

Firstly, we need to get the camera's position. What we know now are the position of the cue ball (cueX, cueY, cueZ), the position of the target ball (targetX, targetY, targetZ), and the distance between the camera and the target ball, we need to calculate the X, Y, Z coordinate of the camera. Since the Z-coordinate of the cue ball, target ball and the camera are the same (2.7), we only need to consider the X, and Y coordinates of the camera.

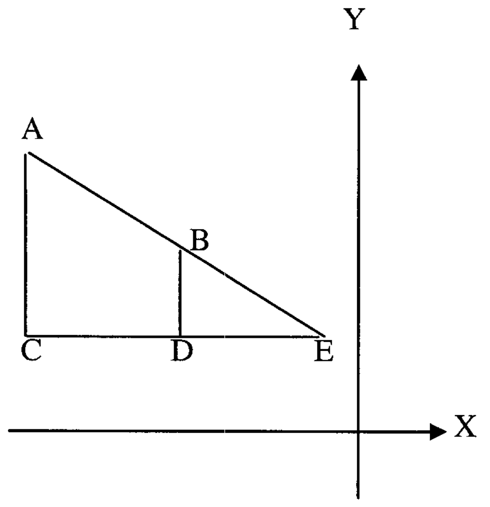


Figure 4.2 Calculate moving distance

See figure on the left, suppose A is cue ball, E is target ball and B is the camera.

The distance between the cue ball and the target ball is:

$|AE| = \sqrt{(\text{targetX}-\text{cueX})^2-(\text{targetY}-\text{cueY})^2}$

Suppose the distance between the target ball and the camera |BE| is increment (a constant).

$$\frac{|AE|}{|BE|} = \frac{|CE|}{|DE|} = \frac{|AC|}{|BD|}$$

|CE|=| cueX -targetX |

|DE|=|cameraX -targetX |

|AC|=|cueY- targetY |

|BD|=|cameraY- targetY |

So:

cameraX=(increment*(cueX-targetX))/|AE| + targetX

cameraY=(increment*(cueballY-targetY)/|AE| + targetY

In fine window, we provide two kinds of camera's movement: Zoom In and Zoom Out.

➢ **Zoom In**

When the Zoom In button is clicked, the target ball should become a bit larger. Here we change the alpha value of the viewing volume for the perspective projection, that is, we reduce the alpha value by 1.0 degree each time, and we limit the bound of the alpha value.

```
if(zoom_in)
  {
          if(fine_alpha >= 5.0)
          fine_alpha -=1.0;
mainWindow->RefreshWin();
  }
```

➢ **Zoom Out**

When the Zoom Out button is clicked, the target ball should seem to be a bit smaller. Here we change the alpha value of the viewing volume for the perspective projection, that is, we increase the alpha value by 1.0 degree each time, and we limit the upper bound of the alpha value to 180 degrees.

```
if(zoom_out)
{
    if(fine_alpha <= 180.0)
    fine_alpha +=1.0;
    mainWindow->RefreshWin();
}
```

4.4 Implementation Problems and Solutions

4.4.1 Draw a circle in Fine Adjust Window

In the Fine Adjust window, whenever the target ball is selected, a circle will be displayed in front of the target represent the position where the cue ball will hit the target ball.

In order to generate such a circle, firstly we need to draw a circle at (0.0, 0.0, 0.0), then rotate the circle along the line which is perpendicular to the cue ball and target ball line, finally move the circle to the target ball position. (See figure 4.3). The most important part is how to find the rotation axis, the calculation is as follows (since all the balls are in the same plane, here we are concerned only with the X and Y value):

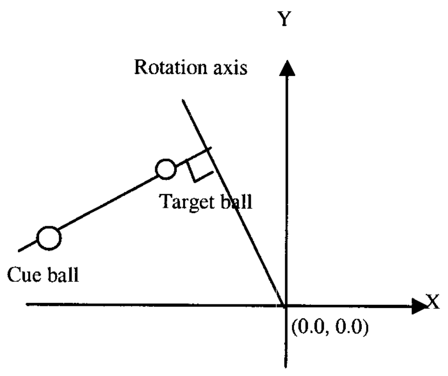


Figure 4.3 Calculate Circle rotation

Suppose the cue ball's coordinate is (cueX, cueY), and target ball's coordinate is (targetX, targetY), and the line of cueball and target ball is:

$A_1X + B_1Y + C_1 = 0$

The rotation axis is:
$A_2X + B_2Y = 0$ (the rotation axis is throgh the original point)

Since the rotation axis is perpendicular to the cue ball and target ball line, $A_1A_2 = -B_1B_2$, we get three equations:

$$\begin{cases} A_1X + B_1Y + C_1 = 0 \\ A_2X + B_2Y = 0 \end{cases}$$

$$A_1A_2 = -B_1B_2$$

$$\Rightarrow \begin{cases} X = -(A_1 * C_1) / (A_1^2 + B_1^2) \\ Y = -(B_1 * C_1) / (A_1^2 + B_1^2) \end{cases}$$

Here the cue-ball target-ball line is through the cue-ball point and target-ball point, so the equation of the line can also be:

$$(x\text{-cueX}) / (\text{targetX-cueX}) = (y\text{-cueY}) / (\text{targetY-cueY})$$

$\Rightarrow$ (targetY-cueY)*x- (targetX-cueX)*Y- (targetY-cueY)*cueX+ (targetX-cueX)*cueY=0

$$\Rightarrow \begin{cases} A_1 = \text{targetY-cueY} \\ B_1 = \text{targetX-cueX} \end{cases}$$

$$C_1 = -\ (\text{targetY-cueY})*\text{cueX} + (\text{targetX-cueX})*\text{cueY}$$

4.4.2 Issues with Fine Adjust Window

When drawing Fine Adjust Window using the approach discussed in 4.4.1, we will meet some unexpected cases if we draw the cue ball as in the real Snooker table. If the target ball is very close to the cue ball, the camera will locate right behind the cue ball, in this case, we can only see a very big cue ball, the target ball and the mapping circle are all blocked by the cue ball. In the Fine Adjust

Window, what we care about are the target ball and the mapping circle of the cue ball, the cue ball itself is not a critical factor, so we skip drawing the cue ball in the Fine Adjust Window to solve this issue.

In the 2D version, the Fine window is a manually-drawn pseudo-3D view, all the balls on the line between cue ball and target ball will be displayed, that's to say, if a new ball enter the view during adjusting, the camera focus will be changed to this new ball. But it's difficult to implement it in the 3D version. In 3D Snooker Simulator, Fine Adjust Window supplies a real 3D view, it's not easy to get the position of the new ball. So, to simplify the case, in Fine Adjust Window, we ignore the new ball enter the view, it's up to the player to decide if some other ball is on the way of the cue ball by observing the real Snooker table.

4.4.3 Continuous moving by setting Timer

In the Snooker Simulator, some functions are supplied to control the camera, such as turning, zooming. To improve the UI friendliness, we allow the user to press the button, and the camera will keeping moving (zooming, turning) till the user release the mouse button instead of keeping clicking the button. To implement this feature, we declare a flag variable for each function, when the button is pressed, the flag is set to true, when the button is released, the flag will be set to false. We use *glutSetTimerFunc* to set a timer, the applications will check the flags periodically, if the flag is false, nothing is done, if the flag is true, some bound functions will be executed for moving, zooming or turning. We also use this mechanism to implement auto-hidable windows.

It's important to define how long the timer interval should be set. Too short interval will degrade the performance, too long interval will make the moving not smooth. After testing, we use 100ms as the timer interval to balance the performance and moving effect.

4.4.4 Menu Selection

To gain a clear and friendly user interface, we discard the old standard dropdown menu system. Two sets of menus are supplied, one is the main menu page, and the other one is the playtime taskbar. Main menu page lists all main menus and some functions not available during the play-time such as Time Testing, the player can get what main functions the game supplies from the opening screen.

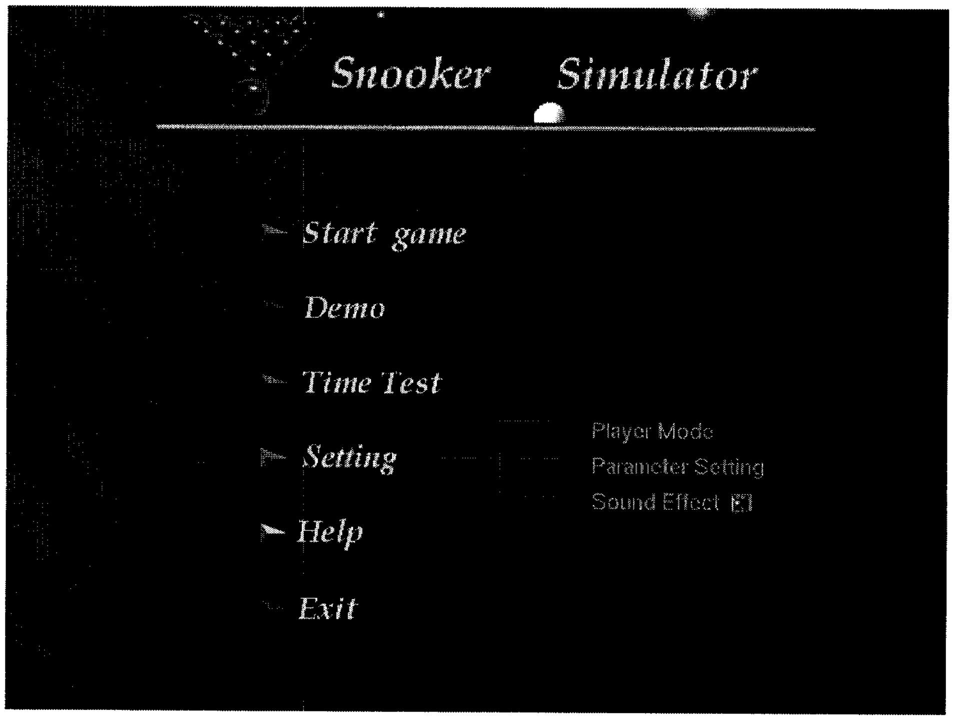


Figure 4.4 Main Menu Page

The Main Menu Page includes following functions:

- Start game
- Demo
- Time Testing
- Setting
- Help
- Exit the application

The playtime taskbar is simulating the taskbar of window system, it allows the player to get all the accessible functions at any time very easily. Though the icons on the taskbar are very simple and small, with Tooltips help, the players can understand their functions without any difficulty.

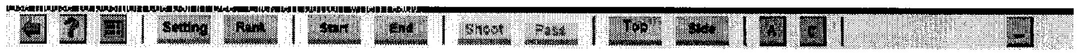


Figure 4.5 Play-time Taskbar

The main functions supplies in taskbar include:

- Exist the game
- Help
- Scoreboard control
- Setting
- Rank
- Start a game
- Stop a game
- Shoot
- Pass
- Switch to Top View
- Switch to Side View
- Hide/Show Adjustment window
- Hide/Show Control Panel
- Minimize the game

4.4.5 Blending

In the 3D version, all the menus are made of texture mapping. Sometimes, we choose a background image firstly, and then put the menu images on it. The background should be transparent when the menu image combined. In OpenGL, new fragment will overwrite any existing color values if blending is not applied. With blending, we can control how the existing color value should be combined with the new fragment's value. Here we use ***glBlendFunc( )*** to specify how the source color and destination color are combined.

For example, in our taskbar, we set the alpha value of the background (destination) is 0.0, and set the alpha value of menu image (source) is 1.0, then we use glEnable(GL_BLEND) to enable blending, glBlendFunc(GL_ONE, GL_SRC_ALPHA) to set the source factor to GL_ONE and destination factor to GL_SRC_ALPHA which equal to the destination's alpha value (here is 0.0), in such way, the background image can be transparent when the menu image above it. (See figure4.4).

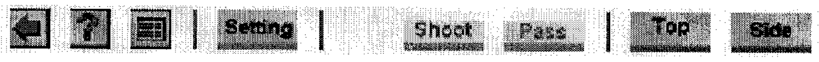
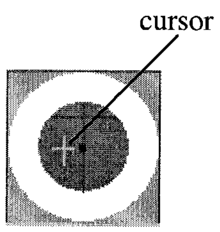


Figure 4.6 Taskbar with blending

4.4.6 Change the cursor when mouse entering into certain area

In order to make some special effect, sometimes we need to change the cursor when the mouse moves to certain area. For example, to set the spin, you need to move the mouse to the white green ball image and click. Here we change the curse to crosshair when the mouse moves to green area so as to let the user know which area he can set the spin.



To achieve this effect, we use ***glutSetCursor( )*** to change the cursor, and register ***glutPassiveMotionFunc( )*** callback after.

4.4.7 Simulating Ball-like cursor on Snooker Table

Ball-like cursor is used for locating, selecting a ball in snooker table. Glut supplies circle cursor, but it does not meet our necessity since the size of the circle curse cannot be changeable as the camera zooming. According to our design, the cursor should be the same size as balls, and the size of the cursor should be able to change correspondingly when zooming in and out, then we can preview the ball position from the cursor. To obtain such effect, a BallCursor class is created to simulate a customized ball-like cursor. When mouse moves in the snooker table, the BallCursor object will take place the real cursor. This ball-like cursor does not perform as well as the system-supplied cursor, there is a little delay to the mouse movement, but the player may not feel it if the machine is fast enough.

4.4.8 Position balls on Snooker Table

In a game, the player needs to position the cue ball or select a position as the target of the cue ball. Though Top View and Side View are available for the player, we only allow the player to do these actions in the Top View. In the side view, the coordination system is distorted, it's very difficult to transform the mouse click position from window coordination to world coordination. If the player clicks mouse, we suppose he want to position, the view will switch back to Top View automatically.

4.4.9 Multiple windows management

In Snooker Simulator, the main scene is divided into five parts: Scoreboard, Snooker table, Control panel, Adjust window, and Task Bar. Each part is relatively independence to each other, has its own display function and callbacks. To support this characteristic, we implement the system as a multi-level window system, the main window is divided into different subwindows, each with its own OpenGL context and callbacks. In detail, we use ***glutCreateSubWindow( )*** to create each subwindow (which is encapsulated in CreateWin function of VirtualWindow class), write its own init function, and register its own display function and callbacks. Whenever we want any window be the current window, we use ***glutSetWindow( )*** (which is encapsulated in VirtualWindow class as SetAsCurrentWindow function) to set it. But since OpenGL requires some functions and variable be global, till now, we cannot implement OpenGL callback functions as class methods, so we cannot encapsulate everything into class. Current multi-level window system is a relatively loose system, say subwindow and its parent are relatively independent. Though they're physical related, the subwindows deal with they own events individually, not controlled by their parent window.

The relation between the main window and subwindows is show below:

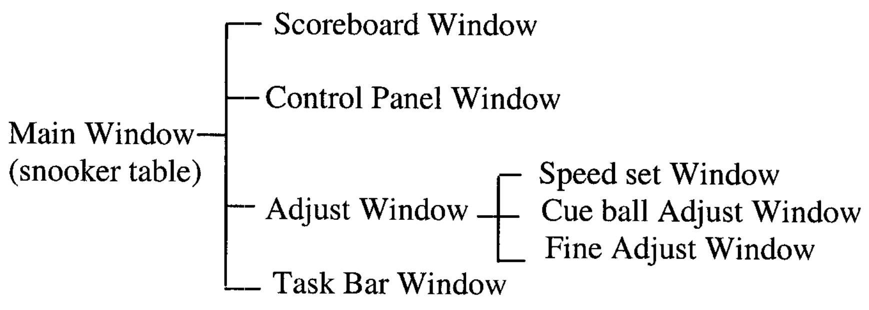


Figure 4.7 Multi-window structure

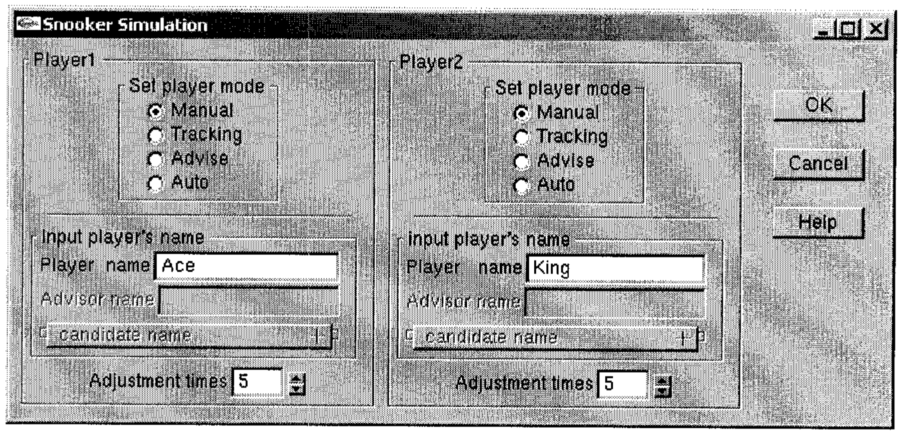
4.4.10 Tool Tips

As a game, a friendly user interface is very important. The users should be able to get help information easily and quickly when it's needed. A player may be impatient if he has to call the help window frequently and has to click OK to close the window every time. Besides making the UI as simple as possible, we still supply tooltips for important controls to give users immediate help. When the mouse moves over a control, tooltips is displayed to give user more information on the control, when the mouse leaves the control, the tooltips will hide automatically.

4.4.11 Glui Window

GLUI is a third-party GLUT-based C++ user interface library, which provides controls such as buttons, checkboxes, radio buttons, spinners, and listboxes for creating UI, controls easily in OpenGL applications. It is window-based independent, relying on GLUT to handle all system-dependent issues, such as window and mouse management.

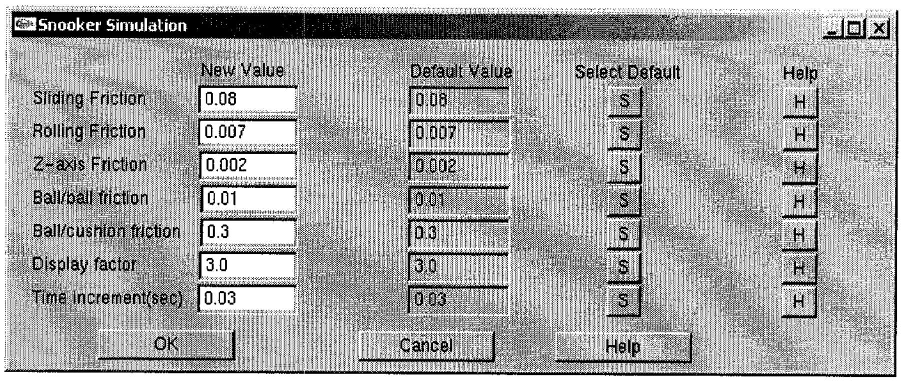
Though GLUI supplies a lot of useful controls, but it's difficult to customize the controls, such as color. In addition, its auto-layout mechanism prevents users to organize the controls to meet their special requirements. All these limitations make the GLUI not suitable for a game context. So, in Snooker Simulator, we only use GLUI to create our setting windows -- Player Mode Setting window and Parameter Setting Window (See figure 4.6 and 4.7). The GLUI window style is different from our own-created windows, but it saves us a lot of time to deal with the settings, and it only displays in a relatively independent context, it doesn't break the style uniform of the user interface.



Player1
Set player mode: [✓ Manual] Tracking Advise Auto
Input player's name
Player name: Ace
Advisor name:
Adjustment times: 5

Player2
Set player mode: [✓ Manual] Tracking Advise Auto
Input player's name
Player name: King
Advisor name:
Adjustment times: 5

Figure 4.8 Player Mode Setting



	New Value	Default Value	Select Default	Help
Sliding Friction	0.08	0.08	S	H
Rolling Friction	0.007	0.007	S	H
Z-axis Friction	0.002	0.002	S	H
Ball/ball friction	0.01	0.01	S	H
Ball/cushion friction	0.3	0.3	S	H
Display factor	3.0	3.0	S	H
Time increment(sec)	0.03	0.03	S	H

Figure 4.9 Parameter Setting

4.4.12 Display List

3D object displaying is more complicated than 2D graphics, though hardware, such as 3D graphic accelerate card, makes 3D object displaying much faster than before, it's still time-consuming. Display list is a set of pre-defined OpenGL statements, which have been compiled. Once it's recorded, it can be called many times at any time, and it can be executed much faster than non-listed statements. To optimize the performance, display list is used for some complicated objects, such as the snooker table.

4.4.13 Calculate Time-increment for best performance

Both 3D and 2D version supply a function of time-testing – the application simulates a full-speed hitting and give player a suggestion on Time-increment setting according to the simulating time and real time so as to gain best performance based on working machine's property. But, 3D graphics rendering is slower than 2D graphics rendering though OpenGL has already supplied best performance. Using the existing formula cannot get a proper result.

```
sim_time = path->time();

act_time = (finish - start) / CLK_TCK;

new_dt = Time-increment * (act_time / sim_time);
```

After testing, we added a factor for calculating the simulating time, that is:

```
sim_time = path->time () / 6;
```

With this modification, we can gain a satisfying performance using the suggested Tim-increment setting. Anyhow, this factor may not be suitable for all machines, we have no way to test it on many kinds of machines to get a generic formula.

4.4.14 Fix out-of-virtual-memory defect

During the system testing, we encountered a big problem that may causes the system out of virtual memory and crashed. It mainly happened when performing the demo, it may happen at any time (5-60 minutes) after the demo starts. It seems the applications tries to find the next target ball, but failed, the used memory increasing dramatically up to 150M and cause the system out of virtual memory. When tracing into the codes, we found that a variable may be 0 and cause a "divided by zero" exception, but the system didn't throw out the exception, we suppose it as the reason of the issue. After solving this problem, the performance got some improvement, but the memory issue still happened.

Finally, we found second displaying of Snooker table in the Fine Adjust window caused the problem. When we remove the Fine Adjust window or skipped the display table function in that table, the memory issue disappeared. We analyze that two display functions share the same global variables, confliction may happen in some special cases. So, we declare different global variables for each one, but it still didn't solve the problem completed though some improvement achieved. We continue trace into the display function of the Table class, and found it happened when drawing corners of the table, we tried many methods to solve it but failed. Since, the table corners are not important in the Fine Adjust window, as a workaround, we skip drawing table corners in the Fine Adjust window, and solve the problem successfully. After several long-time testing (up to 8 hours), memory issue never happened again. Though it's not the best solution, but as a trade-off between performance and developing time, it's still acceptable.

5. Conclusion

Porting Snooker Simulator from 2D to 3D is a complicated project. Though our project mainly focuses on graphic and UI implementation, we experienced almost every step of software development -- understanding the existing application, redesign, 3D object modeling, 3D graphic and UI implementation, testing, system integration, system testing and software maintenance.

This project involves a lot of concepts and technology across many fields of computer science, such as software reverse engineering, object-oriented analysis, design and programming, computer graphic programming, User Interface design, software and system testing. Through working on this project, we got a big improvement with all these knowledge and a deeper understanding on software development, esp. the game development.

After long-time hard working, we reserve all the functionalities of 2D version in the new 3D Snooker Simulator, implement a real-like, smooth 3D graphic and friendly game-like user interface. We achieved almost all our project goals.

Of course, there're still some aspects need to be improved, such as 3D movement performance, anyhow, we still can feel some flashing when ball is moving, especially when the machine is not fast enough. In addition, though we're trying to make the application to be platform-independent, there still exist two functions depending on Win32 API – MessageBox and PlaySound. To make 3D Snooker Simulate to be able to run other operating system, some modifications are still needed.

Conclusively, after working on this project, we really feel 3D game development is an interesting, exciting and challenging field. We're fascinated by it.

6. References

1. [The Red Book]

OpenGL Architecture Review Board, OpenGL Programming Guide, second edition, 1997, Addison-Wesley Developers Press. ISBN: 0-201-46138-2

2. [The Snooker Simulation]

Peter Grogono, The Snooker Simulation

http://www.cs.concordia.ca/~faculty/grogono/snooker.html

3. [OpenGL Game]

Dave Astle, et al , OpenGL Game Programming, 2002, Premier Press. ISBN: 0761533303. http://www.codeguru.com/columns/gamedev/OpenGL-01.html

4. [OpenGL Example]

Xiang, Shiming OpenGL Programming and Examples, 1999, Publishing House of Electronic Industry, China. ISBN:7-5053-5625-9/TP • 2880

Appendix A. Main Source code for 3D implementation

1. Display parameter set window using GLUI

```
void ParameterSetWin::ShowWin()
{
    RefreshData();
    SaveOld();
    {
            m_gluiWin->show();
            m_isVisible = true;
    }
}


void ParameterSetWin::AddControls()
{


    panel1 = m_gluiWin ->add_panel("", GLUI_PANEL_NONE);

    m_gluiWin->add_statictext_to_panel(panel1, "                     New Value");
    setText1 = m_gluiWin->add_edittext_to_panel(panel1, " Sliding Friction       ",
    GLUI_EDITTEXT_FLOAT, &t1);
    setText1->set_w(200);
    setText1->set_float_val(paraManager->get_sliding_friction());// 0.0800);
    setText1->set_alignment(GLUI_ALIGN_LEFT);

    setText2 = m_gluiWin->add_edittext_to_panel(panel1, " Rolling Friction       ",
    GLUI_EDITTEXT_FLOAT, &t2);
    setText2->set_w(200);
    setText2->set_float_val( paraManager->get_rolling_friction() );//0.0070);
    setText2->set_alignment(GLUI_ALIGN_LEFT);

    setText3 =m_gluiWin->add_edittext_to_panel(panel1, " Z-axis Friction       ",
    GLUI_EDITTEXT_FLOAT, &t3);
    setText3->set_w(200);
    setText3->set_float_val(paraManager->get_z_spin_friction() );//0.0020);
    setText3->set_alignment(GLUI_ALIGN_LEFT);

    setText4 =m_gluiWin->add_edittext_to_panel(panel1, " Ball/ball friction       ",
    GLUI_EDITTEXT_FLOAT, &t4);
    setText4->set_w(200);
    setText4->set_float_val(paraManager->get_ball_ball_friction() );//0.0100);
    setText4->set_alignment(GLUI_ALIGN_LEFT);

    setText5 =m_gluiWin->add_edittext_to_panel(panel1, " Ball/cushion friction  ",
    GLUI_EDITTEXT_FLOAT, &t5);
    setText5->set_w(200);
    setText5->set_float_val(paraManager->get_ball_cushion_friction()); //0.3000);
    setText5->set_alignment(GLUI_ALIGN_LEFT);
```

```
setText6 =m_gluiWin->add_edittext_to_panel(panel1, " Display factor          ",
GLUI_EDITTEXT_FLOAT, &t6);
setText6->set_w(200);
setText6->set_float_val( paraManager->get_display_factor()); //3);
setText6->set_alignment(GLUI_ALIGN_LEFT);

setText7 =m_gluiWin->add_edittext_to_panel(panel1, " Time increment(sec)  ",
GLUI_EDITTEXT_FLOAT, &t7);
setText7->set_w(200);
setText7->set_float_val(paraManager->get_time_increment());//0.0040);
setText7->set_alignment(GLUI_ALIGN_LEFT);

m_gluiWin->add_button("OK", 0, ParameterOK);


m_gluiWin->add_column(false);

panel2=m_gluiWin->add_panel("", GLUI_PANEL_NONE);
m_gluiWin->add_statictext_to_panel(panel2, "            Default Value");

defaultText1 =m_gluiWin->add_edittext_to_panel(panel2, " ", GLUI_EDITTEXT_FLOAT,
default_parameter[1]);
defaultText1->set_float_val(paraManager->get_default_sliding_friction());//0.0800);
defaultText1->disable();

defaultText2 =m_gluiWin->add_edittext_to_panel(panel2, " ", GLUI_EDITTEXT_FLOAT,
default_parameter[2]);
defaultText2->set_float_val(paraManager->get_default_rolling_friction());//0.0070);
defaultText2->disable();

defaultText3 =m_gluiWin->add_edittext_to_panel(panel2, " ", GLUI_EDITTEXT_FLOAT,
default_parameter[3]);
defaultText3->set_float_val(paraManager->get_default_z_spin_friction());//0.0020);
defaultText3->disable();

defaultText4 =m_gluiWin->add_edittext_to_panel(panel2, " ", GLUI_EDITTEXT_FLOAT,
default_parameter[4]);
defaultText4->set_float_val(paraManager->get_default_ball_ball_friction());//0.0100);
defaultText4->disable();

defaultText5 =m_gluiWin->add_edittext_to_panel(panel2, " ", GLUI_EDITTEXT_FLOAT,
default_parameter[5]);
defaultText5->set_float_val(paraManager->get_default_ball_cushion_friction());//0.3000);
defaultText5->disable();

defaultText6 =m_gluiWin->add_edittext_to_panel(panel2, " ", GLUI_EDITTEXT_FLOAT,
default_parameter[6]);
defaultText6->set_float_val(paraManager->get_default_display_factor());//3);
defaultText6->disable();

defaultText7 =m_gluiWin->add_edittext_to_panel(panel2, " ", GLUI_EDITTEXT_FLOAT,
 set_parameter[7]);
defaultText7->set_float_val(paraManager->get_default_time_increment());
defaultText7->disable();
m_gluiWin->add_button("Cancel", 0, ParameterCancel);
```

```
    m_gluiWin->add_column(false);

    panel3=m_gluiWin->add_panel("", GLUI_PANEL_NONE);
    m_gluiWin->add_statictext_to_panel(panel3, "   Select Default");
    s_button1=m_gluiWin->add_button_to_panel(panel3, "S", SET_SLIDING, ParameterReset);
    s_button1->set_w(20);
    s_button2=m_gluiWin->add_button_to_panel(panel3, "S", SET_ROLLING, ParameterReset);
    s_button2->set_w(20);
    s_button3=m_gluiWin->add_button_to_panel(panel3, "S", SET_ZAXIS, ParameterReset);
    s_button3->set_w(20);
    s_button4=m_gluiWin->add_button_to_panel(panel3, "S", SET_BALL_FRICTION,
    ParameterReset);
    s_button4->set_w(20);
    s_button5=m_gluiWin->add_button_to_panel(panel3, "S", SET_CUSION_FRICTION,
    ParameterReset);
    s_button5->set_w(20);
    s_button6=m_gluiWin->add_button_to_panel(panel3, "S", SET_DISPLAY_FACTOR,
    ParameterReset);
    s_button6->set_w(20);
    s_button7=m_gluiWin->add_button_to_panel(panel3, "S", SET_TIME_INCREMENT,
    ParameterReset);
    s_button7->set_w(20);
    m_gluiWin->add_button("Help", SET_PARAMS_HELP, ParameterHelp);

    m_gluiWin->add_column(false);
    panel4=m_gluiWin->add_panel("", GLUI_PANEL_NONE);
    m_gluiWin->add_statictext_to_panel(panel4, "        Help");
    h_button1=m_gluiWin->add_button_to_panel(panel4, "H", SET_SLIDING_HELP,
    ParameterHelp);
    h_button1->set_w(20);
    h_button2=m_gluiWin->add_button_to_panel(panel4, "H", SET_ROLLING_HELP,
    ParameterHelp);
    h_button2->set_w(20);
    h_button3=m_gluiWin->add_button_to_panel(panel4, "H", SET_ZAXIS_HELP, ParameterHelp);
    h_button3->set_w(20);
    h_button4=m_gluiWin->add_button_to_panel(panel4, "H", SET_BALL_FRICTION_HELP,
    ParameterHelp);
    h_button4->set_w(20);
    h_button5=m_gluiWin->add_button_to_panel(panel4, "H", SET_CUSION_FRICTION_HELP,
    ParameterHelp);
    h_button5->set_w(20);
    h_button6=m_gluiWin->add_button_to_panel(panel4, "H", SET_DISPLAY_FACTOR_HELP,
    ParameterHelp);
    h_button6->set_w(20);
    h_button7=m_gluiWin->add_button_to_panel(panel4, "H", SET_TIME_INCREMENT_HELP,
    ParameterHelp);
    h_button7->set_w(20);
}

void ParameterSetWin::RestoreOld()
{
    t1 = ot1;
    t2 = ot2;
    t3 = ot3;
    t4 = ot4;
    t5 = ot5;
    t6 = ot6;
    t7 = ot7;

}

void ParameterSetWin::SaveOld()
{
    ot1 = t1;
    ot2 = t2;
    ot3 = t3;
    ot4 = t4;
    ot5 = t5;
    ot6 = t6;
    ot7 = t7;
}

void ParameterSetWin::Update()
{
    m_gluiWin->sync_live();
}

void ParameterSetWin::ResetAsDefault(int index)
{
    switch(index)
    {
    case SET_SLIDING:
            t1 = paraManager->get_default_sliding_friction();
            break;
    case SET_ROLLING:
            t2 = paraManager->get_default_rolling_friction();
            break;
    case SET_ZAXIS:
            t3 = paraManager->get_default_z_spin_friction();
            break;
    case SET_BALL_FRICTION:
            t4 = paraManager->get_default_ball_ball_friction();
            break;
    case SET_CUSION_FRICTION:
            t5 = paraManager->get_default_ball_cushion_friction();
            break;
    case SET_DISPLAY_FACTOR:
            t6 = paraManager->get_default_display_factor();
            break;
    case SET_TIME_INCREMENT:
            t7 = paraManager->get_default_time_increment();
            break;
    }
    m_gluiWin->sync_live();
}

void ParameterSetWin::UpateParameters()
{
    paraManager->set_sliding_friction(t1);
    paraManager->set_rolling_friction(t2);
    paraManager->set_z_spin_friction(t3);
    paraManager->set_ball_ball_friction(t4);
    paraManager->set_ball_cushion_friction(t5);
    paraManager->set_display_factor(t6);
    paraManager->set_time_increment(t7);
}

void ParameterSetWin::RefreshData()
{
    t1 = paraManager->get_sliding_friction();
    t2 = paraManager->get_rolling_friction();
    t3 = paraManager->get_z_spin_friction();
    t4 = paraManager->get_ball_ball_friction();
    t5 = paraManager->get_ball_cushion_friction();
    t6 = paraManager->get_display_factor();
    t7 = paraManager->get_time_increment();

    m_gluiWin->sync_live();
}
```

2. Draw ball-shape cursor

```
//Display the ball-shape cursor

void BallCursor::Display()

{

    if(m_showCursor)

    {

            glDisable( GL_LIGHTING );

            glDisable(GL_DEPTH_TEST);

            glColor3fv(RGBColors[m_Color]);

            glPushMatrix();


            glTranslatef(m_xPos, m_yPos, 0.0);


            glBegin(GL_POINTS);

            glVertex3f(0.0, 0.0, 0);

            glEnd();


            const int STEPS = 1000;
```

```
                double ang = 0;
                glBegin(GL_LINES);
                for(int k = STEPS; k< 3*STEPS; k++)
                {
                  ang = PI*(k/(2.0 /*m_Radius*/*STEPS));
                  glVertex3f(m_Radius*cos(ang),m_Radius*sin(ang),0);
                }
                for(int p = STEPS; p< 3*STEPS; p++)
                {
                  ang = PI*(p/(/*m_Radius*/2.0*STEPS));
                  glVertex3f(-m_Radius*cos(ang),-m_Radius*sin(ang),0);
                }
                glEnd();
                glPopMatrix();
                glEnable(GL_DEPTH_TEST);
                glEnable( GL_LIGHTING );
        }
}
```

3. Display Fine Adjust Window

```
void FineAdjustWin::Display()
{
        float A, B, C;
        glutSetWindow(m_thisWin);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        float cueballX=cueBall->get_xs();
        float cueballY=cueBall->get_ys();
        float targetXX=cueBall->get_targetX();
        float targetYY=cueBall->get_targetY();


        float a=sqrt(((cueballX-targetXX)*(cueballX-targetXX))+((cueballY-targetYY)*(cueballY-
targetYY)));
        float increment; //=a-2.7; //-3.0;
        if( a<=5.4)
                increment=0.5;
        if (a <30.0 && a>5.4)
                increment=a-3.5;
        if (a >=30.0)
                increment=20.0;
        if(a>0)
        {
                target_positionX=(increment*(cueballX-targetXX))/a + targetXX; //cueballX;
                target_positionY=(increment*(cueballY-targetYY))/a + targetYY; //cueballY;
                target_positionZ=2.7; //18.0;
        }
        else
        {
                target_positionX = 0;
                target_positionY = 1;
                target_positionZ = 2.7;
        }

        target_aimX=cueBall->get_targetX();
        target_aimY=cueBall->get_targetY();
        target_aimZ=2.7;
        A=target_aimY-cueballY;
        B=-(target_aimX-cueballX);
        C=-(A*cueballX)-B*cueballY;
        float pp = A*A+B*B;
        float roX = 0;
        float roY = 0;
        if(pp != 0)
        {
                roX=-(A*C)/pp;
                roY=-(B*C)/pp;
        }

        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();

        gluPerspective(fine_alpha,/*(adjustWidth - cueWinWidth + 2)*/184/(2*SPIN_RAD + 2), 0.05,
500.0);

        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, gray);

        gluLookAt(target_positionX, target_positionY, target_positionZ,
                target_aimX, target_aimY, target_aimZ, 0.0, 0.0, 1.0);

        glLightfv(GL_LIGHT0, GL_POSITION, pos);
        glLightfv(GL_LIGHT0, GL_AMBIENT, ambient_light);
```

```
table2->draw_table();

for(int i = 2; i <NUMBALLS; i++)  //skip drawing cue ball
{
        if(balls[i])
                balls[i]->DrawBall();
}

glDisable(GL_LIGHTING);
glDisable(GL_DEPTH_TEST);
glColor3f(1.0, 1.0, 1.0);

glPushMatrix();
glPointSize(3.0);
glBegin(GL_POINTS);
glVertex3f(target_aimX, target_aimY, 2.7);
glEnd();
glPopMatrix();

glPushMatrix();
glTranslatef(target_aimX, target_aimY, 2.7);
glRotatef(90.0, roX, roY, 0.0);
const int STEPS = 1000;
double ang = 0.0;
glBegin(GL_LINES);
for(int k = STEPS; k< 3*STEPS; k++)
{
```

```
            ang = PI*(k/(2.0*STEPS));

            glVertex3f(2.5*cos(ang),2.5*sin(ang),2.7);

        }

        for(int p = STEPS; p< 3*STEPS; p++)

        {

            ang = PI*(p/(2.0*STEPS));

            glVertex3f(-2.5*cos(ang),-2.5*sin(ang),2.7);

        }

        glEnd();

        glPopMatrix();

        glEnable(GL_DEPTH_TEST);


        glEnable(GL_LIGHTING);

        glFinish();

        glutSwapBuffers();

    }
```

4. Display main window

```
//display the main window

void display(void)

{

    float pointX=0, pointY=0


    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glMatrixMode(GL_PROJECTION);

    glLoadIdentity();

    gluPerspective(alpha,w/h, 0.5, 500.0);
```

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();


if (VIEW_MODE==SIDE_VIEW)
{
        gluLookAt(x_distance,y_distance,z_distance, x_aim, y_aim, 2.7, 0.0,0.0,1.0);
}
else if (VIEW_MODE==TOP_VIEW)
{
        gluLookAt(top_positionX, top_positionY, top_positionZ, top_aimX, top_aimY,
                top_aimZ, 0.0, 1.0,0.0);


}


if(VIEW_MODE==SIDE_VIEW) //!=TOP_VIEW)
{
        glRotatef(angle, 0.0,0.0,1.0);
}
else
        glTranslatef(0.0,15.0,0.0);


glLightfv(GL_LIGHT0, GL_POSITION, pos);
glLightfv(GL_LIGHT0, GL_AMBIENT, ambient_light);
table->draw_table();


for(int i = 0; i <NUMBALLS; i++)
{
        if(balls[i])
```

```
                {
                        balls[i]->DrawBall();
                }
        }


        cue->DrawCue();
        path->DrawPath();


        ballCursor->Display();
        glFinish();
        glutSwapBuffers();
    }
```

Appendix B. User Manual

Introduction

Introduction to the 3D Snooker Simulator

The 3D Snooker Simulator is a real 3D snooker game. Current version is a window-based standalone version.

It supplies four play modes – manual, track, advise, and auto.

♦ **Manual.** In manual mode, you have to do all the work of playing a shot yourself. To set a player to manual mode, first click on the `Manual` button for either Player 1 or Player 2, and then enter the player's name in the `Name` box at the top of the dialog box.

♦ **Tracking.** Tracking mode is similar to manual mode except that Snooker displays the "tracks" that the balls will follow. The tracks are not displayed until you have set the ball's speed. After you have set the ball's speed, Snooker responds to any change in the controls by re-displaying the tracks. To set a player to manual mode, first click on the `Tracking` button for either Player 1 or Player 2, and then enter the player's name in the `Name` box at the top of the dialog box.

The track of a ball usually consists of a white part and a black part. The white part corresponds to sliding motion and the black part to rolling motion. The cue ball will slide after impact unless you hit it on the "natural roll" line, indicated by a horizontal red line on the ball image at the center of the control window.

♦ **Advise.** In advise mode, Snooker suggests a shot for you to play. You can accept the suggestion and simply play the shot, or you can use the controls to change the shot.

To set a player to advise mode, first click on the `Advise` button for the player. Next, choose an advisor from the `Auto-Players` window near the bottom of the dialog box. Finally, enter the player's name in the `Name` box at the top of the dialog box

- **Automatic.** In automatic mode, Snooker plays the shot itself. To set a player to automatic mode, first click on the `Automatic` button for the player and then choose a player from the `Auto-Players` drop-down list.

The players are allowed to change the environment properties, such as ball-ball collision, ball-cushion collision, etc.

To gain the best 3D performance, the player can change display factor and time increment. The Snooker Simulator supplies a Time-testing function to calculate the best time increment automatically.

The game is playing at full-screen mode. The player can go to desktop by clicking the Minimizing the game button at any time and go back to the game by clicking the icon of the game on the system task bar.

Introduction to Snooker Game

How to Play Snooker

The objective of snooker is to pot balls. If the position prevents you from potting a ball, you try to prevent your opponent from potting balls.

The game begins with 15 red balls on the table. A player starts by trying to pot a red. If the shot is successful, the player nominates a colored ball, and tries to pot that. If the color goes in, the player goes for another red, and so on.

When the last red has been potted, the player nominates and tries to pot a color. After this color has been potted, it is re-spotted, and all of the colors are potted in order of increasing value: yellow, green, brown, blue, pink, and black.

If a player fails to pot a ball, the "innings" ends, and the other player comes to the table.

The following shots are "fouls".

- ♦ The cue ball hits the wrong ball first. (The "wrong ball" is a color if the player is aiming for reds, and a red or a color other than the nominated color if the player has nominated a color.)
- ♦ A ball other than a red or the nominated color goes into a pocket. Potting more than one red in a single shot is allowed.
- ♦ The cue ball does not hit any balls at all.
- ♦ The cue ball goes into a pocket, either directly or after hitting another ball.
- ♦ The cue ball goes off the table. (This should not happen in The Snooker Simulation.)

After a foul, the player's innings ends and the other player comes to the table with an option: to play a shot or to "pass". In The Snooker Simulation, the `Pass` button (on the right of the `Shoot` button) is highlighted after a foul. The incoming player passes by clicking on it.

In The Snooker Simulation, it is not usually necessary to "nominate" a color, because Snooker can determine which ball the cue ball will strike first. However, if you set the shot up in such a way that the cue-ball will not strike a color, Snooker will display a dialog box asking which color you are trying to hit.

Reds that are potted remain off the table. After a color has been potted, it is returned to the table and "spotted". The rules for spotting a ball are quite complicated, because there may be another ball on or close to the color's normal spot. You do not need to worry about these rules, because Snooker always spots balls correctly.

Snookers

The word "snooker" is used in two senses in the game of Snooker. If your opponent plays a shot that leaves the balls positioned so that you cannot hit any legal ball directly, you are "snookered". There is no score or penalty associated with a snooker, but your opponent is hoping to gain points either because you miss or because, in escaping from the snooker, you will leave an easy shot.

Suppose that your opponent plays a foul shot that leaves you without a clear shot on a legal ball. You have a "clear shot" if you can hit the object ball at any desired angle without your shot being blocked by another ball. If you do not have a clear shot on at least one legal ball, you are "snookered by a foul" and you have a "free ball". This means that you can nominate any ball on the table and try to pot it. The ball you have nominated is treated, for scoring purposes, as if it was a legal ball.

For example, suppose there is one red on the table. Your opponent tries to pot it, but misses altogether and leaves you without a clear shot. You claim a "free ball", nominate the green as your free ball, and pot it. You score one point for the green (as if it was a red), and the green is repotted. You then nominate a color and try to pot it, just as if you had potted a red.

Since Snooker implements the "snookered by a foul" and "free ball" rules, you do not have to worry about the details.

Scoring

Since Snooker keeps the score in the window that appears below the table image on the screen, it is not necessary to describe scoring in detail here. Here is a summary of the scoring rules.

♦ If you pot a red, you get one point. If more than one red goes into a pocket, you get one point for each red potted.

♦ If you pot a color, you get the value of the color: yellow, 2 points; green, 3 points; brown, 4 points; blue, 5 points; pink, 6 points; and black, 7 points.

♦ If you play a foul shot, your opponent receives penalty points. The minimum penalty is 4 points. If the foul involves a ball with a value higher than 4 points, the penalty is the value of the ball. A foul on the blue incurs 5 penalty points; on the pink, 6 penalty points; and on the black, 7 penalty points.

The program displays a quantity called "Reserve" in the Information box. The reserve is calculated as follows:

1. Calculate the "points on the table" assuming that the current player will pot a black after each red. Thus the number of points on the table is 8 times the number of reds plus the value of all the colors, namely, 27. For example, at the beginning of a frame, there are 15 reds on the table and each player has a reserve of $8 \times 15 + 27 = 147$.
2. Add the current player's score.
3. Subtract the other player's score.

The reserve is therefore the amount by which the current player would win by playing a "perfect" game. A player whose reserve is negative must force the other player to foul in order to win the game. An automatic player with a small, negative reserve may attempt to snooker the other player. When the shortfall gets large enough, the player resigns.

Installation

System Requirement

- Hardware (suggested):

  P4 1.5G CPU, 256M memory, at least 10M available hard disk space. 32M Video memory. 3D graphic accelerate card. Sound card and speaker.

- Software:

  Windows 9x, 2000 (not tested on XP and NT). Glut32.dll (included with game)

Install/Uninstall:

- Install: Double click Snooker_Setup.exe, follow the installation wizard, select the target folder. The installation wizard with create an icon on desktop and a shortcut in the Start->Programs menu system.

- Uninstall: click Start on the taskbar, click Programs, go to Snooker Simulator (3D), and click Uninstall Snooker Simulator. All installed files will be removed from the machine.

Play game

Start the Snooker Simulator

- Double click the Snooker Simulator (3D) icon on the desktop; or
- Click Start on the taskbar, click Programs, go to Snooker Simulator (3D), click 3D Snooker Simulator

Introduction to Scenes:

1. Main menu page

After start the Snooker Simulator, the Main Menu Page is the first scene your will meet.

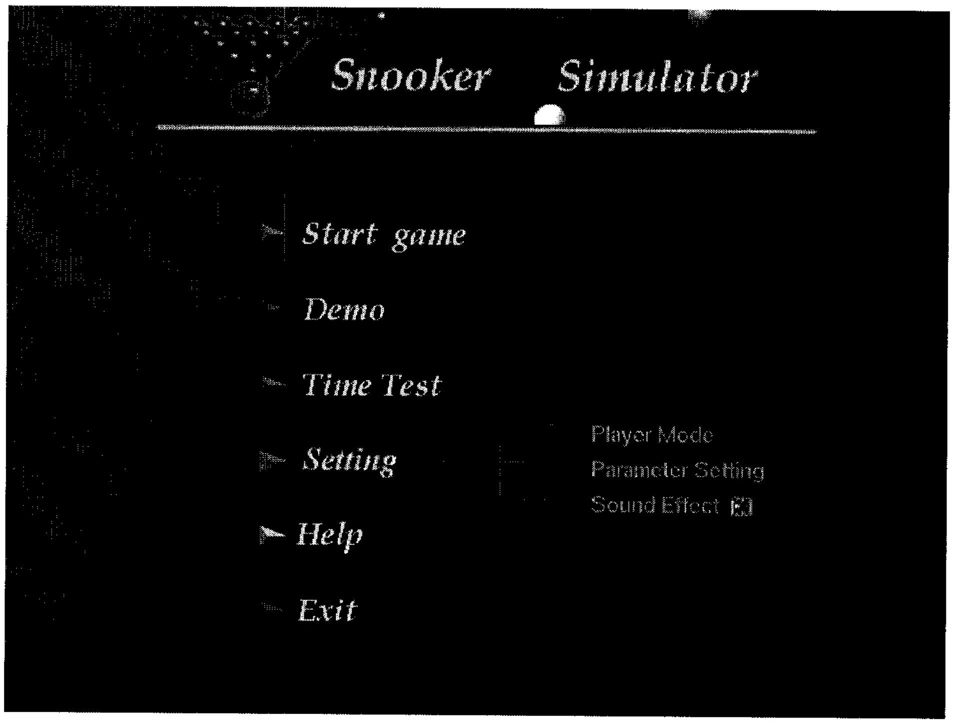


Figure A.1 Main menu page

In this page, the player can:

- Start a game. Click Start game. The default play mode is manual for all players
- Demo. Click Demo. The compute will keep playing game automatically till the player stop it
- Time testing. Click Time test. The computer will simulate a full-speed hit and calculate the best Time increment according to the computer's performance
- Setting. Click Setting. Three sub-functions supplied.
  - Player Mode. Change players' properties
  - Parameter Setting. Change environment parameters
  - Sound Effect. Toggle to turn on/off sound effect
- Leave the game, click Exit
- Read helps, click Help, then click on the help topic

2. GAME PAGE

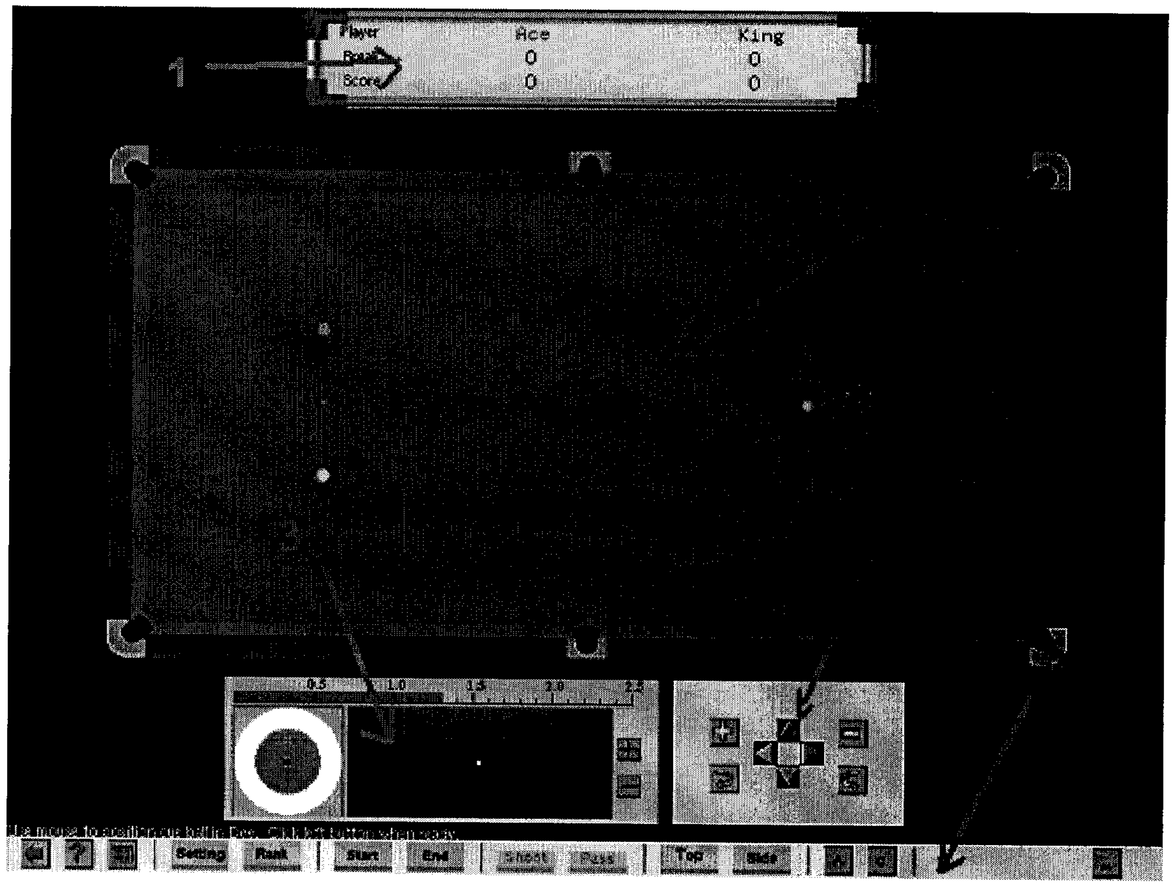


Figure A.2 Game page

The Game page consists of 5 parts:

- **Scoreboard**: show players current score and statistics on all the frames;
- **Snooker table**: the area where the players actually play the game
- **Adjust Window**: set cue speed, where the cue hits on the cue ball and adjust the hitting position between cue ball and target ball more precisely
- **Control panel**: switch between different views, adjust the camera's position and focal distance
- **Taskbar**: list all the functions available for the game

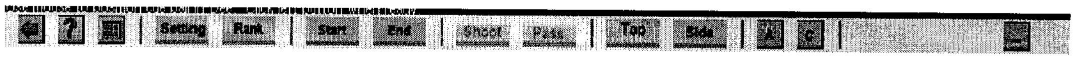


Figure A.3 Taskbar

From left to right, the buttons are for:

- Finish game and go back to main menu
- Show help
- Toggle the Scoreboard between detail report and simple report
- Show setting panel
- Show rank of auto players
- Start a game
- Stop the current game
- Shoot
- Pass
- Switch to Top view of the snooker table

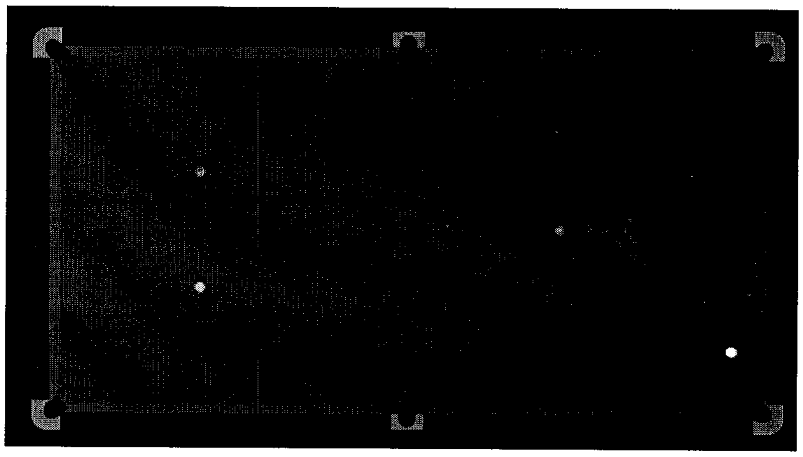


Figure A.4 Snooker table at Top View

- Switch to Side view of the snooker table

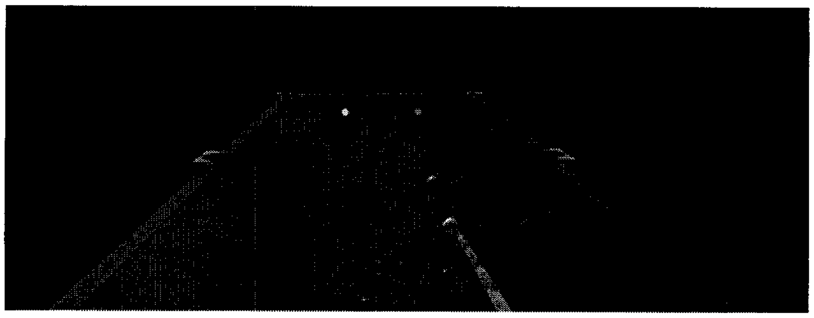


Figure A.5 Snooker table at Side View

- Hide/Show the adjustment window
- Hide/Show the control panel
- Minimize the game

Playing a Shot

To play a shot, you first have to set the "controls". There are four controls to set: they determine the coarse direction, fine direction, speed, and spin of the cue ball.

- **Coarse Direction.** To set the direction, move the mouse cursor over the playing-area of the table. (The cursor is a small circle while it is within the playing-area.) Choose a point through which you want the cue ball to pass, and click the left mouse button.

- **Fine Direction.** Move the mouse cursor into the 3D view (fine adjustment) at the bottom right of the Adjustment Window. Clicking the left mouse button in this area will make small changes to the direction of the cue ball. The amount by which the direction changes depends on how far the mouse cursor is from the center of the 3D view: if it is close to the center, the change will be very small. If the mouse cursor is left of center, the direction changes in the counterclockwise direction; if the mouse cursor is right of center, it changes in the clockwise direction.

- **Speed.** Move the mouse cursor into the calibrated scale at the top of Adjustment Window. The real speed for your current mouse position will display. Click on the position with your ideal speed to set the cue speed. The red, thermometer-like bar indicates the speed you have chosen. The maximum speed is 2.5 meters per second.

- **Spin.** Move the mouse cursor into the white and green circular area at the bottom left of the Adjustment Window. The small spot in the circle will move to the cursor position. This spot corresponds to the point where the cue will strike the cue ball. If it is near the top of the green area, you are applying top spin ("follow"); a point near the bottom corresponds to back spin ("draw"); and points to the left and right sides correspond to left and right side-spin ("english") respectively. You cannot move the impact point outside the green circle because impacts near the edge of the ball are likely to cause a miscue.

If the player is in manual mode, there will be no visible response to changing a control except that the 3D view will change in response to a change of direction. In tracking and advise modes, Snooker will display the paths that the balls will take each time you change a control.

When you have adjusted the controls to your satisfaction, move the mouse cursor to `Shoot` button on the taskbar and click it. The `Shoot` button is highlighted whenever the current player is allowed to play a shot.

If you reach the adjust limitation, no action but shooting will take effect.

Question and Answer

1. What is DEMO

Snooker will play until you stop it. Each frame will be played by a randomly-chosen pair of automatic players. To stop the demonstration, click the END on the taskbar or press the ESC key, the demo will stop until the balls come to rest at the end of a shot.

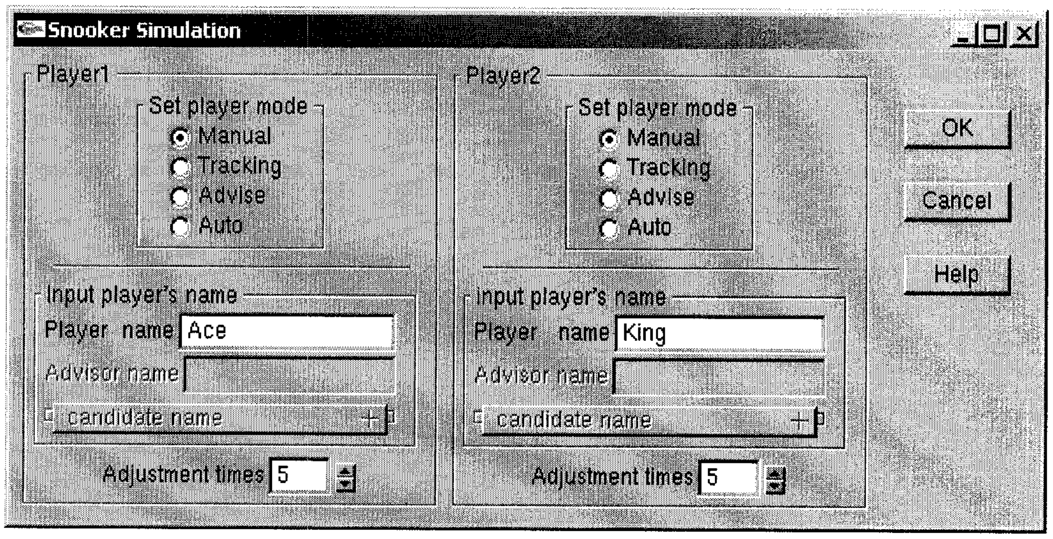
2. What is Time Testing

The program will perform a full-speed hitting to find the best value for the parameter `Time Increment'. The players can take this suggested value by click "Yes" when a message is prompted.

3. How to Set Players' mode

Click Setting->Player Mode on Main Menu Page, or click Setting on the taskbar.

A Player mode setting window will be shown as:



Player1
Set player mode: [✓ Manual] Tracking Advise Auto
Input player's name
Player name: Ace
Advisor name:
candidate name
Adjustment times: 5

Player2
Set player mode: [✓ Manual] Tracking Advise Auto
Input player's name
Player name: King
Advisor name:
candidate name
Adjustment times: 5

Figure A.6 Player mode set window

In this window, you can set properties for each player. You can:

➢ Change player mode:

For Manual and tracking mode, you have to input the player's name

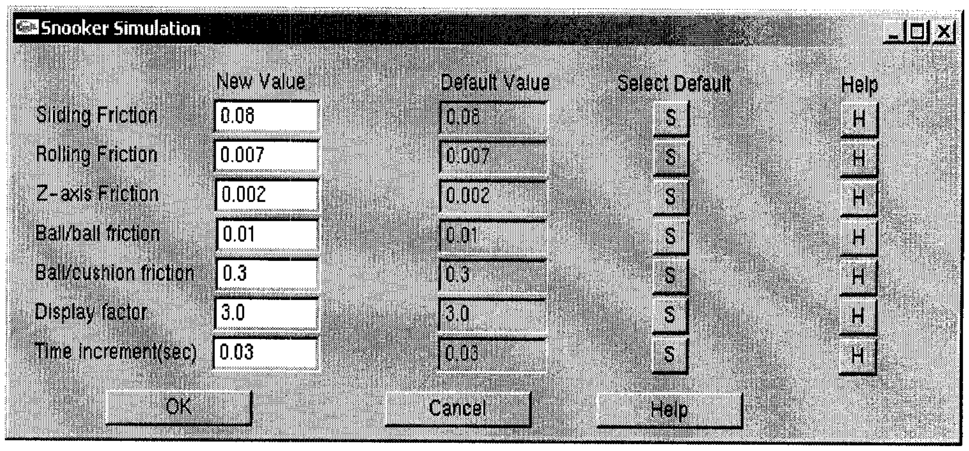
For Advise and auto mode, candidates and their performance will be shown, select one person, the name will input in the proper field automatically.

➢ Change adjustment times: maximum times allowed to adjust before shooting

Click OK to save the modifications, click cancel to cancel the modifications. For more information click help.

4. How to set the Environment Parameters

Click Setting->Parameter Setting on Main Menu Page, or click Setting on the taskbar. A Player mode setting window will be shown as:



	New Value	Default Value	Select Default	Help
Sliding Friction	0.08	0.08	S	H
Rolling Friction	0.007	0.007	S	H
Z-axis Friction	0.002	0.002	S	H
Ball/ball friction	0.01	0.01	S	H
Ball/cushion friction	0.3	0.3	S	H
Display factor	3.0	3.0	S	H
Time increment(sec)	0.03	0.03	S	H

Figure A.8 Environment parameters set window

In this window, you're allowed to change values for Sliding Friction, Rolling Friction, Z-axis Friction, Ball/ball friction, Ball/cushion friction, Display factor and Time increment.

Change value: click the proper white edit box, input new value

Reset the value to default: click the S button for the target parameter

For the help on this parameter, click the H button for the target parameter

Click OK to save the settings, click Cancel to reserve the old setting.

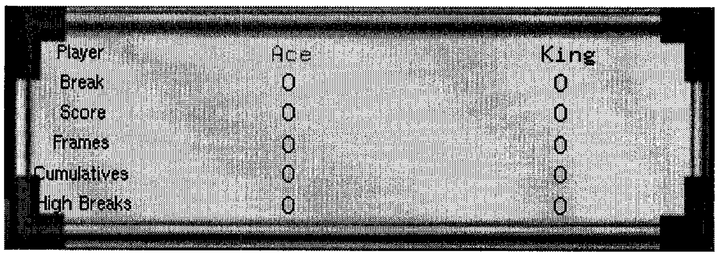
For more information, click Help

See details on Q&A14 – How about Environment Parameters

5. What's the difference between Detail report and Simple report?

Detail report contains:

Players' name, current player, break, Score, Frames, Cumulative and High breaks

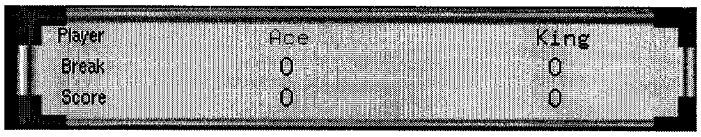


Player	Ace	King
Break	0	0
Score	0	0
Frames	0	0
Cumulatives	0	0
High Breaks	0	0

Figure A.9 Detail Report

Simple report contains:

Players' name, current player, break, Score



Player	Ace	King
Break	0	0
Score	0	0

Figure A.10 Simple Report

How to switch:

- Click on the Scoreboard, or
- Click the button on the taskbar.

6. What is the Tracking Mode

Tracking mode will show the moving track of balls when you select a target ball. This will help the player preview the result of his shoot.

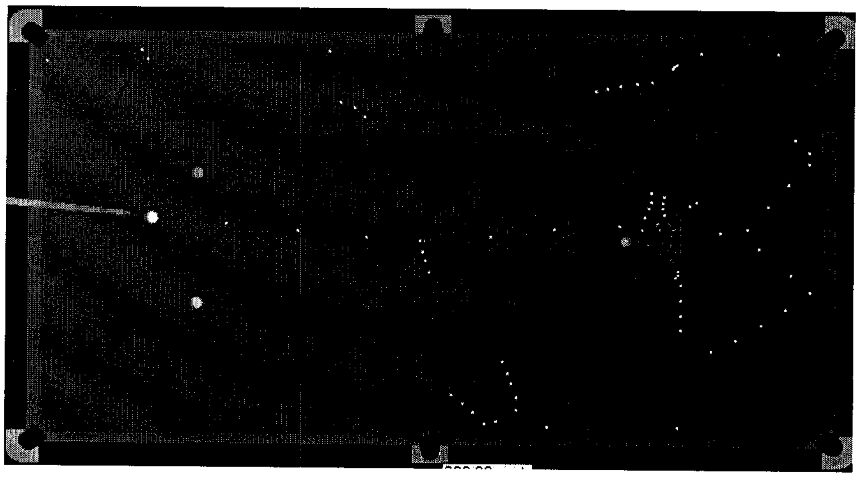


Figure A.11 Tracking mode

7. What is the Advisor Mode

Advisor mode is still played by you, but the your selected advisor will give you suggestion before you shoot (show movement tracking as Tracking mode), you can adjust speed or direction on the suggestion shoot or ignore the suggestions.

8. Can I leave the game temporarily without closing it?

Yes. In the Game page, click the rightmost Minimize button on the taskbar will make you go back to desktop. Click the icon on the system taskbar will bring you back to the game.

9. How to close the game

If you are in Main Menu Page, click Exit or press Escape

If you are in Game Page, click leftmost button on the taskbar or press Escape will bring you back to Main Menu Page, then click Exit or press Escape again.

10. How to adjust in Adjustment Window

This window allows you to adjust cue speed, hitting position of cue on cue ball, hitting position between cue ball and target.

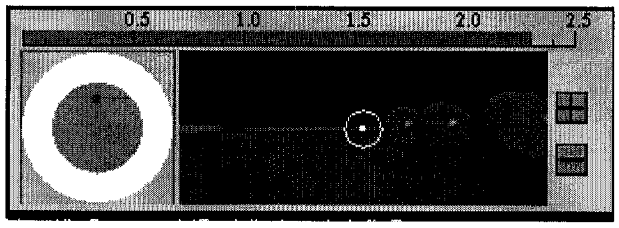


Figure A.12 Fine-Adjust window

To set cue speed: move the mouse about the scaled bar on the top, the real speed will be shown, when get the ideal speed, click the mouse on the spot.

To set the hitting position of cue on cue ball (direction): move mouse to the green area of the white ball on the left, when cursor changed to cross, you're in the valid area, click on the ideal position.

To adjust the hit position between cue ball and target ball: The white circle is the mapping of cue ball, click on the right window will move the circle a bit to left or right, depend on the distance between mouse and circle (the larger the distance is, the more the circle will move). If the ball is too small to too big, click the + and – button on the right.

Attention: During adjusting, if new ball move between the cue ball and target ball, it will not show in the adjustment window. It's suggested to observe the snooker table at the same time to ensure no other ball will block the cue ball.

11. What is Adjustment Limit

In manual modes (that is, **Manual**, **Tracking**, and **Advise**), you can set the number of adjustments that can be made for each shot in the `Player Mode setting window`. If the Adjustments field is set to a small number, such as 5 or 10, the player can make only a limited number of changes before playing the shot. Two players with different skill levels can used the Adjustments field as the basis for handicapping.

12. What is Automatic Players

You can adjust the level of skill and style of play of each automatic player by editing the text file `players.dat` (see the section "**Editing the Data Files**" below). The program maintains rankings for the players. You can see the current rankings by clicking on `Rank` Button on the taskbar.

13. How to edit the Data Files

The folder in which you have stored The Snooker Simulation contains two data files: "`players.dat`" containing information about the automatic players and "`settings.dat`" containing information about the table parameters. By editing these files you can customize Snooker and avoid wasting time changing settings when the program is running.

Both files are in "plain ASCII" format. This means that you should edit them with an ASCII text editor, such as Microsoft Notepad. If you edit them with a word processor, such as Microsoft Word or Corel WordPerfect, you should save them in "ASCII" or "text" mode after making the changes.

The file "`players.dat`" should look something like this:

```
#comment            ACC LON STR PRE POS DEF HI  CLU GTL PER
Hardy Potts         10  10  5   10  10  10  10  10  10  10
Careful Kate        7   7   10  10  10  10  6   3   9   10
Sharky Pete         10  10  2   8   10  10  2   2   5   10
Gofer Black         7   6   9   6   9   6   10  5   8   8
Artful Ann          8   4   7   8   5   2   5   10  4   7
Willy Wacker        4   9   9   2   2   2   5   10  0   4
Duffer Dave         2   2   8   1   1   1   8   3   5   1
#end
```

Snooker ignores lines beginning with "#". Each of the other lines consists of the name of a player followed by exactly ten numbers. The numbers must be integers (no fractions or decimals), not less than zero, and not greater than 10. The numbers determine the style of the automatic player. They are answers to the following questions, with 0 indicating "definitely not" and 10 indicating "definitely yes".

1. I am an accurate potter.
2. I am good at long pots.
3. I prefer straight pots to cuts.
4. I prepare each shot very carefully.
5. I am good at positional play.
6. I am good at defensive play.
7. I go for balls with high scoring value.
8. I break up clusters of balls whenever possible.
9. I prefer gentle shots to strong shots.
10. I give up only when I have no chance of winning.

You can change the values of the existing players or you can create new players of your own.

The file "settings.dat" contains information that describes properties of the snooker table. Each line consists of a number followed by an explanation in English. Only the number is important; Snooker ignores the explanation. The numbers are all fractions less than 1, except the display factor, which is an integer greater than 1. Snooker does not check to see if the numbers in the file are reasonable. If you enter unlikely values, Snooker may behave — or misbehave — in curious ways. For example, if rolling friction is zero, the balls will roll for a very long time. If ball/cushion friction is also zero, they will roll forever! The practical problem with zero friction is that the program will try to construct infinitely long trajectories for the balls and will crash.

Here is the file `settings.dat` with default value.

```
0.08    Sliding friction
0.007   Rolling friction
0.002   Spin friction
0.01    Ball/ball friction
0.3     Ball/cushion friction
3       Display factor
0.03    Time increment (seconds)
```

The data in this file is used to set default value, during the game, you can change all the settings in Parameter Setting window. But the new setting will not saved to this file.

14. How about Environment Parameters

The Snooker Simulation allows you to change the properties of the table you are playing on and to adjust the speed of the game to your computer. To change the properties, select `Settings` from the menu bar and then `Parameters`. Snooker will open a dialog box that you can use to read and change various values.

♦ **Sliding Friction.** This is the coefficient of friction between a sliding ball and the cloth on the table. A ball slides for a short time after being struck by the cue or another ball. The effect of friction is particularly important for the cue ball: a high coefficient of friction makes back spin ("draw") and top spin ("follow") more effective.

♦ **Rolling Friction.** After sliding for a while, a ball begins to roll. Its speed is affect by friction, but much less than when it is sliding. Rolling friction is typically about one-tenth the value of sliding friction.

♦ **Spin Friction.** Spin friction is the friction experienced by a ball spinning about a vertical axis. (Spin friction is also called "Z-axis friction" because the Z-axis is the vertical axis.) The vertical spin affects the behavior of a ball when it hits a cushion and, to a lesser extent, when it hits another ball. If spin friction is high, the ball will lose its vertical spin as it travels and will not be deflected by spin when it hits a cushion. If spin friction is low, the ball will retain its vertical spin and be deflected even at a distant cushion.

♦ **Ball-Ball Friction.** The ball-ball friction is the friction experienced by balls that are in contact during a collision. It is small and, ideally, would be zero. After a collision between frictionless balls, one ball travels along the line of centers at the time of collision and the other travels at right angles to this direction. If there is ball-ball friction, these directions are altered slightly. The automatic players in Snooker occasionally miss long pots: this is because they do not allow for ball-ball friction. If you set ball-ball friction to zero, you will find that the automatic players do not miss pots as often as they do when the default value is used.

♦ **Ball-Cushion Friction.** This is the friction experienced by a ball when it is sliding against a cushion. If the ball rolls directly into a cushion, there is very little sliding, and ball-cushion friction has little effect. If, however, the ball hits the cushion obliquely, or is spinning about a vertical axis, the ball-cushion friction will affects its motion after the impact. If ball-cushion friction is small (less than 0.1), the cushions will be "fast" (balls will not lose much speed) but side-spin will have little effect. If ball-cushion friction is large (0.4 or more), the cushions are slower but side-spin is effective. When ball-cushion friction is set to 0.3 (the default value), the effects are quite close to what you would see on a real snooker table.

♦ **Display Factor.** The display factor determines how often the screen is refreshed while the balls are in motion.

♦ **Time Increment.** The time increment determines how often Snooker computes the positions of the balls.

Suppose that the display factor is set to 3 and the time increment is set to 0.01. This means that:

♦ The positions of the balls will be computed every 0.01 seconds or, in other words, 100 times each second.

♦ The screen will be refreshed at every third computation: in this case, 100/3 = 33 times per second.

The accuracy of the simulation depends on the time increment: the smaller the value, the more accurate the simulation. But, if the value is too small, the simulation will be unacceptably slow.

The speed of the balls on the screen, and the jerkiness of their motion, depends on the display factor. If the display factor is 1, the motion will be smooth but slow. If the value is large, say 10, the balls will move ten times faster, but in jerks.

You should try to choose settings of the display factor and time increment that give smooth motion that is fast enough. Snooker provides a "timing test" to help you. Use it as follows:

- Start Snooker or, if it is running, end the current frame.
- Select `Settings` from the menu bar and then `Timing Test`. When the message window appears, click on `OK`.
- Snooker will play a "lag shot" which sends the cue ball up and down the table. It will compute both the simulated time and the actual time required for this shot, and will report the value of the time increment that would make them equal.
- If you click on `OK` in the final message box, Snooker will set the value of the timing increment to the value that it has computed.

Although the value of the timing increment is now correct, you may still want to change the display factor. Here is a rule of thumb that will help you: find the smallest value of the display factor for which the timing test gives a value of 0.01 or less to the timing increment. When you have found the best value for the display factor, run the timing test again to get the appropriate value of the time increment.

If you want to save the timer settings permanently, so you do not have to change them every time you pay, edit the file "`settings.dat`", as described in Q&A13.