**Chapter 1**

**INTRODUCTION**

**1.1 What is OpenGL?**

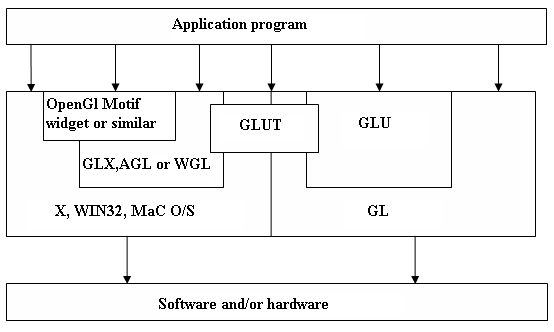
OpenGL provides a set of commands to render a three dimensional scene. That means you provide the data in an OpenGL-useable form and OpenGL will show this data on the screen (render it). It is developed by many companies and it is free to use. You can develop OpenGL-applications without licensing.

OpenGL is a hardware- and system-independent interface. An OpenGL-application will work on every platform, as long as there is an installed implementation.

Because it is system independent, there are no functions to create windows etc., but there are helper functions for each platform. A very useful thing is GLUT.

Most of our applications can be accessed in Open GL using three libraries:-

* GL – (or OpenGL in windows), has names that begin with GL and are stored in a library.
* GLU – (or OpenGL Utility Library), uses only GL functions but contains code for creating common objects and viewing.
* GLUT – (or OpenGL Utility Toolkit), provides the minimum functionality that is expected in modern windowing system.



**Figure 1.1** Block Diagram of OpenGL utility toolkit.

**1.2** **What is GLUT?**

GLUT is a complete API written by Mark Kilgard which lets you create windows and handle the messages. It exists for several platforms, that means that a program which uses GLUT can be compiled on many platforms without (or at least with very few) changes in the code.

**1.3** **How does OpenGL work?**

OpenGL bases on the state variables. There are many values, for example the color, that remain after being specified. That means, you can specify a color once and draw several polygons, lines or whatever with this color then. There are no classes like in DirectX. However, it is logically structured. Before we come to the commands themselves, here is another thing:

To be hardware independent, OpenGL provides its own data types. They all begin with "GL". For example GLfloat, GLint and so on. There are also many symbolic constants, they all begin with "GL\_", like GL\_POINTS, GL\_POLYGON. Finally the commands have the prefix "gl" like glVertex3f(). There is a utility library called GLU, here the prefixes are "GLU\_" and "glu". GLUT commands begin with "glut", it is the same for every library.

A very important thing is to know, that there are two important matrices, which affect the transformation from the 3d-world to the 2d-screen: The projection matrix and the model view matrix. The projection matrix contains information, how a vertex – let's say a "point" in space – shall be mapped to the screen. This contains, whether the projection shall be isometric or from a perspective, how wide the field of view is and so on.

**1.4** **How can I use GLUT?**

GLUT provides some routines for the initialization and creating the window (or full screen mode, if you want to). Those functions are called first in a GLUT application:

In your first line you always write glutInit(&argc, argv). After this, you must tell GLUT, which display mode you want – single or double buffering, color index mode or RGB and so on. This is done by calling glutInitDisplayMode(). The symbolic constants are connected by a logical OR, so you could use glutInitDisplayMode(GLUT\_RGB | GLUT\_SINGLE). In later tutorials we will use some more constants here.

After the initialization you call glCreateWindow() with the window name as parameter.

Then you can (and should) pass some methods for certain events. The most important ones are "reshape" and "display". In reshape you need to (re)define the field of view and specify a new area in the window, where OpenGL is allowed to draw to.

Display should clear the so called color buffer – let's say this is the sheet of paper – and draw our objects.

You pass the methods by glut\*Func(), for example glutDisplayFunc(). At the end of the main function you call glutMainLoop(). This function doesn't return, but calls the several functions passed by glut\*Func.

**Chapter 2**

**SYSTEM REQUIREMENTS**

**2.1 Hardware Requirements**

Minimum hardware specification

* Processor : Intel Core processor.
* Main memory : 128 MB RAM, 256 MB recommended.
* Hard Disk : 110 MB of hard disk space required
* Display : 800 x 600 or higher-resolution display with 256 colors.

**2.2 Software Requirements**

Minimum software specification

* Operating system :Windows XP/ Windows 7
* Visual Studio 2005
* OpenGL Library

**Chapter 3**

**SYSTEM DESIGN**

The most important aspect of the project is its design phase . This chapter gives the overall design of the project . Initially when we execute , the output window will be displayed . The sequence of operation can take place based on the user's input . The flow of execution is depicted in the following block diagram .

**Purpose and Scope**

This section provides a brief description of the Systems Design Document’s purpose and scope.

**Project Executive Summary**

This section provides a description of the project from a management perspective and an overview of the framework within which the conceptual system design was prepared. If appropriate, include the information discussed in the subsequent sections in the summary.

**System Overview**

This section describes the system in narrative form using non-technical terms. It should provide a high-level system architecture diagram showing a subsystem breakout of the system, if applicable. The high-level system architecture or subsystem diagrams should, if applicable, show interfaces to external systems. Supply a high-level context diagram for the system and subsystems, if applicable. Refer to the requirements trace ability matrix (RTM) in the Functional Requirements Document (FRD), to identify the allocation of the functional requirements into this design document.

**Design Constraints**

This section describes any constraints in the system design (reference any trade-off analyses conducted such, as resource use versus productivity, or conflicts with other systems) and includes any assumptions made by the project team in developing the system design.

**Future Contingencies**

This section describes any contingencies that might arise in the design of the system that may change the development direction. Possibilities include lack of interface agreements with outside agencies or unstable architectures at the time this document is produced. Address any possible workarounds or alternative plans.

**Document Organization**

This section describes the organization of the Systems Design Document.

**Shadow Cube**

**Figure 3.1** Design of the program.

**L/l**

**F/f**

**1**

**2**

**3**

**Esc**

**Mouse interaction**

**Keyboard interaction**

**Option Menu**

**User Interface**

**Start Motion**

**Toggle Fog**

**Stop Motion**

**Toggle Lightening**

**Fog Type**

**Chapter 4**

**IMPLEMENTATION**

**4.1FUNCTIONS**

**4.1.1 glLightfv Function**

The glLightfv function returns light source parameter values.

**SYNTAX**

void glLightfv(GLenum light, GLenum pname, GLfloat \*params);

**PARAMETERS**

* **light**

The identifier of a light. The number of possible lights depends on the implementation, but at least eight lights are supported. They are identified by symbolic names of the form GL\_LIGHT where i is a value: 0 to GL\_MAX\_LIGHTS - 1.

* **Pname**

A light source parameter for light. The following symbolic names are accepted:

**GL\_DIFFUSE** **:** The parameter contains four integer or floating-point values that specify the diffuse RGBA intensity of the light. Integer values are mapped linearly such that the most positive representable value maps to 1.0, and the most negative representable value maps to -1.0. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The default diffuse intensity is (0.0, 0.0, 0.0, 1.0) for all lights other than light zero. The default diffuse intensity of light zero is (1.0, 1.0, 1.0, 1.0).

**GL\_SPECULAR** **:** The parameter contains four integer or floating-point values that specify the specular RGBA intensity of the light. Integer values are mapped linearly such that the most positive representable value maps to 1.0, and the most negative representable value maps to 1.0. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The default specular intensity is (0.0, 0.0, 0.0, 1.0) for all lights other than light zero. The default specular intensity of light zero is (1.0, 1.0, 1.0, 1.0).

**glLightfv(GL\_LIGHT1,GL\_POSITION,pos);**

**4.1.2 glPushMatrix, glPopMatrix Function**

The glPushMatrix and glPopMatrix functions push and pop the current matrix stack.

**SYNTAX**

void glPushMatrix, glPopMatrix(void);

**PARAMETERS**

This function has no parameters.

* glPushMatrix();
* glPopMatrix();

**4.1.3 glEnable, glDisable Function**

The glEnable and glDisable functions enable or disable OpenGL capabilities.

**SYNTAX**

void glEnable, glDisable(GLenum cap);

**PARAMETERS**

A symbolic constant indicating an OpenGL capability.

* glEnable(GL\_CULL\_FACE);
* glDisable(GL\_CULL\_FACE);

**4.1.4 glColor3f Function**

Sets the current color.

**SYNTAX**

void glColor3f(GLfloat red, GLfloat green, GLfloat blue);

**PARAMETERS**

* + red

The new red value for the current color.

* + green

The new green value for the current color.

**4.1.5 glBegin, glEnd Function**

The glBegin and glEnd functions delimit the vertices of a primitive or a group of like primitives.

**SYNTAX**

void glBegin, glEnd(GLenum mode);

**PARAMETERS**

* + mode: The primitive or primitives that will be created from vertices presented between glBegin and the subsequent glEnd. The following are accepted symbolic constants and their meanings:

|  |  |
| --- | --- |
| GL\_LINES | Treats each pair of vertices as an independent line segment.  Vertices 2n - 1 and 2n define line n. N/2 lines are drawn. |
| GL\_LINE\_STRIP | Draws a connected group of line segments from the first vertex to the last. Vertices n and n+1 define line n. N - 1 lines are drawn. |
| GL\_LINE\_LOOP | Draws a connected group of line segments from the first vertex to the last, then back to the first. Vertices n and n + 1 define line n. The last line, however, is defined by vertices N and N lines are drawn. |
| GL\_TRIANGLES | Treats each triplet of vertices as an independent triangle. Vertices 3n - 2, 3n - 1, and 3n define triangle n. N/3 triangle are drawn. |
| GL\_QUADS | Treats each group of four vertices as an independent quadrilateral. Vertices 4n - 3, 4n - 2, 4n - 1, and 4n defined quadrilateral n. N/4 quadrilaterals are drawn. |

**4.1.6 glutSwapBuffers**

glutSwapBuffers swaps the buffers of the current window if double buffered.

**SYNTAX**

void glutSwapBuffers(void);

glutSwapBuffers promotes the contents of the back buffer of the layer in use of the current window to become the contents of the front buffer. The contents of the back buffer

then become undefined. An implicit glFlush is done by glutSwapBuffers before it returns.

**4.1.7 glVertex3f Function**

Specifies a vertex.

**SYNTAX**

void glVertex3f(GLfloat x, GLfloat y, GLfloat z);

**PARAMETERS**

|  |  |
| --- | --- |
| X | Specifies the x-coordinate of a vertex. |
| Y | Specifies the y-coordinate of a vertex. |
| Z | Specifies the z-coordinate of a vertex. |

glVertex3f(-3.0,3.0,4.0);

**4.1.8 glRotatef Function**

The glRotate and glRotatef functions multiply the current matrix by a rotation matrix.

**SYNTAX**

void glRotate( GLfloat angle, GLfloat x, GLfloat y, GLfloat z);

**PARAMETERS**

|  |  |
| --- | --- |
| Angle | The angle of rotation, in degrees. |
| X | The x coordinate of a vector. |
| Y | The y coordinate of a vector |
| Z | The z coordinate of a vector. |

glRotatef(xrot,1.0,0.0,0.0);

**4.1.9 glTranslate Function**

The glTranslate and glTranslatef functions multiply the current matrix by a translation matrix.

**SYNTAX**

void glTranslate( x, y, z);

**PARAMETERS**

* + x, y, z

The x, y, and z coordinates of a translation vector.

* glTranslatef(0.0,0.0,-1.0);

**4.1.10 glScalef Function**

The glScaled and glScalef functions multiply the current matrix by a general scaling matrix.

**SYNTAX**

void glScalef(GLfloat x, GLfloat y, GLfloat z);

**PARAMETERS**

|  |  |
| --- | --- |
| X | Scale factors along the x axis. |
| Y | Scale factors along the y axis. |
| Z | Scale factors along the z axis. |

glScalef(1.0,1.0,-1.0);

**4.1.11 glClear Function**

The glClear function clears buffers to preset values.

**SYNTAX**

glClear(GLbitfield mask);

**PARAMETERS**

* mask

Bitwise OR operators of masks that indicate the buffers to be cleared. The four masks are as follows.

|  |  |
| --- | --- |
| Value | Meaning |
| GL\_COLOR\_BUFFER\_BIT | The buffers currently enabled for color writing. |
| GL\_DEPTH\_BUFFER\_BIT | The depth buffer. |

glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT);

**4.1.12 glMatrixMode Function**

The glMatrixMode function specifies which matrix is the current matrix.

**SYNTAX**

void glMatrixMode(GLenum mode);

**PARAMETERS**

* mode

The matrix stack that is the target for subsequent matrix operations. The mode parameter can assume one of three values:

|  |  |
| --- | --- |
| Value | Meaning |
| GL\_MODELVIEW | Applies subsequent matrix operations to the model view matrix stack. |

glMatrixMode(GL\_MODELVIEW);

**4.1.13 glLoadIdentity Function**

The glLoadIdentity function replaces the current matrix with the identity matrix.

**SYNTAX**

void glLoadIdentity(void);

**PARAMETERS**

This function has no parameters.

* + glLoadIdentity();

**4.1.14 glViewport Function**

The glViewport function sets the viewport.

**SYNTAX**

void glViewport(x, y,width, height);

**PARAMETERS**

|  |  |
| --- | --- |
| x, y | The lower-left corner of the viewport rectangle, in pixels. The default is (0,0). |
| width, height | The width and height, respectively, of the viewport. When an OpenGL context is first attached to a window, width and height are set to the display. |

glViewport(0,0,w,h);

glMatrixMode Function

glLoadIdentity Function

**4.1.15 gluPerspective Function**

set up a perspective projection matrix

**SYNTAX**

void gluPerspective( GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar );

**PARAMETERS**

|  |  |
| --- | --- |
| Fovy | Specifies the field of view angle, in degrees, in the y direction. |
| Aspect | Specifies the aspect ratio that determines the field of view in the x direction. The aspect ratio is the ratio of x (width) to y (height). |
| ZNear | Specifies the distance from the viewer to the near clipping plane (always positive). |
| zFar | Specifies the distance from the viewer to the far clipping plane (always positive). |

gluPerspective(50.0,(float)w/(float)h,1.0,20.0);

**STATIC INIT FUNCTION** glEnable, glDisable Function  
 glLightfv Function

**4.1.16 glMaterialfv Function**  
 The glMaterialfv function specifies material parameters for the lighting model. **SYNTAX** void glMaterialfv(GLenum face, GLenum pname, const GLfloat params);  
**PARAMETERS**

Face:The face or faces that are being updated. Must be one of the following: GL\_FRONT, GL\_BACK, or GL\_FRONT and GL\_BACK.

pname: The material parameter of the face or faces being updated. The parameters that can be specified using glMaterialfv, and their interpretations by the lighting equation, are as follows.

GL\_SPECULAR:The params parameter contains four integer or floating-point values that specify the specular RGBA reflectance of the material. Integer values are mapped linearly such that the most positive representable value maps to 1.0, and the most negative representable value maps to -1.0. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped.   
**4.1.17 glutDisplayFunc Function** glutDisplayFunc sets the display callback for the current window. **SYNTAX**

void glutDisplayFunc(void (\*func)(void));  
**PARAMETERS**

Func: The new display callback function.   
 glutDisplayFunc(display);  
**4.1.18 glutAddMenuEntry**  glutAddMenuEntry adds a menu entry to the bottom of the current menu.

**SYNTAX** void glutAddMenuEntry(char \*name, int value);  
 Name: ASCII character string to display in the menu entry.  
 Value: Value to return to the menu’s callback function if the menu entry is selected.

**4.1.19 glutAttachMenu** glutAttachMenu attaches a mouse button for the current window to the identifier of the current menu;  
**SYNTAX**

void glutAttachMenu(int button);  
 Button: The button to attach a menu or detach a menu.

**4.1.20 glDepthMask**

It specifies the function used to compare each incoming pixel value with value present in depth buffer. The comparison is performed only when depth testing is enabled.

**SYNTAX**

void glDepthMask(GLboolean flag);

**4.1.21 glFrustum**

It describes a perspective matrix that produces a perspective projection.

**SYNTAX**

void glFrustum(GLdouble left,GLdouble right,GLdouble bottom,GLdouble top,GLdouble nearval,GLdouble farval);

**4.1.22 glCullFace**

It specifies whether front or back facing facets are culled (as specified by model) when facet culling is enabled.

**SYNTAX**

void glCullFace(GLenum mode);

**4.1.23 glShadeModel**

It specifies the symbolic value representing a shading technique. Accepted values are GL\_FLAT and GL\_SMOOTH. The initial value is GL\_SMOOTH.

**SYNTAX**

void glShadeModel(GLenum mode);

**4.1.24 glBlendFunc**

In RGBA mode , pixels can be drawn using a func that blends the incoming(source) RGBA values with the RGBA values that are already in the frame buffer(destination).

**SYNTAX**

void glBlendFunc(GLenum sfactor,GLenum dfactor);

**4.1.25 glPolygonStipple**

Polygon stippling like line stippling makes out certain fragments produced by rasterization, creating a pattern . Stippling is independent of polygon anitialising.

**SYNTAX**

void glPolygonStipple(const GLubyte\* pattern);

**4.1.26 glClearIndex**

It specifies the index used by glclear to clear the color index.

**SYNTAX**

void glClearIndex(GLfloat c);

**4.1.27 glClearDepth**

It specifies the depth value used by glclear to clear the depth buffer.Values specified by glClearDepth are clamped to the range 0 1.

**SYNTAX**

void glClearDepth(GLclampd depth);

**4.1.28 glMultMatrix**

It multiplies the current matrix with the ones specified using m,and replaces the current matrix with the product.

**SYNTAX**

void glMultMatrixf(const GLfloat\* m);

**4.1.29 glFogfv**

The glFogfv function specifies fog parameters.

**SYNTAX**

void glFogfv(GLenum pname,const GLfloat\* params);

**4.1.30 glCallList**

The glCallList function executes the display list.

**SYNTAX**

void glCallList(GLuint list);

**4.1.31 glEndList**

It contains the GL commands that have been stored for subsequent execution.

**SYNTAX**

void glEndList(void);

**4.1.32 glNormal**

The current normal is set to the given coordinates whenever glNormal is issued.

**SYNTAX**

void glNormal3f(GLfloat nx,GLfloat ny,GLfloat nz);

**4.2 MAIN FUNCTIONS**

**4.2.1 glutInit Function**  
 glutInit is used to initialize the GLUT library.  
**SYNTAX**

glutInit(int \*argcp, char \*\*argv);   
**PARAMETERS**  
 argcp:A pointer to the program's unmodified argc variable from main.

Argv:The program's unmodified argv variable from main  
 glutInit(&argc,argv);

**4.2.2 glutInitDisplayMode Function** glutInitDisplayMode sets the initial display mode.  
**SYNTAX**

void glutInitDisplayMode(unsigned int mode);  
**PARAMETERS**

Mode: Display mode, normally the bitwise OR of GLUT display mode bit masks. See values below:   
GLUT\_RGB An alias for GLUT\_RGBA.GLUT\_DOUBLE Bit mask to select a double buffered window. This overrides GLUT\_SINGLE . If it is also specified. GLUT\_DEPTH Bit mask to select a window with a depth buffer.   
glutInitDisplayMode(GLUT\_RGB|GLUT\_DEPTH|GLUT\_DOUBLE);   
**4.2.3 glutInitWindowPosition, glutInitWindowSize Functions**  
 glutInitWindowPosition and glutInitWindowSize set the initial window position and size respectively.  
**SYNTAX** void glutInitWindowSize(int width, int height);  
 void glutInitWindowPosition(int x, int y);  
**PARAMETERS**  
 Width in pixels. HeightHeight in pixels. XWindow X location in pixels. YWindow Y location in pixels. glutInitWindowSize(300,300);

**4.2.4 glutCreateWindow Function** glutCreateWindow creates a top-level window.  
**SYNTAX** int glutCreateWindow(char \*name);  
**PARAMETERS**

Name: ASCII character string for use as window name.   
glutCreateWindow("SHADOW CUBE");

**4.2.5 glutKeyboardFunc**  
 glutKeyboardFunc sets the keyboard callback for the current window.  
**SYNTAX** void glutKeyboardFunc(void (\*func)(unsigned char key, int x, int y));  
**PARAMETERS**  func: The new keyboard callback function  
 Key: callback parameter is the generated ASCII character.

x and y: callback parameters indicate the mouse location in window relative coordinates when the key was pressed.

**4.2.6 glutVisibilityFunc**

It sets the visibility callback for the current window. The visibility callback for a window is called when the visibility of a window changes.

**SYNTAX**

void glutVisibilityFunc(void(\*func)(int state));

**4.2.6 glutMainLoop Function** glutMainLoop enters the GLUT event processing loop.  
**SYNTAX**  
 void glutMainLoop(void);  
 glutMainLoop();

**Chapter 5**

**TESTING**

* 1. **Introduction to testing**

Verification and validation is a generic name given to checking processes, which ensures that the software confirms to its specifications and meets the demands of users.

* **Validation**

Are we building the right product?

Validation involves checking that the program has implanted meets the requirement of the users.

* **Verification**

Are we building the product right?

Verification involves checking that the program confirms to its specification.

**5.2 Stages in the Implementation of Testing**

* **Unit testing**

Each individual unit is tested for correctness. These individual components will

be tested to ensure that they operate correctly.

* **Integration testing**

A module is a collection of dependent components such as a function. A module encapsulates related components so can test without other system modules.

* **System testing**

The Sub-systems are integrated to make up the entire system. The errors that result from unanticipated interaction between sub-systems and system components are removed.

* **User Acceptance testing**

This is the final stage in the testing process before the system is tested for operational use. Any requirement problem or requirement definition problem

revealed from acceptance testing are considered and made error free.

* **Test plan**

Careful planning is needed to the most of testing and controlled testing cost..

**5.3 Test Cases**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Testcase ID** | **Testcase description** | **Input** | **Actual output** | **Expected output** | **Remarks** |
| 1. | Display of the cube when it start to rotate. | Press right button option using mouse. | Cube will start to rotate. Refer to Figure 6.1. | Cube must start to rotate. | Pass |
| 2. | Display of the cube when it stop to rotate. | Press right button option using mouse. | Cube will stop its rotation. Refer to Figure 6.2. | Cube must stop its rotation. | Pass |
| 3. | Display of cube and wall when toggle fog is enabled. | Press right button option using mouse. | Cube and the wall will be blurred. Refer to Figure 6.3. | Cube and the wall must be blurred. | Pass |
| 4. | Display of cube and wall when toggle lightening is enabled. | Press right button option using mouse. | Cube and wall light will be brightened. Refer to Figure 6.4. | Cube and wall light must be brightened. | Pass |
| 5. | Display of cube and wall when toggle lightening is disabled. | Press right button option using mouse. | Cube and wall will slightly looks dull. Refer to Figure 6.5. | Cube and wall must slightly look dull. | Pass |
| 6. | Display of the cube and wall when linear fogging is enabled. | Press right button option using mouse. | The fog colour will blend with original colour. Refer to Figure 6.6. | The fog colour must blend with original colour. | Pass |
| 7. | Display of the cube and wall when exponential fogging is enabled. | Press right button option using mouse. | The fog colour will blend with original colour with exponential value. Refer to Figure 6.7. | The fog colour must blend with original colour with exponential value. | Pass |
| 8. | Display of the cube and wall when exponential fogging is enabled. | Press right button option using mouse. | The fog colour will blend with original colour with double exponential value. Refer to Figure 6.8. | The fog colour must blend with original colour with double exponential value.. | Pass |

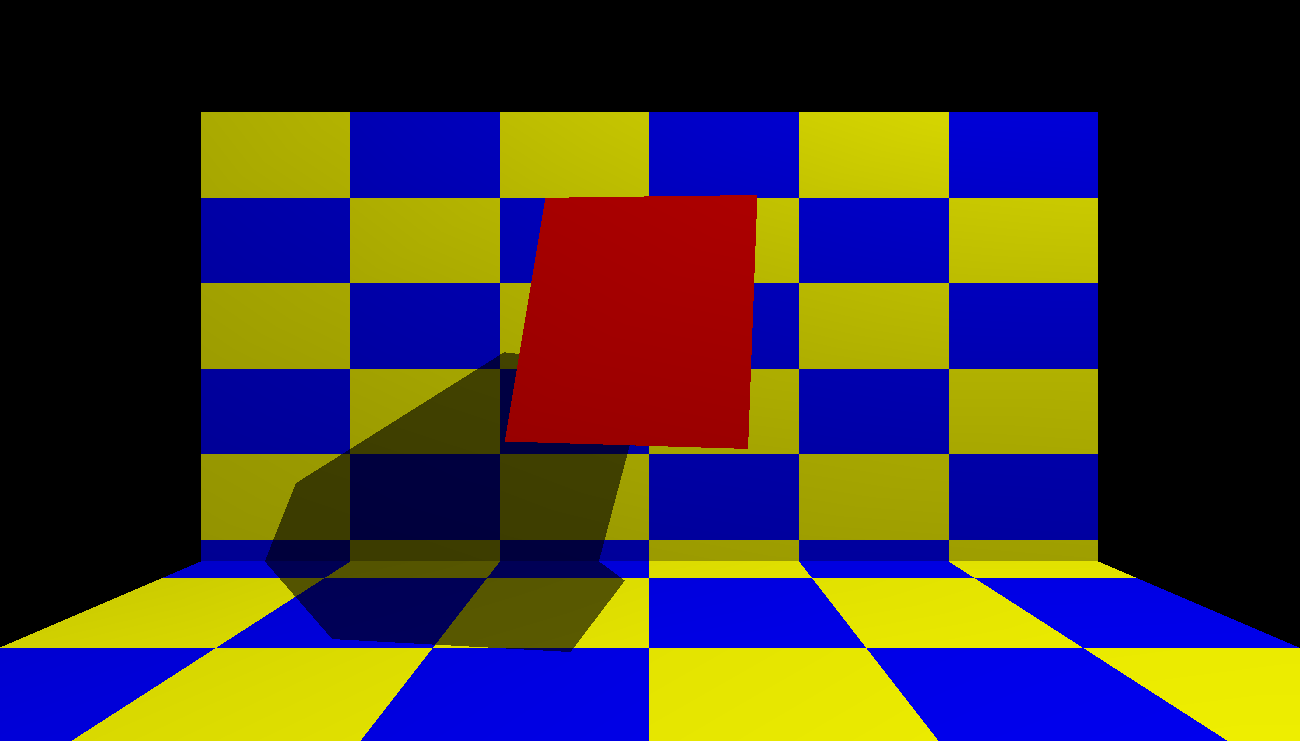
**Figure 5.1** Test Cases.

**Chapter 6**

**SNAPSHOTS**

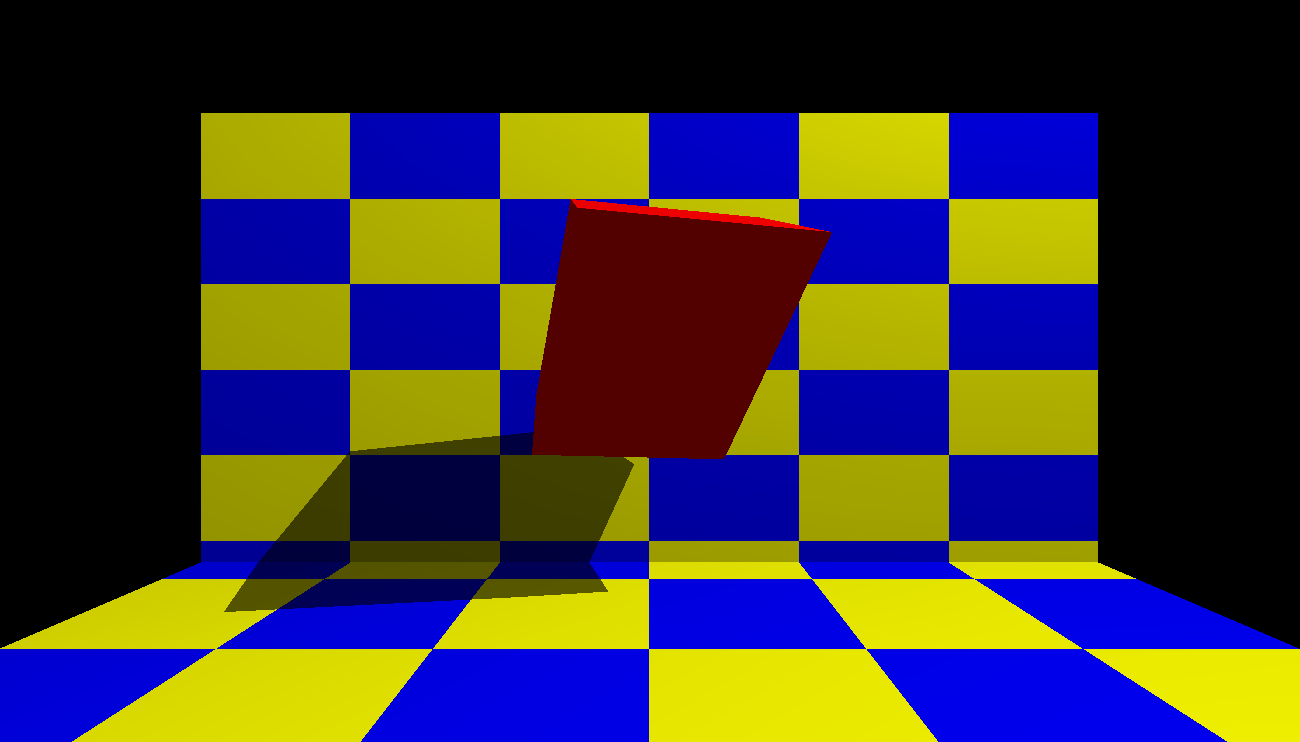
The chapter involves snapshots of the project which provide an idea project. These snapshots depict different simulations of the output.

**START MOTION**



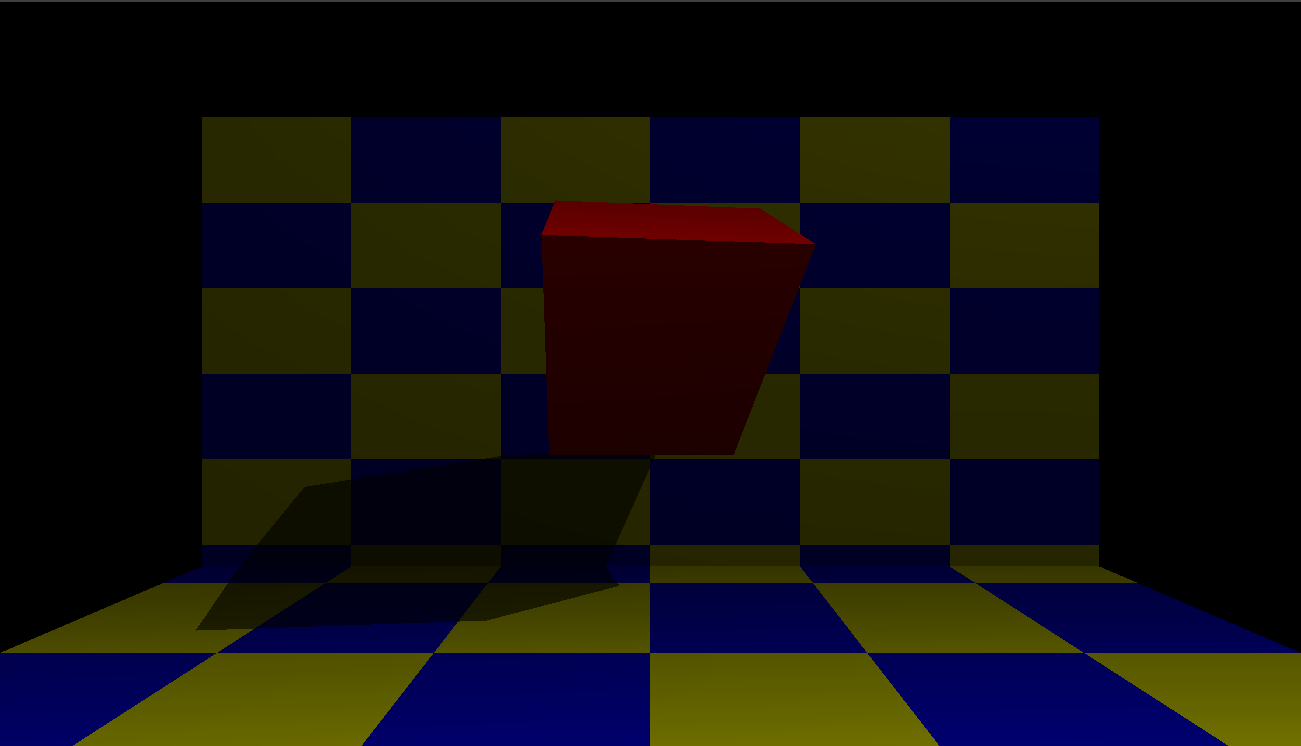
**Figure 6.1** when the cube starts rotating.

By clicking the right button, we get a menu which contains start motion, stop motion, toggle fog,toggle lightening,fog types. Here the start motion option is selected.

**STOP MOTION** 

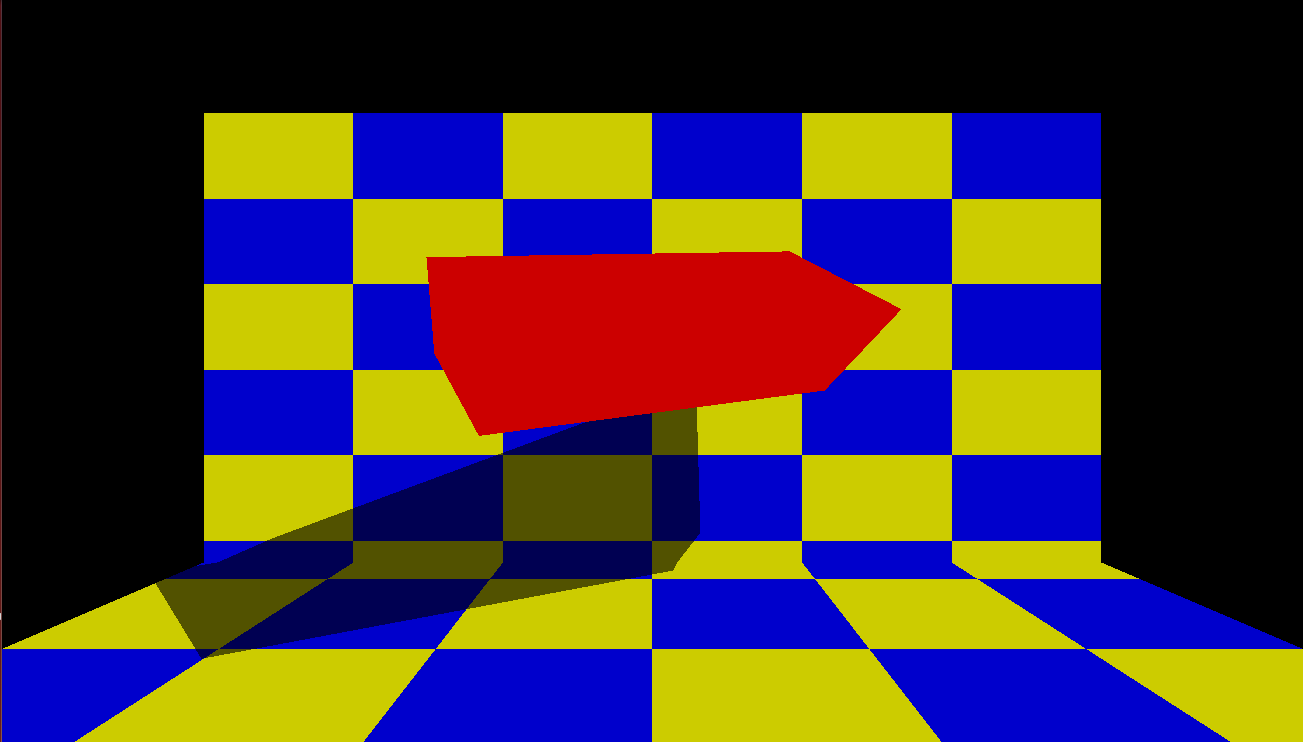
**Figure 6.2** when the cube stop rotating.

By clicking the right button, we will get this output when we give stop motion.

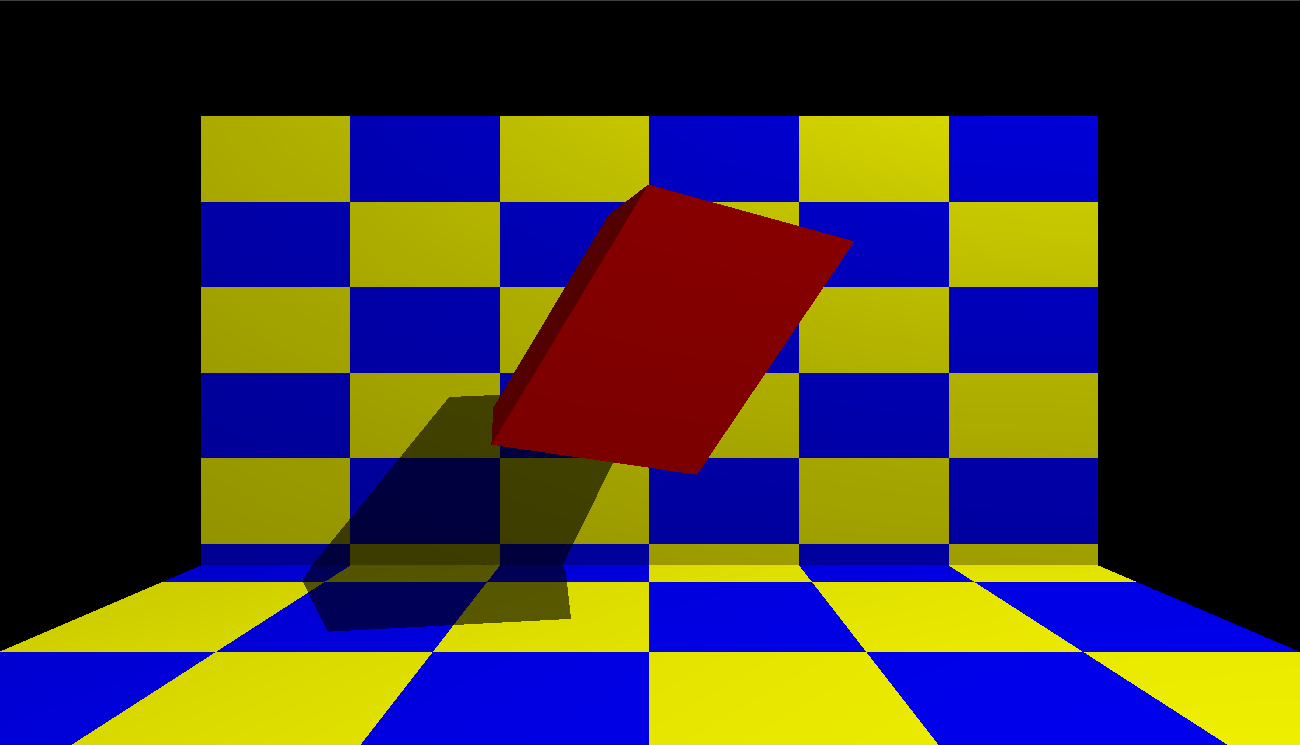
**TOGGLE FOG** 

**Figure 6.3** Toggle fog(on).

By clicking the toggle fog button, the wall and the cube are blurred.

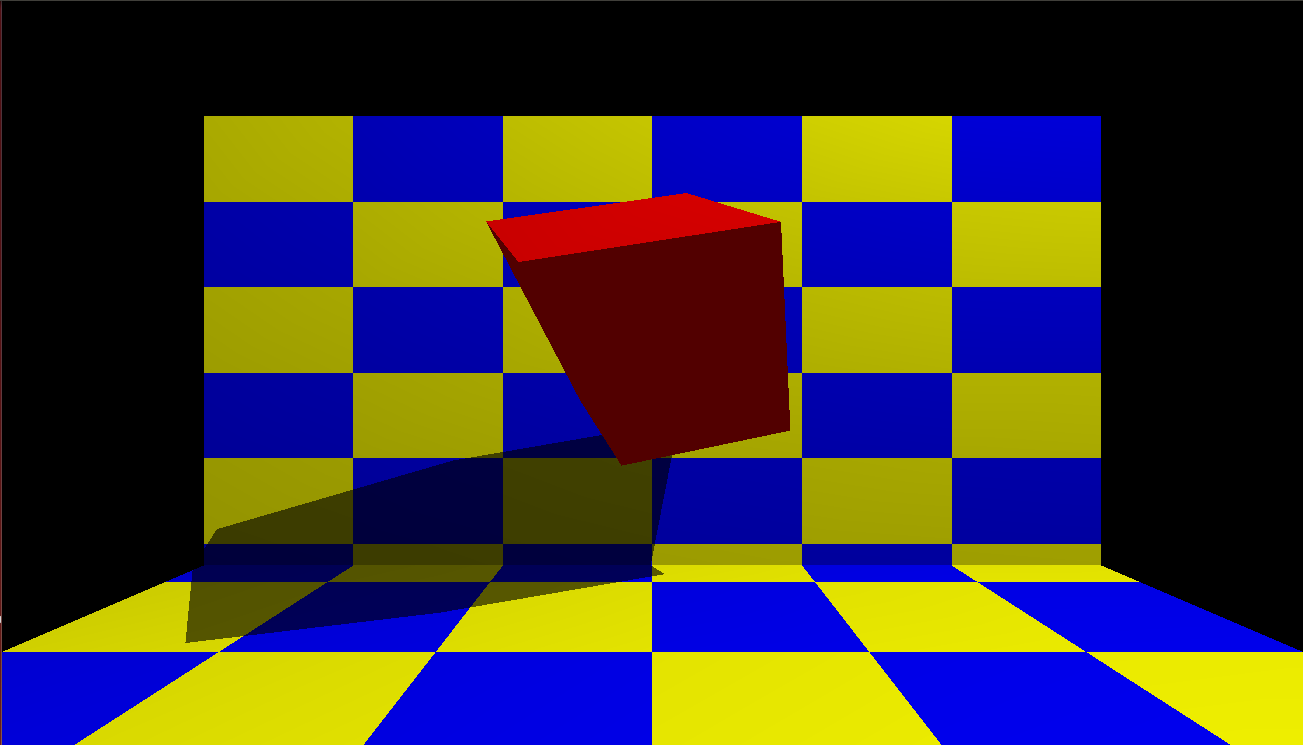
**TOGGLE LIGHTENING** 

**Figure 6.4** toggle lightening(on).

By clicking the toggle lightening button, the light of the wall and the cube is brightened.  


**Figure 6.5** toggle lightening(off).

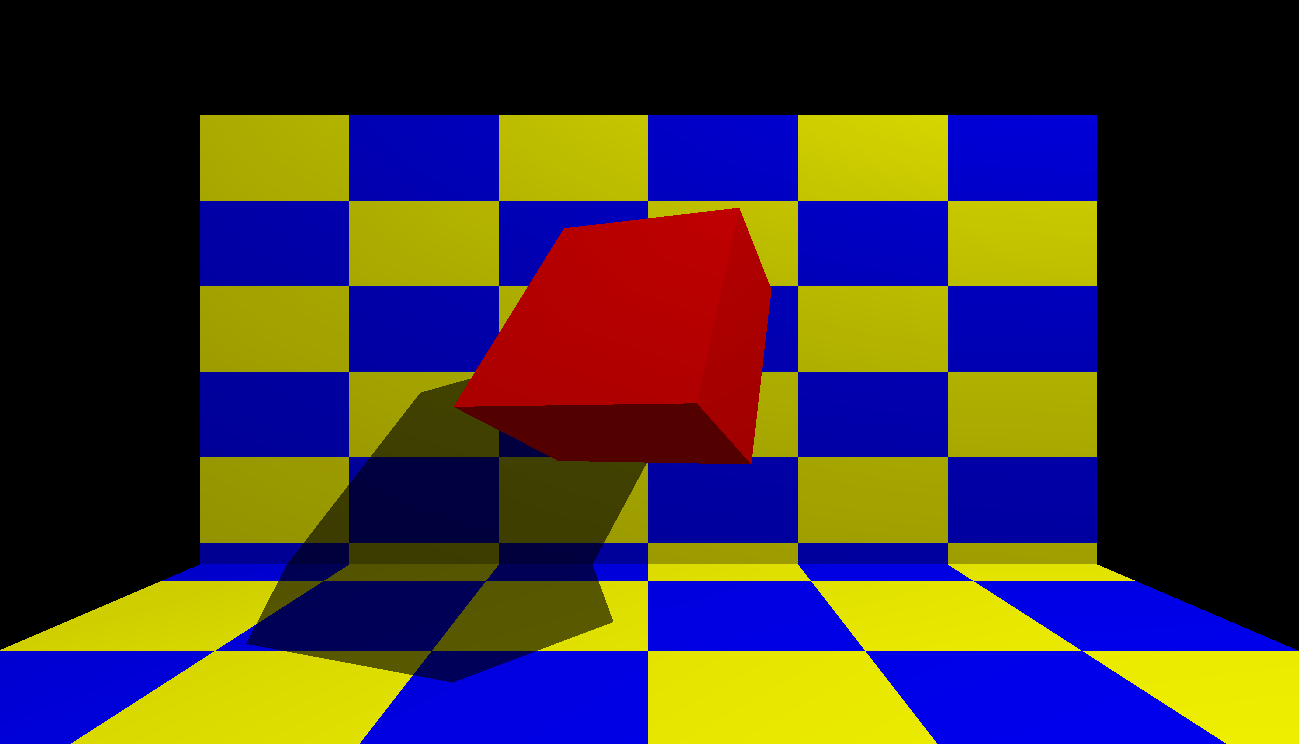
By clicking the toggle lightening button once again the light of the wall and the cube slightly looks dull.

**FOG TYPE**

**Figure 6.6** linear fog.

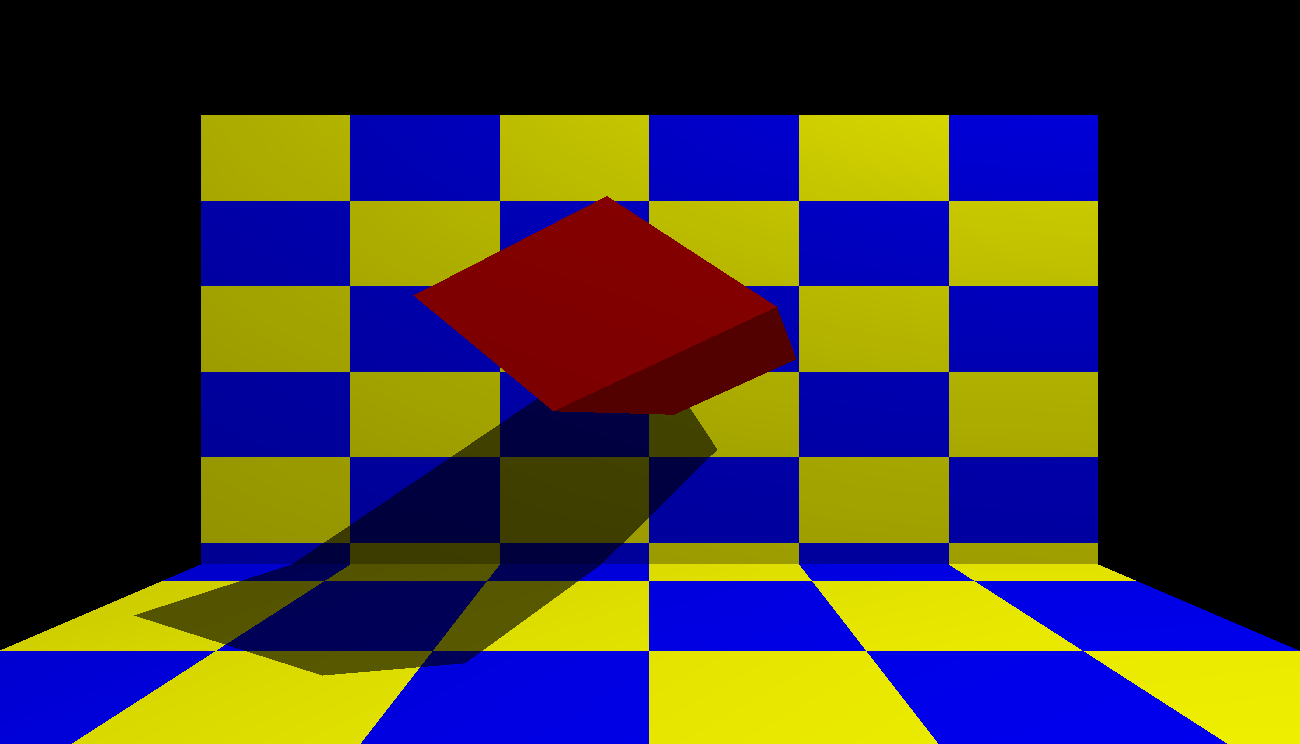
By clicking the fog type option, it gives a sub menu which consists of 3 types of fog types:  
 Linear fog , Exponential fog , Double exponential fog

Linear fog blends the fog color with original color based on the distance you view the object from your scene.



**Figure 6.7** Exponential fog.

Light intensity decreases exponentially with the distance from the object.



**Figure 6.8** Double Exponential fog.

The fog color will blend with original color with double exponential value.