VIRTUAL ARCHITECT

Abhiram R
04CO03

Murali A Varma
04CO24

Under the guidance of
Mrs. Saumya Hegde

In partial fulfillment of the requirements for the award of the degree of
Bachelor of Technology
In
Computer Engineering

Department of Computer Engineering
National Institute of Technology Karnataka - Surathkal

April 2008

VIRTUAL ARCHITECT

Abhiram R
04CO03

Murali A Varma
04CO24

Under the guidance of
Mrs. Saumya Hegde

In partial fulfillment of the requirements for the award of the degree of
Bachelor of Technology
In
Computer Engineering

Department of Computer Engineering
National Institute of Technology Karnataka - Surathkal

April 2008

DEPARTMENT OF COMPUTER ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA, SURATHKAL
SRINIVAS NAGAR -575 025 KARNATAKA, INDIA

CERTIFICATE

This is to certify that the project entitled "**Virtual Architect**" is the record of the bona fide work done by **Abhiram R(04CO03) and Murali A Varma (04CO24)**, in the partial fulfillment of the requirement for the award of the Degree of **Bachelor of Technology** in **Computer Engineering** of National Institute of Technology Karnataka, Surathkal during the Academic Year 2007 - 2008.

Project Guide Head of the Department

Signature of Examiners

1. . . . . .
2. . . . . .

ACKNOWLEDGEMENT

The successful completion of our project gives us an opportunity to convey heartfelt regard to each and everyone who has been instrumental in shaping up this success.

We consider it a pleasure to thank Mrs. Saumya Hegde, Department of Computer Engineering, NITK, Surathkal for having given us an opportunity to work under her, and for her able guidance and encouragement during the course of our project work which was of great help in successful completion of the project.

We are indebted to Department of Computer Engineering, NITK for giving us the opportunity to carry out the project and providing the necessary infrastructure.

A lot of friends were a great source of help along the way. We would like to thank Mr. Sharath P R for his valuable support.

Finally, we would like to thank everyone who has been directly or indirectly responsible for their help.

ABSTRACT

The aim of *Virtual Architect* was to make a 2D to 3D interface where a user would be able to draw a top view of a structure in the 2D sense and the software would convert that into its 3D equivalent, thereby providing a better perspective.

The above objectives have been achieved. We have successfully implemented the 2D to 3D conversion using OpenGL. In addition to this, we have also implemented the importing of 3D objects (Created using 3d StudioMax). These objects may be placed anywhere in the structure and they will show up in their 3D form as well. We have assigned costs to each of the entities and we have a cost estimation feature as well.

The end user is able to construct a virtual home with different types of walls in a 2D interface to see it being mapped into its realistic 3D equivalent. The user may also place objects such as chairs and tables in his structure to see how they would look in real life. In addition to all this, the user is able to get a cost estimate of the structure he is building. The software can be a great asset to budding architects and architecture enthusiasts.

TABLE OF CONTENTS

TABLE OF FIGURES

1 Introduction

Architecture has always laid a tremendous emphasis on the blueprint of the building. Before the widespread use of computers in other professions, architecture dealt mainly with drawing boards and drafters. With computers came CAD and other design based tools, which simplified the act of drawing the blueprint of the building before actually constructing it. Over the years, a large number of commercial software have been written whose sole purpose is to assist the architect in creating the blueprint of the building.

Our project aims to go one step further. In addition to allowing the creation of a blueprint, *Virtual Architect* converts the given blueprint into its 3D equivalent so that the architect can get a better idea about how the structure looks and make the necessary changes.

For example, say the architect draws a blueprint of a building and then constructs it. During construction, he may realize that there are some design flaws that he overlooked in the blueprint. For example, a passage is too narrow, ventilation in a room is inadequate and so on. In this case, it would be too late to make any changes. Instead, if he used the software, he would be able to walk through a 3D model of his structure, analyzing it for potential flaws and correcting them as he navigates.

There are quite a few software packages which offer all these features. But, all of these are commercial and are sold for extremely high prices. There are very few open source software that can accomplish all of the above.

We have aimed to create a software that can accomplish a few of the above mentioned tasks. We have used OpenGL – An open source graphics library to make our software. OpenGL is an extremely popular and widely used library. Its main advantage is that it is simple to understand and use. On the other hand, OpenGL suffers from some limitations with respect to performance and options. OpenGL is primarily written for the C++ programming language. Hence, we have used Visual C++ in conjunction with OpenGL to do our project. Other approaches to the project could include the use of DirectX or Direct3D instead of OpenGL.

The rest of the report is structured as follows. Chapter 2 contains our literature review which highlights the papers we looked through and the various websites we went through as we learnt OpenGL. Chapter 3 describes the Scope and Methodology of the project. Chapter 4 contains a detailed report on the Work Done on the project. Finally, Chapter 5 describes the future scope of the project and the various additions/improvements that can be made to the existing project.

2 Literature Review

We did not read too many papers about OpenGL. We got the idea of doing this project by reading the paper cited in [1].

The paper deals with the general applications of OpenGL. It helped us understand the basic concepts of OpenGL and gauge the capabilities of OpenGL and what we could achieve by applying it in a project.

From here on, our main source of information become the Internet and books. OpenGL, being an open source graphics library, is mainly learnt online. There are numerous websites that contain OpenGL tutorials for download. The websites have all been mentioned in the references section.

As far as books are concerned, we made use of the book cited in [2] the most. Although this book did not contain information specific to our project, it did contain enough information about the various libraries.

A big part of our project deals with the importing of 3D objects created in the software 3dsMax. We used [4] to learn 3dsMax and then the various websites for tutorials regarding the same. These websites, too, have been mentioned in the references section.

2.1 About OpenGL

OpenGL (Open Graphics Library) is a standard specification defining a cross-language cross-platform API for writing applications that produce 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex three-dimensional scenes from simple primitives. OpenGL was developed by Silicon Graphics Inc. (SGI) in 1992 and is widely used in CAD, virtual reality, scientific visualization, information visualization, and flight simulation. It is also used in video games, where it competes with Direct3D on Microsoft Windows platforms.

OpenGL is a specification based on the C programming language. It is built around the concept of a finite state machine, though more recent OpenGL versions have transformed it into much more of an object based system. Though the specification is built on C, it can be implemented in other languages as well.

2.2 Java OpenGL

Java OpenGL (JOGL) is a wrapper library that allows OpenGL to be used in the Java programming language. It is currently being developed by the Game Technology Group at Sun Microsystems, and is the reference implementation for JSR-231 (Java Bindings for OpenGL).

JOGL allows access to most features available to C programming language programmers, with the notable exception of window-system related calls in GLUT (as Java contains its own higher-level windowing systems, AWT and Swing), and some extensions.

2.3 VC++ and OpenGL

Since OpenGL is a C based library, its features suit Visual C++ more than they suit Java. This is why most programmers prefer this combination over JOGL.

We have tried both of the above and find that VC++ suits OpenGL more and hence our project has been made in the same.

3 Scope and Methodology

3.1 Scope

With the world's population increasing at such a rapid rate, perhaps the only other thing increasing as much is the price of land. Land is slowly becoming the most valuable asset that one can have. In these times, real estate becomes more than just a business. Architecture, as a profession has gained importance because of all these factors. Smart builders are the need of the day. Today's builders no longer rely on paper and pencil. Instead, they rely on specially made software packages that make their job easier. Visualization tools are very big in the software market today for this specific reason and they are getting better and better with each subsequent update.

Our project is one such visualization tool and helps in the process of small building. The social impact of this is that it prevents the wastage of resources such as land and materials and helps build a cost effective structure.

3.2 Methodology

Our main task was to get the 2D drawing from the user, convert it into its 3D equivalent using the OpenGL libraries and then allow the user to have a walkthrough of the structure. Since there was no literature on the exact same problem statement stated above, we had to proceed step by step before arriving at the final software. Our entire project life cycle saw the following phases

1) Creating a user friendly 2D interface for drawing, which contained all the necessary options
2) Creating the 3D models of simple objects such as chairs & tables in 3dsMax so that they could be imported later on
3) Importing the objects created in 3dsMax into our 3D view wherever required
4) Converting the 2D drawing into it's 3D equivalent using OpenGL
5) Integrating the 2D and 3D view and providing the walkthrough, lighting effects and other options

Our project required us to port our code from Java to VC++ for some reasons that are mentioned below. Modular programming helped us a lot as it helped in the easy isolation and rectification of errors. It also made the code structured and easy to comprehend.

4 Work done

This project required a breakdown of work as was specified in the Methodology section. The implementation of this project saw 5 tangible modules and each of these went through the following design phases:

Analysis & Specification: A detailed study of what was expected out of the individual modules was made and a formal specification of the functionalities of each was laid out

Design Phase: Based on the specifications, each module was designed keeping in mind further expansions and developments

Implementation phase: The different modules were implemented in their respective environments based on the framework decided earlier

Testing and Deployment: The modules were first tested individually and then as part of sample applications. The results were used to draw up additional specifications required for final deployment of the project.

In the remainder of this section, we will elaborate on each of the five modules in the above four development design phases. Since our project is largely implementation based, our implementation phase will be substantially longer than the other phases.

4.1 Analysis and Specification

2D Interface

The main and only input to the software is obtained in this module. Hence, our main goal was to make this as simple and user friendly as possible. We did a thorough analysis of existing 2D graphics editors such as MS Paint before arriving at a conclusion as to what exact features were needed in our 2D interface. Finally, the following specifications were decided upon and implemented:

- Complete mouse interface for simple drawing
- Simple tabs to represent drawing options to make it user friendly
- Options for selecting drawn objects and moving them as needed
- Option to specify cost of the various building blocks

Creating objects in 3DsMax

Our project required us to create 3D objects. For this, we chose the software package 3DsMax 9 developed by Autodesk™. This involved the creation of the simplest objects with different textures and saving them for further use.

Importing created objects

This was mainly an implementation – only phase. A potential problem existed in this phase when we were using Java™. The importing required the object to look like it's rendered self so that it looked real enough in the 3D view of our software.

Converting 2D design to 3D view

This is the single most important step of the program and this is the one that does justice to the problem statement. As and when the user draws something in the 2D window, we had to draw its equivalent in the 3D domain, true to co-ordinates and scaling factors.

Perhaps the single most important detail to be looked at in this phase was the specification of a co-ordinate system, one that could be used for all subsequent purposes. This system needed to be simple to use and ensure that the relative position of objects with respect to each other was always constant.

Integration of the 2D & 3D view

This was the final output of our software and showed the user both the 2D blueprint and its 3D equivalent in adjacent windows, thereby allowing him to make any rectifications, if needed. This phase required the following specifications:

- Keys to enable walking front and back
- Keys to enable moving up and down
- Keys to enable rotation of the user's view
- Options to select created objects and move them as needed
- Some lighting effects

4.2 Design

Our project did not involve much of a design phase, as most of it is mainly implementation. However, there were a lot of minute details that were needed to be considered before starting the implementation phase. These details have been documented in this section. The main aim of most of the design details has been to simplify the implementation phase.

2D Interface

The 2D interface was designed keeping in mind the specifications mentioned earlier. The 2D interface uses a mouse for all its operations. It has the following options:

- Drawing four different types of walls, which are automatically drawn as rectangles upon dragging with the mouse
- Adding doors adjacent to walls
- Adding simple objects such as chairs and tables
- Selection, moving and deletion of created objects was also added

Creating objects in 3DsMax

Once the window size was gauged, we had to estimate the size of the objects needed to be created in 3DsMax. This phase did not involve too much design.

Importing created objects

We planned to create data structures to store all the imported objects from 3DsMax. We encountered a problem during the implementation phase here, as will be explained further.

Converting 2D design to 3D view

This part required careful design, especially in the design of the co-ordinate system to ensure consistency between objects and their relative co-ordinates. We also needed to ensure that any changes that are made in the 2D view are mapped immediately into the 3D perspective.

Integration of the 2D & 3D view

For the integration of the two perspectives, we designed two adjacent windows of appropriate sizes with the 2D view on the left. As and when something was drawn in the 2D view, it was mapped into its 3D equivalent.

4.3 Implementation

Given the nature of our project, the implementation phase was going to be the key phase. The project entirely depended on this phase. We initially started off by working with JOGL (Java™ and OpenGL) as we were more comfortable with the language and the libraries were easier to use. However, we were forced to make a change midway. This was due to the fact that JOGL did not support the importing of 3D objects created in 3DsMax. This was a feature of our project that we could not do away with and so we were forced to change our platform to VC++. OpenGL

was built for C++ and hence there was no difficulty in importing 3D objects here. Following is a detailed report about the implementation phase of the five modules of the project. We have also attached screenshots of our software in this section. Some of the screenshots are in our Java™ version to help in comparison between the two.

4.3.1 2D Interface

The 2D interface was created keeping the specifications outlined in the design phase in mind. Mouse interface was implemented and drawing was made simple and user friendly. Here are a few screenshots and the technical specifications with respect to the implementation.

Initially, we had made the software in Java. The interface for Java was very simple and not fully developed since we shifted the programming language pretty early in the implementation phase. Nevertheless, it had basic options which were later incorporated into the VC++ version. The following picture shows the 2D interface in Java.

Figure 1 : The Java GUI

Virtual Architect in VC++ (Figure 2) contains 3 windows for the front-end, namely - 2D Drawing Area, 3D Navigation Area and Options. The GUI fills up the complete computer screen with the 2D interface on the left hand side, the 3D interface on the right hand side and the Options Window at the bottom.

The first thing that we needed to take care of was the drawing in the 2D area. This was done using Win32 programming in VC++. For example, in the case of drawing of walls, the user would first click the point where he wants the wall to start from. Then he will keep dragging the mouse till the end point. During the dragging phase, a shadow wall will be produced which gives the user an idea of how it would look like. The start and end points of the wall are obtained based on the click and release positions.

From this we generate four straight lines which represent the top view of the wall. This requires a lot of trigonometric calculations based on the angle of the wall drawn (as seen from the top).

Suppose the start position of the wall is *(StartX, StartY)* and the ending position is *(EndX, EndY)* and the width of the wall is wt = 4, we store values in the *wallList* arrays. Each wall is represented by four lines with the ending point of one line leading up to the starting point of the next line. This means that we need four vertices to represent each wall. This is computed as follows:

```
int wt = 4;
int x1 = StartX, y1 = StartY, x2 = EndX, y2 = EndY;
double theta = atan((double)(y2-y1)/(double)(x2-x1));
wallListx[i][0] =  x1+(int)((wt/2.0f)*sin(theta));
wallListy[i][0] = y1-(int)((wt/2.0f)*cos(theta));

wallListx[i][1] =  x2+(int)((wt/2.0f)*sin(theta));
wallListy[i][1] = y2-(int)((wt/2.0f)*cos(theta));

wallListx[i][2] =  x2-(int)((wt/2.0f)*sin(theta));
wallListy[i][2] = y2+(int)((wt/2.0f)*cos(theta));

wallListx[i][3] =  x1-(int)((wt/2.0f)*sin(theta));
wallListy[i][3] = y1+(int)((wt/2.0f)*cos(theta));
```

Here 'i' represents the wall number, which can go upto a maximum of 100. drawWall() draws the desired rectangle on the 2D interface.

Figure 2 : The VC++ GUI

```
void drawWall(int x1,int y1,int x2,int y2, float wt,HDC hdc)
{
	double theta = atan((double)(y2-y1)/(double)(x2-x1));
	MoveToEx(hdc,x1+(int)((wt/2.0f)*sin(theta)),
	(y1-(int)((wt/2.0f)*cos(theta)), NULL));
	LineTo(hdc, x2+(int)((wt/2.0f)*sin(theta)),
	(y2-(int)((wt/2.0f)*cos(theta))));

	MoveToEx(hdc, x2- (int)((wt/2.0f)*sin(theta)),
	(y2+(int)((wt/2.0f)*cos(theta)), NULL));
	LineTo(hdc, x2+(int)((wt/2.0f)*sin(theta)),
	(y2-(int)((wt/2.0f)*cos(theta))));

	MoveToEx(hdc, x1-(int)((wt/2.0f)*sin(theta)),
	(y1+(int)((wt/2.0f)*cos(theta)), NULL));
```

```
        LineTo(hdc, x2-(int)((wt/2.0f)*sin(theta)),
        (y2+(int)((wt/2.0f)*cos(theta))));

        MoveToEx(hdc, x1-(int)((wt/2.0f)*sin(theta)),
        (y1+(int)((wt/2.0f)*cos(theta)), NULL));

        LineTo(hdc, x1+(int)((wt/2.0f)*sin(theta)),
        (y1-(int)((wt/2.0f)*cos(theta))));
}
```

(x1,y1) and *(x2,y2)* represent the starting and ending coordinates of the wall. *wt* is the width of the wall. The value passed is 4. *MoveToEx()* moves the cursor to the given coordinate. *LineTo()* draws a line from the current position to the specified position. *hdc* represents the current drawing context. As can be seen, the value of theta determines the tilt or the angle

In the case of other objects like chairs and tables, the user just needs to provide the central position with a single click. For a chair, we have the *chairList* array which stores the coordinates of center point of the chair. A similar array stores the coordinates of tables and doors.

For a chair, we have a *drawChair()* function that draws the chair as a square in the 2D Drawing Area:

```
void drawChair(int x1, int y1, float size, HDC hdc)
{
	MoveToEx(hdc, x1-size/2, y1-size/2, NULL);
	LineTo(hdc, x1-size/2, y1+size/2);

	MoveToEx(hdc, x1-size/2, y1+size/2, NULL);
	LineTo(hdc, x1+size/2, y1+size/2);

	MoveToEx(hdc, x1+size/2, y1+size/2, NULL);
	LineTo(hdc, x1+size/2, y1-size/2);

	MoveToEx(hdc, x1+size/2, y1-size/2, NULL);
	LineTo(hdc, x1-size/2, y1-size/2);
}
```

x1 and *y1* represent the coordinate of the chair. *size* is the size of the square. For a chair the value passed is 6. *MoveToEx()* moves the cursor to the given coordinate. *LineTo()* draws a line from the current position to the specified position. *hdc* represents the current drawing context.

A similar function exists for table as well. But since this is a rectangle, the *drawTable()* is a little different from the *drawChair()* function.

```
void drawTable(int x1,int y1, float size, int horiz, HDC hdc)
{
	if(horiz == 0){
		MoveToEx(hdc, x1-2*size/2,y1-size/2, NULL);
		LineTo(hdc, x1-2*size/2,y1+size/2);

		MoveToEx(hdc, x1-2*size/2,y1+size/2, NULL);
		LineTo(hdc, x1+2*size/2,y1+size/2);

		MoveToEx(hdc, x1+2*size/2,y1+size/2, NULL);
		LineTo(hdc, x1+2*size/2,y1-size/2);

		MoveToEx(hdc, x1+2*size/2,y1-size/2, NULL);
		LineTo(hdc, x1-2*size/2,y1-size/2);
	}
	else{
		MoveToEx(hdc, x1-size/2,y1-2*size/2, NULL);
		LineTo(hdc, x1-size/2,y1+2*size/2);

		MoveToEx(hdc, x1-size/2,y1+2*size/2, NULL);
		LineTo(hdc, x1+size/2,y1+2*size/2);

		MoveToEx(hdc, x1+size/2,y1+2*size/2, NULL);
		LineTo(hdc, x1+size/2,y1-2*size/2);

		MoveToEx(hdc, x1+size/2,y1-2*size/2, NULL);
		LineTo(hdc, x1-size/2,y1-2*size/2);
	}
}
```

4.3.2 Creating objects in 3DsMax

3DsMax is a very powerful software created by Autodesk™ created primarily for the purpose of 3D modeling. Hence, we made use of it to create the objects we needed for our project. 3DsMax contains an extremely user-friendly interface with various options to help the user navigate the 3D world.

Following this is an illustration of the sample workspace that we used to create our 3D objects.

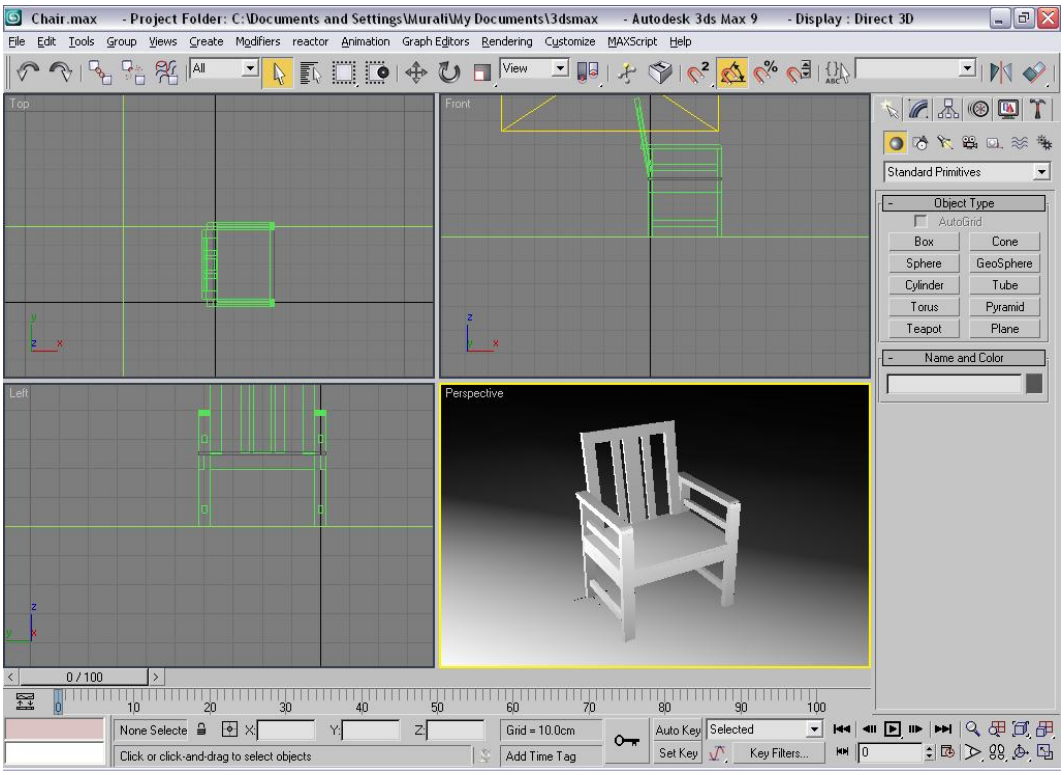

Figure 3 : 3DsMax Workspace

Figure 4 : Rendered Chair

4.3.3 Importing created objects

Once the needed objects were created in 3DsMax, the next important task was to import them into our project. This was where we encountered problems with JOGL. Java™ did not have libraries to support this operation and this caused us to rewrite all our code in VC++. Following this is an example of an object imported from 3DsMax and the code, logic required to import it from 3DsMax. This object was plainly made using boxes of various sizes and using tools like extrude, bevel, rotate, scale etc. which are all available in the 3D Studio Max 9 software.

We have an array for storing the chair textures, namely - *chairTex*. Its value determines which chair to be imported and thereby determines the texture on the chair. We also need to scale the chair to an appropriate size and position it in the correct coordinate.

```
if(chairTex[noOfChairs]==0)
      room_chair[noOfChairs+delChairs].Initialize("Data/3ds/room_chai
      r_1.3DS");//load the file into this instance
else if(chairTex[noOfChairs]==1)
      room_chair[noOfChairs+delChairs].Initialize("Data/3ds/room_chai
      r_2.3DS");
else if(chairTex[noOfChairs]==2)
      room_chair[noOfChairs+delChairs].Initialize("Data/3ds/room_chai
      r_3.3DS");

//scale the model down to 30-millionths its natural size
room_chair[noOfChairs+delChairs].SetScalar(0.0003f);
```

```
//position the model at (StartX,0,StartY)
room_chair[noOfChairs+delChairs].SetPosition(StartX,0.0f,StartY);
noOfChairs++;
```

A similar pattern follows for tables and doors. Now that we made the 3D objects, we had to import them into the scene. The next section explains how we convert the 2D into 3D objects.

4.3.4 Converting 2D design to 3D view

This is the part of the code that actually does the conversion from 2D to 3D, the basic aim of our project. Following this, we will show a couple of example mappings and explain logic and co-ordinate system. We have included most of the Integration phase here itself since the 2 phases are almost the same with respect to implementation.

The first object that we used was the wall. This was constructed in the OpenGL framework using 5 quads, one each on the four sides and one to cover the top of the wall. We assumed a constant height for the wall, meaning that the upward axis in the OpenGL framework (Y-axis) was a constant value of 10. The x-value in the 2D area was equivalent to the x-value in the 3D world. But the y-value in the 2D area matches to the z-value in the 3D realm. This was easily implemented using the 5 quads as explained earlier.

The problem came up while importing using textures. We had a lot of bitmaps for the wall textures. But these couldn't be directly pasted on the wall since the image will be fully stretched across one full side. This would make it look unnatural. So, we had to tile the image across the whole wall, the number of tiles depending on the length of the wall.

This also required a lot of trigonometric calculations. The basic idea was to calculate the length of the wall and apply each bitmap on a specified length of the wall. This way, the wall would look natural. If the wall

length was not an exact multiple of the bitmap length, a partial bitmap would be applied on the remaining piece of the wall. This was the technique used for all the five sides of the wall based on the values stored in the *wallList* arrays.

```
for(int i=0;i<noOfWalls;i++) //noOfWalls is the total number of walls in the world
{
	for(int k = 0;k<4;k++)
	{
		int x0 = wallListx[i][k];
		int y0 = wallListy[i][k];
		int x1 = wallListx[i][(k+1)%4];
		int y1 = wallListy[i][(k+1)%4];
		//Bind currently selected texture to be applied to the wall
		glBindTexture(GL_TEXTURE_2D, texture[wallListTex[i]]);
//Every four glVertex3f() coordinates are used to form a quadrilateral
		glBegin(GL_QUADS);
		int wallHeight = 10;
		float hypotenuse = (float)hypot((x1-x0),(y1-y0));
		float cosComp = wallHeight* (x1-x0)/hypotenuse;
		float sinComp = wallHeight* (y1-y0)/hypotenuse;
		float j;
		for(j=0.0f;j+1.0f<hypotenuse/wallHeight;j+=1.0f)
		{
//glTexCoord2f() assigns the coordinate of the texture to the next specified vertex (in glVertex3f())
```

```
                    glTexCoord2f(0.0f,
0.0f);glVertex3f(x0+(float)j*cosComp, 0, y0+(float)j*sinComp);
                    glTexCoord2f(1.0f,
0.0f);glVertex3f(x0+(float)(j+1.0f)*cosComp,0,
y0+(float)(j+1.0f)*sinComp);
                    glTexCoord2f(1.0f,
1.0f);glVertex3f(x0+(float)(j+1.0f)*cosComp,wallHeight,
y0+(float)(j+1.0f)*sinComp);
                    glTexCoord2f(0.0f,
1.0f);glVertex3f(x0+(float)j*cosComp,                    wallHeight,
y0+(float)j*sinComp);
                    //This is for the top side of the wall
                    if(k==0)
                    {
                          glTexCoord2f(0.0f,
0.0f);glVertex3f(x0+(float)j*cosComp,                    wallHeight,
y0+(float)j*sinComp);
                          glTexCoord2f(1.0f,
0.0f);glVertex3f(x0+(float)(j+1.0f)*cosComp,             wallHeight,
y0+(float)(j+1.0f)*sinComp);
                          glTexCoord2f(1.0f,
1.0f);glVertex3f(wallListx[i][3]+(float)(j+1.0f)*cosComp, wallHeight,
wallListy[i][3]+(float)(j+1.0f)*sinComp);
                          glTexCoord2f(0.0f,
1.0f);glVertex3f(wallListx[i][3]+(float)j*cosComp,        wallHeight,
wallListy[i][3]+(float)j*sinComp);
                    }
             }
             //This is for the last part of the wall when the wall
length is not an exact multiple of the bitmap length
```

```
            glTexCoord2f(0.0f,
0.0f);glVertex3f((float)x0+(float)j*cosComp,                        0,
(float)y0+(float)j*sinComp);
            glTexCoord2f(hypotenuse/wallHeight-j,
0.0f);glVertex3f(x1, 0, y1);
            glTexCoord2f(hypotenuse/wallHeight-j,
1.0f);glVertex3f(x1, wallHeight, y1);
            glTexCoord2f(0.0f,
1.0f);glVertex3f((float)x0+(float)j*cosComp,             wallHeight,
(float)y0+(float)j*sinComp);
            if(k==0)
            {
                  glTexCoord2f(0.0f,
0.0f);glVertex3f((float)x0+(float)j*cosComp,             wallHeight,
(float)y0+(float)j*sinComp);
                  glTexCoord2f(hypotenuse/wallHeight-j,
0.0f);glVertex3f(x1, wallHeight, y1);
                  glTexCoord2f(hypotenuse/wallHeight-j,
1.0f);glVertex3f(wallListx[i][2], wallHeight, wallListy[i][2]);
                  glTexCoord2f(0.0f,
1.0f);glVertex3f((float)wallListx[i][3]+(float)j*cosComp, wallHeight,
(float)wallListy[i][3]+(float)j*sinComp);
            }
            glEnd();
      }
}
```

4.3.5 Integration of the 2D & 3D view

After the successful conversion of 2D to 3D, we needed to add the features needed for the walkthrough.

We added options for moving front, back, up and down. Apart from this, we also added options for rotation and some lighting effects. Instead of the user moving in the world, we decided to move the all objects backward if the user wants to go forward. Similarly a left rotation was implemented by rotating all the objects to the right. The controls used were:

Up arrow - Go forward

Down arrow - Go backward

Left arrow - Rotate left

Right arrow - Rotate right

Numpad 8 - Rotate down

Numpad 2 - Rotate Up

Numpad 9 - Move up

Numpad 3 - Move down

Following this are some screenshots that illustrate the above effects.

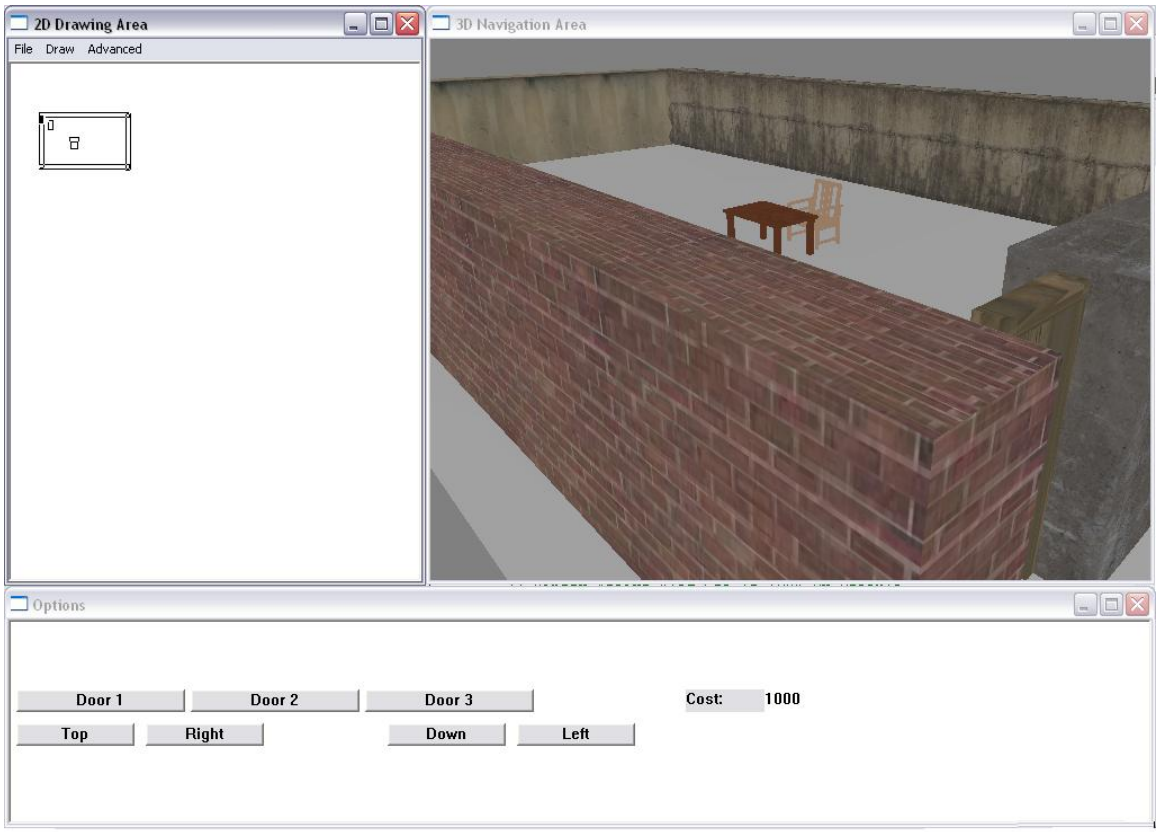

Figure 5 : Sample Home

4.3.6 Options Window

To provide more control over the entire environment, we used to the Options window to give options to the user to choose texture and orientation of the currently selected object. In the case of walls, we have 4 options for textures, namely - brick, mossy, rocky and stone textures. For chairs and tables we have 3 textures, namely - dark brown, light brown and rusty textures. Pressing any of these buttons in the Options window changes the corresponding property of the object. For example, if there are already four chairs in the world and the user chose light brown

texture and left facing for another chair, then the values assigned will be as follows:

```
chairTex[4] = 1; //5th chair corresponds to 4 since numbering starts from 0
chairDir[4] = 3; //0,1,2,3 represents top, right, down, left repectively
```

We also have a textbox in the Options window which inputs a cost/unit value for the wall. The total cost of the wall will depend on its length. There is also a cost textbox for chairs, tables and doors. This is the cost for each component. The value can be changed according to the user's choice by inputting suitable values. The cost is stored as *wallCost[]* in the case of walls and *chairCost[]*, *tableCost[]* and *doorCost[]* for the other objects. An incorrect input will result in an error message to popup.

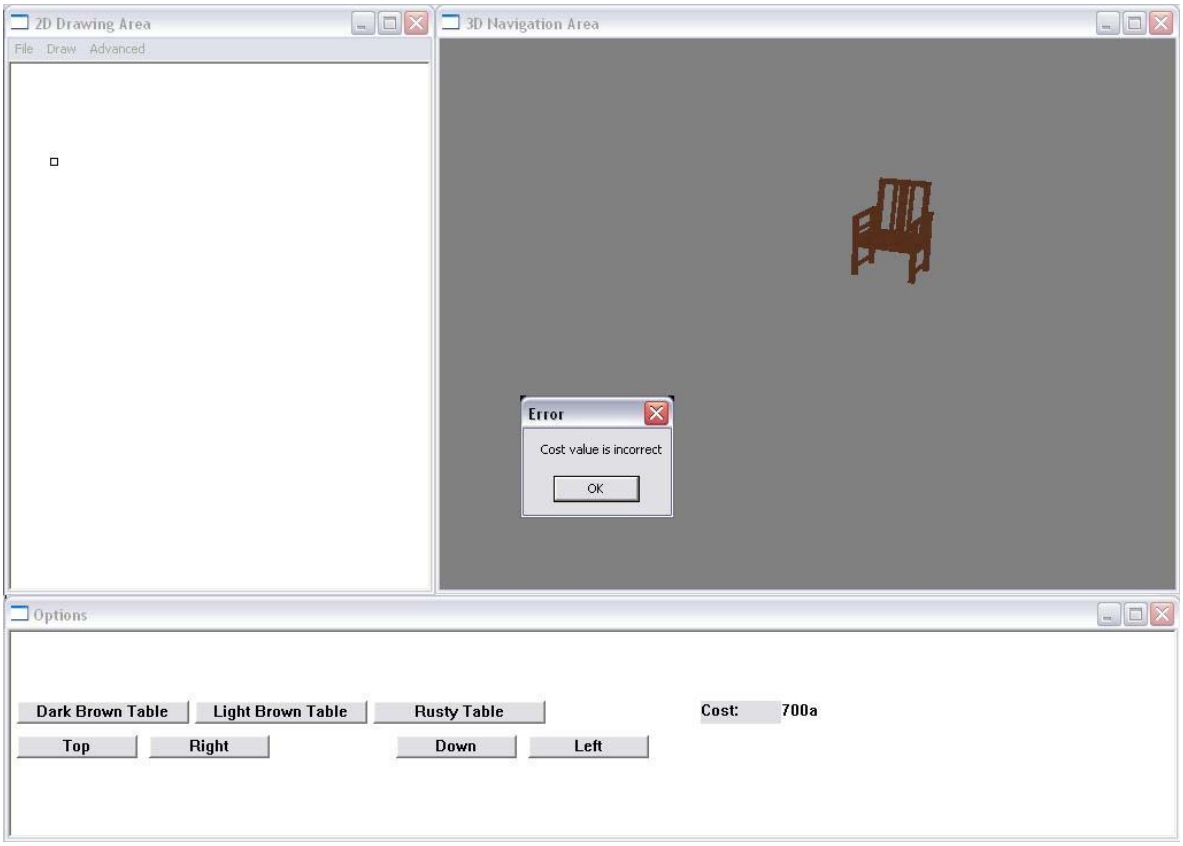

Figure 6 : Error Checking

The user can choose to see the total cost of the constructed building as a popup by selecting the option from the File menu.

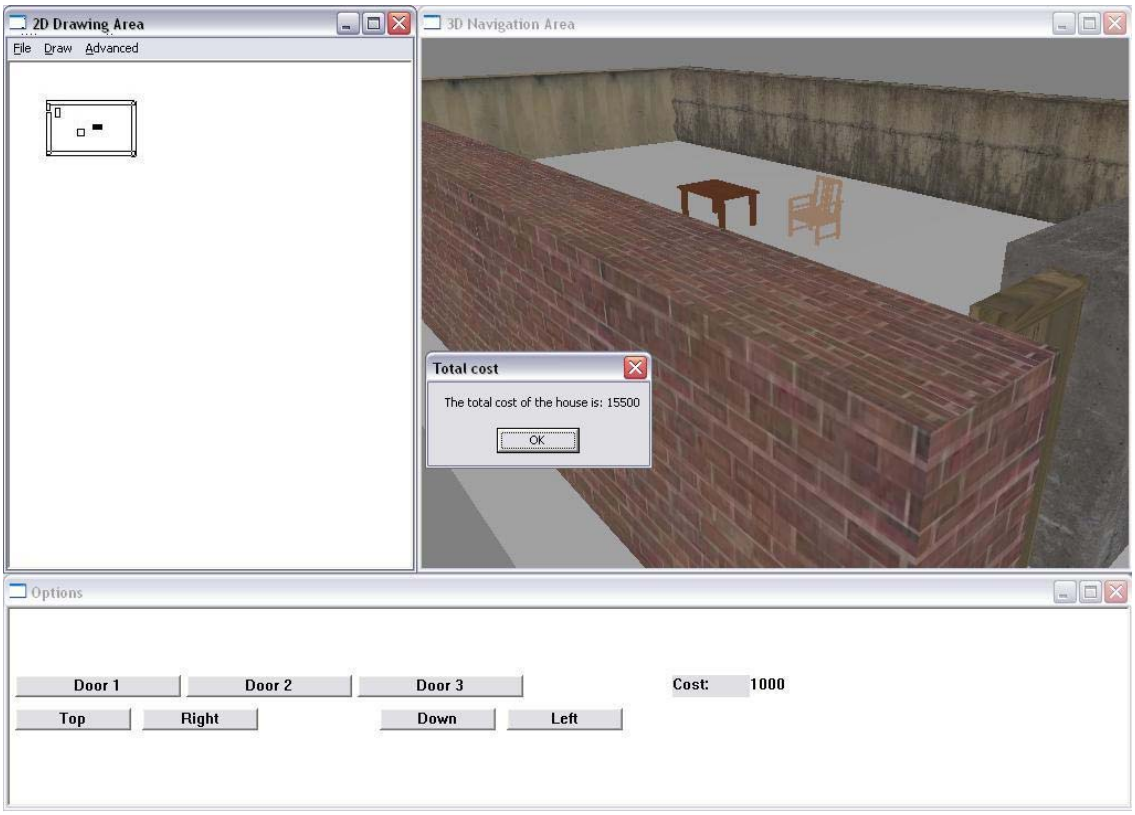

Figure 7 : Cost estimation

4.3.7 Selection, Movement and Deletion

Another very useful feature in this software is the ability to select objects by the click of a button. The nearest object to the point of clicking is chosen. If the object is too far away from the point of clicking, then no object is chosen. The closest object is calculated using the *closest()* function.

```
int closest(int x, int y)
{
	double mindist=10;
	for(int i=0;i<noOfChairs;i++)
	{
		if(hypot(chairList[i][0]-x,chairList[i][1]-y) < mindist)
```

```
            {
                  seli=1;
                  selj=i;
                  mindist=hypot(chairList[i][0]-x,chairList[i][1]-y);
            }
      }
      for(i=0;i<noOfTables;i++)
      {
            if(hypot(tableList[i][0]-x,tableList[i][1]-y) < mindist)
            {
                  seli=2;
                  selj=i;
                  mindist=hypot(tableList[i][0]-x,tableList[i][1]-y);
            }
      }
      for(i=0;i<noOfDoors;i++)
      {
            if(hypot(doorList[i][0]-x,doorList[i][1]-y) < mindist)
            {
                  seli=3;
                  selj=i;
                  mindist=hypot(doorList[i][0]-x,doorList[i][1]-y);
            }
      }
      return seli*100+selj;
}
```

The parameters are the coordinates of the point of clicking. *mindist* is the maximum distance upto which a click will select an object. *seli* = 1 represents a chair, 2 and 3 represents table and door respectively. *selj* represents the number of the object in the selected category. For example, *seli*=2 and *selj*=4 means that the selected object is a table and that the selected table number is 5(numbering starts from 0).

The selected object is identified by filling the full object with black colour. This is done by drawing the object in black colour and reducing the size of the object till 0. For example, when a chair is selected, it is represented in the 2D are using the following code.

```
if(seli == 1)
{
	SetROP2(hdc, R2_WHITE); //Set the pen colour to white
	for(int i=0;i<6;i++)
	drawChair(chairList[selj][0], chairList[selj][1], i, hdc);
	SetROP2(hdc, R2_BLACK); // set the pen colour to black
	drawChair(chairList[selj][0], chairList[selj][1], 6, hdc);
}
```

Once an object is selected, the user has the option of moving it anywhere in the world. There is also the option of deleting the chosen object. It is moved in the 2D area by deleting the current object by drawing the object with a white pen and drawing the same object at a new position using a

black pen. The object is moved in the 3D area by varying the value of *wallList* arrays, *chairList[]*, *tableList[]* and *doorList[]* values.

The controls used are:

/ - Move selected object left

* - Move selected object right

- - Move selected object top

+ - Move selected object down

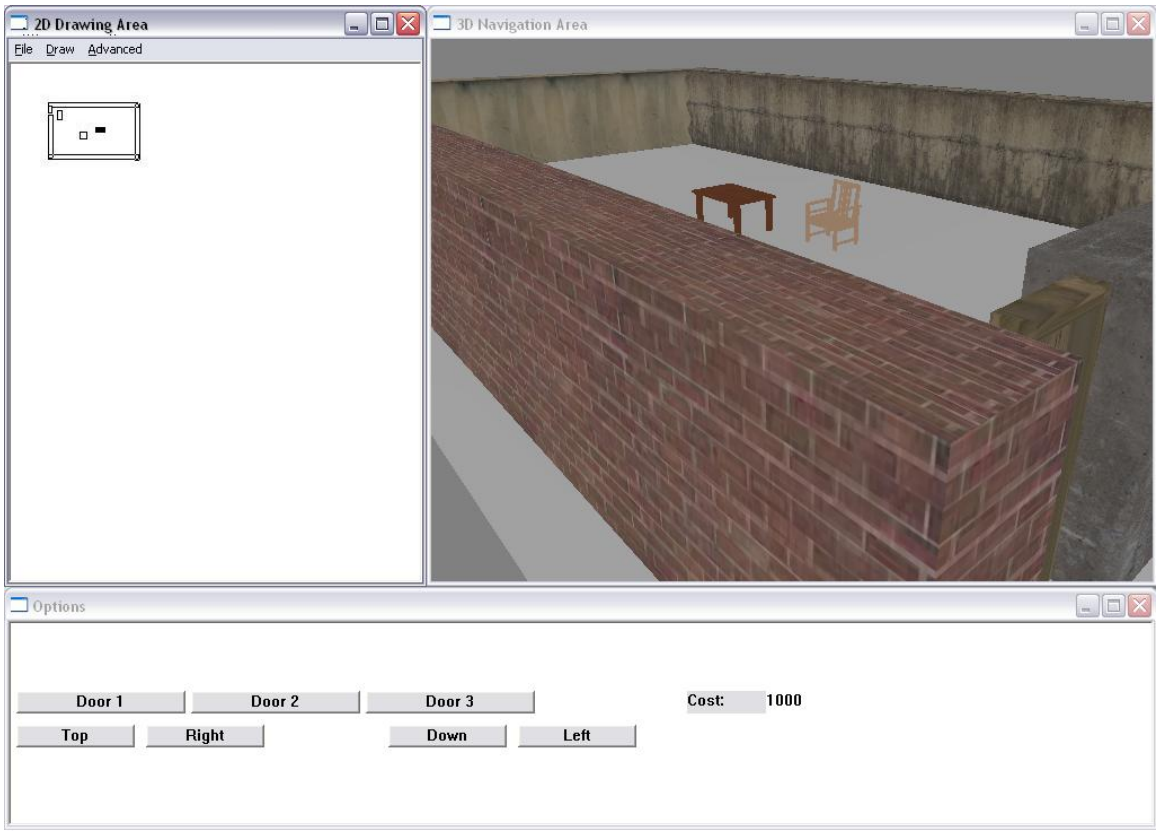

Figure 8 : Moving objects

Finally, an object is deleted by pressing the delete button. This requires the removal of the entries in the corresponding arrays. For example, if the 5th door was deleted and there is a total of 8 doors, then the 5th door is replaced by the 6th door the 6th door by the 7th door and the 7th by the

8th. Next we delete the 8th door and reduce the number of doors by one. In the OpenGL we couldn't uninitialize the object assigned to the 3ds model. Thus we had to keep a count of the number of deleted objects just so that each time a new object, say a chair, is created, it is not assigned to *noOfChairs*, but to *noOfChairs+delChairs*. The other parameters are all assigned to *noOfChairs*.

4.4 Testing and Results

Once our implementation was complete, the testing phase began. As the case is with most graphics based software, there were quite a few issues to be dealt with. All the modules were tested individually for flaws and then integrated. Here are a few of the problems we faced while working on our project:

- When dealing with the 2D interface, we encountered some problems with respect to the drawing, where it used to overlap existing drawings and replace them. This was encountered by placing a flag for each co-ordinate
- When an object was deleted, the global object array used to shift to cover the free space but the object initialization function did not. Hence, this had caused a problem

- When we were working with Java™, we encountered the serious problem of not being able to import 3D objects and hence we had to switch platforms
- We encountered a performance issue when we placed too many objects. Speeds were considerably lowered when the number of objects went beyond twenty or so
- The buttons in the GUI were giving us some issues by overlapping with each other
- Textured flooring was found to affect performance

The results of our project are as shown in the Implementation phase. All the above bugs have been corrected and our project is fully functional. There are many ways in which it can be improved, as will be observed in the following section.

5 Conclusions and Future Work

With the implementation of this project, we have shown that it is not very hard to develop commercial software, especially meant for the real estate industry. Open source is the future and all Computer Engineers must strive to develop more and more useful software.

Results show that OpenGL is not as powerful as advertised. For a full fledged version of a *Virtual Architect* we would probably have to use more powerful libraries such as DirectX or Direct3D. That being said, OpenGL is extremely useful too and has enabled us to come up with the functional project. The cost estimation feature makes this kind of project extremely useful for people constructing homes within a given budget.

The next step in the project would be to add more objects, which is a hard job as they require careful modeling in 3DsMax followed by importing. An important feature that can be added to this is the photo-realistic snapshot. The concept behind this is called *backtracking.* We pick a pixel and observe all the light falling on it, the intensity and the angle of all the light as well. When we represent all these details for all the points in the given frame, we arrive at the photo-realistic snapshot which will look extremely real complete with perfect lighting effects and shadows.

6 References

[1] Michael Carr, *"Visualization with OpenGL : 3D made easy"*, International IEEE conference on Multimedia held in 1999

[2] Dave Shreiner, Jackie Neider, Tom Davis and Mason Woo, *"OpenGL Programming Guide – The Official Guide to learning OpenGL"*

[3] Richard S Wright, Benjamin Lipchak and Nicholas Haemel, *"OpenGL Super Bible ; Comprehensive Tutorial and Reference"*

[4] Murdock and Kelly L, *"Wiley 3DsMax 9 Bible"*

[5] *www.nehe.gamedev.net*

[6] *http://www.winprog.org/tutorial*

[7] *http://www.functionx.com/win32*