

Comp 510 Project Report

# **PH7: A Virtual Museum Based on OpenGL and Glut**

---

Hassan Salehe Matar  
Pirah Noor Soomro

6th June, 2016

## Summary

We present a simple virtual museum implemented entirely using OpenGL and Glut as a final project for the Comp 510: Computer Graphics course. In this project we exercise with different features of OpenGL and various concepts we learned from the course lectures. Some of the concepts explored in this project includes: modeling objects using vertices; instancing; translations, rotations, and scaling as object transformations; shading; texture mapping; user input interactions; projections; blending; and hierarchical modeling.

The contents of the museum are the modeled room with two windows on each side of the longer side walls and three windows on the rear wall and a door at the front wall. The museum has a front door which can be opened or closed by mouse-clicking it. Moreover we modeled a roof of the museum by mimicking a structure and look of *corrugated* iron sheets with blue color. On the rear wall, at the top, we have a continuously rotating ventilating fan which can be stopped or restarted by clicking on it. Inside the rooms we have six stands with colorful looks where we put antiquity objects on top of them. We used PLY model object files from Florida State University to create these antique objects by parsing the files, performing appropriate transformations and placing them on the stands. The antiquity objects can be manipulated by clicking on them. Additionally, they can continuously rotate if the museum visitor types letter 'A' on the keyboard. We also modeled a sun at a distant location from the museum building. We use the central vertex of this modeled sun as the source of the light illuminating the museum and the objects inside.

The main contributions of this project are

- (a) A nicely modeled building for the museum.
- (b) Reading, transforming and properly positioning the museum antique objects
- (c) A number of user interactions with the museum and the antique objects

In the rest of this report we discuss various aspects in our project. We discuss how we modelled different parts of the museum. We describe the parsing and transformations of the antique objects. We also explain how we used different concepts that we learned in class to implement our project. We also describe how the user can interact with the museum. Finally we provide information about the source code of the project and how to run it.

---

## 1. Introduction

In this report we address four things. First we focus on the features or objects, including the building, found on our virtual museum. Second, we discuss different concepts and techniques we used in this project and how they are related to the concepts we learned in class. Third we explain different interactions a user can perform with the virtual museum. And finally we provide relevant information on how to access, compile and run the source code.

## 2. The objects

All the objects in the scene are arranged in a tree structure. As some objects have close transformation relation -- and to preserve that relation -- objects are arranged in a hierarchy (hierarchical modeling). The hierarchy of objects in the museum scene is shown in Figure 1. The room object is at the root of the hierarchical tree. The room object models the front, rear, left and right walls, and the room floor. The children objects to room are the modeled sun, the window frames, the stand for the antique objects in the museum, the front door frame, and the upper headers of the front and rear walls. Each window frame holds two panels each for its left and right sides. Similarly, the door frame has two panels. We separated the panels, in the case of the door in particular, to animate them individually when the door opens or closes. Moreover the ventilating fan is attached to the rear, upper wall.

The stands hold the antique objects inside the museum room. We have six stands, five of which are the instances of the first stand. The antique objects on top of these stands are the skull, walkman, wheel, lamp, airplane, and a spare part. These objects are the leaf nodes whose parents are the stands in the hierarchical representation of objects as shown in Figure 1.

We implemented each object in Figure 1 as a separate C++ class. This has a number of advantages. It forms loose decoupling among objects thus simplifying reasoning of individual objects. Moreover it facilitates implementation collaborations in our team because each team member can work on different classes.

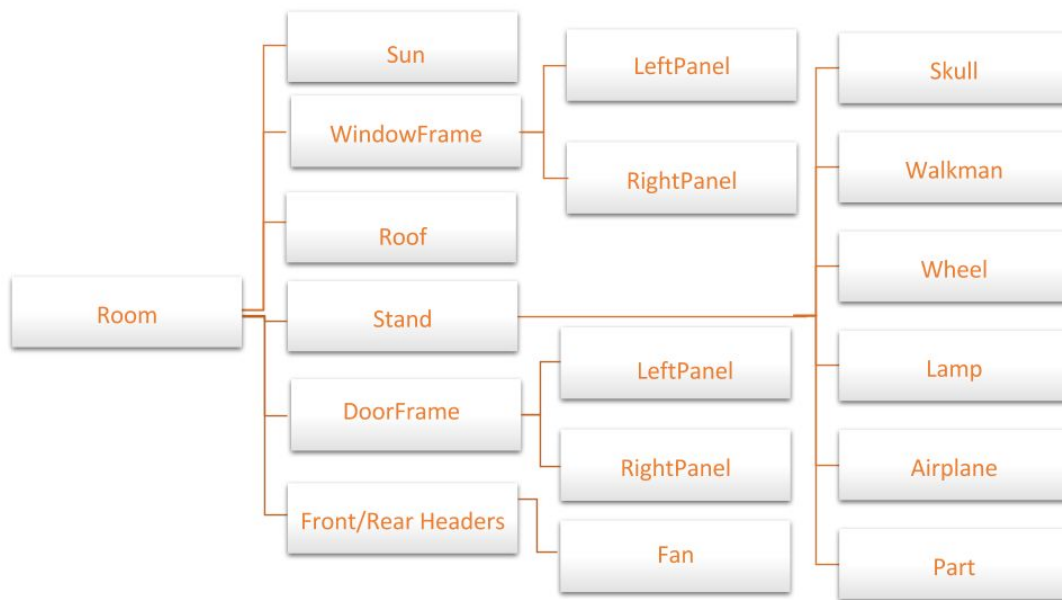


Figure 1. The museum objects in hierarchical order

The tree structure in Figure 1 is constructed in the file `Museum.h`. Each object is created and appended to its parent node as a child. Then to display the objects or update their information, a C++ *queue* is used to push and pop the tree nodes in order to access them in breadth-first manner. Listing 1 below is the code for appending a child to a particular node. Each object keeps the list of pointers to child nodes.

```

1 void appendChild(Object * child)
2 {
3     children.push_back( child );
4 }

```

Listing 1: a function for appending the child objects of a an object node in the hierarchical tree

Each class of an object in the museum inherits a base class called *Object* (found in file *Object.h*) which contains fields and methods that are common for all objects. Moreover it provides virtual functions for individual objects to implement differently depending on individual properties and the related manipulations. Moreover, individual objects can override methods already defined in *Object* to provide custom, additional functionalities.

Table 1 below is a sample of method signatures implemented by each object or inherited from the *Object* base class:

Method name	Description
<code>void initializeDataBuffers( GLuint program )</code>	For initializing the vertex buffer objects and sending data to the shaders
<code>void changeShading()</code>	For switching between Gouraud and Phong shading
<code>void changeReflection()</code>	For switching between Modified Phong and Phong reflect.
<code>void reshape( int w, int h )</code>	Executed when the window is resized
<code>void toggleAuto()</code>	For automatic rotation of antique objects
<code>void changeProjection()</code>	Switching between perspective and orthogonal projections
<code>virtual void checkIfPicked( unsigned char pixel[4] );</code>	Each object receives the pixels picked and Takes appropriate actions based on that
<code>virtual void idle() = 0;</code>	Executed when the museum is idle.
<code>virtual void rotateLeft( GLfloat delta ) = 0;</code>	For rotating the museum or individual objects around Y-axis.
<code>virtual void rotateUp( GLfloat delta ) = 0;</code>	For rotating the museum or individual objects around X-axis

Table 1: Some of the common functions implemented in Object class or by individual objects

## 2.1. Construction of the museum objects

We will now explain the construction of each object in the museum in detail. Generally there are two main categories of objects; the building, and the antiquities.

### 2.1.1. Constructing the building

The building consists of one roof, one room with side walls, a door frame, two door panels (left and right), seven windows each with a window frame and two panels. Moreover it has a front header and a rear header, a ventilating fan, and the sun (it is not the part of building but included in this category just for simplicity). We mathematically calculated all the vertices of these parts of the building. In this section, we discuss each of these objects.

#### 2.1.1.1. The room

The room consists of a rectangular cuboid without the top face. It is centered at origin (0, 0, 0) of the vertex coordinate system. The dimension of the cuboid is 1.6 x 1.0 x 1.0. Figure 2 shows the geometric representation of the room. Each face is customized with sub-faces (quads) to leave spaces for the door and the windows.

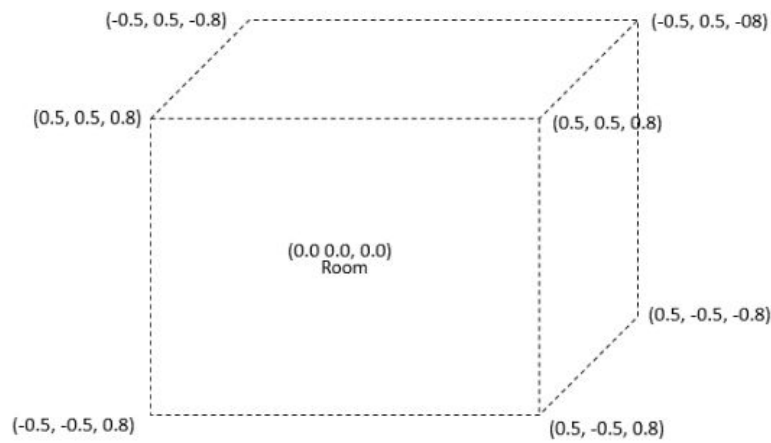


Fig 2: The geometric representation of the room

To better understand the geometry let us study each wall in more detail.

**The front wall:** In front wall presented on Figure 3 we draw four(4) quads. The main purpose of drawing these four quads is to leave an empty space in the middle for the door which is constructed as a separate object. These faces are on the left, right, top and bottom parts of the wall. Just to make things simpler we have given IDs to the vertices. More details on the implementation and the exact values of these vertices is found in *Room.h* source file.

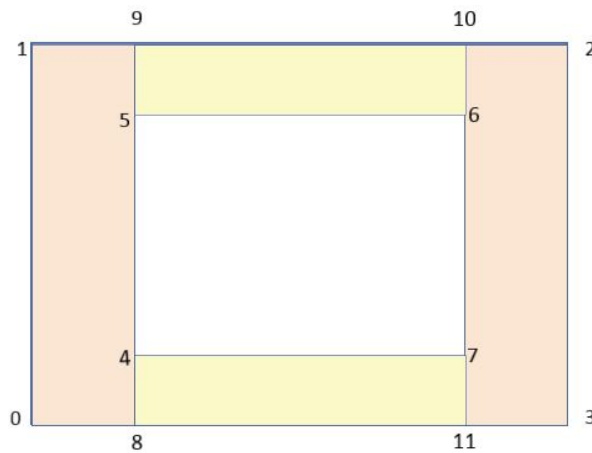


Fig 3: Geometric representation of the front wall which leaves empty space at the middle for the door. The vertices are numbered to simplify construction of side faces.

**The rear wall:** The rear wall of the room contains 3 small windows, shown in white in Figure 4. In order to make space for the windows, 6 quads each containing two triangles are created. Since the size of the wall is  $1.0 \times 1.0$ , each window hole has size  $0.2 \times 0.2$ . Therefore the top quad has size  $0.1 \times 1.0$  and each of the faces separating the windows has the size of  $0.2 \times 0.1$ . The lower face which is the largest has the size of  $0.7 \times 1.0$ . The implementation specifics are found in *Room.h*.

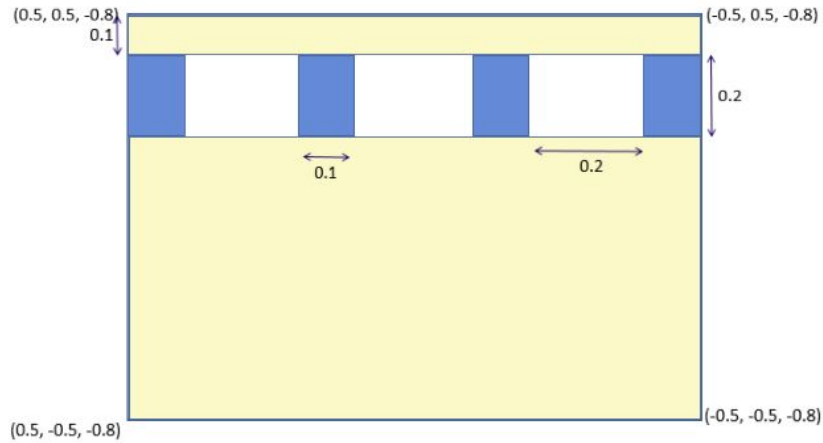


Figure 4: The rear wall. The yellow and blue rectangles are the quads and the white squares are the empty spaces left for the 3 small windows.

**The side walls:** The left and right side walls are identical where the right wall is an instance of the left side wall formed by rotating it  $180^\circ$  around the Y-axis of the vertex coordinate system. Each of these walls has 5 quads and spaces for two windows which are bigger than the rear windows, as shown on Figure 5.

We modeled the color of the room to resemble that of earth (soil) which given to all vertices of this basic room infrastructure. For each of the triangles for the wall faces, we calculate the normals and send them to the GPU.

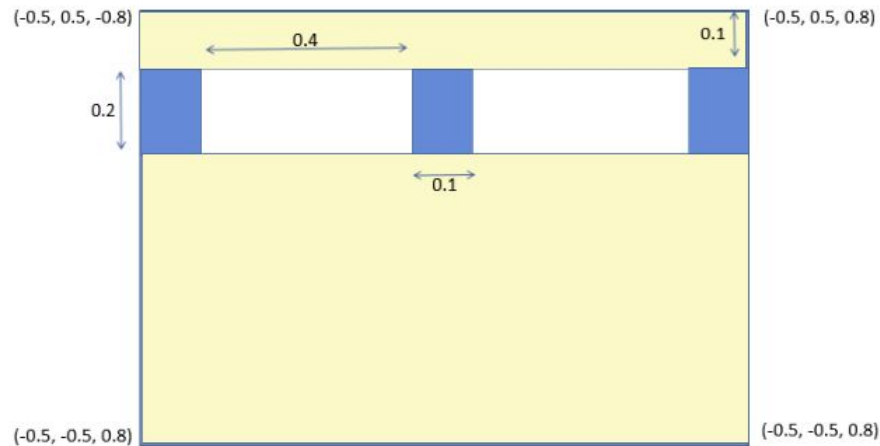


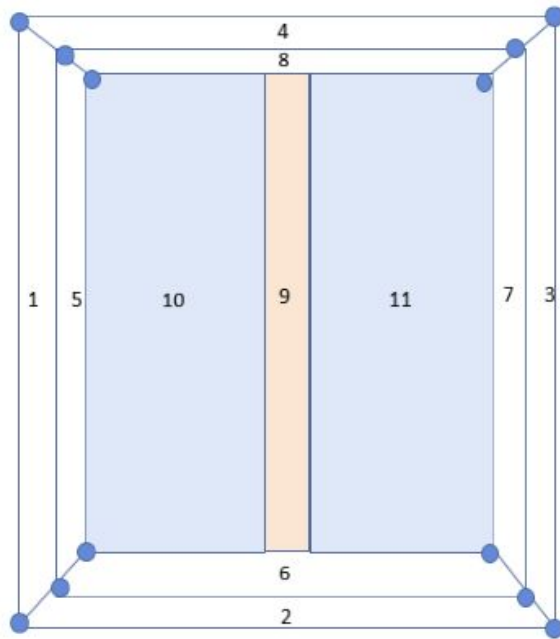
Fig 5: A representation of one side wall. The white rectangles are the empty spaces for the windows and the remaining rectangles are the quads.

### 2.1.1.2. The door of the museum

The door is designed in such a way that it has two panels which open on either sides of the door. Basically each of the left and right panels of the door rotates at its individual axis to animate the opening and closing of the door. Section 3.8 explains in detail how the door operates. Architecturally, door is designed with 11 geometrical faces (rectangles and trapezoids) as shown in Figure 6(a) to give the feel of a door mounted to a wall as shown in Figure 6(b). The exact detail of the vertices and its multiple instances can be found in *DoorFrame.h*, *LeftDoor.h* and *RightDoor.h*

**The door frame:** The trapezoids 1 to 8 constitutes the door frame. There are two door frames, one frame covers full door's skeleton and the other sub frame outlines individual door instances. The door frames are independent of the door so they can have their own color, normals and textures. Whereas the modelview of outer frame is attached with the room's modelview and that of the inner frames modelview follows transformation matrix of the door.



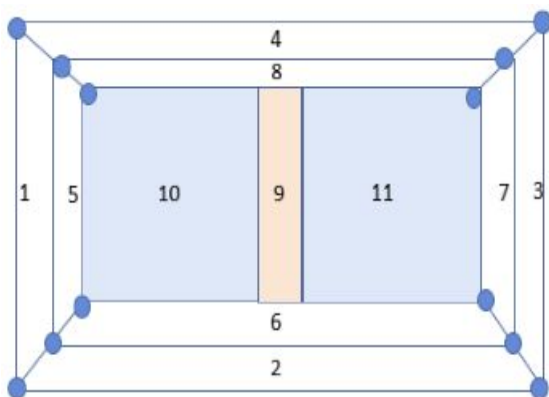


(a) showing different faces identified by numbers. 10 and 11 are the left and right panels which open and close on either sides (b) The final looks of the door with the outer frame, two individual door panels, and the handle

Figure 6: The geometry and look of the front door

### 2.1.1.3. The windows

The windows geometry is almost the same as of the door. Their positions however differ. The sizes of rear wall windows is different from the side windows.



(b) Windows of side wall

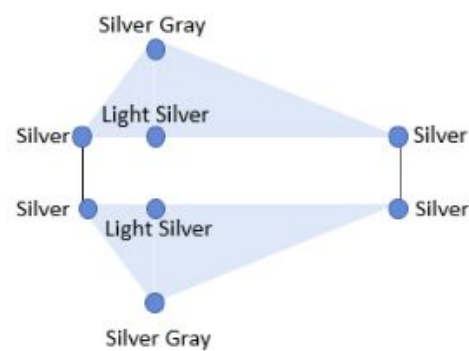


(c) Rear Windows

(a) Windows geometry

Fig 7: The geometric representation and the feel of the the windows in the museum.

**The handle for the door and the windows:** Door and windows have a modeled handle with a mixture of gray colors of varying intensity. This handle is attached to all instances of right panel of the windows and the door. The handle is static and it does not have its own movement and thus it serves to provide a nice look to the windows and the door. Let's examine the geometry of handle.



(a) The color distribution at vertices



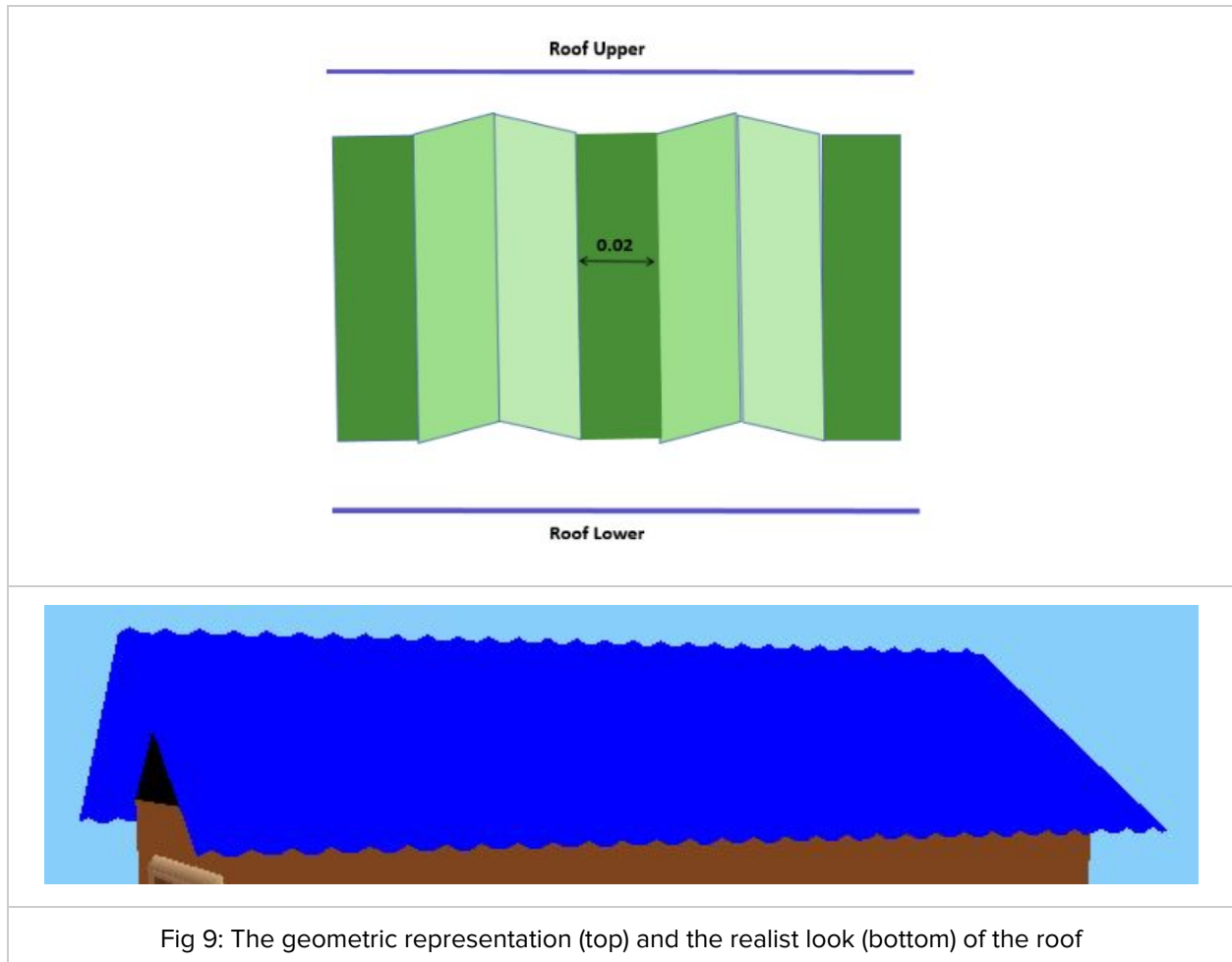
(b) After rendering

Fig 8: Color distribution at vertices of the handle as well as how it looks after rendering.

To give an effect of shading, colors are assigned to all vertices (Figure 8(a)). All the fragments get the interpolated color value, depending on the nearest vertex, hence an effect of shading appears in the object. This phenomenon is clearly observable in the final look of handle in Figure 8(b).

#### 2.1.1.4. The roof

We modeled the roof as a strip of planes to create a look and feel of corrugated metal (iron) sheets. Two consecutive face planes meet at edges forming an acute angle of 36 degrees. We construct roof on both sides of the room. Figure 9 illustrates the architectural representation (top) and how the roof looks like after rendering (bottom). The full implementation of the roof is found in *Roof.h* in the project source codes.



#### 2.1.1.5. The sun

The sun is a combination of 6 triangles with a common vertex forming a hexagon. The color of the common vertex is yellow while the color at each of the remaining vertices resembles the scene background color. When rendered these triangles simulated a sun shining at the edges of the outer vertices. Figure 10 shows the sun modeled as a hexagon of six triangles with a common vertex at the center. The image on the right shows how it looks when rendered.

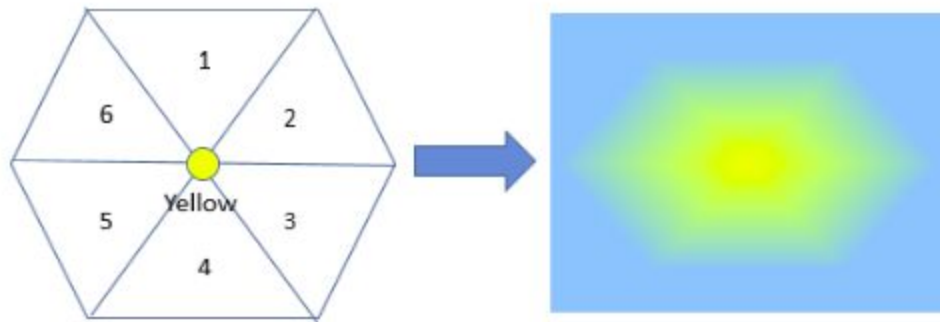


Figure 10: The model and the look of the sun

#### 2.1.1.6. The ventilating fan

The ventilating fan is found at the upper part of the rear wall of the museum. This fan continuously rotates using the “idle” function implemented in *Fan.h*. There are two aspects to this fan. First, we created a hexagonal hole at the middle of the upper part of the wall. This hole is surrounded by triangles with vertices on the edges of the hexagon and the three corners of the header of the wall (the top, left and right corners). The second aspect is the modeling of the fan wings which continuously rotate. The left image on Figure 11 shows a single wing. It consists of two triangles forming a hill-like face. Then we place three of these wings to form the rotating wings. Moreover, we add one more wing and elongate it to form a handle as it appears in the middle and right images in Figure 11.

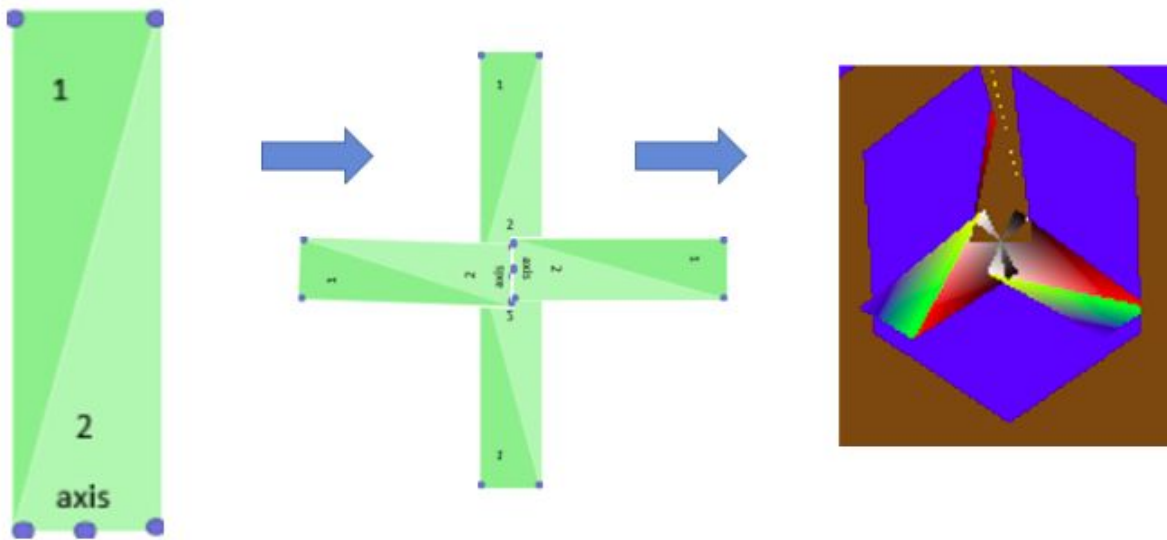


Fig 11: The ventilating fan construction and the look after rendering

### 2.1.2. Constructing the antiquities

The second class of objects in virtual museum consists of antiquities. Now we will define the structure, rendering and transformations of these museum objects. The objects under discussions are;

Stand, Human skull, Walkman, Airplane, part, lamp and wheel. Generally speaking, these objects have nothing to do with history information. We have just selected these objects to fill to act as historical objects as a proof of concept. Therefore, there is no need to know any historical information about them. Next we are going to see their geometry and rendering in detail.

#### 2.1.2.1. Stands to hold the antiquities

The antiquity objects “seat” on stands. There are six instances of a stand in the room; each for one antique object. The structure of a stand is very simple as it is an elongated cuboid. Each instance is a translated version of the first stand. The vertices are defined in *Stand.h* and various colors are assigned to the vertices to give a colorful look as shown in Figure 12.

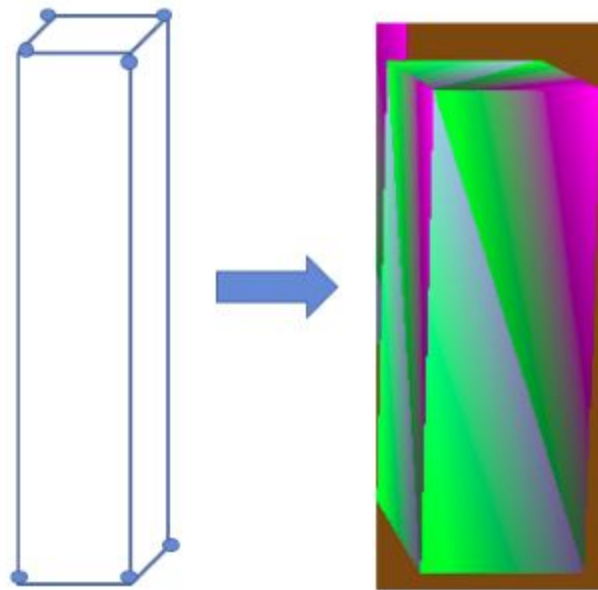


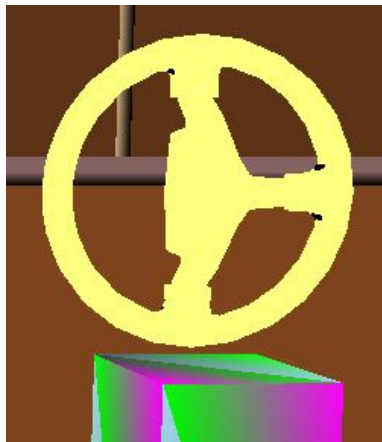
Fig 12: Stand: the model and its look after rendering

### 2.1.2.2. Antique objects

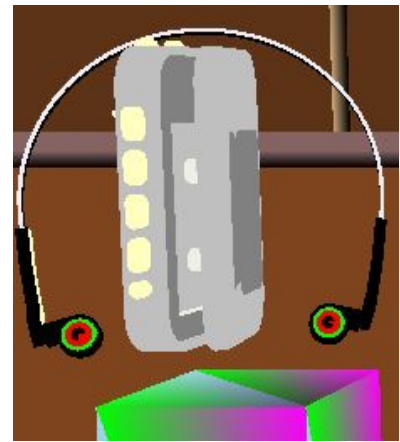
We have six antique objects selected from a list of *ply* object models from the website of John Burkardt of Florida State University[1]. These objects are the skull, the steering wheel, the walkman, aeroplane, (spare) part, and a lamp. We used a ply parser called *rpoly*[2] to parse the ply model files. Then we located vertices and colors to appropriate arrays. Then we applied a number of transformations on the objects to fit them into our museum. The finally rendered images of these objects are shown in Figure 13.



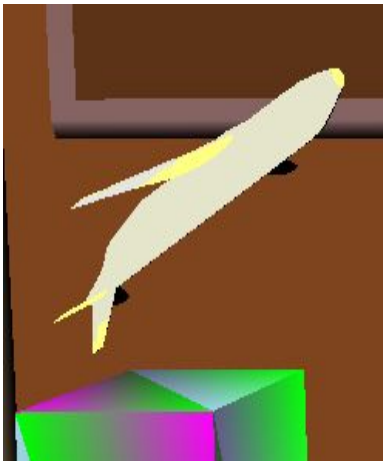
Skull



Wheel



Walkman



Airplane



Part



Lamp

Fig 13: Final look of antique objects after parsing and applying a number of transformations

In the following section we look at the format of the ply object files and how they are parsed before rendering. Most of the information has been adapted from[2].

**PLY Files:** “A PLY file contains the description of one object. This object is composed by elements, each element type being defined by a group of properties. The PLY file format specifies a syntax for the description of element types and the properties that compose them, as well as comments and meta-information” [2].


The element type descriptions come in a header, which is followed by element instances. Element instances come grouped by their type, in the order of declaration. Each element instance is defined by the value of its properties. Properties values also appear in the order of their declaration. Here is a sample PLY file describing a triangle adapted from[2]:

```
ply
format ascii 1.0
comment this is a simple file
obj_info any data, in one line of free form text
element vertex 3
property float x
property float y
property float z
element face 1
property list uchar int vertex_indices
end_header
-1 0 0
0 1 0
1 0 0
3 0 1 2
```

The header starts from a string “ply\n” that indicates that file is in ply format and ends with “end\_header” indicating that necessary file information has ended and actual data follows.

The lines which start with comment (#) are comments. The lines that start with obj\_info contains meta information. These two properties are optional and their order does not matter.

In the sample PLY file, the first element type is declared with name vertex, here we have 3 instances of vertex. The properties following describe what a vertex element looks like. Each vertex is declared to consist of 3 scalar properties, named x, y and z. Each scalar property is declared to be of type float.



Next, the face element type is declared, of which only 1 instance will be given because we are going to make only one triangle. This element consists of a list property, named `vertex_indices`. Lists are sequences on which the first value, the *length*, gives the number of remaining values. Following the header, come the elements, in the order they were declared in the header. First we have list of vertices and then the list that starts with 3, indicates that following values represent the indices of vertices of a list. Here “3 0 1 2” says that a face has vertices 0,1, and 2. To read this file, we used a library RPLY its source can be found from [2]. There are special methods to read necessary information from the ply file such as:

```
p_ply ply_open(const char *name, p_ply_error_cb error_cb, long idata, void *pdata)
```

This method Opens a PLY file for reading, checks if it is a valid PLY file and returns a handle to it.

```
int ply_get_ply_user_data(p_ply ply, void *pdata, long *idata)
```

It Retrieves user data from the ply handle.

```
int ply_read_header(p_ply ply)
```

It Reads and parses the header of a PLY file.

```
long ply_set_read_cb(p_ply ply, const char *element_name, const char *property_name, p_ply_read_cb read_cb, void *pdata, long idata )
```

This routine sets up the callback to be invoked when the value of a property is read.

There are also other useful functions which can be found from manual [3].

## 3. Concepts as in class

In this section we briefly discuss various concepts used in our project that we learned from the course lectures.

### 3.1. modeling objects using vertices

One of the concepts we learned in class was to model different scene objects using vertices . Three vertices form a triangle and a combination of triangles forms faces or surfaces of objects in the graphics scene. In our project we have demonstrated the user of vertices to create all the



components of the museum with an exception of the antiquity objects. The full details of the objects modeled in this project are in section 2.

### 3.2. Object transformations: translations, rotations, and scaling

We apply various transformations to all the objects in museum. These transformations are represented in modelview matrices of the individual objects. As an example, we apply rotations, zooming, and translations to the museum building and its components. The overall model view matrix of these transformations is stored in the root node (the room) of the hierarchical tree of the museum objects. Moreover it is distributed to all nodes in the tree so that the overall effect of these translations is applied to all the museum objects.

### 3.3. Shading

In the vertex and fragment shaders we implemented the Phong shading model and the Gouraud shading model. Moreover, we implemented both the Phong, and the Modified Phong reflectance models. To facilitate shading, we calculate normals of all the objects we designed and the antiquities. Then we send the normals to the shaders for calculating shading.

### 3.4. Texture mapping

Texture mapping is defined as “*The application of patterns or images to three-dimensional graphics to enhance the realism of their surfaces*”. We implemented texture mapping in front head triangle of the museum. We drew a simple texture image on *Paint*, an image editor on Windows OS, and mapped it to the front header of the museum. As the header is just a triangle, we tried to apply three simple *st* coordinates, based on our intuition as follows:

```
vec2 tex_coords[3]={  
                                vec2(0.0,  0.0),  
                                vec2(1.0,  0.0),  
                                vec2(0.5,  1.0)  
};
```

Listing 3: Defining texture coordinates.

### 3.5. User input interactions

We implemented a number of ways a user can interact with the museum by using keyboard special functions and keys as well as the left mouse button. Simple interactions with the museum involves rotation, translation, zooming, and changing museum appearance features like projections and shading. Moreover, the user can select different parts of the museum and the objects inside it by clicking using the left button of the mouse. These features are discussed in detail in section 4.

### 3.6. The help message

To get help information on how to use the system the user has to click letter “h” or “H” on the keyboard. This triggers a function `Museum::help()` in `Museum.h` file. This function prints the help information into the terminal (console) and the user has to look at it to see the help message. The help message has been revised and simplified. Figure 14 shows the snapshot of the message printed on the console when user clicks letter “h” or “H” while the museum is active.

```
=====
=                                     =
= Welcome to PH7: The Virtual museum developed only on OpenGL. =
=                                     =
=====
=                                     =
=          HOW TO INTERACT WITH THE MUSEUM          =
=                                     =
=====
= Keys          == Function          =
=====
= H or h        == To get this help message. =
= Mouse-click   == To pick an(any) object.   =
= I or i        == To reset the museum       =
= Z             == To zoom IN               =
= z             == To zoom out              =
= f             == To move forward           =
= F             == To move backward          =
= S or s        == To change shading         =
= A or a        == To turn auto rotation on/off of antiques =
=              ==                          =
= Left arrow    == To rotate Left            =
= Right arrow   == To rotate Right           =
= Up arrow      == To rotate up              =
= Down arrow    == To rotate down            =
= Click on door == To open / close the door  =
= Q or q        == To close/quit the museum  =
= Click window  == To open/close window (blend) =
=====
```

Fig. 14: A snapshot of a Linux terminal with the help message from the museum

### 3.7. Selecting objects

We implement the picking technique we learned in class to select various objects in the museum. Depending on the object, a click on it triggers an action. Objects active for picking on a mouse click are the museum door, the windows, the rear ventilation fan, and the museum antiquities.

### 3.8. Opening / closing the museum entrance door

A click on the closed door opens it slowly until it stops after 135 degrees of rotation. When user clicks on a door panel which is open already, the door closes itself in the same manner as it opened but this time in a reverse direction.

There are two panels of the door mounted to the door frame on either hinges, each. When opening or closing of the panels is animated; it is the panels which rotate around the hinges (along the Y-axis). The general transformation matrix operation for these rotations involves translation of individual panels to the origin, rotation around Y-axis at the origin, and translation back to the hinges. This is reflected better in the source code files *LeftDoor.h* and *RightDoor.h* in the function `calculateModelViewMatrix()`.

### 3.9. projections

We implemented both perspective and orthogonal projections. However, the museum looks better with perspective projection.

### 3.10. blending

We use blend functions, as taught in class, to simulate the opening and closing of the side windows of the museum. Initially when the museum launches all the windows are closed. Then user can open the windows by clicking on them. Similarly the user can close the windows by clicking on them.

Basically, we keep a flag which is toggled whenever a user clicks on the windows. Using this flag we enable/ disable blending of the windows.

```

if( canOpenWindow )
    enableBlending()

DisplayWindow()

if( canOpenWindow )
    disableBlending( )

```

Listing 4: A pseudocode for blending windows. This is implemented in Window.h

### 3.11. Hierarchical modeling

Hierarchical modeling is done when there is dependence relationship among the objects. In the museum the overall movement of individual objects is dependent on the movement of their parent objects. For example the room holds the stands and stand holds the skull. In this kind of environment, as discussed before, some behaviors and properties remain the same across objects in the hierarchical tree and some change on individual objects. We implemented hierarchical modeling by constructing an object tree shown on Figure 1. Much details about our implementation and how these objects interact can be found in section 2.

## 4. User interactivity

In this section we discuss a number of ways a user can interact with our museum. The user interactions can be achieved by pressing specific keys on keyboard and the left button of the mouse. Table 2 summarizes different keys implemented in the museum to interact with it.

Function	Key	Function	Key
Get help	h or H	Automatic Rotation on/off	a or A
Pick an object	mouse-click	Rotate Left	Left arrow key
Reset All	i or I	Rotate Right	Right arrow key
Zoom In	Z	Rotate Up	Up Arrow Key
Zoom out	z	Rotate Down	Down arrow key
Move Forward	f	Pick one object to rotate	left Mouse key
Move Backward	F	Open The door	click on door
Switch between Gouraud and Phong shading models	s or S	Blending Windows	click on the window
Switch between Phong / Modified phong reflections	r or R	Quit/ close	q or Q
Change Projection	p or P		

Table 2 : Keyboard keys to interaction with the museum

## 4.1. Rotating whole museum

The museum as a whole can be rotated through special keyboard keys. We have implemented rotation around X-axis and Y-axis. Left and right arrow keys provide rotation around X-axis while up and down arrow keys provide rotation around Y-axis.

## 4.2. Moving the museum

The museum can be moved in such a way that gives an effect of going away or closer to the user. This is implemented by simultaneously scaling and translating the building, as shown on listing 5. The user can press f or F key on the keyboard to move the museum.

```
void moveForward( GLfloat delta ) {  
    Distance[Zaxis] += delta;  
    scaleFactor += delta;  
    glutPostRedisplay();  
}
```

Listing 5: A function which changes rotation angles and scaling to move the museum forwards or backwards

To move backward, we call the same **moveForward** method with negative value of delta.

## 4.3. Rotating, and focusing on individual antiquity objects.

There are two modes of rotating individual objects of the museum. First, the key “a” or “A” toggles the automatic rotation on and off. When automatic rotation is *on*, all the antique objects start rotating around their own central axis. When the auto rotation mode is off, user can select any object by clicking on it. By clicking the object the picking procedures starts and program triggers the relevant callback move method of that object. The object rotates at 30 degrees along the Y-axis on each click. Meanwhile user can go closer to that object by pressing “f”, “F”, “z”, or “Z”. To facilitate the Individual rotations, model view of each object is multiplied with the parent model view as shown in the pseudocode in listing 6.

```
toTheOrigin := Translate( -standLocation );  
rotation    := RotateY( 30 ) * toTheOrigin;  
toTheStand  := Translate( standLocation ) * rotation;  
model_view  := parent_model_view * toTheStand;
```

Listing 6: Calculation of final model view matrix of antique object after rotation.

#### 4.4. Exiting the museum

To apparently get out of the museum, press "i". This actually resets the state of all objects to initial values. Thus it appears that user is out of museum with doors closed. Or to exit from the program press q or Q. It closes the program with a "good bye" message printed on terminal.

### 5. Accessing the code

We are planning to make the source code publicly available after the consulting with the course professor. Therefore, in this section, we provide information on how to access them on the internet once they are publicly available. Moreover we explain how to compile the project and run it.

#### 5.1. How to get online

To facilitate developments in the projects without conflicts, we used the *git* version control system and hosted our source code on [www.bitbucket.org](http://www.bitbucket.org). Once the code is ready to access online, users will download it from the following URL <https://bitbucket.org/ph7comp510/public>

#### 5.2. Project structure

In Figure 15 we provide a file structure of our project to simplify the navigation to find source code files. The boxes in yellow represent directories (folders) and that in light blue are the files. We tried to give self-describing names to the directories and source files.

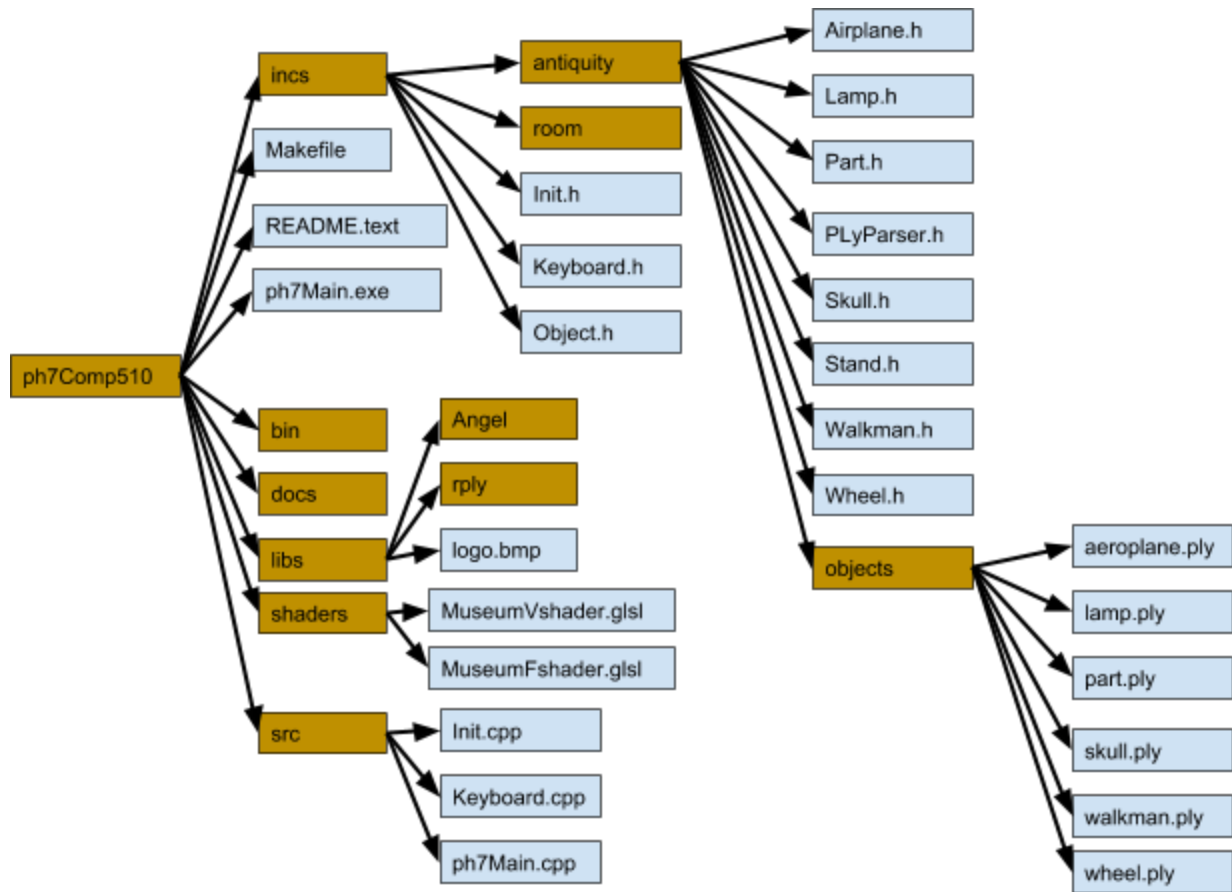


Fig 15: The file structure of the project to facilitate navigation. The yellowish boxes represent directories and light blue boxes are the files.

### 5.3. Prerequisites for compiling and executing the museum

We tested our code on Ubuntu Linux machines with GNU C/C++ Compiler not less than version 4.9.0. We also used GNU make to compile our project. To successful run or program, modern OpenGL, Glut and Glew libraries should be installed on a testing machine. Moreover, we did test our application on machines with Intel graphics drivers and NVIDIA drivers. We recommend running on a machine with NVIDIA drivers because we hardly saw the shading on various objects in the scene when we used computers with Intel drivers.

---

## 5.4. How to build

We provide a zip file containing our project together with this report. To build our program unzip the project file. This creates a folder named “ph7Comp510”. Open the Linux terminal and change your directory location to “ph7Comp510” where the project source codes are found. Finally type the command *make* and press enter. If everything is successful an executable named `ph7Museum.exe` will be generated in the `ph7Comp510` folder.

## 5.5. How to run

To run the program type “`./ph7Museum.exe`” and press enter. A help message will automatically be printed in the console while the program launches. This message outlines a number of keyboard functions and keys a user can use to interact with the museum.

## 5.6. Third part libraries

We used the freely available header files supplied by Ed Angel. We also used an `rply` parser [2] developed by Diego Nehab and made publicly available under the MIT License.

# 6. References

1. <http://people.sc.fsu.edu/~jburkardt/data/ply/>
2. <http://w3.impa.br/~diego/software/rply/>
3. [http://w3.impa.br/~diego/software/rply/#ply\\_get\\_ply\\_user\\_data](http://w3.impa.br/~diego/software/rply/#ply_get_ply_user_data)